UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS (IIMAS)
INTELIGENCIA ARTIFICIAL

AN ARTIFICIAL CREATIVE DESIGN AGENT

TESIS
QUE PARA OPTAR POR EL GRADO DE:
DOCTOR EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
JESÚS PÉREZ ROMERO

TUTORA PRINCIPAL:
DRA. WENDY AGUILAR MARTÍNEZ
IIMAS, UNAM

MIEMBROS DEL COMITÉ TUTOR:
DRA. KATYA RODRÍGUEZ VÁZQUEZ
IIMAS, UNAM
DR. ANDRÉS GÓMEZ DE SILVA GARZA
ITAM

CIUDAD UNIVERSITARIA, CD. MX., DICIEMBRE, 2023

Resumen

En esta tesis presentamos el *framework* para sistemas computacionales de diseño creativo *"computational creative design systems framework (CCDSF)"*, formulado desde una perspectiva mixta de los campos de investigación en diseño y creatividad computacional, para describir, analizar y modelar sistemas computacionales de diseño creativo *"computational creative design systems (CCDSs)"* ya sea como agentes autónomos o como sistemas colaborativos donde participan dos o más agentes de diseño en modo de interacción exclusivamente computacional o humano-computadora.

El *CCDSF* extiende y unifica tres marcos teóricos existentes: *i)* el *framework* de diseño función-comportamiento-estructura *"function-behavior-structure framework for designing (FBS)"*, para integrar conocimiento específico del dominio del diseño; *ii)* el *framework* de sistemas creativos *"creative systems framework (CSF)"*, para modelar la actividad de diseñar como un proceso creativo; y *iii)* la teoría de espacios conceptuales *"conceptual spaces theory (CST)"*, para establecer un sistema de representación neutral.

El núcleo del *CCDSF* es un esquema que estructura la actividad de diseñar a través de conceptos de diseño que son producidos por procesos de diseño instrumentados por estrategias de percepción y acción específicas a uno o varios agentes, que operan dentro de una jerarquía de niveles de diseño que forman capas (entre dos niveles adyacentes). Cada capa puede considerarse como si fuera un sistema creativo de acuerdo con el *CSF* y, dependiendo de las estrategias de sus agentes, manifestar creatividad exploratoria, combinacional o transformacional ya sea al nivel del dominio o a un nivel meta-creativo.

Esta arquitectura permite modelar *CCDSs* en términos de definición/generación/evaluación de artefactos -considerados creativos- (perspectiva de creatividad computacional) y del proceso cognitivo situado para diseñar estos artefactos (perspectiva de investigación en diseño).

Demostramos nuestro *framework* a través de la descripción de un sistema existente y de la creación de dos *CCDSs* como prueba de concepto en el campo del diseño de videojuegos: a) un pequeño *CCDS* autónomo para diseñar mecánicas de juego, y b) un *CCDS* colaborativo humano-computadora, más completo y robusto, para diseñar videojuegos completos del tipo *puzzle*, incluyendo un agente de diseño computacional colaborativo. Dada la naturaleza interactiva de este sistema, se aplicaron algunas prácticas de diseño de experiencia del usuario y de interacción al diseñar y evaluar su interfaz. Además, una evaluación de usuario que puso a prueba el rol del diseñador humano, sugirió que éste podría haberse beneficiado técnica y creativamente al participar en sesiones de diseño colaborativo con su asistente computacional.

Abstract

In this thesis, we present the computational creative design systems framework (CCDSF), formulated from a design research and computational creativity mixed perspective, to describe, analyze, and model computational creative design systems (CCDSs), either as autonomous computational design agents or as collaborative systems involving two or more design agents in computational-only or human-computer interaction modes.

The CCDSF extends and unifies three existing frameworks: *i)* the function-behavior-structure framework for designing (FBS), to integrate design domain-specific knowledge; *ii)* the creative systems framework (CSF), to model design as a creative process; and *iii)* the conceptual spaces theory (CST), to establish a neutral representational system.

The CCDSF core is a schema that structures the activity of designing through design concepts produced by design processes instrumented by agent-specific perception and action strategies within a hierarchy of design levels that form layers (between two adjacent levels). Each layer could be considered a CSF-like creative system and, depending on its agents' strategies, display exploratory, combinational, or transformational creativity at domain- or meta-creative levels.

Such architecture allows modeling CCDSs in terms of definition/generation/evaluation of –deemed creative– artifacts (computational creativity perspective) and the situated cognitive process of designing these artifacts (design research perspective).

We demonstrate our framework by describing an existing system and creating two CCDSs as proof-of-concept (PoC) in the field of video game design: *a)* a small autonomous CCDS to design game mechanics, and *b)* a more comprehensive and robust human-computer collaborative CCDS to design complete puzzle-type video games, including a collaborative computational design agent. Due to the interactive nature of this system, some user experience and interaction design practices were applied in designing and assessing its graphical user interface (GUI). Moreover, a user evaluation testing the human designer's role suggested that the human designer could have gained technical and creative advantages by participating in collaborative design sessions with its computational counterpart.

Contents

Abbreviations

BFS	Breadth-first search
CAD	Computer aided design
CCDS	Computational creative design system
CCDSF	Computational creative design systems framework
CSF	Creative systems framework: Framework for describing exploratory creative systems formulated by Geraint A. Wiggins
CST	Conceptual spaces theory: Theory of conceptual representations developed by Peter Gärdenfors
DFS	Depth-first search
EA	Evolutionary algorithm
EC	Evolutionary computation
ECS	Exploratory creative system
ES	Evolution strategy
FBS	Function-behavior-structure framework for designing developed by John S. Gero
FPS	Fitness proportional selection
GA	Genetic algorithm
GDC	Game developer conference
GE	Grammatical evolution
GUI	Graphical user interface
HCI	Human-computer interaction
IxD	Interaction design

MDA	Mechanics-dynamics-aesthetics game design framework
ML	Machine learning
MLP	Multilayer perceptron
MSE	Mean squared error
PLEX	Playful experiences framework
PLEXQ	Playful experiences questionnaire
PoC	Proof-of-concept
sFBS	Situated version of Gero's FBS framework developed by Gero and Kannengiesser (2002)
UI	User interface
UX	User experience
UxD	User experience design

List of Figures

List of Tables

1 Introduction

From a unified perspective of design and computational creativity, how could an artificial design agent, capable of creative collaboration with a human designer, be modeled and built?

1.1 Background

Since the emergence of the first computer aided design (CAD) tools in the mid-1960s, the application of computing and artificial intelligence (mainly evolutionary computing techniques) to the design domain has been focused mostly on the modeling of designed objects and their optimization on specific performance criteria, supported by analytical methods, computational simulations, and human evaluation. However, these processes correspond only to a part of what design implies.

Various descriptive and prescriptive theories and models that attempt to provide a complete and integrated explanation of the activity of designing have been proposed in the design research field[1]. One notable example is the function-behavior-structure framework for designing (FBS) developed by Gero (Gero 1990; Gero and Kannengiesser 2014). This design cognition oriented framework stands on the axiom stating that "the foundations of designing are independent of the designer, their situation and what is being designed" (Gero and Kannengiesser 2014, pg.

[1]Some representative design theories and models included in (Chakrabarti and L. Blessing 2015; Chakrabarti and L. T. M. Blessing 2014): ARIZ/TRIZ, Theory of Technical Systems (TS), General Design Theory (GDT), Extended GDT, Logic of Design, Mathematical Theory of Design, Universal Design Theory (UDT), $K^{L}D^{E}{}_{0}$—Theory, Axiomatic Design Theory, Theory of Synthesis, C-K (concept-knowledge) Theory, Infused Design, Domain Independent Design Theory, Integrated Model of Designing (IMoD), CPM/PPD framework (Characteristics-Properties Modeling/Property-Driven Development or Design), Systematized Theory for Concept Generation, Domain Theory, Integral Design Methodology (IDM).

264), an idea supported by research (Gero, Kannengiesser, and Pourmohamadi 2014; Gero, Kannengiesser, and Williams 2014; Kan and Gero 2011; Kannengiesser and Gero 2015; Reymen et al. 2006) that suggests that designing is an act independent of the application domain. A revised version of the FBS framework, the situated FBS framework (sFBS) (Gero and Kannengiesser 2002, 2004, 2014), incorporates the situated nature of designing as a recursive process of interaction between the designer actions and her/his interpretations of the results of these actions. In this way, sFBS provides elements to describe the objects being designed, the process to design those objects, as well as situatedness aspects of the designer that executes the design process.

It has become common to confer a creative connotation to designing, the designers, or the objects they design. This is due to the fact that by the act of designing, humans have conceived many physical and non-physical artificial objects that have been deemed useful, valuable, innovative, and even unexpected. Moreover, many of these objects have transformed our environment and shaped our lifestyle.

Some criteria have been proposed to set the distinction between creative design, non-creative design, and no-design at all (Arciszewski, Michalski, and Wnek 1997; Gero 2000; Gómez de Silva Garza 2017; Howard, Culley, and Dekoninck 2008), as well as specific mechanisms that may lead to creative solutions:

- In (Bentley and Wakefield 1998), it is highlighted the possibility for an evolutionary computer system to become capable of producing creative and diverse designs if one considers that this kind of system and its operators (e.g., mutation, recombination, selection) are inspired by natural evolution, an eminently creative designer.
- SCAMPER (Eberle 1996) is a mnemonic format checklist to improve fluent, flexible, and elaborative thinking. Checklist items are: to replace an element or component with another of equivalent functionality (substitute); to mix elements, ideas, and concepts (combine); to make changes to a concept with other original purposes to fit the design purpose (adjust); to make changes to properties within their limits (modify/magnify/minify); to find other concept's purposes (put to another use); to remove elements or components, or simplify the design (eliminate), and to change the order or the arrangement of elements of a concept (reverse/rearrange).
- In (Gero 2000), some creative and innovative design processes are described: the union of concepts or subsets of variables of them (combination), the alteration of one or more concept's variables (transformation), the transference of a concept's aspect to another matched domain (analogy), the recognition of properties beyond the concept's intentional properties (emergence), and the abductive attribution of higher-level properties to concepts (first principles).
- In the C-K theory (Hatchuel and Weil 2007), two spaces are recognized when designing, the concept space containing ideas or objects, and the knowledge space containing the information that grounds the concepts or allows them to become realities. This theory considers that being creative requires expanding these spaces in order to have the required knowledge and desired properties, a process similar to set forcing in modern mathematics.
- The theory of inventive problem solving (or its literal translation from Russian: theory of the

resolution of invention-related tasks “Tieoriya Riesheniya Izobrietatielskij Zadach”) TIPS/TRIZ (Webb 2002) provides a process and a toolset to achieve creative solutions. These ideas were formulated through a systematic analysis of thousands of patents across different fields and sciences. In TRIZ, a design problem must be decomposed into its most elementary components in terms of 39 technological features. These features are cross-compared to find contradictions (a trade-off between two conflicting features) between pairs of them that might be solved by applying one or more solution patterns called inventive principles (40 in total). This process repeats for each unsolved contradiction, including new ones that emerged as a side effect of applying an inventive principle.

- In (Yilmaz et al. 2016), the authors compiled and analyzed several design process outcomes to extract 77 design heuristics that represent different forms a designer appears to introduce intentional variations into concepts as cognitive shortcuts for quantity and variety generation.

We consider that many of these mechanisms may correspond to concrete instances of exploratory, combinational, and transformational creativity strategies (or a combination of them) identified by Boden (Boden 2004) to come up with “ideas” (abstract or concrete) that could be judged as creative (i.e., novel, surprising, and valuable). According to her theory, all concepts, including those individually (p-creative) or historically (h-creative) not yet seen or discovered, exist inside structured spaces of possibilities within a realm (e.g., domain, style of thought, object category) called conceptual spaces, on which the three creativity strategies operate. Exploratory creativity produces concepts by traversing a conceptual space, combinational creativity involves combining “familiar” concepts in a novel way, and transformational creativity makes previously unthinkable concepts reachable by transforming the underlying space. She also states that a creative agent requires evaluative mechanisms to determine the value of the generated or discovered concepts, not only through implicit validation or by interaction with third parties, but through self-awareness that enables the creator to modify the creator’s evaluation mechanisms (Boden 1998).

Although some specific mechanisms focused on achieving creative results may denote (mainly by their name) the idea of combinational creativity, they are actually instances of exploratory creativity in the sense that the combination of concepts occurs in the same conceptual space, employing the same set of dimensions or attributes. The concepts resulting from this type of “combinations” are concepts described by the same set of dimensions as the source concepts (e.g., crossover operator in a standard canonical genetic algorithm). Mechanisms such as analogy are mechanisms that better represent combinational creativity since they combine concepts that are “familiar” in the context of their respective (separated) conceptual spaces.

Wiggins formalizes Boden’s conceptual-space-based nature of the creative process into his creative systems framework (CSF) (Wiggins 2006), a computational framework to describe generic creative processes as exploratory creative systems (ECSs) composed of rulesets to define and restrict the underlying conceptual space of the creative domain, to traverse the space or a superset of it (universe) to generate or discover concepts, and to evaluate concepts and determine their quality according to some criteria. An ECS essentially abstracts the exploratory strategy to search for creative concepts within a single object-level or domain-level space. However, as argued by Wiggins, one or more ECSs can be defined at a meta-level to search for: new ways of defining the domain-level space and its concepts, alternative means to evaluate members of that space, or different methods of exploring it. Such a system would be considered as a transformational creativity strategy if and only if the obtained meta-concepts result in changes at the domain-level space that lead to finding valuable and novel concepts previously inconceivable, non-assessable or unreachable.

1.2 Motivation and aim

Within the scope of an ECS, the three Boden's creativity searching strategies can be visualized as particular types of exploration, differing on nature and the number of spaces being traversed. If an exploratory creativity strategy can be visualized as the productive (in terms of creative results or behavior) traversal of a single object or domain-level conceptual space, we could visualize the combinational creativity strategy as the fruitful traversal of two or more domain-level conceptual spaces that give rise to combined concepts, and the transformational creativity strategy as the productive traversal of one or more meta-spaces, or an object-level space broader than an agreed one (within a culture or domain). This visualization is essential for our research, without denying the non-triviality and broader possibilities of each of these strategies.

Although FBS/sFBS and CSF can be useful to describe computational creative design systems (CCDSs), we find the following limitations. On the one hand, the CSF, being an abstract and generic framework for any potentially creative domain, does not capture specific details of the design domain and does not incorporate the element of situatedness as a possible source of transformational creativity (i.e., by situatedness, the space or spaces being traversed by the designer can be "naturally" shaped and transformed as a result of her/his continuous perception/action loop during a design session). We believe that trying to build a computational creative agent in the design domain solely from the CSF could lead the system's creator to think about the search methods or artificial intelligence techniques to generate certain types of artifacts instead of considering first the processes to be modeled and then think about specific methods or techniques that could support those processes. On the other hand, the FBS/sFBS frameworks were not conceived with creativity-related goals in mind but to describe the cognitive process (whether considered creative or not) performed by a designer when designing and the transformations resulting from this process. Additionally, FBS/sFBS do not consider the meta-level mechanisms to potentially exhibit transformational creativity.

> ***Research hypothesis:*** *Creating an integrated framework that combines the sFBS design framework with the CSF computational creativity framework is expected to yield a robust tool for comprehensively describing and modeling computational creative design systems. This tool would facilitate the characterization and modeling of human and artificial agents engaged in a design process by performing diverse domain-specific tasks and exhibiting varying degrees of creative responsibility.*

In this work, we propose to extend and unify the CSF and FBS/sFBS frameworks by incorporating the strengths of each other. That is, by integrating design domain-specific knowledge based on FBS/sFBS generalized concepts with an extended version of CSF that enables the modeling of design as a creative process, either performed by an autonomous agent or as collaborative systems integrating different agents, including both pure computational and human-computer scenarios. In this way, the CSF approach can be used to force a creator of a system to equip a potential creative design agent with the essential knowledge required to define and recognize artifacts within its domain, to generate new ones, and to evaluate its own creations (at least at the Boden's explorative strategy level); while the FBS/sFBS elements can be used to enable the system's creator to bring these creative system's components to a higher level of detail in a manner consistent with the design's cognitive process and transformations.

We call the unified and extended version computational creative design systems framework (CCDSF), a framework to describe, analyze and model computational creative design systems (CCDSs). It is supplemented with a representational system based on the Gärdenfors' conceptual spaces theory (CST) (Gärdenfors 2004), which provides a cognitive oriented mechanism that is neutral to actual implementation mechanisms such as connectionist or symbolic systems. By employing a CST-based representation system, we make the framework compatible with both the conceptual-space nature of the creative process and the implementation-level representation systems.

We believe that by unifying both design research and computational creativity perspectives, the exploratory, combinational, and transformational creativity strategies in the design domain could be enabled by 1) specific space traversal strategies implemented by computational and artificial intelligence techniques, 2) by the nature and number of spaces being traversed, and 3) as a side effect of the situated nature of designing. Additionally, in our opinion, the strategies that Boden proposes to achieve potential creative results (formalized by the CSF), may optimize the creative scope of a system, independently of different creative/non-creative design criteria.

To illustrate and test the components of the proposed framework, we aim first to analyze and describe, in terms of the framework, an existing system in the domain of game design and then to model and build a collaborative CCDS as an interactive proof-of-concept (PoC) in the puzzle-type video game design domain. We plan to follow an incremental strategy to build this CCDS –including its graphical user interface (GUI)–, starting from a non-collaborative computational agent (design support tool) and gradually increasing its creative responsibilities and collaboration skills towards a co-creative agent (Kantosalo and Toivonen 2016) capable of actively participate in a human-computer co-design environment. Due to its interactive nature (collaboration enabler), it is considered to apply some relevant techniques and principles from the human-computer interaction (HCI), interaction design (IxD), and user experience design (UxD) fields. Also, for completeness, we intend to model and build another small autonomous CCDS for game rule design.

1.3 Objectives

The general objective of this work is to formulate and formalize the integration of Gero's sFBS framework of designing with Wiggins' formalization to Boden's conceptual theory of creativity (CSF) and Gärdenfors' theory of conceptual spaces (CST).

The following specific objectives are derived from the general objective:

1. To test the proposed framework's potential to comprehensively describe an existing system's design-related and creative components.
2. To test the effectiveness of the proposed framework in directing the modeling and development of an artificial agent capable of creative collaboration within a human-computer context.
    - 2.1. To identify the particularities of the puzzle-type video game design domain to build the system as a proof-of-concept (PoC) of the proposed framework.
    - 2.2. To set the properties and criteria to build the user experience (UX) and user interface (UI) of the PoC.
    - 2.3. To determine the computational mechanisms for the PoC implementation.
    - 2.4. To evaluate the UX of the PoC and its impact and influence on the creative process from the user perspective.

1.4 Document outline

This work is organized as follows. Chapter 2 describes the FBS/sFBS, CST, and CSF frameworks, as well as a collaborative extension of the CSF, other related computational creative design frameworks and models, and theoretical key aspects of the puzzle-genre video game design domain. In Chapter 3, we develop and formalize our proposed integrated framework for both autonomous and collaborative systems. Chapter 4 employs our proposed framework to describe and analyze an existing video game design system. Then, we build a PoC in the puzzle-genre video game design domain: a collaborative system in Chapters 5 to 8 and a small autonomous system in Chapter 9. Finally, we provide our conclusions in Chapter 10.

2 Related work

In this chapter, we describe the frameworks of design, representation, and computational creativity on which the formulation of our framework is based, as well as some other works also proposed as computational creative design frameworks or models. We also describe the theoretical basis of the puzzle-genre video game design domain that will support the construction of our PoC.

2.1 The function-behavior-structure framework for designing (FBS) and situated FBS framework (sFBS)

Gero's function-behavior-structure framework for designing (FBS) provides three ontological categories –function, behavior, and structure– of objects to describe the objects being designed, and eight fundamental processes for designing –formulation, synthesis, analysis, evaluation, documentation, functional reformulation, behavioral reformulation, and structural reformulation– to describe the process performed to design the objects.

Gero and Kannengiesser's situated FBS framework (sFBS) models the situated nature of designing as the interaction between three worlds (or spheres) of the designer: the external world that contains things in the environment of the designer, the interpreted world that contains experiences, percepts, and concepts formed by the designer's interactions with the external world, and the expected world (considered part of the interpreted world) that contains expectations and hypothesis of the designer about the current state of the world.

These three designer's worlds give rise to situated ontological categories of objects to denote the world a representation of an object resides in (i.e., external, interpreted, or expected objects). The situated objects are inter-world and intra-world connected by four types of interaction processes –interpretation, constructive memory, focusing, and transformation or action– which are instantiated into 20 specialized design operations[1] (in addition to the behavioral comparison operation) that realize the fundamental processes for designing originally described in the FBS.

2.2 The conceptual spaces theory (CST)

The conceptual spaces theory (CST) is a cognitive-oriented approach to modeling representations proposed by Gärdenfors as a bridge between the symbolic and connectionist representation approaches.

Concepts (what is perceived, remembered, or imagined) are represented through conceptual spaces, which are geometric structures built from quality dimensions. Quality dimensions are representations of perceived characteristics of an object and allow judging similarity between objects. From a modeling perspective, these dimensions represent the framework used to characterize objects and relations among them. Some quality dimensions seem to be innate (embedded dimensions in the case of artificial agents); then, through learning processes, new concepts and dimensions are acquired (some of them, culturally dependent), resulting in the expansion of conceptual spaces. The dimensions of a space can be used to create new higher-order dimensions, which are general patterns (regions) that arise from relations among concepts located in a conceptual space.

The representation of a concept can be organized into domains, which are a type of conceptual space spanned by a small group of integral dimensions (e.g., HSL [hue-saturation-luminosity] color, 2D size). A concept is capable of characterizing both categories of objects (within a concreteness range) and concrete objects (e.g., office chairs, ergonomic office chairs, a particular office chair), which are represented within a conceptual space as regions and points, respectively. These regions represent properties in the context of single dimensions or domains (single-domain) (e.g., small, big, bluish), and "proper" concepts in the context of "proper" conceptual spaces (multi-domain) consisting of separable domains.

The constructive utility of the CST not only applies for modeling concepts that represent objects but other types of concepts such as dynamic properties; actions (force patterns) and events (Gärdenfors and Warglien 2012); functional properties (Gärdenfors 2007); age and time, visuospatial domains, intensions (Gärdenfors 2014); emotions, visual and physical domains, category domain, value domain, goals domain (Gärdenfors 2017); and whole-part relationships (Rama Fiorini, Gärdenfors, and Abel 2015).

[1]The 20 sFBS specialized design operations: 1) functional requirements interpretation, 2) behavioral requirements interpretation, 3) structural requirements interpretation, 4) functional constructive memory, 5) behavioral constructive memory, 6) structural constructive memory, 7) functional focusing, 8) behavioral focusing, 9) structural focusing, 10) function-behavior derivation, 11) behavior-structure derivation, 12) structural external representation, 13) structural interpretation, 14) structure-behavior derivation, 15) behavioral comparison, 16) behavior-function derivation, 17) behavioral external representation, 18) functional external representation, 19) behavioral interpretation, and 20) functional interpretation.

2.3 The creative systems framework (CSF)

Wiggins' creative systems framework (CSF) formalizes a Boden's conceptual space based creative process as an exploratory creative system (ECS):

$$\langle \mathcal{L}, \mathcal{U}, [\![\cdot]\!], \langle\!\langle \cdot, \cdot, \cdot \rangle\!\rangle, \mathscr{R}, \mathscr{E}, \mathscr{T} \rangle,$$

where

- $\mathcal{U}$ is the universe, a superset of the conceptual space $\mathcal{C}$, capable of describing any type of concept;
- $\mathscr{R}$ is a ruleset that define the conceptual space $\mathcal{C}$ that contains the concepts of interest, including empty and partial concepts in which all or some of the values of their attributes are undetermined;
- $\mathscr{T}$ is a ruleset that captures specific space traversal strategies;
- $\mathscr{E}$ is a ruleset that defines the criteria and methods to determine the value of concepts;
- $[\![\cdot]\!]$ is selection interpreter that builds a function that selects valid or valued concepts from a set of concepts, according to definitional or evaluative rulesets;
- $\langle\!\langle \cdot, \cdot, \cdot \rangle\!\rangle$ is an exploration interpreter that builds a function to generate/discover concepts according to the traversal rules, informed by definitional and evaluative rules;
- and $\mathcal{L}$ is a language used to express the rulesets (e.g., a programming language).

The ECS components work as follows:

- The function built by the selection interpreter will select a subset of valid concepts from a given set of concepts according to the definitional ruleset. If the given set were the universe, the selected concepts would be all the concepts that compose the conceptual space: $\mathcal{C} = [\![\mathscr{R}]\!](\mathcal{U})$.
- The function built by the selection interpreter will select a subset of valued concepts from a given set of concepts according to the evaluative ruleset. If the given set were the conceptual space, the selected concepts would be all the concepts that are not only valid but valued: $[\![\mathscr{E}]\!](\mathcal{C})$.
- The function built by the exploration interpreter will discover a new set of concepts from a given set of concepts according to the traversal ruleset and informed by both definitional and evaluative rulesets. The expression $\langle\!\langle \mathscr{R}, \mathscr{E}, \mathscr{T} \rangle\!\rangle^{\diamond}(\{\top\})$ represents the consecutive execution of an ECS starting from an empty concept $\top$, where the concepts produced in one iteration are fed back as input concepts for the next iteration.

This framework identifies and breaks down two properties of creative systems: uninspiration and aberration. Uninspiration occurs in the creative process when the system is not able to find valued concepts in one of the following ways: hopeless uninspiration, conceptual uninspiration, and generative uninspiration. Hopeless uninspiration is a situation where there are no valued concepts in the universe, which means that the system is not capable of finding any valued concept in its universe: $[\![\mathscr{E}]\!](\mathcal{U}) = \varnothing$. Conceptual uninspiration is the situation where there are no valued concepts within the conceptual space explored, indicating a discordance between the definitional and evaluative rulesets (one of the rulesets or both are not adequate) for the system's creative domain: $[\![\mathscr{E}]\!]([\![\mathscr{R}]\!](\mathcal{U})) = \varnothing$. Generative uninspiration occurs when the traversal strategy does not allow finding any valued concept (although they probably exist): $[\![\mathscr{E}]\!](\langle\!\langle \mathscr{R}, \mathscr{E}, \mathscr{T} \rangle\!\rangle^{\diamond}(\{\top\})) = \varnothing$.

An aberration occurs when the creative system finds concepts that do not comply with the definitional rules, which means that these concepts do not belong to the current conceptual space: $\mathscr{B} = \langle\!\langle \mathscr{R}, \mathscr{E}, \mathscr{T} \rangle\!\rangle^{\diamond}(\{\top\}) \setminus [\![\mathscr{R}]\!](\mathcal{U})$, $\mathscr{B} \neq \varnothing$. An aberration can be perfect when all of the invalid concepts happen to be valued (i.e., $[\![\mathscr{E}]\!](\mathscr{B}) = \mathscr{B}$), productive when some of the invalid concepts are valued (i.e., $[\![\mathscr{E}]\!](\mathscr{B}) \subset \mathscr{B}$), or pointless when none are valued (i.e., $[\![\mathscr{E}]\!](\mathscr{B}) = \varnothing$).

2.4 Human-computer collaborative creative systems

Kantosalo and Toivonen (2016) extend Wiggins' CSF to formally analyze and describe a co-creative system scenario both at the system level and the agents involved. The authors describe two types of collaborative agents: a complete agent capable of constructing and modifying a shared concept by her/him/it-self (i.e., the agent is equipped with the three rulesets $\mathscr{R}$, $\mathscr{E}$, and $\mathscr{T}$ of an ECS); and an incomplete agent that is not capable of performing all of the tasks of the creative process (although a complete agent may choose to behave as an incomplete one). Additionally, they identify two modes of collaboration: task-divided collaboration and alternating co-creativity.

In the task-divided mode, each collaborator performs different subtasks (i.e., definition, traversal, evaluation) so that an incomplete computational agent will only be able to collaborate in this mode and will perform the role of definer, generator, or evaluator (or a combination of two).

In the alternating co-creativity mode, each complete agent takes a turn to build and modify a common concept. This mode will be symmetrical if, after being unable to continue operating due to some conflict, the agent is able to modify its rules through transformational creativity and continue operating. It will be asymmetrical if the agent must skip turns to wait for conflict resolution.

The nature of a complete agent role in the alternating mode is defined by how the agent chooses to react to human input. At one extreme, the agent behaves complacently and tries to conform to human-generated ideas and evaluations, and at the other, the agent behaves provocatively and tries to be more challenging to the human. Within the alternating mode, the authors identify four situations that could lead to conflicts in the creative process. The first, called universal mismatch, may occur if one of the collaborative agents generates a concept that is outside the universe of the other agent (i.e., $c_h \notin \mathcal{U}_c$ or $c_c \notin \mathcal{U}_h$). The second, called conceptual mismatch, can arise if any of the collaborative agents are unable to recognize as valid the concept generated by the other agent (i.e., the concepts generated by one agent are not selected by the definitional rules of the definer agent: $[\![\mathscr{R}_h]\!](\{c_c\}) = \varnothing$ or $[\![\mathscr{R}_c]\!](\{c_h\}) = \varnothing$). The third, called artistic disagreement, can arise if one of the two agents does not consider the concept delivered by the other to be valuable (i.e., the concepts generated by one other agent are not selected by the evaluative rules of the evaluator agent: $[\![\mathscr{E}_h]\!](\{c_c\}) = \varnothing$ or $[\![\mathscr{E}_c]\!](\{c_h\}) = \varnothing$). Finally, the fourth conflict, called generative impotence, is present if one of the two agents is unable to continue their creative process based on the concept generated by the other collaborator (i.e., the execution of the exploration interpreter of one agent from concepts generated by the other agent results in an empty set: $\langle\!\langle \mathscr{R}, \mathscr{E}, \mathscr{T}_h \rangle\!\rangle(\{c_c\}) = \varnothing$ or $\langle\!\langle \mathscr{R}, \mathscr{E}, \mathscr{T}_c \rangle\!\rangle(\{c_h\}) == \varnothing$).

2.5 Other computational creative design frameworks

Howard, Culley, and Dekoninck (2008) proposed an *integrated model of the creative design process* as an extension to Gero's original FBS framework. In their model, the authors map the creative processes of analysis, generation, and evaluation into the fundamental processes for designing and their underlying design operations proposed by the FBS. The authors describe their creative process of analysis as the continuous construction of design information that occurs in the course of designing, acting over all of the transformation and comparison operations defined by the FBS; although the original FBS does not contemplate this process, it is considered by the situated version of the FBS through the continuous processes of interpretation, reflection, and focusing. The creative process of evaluation is mapped into the comparison and FBS-analysis operations, and the creative process of generation is mapped into the FBS operations of synthesis, derivation of behavior from the function, and the three reformulations.

Kelly and Gero (2015) developed a *situated computational creativity model* that considers the interpretation process as the primary trigger of movements between frames, understanding the frame as the design situation, and the conceptual space the system traverses. The movement from one conceptual space to another can be beneficial for the creative task since it can locate the system in a more favorable zone within a universe of possibilities. Interpretation is modeled as the interaction between experience (what the system knows), expectations (what is in the current framework), and perception (what is interpreted).

The model employs a modified CSF framework, establishing a set of R rules to generate a conceptual space, a set of N rules to interpret stimuli and create internal representations, and a set of T rules to define a space traversal strategy. The universe U is restricted to a viable universe for the system, and the conceptual space C is not built from the entire universe but only from the previous experience of designer $U!$. The movement between conceptual spaces is established through the exploration interpreter $\langle\!\langle \cdot, \cdot, \cdot \rangle\!\rangle$, which after being executed, creates the new conceptual space (move to another space) C_{i+1} according to the sets of rules and based on the current conceptual space C_i, making the space traversal to depend solely on previous experiences in the same design context and not on the whole previous experience $U!$. Both expectations E and situated interpretations I are described in the model, distinguishing between explicit expectations E^* (those to which the system is attending) and explicit expectations E' (those resulting from experience). Interpretations in this model are dynamically constructed from a source (external) that is interpreted as percepts P, which are interpreted as concepts C, which are interpreted as a situation s. Interpretations in this model are built dynamically within a hierarchy of abstractions; the source x is interpreted as a set of percepts P to build a first stage interpretation I_P, which is interpreted a set of concepts C to build a second stage interpretation I_C, which is interpreted as a design situation s to build the third and final interpretation stage I_s. This interpretation mechanism is chained in such a way that previous interpretations and current expectations influence new interpretations.

Quanz et al. (2020), proposed an implementation-oriented *co-creative design framework* composed of three main components: a creator, an iterator, and evaluators. The creator component captures the design space constraints, receives human designer sketches, and generates designs and variants of designs. The evaluators predict several design scores, such as novelty, cost, and consumer appeal, whose influence on the design is controlled by the human designer. Finally, the iterator component represents the link between the creator/evaluator and a human designer in order to progressively evolve designs and enable design space exploration based on the generation of design proposals and learning of human preferences. The authors illustrate their framework through a perfume bottle design tool. The peculiarity of this framework consists in the incorporation of multiple machine learning (ML) methods as a strategy to implement the proposed components that may address better the adaptation to the diversity of the human designer needs (e.g., learning methods) and its potential application to diverse domains (e.g., transfer learning, style transfer).

2.6 Puzzle video game design

According to Juul (2005b), a game can be defined as a goal-oriented rule-based system with a variable and quantifiable outcome (a final state of a game), where a player exerts effort to influence that outcome and feels emotionally attached to it. In Juul's definition, a goal is an assignment of value to the game's possible outcomes and refers to the game as an activity, not as an object. In that same sense, Schell (2008) defines a game as a problem-solving activity, approached with a playful attitude.

Based on the reflections of the computer game designer, author of the first video game design theory book, and founder of the game developer conference (GDC), Chris Crawford (1984, 1990), a puzzle game can be defined as a formal interactive system that offers a challenge to the player, clearly defined as goals, through entertaining behaviors established by actual or fictional physical properties, explicit rules, and algorithms. Crawford emphasizes that a puzzle game does not present a purposeful opponent to the player to be defeated, but the opponent is the challenge itself. A puzzle game is essentially a game with a dominant strategy that makes the player stop and think (Schell 2008).

Three aspects stand out of what constitutes a game: *i)* the underlying rules, *ii)* the behavior enabled, and outcomes obtained in response to the player's actions; and *iii)* the emotional, playful aspect. The mechanics-dynamics-aesthetics game design framework (MDA) (Hunicke, Leblanc, and Zubek 2004) captures these aspects as rules, system, and "fun" components and establishes their design counterparts: mechanics, dynamics, and aesthetics, each acting as a "lens" or a "view" of a game. As of our proposed designing framework, MDA recognizes that from the perspective of the designer, the mechanics give rise to dynamic system behavior (game dynamics), which in turn leads to particular aesthetic experiences.

2.6.1 Game experience

In the MDA framework, the aesthetics view describes the desirable emotional responses evoked in the player when interacting with the game. We argue that MDA's aesthetic view can be interpreted as the game's playful experience of the player[2]. According to the playful experiences framework (PLEX) (Korhonen, Montola, and Arrasvouri 2009), a playful experience is an emergent approach for a player to enjoy playing a game. The PLEX framework provides a catalog of 22 playful experiences categories (Arrasvuori, Boberg, and Korhonen 2010) as different possibilities to design and evaluate the playful experience of a player when playing a game. There are auxiliary tools such as the PLEX cards (Lucero and Arrasvouri 2010), and the playful experiences questionnaire (PLEXQ) (Boberg et al. 2015) to assist designers and evaluators in modeling/identifying desirable/emerged playful experiences.

The PLEX cards are a set of cards designed to communicate each of the PLEX categories to inspire designers while designing for playfulness. Each PLEX card consists of two pictures (carefully selected to avoid feeling them stereotypical) and a central ribbon (Fig. 2.1).

PLEXQ is a validated 5-point Likert scale (strongly disagree, disagree, neither agree nor disagree, agree, strongly agree) questionnaire to reliably measure 17 of the 22 PLEX playful experiences categories. The questionnaire consists of 51 items (3 for each category), developed from qualitative studies, interviews with video game players, and brainstorming sessions with experts.

PLEX framework may provide a useful foundation to build our set of design dimensions with PLEX cards and PLEXQ questionnaire as support tools to design and analyze game experiences from the perspective of playfulness.

For our tool, the PLEX playful experiences categories could function as design dimensions with PLEX cards and PLEXQ questionnaire as support tools to design and analyze game experiences from the playfulness point of view.

[2]We believe the term 'game experience' to be more appropriate than 'aesthetics' since aesthetics may lead to confusion with other aspects, such as the video game's visual appearance.

Table 2.1: PLEX playful experiences categories

Experience category	Brief description: The playful experience emerges from...
Captivation	Forgetting one's surroundings
Challenge	Testing abilities in a demanding task
Competition	Contest with oneself or an opponent
Completion	Finishing a major task or reaching closure
Control	Dominating, commanding or regulating
Cruelty	Causing mental or physical pain
Discovery	Finding something new or unknown
Eroticism	A sexually arousing situation
Exploration	Investigating an object or situation
Expression	Manifesting oneself creatively
Fantasy	An imagined situation
Fellowship	Friendship, communality or intimacy
Humor	Fun, joy, amusement, jokes or gags
Nurture	Taking care of oneself or others
Relaxation	Relief from bodily or mental work
Sensation	Excitement by stimulating senses
Simulation	An imitation of everyday life
Submission	Being part of a larger structure
Subversion	Breaking social rules and norms
Suffering	Loss, frustration or anger
Sympathy	Sharing emotional feelings
Thrill	Excitement derived from risk or danger

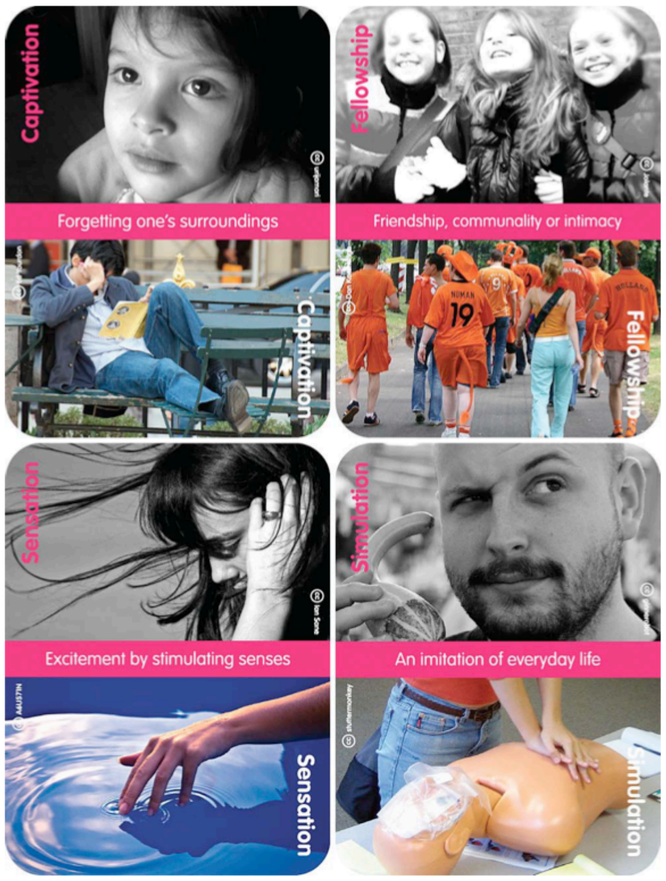

Captivation
Forgetting one's surroundings
Captivation
Fellowship
Friendship, communality or intimacy
Fellowship
Sensation
Excitement by stimulating senses
Sensation
Simulation
An imitation of everyday life
Simulation

Figure 2.1: PLEX cards: The top half picture is a black and white picture displaying faces to focus on the emotion; the central ribbon shows some concrete examples of the emotion or short descriptions of it; finally, the bottom half picture is a color picture displaying concrete examples from everyday life. Both top and bottom half pictures are tagged with their corresponding PLEX category name.

2.6.2 Game dynamics

MDA's dynamics view describes the run-time behavior of the mechanics acting on player inputs and others' outputs over time. In addition to the run-time (playtest-based) behavior, we argue that the dynamics view should also be made up of the game's behavior derived directly from game mechanics analysis (i.e., intrinsic dynamics). Browne employs a model to measure the games generated by his Ludi system (2011) based on self-play (automated playtest) and intrinsic game characteristics. He distinguishes between intrinsic and extrinsic criteria, the latter subdivided into viability and quality criteria. Intrinsic criteria are those based on rules and equipment, whereas extrinsic criteria are derived from the automated self-play. Viability criteria are based on game outcomes and essentially determine the playability of a game, whereas quality criteria are based on the player moves and attempts to gauge players' engagement.

Another type of run-time behavior is the gameplay scenario, defined as the set of patterns of interaction of the player with the game that emerges during playtesting (Lindley 2004). Maze-based, twitch-based, and logic-based are examples of gameplay scenarios that may be present in some puzzle video games (Game Design Ed 2015a,b).

According to Crawford (1984), in maze gameplay scenarios, the player must move through a labyrinth and might be required to make his/her way to an exit. In addition, there might be chasing objects that pursue the player through the maze. In «Puzzle Explorer»[3] (2015a) a maze-based puzzle is described as a puzzle in which a player attempts to reach a goal on a game board by finding the correct path within a complex, branching network of paths.

On the other hand, as explained in «Puzzle Explorer» (2015b), twitch-based gameplay scenarios challenge the player's reaction time and dexterity, perhaps through strategically placed obstacles and AI-based chasing objects. Twitch gameplay keeps players actively engaged with quick feedback to their actions (Wikipedia contributors 2020).

Finally, as inferred from (Pustejovsky 2013), logic-based gameplay scenarios require different kinds of reasoning to resolve the puzzle and challenge the player analytically. Logic-based puzzles challenge the player's mind by making him/her holding the puzzle as a system in their head (parts of the puzzle or the puzzle as a whole) and thinking a few steps ahead of their current situation (i.e., make and test hypotheses) (Game Design Ed 2015b). Making the player stop and think, as considered by Schell to describe a puzzle becomes even more characteristic in logical puzzles.

2.6.3 Game mechanics

MDA's mechanics view describes the particular components of the game at the level of data representation and algorithms. Adams and Dormans (2012) describe game mechanics as the core of a game, composed of rules, processes, and data that determine what players can do and how the game will react to their actions (e.g., how play progresses, what happens when..., what conditions determine victory or defeat). According to them, most of game mechanics are hidden from the player, that is, the rules encoded into game mechanics are different from those explicitly made known to the player as instructions (i.e., what people commonly refer to as game rules). In a similar direction, Schell (2008) defines game mechanics as objective sets of rules describing the underlying model of a game and, at the same time, recognizes a rules category of game mechanics as the most fundamental category that make all the other types of mechanics possible. These rules, according to Juul (Juul 2005b), specify limitations and affordances in the game.

The «Puzzle explorer» video series (Game Design Ed 2015b) demonstrates how a puzzle video game can be designed by arranging different objects (e.g., obstacles, tiles, collectibles, player token, pushables, actionables, enemies, keys) within a squared grid board. This procedure is consistent with the activity of designing video game levels, defined by Juul (2005a) as the design of the physical layout in a game, and by Schell (2008) as the arrangement of the elements of the game in ways that are fun and interesting.

[3]«Puzzle Explorer» is a video playlist of the YouTube channel «Game Design Ed» about the tool and game developed by YETi CGI and published by Fingerprint, Inc. called «National Geographic Puzzle Explorer» (not available at the time of this research).

According to Browne (2013), a puzzle game level is a specific instance of a puzzle game defined by a specific grid (game space) and game objects arrangement. We believe that the introduction of sufficiently different low-level game mechanics (or a different composition of them) may allow not only the production of a new level but also a new puzzle game variation or even a completely new puzzle game.

Game level design is, in fact, an instance of game mechanics design since game levels are considered a type of game mechanics. According to Adams and Dormans (2012), a game level can be considered a progression game mechanics type that dictates how a player can move through the game world. Similarly, the «Game Mechanics Wiki» (GMW contributors 2011) states that a game level (sometimes used interchangeably with map) is a game mechanic that makes up the playable game world.

In addition to the level-type game mechanic, one can recognize other categories of game mechanics in almost any puzzle video game (like those demonstrated in the «Puzzle Explorer» video series). The squared grid board is a space game mechanic, defined by Schell (2008) as a mathematical construct that defines the various places that can exist in a game, and how those places are related to one another.

The objects being placed and arranged by the designer in the game space fall into the game object category of game mechanics, considered by Schell (ibid.) as the “nouns” of game mechanics and defined as any object that can be seen or manipulated by the player in a game. Continuing with Schell, the static (never changing through-out the game) and dynamic attributes (e.g., appearance, position), and states of any game object are also categories of game mechanics.

Finally, in puzzle video games, we also find rules that encode actions (the “verbs” of game mechanics [ibid.]) and end conditions game mechanics. Action game mechanics encode changes (and applicable constraints) in the states and game objects attributes’ values in response to the player’s input (operative actions [ibid.]) or as side effects of other game object’s changes. On the other hand, end conditions game mechanics encode situations that cause a game to end with a player victory (win condition) or a game-over situation (loss condition). In terms of game outcomes, Juul (2005a) states that winning a game means that the game ends with the outcome of having a positive value assigned to the player (i.e., the goal was achieved), and losing a game means that the game ends with the outcome of having a negative value assigned to the player (i.e., the necessary conditions to achieve the goal no longer exist).

2.7 Rule-based game engine

Based on the definitions in Section 2.6.3, we may consider a puzzle video game (from the game mechanics perspective) essentially as a rule-based system, where a set of if-then rules (ruleset) governs the run-time interaction with the player and game outcomes. Thus, a puzzle video game engine could work exactly as a rule-based system engine where initially, the rules are loaded to build up the system’s working memory with initial facts. Then, for each game tick or player’s move, the rules are fired to produce a new set of facts as the new game state. A relevant example of such an engine is PuzzleScript (Lavelle 2013), an open source HTML5-based platform developed by Stephen Lavelle that provides both, a scripting language to describe a tile-based puzzle video game, and the engine that executes a playable version of the game described.

In essence, to describe a video game in PuzzleScript, a designer *i)* labels and defines the visual aspect of all of the game objects appearing in the game (game object design), *ii)* assigns each game object into a layer (the base 'Background' layer is mandatory), *iii)* places a single game object or a combination of game objects into specific cells of a rectangular game board (game level design), *iv)* optionally writes one or more win conditions (end conditions design), *v)* and writes one or more game rules (game rule design) that govern the interaction between the player (movements and actions) and the game objects, and the interaction between game objects.

Game rules in PuzzleScript are, in fact, if-then rules in the form of substitution patterns. PuzzleScript game engine searches for game board cell patterns defined at the 'if' (left) part of a rule and replaces every matched pattern with the patterns specified at the 'then' (right) part of the rule. A pattern is just a series of cells where each cell is described by an optional modifier and a game object. Every game object assigned into a layer can be used (be referenced) in any win condition or game rule part. If required, game objects can be grouped into 'virtual' game objects (or aliases) and then, use them in win conditions and game rules as regular game objects.

3

CCDSF - Computational creative design systems framework

3.1 Layered Designing Schema: a model of the activity of designing

According to Gero's research on design cognition (Gero 1990; Gero and Kannengiesser 2000), the act of designing can be defined as the goal-oriented, decision-making, constrained, exploratory, and situated activity to transform –through a series of fundamental processes– a set of representations of a required or expected object (design problem) into a design description of an object that meets expectations (design solution or designed object) and provides sufficient information about it for its construction.

Requirements & expectations
Designing
Design descriptions

Figure 3.1: Designing activity

The object being designed or *design object* can be of any class, including artifacts (digital or physical), processes, systems, and services. Both expected and designed objects are representations of the same design object and can be expressed by any appropriate means or can correspond to the object itself –if one considers the object as a representation of the solution being designed.

Based on the previous definition, on Gero and Kannengiesser's FBS/sFBS frameworks (Gero 1990; Gero and Kannengiesser 2002, 2004, 2014), and on Gärdenfors' CST (Gärdenfors 2004), we propose to model the activity of designing in terms of three main components: *design concepts* (i.e., representations of design objects), produced trough a design process modeled by *fundamental processes for designing* instrumented by agent-based *perception/action strategies*, within a hierarchy of what we call *design levels*. We describe these components in the next subsections.

3.1.1 Design levels

A design level L is a perspective from which a design object is both described and detailed. On the one hand, the most general perspective from which any design object can be described is through ontological categories, such as the *function* (what the design object is for?), *behavior* (what the design object does?), and *structure* (what the design object is?) categories defined in the FBS. Other FBS-like frameworks go even further and recognize additional categories above function, such as *purpose* (Galle 2009; Salustri 2015), *needs* (Cascini, Fantoni, and Montagna 2013; Uflacker and Zeier 2008), and *user and goals* (Cascini, Fantoni, and Montagna 2011). On the other hand, the abstraction level is another type of perspective that expresses the degree of detail or granularity of the produced designs. For example, conceptual design, preliminary design, and detailed design in the engineering design process (Fig. 3.3), or low-fidelity wireframing, high-fidelity wireframing, mockup, and final UI design in a user interface design process.

Definition 3.1.1. A *design level* L is a unified perspective from which a design object is described in terms of an ontological category at a given level of detail.

An example of a design level could be one defined by the behavioral ontological category and the conceptual abstraction level, resulting in what could be called a behavioral conceptual design level. There might be, however, one or more abstraction levels defined for more than one ontological category (e.g., behavioral conceptual design, structural conceptual design).

3.1.1.1 Hierarchy of design levels

Within an application-domain-specific design process, a hierarchy of design levels $L_1, L_2, \dots, L_n$ can be recognized, in such a way that: *i)* L_{i-1} is the level below level L_i, for $1 < i \leq n$, (to make the notation shorter, from now on we refer to L_i as L, L_{i-1} as L^-, L_{i+1} as L^+, and LL^- as what we call a *design layer* formed between L and L^-); and *ii)* different required objects described at the design level L, can be satisfied by different design objects described at the design level L^- –or different objects described at L^- level, can realize different required objects described at L level (Fig. 3.2), that is, the relationships between design levels within a hierarchy are many-to-many and bidirectional (see example of aircraft design domain in Fig. 3.4).

The hierarchy built-up by design levels reflects the top-bottom approach a designer should take to make decisions (e.g., the designer decides on behaviors that realize functions, then structures that perform those behaviors). In this way, the activity of designing can also be regarded as the transformation of a high-level description of a design object into a low-level description of the same object.

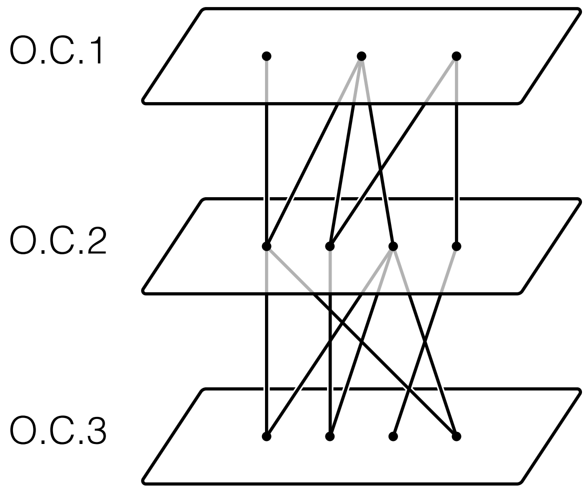


Figure 3.2: Relationships between ontological categories (O.C) of design objects

Definition 3.1.2. A *design layer* LL^{-} is formed between two adjacent design levels L and L^{-} of the hierarchy and may constitute, at least conceptually, an independent act of designing by itself.

The hierarchy of design levels leads us to propose that full designing (from initial requirements up to final design description) could be decomposed into smaller and specialized designing instances located at each layer LL^{-} that emerges from the hierarchy. Therefore, the design descriptions (solutions) produced at a given layer would represent a subset of requirements (part of the problem) for the layer below (or the final output if no further layers) (Fig. 3.5). Even when intermediate outputs may not be produced for final construction purposes, they may constitute an incremental approximation to the final design description.

3.1.1.2 Modeling design levels as design spaces

In the proposed layered schema, any design level L gives rise to a design space $\mathcal{D}_L$, which represents a set of design objects that can be described from the perspective of that design level. Inspired by the CST, we propose to define design spaces in terms of design dimensions $\delta \in \Delta$.

We employ dimensions instead of variables (as in the FBS), considering that not every type of variable may assume one value from a given set of values. From the perspective of the FBS, although behavioral and structural dimensions can resemble variables, functional dimensions (as hypothesized by Gärdenfors) correspond to subjective abstract dimensions constituted by behavioral dimensions, being subjective as function-behavior mappings (behaviors afforded by objects to achieve a purpose) are up to individual, social, and cultural knowledge.

Definition 3.1.3. A *design dimension* $\delta_L \in \Delta_L$: *i)* is a parameter that represents a characteristic of a design object within the context of a design level L, and *ii)* enables determining similarity and dissimilarity between two objects with respect to the represented characteristic.

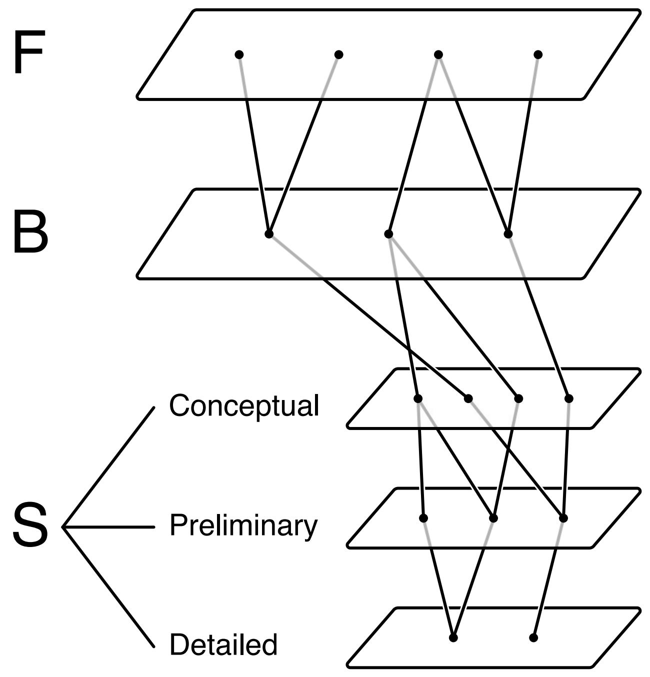


Figure 3.3: FBS ontological categories and engineering design process

A design dimension can be either separable or integral (Gärdenfors 2004, pg. 24) if it cannot be separated from other dimensions to be meaningful in its context of use (e.g., when characterizing the 2D-size of an object, one cannot set a value to the length dimension without assigning a value to the height dimension). Additionally, design dimensions can be classified as *base design dimensions*, *compound design dimensions* (domains in the CST), and *higher-order design dimensions*:

- A *base design dimension* $\dot{\delta}_L \in \Delta_L$ may resemble the concept of mathematical variable; it is defined by a metric space specified by a range r (an arbitrary non-empty set containing all possible values the dimension can take) and a metric[1] m on this range:

$$\dot{\delta}_L \coloneqq \langle r, m \mid r \in \mathscr{V} \setminus \{\varnothing\} \rangle,$$

  where $\mathscr{V}$ is the universal class (the class of which all sets are members).

- A *compound design dimension* $\ddot{\delta}_L \in \Delta_L$ is a bundle of one or more integral design dimensions δ_L, including other integral compound dimensions:

$$\ddot{\delta}_L \coloneqq \left\langle \langle \delta_{L_i} \rangle, m \right\rangle, \text{ where}$$

$$\langle \delta_{L_i} \rangle \coloneqq \left\langle \delta_{L_1}, \ldots, \delta_{L_n} \,\middle|\, \delta_{L_i} \in \Delta_L \; \forall i \in \{1..n\} \right\rangle;$$

[1]The metric could be a generalization of a metric resulting in a generalized metric space.

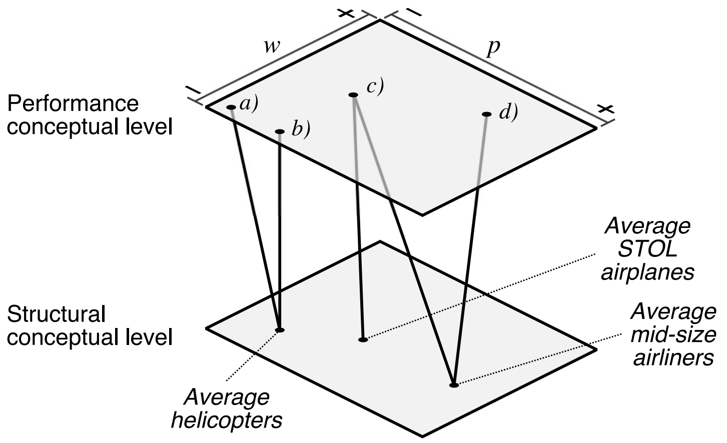

w
p
Performance conceptual level
a)
b)
c)
d)
Average STOL airplanes
Structural conceptual level
Average mid-size airliners
Average helicopters

Figure 3.4: Aircraft design domain example: In a hierarchy of two design levels, performance conceptual and structural conceptual, various expected flying vehicles (e.g., *a*, *b*, *c*, *d*) can be defined in terms of their performance determined by two characteristics, the minimum length of the runway *w* required by the vehicle to perform take-off and landing procedures, and their payload capacity *p*. On the other hand, various families of aircrafts (e.g., average helicopters, average mid-size airliners, average STOL [short take-off landing] airplanes) can be defined in terms of various structural conceptual features (not depicted). Performance conceptual flying vehicles represent: *a*) a negligible runway length, small payload capacity; *b*) a negligible runway length, small-to-medium payload capacity; *c*) a medium length runway, small-to-medium payload capacity; and *d*) a large runway, medium-to-large payload capacity. Both *a*) and *b*) performances can only be satisfied by the average helicopters family, performance *c*) can be satisfied by both the average STOL airplanes and mid-size airliners families, and performance *d*) can only be satisfied by the average mid-size airliners family. In the opposite direction, the average helicopters family can realize *a*) and *b*) performance expectations, the average STOL airplanes family can realize performance expectation *c*), and the average mid-size airliners family can realize two performance expectations *c*) and *d*).

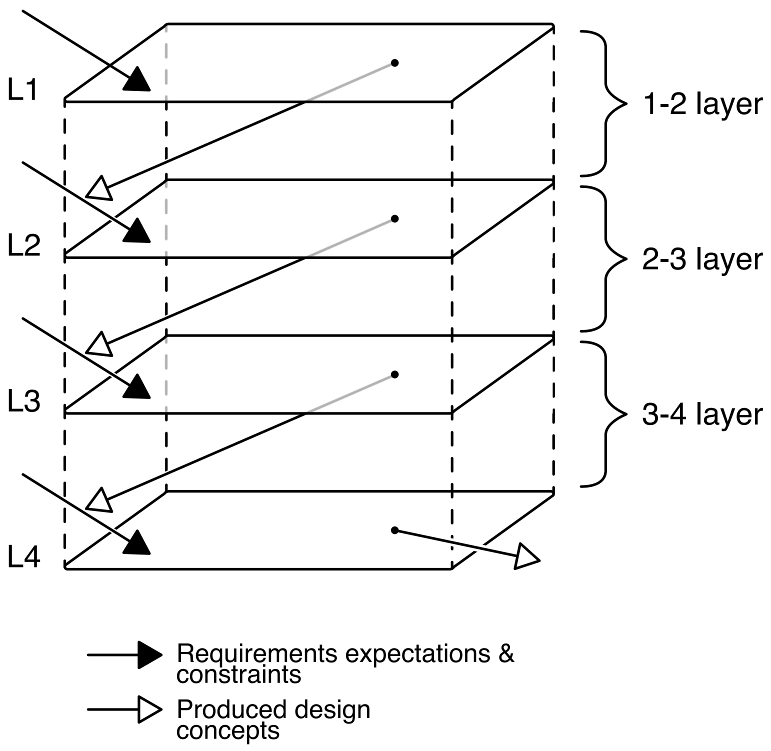


Figure 3.5: Chained design levels and layers

its range $\operatorname{ran}\left(\ddot{\delta}_L\right)$, is the cartesian product of its integral dimensions' ranges:

$$\operatorname{ran}(\delta_L) := \begin{cases} \delta_{L(1)} & \text{if } \delta_L \text{ is a } \dot{\delta}_L \\ \displaystyle\prod_{i=1}^{|\delta_L|} \operatorname{ran}\left(\delta_{L_i}\right) & \text{if } \delta_L \text{ is a } \ddot{\delta}_L, \end{cases}$$

where $\delta_{L(1)}$ is the first element (i.e., its range) of the tuple that represents the design dimension δ_L.

- Finally, a *higher-order design dimension* $\dddot{\delta}_L \in \mathbf{\Delta}_L$ is a special type of compound dimension integrated by (possibly) constrained dimensions[2] δ'_{L^-} defined at the design level L^- below; in other words, it is a dimension defined as projected regions over lower-level design spaces:

$$\dddot{\delta}_L := \left\langle \langle \delta'_{L^-i} \rangle,\, m \right\rangle, \text{ where}$$

$$\langle \delta'_{L^-i} \rangle := \langle \delta'_{L^-1}, \ldots, \delta'_{L^-n} \mid \exists \delta_{L^-i} \in \mathbf{\Delta}_{L^-} : \delta_{L^-i} = \operatorname{orig}\left(\delta'_{L^-i}\right) \forall i \in \{1..n\} \rangle,$$

and $\operatorname{orig} : \mathbf{\Delta}_L \to \mathbf{\Delta}_L$, is a function that maps a given possibly constrained design dimension δ'_L to the original unconstrained dimension δ_L.

[2]A design dimension is said to be constrained when its range is reduced to a subset (region) of its original full range, providing another aspect to model the *decision-making* nature of designing. Both full-range dimensions and any of the constrained dimensions derived from them are considered as members of the same set of dimensions.

At a given design level L, there might not be a set of possible base design dimensions defined (such as in the FBS functional category); instead, one can configure higher-order design dimensions from dimensions of the immediate lower level L^{-}.

In general, Gärdenfors suggests employing the Euclidean metric to determine the distance between constitutive integral base dimensions (inside a non-nested compound dimension) and Manhattan metric to determine the distance between separable dimensions (Gärdenfors 2004, pg. 25).

Definition 3.1.4. A *design space* $\mathcal{D}_L$ is a non-empty collection of full ranged or constrained design dimensions belonging to an L design level (intensional definition):

$$\mathcal{D}_L := \begin{cases} \langle \delta'_{L_1}, \ldots, \delta'_{L_n} \mid \exists \delta_{L_i} \in \Delta_L : \delta_{L_i} = \mathrm{orig}\left(\delta'_{L_i}\right) \forall i \in \{1..n\} \rangle & \text{(constrained)} \\ \langle \delta_{L_1}, \ldots, \delta_{L_n} \,\middle|\, \delta_{L_i} \in \Delta_L \ \forall i \in \{1..n\} \rangle & \text{(full-ranged)} \end{cases}$$

A design space can also be considered as the set of all design objects representable by a set of design dimensions (extensional definition).

A consequence of the proposed layered schema is that a given design layer LL^{-} will contain two design spaces (one for each bounding level), functionally differentiated by the role they represent in the design process. The upper bounding space, which we call *evaluation space*, acts as the space of possible required design objects, and the lower one, which we call *solution space*, acts as the space of possible design objects that represent potential solutions for those requirements. Thus, the design space $\mathcal{D}_L$ corresponds to the evaluation space $\mathcal{E}_{LL^{-}}$, and the design space $\mathcal{D}_{L^{-}}$ corresponds to the solution space $\mathcal{S}_{LL^{-}}$, which will become the evaluation space of the layer below: $\mathcal{D}_L = \mathcal{S}_{L^{+}L} = \mathcal{E}_{LL^{-}}$ (Fig. 3.6).

Our design space enables modeling the *explorative* nature of designing as the traversal of the space directed by design decisions and unifies the idea of a conceptual space in the CST with the notion of all of the «spaces» described by the sFBS, and CSF. The sFBS considers a design state space as the space of possible design solutions, on the other hand the CSF describes a conceptual space $\mathscr{C}$ containing all *valid* concepts according to the $\mathscr{R}$ ruleset, a set we name $\mathcal{V}$ of *valued* concepts according to the $\mathscr{E}$ ruleset, and a set $\mathscr{C}_?$ of concepts unknown to the creator (i.e., potential *novel* concepts).

3.1.2 Design objects

During designing, several representations of the design object and perhaps other related objects are involved; at least, there is one representing the required object and another one representing the designed object. The representation of the required object is described in terms of the upper design level of a given design layer with the purpose of defining requirements and constraints. On the other hand, the representation of the designed object describes the design solution in terms of the lower design level of the layer. Besides these core representations, a design agent may produce or consume other representations (mental, virtual, or physical) for other purposes, such

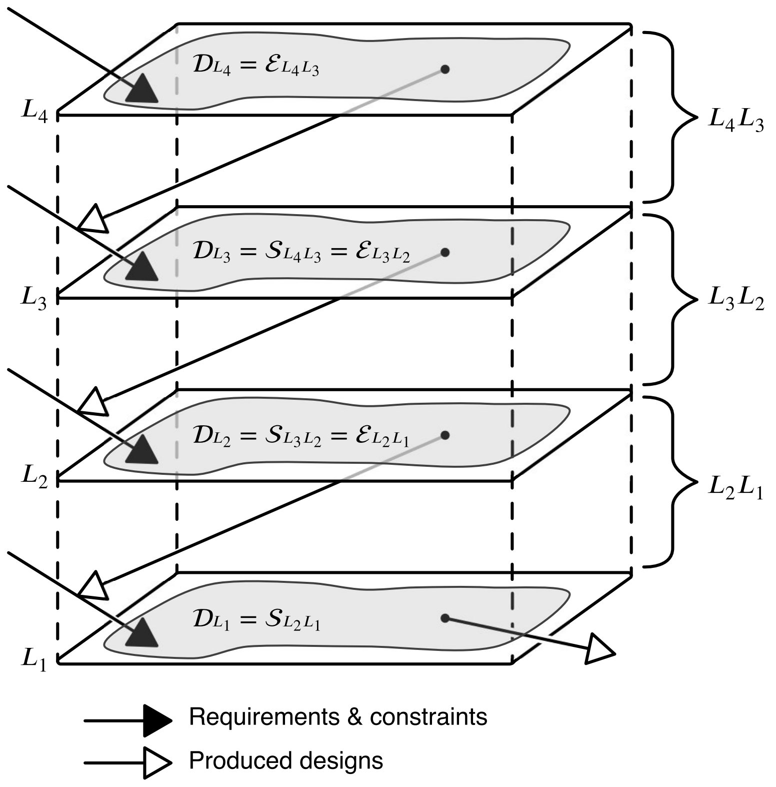

$\mathcal{D}_{L_4} = \mathcal{E}_{L_4L_3}$
$\mathcal{D}_{L_3} = \mathcal{S}_{L_4L_3} = \mathcal{E}_{L_3L_2}$
$\mathcal{D}_{L_2} = \mathcal{S}_{L_3L_2} = \mathcal{E}_{L_2L_1}$
$\mathcal{D}_{L_1} = \mathcal{S}_{L_2L_1}$
L_4
L_3
L_2
L_1
L_4L_3
L_3L_2
L_2L_1
Requirements & constraints
Produced designs

Figure 3.6: A layered designing schema composed of four levels $L_1 \dots L_4$. Three layers emerge from the design levels hierarchy $L_2L_1 \dots L_4L_3$. Black arrows denote the flow of requirements and constraints at a given design level. White arrows denote the flow of the produced design solutions that joins the requirements flow in non-lowest levels. Every design level gives rise to a design space that has a two-folded function in non-extreme design layers.

as communication, testing, analysis, or as auxiliary representations to support designer's own reflection, and interpretation of referential objects (e.g., inspiring set) or other external objects. Then, representations of the same design object differ in the design level they are expressed at, and their purpose in the design process.

3.1.2.1 Modeling design objects as design concepts

We adopt the term *design concept* to refer to a representation of a design object independently of its underlying design level and its purpose in the process (Fig. 3.7). This term unifies the notions of concept in the CST –a means of representation–, and the CSF –a catch-all term for the target of a creative system processes (i.e., validation of, evaluation of, and search/discovery/generation of)–, with the idea of representations in the FBS/sFBS.

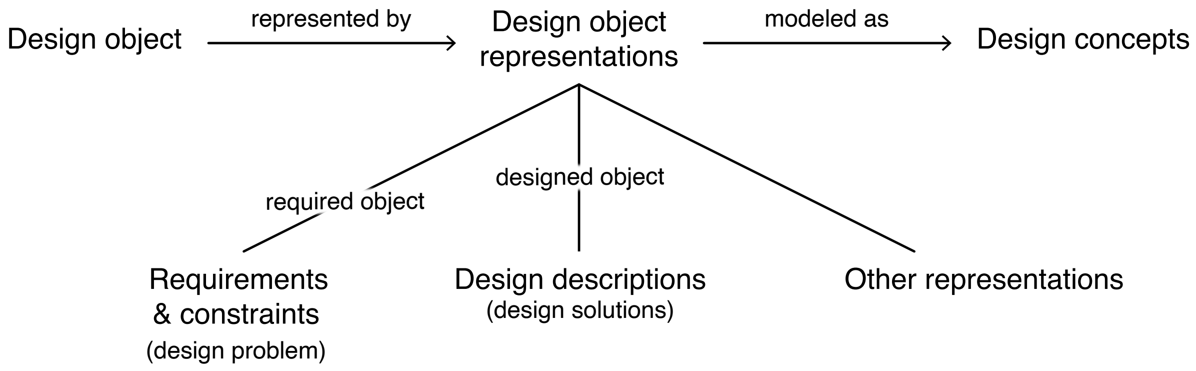


Figure 3.7: Design concepts as representations of design objects

Definition 3.1.5. A *design concept* is a non-empty subset of a design space (extensional definition).

This definition implies that design spaces are also design concepts and that every concept is located within a design space. The difference between a design concept and a design space lies in whether something is defined intentionally or extensionally. A design space describes design concepts intentionally and it is structured at its most basic level by base design dimensions. On the other hand, design concepts, whether abstract or concrete, are extensional samples of a design space. We then consider base design dimensions as the most basic building blocks to represent not only design spaces, but any design concept x_L:

$$x_L := \begin{cases} \langle x_1, \dots, x_{|\mathcal{D}_L|} \mid x_i \subseteq \operatorname{ran}(\delta_{L_i}) \wedge \delta_{L_i} \in \Delta_L \forall i \in \{1..|\mathcal{D}_L|\} \rangle & \text{(intensionally)} \\ x \subseteq \mathcal{D}_L & \text{(extensionally)} \end{cases},$$

where $|\mathcal{D}_L|$ is the number of design dimensions that compose the space $\mathcal{D}_L$.

A designer constructs a design concept by setting regions on each of the constitutive dimensions of the design space. A region may span the full range of the dimension, a subrange, or a singleton set (a single value). This approach allows the designer to produce or represent a broad range of design concepts, from the most general, where no dimension is constrained (i.e., the underlying design space itself), to non-concrete concepts (e.g., conceptual objects, family of objects), to concrete objects, where all dimensions have a particular and valid value assigned. This construction pattern for building design concepts (and spaces) models the *constrained* and *decision-making nature* of designing described by Gero (1990), a designer requires to decide not only which design dimensions represent the class of the design object (design space modeling), but the values of these dimensions that represent a potential solution (design concept modeling). These decisions may, however, be subject to some externally defined or self-imposed constraints (the constrained nature of the activity of designing), which can be represented by the same working set of design dimensions.

In our approach, the notions of empty and partial concepts defined in Wiggins' CSF take a slightly different interpretation. The "indetermination" status of the value of a dimension is modeled by leaving that dimension unconstrained, that is, an empty design concept will span the full underlying design space, and a partial design concept will have some of its dimensions unconstrained. Our schema fulfills the empty concept axiom defined in the CSF since every design space will include an empty design concept:

$$\prod \left\langle x_1, \ldots, x_{|\mathcal{D}_L|} \,\middle|\, x_i = \operatorname{ran}\left(\delta_{L_i}\right) \wedge \delta_{L_i} \in \Delta_L \ \forall i \in \{1..|\mathcal{D}_L|\} \right\rangle = \operatorname{ran}(\mathcal{D}_L).$$

From this perspective, the activity of designing could also be regarded as a process of progressive constraining/loosening of the design space (in addition to adding/removing dimensions) up to the desired level of concreteness of the design concept (Fig. 3.8).

We provide a formal *structural recursive construction* $\boldsymbol{\Gamma}$ to build all and every possible space, from base dimensions to compound dimensions, to design spaces, up-to the universe:

$$\boldsymbol{\Gamma}_k(X) := \begin{cases} X & : k = 0 \\ \bigcup\!\!\!\!\!\!\!\hat{\times}\;\; \boldsymbol{\Gamma}_{k-1}(X) & : k = 1, k = 2 \\ \bigcup_{i=1}^{2} \boldsymbol{\Gamma}_i(X) \cup \bigcup\!\!\!\!\!\!\!\hat{\times}\;\; \bigcup_{i=1}^{2} \boldsymbol{\Gamma}_i(X) & : k = 3 \\ \boldsymbol{\Gamma}_{k-1}(X) \cup \bigcup\!\!\!\!\!\!\!\hat{\times}\;\; \boldsymbol{\Gamma}_{k-1}(X) & : k > 3 \end{cases}$$

$$\boldsymbol{\Gamma}(X) := \bigcup_{k=2}^{\infty} \boldsymbol{\Gamma}_k(X)$$

Where $\bigcup\!\!\!\!\!\!\!\hat{\times}\;\; \boldsymbol{S}$ is defined as $\bigcup_{n=1}^{\infty} \boldsymbol{S}^n$, and $\boldsymbol{S}^n$ is defined as the cartesian power

$$\left\{ \langle x_1, \ldots, x_n \rangle \,\middle|\, x_i \in \boldsymbol{S} \ \forall i \in \{1..n\} \right\}.$$

From this construction and a set of all possible base design dimensions $\dot{\Delta}_L$, one could identify the key spaces (and many more) to be used in our framework: only base dimensions $\boldsymbol{\Gamma}_0(\Delta_L)$,

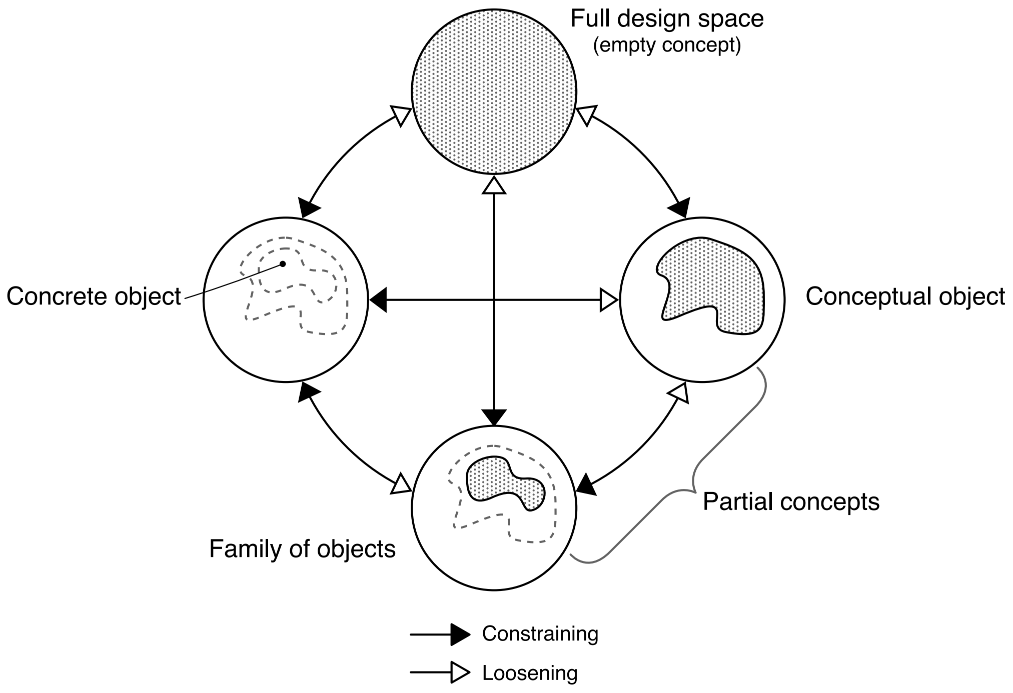


Figure 3.8: The full shaded circle at the top represents a concept corresponding to the full underlying design space (i.e., empty concept). The designer may constrain (black arrows) or loosen (white arrows) the space within a range of concreteness of the design concept, from/to partial concepts (right and bottom shaded shapes) to/from specific points that represent fully concrete objects (left space) in the space.

only non-nested compound dimensions $\boldsymbol{\Gamma}_1(\Delta_L)$, only non-nested design spaces or 1-nested compound dimensions $\boldsymbol{\Gamma}_2(\Delta_L)$, up-to $(k-1)$-nested compound dimensions and design spaces $\boldsymbol{\Gamma}_{k>2}(\Delta_L)$, and finally the whole universe of design spaces $\boldsymbol{\Gamma}(\Delta_L)$.

From this procedure, we also provide a formal description of the construction of all and every possible design concept within a universe:

$$\{x_{L_k}\} \coloneqq \boldsymbol{\Gamma}_k\Big(\bigcup\nolimits_f \Delta_L\Big),$$

where k sets the underlying space: $k = 1$ to build design properties[3] based on non-nested compound dimensions, $k = 2$ to build concepts within non-nested design spaces, and $k > 2$ to build concepts within nested design spaces ($\boldsymbol{\Gamma}$ without subscript parameter would mean building concepts within the whole universe including n-nested compound dimensions and design spaces);

$$\bigcup\nolimits_f X \coloneqq \bigcup\Big\{ f(x) \Big| x \in X \Big\};$$

and f is a function that sets the desired output abstraction level: $\wp_{\geq 1} \circ \mathrm{ran}$, a function composition (power set of range) for all possible concepts (abstract and concrete), $\wp_{=1} \circ \mathrm{ran}$ for only concrete objects, and $\wp_{>1} \circ \mathrm{ran}$ for only abstract concepts. For completeness, $\wp_{\geq 0} \circ \mathrm{ran}$ would include partial (and empty) concepts according to the original definition in the CSF. The $\wp$ (power set) with a subscript modifier filters the resulting power set to subsets with cardinality determined by the modifier.

These construction procedures for spaces and concepts are provided for formalization purposes and to set a clear notion of what the universe, design spaces, and dimensions are; how concepts are structured on those spaces; and to illustrate the combinatorial explosion that involves designing (and similar creative domains).

3.1.3 Design process

When operating on a design project, design agents perform the activity of designing by adhering to a specific design process (a series of steps to come up with a designed object), whether holistically (e.g., conceptual design in the engineering design process), or by comprising designing different facets or components of an object separately (e.g., game levels, rules, sounds, and characters in game-mechanics design); that is, separated yet interrelated design projects for each facet or component[4].

[3]According to Gärdenfors (2004), a property is a region in a domain. Therefore, a design property would be a region in the space defined by a compound design dimension (e.g., bluish, greenish, and purplish are properties within the RGB color compound design dimension).

[4]In this work we focus on single design processes, that is, an holistic process or the design of a single facet or component.

Based on the eight fundamental processes for designing described by the FBS/sFBS frameworks, we propose to model a specific design process in terms of six generalized fundamental processes for designing that transform design concepts within and across the three sFBS situational worlds or spheres of the designer.

3.1.3.1 Modeling the design process through fundamental processes for designing

Definition 3.1.6. A *fundamental process for designing* is a process that renders a major stage in design progression and is instrumented by a series of intra-world and inter-world related operations performed by one or more design agents on a set of design concepts within the scope of a design layer to formulate and re-formulate the design problem and to produce and assess potential solutions.

1. *Formulation*: Transforms a set of external or memory-recalled design concepts representing requirements and constraints into a set of expected design concepts that settles the evaluation and solution spaces.
2. *Synthesis*: Transforms a set of expected design concepts defined in terms of the evaluation space dimensions into a set of design concepts representing potential solutions within the solution space.
3. *Analysis*: Interprets and transforms a set of design concepts (including potential solutions) into a set of higher-level design concepts defined in terms of the evaluation space.
4. *Evaluation*: Assesses potential design solutions or external design concepts by comparing them against expected design concepts.
5. *Documentation*: Builds external design descriptions of produced design concepts.
6. *Reformulation*: Modifies (or recreates) the evaluation or solution spaces (or both) from new or modified requirements and constraints, produced solutions performance, reinterpretation, or new inspiring sets.

FBS/sFBS consider the fundamental processes for designing as processes spanning the *full design project* (from the initial highest level requirements to the lowest level, produced design concepts). Although in our schema, the fundamental processes span a single design layer, it is also possible to recognize those processes from the perspective of the full design project (all stacked layers) (Fig. 3.9). The latter can be accomplished by: a) accumulating formulation, synthesis, analysis, and evaluation of a single layer as part of the formulation process of the layer below (down to the lowest layer); b) grouping documentation processes of each layer into the global documentation process; and c) considering synthesis, analysis, and evaluation processes at the lowest layer as full-level design processes.

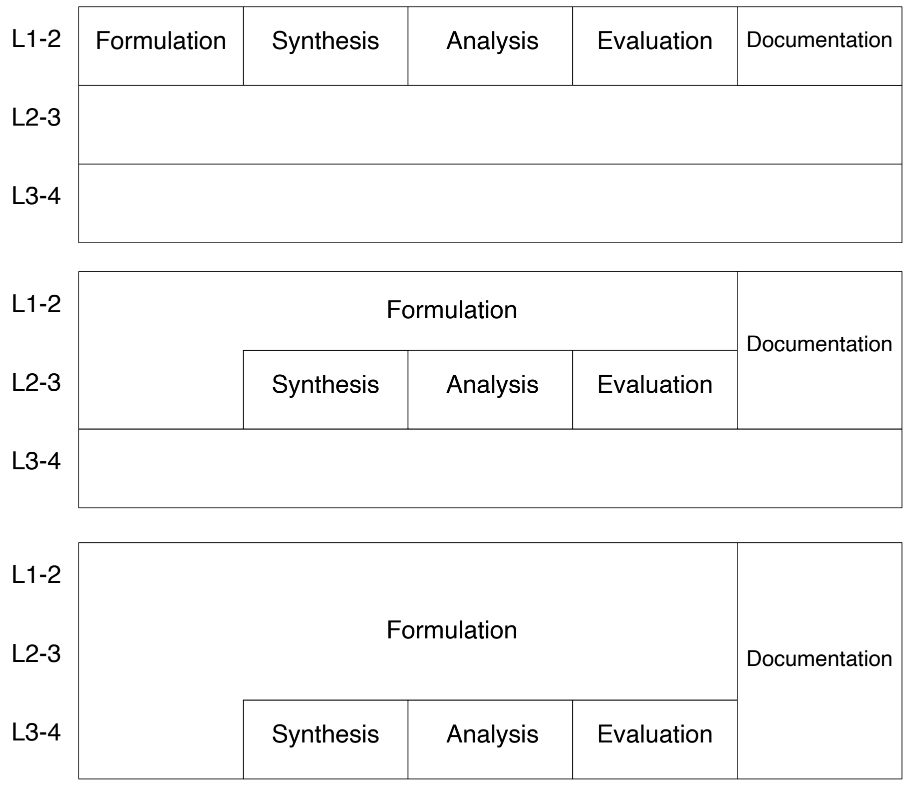


Figure 3.9: Full design project processes

3.1.3.2 Instrumenting the fundamental processes for designing through perception-action strategies

Based on and generalized from the 20 specialized sFBS design operations, we propose that the underlying operations of the fundamental processes for designing can be instrumented by agent-specific instances of perception and action strategies (procedural knowledge) that capture the situated nature of designing as well as general and domain-specific design capabilities of design agents.

We employ the symbol X_L to refer to a set of design concepts represented at L design level; and $\hat{X}_L$, $\check{X}_L$, and $\vec{X}_L$ to indicate that the design concepts reside in the external, interpreted or expected world of the agent, respectively.

3.1.3.2.1 Perception strategies

- *Interpretation* $\hat{X}_L \rightrightarrows \check{X}_L$: Creates or updates internal versions (interpretations) of external design concepts under the influence of current expectations.
- *Reflection* $\check{X}_L \rightrightarrows \check{X}_L'$: Augments or modifies interpreted design concepts based on experience, knowledge, expectations, and preferences.

- *Focusing* $\check{X}_L \rightleftarrows \vec{X}_L$: Sets or updates current goals and expectations by transferring selected design concepts from the interpreted world into the expected world, and from the expected world back to the interpreted world (sets the current situation design space).
- *Comparison* $\vec{X}_L \leftrightarrow \check{X}_L$: Contrasts design concepts (already internalized) with expected[5] design concepts.

3.1.3.2.2 Action strategies

- *Synthetical derivation* $\vec{X}_L \Mapsto \vec{X}_{L^-}$: Produces expected design concepts within the solution space from expected design concepts defined in terms of the evaluation space.
- *Analytical derivation* $\check{X}_L \mapsto \check{X}_{L^+}$: Produces internal design concepts defined in terms of the evaluation space from interpreted design concepts defined in terms of the solution space.
- *Externalization* $\vec{X}_L \rightarrow \hat{X}_L$: Creates or modifies external design concepts from expected design concepts.

A *design layer*, which can be considered as a design process on its own, is modeled by instances of each of the fundamental processes instrumented by instances of perception and action strategies (Fig. 3.10).

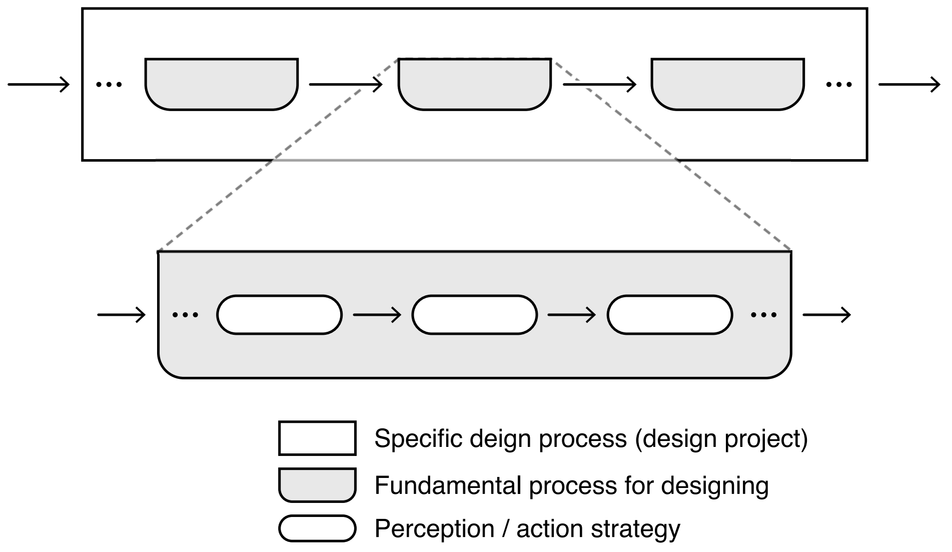

Specific deign process (design project)
Fundamental process for designing
Perception / action strategy

Figure 3.10: Design process modeling through fundamental processes for designing and perception/action strategies

At any given design layer, and depending on the design agent role being modeled, the instances of perception-action strategies are required to be specialized in one or both of its bounding design levels. The design layer may require the design agent to know working sets of design

[5]The term “expected” denotes that the concepts reside in the expected world and represent potential (“mental” or simulated) solutions that will hopefully satisfy the design requirements (evaluation space concepts) when they become external and analyzed.

dimensions (and perhaps design concepts) to build or traverse both the evaluation space and the solution space of that layer.

When considering the FBS ontological categories, we hypothesize that deriving expected behavior from required functions might be operationally equivalent to derive structure from expected behavior, and by applying the same rationale, the process of deriving actual behavior from a structure might be operationally equivalent to derive actual functions from behaviors. This pattern leads us to propose *synthetical* and *analytical* derivation strategies, and to define *synthesis* and *analysis* as fundamental processes available at any given design layer.

3.2 Computational Creative Design Systems

We propose to define a computational creative design system (CCDS) of n design levels[6] through a Layered Designing Schema of $n-1$ layers, where each layer is an *exploratory creative system (ECS)* that is defined using an extended version of Wiggins' CSF (as illustrated in Fig. 3.11). This architecture not only reflects our proposed schema but enables fine-grained task and role modeling. It is worth noting that the hierarchical arrangement implies by no means a sequential execution of layers, so two or more layers can be executed simultaneously.

3.2.1 Exploratory Creative Systems

In terms of the CSF we formally describe a CCDS layer as:

$$\left\langle \mathcal{L}, \left\langle \langle \Delta_{LL^-}, \mathcal{U}_{LL^-} \rangle, \langle \mathring{\Delta}_{LL^-}, \mathring{\mathcal{U}}_{LL^-} \rangle \right\rangle, [\![\cdot, \cdot]\!], \langle\!\langle \cdot, \cdot, \cdot, \cdot \rangle\!\rangle, \mathscr{R}_{LL^-}, \mathscr{E}_{LL^-}, \mathscr{P}_{LL^-}, \mathscr{T}_{LL^-} \right\rangle,$$

where

- $\mathcal{L}$ is a language used to express rulesets (including design dimensions, concepts, and spaces);
- $\langle \Delta_{LL^-}, \mathcal{U}_{LL^-} \rangle$, $\langle \mathring{\Delta}_{LL^-}, \mathring{\mathcal{U}}_{LL^-} \rangle$ represent the universe and designer universe, respectively;
- $[\![\cdot, \cdot]\!]$ and $\langle\!\langle \cdot, \cdot, \cdot, \cdot \rangle\!\rangle$ are extended versions of selection and exploration interpreters;
- and $\mathscr{R}_{LL^-}$, $\mathscr{E}_{LL^-}$, $\mathscr{P}_{LL^-}$, and $\mathscr{T}_{LL^-}$ are definitional, evaluative, perception, and action rulesets, respectively.

We propose that the rulesets can be constituted by both *declarative* (design concepts and dimensions) and *procedural* (perception/action strategies) knowledge provided by the designing schema.

[6]The n^{th} level corresponds to the highest design level

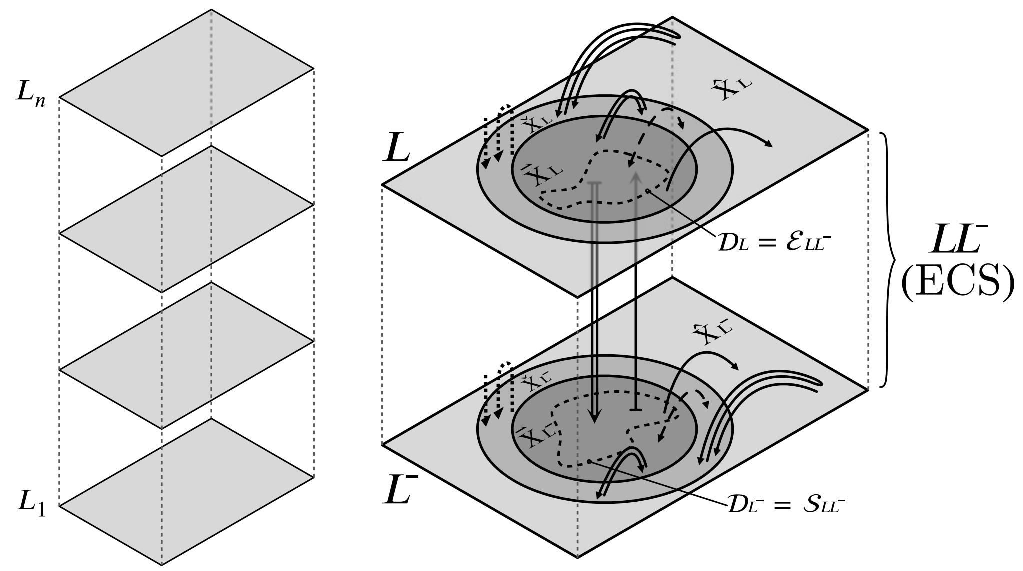

L_n
L_1
L
L^-
$\mathcal{D}_L = \mathcal{E}_{LL^-}$
LL^-
(ECS)
$\mathcal{D}_{L^-} = \mathcal{S}_{LL^-}$

Figure 3.11: CCDS architecture. The CCDS (on the left) is architectured as an arrangement of stacked extended ECSs, where each ECS (on the right) realizes one layer of the underlying designing schema. A CCDS layer is implemented by its own set of perception/action strategies, which connect design concepts located in the external, interpreted, and expected worlds of the designer, including design concepts (residing in the expected worlds) that constitute each of the design spaces of the layer. [Designer's worlds and design operations' symbols inspired on sFBS authors' original work in (Gero and Kannengiesser 2014)].

3.2.1.1 Universe and designer universe

In our schema, the universe $\mathcal{U}_L$ defined in Wiggins' CSF (Wiggins 2006) would be not only an infinite design space containing all possible design concepts representable by another (universal and most likely infinite) set Δ_L of all possible design dimensions in the context of an L design level but also the set of all possible design spaces within that level.

A consequence of the situated nature of designing is that different designers will come up with different interpretations and designs even if they started from the same set of initial requirements and externally defined constraints; furthermore, from the same requirements, a designer may achieve different designs in different design sessions. This condition is the result not only of the diversity of perception-action strategies among designers but their particular knowledge, experience, and preferences. We then propose to separate the notions of universe $\mathcal{U}_L$ and *designer universe* $\mathring{\mathcal{U}}_L \subset \mathcal{U}_L$, if the former is the set of all possible design concepts describable from the set Δ_L of all possible design dimensions, we define the latter as the set of all possible design concepts representable by the set $\mathring{\Delta}_L$ of design dimensions known to a particular designer (built-in and acquired) (Fig. 3.12a). The designer universe sets the creative boundaries of the designer (i.e., the design agent will not be able to operate on whatever he/she/it is unable to conceive) (Fig. 3.12b).

The relative complement $\mathcal{U}_L \setminus \mathring{\mathcal{U}}_L$ of the designer universe with respect to the universe sets the room not only for potential expansion of the agent's knowledge either by experience, training, learning, or reasoning but for the design agent to operate towards CSF's $\mathscr{R}/\mathscr{E}$-transformational creativity without requiring meta-creative exploratory systems. The distinction between the universe and designer universe is also required to enable collaborative scenarios, where different agents operate on their universes but share a region (intersection of their universes) within a bigger universe; if this region did not exist, the collaboration between agents would be impossible:

$$\bigcap\left\{\mathring{\mathcal{U}}_{L_1}, \dots, \mathring{\mathcal{U}}_{L_n} \,\middle|\, \mathring{\mathcal{U}}_{L_i} \subset \mathcal{U}_L \; \forall i \in \{1..n\}\right\} \neq \varnothing.$$

The designer universe comprises part of the declarative knowledge of the design agent. Therefore, in order to discern between effectively created artifacts (during the current design session and previous design sessions), and conceivable yet not experienced ones, we define two partitions of the designer universe, one containing the designer experience $\mathring{\mathcal{U}}^{!}_{L} \subseteq \mathring{\mathcal{U}}_L$, and the other covering the parts of the designer universe where the agent lacks experience $\mathring{\mathcal{U}}^{?}_{L} \subseteq \mathring{\mathcal{U}}_L$ (Fig. 3.12a).

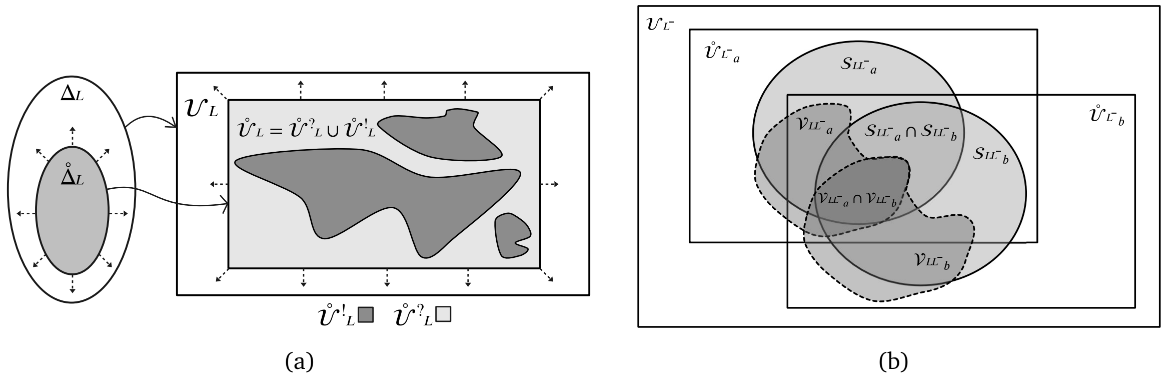

Δ_L
$\mathring{\Delta}_L$
$\mathcal{U}_L$
$\mathring{\mathcal{U}}_L = \mathring{\mathcal{U}}^{?}_{L} \cup \mathring{\mathcal{U}}^{!}_{L}$
$\mathring{\mathcal{U}}^{!}_{L}$ ■ $\mathring{\mathcal{U}}^{?}_{L}$ □
(a)
$\mathcal{U}_{L^-}$
$\mathring{\mathcal{U}}_{L^-_a}$
$\mathcal{S}_{LL^-_a}$
$\mathring{\mathcal{U}}_{L^-_b}$
$\mathcal{V}_{LL^-_a}$
$\mathcal{S}_{LL^-_a} \cap \mathcal{S}_{LL^-_b}$
$\mathcal{S}_{LL^-_b}$
$\mathcal{V}_{LL^-_a} \cap \mathcal{V}_{LL^-_b}$
$\mathcal{V}_{LL^-_b}$
(b)

Figure 3.12: (a): Universe $\mathcal{U}_L$ and designer universe $\mathring{\mathcal{U}}_L$ spanned by a universal set of design dimensions Δ_L and known dimensions to the designer $\mathring{\Delta}_L$, respectively. The designer universe is partitioned into the designer experience $\mathring{\mathcal{U}}^{!}_{L}$ and its complement $\mathring{\mathcal{U}}^{?}_{L}$. (b): The intersection of the solution spaces $\mathcal{S}_{LL^-_a}$ and $\mathcal{S}_{LL^-_b}$ of two participant designers a and b, as well as the sets $\mathcal{V}_{LL^-_a}$ and $\mathcal{V}_{LL^-_b}$ (and their intersection) of valued design concepts which are the concepts selected by the interpreter $[\![\cdot, \cdot]\!]$ according to evaluative rulesets of the designers defined in terms of their corresponding evaluation spaces $\mathcal{E}_{LL^-_a}$ and $\mathcal{E}_{LL^-_b}$.

3.2.1.2 Definitional $\mathscr{R}$ and evaluative $\mathscr{E}$ rulesets

Requirements, expectations, constraints, inspiring sets, and working sets of design dimensions constitute declarative knowledge in the form of design dimensions and concepts that can be encoded into definitional $\mathscr{R}$ and evaluative $\mathscr{E}$ rulesets to assist a design agent in setting the criteria to assess the validity and value of design concepts.

In the context of an LL^- design layer (an ECS), $\mathscr{R}_{LL^-}$ and $\mathscr{E}_{LL^-}$ rulesets would define the solution $\mathcal{S}_{LL^-}$ and evaluation $\mathcal{E}_{LL^-}$ design spaces, respectively. Also, evaluative $\mathscr{E}_{LL^-}$ rulesets of non-top layers would need to include the set $\langle\!\langle \mathscr{R}_{L^+L}, \mathscr{E}_{L^+L}, \mathscr{P}_{L^+L}, \mathscr{T}_{L^+L} \rangle\!\rangle^{\diamond}(\varnothing)$ of design concepts produced at the layer above, corresponding to the LL^- expected design objects.

In addition to declarative knowledge, $\mathscr{R}$ and $\mathscr{E}$ rulesets may include cultural and domain agreed procedural knowledge in the form of analytical derivation $\mapsto$, focusing $\rightleftarrows$, and comparison $\leftrightarrow$ strategies that implement "standard" procedures to apply the definitional or evaluative criteria set by declarative knowledge.

$$\mathscr{R}_{LL^-} := \left\{ \underbrace{X_{L^-},\ \hat{X}_{L^-}}_{L^-\text{ requirements \& constraints}},\ \underbrace{\Delta_{L^-}' \subset \Delta_{L^-}}_{L^-\text{ working set of design dimensions}},\ \underbrace{\left\{\left(\check{X}_{L^-} \rightleftarrows \vec{X}_{L^-}\right)_k\right\}}_{L^-\text{ standard focussing strategies}},\ \underbrace{\left\{\left(\vec{X}_{L^-} \leftrightarrow \check{X}_{L^-}\right)_k\right\}}_{L^-\text{ standar comparison strategies}} \right\}$$

$$\mathscr{E}_{LL^-} := \left\{ \underbrace{X_L,\ \hat{X}_L}_{L\text{ requirements \& constraints}},\ \underbrace{X_L \supset \langle\!\langle \mathscr{R}_{L^+L},\ \mathscr{E}_{L^+L},\ \mathscr{P}_{L^+L},\ \mathscr{T}_{L^+L} \rangle\!\rangle^{\diamond}(\varnothing)}_{\text{concepts produced at the} L^+L\text{ layer}},\ \underbrace{\Delta_L' \subset \Delta_L,\ \Delta_{L^-}' \subset \Delta_{L^-}}_{L,L^-\text{ working sets of design dimensions}},\ \underbrace{\left\{\left(\check{X}_L \rightleftarrows \vec{X}_L\right)_k\right\}}_{L\text{ standar focussing strategies}},\ \underbrace{\left\{\left(\vec{X}_L \leftrightarrow \check{X}_L\right)_k\right\}}_{L\text{ standar comparison strategies}},\ \underbrace{\left\{\left(\check{X}_{L^-} \mapsto \check{X}_L\right)_k\right\}}_{\text{standar analytical derivation strategies}} \right\}$$

3.2.1.3 Perception $\mathscr{P}$ and action $\mathscr{T}$ rulesets

Wiggins introduced a $\mathscr{T}$ ruleset in his framework to allow the implementation of different strategies to traverse the conceptual space (and universe) under the influence or independently of definitional and evaluative rulesets. From our designing schema, the design space traversal is directed by design decisions formulated through perception and action strategies.

We then introduce a new perception $\mathscr{P}$ ruleset to encode different perception strategies and keep $\mathscr{T}$ ruleset to encode different action strategies. These rulesets make explicit the situated nature of designing and enable modeling more realistic scenarios (e.g., a highly skilled and experienced designer, a "talented" executer yet inexperienced designer, different self-motivation and preferences, different perception models - same style of execution, or same perception model - different styles of execution).

$$
\mathscr{P}_{LL^{-}} := \left\{ \underbrace{\left\{\left(\hat{X}_{L} \subsetneq \check{X}_{L}\right)_k\right\}, \left\{\left(\hat{X}_{L^{-}} \subsetneq \check{X}_{L^{-}}\right)_k\right\}}_{L,\, L^{-} \text{ interpretation strategies}},\ \underbrace{\left\{\left(\check{X}_{L} \mathrel{\overset{\dashrightarrow}{\dashrightarrow}} \check{X}_{L}'\right)_k\right\}, \left\{\left(\check{X}_{L^{-}} \mathrel{\overset{\dashrightarrow}{\dashrightarrow}} \check{X}_{L^{-}}'\right)_k\right\}}_{L,\, L^{-} \text{ reflection strategies}}, \right.
$$

$$
\left. \underbrace{\left\{\left(\check{X}_{L} \rightleftarrows \vec{X}_{L}\right)_k\right\}, \left\{\left(\check{X}_{L^{-}} \rightleftarrows \vec{X}_{L^{-}}\right)_k\right\}}_{L,\, L^{-} \text{ focusing strategies}},\ \underbrace{\left\{\left(\check{X}_{L} \leftrightarrow \check{X}_{L}'\right)_k\right\}, \left\{\left(\check{X}_{L^{-}} \leftrightarrow \check{X}_{L^{-}}'\right)_k\right\}}_{L,\, L^{-} \text{ comparison strategies}} \right\}
$$

$$
\mathscr{T}_{LL^{-}} := \left\{ \underbrace{\left\{\left(\vec{X}_{L} \Mapsto \vec{X}_{L^{-}}\right)_k\right\}}_{\text{synthetical derivation strategies}},\ \underbrace{\left\{\left(\vec{X}_{L} \rightarrow \hat{X}_{L}'\right)_k\right\}, \left\{\left(\vec{X}_{L^{-}} \rightarrow \hat{X}_{L^{-}}'\right)_k\right\}}_{L,\, L^{-} \text{ externalization strategies}}, \right.
$$

$$
\left. \underbrace{\left\{\left(\check{X}_{L^{-}} \mapsto \check{X}_{L}\right)_k\right\}}_{\text{analytical derivation strategies}} \right\}
$$

3.2.1.4 Interpreters

In our CCDS, the CSF exploration interpreter $\langle\!\langle \cdot,\ \cdot,\ \cdot \rangle\!\rangle$ is now directed by $\mathscr{P}$ and $\mathscr{T}$ rulesets (i.e., $\langle\!\langle \mathscr{R},\ \mathscr{E},\ \mathscr{P},\ \mathscr{T} \rangle\!\rangle$), enabling the production of design concepts from different, perhaps divergent interpretations of the same design requirements and expectations.

The selection interpreter $[\![\cdot]\!]$, as described in the CSF, should select the same subset of design concepts when applied to the same set of design concepts and is informed by the same $\mathscr{R}$ and $\mathscr{E}$ ruleset. This condition, however, does not reflect the bias introduced by the perception of specific design agents. To remedy this situation, we also modify the selection interpreter to be influenced by $\mathscr{P}$ ruleset (i.e., $[\![\mathscr{R},\ \mathscr{P}]\!]$ and $[\![\mathscr{E},\ \mathscr{P}]\!]$).

Our CCDS selection interpreter then allows modeling a more precise scenario where different agents having diverse experiences, motivations, objectivity or subjectivity levels, and preferences, interpret the same definitional or evaluative rulesets but not necessarily come to an agreement regarding the validity or value of a given design concept.

3.2.1.5 Design agent self-defined rulesets

When design agents interpret, augment, and reflect on design dimensions and concepts encoded into externally defined $\mathscr{R}$ and $\mathscr{E}$ rulesets, they are actually producing their own versions of rulesets $\mathring{\mathscr{R}}$ and $\mathring{\mathscr{E}}$. As designing progresses, design agents are, in principle, free to modify their rulesets according to each design situation. Even in the absence of explicitly defined external rules (a self-initiative design session) or ill-defined, design agents may build their rulesets from experience or by interpreting design concepts from an external inspiring set.

3.3 Collaborative creative design systems

A collaborative creative design system would require at least two design agents working in a common design assignment or project, although sharing the same responsibilities in the design process is not required to establish collaboration, which can be established between human and artificial agents, or between artificial agents exclusively[7].

3.3.1 Interaction mechanism

Collaborative design agents require some means of interaction, which could be modeled as a message interchange mechanism between agents instrumented by extended versions of perception and action strategies encoded into $\mathscr{P}$ and $\mathscr{T}$ rulesets and additional classes of dimensions[8] (e.g., interaction, communication dimensions). A request or input message is interpreted by the receiver agent based on current expectations and design situation; then, the agent may execute definitional, evaluative, or action strategies accordingly ($\mathscr{R}$, $\mathscr{E}$, or $\mathscr{T}$ rulesets), and finally prepare and send back a response or a new request message via externalization.

3.3.2 Shared and individual rulesets

In a collaborative scenario, a design agent creates and maintains (through the agent's interpretation and reflection strategies) individual $\mathring{\mathscr{R}}$ and $\mathring{\mathscr{E}}$ rulesets[9] based on common $\mathscr{R}$ and $\mathscr{E}$ rulesets shared by all participant agents:

$$\mathscr{R} \rightrightarrows \mathring{\mathscr{R}} \dashrightarrow \mathring{\mathscr{R}},\ \mathscr{E} \rightrightarrows \mathring{\mathscr{E}} \dashrightarrow \mathring{\mathscr{E}}$$

At any given design situation s, common rulesets could be modified ($\mathscr{R}/\mathscr{E}$-transformational) by promoting some rules or subsets of them from the ambit of one or more agents capable of reformulation, with or without keeping all or a subset of previously common rules:

$$\mathscr{R}\mathit{LL}^{-}{}_{(s)} = \mathscr{R}\mathit{LL}^{-}{}_{(s-1)}{}' \subseteq \mathscr{R}\mathit{LL}^{-}{}_{(s-1)} \cup \mathring{\mathscr{R}}\mathit{LL}^{-}{}_{a}{}' \subseteq \mathring{\mathscr{R}}\mathit{LL}^{-}{}_{a} \cup \mathring{\mathscr{R}}\mathit{LL}^{-}{}_{b}{}' \subseteq \mathring{\mathscr{R}}\mathit{LL}^{-}{}_{b} \cup \dots,$$

$$\mathscr{E}\mathit{LL}^{-}{}_{(s)} = \mathscr{E}\mathit{LL}^{-}{}_{(s-1)}{}' \subseteq \mathscr{E}\mathit{LL}^{-}{}_{(s-1)} \cup \mathring{\mathscr{E}}\mathit{LL}^{-}{}_{a}{}' \subseteq \mathring{\mathscr{E}}\mathit{LL}^{-}{}_{a} \cup \mathring{\mathscr{E}}\mathit{LL}^{-}{}_{b}{}' \subseteq \mathring{\mathscr{E}}\mathit{LL}^{-}{}_{b} \cup \dots;$$

[7]In this work we are focussing on human-computer collaboration schemas

[8]Gärdenfors CST is conceived as a general-purpose theory of representation through concepts.

[9]In particular, the rules regarding to declarative knowledge in the form of design concepts and dimensions

furthermore, one or more design agents can also be the source of initial common $\mathscr{R}$ and $\mathscr{E}$ rulesets or subsets of these rules (e.g., a project leader, most experienced designer):

$$\mathscr{R}_{LL^-} = \mathring{\mathscr{R}}_{LL^-{}_a}{}' \subseteq \mathring{\mathscr{R}}_{LL^-{}_a} \cup \mathring{\mathscr{R}}_{LL^-{}_b}{}' \subseteq \mathring{\mathscr{R}}_{LL^-{}_b} \cup \dots,$$

$$\mathscr{E}_{LL^-} = \mathring{\mathscr{E}}_{LL^-{}_a}{}' \subseteq \mathring{\mathscr{E}}_{LL^-{}_a} \cup \mathring{\mathscr{E}}_{LL^-{}_b}{}' \subseteq \mathring{\mathscr{E}}_{LL^-{}_b} \cup \dots$$

3.3.3 Collaboration modes

We consider *task-divided collaboration* (Kantosalo and Toivonen 2016) and co-creative collaboration as two iconic collaboration modes within a range of possibilities. When considering human design teams, they usually fluctuate between different collaboration modes and could even switch to periods of autonomous work to incorporate to teamwork afterwhile. It is also common to outsource some non-critical tasks (depending on the particular design object and process) to focus on critical ones without affecting the final result.

If collaboration modes are combined, they may provide flexibility to the agents and probably be useful as a strategy to resolve a stalled design session, to take advantage of highly skilled agents in selected processes for designing, or to improve design agents' performance by expanding their universes, or by learning from their colleagues. These types of switching may become triggers for moving the design situation to a more favorable region of the space of possibilities.

3.3.3.1 Task-divided collaboration

In a *task-divided collaboration* mode, two or more incomplete design agents (do not perform the whole set of fundamental processes for designing, either purposely or for being unskilled) participate by performing different tasks or roles which are modeled by instances of fundamental processes for designing and their underlying perception and action strategies ($\mathscr{P}$ and $\mathscr{T}$ rulesets). Our schema allows defining fine-grained roles ranging from full design schema roles down to level-specific design roles. A collaborative incomplete agent a must be equipped with at least one externalization strategy encoded into the agent's action ruleset $\mathscr{T}_{LL^-{}_a}$ for each design level covered by this agent; and a full perception ruleset $\mathscr{P}_{LL^-{}_a}$ implementing interpretation, reflection, focusing, and comparison strategies for each covered design level. Depending on the agent role being modeled, the agent will also be required to implement additional action strategies encoded in the $\mathscr{T}_{LL^-{}_a}$ or vía individual definitional $\mathring{\mathscr{R}}_{LL^-{}_a}$ or evaluative $\mathring{\mathscr{E}}_{LL^-{}_a}$ rulesets. The formal description of a task-divided collaboration mode depends on the specific roles of the participant agents (see example in appendix 4).

3.3.3.2 Co-creative collaboration

In co-creative collaboration mode, two or more complete agents work in a common assignment or project, share a design object, and share the same responsibilities. In terms of Wiggins' CSF, a design layer in co-creative collaboration mode can formally be defined as:

$$\Big\langle \mathcal{L}, \Big\langle \langle \Delta_{LL^-}, \mathcal{U}_{LL^-} \rangle, \Big\langle \bigcup_{a\in A} \mathring{\Delta}_{LL^-{}_a}, \bigcup_{a\in A} \mathring{\mathcal{U}}_{LL^-{}_a} \Big\rangle \Big\rangle, [\![\cdot, \cdot]\!], \langle\!\langle \cdot, \cdot, \cdot, \cdot \rangle\!\rangle,$$

$$\mathscr{R}_{LL^-} \cup \Big(\bigcup_{a\in A} \mathring{\mathscr{R}}_{LL^-{}_a} \Big), \mathscr{E}_{LL^-} \cup \Big(\bigcup_{a\in A} \mathring{\mathscr{E}}_{LL^-{}_a} \Big), \bigcup_{a\in A} \mathscr{P}_{LL^-{}_a}, \bigcup_{a\in A} \mathscr{T}_{LL^-{}_a} \Big\rangle.$$

This definition incorporates both the designer universes and rulesets for each participant complete agent $a \in A$.

A possible interaction mechanism for a co-creative mode could be the *collaborative alternating mode* (Kantosalo and Toivonen 2016), where agents take turns to work on the design with or without the possibility to skip turns if needed or desired (symmetric vs. alternating co-creativity).

Another possible interaction approach is based on *mixed-initiative interaction* (Carbonell 1970; Novick and Sutton 1997; Yannakakis, Liapis, and Alexopoulos 2014). This mechanism establishes how the participant agents negotiate to determine which agent has control over the current design situation, how control can be switched to another agent, and how agents may take control over design. This kind of interaction may require an even more sophisticated extended set of perception and action strategies encoded into $\mathscr{P}$ and $\mathscr{T}$ rulesets as well as additional classes of dimensions.

4

Describing an existing system through the CCDSF

In this chapter, we describe an existing video game design tool called *Sentient Sketchbook* (Liapis, Yannakakis, and Togelius 2013) in terms of our proposed framework.

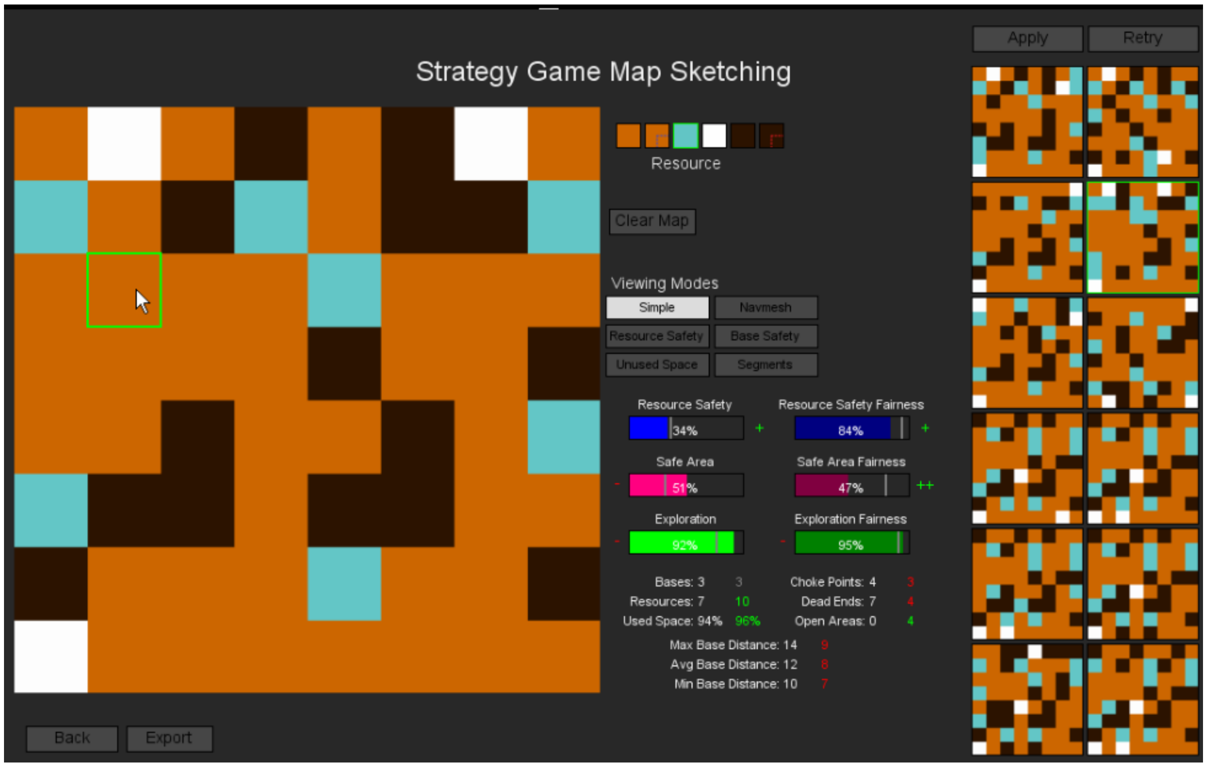
The Sentient Sketchbook is a tool that allows a human designer to sketch via a GUI, a low-resolution map (a small set of tiles) for a strategy game by setting each tile of the map to any of the available types: *passable*, *impassable*, *base*, or *resources*.

The low-resolution map is a high-level abstraction of a game level that contains only the essential game elements. A player is expected to start at a base tile (chosen at random) and collect resources in order to build units that must reach the enemy players' bases and destroy them.

As the human designer modifies the map, the tool determines in real-time the playability and gameplay enabled by the current map configuration, informs the human a series of behavioral dimensions regarding game pace, player balance and map navigation, and generates up-to twelve design suggestions inspired on the current map structure (six suggestions generated via genetic algorithm (GA) optimization on game pace and player balance measures of quality, and six suggestions generated via GA novelty search) from which the human may select an alternative and apply it as the new design state. Before deciding to apply a suggestion, the human designer can select a suggestion, and the tool will inform the quality of the selected suggestion side by side with the reported metrics of the current map (Fig. 4.1).

At any time and on human's request, the tool can produce different behavioral-based views of the current map in place of the default, simple view sim (Fig. 4.2).

Finally, on human's request, the tool is also able to generate a detailed final map, which is a high-resolution version of the sketch, enhanced with optional thematic styles that can be exported

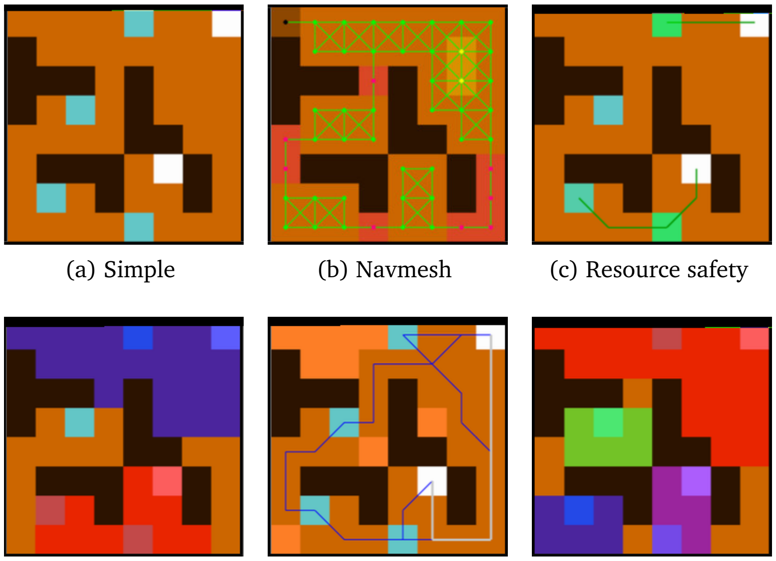


Resource Safety 34% +
Resource Safety Fairness 84% +
Safe Area - 51%
Safe Area Fairness 47% ++
Exploration - 92%
Exploration Fairness - 95%

```
Bases: 3          3        Choke Points: 4    3
Resources: 7      10       Dead Ends: 7       4
Used Space: 94%   96%      Open Areas: 0      4
      Max Base Distance: 14    9
      Avg Base Distance: 12    8
      Min Base Distance: 10    7
```

Figure 4.1: The Sentient Sketchbook tool during a design session. To the left is the sketch editor (the map), far to the right are the automatically generated map suggestions; between these elements is the tile palette, the map viewing modes menu and an overview of map's behavioral dimensions: game pace dimensions (bars on the left), player balance dimensions (bars on the right), map navigation metrics (bottom). The second suggestion of the second column appears as selected, its behavioral dimensions are reported side by side to the current map dimensions with some helper indicators: better performance (green), worse (red), equivalent (gray), larger value (+), smaller value (-), much larger (++), and much smaller (--) [Figure and caption reproduced/adapted from (Liapis, Yannakakis, and Togelius 2013) with permission from Antonios Liapis].

and used for game level generation (Fig. 4.3).


(a) Simple (b) Navmesh (c) Resource safety

(d) Safe area (e) Unused space (f) Segments

Figure 4.2: View modes: *Simple* (a) displays passable tiles (light brown), impassable tiles (dark brown), resources (cyan), and player bases (white). *Navmesh* (b) displays the passable pathways on the map and the location of choke points (red), dead ends (black), and open areas (yellow). *Resource safety* (c) displays the safety value of each resource (in shades of green) and connects bases to their safe resources. *Safe area* (d) shows the tiles close to each base (here in red and blue) which are considered safe. *Unused space* (e) shows all shortest paths between bases and resources, and highlights leftover unused space (orange). *Segments* (f) shows the passable areas (in different colors) which are surrounded solely by chokepoints (Figure and caption reproduced/adapted from (Liapis, Yannakakis, and Togelius 2013) with permission from Antonios Liapis).

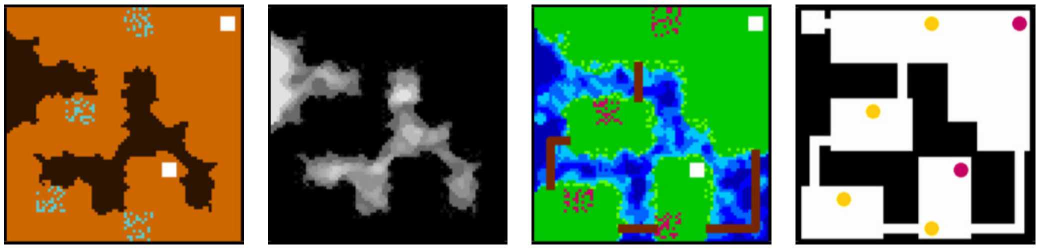

(a) Default $\hat{x}s_{ddef}$ (b) Heightmap $\hat{x}s_{dhm}$ (c) Waterways $\hat{x}s_{dww}$ (d) Dungeon $\hat{x}s_{ddun}$

Figure 4.3: Visualizations of the final map, which offers a higher-detail view of the map sketch in Fig. 4.2. The *Default* (a) visualization adds detail to resource tiles and impassable tiles. *Heightmap* (b) elaborates the impassable regions of Default. *Waterways* (c) treats impassable tiles and chokepoints as water and replaces chokepoints with bridges. *Dungeon* (d) divides the passable segments and dead ends into rooms and treats chokepoints as corridors (Figure and caption reproduced/adapted from (Liapis, Yannakakis, and Togelius 2013) with permission from Antonios Liapis).

4.1 Design levels

The design of game-level maps through the Sentient Sketchbook tool can be modeled with a hierarchy of three design levels based on the three FBS ontological categories and the «sketch» abstraction level explicitly covered by the tool: *functional-sketch* F, *behavioral-sketch* B, and *structural-sketch* S (Fig. 4.4).

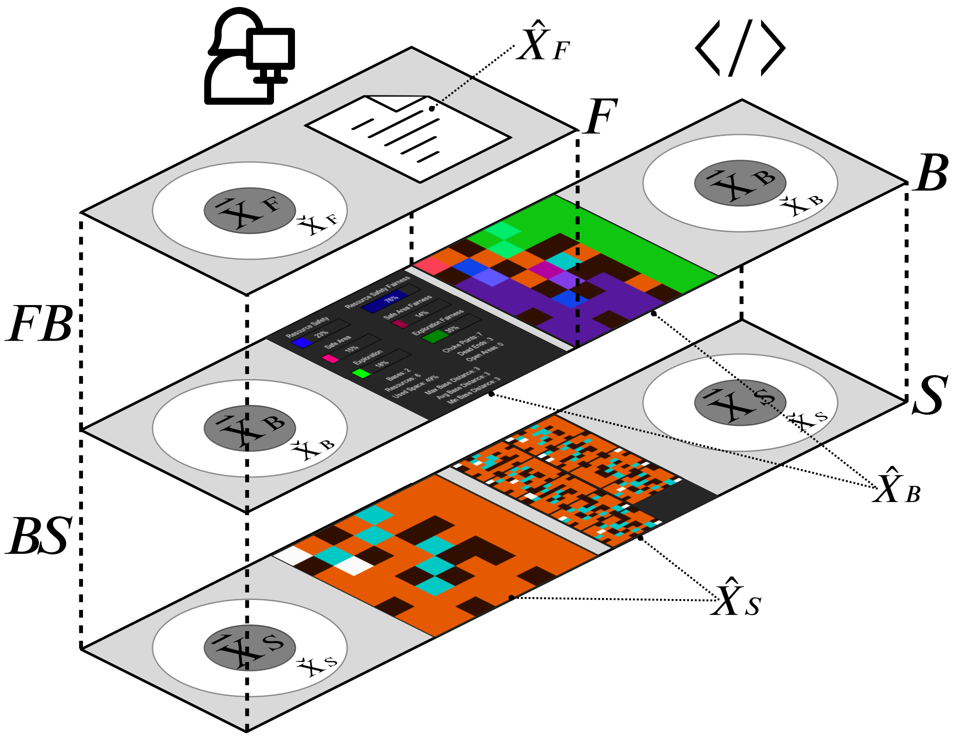


Figure 4.4: The design of game level maps through the Sentient Sketchbook tool. The complete system is modeled with a hierarchy of functional-sketch F, behavioral-sketch B, and structural-sketch S levels. The functional-behavioral layer FB (top-left) is a human-only layer where the human designer possibly externalize her/his design decisions into a piece of paper. The behavioral-structural layer BS (at the bottom) is a collaborative layer with the human designer (left) and the computational tool (right) as participants, sharing external behavioral and structural representations (center). At the behavioral level B there are the behavioral external representations $\hat{X}_B$ integrated by the interface area where the behavioral metrics are reported as well as the behavior-based views (only the segments view depicted) that are overlapped with the structural simple view of the map. At the structural level S there are the structural external representations $\hat{X}_S$ integrated by the simple map view, the set of suggestions generated by the tool, and the set of detailed views (not depicted) generated by the tool. (Sentient Sketchbook screenshots reproduced/adapted with permission from Antonios Liapis. "Computer user" by Llisole; "Software" by Tami Nova; and modified "Writing", original by ArtWorkLeaf; all from thenounproject.com, licensed under Creative Commons Attribution 🅭🅯[CC BY]).

4.1.1 Structural-sketch design level S

The tile types catalog explicitly described in the tool paper, represents the only one structural-sketch dimension **tile type**, formally defined as:

$$\dot{\delta}_{S_{tt}} = \langle \{\text{passable, impassable, bases, resources}\},\ D^c \rangle,$$

where D^c is a custom categorical distance.

4.1.2 Behavioral-sketch design level B.

In the case of behavioral dimensions, the tool authors explicitly describes how game pace and player balance dimensions are composed of other behavioral dimensions, leading us to describe them according to the compound dimensions pattern (Table 4.1). Also, the authors describe another set of dimensions or metrics, mostly pertaining to map navigation (Table 4.2).

Table 4.1: Sentient Sketchbook - Working set of behavioral-sketch design dimensions $\Delta_B{}'$ (1)

Dimension	Definition
Game pace	$\ddot{\delta}_{B_{pace}} = \langle \langle \dot{\delta}_{B_{res}},\ \dot{\delta}_{B_{saf}},\ \dot{\delta}_{B_{exp}} \rangle,\ L^2 \rangle$
Resource safety: The percentage of resources close to only one base.	$\dot{\delta}_{B_{res}} = \langle I,\ L^2 \rangle$
Safe area: The percentage of tiles around a base close only to that base.	$\dot{\delta}_{B_{saf}} = \langle I,\ L^2 \rangle$
Exploration: How difficult each base is to find starting from all enemy bases.	$\dot{\delta}_{B_{exp}} = \langle I,\ L^2 \rangle$
Player balance	$\ddot{\delta}_{B_{bal}} = \langle \langle \dot{\delta}_{B_{bres}}, \dot{\delta}_{B_{bsaf}}, \dot{\delta}_{B_{bexp}} \rangle, L^2 \rangle$
Resource safety balance: The degree to which all bases have equally safe resources.	$\dot{\delta}_{B_{bres}} = \langle I,\ L^2 \rangle$
Safe area balance: The degree to which all bases have a similar number of safe tiles.	$\dot{\delta}_{B_{bsaf}} = \langle I,\ L^2 \rangle$
Exploration balance: The degree to which all bases have equally difficult to find enemy bases.	$\dot{\delta}_{B_{bexp}} = \langle I,\ L^2 \rangle$

$L^2 :=$ Eucliedan distance

$I := [0,\ 1]$ (unit interval)

4.1.3 Functional-sketch design level F.

From the functional perspective, a game designed through this tool should provide playability and enable a particular gameplay. According to the tool authors, a particular gameplay can be defined

Table 4.2: Sentient Sketchbook - Working set of behavioral-sketch design dimensions Δ_{B}' (2)

Dimension	Definition
Metrics	$\ddot{\delta}_{B_m} = \langle\langle\dot{\delta}_{B_{bas}}, \ldots, \dot{\delta}_{B_{us}}\rangle, L^2\rangle$
# **Bases**	$\dot{\delta}_{B_{bas}} = \langle\mathbb{N}_{\mathbb{0}}, L^2\rangle$
# **Resources**	$\dot{\delta}_{B_{res}} = \langle\mathbb{N}_{\mathbb{0}}, L^2\rangle$
# **Chokepoints**: Tiles with two neighboring (passable) tiles.	$\dot{\delta}_{B_{chk}} = \langle\mathbb{N}_{\mathbb{0}}, L^2\rangle$
# **Dead ends**: Tiles with one neighboring tile.	$\dot{\delta}_{B_{de}} = \langle\mathbb{N}_{\mathbb{0}}, L^2\rangle$
# **Open areas**: Tiles with the maximum number of neighboring tiles.	$\dot{\delta}_{B_{oa}} = \langle\mathbb{N}_{\mathbb{0}}, L^2\rangle$
Maximum distance between bases	$\dot{\delta}_{B_{mbd}} = \langle\mathbb{Z}^+, L^2\rangle$
Average distance between bases	$\dot{\delta}_{B_{abd}} = \langle\mathbb{R}^+_{\geq\mathbb{1}}, L^2\rangle$
Minimum distance between bases	$\dot{\delta}_{B_{nbd}} = \langle\mathbb{Z}^+, L^2\rangle$
Used space: Percentage of passable tiles on the shortest path between any two bases or any base and any resource.	$\dot{\delta}_{B_{us}} = \langle I, L^2\rangle$

in terms of the game pace and player balance (two behavioral dimensions) and it is defined by the human designer on the fly. Regarding the playability, the authors provide explicit rules to distinguish between playable and not-playable maps (Table 4.3).

Table 4.3: Sentient Sketchbook - Working set of functional-sketch design dimensions Δ_{F}'

Dimension	Definition
Provide playability: If a map has less than two bases or unreachable bases, the map is not playable.	$\dddot{\delta}_{F_{play}} = \langle\langle\underbrace{\mathbb{N}_{\geq\mathbb{2}}}_{\dot{\delta}'_{B_{bas}}}, L^2\rangle, \langle\underbrace{[1, \infty)}_{\dot{\delta}'_{B_{abd}}}, L^2\rangle\rangle$
Enable x gameplay: Actual x gameplay delineated by the human designer during design session.	$\dddot{\delta}_{F_{gp}} = \langle\ddot{\delta}'_{B_{pace}}, \ddot{\delta}'_{B_{bal}}\rangle$

4.2 Stacked ECSs

The previously defined design levels give rise to two design layers or stacked ECSs : *functional-behavioral* FB, and *behavioral-structural* BS.

The Sentient Sketchbook tool is an example of a system not fitting a precise collaboration mode definition. The functional-behavioral layer is not collaborative (the tool does not even provide direct support for this layer), defined only for the human designer h. On the other hand, the behavioral-structural layer is a hybrid collaborative layer combining task-divided and co-creative modes between the human designer h and the computational tool c.

In terms of the proposed framework, the CCDS enabled by the Sentient Sketchbook tool can be defined as follows:

4.2.1 Functional-behavioral layer:

$$\left\langle \mathcal{L}, \left\langle \langle \Delta_{FB}, \mathcal{U}_{FB} \rangle, \langle \mathring{\Delta}_{FB_h}, \mathring{\mathcal{U}}_{FB_h} \rangle \right\rangle, [\![\cdot,\ \cdot]\!], \langle\!\langle \cdot,\ \cdot,\ \cdot,\ \cdot \rangle\!\rangle, \mathscr{R}_{FB}, \mathscr{E}_{FB} \cup \mathring{\mathscr{E}}_{FB_h}, \mathscr{P}_{FB_h}, \mathscr{T}_{FB_h} \right\rangle$$

4.2.2 Behavioral-structural layer:

$$\left\langle \mathcal{L},\ \left\langle \langle \Delta_{BS},\ \mathcal{U}_{BS} \rangle,\ \langle \mathring{\Delta}_{BS_h} \cup \mathring{\Delta}_{BS_c},\ \mathring{\mathcal{U}}_{BS_h} \cup \mathring{\mathcal{U}}_{BS_c} \rangle \right\rangle,\ [\![\cdot,\ \cdot]\!],\right.$$
$$\left.\langle\!\langle \cdot,\ \cdot,\ \cdot,\ \cdot \rangle\!\rangle,\ \mathscr{R}_{BS},\ \mathscr{E}_{BS} \cup \mathring{\mathscr{E}}_{BS_h} \cup \mathring{\mathscr{E}}_{BS_c},\ \mathscr{P}_{BS_h} \cup \mathscr{P}_{BS_c},\ \mathscr{T}_{BS_h} \cup \mathscr{T}_{BS_c} \right\rangle$$

The working set of functional design dimensions Δ_F' is integrated as a member of the common evaluative $\mathscr{E}_{FB}$ ruleset of the functional-behavioral layer:

$$\Delta_F' = \left\{ \dddot{\delta}_{F_{play}},\ \dddot{\delta}_{F_{gp}} \right\} \subset \Delta_F,\ \Delta_F' \in \mathscr{E}_{FB}.$$

Since the behavioral level $\boldsymbol{B}$ is shared by the two layers of the system, its working set of design dimensions Δ_B' is considered to be a member of two rulesets, the common definitional $\mathscr{R}_{FB}$ ruleset of the functional-behavioral layer and the common evaluative $\mathscr{E}_{BS}$ ruleset of the behavioral-structural layer:

$$\begin{aligned} \Delta_B' = \Big\{ & \dddot{\delta}_{B_{pace}},\ \dot{\delta}_{B_{res}},\ \dot{\delta}_{B_{saf}},\ \dot{\delta}_{B_{exp}},\ \dddot{\delta}_{B_{bal}},\ \dot{\delta}_{B_{bres}},\ \dot{\delta}_{B_{bsaf}},\ \dot{\delta}_{B_{bexp}}, \\ & \dddot{\delta}_{B_m},\ \dot{\delta}_{B_{bas}},\ \dot{\delta}_{B_{res}},\ \dot{\delta}_{B_{chk}},\ \dot{\delta}_{B_{de}},\ \dot{\delta}_{B_{oa}},\ \dot{\delta}_{B_{mbd}},\ \dot{\delta}_{B_{abd}},\ \dot{\delta}_{B_{nbd}},\ \dot{\delta}_{B_{us}} \Big\} \subset \Delta_B, \\ & \Delta_B' \in \mathscr{R}_{FB},\ \Delta_B' \in \mathscr{E}_{BS}. \end{aligned}$$

Finally, the working set of structural design dimensions Δ_S' is integrated into the common definitional $\mathscr{R}_{BS}$ ruleset of the behavioral-structural layer:

$$\Delta_S' = \left\{ \dot{\delta}_{S_{tt}} \right\} \subset \Delta_S,\ \Delta_S' \in \mathscr{R}_{BS}.$$

In the next subsections we break down each ruleset of each layer based on the design concepts and fundamental processes for designing (representing additional declarative and procedural knowledge, respectively) extracted from the Sentient Sketchbook paper.

4.3 Design concepts

In this section we identify a series of common and agent-specific key design concepts that represent requirements and constraints X_{L_r}, and intermediate design objects. These are described for each design level, and are mapped to their corresponding $\mathscr{R}$ and $\mathscr{E}$ rulesets.

4.3.1 Requirements and constraints

The explicit and implicit constraints already established by the authors of the system enable us to define a series of "initial" required design concepts $\hat{x}_{L_r}$ for each of the design levels. These concepts may also act as preliminary templates of their corresponding design spaces.

Initial structural-sketch required design concept
A member of the definitional $\mathscr{R}_{BS}$ ruleset of the behavioral-structural layer, this design concept establishes the structural template of a small sized (8×8) map and sets the structural-sketch solution space $\mathcal{S}_{BS} \subseteq \vec{X}_S$:

$$\hat{x}_{S_r} = \left\langle \mathrm{ran}(\dot{\delta}_{S_{tt}})_i \,\middle|\, \forall i \in \{1..64\} \right\rangle \in \hat{X}_S,\ \hat{x}_{S_r} \in \mathscr{R}_{BS}.$$

Initial behavioral-sketch required design concepts
Defines a behavioral empty concept as a precursor of both behavioral evaluation $\mathcal{E}_{BS} \subseteq \vec{X}_B$ and behavioral solution $\mathcal{S}_{FB} \subseteq \vec{X}_B$ spaces. As a concept of a shared design layer, it is considered a member of both definitional $\mathscr{R}_{FB}$ ruleset of the functional-behavioral layer and the evaluative $\mathscr{E}_{BS}$ ruleset of the behavioral-structural layer:

$$\hat{x}_{B_r} = \Big\langle \underset{\ddot{\delta}B_{metrics}}{\langle \mathbb{N}_0, \mathbb{N}_0, \mathbb{N}_0, \mathbb{N}_0, \mathbb{N}_0, \mathbb{Z}^+, \mathbb{R}^+_{\geqslant 1}, \mathbb{Z}^+, I \rangle}, \underset{\ddot{\delta}B_{pace}}{\langle I, I, I \rangle}, \underset{\ddot{\delta}B_{bal}}{\langle I, I, I \rangle} \Big\rangle \in \hat{X}_B,$$

$$\hat{x}_{B_r} \in \mathscr{R}_{FB},\ \hat{x}_{B_r} \in \mathscr{E}_{BS}.$$

Initial functional-sketch required design concepts
Captures the rules provided by the tool authors to distinguish between playable and not playable maps and sets the basis of the functional evaluation space $\mathcal{E}_{FB} \subseteq \vec{X}_F$, therefore it is integrated into the evaluative $\mathscr{E}_{FB}$ ruleset of the functional-behavioral layer:

$$\hat{x}_{F_r} = \Big\langle \underset{\ddot{\delta}F_{play}}{\big\langle \mathbb{N}_{\geqslant 2}, [1, \infty) \big\rangle}, \underset{\dddot{\delta}F_{gp}}{\langle \langle I, I, I \rangle, \langle I, I, I \rangle \rangle} \Big\rangle \in \hat{X}_F,\ \hat{x}_{F_r} \in \mathscr{E}_{FB}.$$

4.3.2 Intermediate produced design concepts

Evaluative rules $\mathring{\mathscr{R}}_{BS_h}$ of the human designer must include the design concept produced by she/he at the functional-behavioral layer. This design concept resides in the human expected world since its synthesis was performed “mentally”. Another possibility is that the human designer would had externalized the design concept, perhaps through notes or some drawing, in which case the design concept would be considered to reside in her/his external world:

$$\left\{\vec{x}_B = \langle\!\langle \mathscr{E}_{FB},\, \mathscr{E}_{FB},\, \mathscr{P}_{FB_h},\, \mathscr{T}_{FB_h} \rangle\!\rangle^{\diamond}(\varnothing)\right\} \subset \mathring{\mathscr{E}}_{BS_h}.$$

4.4 Fundamental processes for designing

In this section we describe the design process enabled by the Sentient Sketchbook through the identification of each of the fundamental processes for designing and their required perception/action strategies. Considering these strategies as a form of procedural knowledge, we also map them to their corresponding rulesets for each participant agent.

4.4.1 Functional-behavioral layer

Formulation: Ideally, the human designer h interprets the established constraints regarding playability and the rules about how the gameplay is modeled through game pace and player balance:

$$\left\{\hat{X}_F \underset{\rightarrow}{\sqsubset} \check{X}_F,\, \check{X}_F \underset{\dashrightarrow}{\sqsubset} \check{X}_F{}',\, \check{X}_F \rightleftarrows \vec{X}_F,\, \hat{X}_B \underset{\rightarrow}{\sqsubset} \check{X}_B,\, \check{X}_B \underset{\dashrightarrow}{\sqsubset} \check{X}_B{}',\, \check{X}_B \rightleftarrows \vec{X}_B\right\} \subset \mathscr{P}_{FB_h}.$$

Synthesis: The human designer probably decides how many bases should be on the map and, depending on her/his game level design experience and knowledge about the underlying dimensions, take reasoned decisions regarding game pace and player balance dimensions (set specific regions on dimensions’ ranges):

$$\left\{\vec{X}_F \Mapsto \vec{X}_B\right\} \subset \mathscr{T}_{FB_h}.$$

Analysis: Depending on her/his experience and knowledge, the designer may estimate (mentally) the playability and gameplay dimensions:

$$\left\{\vec{X}_B \rightleftarrows \check{X}_B\right\} \subset \mathscr{P}_{FB_h},\ \left\{\check{X}_B \mapsto \check{X}_F\right\} \subset \mathring{\mathscr{E}}_{FB_h}.$$

Evaluation: The designer compares her/his estimates regarding playability and gameplay against her/his expectations, in order to assess previously taken design decisions:

$$\left\{\vec{X}_F \leftrightarrow \check{X}_F\right\} \subset \mathscr{P}_{FB_h}.$$

Reformulation: The human designer may interpret the informed measures of quality of an interesting or unexpected behavior of a design suggestion generated by the tool or by herself/himself, leading the designer to modify her/his knowledge regarding how the gameplay is realized through the underlying behavioral dimensions (functional reformulation):

$$\left\{\hat{X}_B \underset{\rightarrow}{\sqsubset} \check{X}_B\right\} \subset \mathscr{P}_{FB_h},\ \left\{\check{X}_B \mapsto \check{X}_F\right\} \subset \mathscr{T}_{FB_h},\ \left\{\check{X}_F \underset{\dashrightarrow}{\sqsubset} \check{X}_F{}',\, \check{X}_F \rightleftarrows \vec{X}_F\right\} \subset \mathscr{P}_{FB_h}.$$

4.4.2 Behavioral-structural layer

Formulation: A design agent x (whether the human designer h or the tool c) interprets the established constraints regarding behavior and structure to set their design spaces (the tool interprets them from configuration files or from hardcoded rules). It is assumed that the human agent already contains the design object $\vec{x}_B$ obtained from the functional-behavioral layer, as part of her/his expected world. Regarding the tool, we make the assumption that it is not equipped with any special reflection and focusing strategies, therefore, we can consider both the reflection and focusing strategies as mere identity functions:

$$\left\{ \hat{X}_B \mathrel{\underrightarrow{\subset}} \check{X}_B,\ \check{X}_B \mathrel{\underset{\dashrightarrow}{\subset}} \check{X}_B{}',\ \check{X}_B \rightleftarrows \vec{X}_B \right\} \subset \mathscr{P}_{BS_x}.$$

The agents also interpret the empty map (composed of passable tiles only) that is displayed by default via the simple view *sim* when executing the interface for the first time:

$$\left\{ \hat{X}_S \mathrel{\underrightarrow{\subset}} \check{X}_S,\ \check{X}_S \mathrel{\underset{\dashrightarrow}{\subset}} \check{X}_S{}',\ \check{X}_S \rightleftarrows \vec{X}_S \right\} \subset \mathscr{P}_{BS_x},\ \hat{x}_{S_{sim}} \in \hat{X}_S.$$

Synthesis: The human designer creates or modifies the map by placing or erasing a tile or a set of tiles on it:

$$\left\{ \vec{x}_B \mathrel{|\!\!\Rightarrow} \vec{x}_S,\ \vec{x}_S \to \hat{x}_S \right\} \subset \mathscr{T}_{BS_h}.$$

She/he can also select a suggestion generated by the tool and apply it as the new design state:

$$\left\{ \vec{x}_{S_{GA}} \mathrel{\underrightarrow{\subset}} \check{x}_S,\ \check{x}_S \rightleftarrows \vec{x}_S \right\} \subset \mathscr{P}_{BS_h},\ \left\{ \vec{x}_S \to \hat{x}_S \right\} \subset \mathscr{T}_{BS_h}.$$

From the last human's map modification (last design state), the tool c generates and displays up to six suggestions via GA optimization on game pace and player balance dimensions, and up to six suggestions via novelty search GA:

$$\left\{ \vec{X}_B \mathrel{|\!\!\Rightarrow} \vec{X}_{S_{GAo}},\ \vec{X}_{S_{GAo}} \to \hat{X}_{S_{GAo}},\ \vec{X}_B \mathrel{|\!\!\Rightarrow} \vec{X}_{S_{GAn}},\ \vec{X}_{S_{GAn}} \to \hat{X}_{S_{GAn}} \right\} \subset \mathscr{T}_{BS_c}.$$

Analysis: The tool computes all behavioral dimensions from the current map and the suggestions generated:

$$\left\{ \hat{X}_S \mathrel{\underrightarrow{\subset}} \check{X}_S \right\} \subset \mathscr{P}_{BS_c},\ \left\{ \check{X}_S \mapsto \check{X}_B \right\} \subset \mathring{\mathscr{E}}_{BS_c},$$

and displays the measures of quality of the current map:

$$\left\{ \check{x}_B \rightleftarrows \vec{x}_B \right\} \subset \mathscr{P}_{BS_c},\ \left\{ \vec{x}_B \to \hat{x}_B \right\} \subset \mathscr{T}_{BS_c}.$$

Evaluation: The human designer evaluates the behavior of the current map or any generated suggestion by comparing their metrics to her/his expectations:

$$\left\{ \hat{X}_B \mathrel{\underrightarrow{\subset}} \check{X}_B,\ \check{X}_B \leftrightarrow \vec{X}_B \right\} \subset \mathscr{P}_{BS_h}.$$

She/he also may compare the behavior of any suggestion generated by the tool to the behavior of the current map:

$$\left\{ \hat{X}_{B_{GA}} \mathrel{\underrightarrow{\subset}} \check{X}_{B_{GA}},\ \hat{x}_{B_{sim}} \mathrel{\underrightarrow{\subset}} \check{x}_{B_{sim}},\ \check{x}_{B_{sim}} \rightleftarrows \vec{x}_{B_{sim}},\ \vec{x}_{B_{sim}} \leftrightarrow \check{X}_{B_{GA}} \right\} \subset \mathscr{P}_{BS_h}.$$

Documentation: The tool generates an alternative behavioral view of the map and overlays it with the simple *sim* view:

$$\Big\{\langle \vec{x}_B,\, \vec{x}_S\rangle \to \langle \hat{x}_{B_{nm}},\, \hat{x}_{S_{sim}}\rangle,\ \langle \vec{x}_B,\, \vec{x}_S\rangle \to \langle \hat{x}_{B_{rs}},\, \hat{x}_{S_{sim}}\rangle,\ \langle \vec{x}_B,\, \vec{x}_S\rangle \to \langle \hat{x}_{B_{sa}},\, \hat{x}_{S_{sim}}\rangle,$$
$$\langle \vec{x}_B,\, \vec{x}_S\rangle \to \langle \hat{x}_{B_{us}},\, \hat{x}_{S_{sim}}\rangle,\ \langle \vec{x}_B,\, \vec{x}_S\rangle \to \Big\langle \hat{x}_{B_{seg}},\, \hat{x}_{S_{sim}}\Big\rangle\Big\} \subset \mathscr{T}_{BS_c}.$$

The tool also displays behavioral metrics of a selected suggestion side by side with the current map metrics:

$$\Big\{\check{X}_{B_{GA}} \rightleftarrows \vec{X}_{B_{GA}}\Big\} \subset \mathscr{P}_{BS_c},\ \Big\{\vec{X}_{B_{GA}} \to \hat{X}_{B_{GA}}\Big\} \subset \mathscr{T}_{BS_c}.$$

On human's request the tool generates detailed versions of the map:

$$\Big\{\vec{x}_S \to \hat{x}_{S_{ddef}},\ \vec{x}_S \to \hat{x}_{S_{dhm}},\ \vec{x}_S \to \hat{x}_{S_{dww}},\ \vec{x}_S \to \hat{x}_{S_{ddun}}\Big\} \subset \mathscr{T}_{BS_c}.$$

Reformulation: No behavioral nor structural reformulation are performed at this layer since the interface does not provide mechanisms to add, remove, constraint, or loosen the design spaces' underlying dimensions.

4.5 Consolidated rulesets

Having identified and described the minimum components of declarative and procedural knowledge, we can consolidate them into their corresponding sets of rules:

4.5.1 Functional-behavioral layer:

$$\mathscr{R}_{FB} = \Big\{\Delta_B{}',\, \hat{x}_{B_{exp}}\Big\},\ \mathscr{E}_{FB} = \Big\{\Delta_F{}',\, \hat{x}_{F_{exp}}\Big\},\ \mathring{\mathscr{E}}_{FB_h} = \Big\{\check{X}_B \mapsto \check{X}_F\Big\},$$

$$\mathscr{T}_{FB_h} = \Big\{\vec{X}_F \Mapsto \vec{X}_B,\ \check{X}_B \mapsto \check{X}_F\Big\},$$

$$\mathscr{P}_{FB_h} = \Big\{\hat{X}_F \underset{\rightarrow}{\subset} \check{X}_F,\ \check{X}_F \underset{\dashrightarrow}{\subset} \check{X}_F{}',\ \check{X}_F \rightleftarrows \vec{X}_F,\ \vec{X}_F \leftrightarrow \check{X}_F,$$
$$\hat{X}_B \underset{\rightarrow}{\subset} \check{X}_B,\ \check{X}_B \underset{\dashrightarrow}{\subset} \check{X}_B{}',\ \check{X}_B \rightleftarrows \vec{X}_B,\ \vec{X}_B \leftrightarrow \check{X}_B\Big\}.$$

4.5.2 Behavioral-structural layer:

$$\mathscr{R}_{BS} = \left\{\Delta_{S}',\, \hat{x}_{S_r}\right\},\; \mathscr{E}_{BS} = \left\{\Delta_{B}',\, \hat{x}_{B_r}\right\},\; \mathring{\mathscr{E}}_{BS_c} = \left\{\check{X}_{S} \mapsto \check{X}_{B}\right\}$$

$$\mathscr{T}_{BS_h} = \left\{\vec{X}_{B} \Mapsto \vec{X}_{S},\, \vec{X}_{S} \rightarrow \hat{X}_{S}\right\},$$

$$\begin{aligned}\mathscr{P}_{BS_h} = \Big\{&\hat{X}_{B} \subsetneq \check{X}_{B},\, \check{X}_{B} \mathrel{\dot{\subsetneq}} \check{X}_{B}',\, \check{X}_{B} \rightleftarrows \vec{X}_{B},\, \check{X}_{B} \leftrightarrow \vec{X}_{B},\\ &\hat{X}_{S} \subsetneq \check{X}_{S},\, \check{X}_{S} \mathrel{\dot{\subsetneq}} \check{X}_{S}',\, \check{X}_{S} \rightleftarrows \vec{X}_{S},\, \check{X}_{S} \leftrightarrow \vec{X}_{S}\Big\},\end{aligned}$$

$$\begin{aligned}\mathscr{T}_{BS_c} = \Big\{&\vec{X}_{B} \Mapsto \vec{X}_{S_{GAo}},\, \vec{X}_{B} \Mapsto \vec{X}_{S_{GAn}},\, \vec{X}_{B} \rightarrow \hat{X}_{B},\, \langle \vec{x}_{B},\, \vec{x}_{S}\rangle \rightarrow \langle \hat{x}_{B_{nm}},\, \hat{x}_{S_{sim}}\rangle,\\ &\langle \vec{x}_{B},\, \vec{x}_{S}\rangle \rightarrow \langle \hat{x}_{B_{rs}},\, \hat{x}_{S_{sim}}\rangle,\, \langle \vec{x}_{B},\, \vec{x}_{S}\rangle \rightarrow \langle \hat{x}_{B_{sa}},\, \hat{x}_{S_{sim}}\rangle,\\ &\langle \vec{x}_{B},\, \vec{x}_{S}\rangle \rightarrow \langle \hat{x}_{B_{us}},\, \hat{x}_{S_{sim}}\rangle,\, \langle \vec{x}_{B},\, \vec{x}_{S}\rangle \rightarrow \left\langle \hat{x}_{B_{seg}},\, \hat{x}_{S_{sim}}\right\rangle,\\ &\vec{x}_{S} \rightarrow \hat{x}_{S_{ddef}},\, \vec{x}_{S} \rightarrow \hat{x}_{S_{dhm}},\, \vec{x}_{S} \rightarrow \hat{x}_{S_{dww}},\, \vec{x}_{S} \rightarrow \hat{x}_{S_{ddun}}\Big\},\end{aligned}$$

$$\mathscr{P}_{BS_c} = \left\{\hat{X}_{B} \subsetneq \check{X}_{B},\, \check{X}_{B} \mathrel{\dot{\subsetneq}} \check{X}_{B}',\, \check{X}_{B} \rightleftarrows \vec{X}_{B},\, \hat{X}_{S} \subsetneq \check{X}_{S},\, \check{X}_{S} \mathrel{\dot{\subsetneq}} \check{X}_{S}',\, \check{X}_{S} \rightleftarrows \vec{X}_{S}\right\}.$$

5 Building a collaborative CCDS: Building blocks

5.1 Design levels

Founded on the MDA-based views described in Sections 2.6.1 to 2.6.3, we establish three levels through which a designer could design a puzzle video game with our tool: game experience $\boldsymbol{P}$, game dynamics $\boldsymbol{D}$, and game mechanics $\boldsymbol{M}$. Two design layers emerge from these levels: the game experience–dynamics layer $\boldsymbol{PD}$, and the game dynamics–mechanics layer $\boldsymbol{DM}$.

Through this designing schema, a PuzzleScript-based video game is generated from expected game dynamics, which in turn, is derived from a desired player's playful experience, desired either by an external requestor (e.g., stakeholders) or designer's self-initiative. From the generated video game, actual game dynamics are formatively derived through and immediately after playtesting. Finally, the player's actual playful experience is derived summatively (Fig. 5.1).

5.1.1 Game experience $\boldsymbol{P}$ level

In our schema, the design concepts produced at this level describe, in terms of the PLEX playful experiences categories, both the expected to be evoked and the actual evoked game experience in the player when she/he/it plays the game. On the one hand, the designer (and perhaps some stakeholders) defines the desired game experience in terms of playful experiences categories. On the other hand, the player describes the game experience that was evoked after playtesting through a summative assessment in terms of the same playful experiences categories.

Each PLEX playful experience category is mapped into our schema as a design dimension to

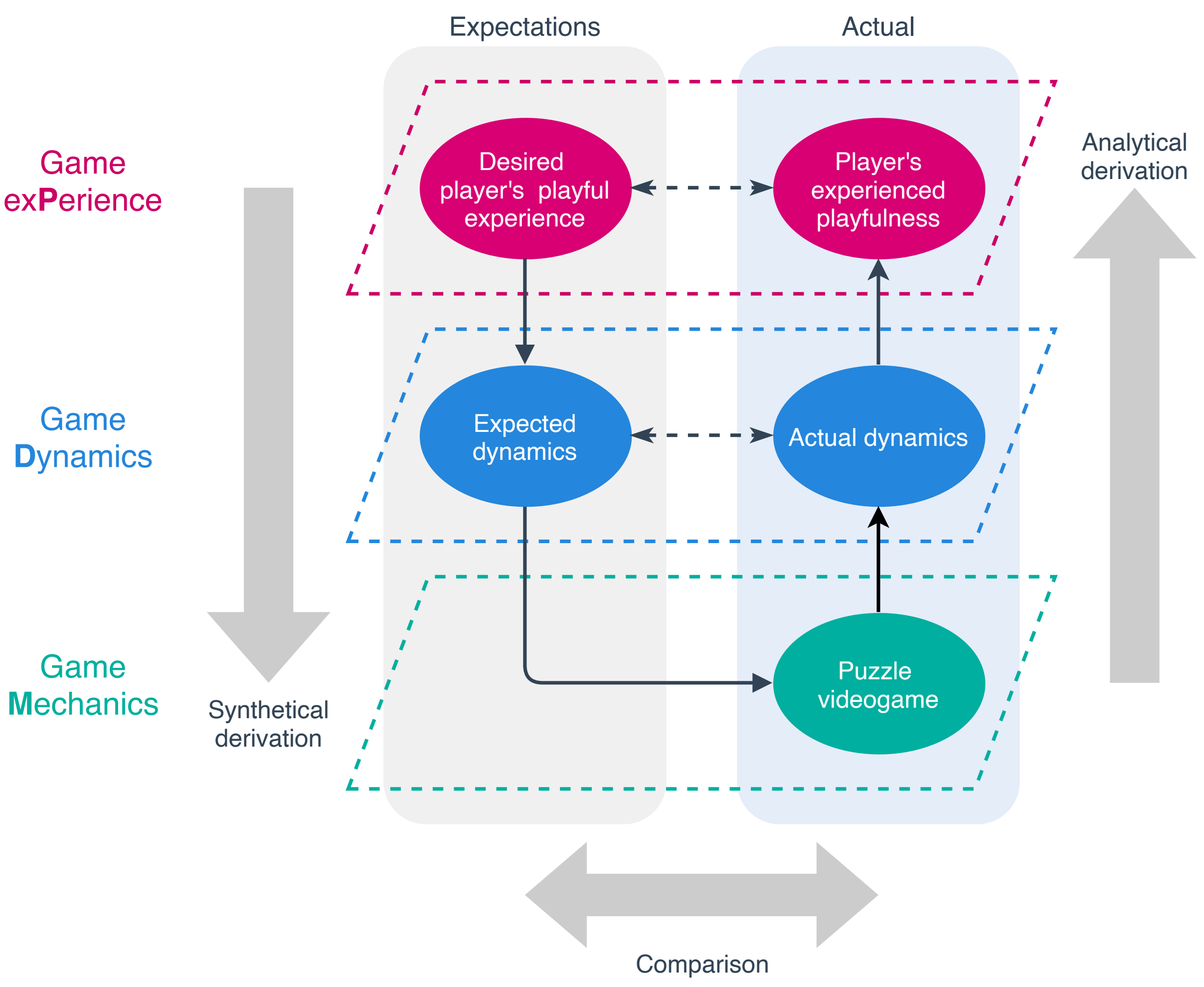


Figure 5.1: Puzzle video game primary designing schema. Depicts the external design concepts production flow.

model desired and actual game experiences, resulting in the formation of the following working set of game experience design dimensions:

$$\Delta'_{\mathcal{P}} := \left\{ \left(\dot{\delta}\mathcal{P}_X \right)_k \right\}_{k \in C_{plex}}, \; \Delta'_{\mathcal{P}} \subset \Delta_{\mathcal{P}},$$

where C_{plex} := {captivation, challenge, competition, … , thrill} (See Table 2.1) is the set of PLEX playful experiences categories.

A $\boldsymbol{k}$ **playful experience** design dimension $\left(\boldsymbol{\dot{\delta}\mathcal{P}_X}\right)_{\boldsymbol{k}}$ represents the intensity of the k PLEX playful experience:

$$\left(\dot{\delta}\mathcal{P}_X \right)_k := \langle I, \, D^c \rangle,$$

where $I := [0, \, 1]$ is the unitary interval,

$$D^c(X, \, Y) := \langle D^{c_o}(X, \, Y), \, J(X, \, Y) \rangle \tag{5.1}$$

is a custom distance composed of an overall categorical assessment $D^{c_o}(X, Y)$ defined as:

$$D^{c_o}(X, Y) := \begin{cases} \text{in-range} & : Y \subseteq X \\ \text{range partially exceeded} & : X \subset Y \land (\inf Y < \inf X \lor \sup Y > \sup X) \\ \text{range exceeded} & : X \subset Y \land \inf Y < \inf X \land \sup Y > \sup X \\ \text{out of range} & : X \cap Y = \varnothing \end{cases}, \tag{5.2}$$

and a set similarity metric (the Jaccard index) $J(X, Y)$ defined as:

$$J(X, Y) := \begin{cases} 1 & : X = Y = \varnothing \\ \dfrac{|X \cap Y|}{|X \cup Y|} & : X \neq \varnothing, Y \neq \varnothing \end{cases}, \tag{5.3}$$

where $0 \leq J(X, Y) \leq 1$.

Although zero intensity may not make sense when defining the desired experience, it allows the system to capture the situation where a player's desired-by-design experience is not evoked at all when playing a game.

Game exPerience
Playful experience [1]
...

Figure 5.2: Game experience design dimensions

The set of design dimensions previously described defines the $\mathcal{D}_P$ design space prototype, which also serves as the prototype of the evaluation space for the PD layer:

$$\mathcal{E}_{PD} = \mathcal{D}_P := \left\langle \left(\dot{\delta}_{P_X}\right)_k \right\rangle_{k \in K \subseteq C_{plex}}, \text{ where } \mathcal{D}_P \subseteq \vec{X}_P.$$

5.1.2 Game dynamics D level

According to what was discussed in Section 2.6.2, the expected and actual design concepts produced at this level could be synthesized and analyzed through intrinsic dynamics and playtest-based dynamics compound design dimensions. Intrinsic dynamics dimensions are those derived directly from video game mechanics. On the other hand, playtest-based dynamics dimensions represent run-time behavior that emerged (or are expected to emerge) from playtesting, whether human or automated. Playtest-based dynamics dimensions are broken down into three compound design dimensions: playability, player engagement, and gameplay scenario.

The following is the full working set of game dynamics design dimensions:

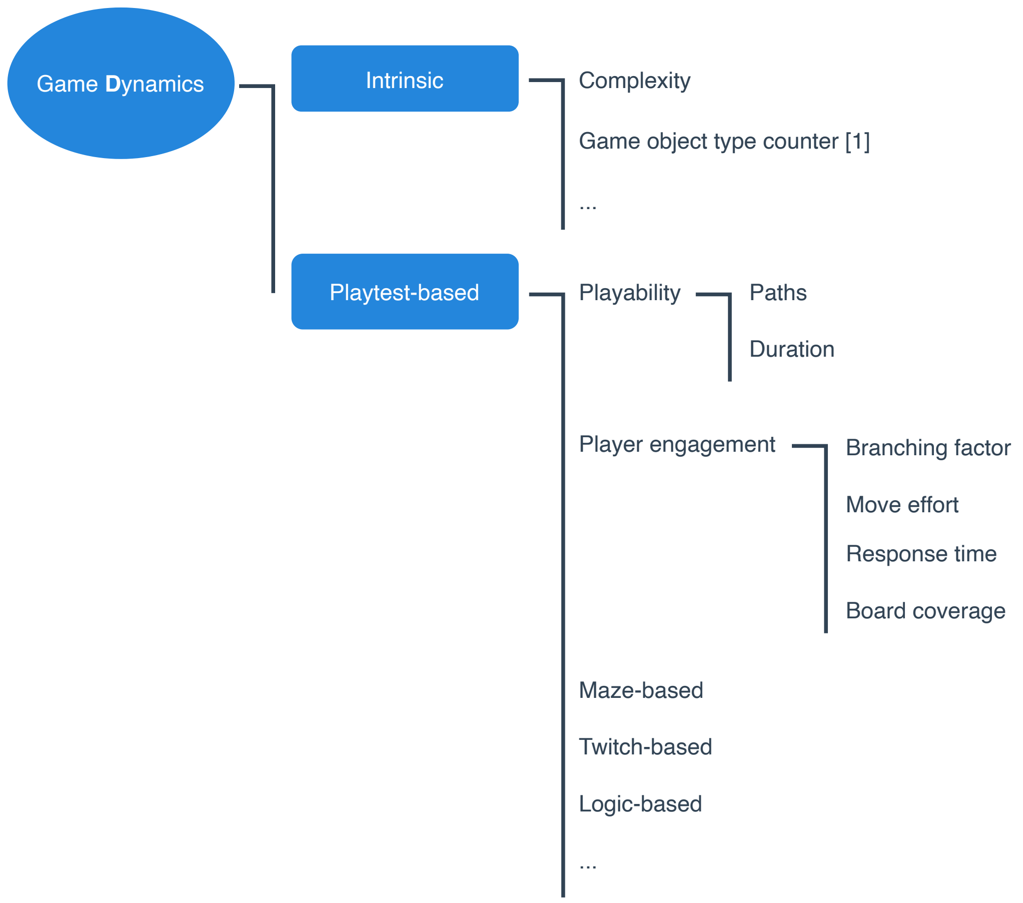


Figure 5.3: Game dynamics design dimensions

$$\Delta'_D := \left\{ \ddot{\delta}_{D_{id}},\ \ddot{\delta}_{D_{pd}} \right\} \cup \left\{ \dot{\delta}_{D_c} \right\} \cup \left\{ \left(\dot{\delta}_{D_{oc}}\right)_k \right\}_{k \in C_{ots}} \cup \left\{ \ddot{\delta}_{D_p},\ \ddot{\delta}_{D_{pe}},\ \ddot{\delta}_{D_{gs}} \right\}$$
$$\cup \left\{ \dot{\delta}_{D_{pt}},\ \dot{\delta}_{D_d} \right\} \cup \left\{ \dot{\delta}_{D_{bf}},\ \dot{\delta}_{D_{me}},\ \dot{\delta}_{D_{rt}},\ \dot{\delta}_{D_{bc}} \right\} \cup \left\{ \left(\dot{\delta}_{D_{gs}}\right)_k \right\}_{k \in C_{gs}}$$

Intrinsic dynamics $\ddot{\delta}_{D_{id}}$: A compound design dimension that represents the behavior of the game that emerged directly from the game mechanics

$$\ddot{\delta}_{D_{id}} := \left\langle \left\langle \dot{\delta}_{D_c},\ \left\langle \left(\dot{\delta}_{D_{oc}}\right)_k \right\rangle_{k \in K \subseteq C_{ots}} \right\rangle,\ D^{id} \right\rangle,$$

where D^{id} is a custom distance between games from the perspective of their intrinsic dynamics.

***Complexity* $\dot{\delta}_{D_c}$:** The number of underlying rules of the game [1] (i.e., the number of rules plus the number of unique game object types present in game rules)

$$\dot{\delta}_{D_c} := \langle \mathbb{W},\ D^c \rangle$$

[1]Design dimensions in italics are based on or borrowed from the Ludi system aesthetic measurement model (Browne 2011)

***k*th game object type counter $\left(\dot{\delta}_{D_{oc}}\right)_k$:** The number of k-type game objects in the game

$$\left(\dot{\delta}_{D_{oc}}\right)_k := \langle \mathbb{W},\ D^c \rangle,$$

where $\mathbb{W}$ is the set of whole numbers (i.e., natural numbers together with zero), $k \in C_{ots}$, and C_{ots} is the set of all types of game object available to the designer.

Playtest-based dynamics $\ddot{\delta}_{D_{pd}}$: A compound design dimension that represents the behavior of the game that emerged from the interaction of the player through game mechanics

$$\ddot{\delta}_{D_{pd}} := \left\langle \left\langle \ddot{\delta}_{D_P},\ \ddot{\delta}_{D_{pe}},\ \left\langle \left(\dot{\delta}_{D_{gs}}\right)_k \right\rangle_{k \in K \subseteq C_{gs}} \right\rangle,\ D^{pd} \right\rangle,$$

where D^{pd} is a custom distance between games from the perspective of their playtest-based dynamics.

Playability $\ddot{\delta}_{D_p}$: A compound design dimension that represents the degree to which a game is playable

$$\ddot{\delta}_{D_p} := \left\langle \left\langle \dot{\delta}_{D_{pt}},\ \dot{\delta}_{D_d} \right\rangle,\ D^p \right\rangle,$$

where D^p is a custom distance between playtest-based dynamics from the perspective of playability.

Paths $\dot{\delta}_{D_{pt}}$: The average number of possible solutions (trajectories towards a solution).

$$\dot{\delta}_{D_{pt}} := \langle \mathbb{R}_{\geq 0},\ D^c \rangle$$

***Duration $\dot{\delta}_{D_d}$*:** The average number of player's moves required to complete the game. According to Browne (2011), a game should be terminated within a reasonable number of moves (not too short, not too long). In our system a move is equivalent to a change of game state after a single movement command (up, down, left, right) or an action command from the player [2]

$$\dot{\delta}_{D_d} := \langle \mathbb{R}_{\geq 0},\ D^c \rangle$$

Player engagement $\ddot{\delta}_{D_{pe}}$: A compound design dimension that represents the degree to which the game may cause the player being engaged or disengaged to playing the game

$$\ddot{\delta}_{D_{pe}} := \left\langle \left\langle \dot{\delta}_{D_{bf}},\ \dot{\delta}_{D_{me}},\ \dot{\delta}_{D_{rt}},\ \dot{\delta}_{D_{bc}} \right\rangle,\ D^{pe} \right\rangle,$$

where D^{pe} is a custom distance between playtest-based dynamics from the perspective of player engagement.

***Branching factor $\dot{\delta}_{D_{bf}}$*:** The average number of move choices per game state. A logic-based puzzle should provide multiple choices, each containing multiple choices (Game Design Ed 2015b)

$$\dot{\delta}_{D_{bf}} := \langle \mathbb{R}_{\geq 0},\ D^c \rangle$$

[2]PuzzleScript supports a single action command (i.e., a single action key or button)

Move effort $\boldsymbol{\dot{\delta}_{D_{me}}}$**:** The average number of lookahead moves visited in the search for each move. A logic-based puzzle should make the player think a few steps ahead of their current situation (Game Design Ed 2015b)

$$\dot{\delta}_{D_{me}} := \langle \mathbb{R}_{\geq 0},\ D^c \rangle$$

Response time $\boldsymbol{\dot{\delta}_{D_{rt}}}$**:** The average amount of time required to formulate each move. A puzzle makes the player stop and think (Schell 2008)

$$\dot{\delta}_{D_{rt}} := \langle \mathbb{R}_{\geq 0},\ D^c \rangle$$

Board coverage $\boldsymbol{\dot{\delta}_{D_{bc}}}$**:** The ratio of the grid board cells visited within the total number of visitable cells. According to Browne (2011), low board coverage indicates that the game's starting position or movement rules may not be suitable for its board topology, or perhaps the board is simply too big

$$\dot{\delta}_{D_{bc}} := \langle I,\ D^c \rangle$$

$\boldsymbol{k}$**th gameplay scenario** $\boldsymbol{\left(\dot{\delta}_{D_{gs}}\right)_k}$**:** The degree to which the game represents a k gameplay pattern as explained in Section 2.6.2. This dimension type might be suitable for the designer to provide new dimensions (new gameplay scenarios) to characterize the gameplay

$$\left(\dot{\delta}_{D_{gs}}\right)_k := \langle I,\ D^c \rangle,$$

where $k \in C_{gs}$, and $C_{gs} := \{\text{maze-based, twitch-based, logic-based, } \dots\}$ is the set of all possible puzzle gameplay scenarios. Our schema provides three particular gameplay scenario design dimensions:

$$\left\{ \left(\dot{\delta}_{D_{gs}}\right)_{mz},\ \left(\dot{\delta}_{D_{gs}}\right)_{tw},\ \left(\dot{\delta}_{D_{gs}}\right)_{lg} \right\} \subset \left\{ \left(\dot{\delta}_{D_{gs}}\right)_k \right\}_{k \in C_{gs}}$$

Maze-based $\boldsymbol{\left(\dot{\delta}_{D_{gs}}\right)_{mz}}$**:** The degree to which the game could be considered a proper maze. The more the player is required to move through a labyrinth and find an exit to solve the puzzle, the more "maze" style the puzzle game would be. This dimension is also influenced by other dimensions such as paths $\dot{\delta}_{D_{pt}}$, and board coverage $\dot{\delta}_{D_{bc}}$

$$\left(\dot{\delta}_{D_{gs}}\right)_{mz} := \langle I,\ D^c \rangle$$

Twitch-based $\boldsymbol{\left(\dot{\delta}_{D_{gs}}\right)_{tw}}$**:** The degree to which the game challenges the player's reaction time and dexterity. Number of enemies $\dot{\delta}_{D_{goc_{enemy}}}$, and complexity $\dot{\delta}_{D_c}$ intrinsic dynamics dimensions may be helper dimensions to determine its value (e.g., the number, speed, and intelligence of pursuer objects influence the pace and difficulty of the game [Crawford 1984])

$$\left(\dot{\delta}_{D_{gs}}\right)_{tw} := \langle I,\ D^c \rangle$$

Logic-based $\boldsymbol{\left(\dot{\delta}_{D_{gs}}\right)_{lg}}$**:** The degree to which the game challenges the player's mind. Possible correlated dimensions: branching factor $\dot{\delta}_{D_{bf}}$, move effort $\dot{\delta}_{D_{me}}$, and response time $\dot{\delta}_{D_{rt}}$

$$\left(\dot{\delta}_{D_{gs}}\right)_{lg} := \langle I,\ D^c \rangle$$

The game dynamics design dimensions define the structure of the two-fold $\mathcal{D}_{D}$ design space, the solution space for the $\boldsymbol{PD}$ layer and the evaluation space for the $\boldsymbol{DM}$ layer:

$$\begin{aligned}\mathcal{D}_{D} := \Big\langle & \Big\langle \underbrace{\dot{\delta}_{D_{c}},\ \Big\langle \big(\dot{\delta}_{D_{ot}}\big)_{k_o} \Big\rangle_{k_o \in K_o \subseteq C_{ots}}}_{\ddot{\delta}D_{id}} \Big\rangle, \\ & \Big\langle \underbrace{\Big\langle \underbrace{\dot{\delta}_{D_{pt}},\ \dot{\delta}_{D_{d}}}_{\ddot{\delta}D_{P}} \Big\rangle,\ \Big\langle \underbrace{\dot{\delta}_{D_{bf}},\ \dot{\delta}_{D_{me}},\ \dot{\delta}_{D_{rt}},\ \dot{\delta}_{D_{bc}}}_{\ddot{\delta}D_{pe}} \Big\rangle,\ \Big\langle \big(\dot{\delta}_{D_{gs}}\big)_{k_s} \Big\rangle_{k_s \in K_s \subseteq C_{gs}}}_{\ddot{\delta}D_{pd}} \Big\rangle \Big\rangle,\end{aligned}$$

where

$$\mathcal{S}_{PD} = \mathcal{E}_{DM} = \mathcal{D}_{D}$$

and

$$\mathcal{D}_{D} \subseteq \vec{X}_{D}$$

5.1.3 Game mechanics $\boldsymbol{M}$ level

A design concept produced at this level represents a puzzle video game as a set of win conditions, game rules and a rectangular grid board (game space) composed of a collection of cells, each containing a background game object and a set of non-background game objects.

In a PuzzleScript-based video game, game objects are squared objects, each spanning a full grid board cell. Background-type game objects (also known as tiles) build the base layer (background layer) or floor of the game. On the other hand, non-background game objects (also known as tokens) are those positioned over the background layer.

Game objects in PuzzleScript are organized into layers starting with the background, mandatory layer. The game engine employs these layers not only to determine the order the game objects are rendered in the screen but to resolve movements and collisions (e.g., two game objects positioned in the same layer cannot coexist in the same cell, they collide).

We use the following working set of design dimensions Δ'_{M} to describe the design of the puzzle video game at the game mechanics level:

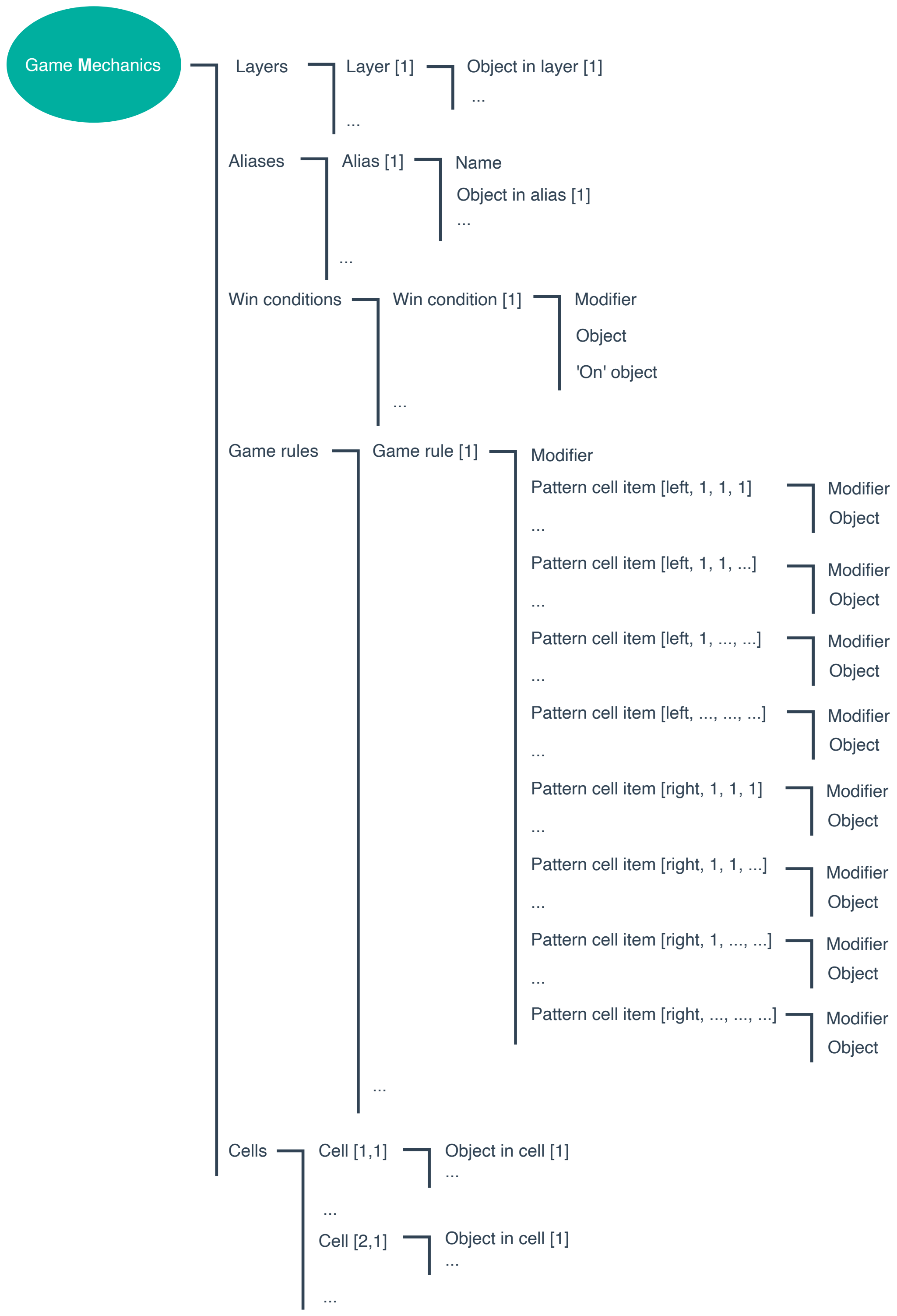


Figure 5.4: Game mechanics design dimensions

$$\Delta'_M := \left\{ \ddot{\delta}_{M_{ls}},\ \ddot{\delta}_{M_{as}},\ \ddot{\delta}_{M_{ws}},\ \ddot{\delta}_{M_{rs}},\ \ddot{\delta}_{M_{cs}} \right\} \cup \left\{ \left(\dot{\delta}_{M_l}\right)_i \right\}_{i \in \mathbb{N}} \cup \left\{ \left(\dot{\delta}_{M_a}\right)_h \right\}_{i \in C_{an}} \cup \left\{ \left(\ddot{\delta}_{M_w}\right)_i \right\}_{i \in \mathbb{N}} \cup \left\{ \left(\ddot{\delta}_{M_r}\right)_i \right\}_{i \in \mathbb{N}} \cup \left\{ \dot{\delta}_{M_{wm}},\ \dot{\delta}_{M_{wo}},\ \dot{\delta}_{M_{won}}, \right\} \cup \left\{ \dot{\delta}_{M_{rm}} \right\} \cup \left\{ \left(\ddot{\delta}_{M_{rc}}\right)_{s,u,v,w} \right\}_{\substack{s \in \{\text{left, right}\},\\ u \in \mathbb{N},\\ v \in \mathbb{N},\\ w \in \mathbb{N}}} \cup \left\{ \dot{\delta}_{M_{rcm}},\ \ddot{\delta}_{M_{rco}} \right\} \cup \left\{ \left(\dot{\delta}_{M_c}\right)_{i,j} \right\}_{\substack{i \in \mathbb{N},\\ j \in \mathbb{N}}}$$

Layers $\ddot{\delta}_{M_{ls}}$: A compound dimension that configures the layers of the game

$$\ddot{\delta}_{M_{ls}} := \left\langle \left(\dot{\delta}_{M_l}\right)_i \right\rangle_{i=1}^{n \in \mathbb{N}}$$

ith layer $\left(\dot{\delta}_{M_l}\right)_i$: A base dimension describing a layer as an aggregation of game object types

$$\left(\dot{\delta}_{M_l}\right)_i := \langle C_{ots},\ D^H \rangle,$$

where D^H is the Hamming distance between game object types.

Aliases $\ddot{\delta}_{M_{as}}$: A compound dimension that defines 'virtual' game object types or synonyms for a group of game object types

$$\dot{\delta}_{M_{as}} := \left\langle \left(\ddot{\delta}_{M_a}\right)_h \right\rangle_{h \in H \subseteq C_{an}},$$

where C_{an} is the set of possible alias names.

h alias $\left(\dot{\delta}_{M_a}\right)_h$: A base dimension that configures an alias as an aggregation of game object types

$$\left(\dot{\delta}_{M_a}\right)_h := \langle C_{ots},\ D^H \rangle.$$

Win conditions $\ddot{\delta}_{M_{ws}}$: A compound dimension defining a series of win conditions according to PuzzleScript's specifications[3]

$$\ddot{\delta}_{M_{ws}} := \left\langle \left(\ddot{\delta}_{M_w}\right)_i \right\rangle_{i=1}^{n \in \mathbb{N}}$$

ith win condition $\left(\ddot{\delta}_{M_w}\right)_i$: A compound dimension that represents a single win condition

$$\left(\ddot{\delta}_{M_w}\right)_i := \left\langle \dot{\delta}_{M_{wm}},\ \dot{\delta}_{M_{wo}},\ \dot{\delta}_{M_{won}} \right\rangle$$

Win condition modifier $\dot{\delta}_{M_{wm}}$: A quantifier that sets the required amount of game objects of a specific type to satisfy the win condition

$$\dot{\delta}_{M_{wm}} := \left\langle C_{wm},\ D^H \right\rangle, \text{ where } C_{wm} := \{\text{no, some, all}\}$$

[3]According to PuzzleScript specification, a game ends in victory condition if and only if all win conditions are satisfied.

Win condition game object $\dot{\delta}_{M_{wo}}$: The type of the game object (or the alias name of a set of object types) quantified by the win condition modifier

$$\dot{\delta}_{M_{wo}} \coloneqq \left\langle C_{ots} \cup C_{an},\ D^H \right\rangle$$

Win condition 'on' game object $\dot{\delta}_{M_{won}}$: An optional base dimension that specifies the type (or alias) of the game object that must share the same board cell as the quantified object (specified in the first part of the win condition) to satisfy the win condition

$$\dot{\delta}_{M_{won}} \coloneqq \left\langle C_{ots} \cup C_{an},\ D^H \right\rangle$$

In our framework, optionality can be modeled either by omitting the dimension from the design space/concept or setting the dimension to the empty set ∅, which is a valid subset of the dimension range.

Game rules $\ddot{\delta}_{M_{rs}}$: A compound dimension defining a series of game rules (actual rules governing player interaction with game objects and interaction between game objects) according to PuzzleScript's specifications

$$\ddot{\delta}_{M_{rs}} \coloneqq \left\langle \left(\ddot{\delta}_{M_r}\right)_i \right\rangle_{i \in I \subseteq \mathbb{N}}$$

ith game rule $\left(\ddot{\delta}_{M_r}\right)_i$: A compound dimension representing a single game rule conformed by two sides, left and right, each side composed of an equal number of p_i symmetric patterns, each pattern constituted by q_u symmetric rule cells composed of t cell items[4]

$$\left(\ddot{\delta}_{M_r}\right)_i \coloneqq \left\langle \dot{\delta}_{M_{rm}},\ \left\langle \left(\ddot{\delta}_{M_{rc}}\right)_{s,u,v,w} \right\rangle_{\substack{s \in \{\text{left}, \text{right}\},\\ u \in \{1..p_i \in \mathbb{N}\},\\ v \in \{1..q_u \in \mathbb{N}\},\\ w \in \{1..t \in \mathbb{N}\}}} \right\rangle$$

Game rule modifier $\dot{\delta}_{M_{rm}}$: An optional dimension that either sets the rule to be fired after consuming the player move (late rule) or defines all possible patterns (and their cells) directions to be considered by the engine when looking for matches (i.e., all directions when the modifier is set to the empty set, only horizontal directions, only vertical directions, or concrete up, down, left, or right directions)

$$\dot{\delta}_{M_{rm}} \coloneqq \left\langle C_{rm},\ D^H \right\rangle,$$

where $C_{rm} \coloneqq$ {late, horizontal, vertical, up, down, right, left}

wth cell item of vth cell of uth pattern of s side $\left(\ddot{\delta}_{M_{rc}}\right)_{s,u,v,w}$: A compound dimension representing a single game rule cell item

$$\left(\ddot{\delta}_{M_{rc}}\right)_{s,u,v,w} \coloneqq \left\langle \dot{\delta}_{M_{rcm}},\ \dot{\delta}_{M_{rco}} \right\rangle$$

[4]According to the PuzzleScript language definition, "in a rule, each pattern to match on the left must have a corresponding pattern on the right of equal length (number of cells)"; also, placing more than one cell items in a single cell is used to specify more than one layered game objects sharing the same board cell (Lavelle 2013).

Cell item modifier $\dot{\delta}_{M_{rcm}}$: An optional modifier that describes the behavior a game object is displaying when the engine performs pattern matching.

$$\dot{\delta}_{M_{rcm}} := \left\langle C_{rcm},\ D^H \right\rangle,$$

$$\text{where } C_{rcm} := \big\{\text{no, stationary, moving, orthogonal, parallel, perpendicular,} \\ \text{>, <, ^, v, up, down, right, left, action, random, randomDir}\big\}.$$

If the modifier is set to ‘no,’ the engine matches a game object absence; when set to ‘stationary,’ the engine matches a static object; if set to ‘moving,’ the engine matches either an action (the action button pressed) on an object or a movement command (up, down, right, left) applied to an object. If the modifier is set to ‘orthogonal,’ the engine matches an object moving (up. down, right, left). ‘>,’ ‘<,’ ‘^,’ and ‘v’ are relative directional modifiers that mean towards, away from, upwards, and downwards respectively. ‘Parallel’ and ‘perpendicular’ are relative group modifiers. ‘random’ and ‘randomDir’ are special modifiers used only on the right side of a game rule to produce the random appearance or random movement of an object, respectively. Finally, setting the modifier to the empty set tells the engine to ignore the specific object behavior; that is, the engine will only look for game object presence (or will leave it as is).

Cell item object type $\dot{\delta}_{M_{rco}}$: The game object type whose behavior is described by the cell item modifier

$$\dot{\delta}_{M_{rco}} := \left\langle C_{ots} \cup C_{an} \cup \{\text{“...”}\},\ D^H \right\rangle$$

Setting both the cell item modifier and cell item object type to the empty set is interpreted by the game engine as a board cell without any non-background game object. Setting the cell item modifier to the empty set and the cell item game object type to the special value “...” (ellipsis) makes the game engine to interpret variable length patterns.

Board cells $\ddot{\delta}_{M_{cs}}$: A compound dimension that represents the rectangular game board as a series of board cells

$$\ddot{\delta}_{M_{cs}} := \left\langle \left(\dot{\delta}_{M_c}\right)_{i,j} \right\rangle_{i=1,j=1}^{n,m\in\mathbb{N}}$$

***i*, *j*th board cell $\left(\dot{\delta}_{M_c}\right)_{i,j}$:** A base dimension representing a single board cell as an aggregation of game objects

$$\left(\dot{\delta}_{M_c}\right)_{i,j} := \left\langle C_{ots},\ D^H \right\rangle$$

The game mechanics design dimensions give rise to the structure of the $\mathcal{D}_M$ design space that corresponds to the solution space of the DM layer:

$$\mathcal{S}_{DM} = \mathcal{D}_{M} := \Bigg\langle \underbrace{\Big\langle \big(\dot{\delta}_{M_l}\big)_{i_l} \Big\rangle_{i_l=1}^{n_l \in \mathbb{N}}}_{\ddot{\delta}M_{ls}},\ \underbrace{\Big\langle \big(\dot{\delta}_{M_a}\big)_h \Big\rangle_{h \in H \subseteq C_{an}}}_{\ddot{\delta}M_{as}},\ \underbrace{\Big\langle \underbrace{\big\langle \dot{\delta}_{M_{wm}},\ \dot{\delta}_{M_{wo}},\ \dot{\delta}_{M_{won}} \big\rangle_{i_w}}_{(\ddot{\delta}M_w)_{i_w}} \Big\rangle_{i_w=1}^{n_w \in \mathbb{N}}}_{\ddot{\delta}M_{ws}},$$

$$\underbrace{\Bigg\langle \underbrace{\Bigg\langle \Bigg\langle \underbrace{\big\langle \dot{\delta}_{M_{rcm}},\ \dot{\delta}_{M_{rco}} \big\rangle_{s_i, u_i, v_i, w_i}}_{(\ddot{\delta}M_{rc})_{s_i, u_i, v_i, w_i}} \Bigg\rangle_{\substack{s_i \in \{\text{left, right}\},\\ u_i \in \{1..p_i \in \mathbb{N}\},\\ v_i \in \{1..q_{u_i} \in \mathbb{N}\},\\ w_i \in \{1..t \in \mathbb{N}\}}} \Bigg\rangle_i}_{(\ddot{\delta}M_r)_i} \Bigg\rangle_{i=1}^{n \in \mathbb{N}}}_{\ddot{\delta}M_{rs}},$$

$$\underbrace{\Big\langle \big(\dot{\delta}_{M_c}\big)_{i_c, j_c} \Big\rangle_{i_c=1,\, j_c=1}^{n_c, m_c \in \mathbb{N}}}_{\ddot{\delta}M_{cs}} \Bigg\rangle.$$

5.2 Design concepts

As described in our framework, any design concept is defined as a series of constrained design dimensions' ranges (i.e., subsets of original ranges)[5]:

$$x_P := \Big\langle \underbrace{{}_{\subseteq}I}_{(\dot{\delta}P_x)_{k_x}} \Big\rangle_{k_x \in K_x \subseteq C_{plex}},$$

$$x_D := \Bigg\langle \underbrace{\Big\langle \underbrace{{}_{\subseteq}\mathbb{W}}_{\dot{\delta}D_c},\ \Big\langle \underbrace{{}_{\subseteq}\mathbb{W}}_{(\dot{\delta}D_{ot})_{k_o}} \Big\rangle_{k_o \in K_o \subseteq C_{ots}} \Big\rangle}_{\ddot{\delta}D_{id}},\ \underbrace{\Big\langle \underbrace{\Big\langle \underbrace{{}_{\subseteq}\mathbb{R}_{\geq 0}}_{\dot{\delta}D_{pt}},\ \underbrace{{}_{\subseteq}\mathbb{R}_{\geq 0}}_{\dot{\delta}D_{d}} \Big\rangle}_{\ddot{\delta}D_p},\ \underbrace{\Big\langle \underbrace{{}_{\subseteq}\mathbb{R}_{\geq 0}}_{\dot{\delta}D_{bf}},\ \underbrace{{}_{\subseteq}\mathbb{R}_{\geq 0}}_{\dot{\delta}D_{me}},\ \underbrace{{}_{\subseteq}\mathbb{R}_{\geq 0}}_{\dot{\delta}D_{rt}},\ \underbrace{{}_{\subseteq}I}_{\dot{\delta}D_{bc}} \Big\rangle}_{\ddot{\delta}D_{pe}},\ \Big\langle \underbrace{{}_{\subseteq}I}_{\left(\dot{\delta}D_{gs}\right)_{k_s}} \Big\rangle_{k_s \in K_s \subseteq C_{gs}} \Big\rangle}_{\ddot{\delta}D_{pd}} \Bigg\rangle,$$

[5]We employ ${}_{\subseteq}X$ as a contracted form of the expression $x \subseteq X,\ x \neq \emptyset$ to generically denote 'a subset of X'

$$x_M := \Bigg\langle \underbrace{\Bigg\langle \underbrace{\left({}_{\subseteq}C_{ots}\right)_{i_l}}_{(\dot{\delta}M_l)_{i_l}} \Bigg\rangle_{i_l=1}^{n_l \in \mathbb{N}}}_{\ddot{\delta}M_{ls}}, \underbrace{\Bigg\langle \underbrace{\left({}_{\subseteq}C_{ots}\right)_h}_{(\dot{\delta}M_a)_h} \Bigg\rangle_{h \in H \subseteq C_{an}}}_{\ddot{\delta}M_{as}},$$

$$\underbrace{\Bigg\langle \underbrace{\Bigg\langle \underbrace{{}_{\subseteq}C_{wm}}_{\dot{\delta}M_{wm}}, \underbrace{{}_{\subseteq}\left(C_{ots} \cup C_{an}\right)}_{\dot{\delta}M_{wo}}, \underbrace{{}_{\subseteq}\left(C_{ots} \cup C_{an}\right)}_{\dot{\delta}M_{won}} \Bigg\rangle}_{(\ddot{\delta}M_w)_{i_w}}_{i_w \in I_w \subseteq \mathbb{N}} \Bigg\rangle}_{\ddot{\delta}M_{ws}},$$

$$\underbrace{\Bigg\langle \underbrace{\Bigg\langle \underbrace{{}_{\subseteq}C_{rm}}_{\dot{\delta}M_{rm}}, \underbrace{\Bigg\langle \underbrace{{}_{\subseteq}C_{wm}}_{\dot{\delta}M_{rcm}}, \underbrace{{}_{\subseteq}\left(C_{ots} \cup C_{an} \cup \{\text{“...”}\}\right)}_{\dot{\delta}M_{rco}} \Bigg\rangle}_{(\dot{\delta}M_{rc})_{s_i, u_i, v_i, w_i}}{}_{\substack{s_i \in \{\text{left, right}\}, \\ u_i \in \{1..p_i \in \mathbb{N}\}, \\ v_i \in \{1..q_{u_i} \in \mathbb{N}\}, \\ w_i \in \{1..t \in \mathbb{N}\}}} \Bigg\rangle}_{(\ddot{\delta}M_r)_i}{}_{i \in I_r \subseteq \mathbb{N}} \Bigg\rangle}_{\ddot{\delta}M_{rs}},$$

$$\underbrace{\Bigg\langle \underbrace{{}_{\subseteq}C_{ots}}_{(\dot{\delta}M_c)_{i_c, j_c}} \Bigg\rangle_{i_c=1, j_c=1}^{n_c, m_c \in \mathbb{N}}}_{\ddot{\delta}M_{cs}} \Bigg\rangle.$$

By employing the design dimensions defined in previous sections, it is possible to describe all design concepts relevant to designing a puzzle video game, both the concepts representing requirements and constraints, and the produced design concepts that should satisfy those requirements.

Example 5.2.1. The following is an example of externally defined requirements and constraints and how can they be modeled through our framework:

a) In a game experience design space spanned by 'challenge' and 'exploration' design dimensions,

b) the game experience evoked in the player has to be primarily 'challenge' (at least 75% intensity), combined with a precise balance of 'exploration' (50%).

c) The game should be a pure maze-based puzzle (i.e., 'maze-based' dimension should be greater than 75%),

d) having 1 player token, 1 finish token and some wall tokens (at least 4) used to constrain player movements within the board.

e) The game must contain a win condition $w \in C_{wc}$ that ends a game with a victory when the player token collects the finish token.

f) The game must be solvable, that is, there must be at least one *path* from the player starting position to the finish tile position (i.e., the range of the *paths* $\dot{\delta}_{D_{pt}}$ design dimension has to be constrained from the set of non-negative real numbers $\mathbb{R}_{\geq 0}$ into the set of positive reals greater or equal to one $\mathbb{R}_{\geq 1}$).

g) Finally, the grid board must be a 8×8 square.

These requirements and constraints correspond to the early expectations for the designing activity and can be modeled as design concepts as follows:

$$\hat{x}_{P_r} := \underbrace{\left\langle \underbrace{[.75, 1]}_{\text{b)}}, \underbrace{\{.5\}}_{\text{b)}} \right\rangle}_{\text{a)}},$$

$$\hat{x}_{D_r} := \left\langle \left\langle \mathbb{W}, \left\langle \underbrace{\{1\}}_{\text{d) player token}}, \underbrace{\{1\}}_{\text{d) finish token}}, \underbrace{\{4..\}}_{\text{d) wall tokens}} \right\rangle \right\rangle, \left\langle \left\langle \underbrace{\mathbb{R}_{\geq 1}}_{\text{f)}}, \mathbb{R}_{\geq 0} \right\rangle, \left\langle \mathbb{R}_{\geq 0}, \mathbb{R}_{\geq 0}, \mathbb{R}_{\geq 0}, I \right\rangle, \left\langle \underbrace{(.75, 1]}_{\text{c)}}, I, I \right\rangle \right\rangle \right\rangle,$$

$$\hat{x}_{M_r} := \Big\langle \langle \{\text{background}, \ldots\} \subset C_{ots}, \{\text{finish}, \ldots\} \subset C_{ots}, \{\text{player}, \text{wall}, \ldots\} \subset C_{ots} \rangle,$$

$$\left\langle \left({\subseteq} C_{ots} \right)_h \right\rangle_{h \in H \subseteq C_{an}}, \left\langle \left\langle \underbrace{\{\text{all}\}, \{\text{player}\}, \{\text{finish}\}}_{\text{e)}} \right\rangle \right\rangle,$$

$$\left\langle \left\langle {\subseteq} C_{rm}, \left\langle {\subseteq} C_{wm}, \subseteq \left(C_{ots} \cup C_{an} \cup \{\text{“…”}\} \right) \right\rangle_{\substack{s_{i_r} \in \{\text{left, right}\}, \\ u_{i_r} \in \{1..p \in \mathbb{N}\}, \\ v_{i_r} \in \{1..q_u \in \mathbb{N}\}, \\ w_{i_r} \in \{1..t_v \in \mathbb{N}\}}} \right\rangle_{i_r \in I_r \subseteq \mathbb{N}} \right\rangle,$$

$$\underbrace{\left\langle {\subseteq} C_{ots} \right\rangle_{\substack{i_c \in \{1..8\}, \\ j_c \in \{1..8\}}}}_{\text{g)}} \Big\rangle,$$

$$C_{ots} := \{\text{background, player, finish, wall, } \ldots\},$$
$$\hat{x}_{P_r} \in \hat{X}_P, \; \hat{x}_{D_r} \in \hat{X}_D, \; \hat{x}_{M_r} \in \hat{X}_M,$$
$$\hat{x}_{P_r} \in \mathscr{E}_{PD}, \; \hat{x}_{D_r} \in \mathscr{R}_{PD}, \; \hat{x}_{D_r} \in \mathscr{E}_{DM}, \; \hat{x}_{M_r} \in \mathscr{R}_{DM}.$$

5.3 System evolution

The possibilities for building a system within the creative responsibility continuum are innumerable. Then we aim to build our collaborative system incrementally, starting with a computational agent with minimum responsibility, providing it later with greater responsibilities and capabilities, even with the possibility of replacing and improving strategies as part of its evolution.

We plan to develop three versions of the computational agent that will collaborate to some degree with a human designer. These versions are not exhaustive but just a minimal set of representatives within the creative responsibility continuum (Table 5.1).

Table 5.1: System roadmap. ① Assistant, ② Apprentice assistant, and ③ "Creative" assistant. This table shows the role of a particular version of the computational agent in the collaborative system for each design layer and fundamental process for designing.

Process	**Observer**	**Helper**	**Suggester**	**Co-responsible**	**Responsible**
			Game experience-dynamics layer *PD*		
Formulation	✗	✗	✗	✗	✗
Synthesis	② Supervised learning	✗	③ Game dynamics ideas	✗	✗
Analysis	② Supervised learning	✗	③ Common artifact & own ideas	✗	✗
Evaluation	✗	✗	✗	✗	① Task dividing mode
Documentation	✗	✗	✗	✗	✗
Reformulation	✗	✗	✗	✗	✗
			Game dynamics–mechanics layer *DM*		
Formulation	✗	✗	✗	✗	✗
Synthesis	② Supervised learning	✗	③ Game ideas	✗	✗

Table continued on next page

Table 5.1 – continued from previous page

Process	**Observer**	**Helper**	**Suggester**	**Co-responsible**	**Responsible**
Analysis	② Supervised learning	① Playable game generation	③ Common artifact & own ideas	① Automatic playtesting	✗
Evaluation	✗	✗	✗	✗	① Task dividing mode
Documentation	✗	✗	✗	✗	✗
Reformulation	✗	✗	✗	✗	✗

In the following chapters (Chapters 6 to 8), we develop these versions of the collaborative system. We formalize them and describe them according to the framework components formulated in Chapter 3 and the design levels defined in Section 5.1. We also describe the computational mechanisms and artificial intelligence techniques used to implement the most relevant perception and action strategies and the design and evaluation aspects of the GUI illustrated with mockups and screenshots of the actual interface.

6 Building a collaborative CCDS: Assistant computational agent

In this version of the system, a computational agent c is co-responsible for the game mechanics analysis process, which implies playing (automatic playtesting) several times, using four search algorithms, a human-designed video game. Agent c is also fully responsible for the evaluation processes in both design layers; that is, it determines and assesses the similarity between expected and actual game experience and between expected and actual game dynamics. Finally, this version of the c agent assists the game mechanics analysis process conducted by the human player (h agent) through the production of the human-playable video game from the computer-playable game.

We then formally define this system version[1]:

Game experience-dynamics layer PD:

$$\Big\langle \mathcal{L},\ \Big\langle \langle \Delta_{PD},\ \mathcal{U}_{PD} \rangle,\ \langle \mathring{\Delta}_{PD_h} \cup \mathring{\Delta}_{PD_c},\ \mathring{\mathcal{U}}_{PD_h} \cup \mathring{\mathcal{U}}_{PD_c} \rangle \Big\rangle,\ [\![\cdot,\ \cdot]\!],\ \langle\!\langle \cdot,\ \cdot,\ \cdot,\ \cdot \rangle\!\rangle,$$

$$\mathscr{R}_{PD} \cup \mathring{\mathscr{R}}_{PD_h} \cup \mathring{\mathscr{R}}_{PD_c},\ \mathscr{E}_{PD} \cup \mathring{\mathscr{E}}_{PD_h} \cup \mathring{\mathscr{E}}_{PD_c},\ \mathscr{P}_{PD_h} \cup \mathscr{P}_{PD_c},\ \mathscr{T}_{PD_h} \cup \mathscr{T}_{PD_c} \Big\rangle,$$

$$\mathscr{R}_{PD} = \left\{ \Delta'_D,\ \hat{X}_{Dr} \right\},\ \mathscr{E}_{PD} = \left\{ \Delta'_P,\ \hat{X}_{Pr} \right\},$$

[1]In terms of framework coverage, this system version's GUI supports all of the processes for designing except for the documentation process.

where

- $\left\langle \langle \Delta_{PD}, \mathcal{U}_{PD} \rangle, \langle \mathring{\Delta}_{PD_h} \cup \mathring{\Delta}_{PD_c}, \mathring{\mathcal{U}}_{PD_h} \cup \mathring{\mathcal{U}}_{PD_c} \rangle \right\rangle$ is the ***PD*** layer's universe represented by both the *whole domain* universe $\langle \Delta_{PD}, \mathcal{U}_{PD} \rangle$, and the designers' universes $\langle \mathring{\Delta}_{PD_h} \cup \mathring{\Delta}_{PD_c}, \mathring{\mathcal{U}}_{PD_h} \cup \mathring{\mathcal{U}}_{PD_c} \rangle$;
- $\mathcal{R}_{PD} \cup \mathring{\mathcal{R}}_{PD_h} \cup \mathring{\mathcal{R}}_{PD_c}$ is the ***PD*** layer's definitional ruleset composed of both the common and shared rules $\mathcal{R}_{PD}$ consisting of the working set of game dynamics design dimensions Δ'_D and external game dynamics requirements $\hat{X}_{Dr}$, and the designers' individual rules $\mathring{\mathcal{R}}_{PD_h}$ and $\mathring{\mathcal{R}}_{PD_c}$;
- $\mathcal{E}_{PD} \cup \mathring{\mathcal{E}}_{PD_h} \cup \mathring{\mathcal{E}}_{PD_c}$ is the ***PD*** layer's evaluative ruleset composed of both the common and shared rules $\mathcal{E}_{PD}$ consisting of the working set of game experience design dimensions Δ'_P and game experience requirements $\hat{X}_{Pr}$, and the designers' individual rules $\mathring{\mathcal{R}}_{PD_h}$ and $\mathring{\mathcal{R}}_{PD_c}$;
- $\mathcal{P}_{PD_h} \cup \mathcal{P}_{PD_c}$ is the ***PD*** layer's perception ruleset composed of both the human $\mathcal{P}_{PD_h}$, and the computational $\mathcal{P}_{PD_c}$ agent rules;
- $\mathcal{T}_{PD_h} \cup \mathcal{T}_{PD_c}$ is the ***PD*** layer's action ruleset composed of both the human $\mathcal{T}_{PD_h}$, and the computational $\mathcal{T}_{PD_c}$ agent rules.

Game dynamics-mechanics layer *DM*:

$$\Bigg\langle \mathcal{L}, \Bigg\langle \langle \Delta_{DM} \cup \Delta_I, \mathcal{U}_{DM} \cup \mathcal{U}_I \rangle, \langle \mathring{\Delta}_{DM_h} \cup \mathring{\Delta}_{I_h} \cup \mathring{\Delta}_{DM_c} \cup \mathring{\Delta}_{I_c}, \mathring{\mathcal{U}}_{DM_h} \cup \mathring{\mathcal{U}}_{I_h} \cup \mathring{\mathcal{U}}_{DM_c} \cup \mathring{\mathcal{U}}_{I_c} \rangle \Bigg\rangle,$$
$$[\![\cdot, \cdot]\!], \langle\!\langle \cdot, \cdot, \cdot, \cdot \rangle\!\rangle,$$
$$\mathcal{R}_{DM} \cup \mathring{\mathcal{R}}_{DM_h} \cup \mathring{\mathcal{R}}_{DM_c}, \mathcal{E}_{DM} \cup \mathring{\mathcal{E}}_{DM_h} \cup \mathring{\mathcal{E}}_{DM_c}, \mathcal{P}_{DM_h} \cup \mathcal{P}_{DM_c}, \mathcal{T}_{DM_h} \cup \mathcal{T}_{DM_c} \Bigg\rangle,$$

$$\mathcal{R}_{DM} = \left\{ \Delta'_M, \hat{X}_{Mr} \right\}, \mathcal{E}_{DM} = \left\{ \Delta'_D, \hat{X}_{Dr}, \hat{x}_{De_h} \right\},$$

where

- $\langle \langle \Delta_{DM} \cup \Delta_I, \mathcal{U}_{DM} \cup \mathcal{U}_I \rangle, \langle \mathring{\Delta}_{DM_h} \cup \mathring{\Delta}_{I_h} \cup \mathring{\Delta}_{DM_c} \cup \mathring{\Delta}_{I_c}, \mathring{\mathcal{U}}_{DM_h} \cup \mathring{\mathcal{U}}_{I_h} \cup \mathring{\mathcal{U}}_{DM_c} \cup \mathring{\mathcal{U}}_{I_c} \rangle \rangle$ is the ***DM*** layer's universe represented by both the *whole domain* universe $\langle \Delta_{DM} \cup \Delta_I, \mathcal{U}_{DM} \cup \mathcal{U}_I \rangle$, and the designers' ones $\langle \mathring{\Delta}_{DM_h} \cup \mathring{\Delta}_{I_h} \cup \mathring{\Delta}_{DM_c} \cup \mathring{\Delta}_{I_c}, \mathring{\mathcal{U}}_{DM_h} \cup \mathring{\mathcal{U}}_{I_h} \cup \mathring{\mathcal{U}}_{DM_c} \cup \mathring{\mathcal{U}}_{I_c} \rangle$;
- ***I*** is an *interaction* category (non-design category) of dimensions;
- $\mathcal{R}_{DM} \cup \mathring{\mathcal{R}}_{DM_h} \cup \mathring{\mathcal{R}}_{DM_c}$ is the ***DM*** layer's definitional ruleset composed of both the common and shared rules $\mathcal{R}_{DM}$ consisting of the working set of game mechanics design dimensions Δ'_M and external game mechanics requirements $\hat{X}_{Mr}$, and the designers' individual rules $\mathring{\mathcal{R}}_{DM_h}$ and $\mathring{\mathcal{R}}_{DM_c}$;
- $\mathcal{E}_{DM} \cup \mathring{\mathcal{E}}_{DM_h} \cup \mathring{\mathcal{E}}_{DM_c}$ is the ***DM*** layer's evaluative ruleset composed of both the designers' individual rules $\mathring{\mathcal{E}}_{DM_h}$ and $\mathring{\mathcal{E}}_{DM_c}$, and the common rules $\mathcal{E}_{DM}$ consisting of the working set of game dynamics design dimensions Δ'_D, game dynamics requirements $\hat{X}_{Dr}$, and the external representation of the (expected) game dynamics design concept $\hat{x}_{De_h}$ produced at the upper ***PD*** layer by the human agent ***h***;

- $\mathscr{P}_{DM_h} \cup \mathscr{P}_{DM_c}$ is the DM layer's perception ruleset composed of both the human $\mathscr{P}_{DM_h}$, and the computational $\mathscr{P}_{DM_c}$ agent rules;
- $\mathscr{T}_{DM_h} \cup \mathscr{T}_{DM_c}$ is the DM layer's action ruleset composed of both the human $\mathscr{T}_{DM_h}$, and the computational $\mathscr{T}_{DM_c}$ agent rules.

6.1 Design process

According to our framework, each fundamental process for designing is modeled through a series of perception and action strategies. Each instance of these strategies represents a different step, activity, or operation within the process. Therefore, in the following sections, we break down the fundamental processes for designing into smaller steps and provide a detailed description and definition in terms of our framework. We have labeled them for descriptive purposes.

In some cases, there are execution dependencies between steps, or situations where some steps are performed in parallel. For these cases, we provide activity diagrams to make these characteristics explicit.

When applicable, we integrate a detailed description of the tool's GUI, emphasizing how the design operations involved are implemented in it and how the different elements of the framework are supported.

6.1.1 Formulation

In this system, the formulation processes for each of the design levels share a pattern of steps or activities depicted in (Fig. 6.1).

6.1.1.1 Game experience formulation

$$\left\{\hat{x}_{Pr} \underset{\rightarrow}{\subset} \check{x}_{P_h},\ \hat{x}_{Pn} \underset{\rightarrow}{\subset} \check{x}_{P_h},\ \check{x}_{P_h} \underset{\dashrightarrow}{\subset} \check{x}_{Pe_h},\ \check{x}_{Pm_h} \underset{\dashrightarrow}{\subset} \check{x}_{Pe_h},\ \check{x}_{Pe_h} \rightleftarrows \vec{x}_{Pe_h}\right\} \subseteq \mathscr{P}_{PD_h},$$

$$\left\{\vec{x}_{Pe_h} \rightarrow \hat{x}_{Pe_h}\right\} \subseteq \mathscr{T}_{PD_h}.$$

External requirements interpretation: h interprets explicit external game experience requirements (i.e., the desired game experience to be evoked in the player)

$$\hat{X}_{Pr} \underset{\rightarrow}{\subset} \check{x}_{P_h}.$$

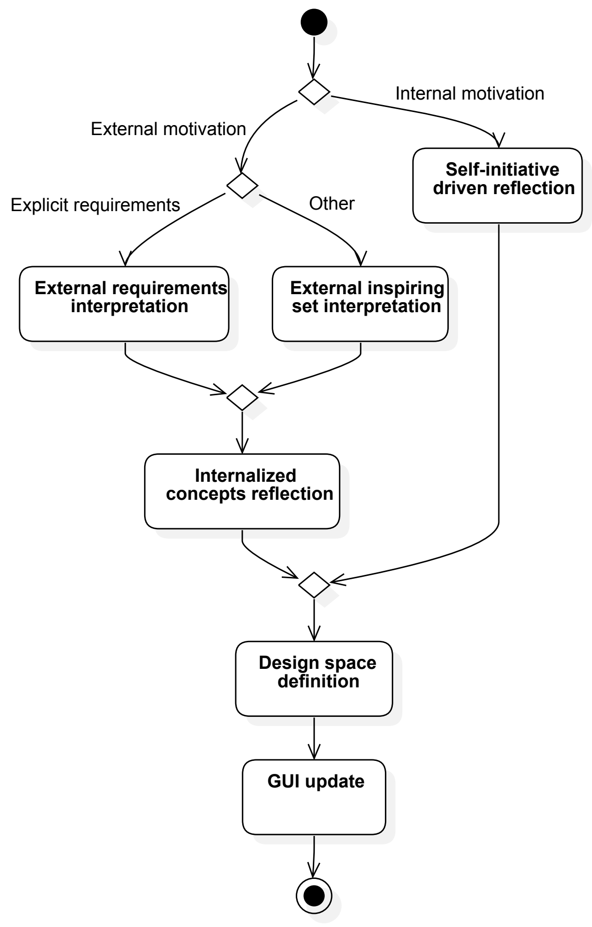

Internal motivation
External motivation
Self-initiative driven reflection
Explicit requirements
Other
External requirements interpretation
External inspiring set interpretation
Internalized concepts reflection
Design space definition
GUI update

Figure 6.1: An activity diagram representing the overall formulation process pattern implemented in the three design levels, game experience $\boldsymbol{P}$, game dynamics $\boldsymbol{D}$, and game mechanics $\boldsymbol{M}$.

External inspiring set interpretation: h interprets a game experience design concept from an external inspiring set (e.g., an article on the Web, an existing game, a conversation)

$$\hat{X}_{Pn} \underset{\rightrightarrows}{\subset} \check{x}_{P_h}.$$

Internalized concepts reflection: Through reflection, and based on her/his knowledge, experience, preferences, and memories, h accommodates and augments the interpreted version of the externally defined game experience, attempts to model it (mentally) in terms of one or more PLEX playful experiences categories, and roughly determines some of their intensity ranges

$$\check{x}_{P_h} \underset{\rightrightarrows}{\subset} \check{x}_{Pe_h}.$$

Self-initiative driven reflection: Through reflection, and based on her/his knowledge, experience, preferences, and memories, h imagines a game experience in terms of one or more PLEX playful experiences categories and roughly determines some of their intensity ranges

$$\check{X}_{Pm_h} \underset{\rightrightarrows}{\subset} \check{x}_{Pe_h}.$$

Design space definition: Through focusing, h establishes her/his game experience design space (i.e., the internal representation of the expected game experience design concept)

$$\check{x}_{Pe_h} \rightleftarrows \vec{x}_{Pe_h},\ \mathcal{D}_{P_h} = \vec{x}_{P_h}.$$

GUI update: Through the game experience panel's expected-experience related widgets, h selects the PLEX cards that correspond with her/his game experience design space's dimensions, setting for each category its intensity range

$$\vec{x}_{Pe_h} \rightarrow \hat{x}_{Pe_h}.$$

The game experience panel of the tool's GUI allows the human designer h configuring her/his game experience design space (i.e., setting her/his expected game experience design concept).

1. The panel starts with an empty design space (Fig. 6.2), so the designer can add one or more game experience design dimensions into her/his design space.

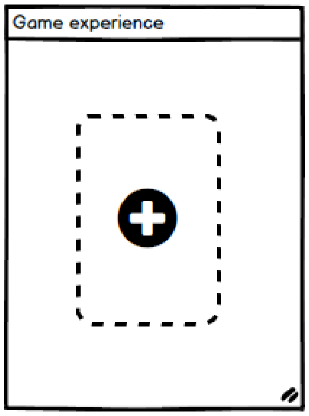


Figure 6.2: Game experience panel - empty game experience design space. The dotted rounded rectangle represents an 'empty' card (i.e., PLEX card).

2. By pressing the '+' button, the designer invokes a 'carousel' widget that allows her/him to navigate through the different PLEX categories and select the desired category (Fig. 6.3).
3. The selected category then becomes a new game experience design dimension (Fig. 6.4).
4. The designer sets the range of the dimension through a double thumb vertical slider (Fig. 6.5), set to its full range by default.
5. The designer can also set the dimension range with the assistance of the PLEXQ questionnaire. With this option, the designer only requires to answer a small 3-item 4-scale[2] Likert survey (Fig. 6.6).

This process is performed for each dimension the designer decides to incorporate into her/his design space.

[2]Although PLEXQ originally uses a 5-scale Likert survey for each item, we employ a 4-scale (Brown 2010) to avoid neutral responses such as 'non-applicable' or 'neither agree nor disagree.'

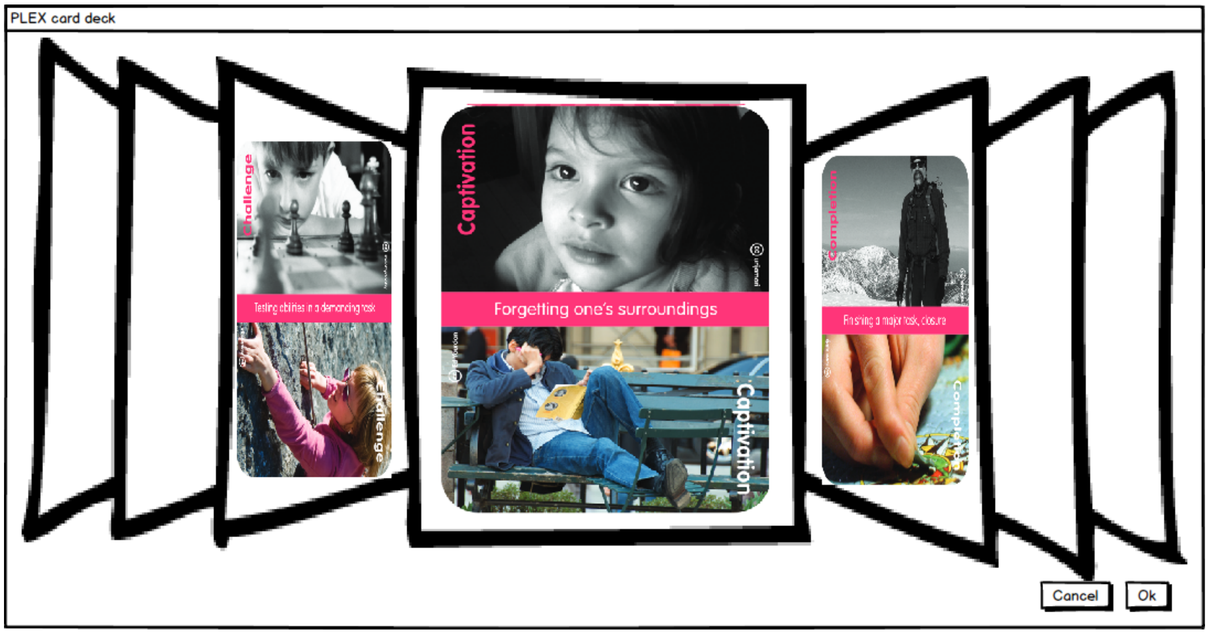

PLEX card deck
Challenge
Testing abilities in a demanding task
Captivation
Forgetting one's surroundings
Completion
Finishing a major task, closure
Cancel
Ok

Figure 6.3: PLEX card deck

6.1.1.2 Game dynamics formulation

$$\mathscr{P}_{PD_h} \supseteq \left\{ \hat{x}_{Dr} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{D_h},\ \hat{x}_{Dn} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{D_h},\ \check{x}_{D_h} \mathrel{\underset{\dashrightarrow}{\subset}} \check{x}_{De_h},\ \check{x}_{Dm_h} \mathrel{\underset{\dashrightarrow}{\subset}} \check{x}_{De_h},\ \check{x}_{De_h} \rightleftarrows \vec{x}_{De_h} \right\} \subseteq \mathscr{P}_{DM_h},$$

$$\mathscr{T}_{PD_h} \supseteq \left\{ \vec{x}_{De_h} \rightarrow \hat{x}_{De_h} \right\} \subseteq \mathscr{T}_{DM_h}.$$

External requirements interpretation: h interprets explicit external game dynamics requirements that may impose constraints such as particular gameplay scenarios, maximum game duration, and the forcing/prevention of specific game objects types' usage in the game

$$\hat{X}_{Dr} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{D_h}.$$

External inspiring set interpretation: h interprets a game dynamics design concept from an external inspiring set

$$\hat{X}_{Dn} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{D_h}.$$

Internalized concepts reflection: Through reflection, and based on her/his knowledge, experience, preferences, and memories, h accommodates and augments the interpreted version of the externally defined game dynamics constraints, attempts to model it (mentally) in terms of the set of design dimensions currently present in the game dynamics panel of the tool's GUI as well as any other design dimensions of the types 'gameplay scenario' and 'game object type counter' that are considered appropriate, and roughly determines some of their ranges

$$\check{x}_{D_h} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{De_h}.$$

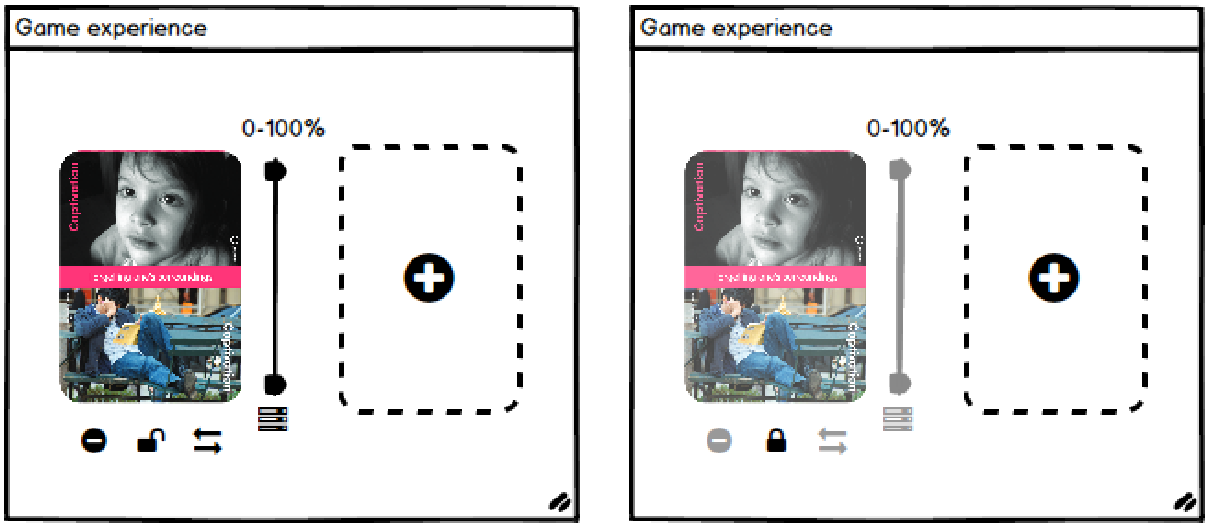


Figure 6.4: Game experience panel with one dimension. The 'minus' button removes the card (i.e., removes the design dimension from the design space). The 'padlock' toggle button allows the designer to lock or unlock both the PLEX card and its range (i.e., to fix some/all design dimensions of the game experience design space). If a card is unlocked (left), the designer is free to remove the dimension, set or update its range, or replace it with another card from the card deck (with the 'opposite directions' parallel arrows' button). If a card is locked (right), the designer cannot replace it, nor remove it, nor modify its range. The '3-stacked bars' button activates the PLEXQ mini-survey to assist in setting the dimension range (only if the card is unlocked).

Self-initiative driven reflection: Through reflection, h imagines a game dynamics in terms of the set of design dimensions currently present in the game dynamics panel of the tool's GUI and any other design dimensions of the types 'gameplay scenario' and 'game object type counter' that are considered appropriate, and roughly determines some of their ranges

$$\check{X}_{Dm_h} \subsetneqq \check{x}_{De_h}.$$

Design space definition: Through focusing, h establishes her/his game dynamics design space (i.e., the internal representation of the expected game dynamics design concept)

$$\check{x}_{De_h} \rightleftarrows \vec{x}_{De_h},\ \vec{x}_{De_h} = \mathcal{D}_{D_h}.$$

GUI update: Through the game dynamics panel's expected-dynamics related widgets, h adds any required 'gameplay scenario' or 'game object type counter' design dimensions and may configure their ranges and the range of any other dimensions to reflect her/his game dynamics design space.

$$\vec{x}_{De_h} \to \hat{x}_{De_h}.$$

The game dynamics panel of the tool's GUI allows the human designer h to configure her/his game dynamics design space (i.e., setting her/his expected game dynamics design concept).

1. The panel starts with a default design space composed of a set of computer-analyzed game dynamics design dimensions and the three default 'gameplay scenario' dimensions whose actual values are derived by the human analyzer (Fig. 6.7).

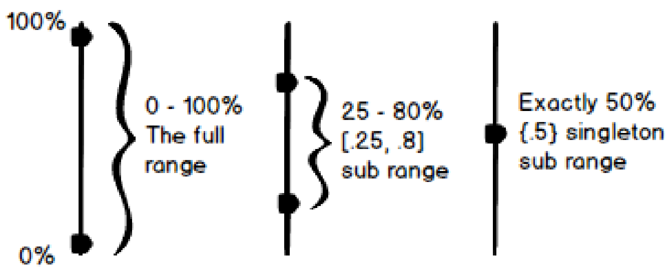


Figure 6.5: Dimension's range vertical slider widget. The double thumb feature of the sliders allows the designer to define a region within the dimension's range, spanning from the full range (each thumb positioned at the slider extremes) to a region, to a specific value (both thumbs positioned at the same value).

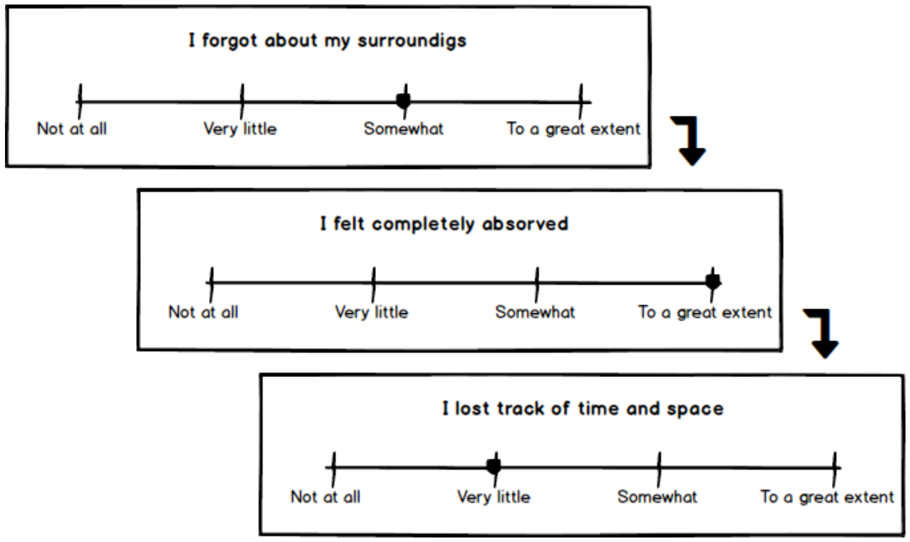


Figure 6.6: PLEXQ 3-item 4-scale Likert mini-survey.

2. Through the panel, the human designer h can add as many 'gameplay scenario' and 'game object type counter' design dimensions into her/his design space as required. By pressing the '+' button of the 'gameplay scenarios' or 'game object type counters' sections, the designer adds a new game dynamics design dimension of any of those types by simply entering/selecting the name of the gameplay scenario or the type of game object, respectively (Fig. 6.8).

3. The h designer configures her/his game dynamics design space by setting the range of each required design dimensions according to her/his mental design space. Dimensions whose original ranges are defined within the unit interval I are represented with horizontal sliders and share functionality with the game experience panel's vertical sliders. On the other hand, dimensions with ranges defined within the non-negative reals $\mathbb{R}_{\geq 0}$, positive integers (natural numbers) $\mathbb{N}$, or non-negative integers $\mathbb{W}$ are represented with a custom interval widget (Fig. 6.9).

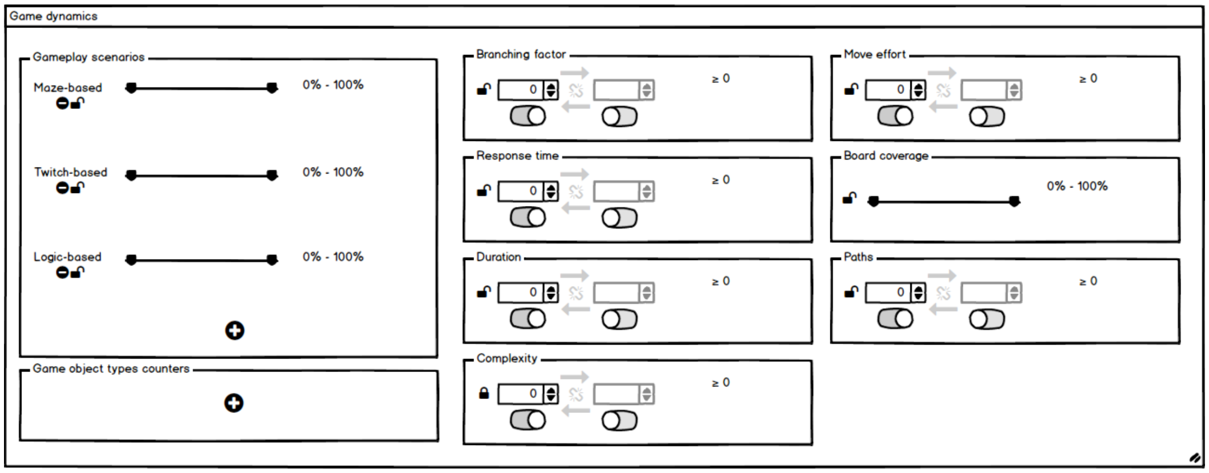


Figure 6.7: Game dynamics panel - default game dynamics design space. Both computer-analyzed and human-analyzed game dynamics design dimensions are grouped and located in their own sections of the panel. All design dimensions are set to their full range by default (i.e., representing the 'empty' design concept). The section dedicated to the 'game object type counter' design dimensions is displayed, but no dimensions are contained in it by default.

6.1.1.3 Game mechanics formulation

$$\left\{ \hat{x}_{Mr} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{M_h},\ \hat{x}_{Mn} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{M_h},\ \check{x}_{M_h} \mathrel{\underset{\dashrightarrow}{\dashrightarrow}} \check{x}_{Ma_h},\ \check{x}_{Mm_h} \mathrel{\underset{\dashrightarrow}{\dashrightarrow}} \check{x}_{Ma_h},\ \check{x}_{Ma_h} \rightleftarrows \vec{x}_{Ma_h} \right\} \subseteq \mathscr{P}_{DM_h},$$

$$\left\{ \vec{x}_{Ma_h} \rightarrow \hat{x}_{Ma} \right\} \subseteq \mathscr{T}_{DM_h}.$$

External requirements interpretation: *h* interprets explicit external game mechanics requirements that may impose some constraints such as the game board shape and size, the forcing/prevention of specific game objects' usage in the game, the placement of specific game objects in the board, and pre-existing game rules and win conditions

$$\hat{X}_{Mr} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{M_h}.$$

External inspiring set interpretation: *h* interprets a game from an external inspiring set

$$\hat{X}_{Mn} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{M_h}.$$

Internalized concepts reflection: Through reflection, and based on her/his knowledge, experience, preferences, and memories, *h* accommodates and augments the interpreted version of the externally defined game mechanics constraints, attempts to model them (mentally) in terms of the set of design dimensions currently present in the game editor of the tool's GUI and roughly determines some of their ranges. For example, *h* may resolve that the game would require one player token and one finish token (perhaps positioned in random cells at this time) so that a canonical maze win condition could be fired

$$\check{x}_{M_h} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{Ma_h}.$$

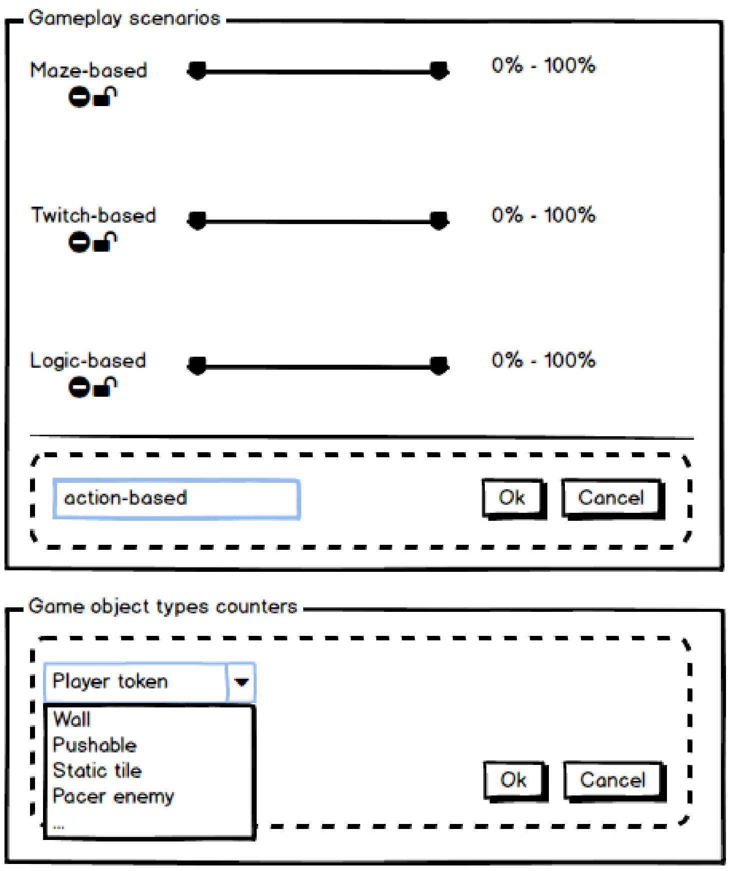


Figure 6.8: Game dynamics panel - creating new design dimensions. Adding a new gameplay scenario (top panel). Adding a new game object type counter (bottom panel).

Self-initiative driven reflection: Through reflection, h imagines a game in terms of the set of design dimensions currently present in the game editor of the tool's GUI and roughly determines some of their ranges

$$\check{X}_{Mm_h} \subseteq \check{x}_{Ma_h}.$$

Design space definition: Through focusing, h establishes her/his game mechanics design space (i.e., the internal representation of the actual game mechanics design concept)

$$\check{x}_{Ma_h} \rightleftarrows \vec{x}_{Ma_h}, \vec{x}_{Ma_h} = \mathcal{D}_{M_h}.$$

GUI update: h roughly sketches the game via the game editor panel to reflect her/his current game mechanics design space, such as shaping and sizing the game board, configuring layers and aliases, and placing specific game objects into the board or entering specific win conditions or game rules if dictated by some constraints

$$\vec{x}_{Ma_h} \rightarrow \hat{x}_{Ma}.$$

The game editor panel of the tool's GUI allows the human designer h configuring her/his game mechanics design space.

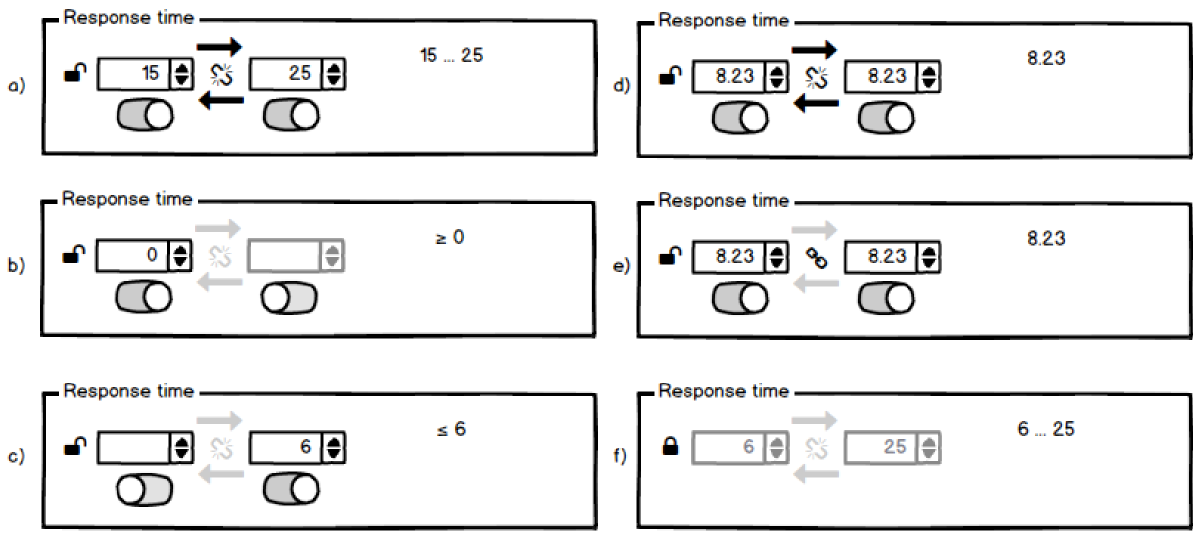


Figure 6.9: Game dynamics panel - custom interval widget. The custom widget provides mechanisms to define intervals. Values can be typed directly (the widget will prevent invalid entered values) or selected with the number-stepper up and down arrows. Each of the two number-stepper widgets represent one side of a closed interval (a). Any of the two sides of the interval can be disabled (one at a time) with their switches to set either left-closed (b) or right-closed intervals (c). Right and left arrow buttons set the range of the design dimension to a single value (d). The 'right arrow' button copies the value from the right to the left, and the 'left arrow' button does the opposite. A dimension's range can be configured to be single-valued by enabling/disabling the synchronization feature through the 'chain link' toggle buttons (broken link and closed link) (e). As with the sliders, this custom widget can be locked or unlocked through the 'padlock' toggle button (f).

1. The panel starts with a default design space composed of two fixed sets of game object types (background and non-background), two empty layers (the background and the first non-background layer), a 1×1 squared grid board or game space, and the background (empty) alias (Fig. 6.10).
2. Through a pair of number-stepper controls (rows and columns), the designer can shape and size the game space (i.e., the *cells* design dimension). Adding a cell is equivalent to adding a dimension into the design space (Fig. 6.11).
3. The designer can configure the *layers* design dimension by creating individual layer containers and populating them with game object types picked from the game object pools. Reactively, the tool's GUI keeps synced both the background layer container and the default *background* container that corresponds to the *alias* dimension labeled as 'background'[3] (Fig. 6.12).
4. The designer can configure the *aliases* design dimension by creating individual alias containers (different from the *background* auto-created alias), labeling them, and populating them with objects already placed in any of the layers (Fig. 6.13).
5. If required by some requirements or initial inspiration, the designer may

[3]By keeping the alias design dimension synced with the background layer, the designer may be able to use the *background* alias in both win conditions and game rules.

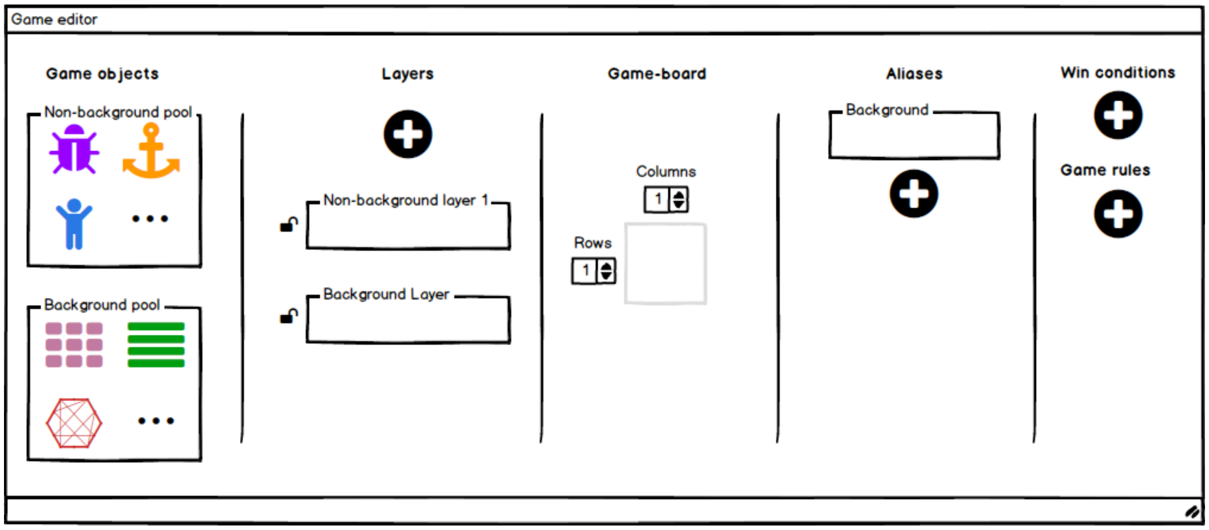


Figure 6.10: Game editor panel - default game mechanics design space. Several dimensions are represented in the game editor: the section *Game objects* in the first column represents the set of all possible game objects available in the tool (i.e., C_{gos}) partitioned into two game object containers or pools, the pool of background game objects and the pool of non-background game objects, respectively. In the second column, the layer design dimensions are represented as stacked game object containers (empty by default). In the third column, *cell* type compound design dimensions are represented through a grid of light gray cells that shape the game board (1×1 by default). In the fourth column, the alias design dimensions are represented as labeled game object containers (empty by default), including the default *background* alias. Finally, the win conditions and game rules are represented in the last column. Both types of dimensions are empty by default.

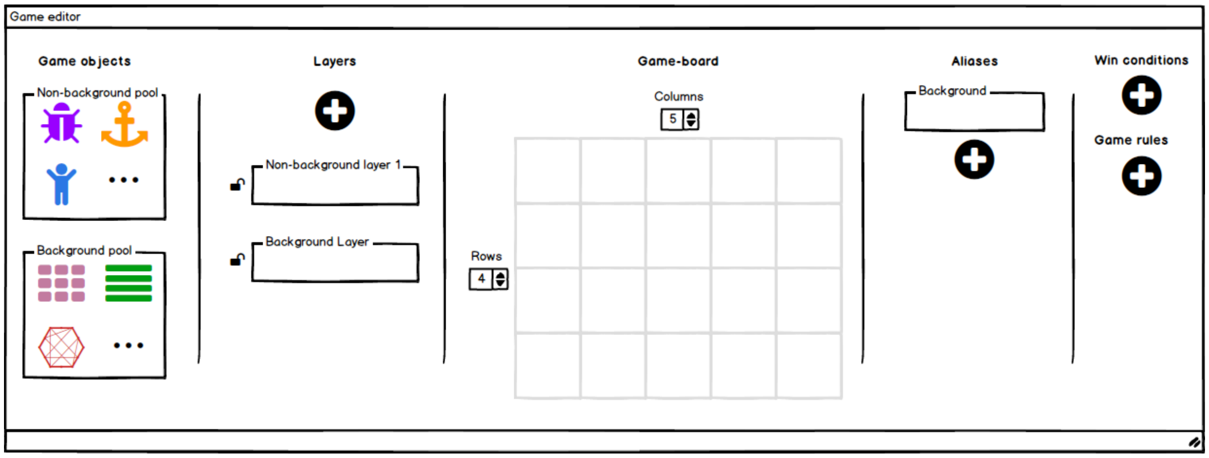


Figure 6.11: Game editor panel - Game space defined after setting the game board size.

(a) place specific game objects into specific cells (See Section 6.1.2.2 and Fig. 6.15),

(b) predefine specific win conditions (See Section 6.1.2.2 and Fig. 6.17),

(c) and predefine the number of game rules, their structure (number of patterns per rule and number of cells per pattern), and specific game rules (See Section 6.1.2.2 and Fig. 6.18).

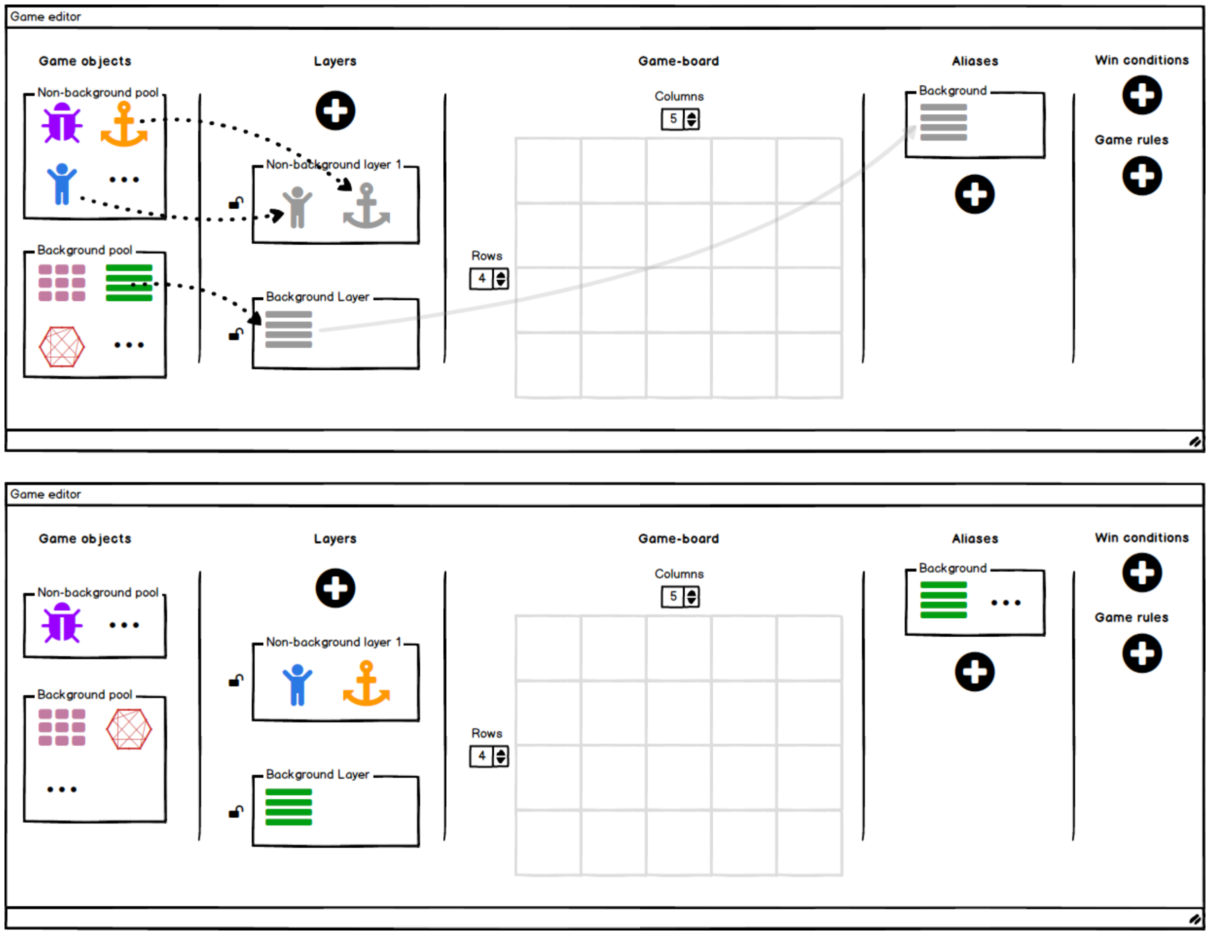


Figure 6.12: Game editor panel - Configuring layers. In the upper figure, the human designer drags and drops a background object type from the background pool into the background layer and a non-background object from the non-background pool into the first non-background layer. In the lower figure, the objects are placed in their corresponding layers. Any background game object added into the background layer is automatically added into the *background* alias container. The GUI allows the designer to add new layers (with the '+' button) on top of the default background and first non-background layers, and to lock individual layers with the 'padlock' button.

6.1.2 Synthesis

6.1.2.1 Game dynamics synthesis

$$\left\{ \vec{x}_{Pe_h} \Mapsto \vec{x}_{De_h},\, \vec{x}_{De_h} \rightarrow \hat{x}_{De_h} \right\} \subseteq \mathscr{T}_{PD_h}$$

Design synthesis: Based on her/his expected game experience, h creates or updates a mental expected game dynamics design concept by setting specific values or regions on the design dimensions that span the game dynamics design space

$$\vec{x}_{Pe_h} \Mapsto \vec{x}_{De_h}.$$

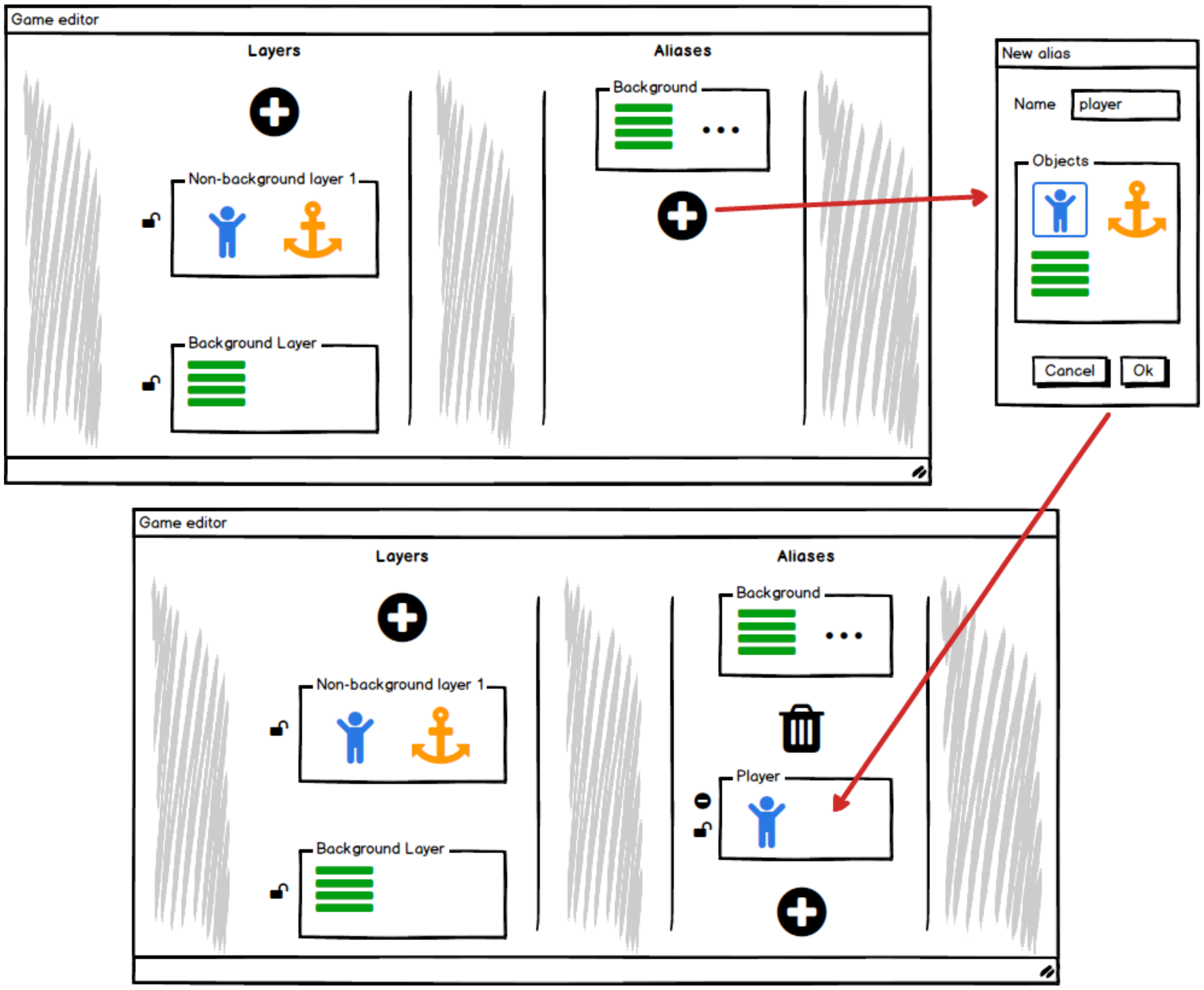


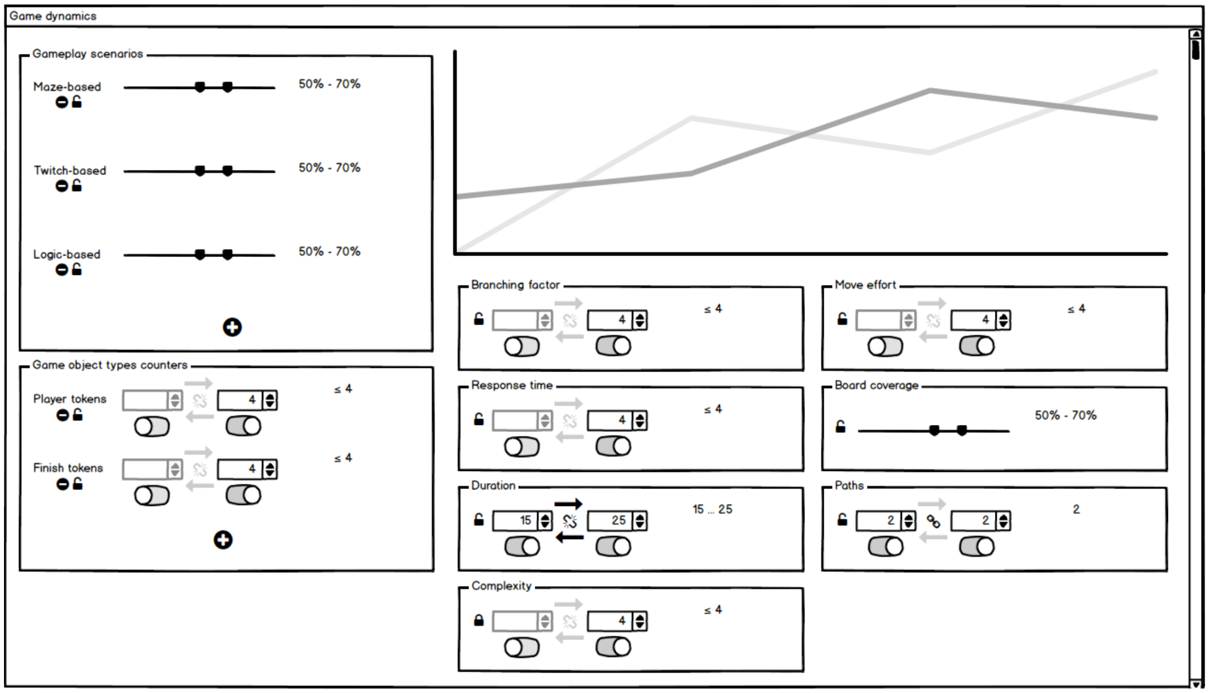
Figure 6.13: Game editor panel - Configuring aliases. In the top-left figure, the human designer presses the 'plus' button to invoke the 'New alias' dialog window at the right. In the dialog window, the designer enters the new alias name and selects one or more game objects from the pool in the bottom. The pool contains the set of all game objects already assigned into layers. The figure on the bottom shows the created alias. The GUI allows the designer to lock individual aliases with the 'padlock' button.

GUI update: h manipulates the game dynamics panel's expected-dynamics related widgets to reflect her/his new or updated expected game dynamics design concept

$$\vec{x}_{De_h} \rightarrow \hat{x}_{De_h}.$$

To externalize her/his synthesized game dynamics design concept, the designer h uses the game dynamics panel of the tool's GUI and follows a workflow very similar to that of the formulation process, with the exception of adding or removing design dimensions, or widening their ranges (Fig. 6.14).

The GUI does not prevent the designer from performing those actions (i.e., adding or removing design dimensions, or widening their ranges); therefore, if the designer adds or removes dimensions or broadens the range of one or more dimensions, it would merely mean that the designer is externalizing a reformulation's result or a simultaneous synthesis-reformulation. This is practical evidence that the fundamental processes for designing are not a list of activities that are always performed in a specific order but processes performed by the designer as required by her/his current situation and, in some cases, so fast that they seem to be performed simultaneously.


Game dynamics

Gameplay scenarios
Maze-based 50% - 70%
Twitch-based 50% - 70%
Logic-based 50% - 70%

Game object types counters
Player tokens 4 ≤ 4
Finish tokens 4 ≤ 4

Branching factor 4 ≤ 4
Move effort 4 ≤ 4
Response time 4 ≤ 4
Board coverage 50% - 70%
Duration 15 25 15 ... 25
Paths 2 2 2
Complexity 4 ≤ 4

Figure 6.14: Game dynamics panel - a representation of an expected game dynamics design concept. The designer sets the game dynamics design dimensions' ranges to specific values or constrained regions. The game dynamics panel also provides a widget with a chart view of the current game dynamics design concept (top right), which is updated in real-time as the designer defines a dimension's value.

6.1.2.2 Game mechanics synthesis

$$\left\{ \vec{x}_{De_h} \Mapsto \vec{x}_{Ma_h}, \vec{x}_{Ma_h} \rightarrow \hat{x}_{Ma} \right\} \subseteq \mathscr{T}_{DM_h}$$

Design synthesis: Based on her/his expected game dynamics, h creates or updates a mental game (actual design) by setting specific values or regions on the design dimensions that span the game mechanics design space.

$$\vec{x}_{De_h} \Mapsto \vec{x}_{Ma_h}$$

GUI update: Through the game editor, h creates/modifies the 'physical' representation of her/his actual game mechanics design concept

$$\vec{x}_{Ma_h} \rightarrow \hat{x}_{Ma}.$$

The game editor panel of the tool's GUI allows the human designer h to externalize her/his synthesized game.

1. The designer can place game objects (tiles and tokens) at any cell of the game board by simply dragging the desired object into the target cell and dropping it there.
2. If the designer needs to remove an object from a cell, they simply drag it out of the board (Fig. 6.15).

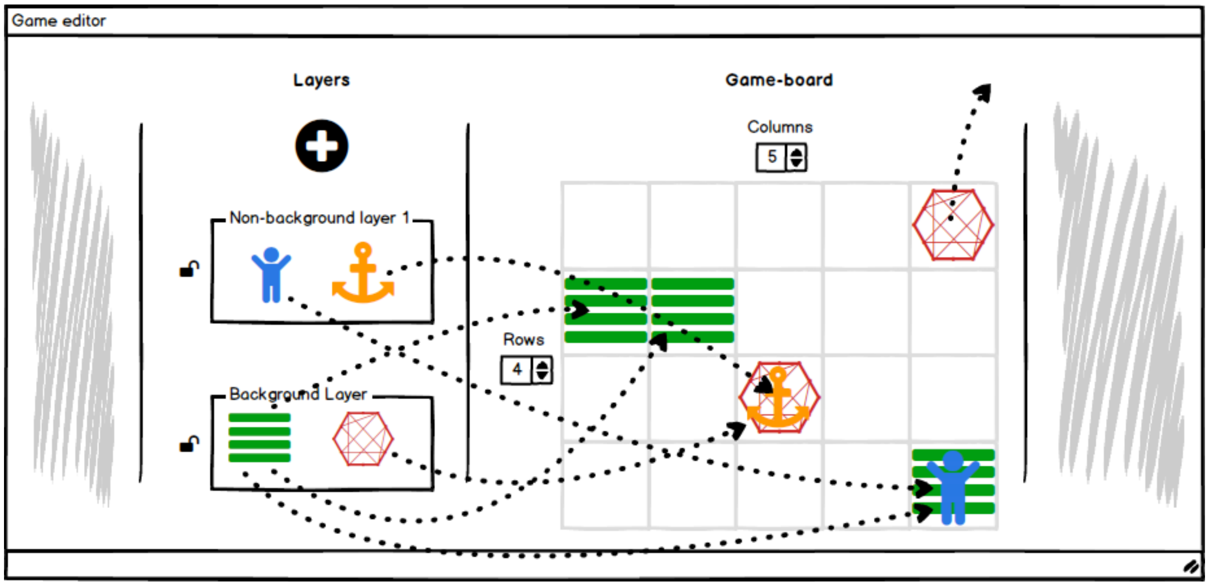


Figure 6.15: Game editor panel - Laying out the game. A cell is nothing more than a stack of objects whose order is governed by how the *layer* design dimensions are defined. The designer can select any object from any of the layer containers and drag it into any of the cells in the game space. The GUI will allow the designer to place (drop) the dragged object only if the cell is empty and the object belongs to the background layer or if the dragged object belongs to a layer higher than the layer to which the last object (the top of the stack) already placed in the cell belongs. No matter the cell contents, placing a background object will override (clear) the cell, leaving only the new background object. The designer can also remove objects (in a stack fashion) from any cell by dragging them out of the grid and dropping them there.

3. The game editor also provides the designer with another, more fluent method for placing/removing objects to/from the game space by using a 'tool selection' and 'stamp tool' UxD metaphors (Fig. 6.16).
4. Win conditions and game rules are 'written' in the game editor fully visually through simple widgets that reassemble the textual structure of the rules and win conditions expressed in the PuzzleScript language. Each of the win conditions and game rules base design dimensions are represented by specific widgets that are hierarchically grouped to represent various compound dimensions. The tool's GUI not only allows the designer to add, edit,

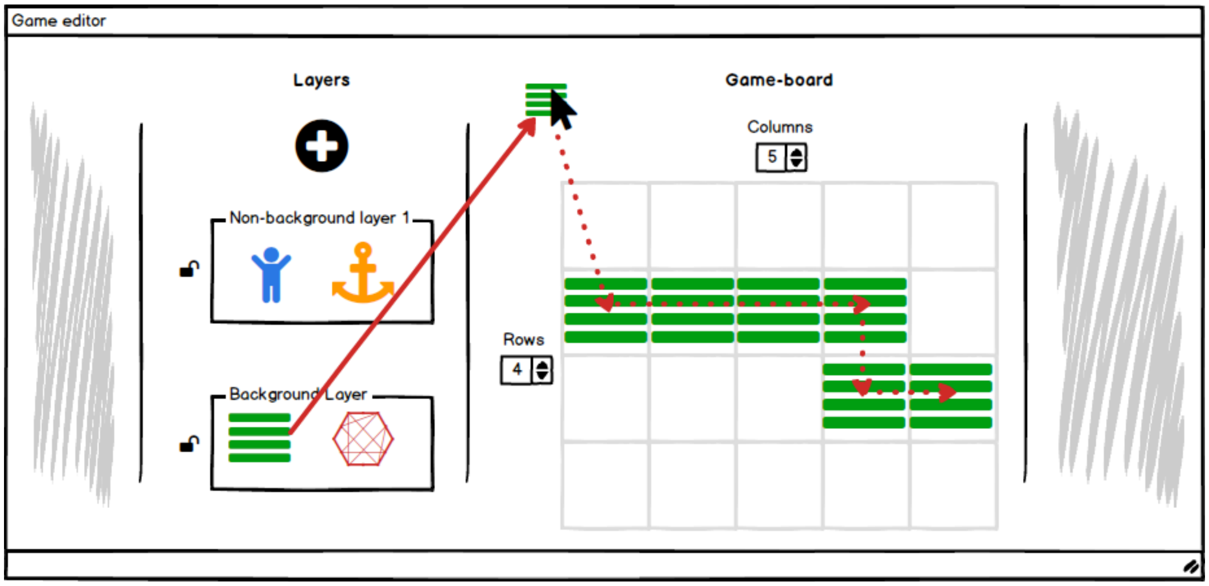


Figure 6.16: Game editor panel - Fluently laying out the game. If the designer selects an object (by clicking on it) from any of the layer containers (tool selection), the regular pointer cursor will be transformed into a copy of the image of the selected object (stamp tool). Having the tool selected, the designer can “stamp” the object in any number of cells over which the cursor hovers while holding the pointer button pressed. If the object at the top of a cell corresponds to the same object selected, the “stamping” action will remove the object from the cell.

remove, lock, and unlock both win conditions and game rules but to add or remove any number of cells per pattern and any number of patterns per game rule to express not only simple but really complex game rules (Figs. 6.17 and 6.18).

6.1.3 Analysis and evaluation

6.1.3.1 Game mechanics analysis

The analysis process at the game mechanics level is a collaborative process that involves dependencies and parallelism between both h and c operations execution (Fig. 6.19).

$$\left\{\hat{x}_{Ma} \subsetneqq \check{x}_{Ma_c},\ \check{x}_{Ma_c} \dashrightarrow \check{x}_{M\alpha_c},\ \check{x}_{Ma_c} \dashrightarrow \check{x}_{I_c},\ \check{x}_{M\alpha_c} \rightleftarrows \vec{x}_{M\alpha_c},\ \check{x}_{Da_c} \rightleftarrows \vec{x}_{Da_c},\ \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c}\right\} \subseteq \mathscr{P}_{DM_c},$$

$$\left\{\vec{x}_{M\alpha_c} \to \hat{x}_{M\alpha_c},\ \vec{x}_{Da_c} \to \hat{x}_{Da_c},\ \vec{x}_{I_c} \to \hat{x}_{I_c}\right\} \subseteq \mathscr{T}_{DM_c},\ \left\{\check{x}_{Ma_c} \mapsto \check{x}_{Da_c}\right\} \subseteq \mathring{\mathscr{E}}_{DM_c},$$

$$\left\{\hat{x}_{M\alpha_c} \subsetneqq \check{x}_{M\alpha_h^c},\ \check{x}_{Da_h} \rightleftarrows \vec{x}_{Da_h}\right\} \subseteq \mathscr{P}_{DM_h},\ \left\{\vec{x}_{Da_h} \to \hat{x}_{Da_h}\right\} \subseteq \mathscr{T}_{DM_h},\ \left\{\check{x}_{M\alpha_h^c} \mapsto \check{x}_{Da_h}\right\} \subseteq \mathring{\mathscr{E}}_{DM_h}.$$

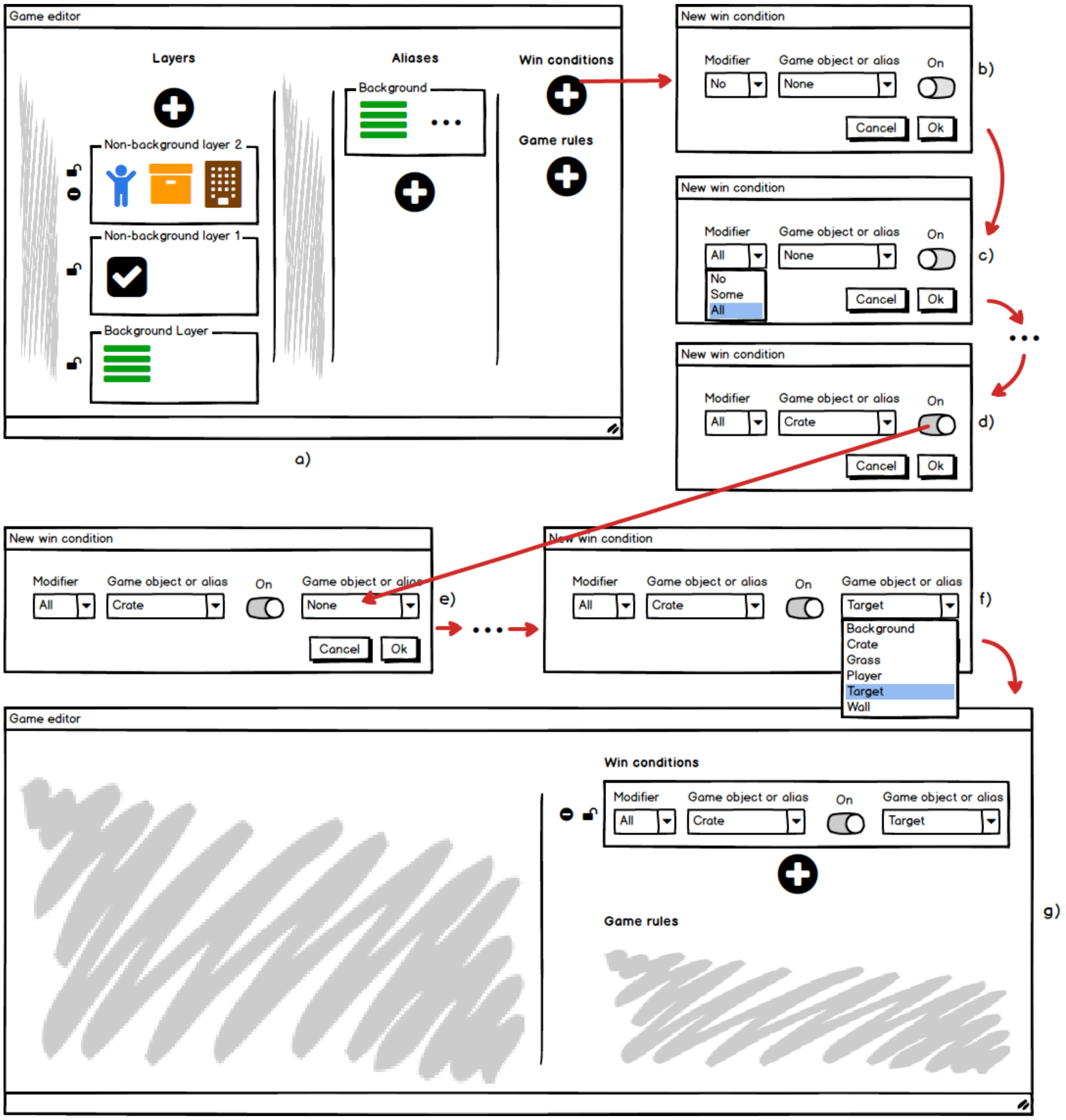


Figure 6.17: Game editor panel - Entering win conditions. The figure shows the steps to enter the win condition **`All Crate On Target`**. The game objects `Crate` (yellow box), and `Target` (black square with a check mark) are currently placed in the second and first non-background layers, respectively, so they are available to be used in win conditions. a) The designer presses the '+' button in the Win conditions section of the game editor. b) A dialog appears with an empty win condition. c) The designer sets both the *modifier* and the *game object* design dimensions of the win condition. d) With a switch widget the designer activates the *on-game-object* design dimension of the win condition. e) The *on-game-object* dimension is added to the dialog. f) The designer sets the *on-game-object* dimension value. g) Finally, the win condition is added to the set of win conditions of the game being designed.

Actual design interpretation (c): whenever h updates her/his actual game design (game me-

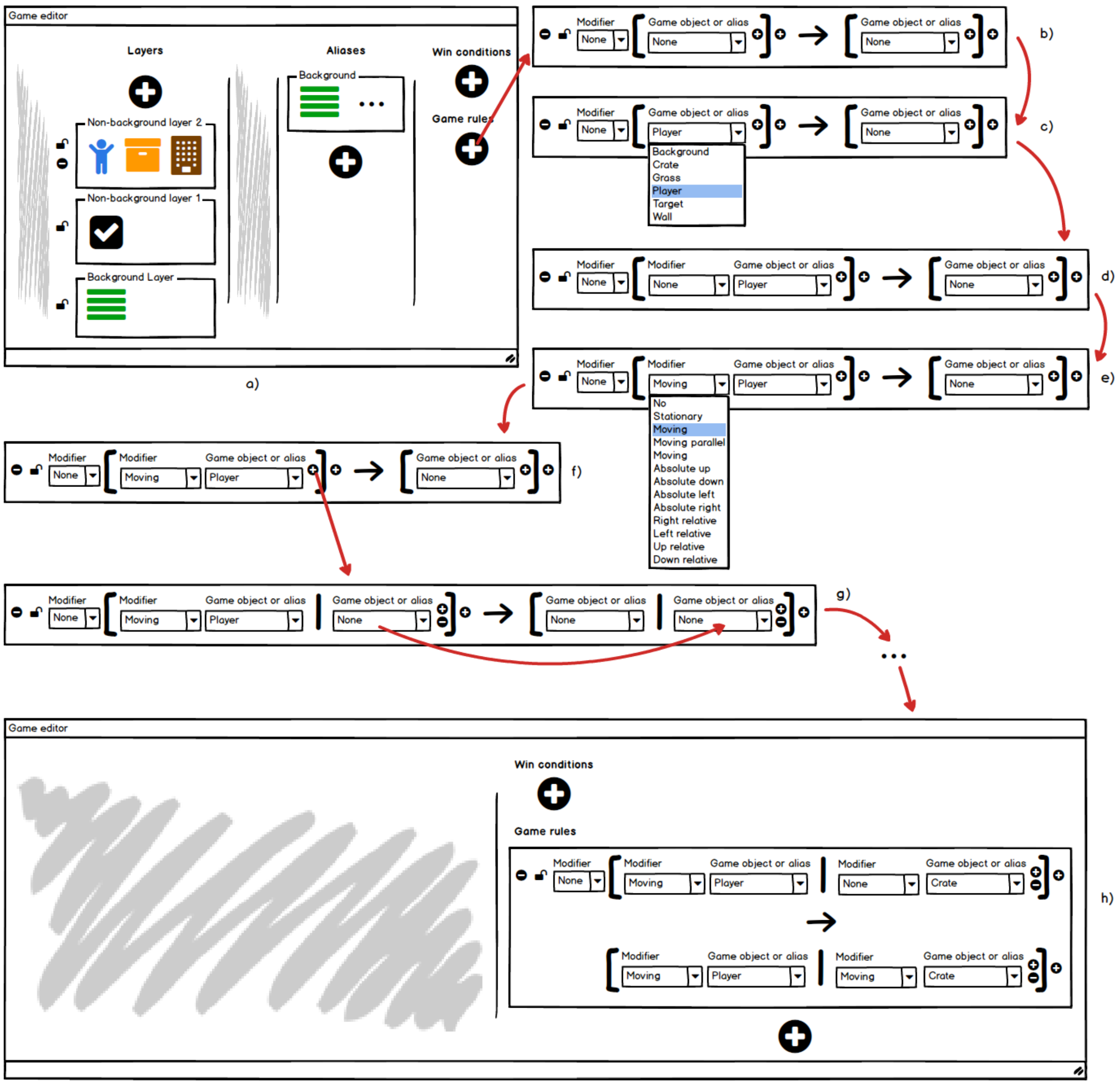


Figure 6.18: Game editor panel - Entering game rules. The figure shows the steps to enter the game rule `[ > Player | Crate ] -> [ > Player | > Crate ]` interpreted as: replace a Player moving towards a contiguous Crate, with the Player moving and the Create also moving into the same direction of the Player's movement. The game objects `Crate` (yellow box), and `Player` (blue human shape) are currently placed in the second non-background layer so they are available to be used in game rules. a) The designer presses the '+' button in the Game rules section of the game editor. b) A simple empty game rule —a rule with no modifier composed of one pattern of one empty cell— is added into the set of game rules of the game. c) The designer sets the first *game object* (`Player`) of the left cell. d) A *modifier* dimension for the newly *game object* is added into the game rule. e) The designer sets the *modifier*. f) The designer presses the '+' button inside the left cell to add another *game object* into the cell. g) A new empty *game object* dimension is added into both the left and the right cells to keep the symmetry required by PuzzleScript. h) Finally, the designer sets the values of the remaining cells' design dimensions (both left and right sides) to complete the rule definition.

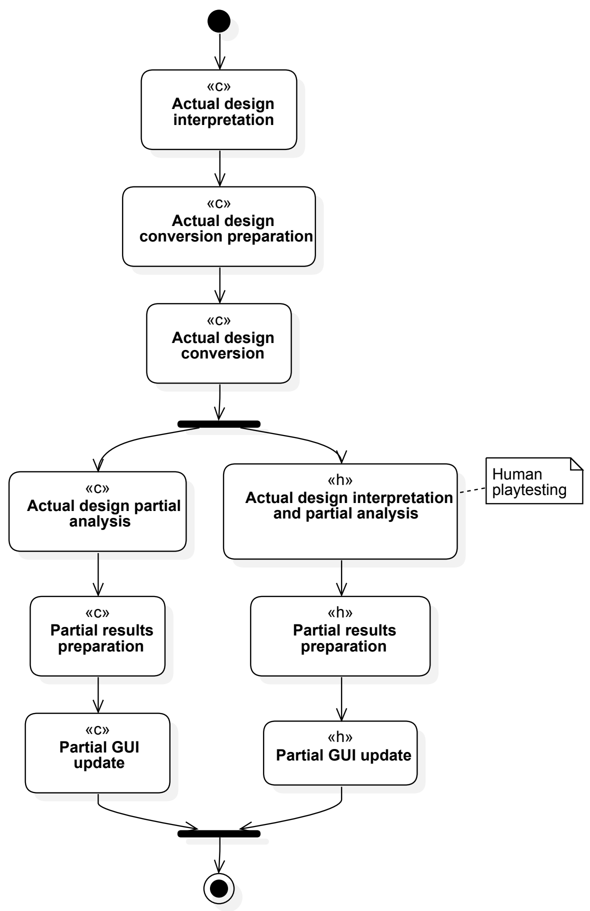

«c»
Actual design interpretation
«c»
Actual design conversion preparation
«c»
Actual design conversion
«c»
Actual design partial analysis
«h»
Actual design interpretation and partial analysis
Human playtesting
«c»
Partial results preparation
«h»
Partial results preparation
«c»
Partial GUI update
«h»
Partial GUI update

Figure 6.19: An activity diagram representing the overall analysis process at the M game mechanics level.

chanics level), c creates/updates an internal version of the actual game design

$$\hat{x}_{Ma} \looparrowright \check{x}_{Ma_c}.$$

Actual design conversion (c): c augments the updated interpreted version of the actual game design with a representation of the game encoded into the PuzzleScript scripting language

$$\check{x}_{Ma_c} \dashrightarrow \check{x}_{M\alpha_c}.$$

Human-playable game preparation (c): c selects the augmented part (PuzzleScript language version) of the actual game design representation

$$\check{x}_{M\alpha_c} \rightleftarrows \vec{x}_{M\alpha_c}.$$

Human-playable game rendering (c): c converts the augmented part of the actual game design representation into a human playable game by compiling and executing the PuzzleScript language code with the PuzzleScript game engine which is linked to a canvas frame of the tool's GUI (Fig. 6.20)

$$\vec{x}_{M\alpha_c} \rightarrow \hat{x}_{M\alpha_c}.$$

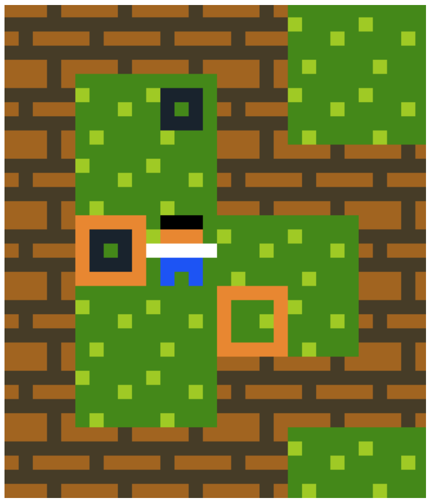


Figure 6.20: Human-playable game

Actual design partial analysis start notification (c): c places a looping progress bar widget labeled as 'Analyzing' in the tool's GUI to notify that analytical derivation has commenced (Fig. 6.21)

$$\check{x}_{M\alpha_c} \dashrightarrow \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

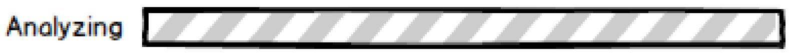


Figure 6.21: Game mechanics–dynamics analytical derivation notificacion.

Actual design partial analysis (c): The c partial analysis process[4] starts with the computation of the intrinsic dynamics design dimensions values. *Complexity* is derived by counting the number of game rules and game object types present in them, while the *game-object-type-counter* design dimensions values are derived by directly counting the number of type-objects placed in the game board.

Next, c performs automatic playtesting by simulating four different players through four search algorithms —breadth-first search (BFS), depth-first search (DFS), Greedy Best-first search, and A^*—, and builds its partial representation of the emerged playtesting-based game dynamics

$$\check{x}_{M\alpha_c} \mapsto \check{x}_{D\alpha_c}.$$

The cost function used by the search algorithms is the function proposed and implemented in the work of Lim and Harrell (2014) as a linear combination of two heuristics: the manhattan distance between win condition objects (i.e., game objects appearing in win conditions), and the manhattan distance between the 'player' object and win condition objects.

[4]The produced game dynamics design concept is partial since c does not derive values of the 'gameplay scenario' design dimensions.

The function is computed at each game state change and, according to its authors, characterizes game states where objects used in win conditions are close together, and game states where the player is closer to objects used in win conditions, intuitively meaning that favored game states are those with shorter distances between win condition objects and the ‘player’ object.

The search algorithms are enhanced with a series of callback functions that are executed at key stages of the search algorithm process: a) at the beginning of an iteration, b) after a search node is expanded, c) after nodes are placed into the OPEN list, and d) at the end of an iteration. These callback points assist the process of deriving actual values of the *branching factor*, *move effort*, and *response time* game dynamics design dimensions by accumulating/averaging their values. Both *duration* and *paths* design dimensions are derived directly from the search algorithm output (i.e., the list of moves representing the solution for the game). Finally, the *board coverage* design dimension value is derived by executing the steps or moves of the solution on the game board. Final values of the playtesting-based game dynamics design dimensions are computed by averaging the four derived sets of values. If a search algorithm does not find a solution for a game within 1000 iterations, the game is considered unsolvable (i.e., an infinite number of moves) and the algorithm is discarded for final values computation.

Partial results preparation (c): c selects its partially derived dynamics representation

$$\check{x}_{Da_c} \rightleftarrows \vec{x}_{Da_c}.$$

Partial GUI update (c): c updates the game dynamics panel’s actual-dynamics related widgets to reflect the partial derived game dynamics

$$\vec{x}_{Da_c} \rightarrow \hat{x}_{Da_c}.$$

The analyzer agent c externalizes its actual game dynamics design concept into the dynamics panel by setting the actual value for each dimension of its (partial) concept in the same (shared) widgets employed by the human designer h to set the expected values (Fig. 6.22).

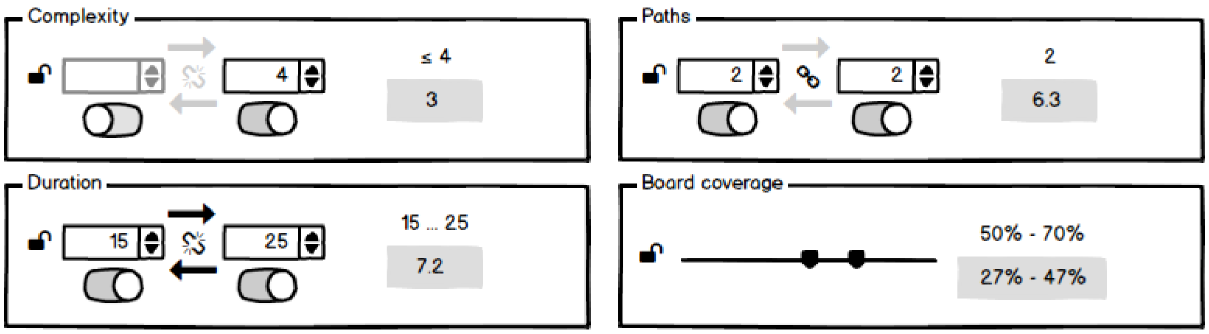


Figure 6.22: Game dynamics panel - setting the actual value of computer-analyzed design dimensions. Actual numerical values computed by the c agent are displayed below the expected values and are visually differentiated from the expected values (in this figure, actual values are displayed over a gray background).

Partial analysis finish notification (c): c removes the *analyzing* looping progress bar from the tool’s GUI to notify that analytical derivation has been completed

$$\check{x}_{Da_c} \dashrightarrow \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

Actual design interpretation and partial analysis (h) : Human playtesting is performed through a perception/action loop in which the player interprets the current state of the game expressed through the game graphics (external representation), plans the next movement or action, and performs it via a physical interface (e.g., game control, keyboard, touch screen). The game modifies its state in response to the player's action, and h interprets the new video game state. This cycle is repeated until reaching an end condition, or when the player abandons the game[5].

From the perspective of our designing framework, each time the player interprets the new state of the video game, a design concept is developed in her/his mind representing the game dynamics that emerge in terms of 'gameplay scenario' design dimensions. For simplicity, we represent the playtesting loop with two operations, the interpretation of the playable game, and the derivation of dynamics from the interpreted game.

$$\hat{x}_{M\alpha_c} \mathrel{\underrightarrow{\overrightarrow{\subset}}} \check{x}_{M\alpha_h^c},\ \check{x}_{M\alpha_h^c} \mapsto \check{x}_{Da_h}.$$

Partial results preparation (h): h focuses on her/his partially derived dynamics representation

$$\check{x}_{Da_h} \rightleftarrows \vec{x}_{Da_h}.$$

Partial GUI update (h): h updates the game dynamics panel's actual-dynamics related widgets to reflect human's dynamics representation

$$\vec{x}_{Da_h} \rightarrow \hat{x}_{Da_h}.$$

The tool's GUI allows a human analyzer h (i.e., the player) externalizing her/his actual 'gameplay scenario' game dynamics in the game dynamics panel. For each 'gameplay scenario' design dimension, h sets the actual value by moving a second double thumb horizontal slider located below the one used in formulation, reformulation or synthesis (Fig. 6.23). As depicted in (Fig. 6.19), once

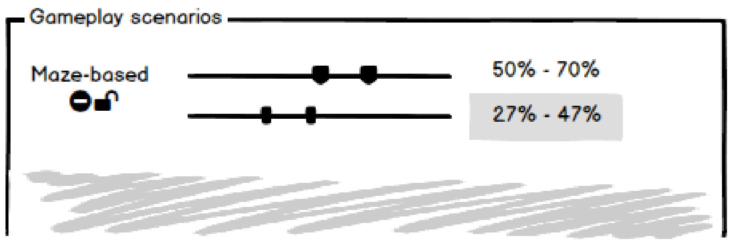


Figure 6.23: Game dynamics panel - setting the actual value of a 'gameplay scenario' design dimension. As in the 'expected-dynamics' related slider, the 'actual-dynamics' slider allows setting a specific value or a range region. The game analyzer may even set a particular value of 0 to express that the associated gameplay scenario was not perceived by the player during playtesting. Both 'expected-experience' and 'actual-experience' sliders are also visually distinguished (in this figure they have different thumbs shapes and styles applied). Actual values are displayed with a different visual style and below the expected values.

the playable versions of the game for h and c agents become available, the process is forked into two parallel processes to complete and externalize the analysis of the game in terms of dynamics

[5]It is worthy of mentioning that the game engine also works as a type of perception/action loop. The game engine interprets the player's actions, modifies the state of the game, and externalizes the new state.

dimensions. After that, the processes are joined, resulting in the dynamics panel displaying a complete expected and actual game dynamics representation (Fig. 6.24). The tool's GUI allows the agents to analyze the game independently (i.e., an agent does not need to wait for the other agent).

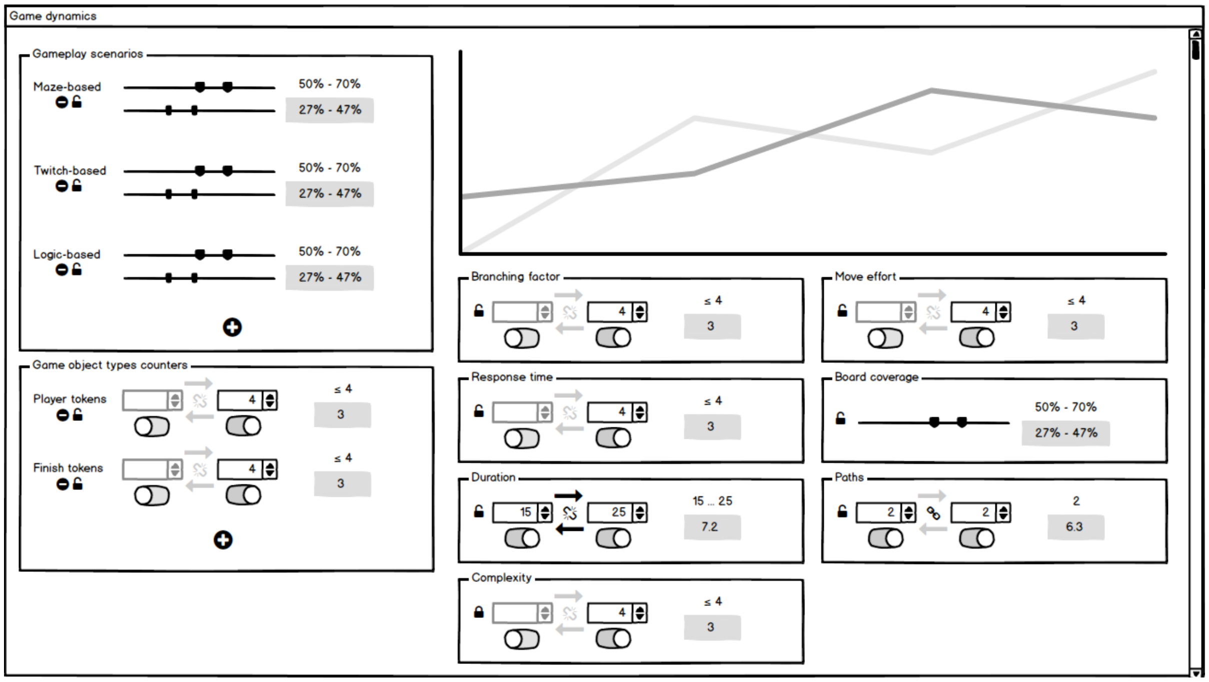


Figure 6.24: Game dynamics panel - game dynamics representation after analyzing the mechanics of a game. Both human-analyzed and computer-analyzed actual values are displayed along with the expected values.

6.1.3.2 Game dynamics evaluation

$$\left\{\left\{\hat{x}_{De_h}, \hat{x}_{Da_h}\right\} \subseteqq \left\{\check{x}_{De_c^h}, \check{x}_{Da_c^h}\right\}, \left\{\check{x}_{Da_c^h}, \check{x}_{Da_c}\right\} \Rrightarrow \check{x}_{Da_c^{ch}}, \check{x}_{De_c^h} \rightleftarrows \vec{x}_{De_c^h}, \check{x}_{Dv_c} \rightleftarrows \vec{x}_{Dv_c}\right\} \subseteq \mathscr{P}_{DM_c},$$

$$\left\{\vec{x}_{Dv_c} \rightarrow \hat{x}_{Dv_c}\right\} \subseteq \mathscr{T}_{DM_c}, \left\{\check{x}_{Da_c^{ch}} \leftrightarrow \vec{x}_{De_c^h},\right\} \subseteq \mathring{\mathscr{E}}_{DM_c}.$$

Expected & actual design interpretation: c interprets both expected and partial actual human-sourced game dynamics from the dynamics panel

$$\left\{\hat{x}_{De_h}, \hat{x}_{Da_h}\right\} \subseteqq \left\{\check{x}_{De_c^h}, \check{x}_{Da_c^h}\right\}.$$

Actual design augmentation: c augments the interpreted version of the partial actual human-sourced game dynamics with its own partial actual game dynamics

$$\left\{\check{x}_{Da_c^h}, \check{x}_{Da_c}\right\} \Rrightarrow \check{x}_{Da_c^{ch}}.$$

Assessment data preparation: c selects the human-sourced expected game dynamics

$$\check{x}_{De^{h}_{c}} \rightleftarrows \vec{x}_{De^{h}_{c}}.$$

Design assessment: c compares actual game dynamics to the human-sourced expected game dynamics by computing the distances associated with the game dynamics design dimensions' metrics

$$\check{x}_{Dv_{c}} = \left(\check{x}_{Da^{ch}_{c}} \leftrightarrow \vec{x}_{De^{h}_{c}}\right).$$

Results preparation: c selects the computed game dynamics' distances (i.e., game dynamics evaluation concept)

$$\check{x}_{Dv_{c}} \rightleftarrows \vec{x}_{Dv_{c}}.$$

GUI update: c reflects the distances between expected and actual game dynamics design concepts in the game dynamics panel

$$\vec{x}_{Dv_{c}} \rightarrow \hat{x}_{Dv_{c}}.$$

To help the human designer interpreting the similarity or dissimilarity between her/his expectations and the actual design, the game dynamics panel of the tool's GUI provide a visual representation of the distances between expected and actual game dynamics design concepts (Figs. 6.25 and 6.26).

6.1.3.3 Game dynamics analysis

$$\left\{ \left\{\hat{x}_{Da_{h}}, \hat{x}_{Da_{c}}\right\} \mathrel{\subseteq\!\!\!\rightrightarrows} \left\{\check{x}_{Da_{h}}, \check{x}_{Da^{c}_{h}}\right\}, \left\{\check{x}_{Da_{h}}, \check{x}_{Da^{c}_{h}}\right\} \mathrel{\overset{\dashrightarrow}{\dashrightarrow}} \check{x}_{Da^{ch}_{h}}, \check{x}_{Pa_{h}} \rightleftarrows \vec{x}_{Pa_{h}} \right\} \subseteq \mathscr{P}_{PD_{h}},$$

$$\left\{\vec{x}_{Pa_{h}} \rightarrow \hat{x}_{Pa_{h}}\right\} \subseteq \mathscr{T}_{PD_{h}}, \left\{\check{x}_{Da^{ch}_{h}} \mapsto \check{x}_{Pa_{h}}\right\} \subseteq \mathring{\mathscr{E}}_{PD_{h}}.$$

Actual design interpretation: c interprets both human-sourced and computer-sourced external representations of the actual game dynamics from the game dynamics panel's widgets

$$\left\{\hat{x}_{Da_{h}}, \hat{x}_{Da_{c}}\right\} \mathrel{\subseteq\!\!\!\rightrightarrows} \left\{\check{x}_{Da_{h}}, \check{x}_{Da^{c}_{h}}\right\}.$$

Actual design combination: h merges the partial interpreted versions of the actual game dynamics

$$\left\{\check{x}_{Da_{h}}, \check{x}_{Da^{c}_{h}}\right\} \mathrel{\overset{\dashrightarrow}{\dashrightarrow}} \check{x}_{Da^{ch}_{h}}.$$

Actual design analysis: From her/his interpretation of the actual game dynamics, h builds-up a mental game experience design concept representing the evoked playfulness when playing the game

$$\check{x}_{Da^{ch}_{h}} \mapsto \check{x}_{Pa_{h}}.$$

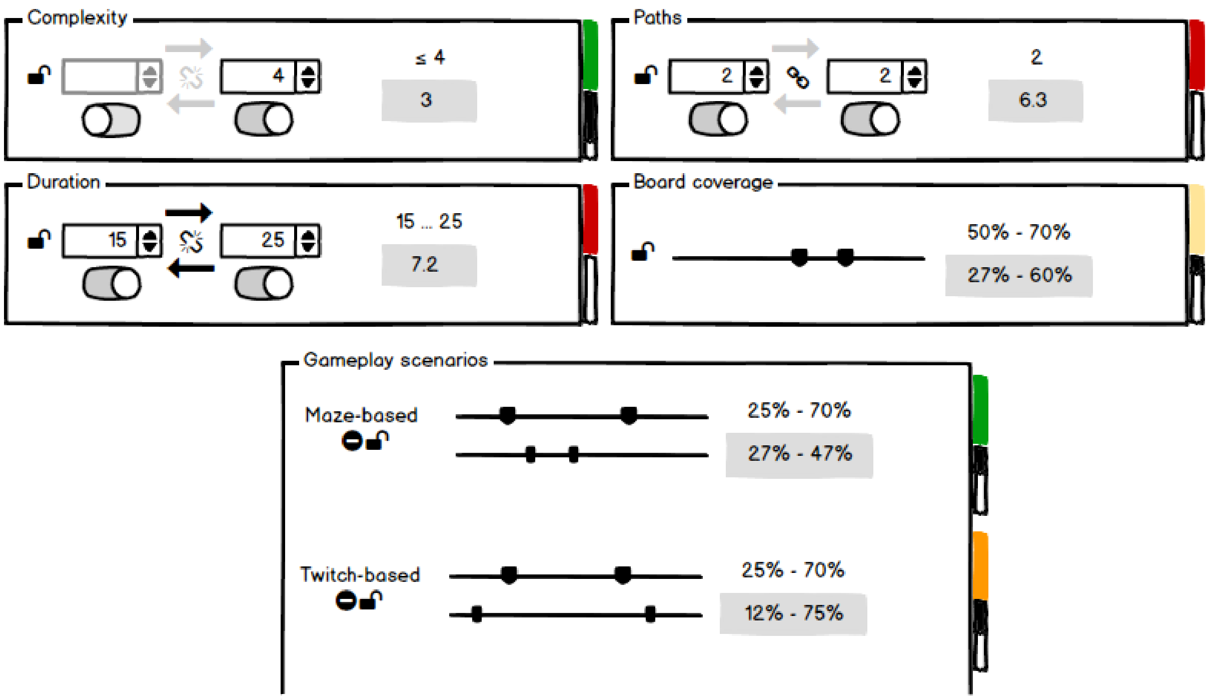


Figure 6.25: Game dynamics panel - evaluation of expected vs actual design concepts (detailed view). The distances between expected and actual game dynamics design concepts are represented with two small stacked bars at the very right of each of the game dynamics design dimensions. The top bar color represents one of the four categories defined in Eq. (5.2): in-range (green), range partially exceeded (yellow), range exceeded (orange), and out of range (red); and the fill length of the bottom bar represents the similarity defined in Eq. (5.3). No bottom bar (i.e., fill length = 0) represents 0% similarity, while a full filled bottom bar represents 100% similarity.

Results preparation: h focuses on her/his evoked game experience

$$\check{x}_{Pa_h} \rightleftarrows \vec{x}_{Pa_h}.$$

GUI update: h manipulates the game experience panel's widgets to reflect her/his evoked game experience

$$\vec{x}_{Pa_h} \rightarrow \hat{x}_{Pa_h}.$$

The tool's GUI allows a human agent h (i.e., the player) to externalize her/his actual game experience in the same panel where the game experience design space was configured through formulation. For each PLEX card representing a game experience design dimension, h sets the actual intensity of the evoked experience (if evoked) by moving a second double thumb vertical slider located at the right of the one used in formulation or reformulation (Fig. 6.27). h analyzer can also set the intensity value through the PLEXQ mini-survey.

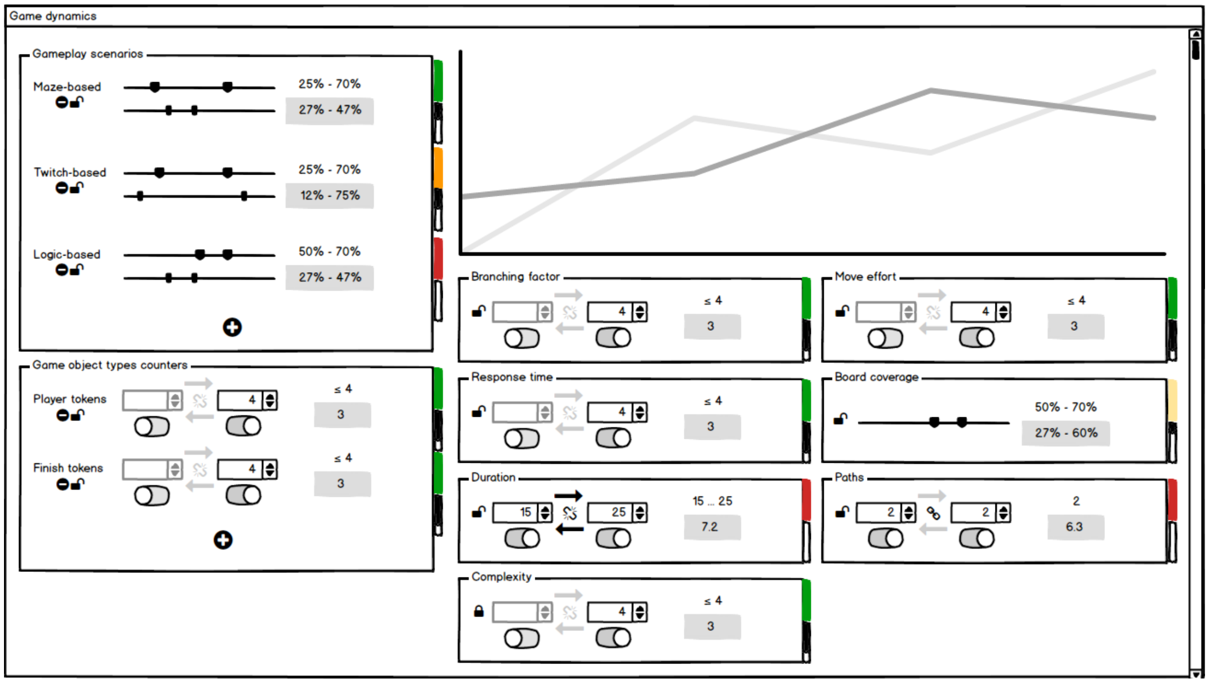


Figure 6.26: Game dynamics panel - evaluation of expected vs actual design concepts (complete view).

6.1.3.4 Game experience evaluation

$$\left\{\left\{\hat{x}_{Pe_h}, \hat{x}_{Pa_h}\right\} \subseteqq \left\{\check{x}_{Pe_c}^{h}, \check{x}_{Pa_c}^{h}\right\}, \check{x}_{Pe_c}^{h} \rightleftarrows \vec{x}_{Pe_c}^{h}, \check{x}_{Pv_c} \rightleftarrows \vec{x}_{Pv_c}\right\} \subseteq \mathscr{P}_{PD_c},$$

$$\left\{\vec{x}_{Pv_c} \rightarrow \hat{x}_{Pv_c}\right\} \subseteq \mathscr{T}_{PD_c}, \left\{\check{x}_{Pa_c}^{h} \leftrightarrow \vec{x}_{Pe_c}^{h},\right\} \subseteq \mathring{\mathscr{E}}_{PD_c}.$$

Expected & actual design interpretation: c interprets both actual and expected human-sourced game experience from the game experience panel

$$\left\{\hat{x}_{Pe_h}, \hat{x}_{Pa_h}\right\} \subseteqq \left\{\check{x}_{Pe_c}^{h}, \check{x}_{Pa_c}^{h}\right\}.$$

Assessment preparation: c selects the expected game experience

$$\check{x}_{Pe_c}^{h} \rightleftarrows \vec{x}_{Pe_c}^{h}.$$

Design assessment: c compares actual game experience to expected game experience by computing the distances associated with the game experience design dimensions' metrics

$$\check{x}_{Pv_c} = \left(\check{x}_{Pa_c}^{h} \leftrightarrow \vec{x}_{Pe_c}^{h}\right).$$

Results preparation: c selects the computed game experiences' distances

$$\check{x}_{Pv_c} \rightleftarrows \vec{x}_{Pv_c}.$$

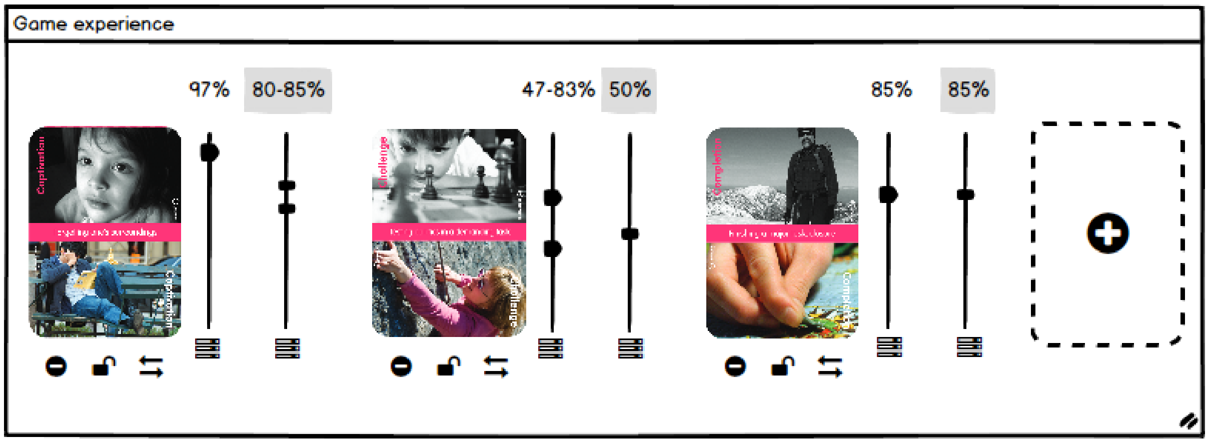


Figure 6.27: Game experience panel - setting the actual game experience. As in the 'expected-experience' related slider, the 'actual-experience' slider allows setting a specific intensity value or a range. The analyzer may even set a particular value of 0 to express that the associated playful experience was not evoked at all.

GUI update: c reflects the distances between expected and actual game experience design concepts in the game experience panel

$$\vec{x}_{Pv_c} \rightarrow \hat{x}_{Pv_c}.$$

To help the human designer interpreting the similarity or dissimilarity between her/his game experience expectations and the evoked game experience, the game experience panel of the tool's GUI provide a visual representation of the distances between expected and actual game dynamics design concepts (Fig. 6.28)

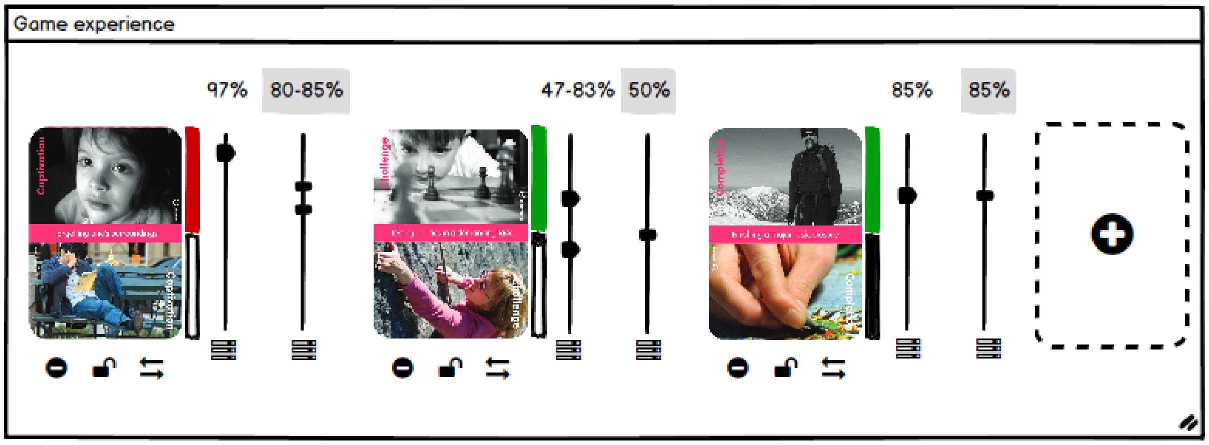


Figure 6.28: Game experience panel - evaluation. The distance between the expected and actual game experience is represented using the same mechanism as in the game dynamics panel. Each dimension (PLEX card) is decorated with a pair of stacked bars on the right side, indicating both its categorical rating and similarity according to the metric defined in Eq. (5.1).

6.1.4 Reformulation

In this system, the formulation processes at the game experience and game dynamics levels share the same pattern of steps which is illustrated in (Fig. 6.29).

6.1.4.1 Game experience reformulation

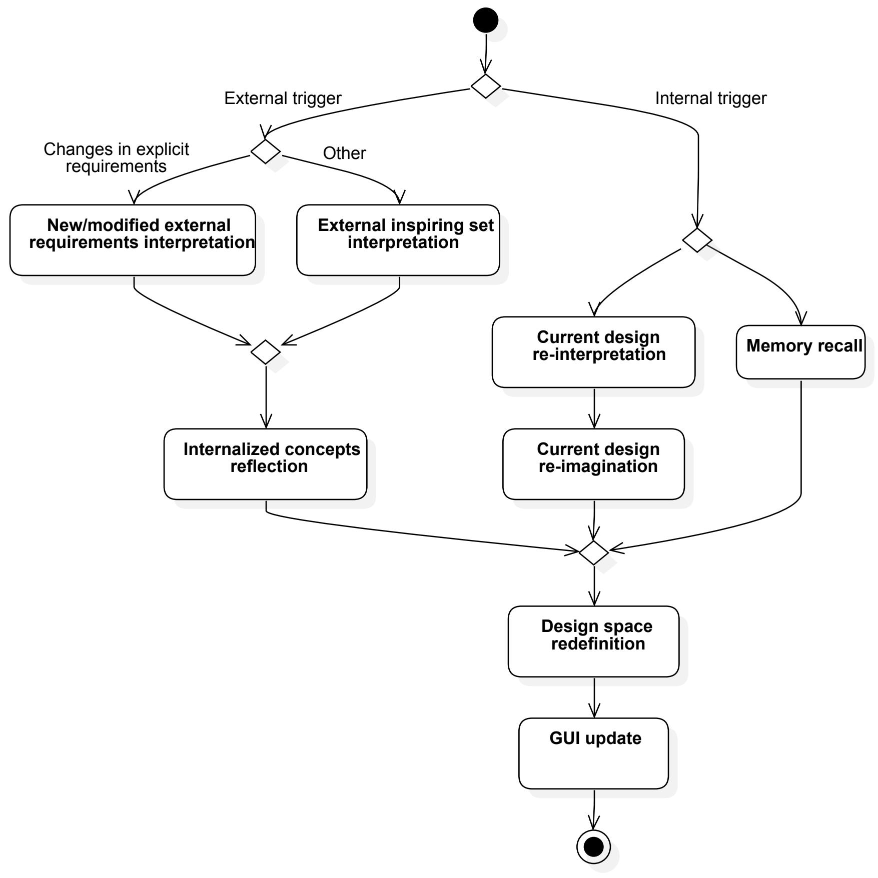

External trigger
Internal trigger
Changes in explicit requirements
Other
New/modified external requirements interpretation
External inspiring set interpretation
Current design re-interpretation
Memory recall
Internalized concepts reflection
Current design re-imagination
Design space redefinition
GUI update

Figure 6.29: An activity diagram representing the overall reformulation process pattern implemented in the game experience ***P*** and game dynamics ***D*** levels.

$$\Big\{ \hat{x}_{Pr} \mathrel{\subseteq\!\!\!\rightarrow} \check{x}_{P_h},\ \hat{x}_{Pn} \mathrel{\subseteq\!\!\!\rightarrow} \check{x}_{P_h},\ \left\{\hat{x}_{Pa_h},\ \hat{x}_{Pe_h},\ \hat{x}_{Pv_c}\right\} \mathrel{\subseteq\!\!\!\rightarrow} \left\{\check{x}_{Pa_h},\ \check{x}_{Pe_h},\ \check{x}_{Pv^c_h}\right\},$$

$$\check{x}_{P_h} \mathrel{\subseteq\!\!\!\dashrightarrow} \check{x}_{Pe_h},\ \left\{\check{x}_{Pa_h},\ \check{x}_{Pe_h},\ \check{x}_{Pv^c_h}\right\} \mathrel{\subseteq\!\!\!\dashrightarrow} \check{x}_{Pe_h},\ \check{x}_{Pm_h} \mathrel{\subseteq\!\!\!\dashrightarrow} \check{x}_{Pe_h},\ \check{x}_{Pe_h} \rightleftarrows \vec{x}_{Pe_h} \Big\} \subseteq \mathscr{P}_{PD_h},$$

$$\left\{ \vec{x}_{Pe_h} \rightarrow \hat{x}_{Pe_h} \right\} \subseteq \mathscr{T}_{PD_h}.$$

New/modified external requirements interpretation: h interprets new or modified external game experience expectations

$$\hat{X}_{Pr} \mathrel{\subseteq\!\!\!\!\rightarrow} \check{x}_{P_h}.$$

External inspiring set interpretation: At any time, h interprets a game experience design concept from an external inspiring set

$$\hat{X}_{Pn} \mathrel{\subseteq\!\!\!\!\rightarrow} \check{x}_{P_h}.$$

Internalized concepts reflection: Through reflection, h accommodates and augments the interpreted version of the externally defined game experience, attempts to model it (mentally) in terms of a possibly different set of PLEX playful experiences categories (different from the current design space) or intensities

$$\check{x}_{P_h} \mathrel{\subseteq\!\!\!\!\dashrightarrow} \check{x}_{Pe_h}.$$

Current design re-interpretation: At any time, h interprets the current actual and expected game experiences from the game experience panel and the distances computed by c between expected and actual game experiences (reported evaluation) [6]

$$\left\{\hat{x}_{Pa_h},\ \hat{x}_{Pe_h},\ \hat{x}_{Pv_c}\right\} \mathrel{\subseteq\!\!\!\!\rightarrow} \left\{\check{x}_{Pa_h},\ \check{x}_{Pe_h},\ \check{x}_{Pv^c_h}\right\}.$$

Current design re-imagination: Through reflection, h reimagines the game experience in terms of a possibly different set of PLEX playful experiences categories or intensities

$$\left\{\check{x}_{Pa_h},\ \check{x}_{Pe_h},\ \check{x}_{Pv^c_h}\right\} \mathrel{\subseteq\!\!\!\!\dashrightarrow} \check{x}_{Pe_h}.$$

Memory recall: By reflection, h may recall an appropriate game experience defined in terms of a possibly different set of PLEX playful experiences categories or intensities

$$\check{x}_{Pm_h} \mathrel{\subseteq\!\!\!\!\dashrightarrow} \check{x}_{Pe_h}.$$

Design space re-definition: h focuses on the new design dimensions and ranges used to model the desired game experience and removes the focus on discarded ones, updating her/his game experience design space

$$\check{x}_{Pe_h} \rightleftarrows \vec{x}_{Pe_h},\ \vec{x}_{Pe_h} = \mathcal{D}_{P_h}.$$

GUI update: h updates her/his game experience design space (i.e., the updated expected experience design concept) via the game experience panel

$$\vec{x}_{Pe_h} \rightarrow \hat{x}_{Pe_h}.$$

The designer follows a very similar GUI workflow as in the formulation process (See GUI update in Section 6.1.1.1). She/he may add new dimensions, replace dimensions, modify their intensities and, additionally, remove dimensions if required (See Fig. 6.4).

[6] A common designer's re-interpretation trigger would be having performed a reformulation on the design level below.

6.1.4.2 Game dynamics reformulation

$$\mathscr{P}_{PD_h} \supseteq \Big\{ \hat{x}_{Dr} \rightrightarrows \check{x}_{D_h},\ \hat{x}_{Dn} \rightrightarrows \check{x}_{D_h},\ \left\{\hat{x}_{Da_h},\ \hat{x}_{De_h},\ \hat{x}_{Dv_c}\right\} \rightrightarrows \left\{\check{x}_{Da_h},\ \check{x}_{De_h},\ \check{x}_{Dv^c_h}\right\},$$
$$\check{x}_{D_h} \dashrightarrow \check{x}_{De_h},\ \left\{\check{x}_{Da_h},\ \check{x}_{De_h},\ \check{x}_{Dv^c_h}\right\} \dashrightarrow \check{x}_{De_h},\ \check{x}_{Dm_h} \dashrightarrow \check{x}_{De_h},\ \check{x}_{De_h} \rightleftarrows \vec{x}_{De_h} \Big\} \subseteq \mathscr{P}_{DM_h},$$

$$\mathscr{T}_{PD_h} \supseteq \left\{ \vec{x}_{De_h} \to \hat{x}_{De_h} \right\} \subseteq \mathscr{T}_{DM_h}.$$

New/modified external requirements interpretation: h interprets new or modified external game dynamics constraints and requirements

$$\hat{X}_{Dr} \rightrightarrows \check{x}_{D_h}.$$

External inspiring set interpretation: At any time, h interprets a game dynamics design concept from an external inspiring set

$$\hat{X}_{Dn} \rightrightarrows \check{x}_{D_h}.$$

Internalized concepts reflection: Through reflection, h accommodates and augments the interpreted version of the externally defined game dynamics constraints. They attempt to model it in terms of different dimensions' ranges or a possibly different set of 'gameplay scenario' and 'game object type counter' types of dimensions

$$\check{x}_{D_h} \dashrightarrow \check{x}_{De_h}.$$

Current design re-interpretation: At any time, h interprets the current actual and expected game dynamics from the game dynamics panel and the distances computed by c between expected and actual game dynamics (reported evaluation)

$$\left\{\hat{x}_{Da_h},\ \hat{x}_{De_h},\ \hat{x}_{Dv_c}\right\} \rightrightarrows \left\{\check{x}_{Da_h},\ \check{x}_{De_h},\ \check{x}_{Dv^c_h}\right\}.$$

Current design re-imagination: Through reflection, h reimagines the game dynamics design concept in terms of different dimensions' ranges or a possibly different set of 'gameplay scenario' and 'game object type counter' types of dimensions

$$\left\{\check{x}_{Da_h},\ \check{x}_{De_h},\ \check{x}_{Dv^c_h}\right\} \dashrightarrow \check{x}_{De_h}.$$

Memory recall: By reflection, h may recall an appropriate game dynamics defined in terms of different dimensions' ranges or a possibly different set of 'gameplay scenario' and 'game object type counter' types of dimensions

$$\check{x}_{Dm_h} \dashrightarrow \check{x}_{De_h}.$$

Design space re-definition: h focuses on the new design dimensions and ranges used to model the desired game dynamics and removes the focus on discarded ones, updating her/his game dynamics design space

$$\check{x}_{De_h} \rightleftarrows \vec{x}_{De_h},\ \vec{x}_{De_h} = \mathcal{D}_{D_h}.$$

GUI update: h updates her/his game dynamics design space via the game dynamics panel

$$\vec{x}_{De_h} \rightarrow \hat{x}_{De_h}.$$

The designer follows a very similar GUI workflow as in the formulation process (See GUI update in Section 6.1.1.2); that is, they may add new ´gameplay scenario' or ´game object type counter' dimensions, modify ranges of current dimensions and, additionally, remove ´gameplay scenario' or ´game object type counter' dimensions if required (See Figs. 6.8 and 6.9).

6.1.4.3 Game mechanics reformulation

This process incorporates a broader set of input sources or triggers for reformulation (e.g., existing games, game design tutorials, the designs produced by the autonomous CCDS) (Fig. 6.30).

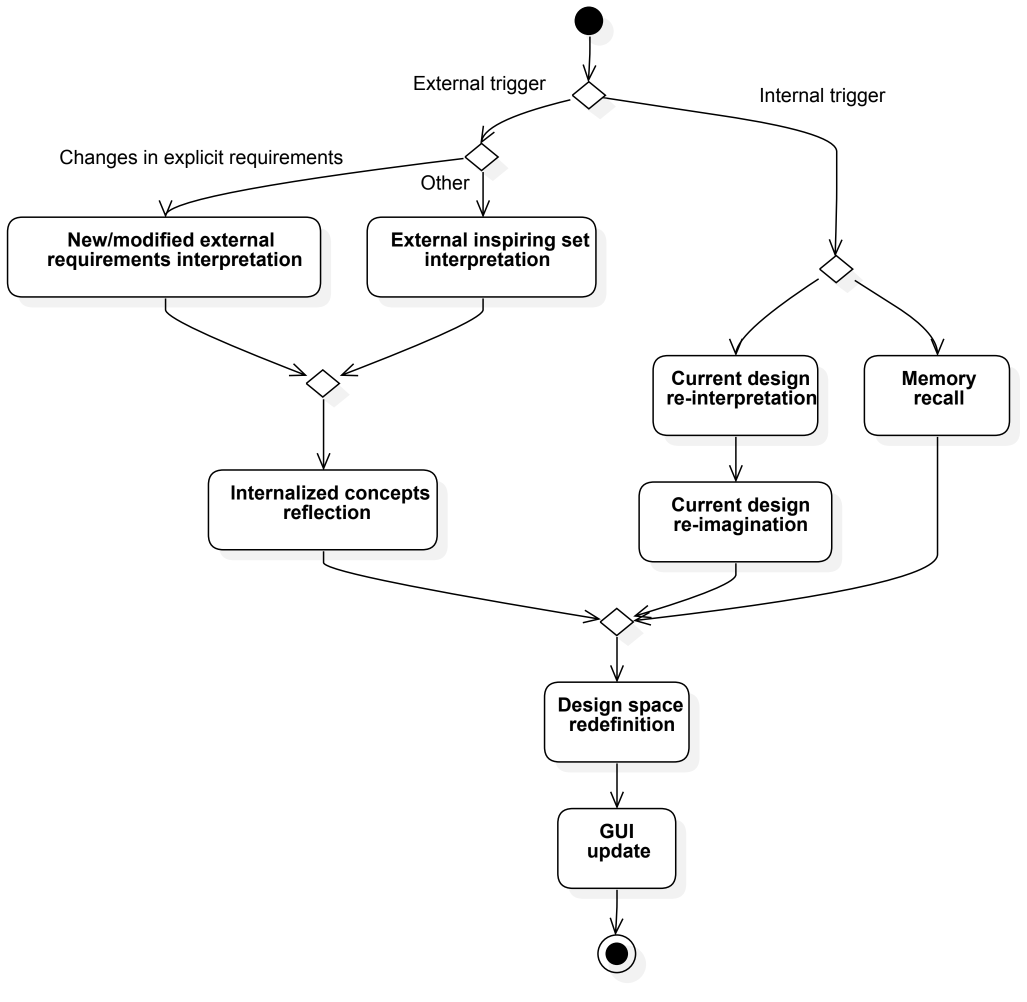

External trigger
Internal trigger
Changes in explicit requirements
Other
New/modified external requirements interpretation
External inspiring set interpretation
Current design re-interpretation
Memory recall
Internalized concepts reflection
Current design re-imagination
Design space redefinition
GUI update

Figure 6.30: An activity diagram representing the overall reformulation process pattern implemented in the game mechanics M level.

$$\Big\{\hat{x}_{M^r} \subsetneq \check{x}_{M_h}, \hat{x}_{M^n} \subsetneq \check{x}_{M_h}, \hat{x}_{M^a} \subsetneq \check{x}_{M^a_h},$$

$$\check{x}_{M_h} \dashrightarrow \check{x}_{M^a_h}, \check{x}_{M^a_h} \dashrightarrow \check{x}_{M^a_h}, \check{x}_{M^m_h} \dashrightarrow \check{x}_{M^a_h}, \check{x}_{M^a_h} \rightleftarrows \vec{x}_{M^a_h}\Big\} \subseteq \mathscr{P}_{DM_h},$$

$$\Big\{\vec{x}_{M^a_h} \rightarrow \hat{x}_{M^a}\Big\} \subseteq \mathscr{T}_{DM_h}.$$

New/modified external requirements interpretation: h interprets new or modified external game constraints

$$\hat{X}_{M^r} \subsetneq \check{x}_{M_h}.$$

External inspiring set interpretation: At any time, h interprets a game design concept from an external inspiring set (e.g., a design generated by the autonomous CCDS)

$$\hat{X}_{M^n} \subsetneq \check{x}_{M_h}.$$

Internalized concepts reflection: Through reflection, h accommodates and augments the interpreted version of the externally defined game mechanics constraints and attempts to model it in terms of different dimensions' ranges (e.g., different/modified layers/aliases, number and structure of game rules and win conditions) or dimensions (e.g., a different game board shape and size)

$$\check{x}_{M_h} \dashrightarrow \check{x}_{M^a_h}.$$

Current design re-interpretation: At any time, h may interprets the game editor's current game

$$\hat{x}_{M^a} \subsetneq \check{x}_{M^a_h}.$$

Current design re-imagination: Through reflection, h reimagines the game in terms of different dimensions or ranges

$$\check{x}_{M^a_h} \dashrightarrow \check{x}_{M^a_h}.$$

Memory recall: By reflection, h may recall a game defined in terms of different dimensions or ranges

$$\check{x}_{M^m_h} \dashrightarrow \check{x}_{M^a_h}.$$

Design space re-definition: h focuses on the new design dimensions and ranges used to model the desired game and removes the focus on discarded ones, updating her/his game mechanics design space

$$\check{x}_{M^a_h} \rightleftarrows \vec{x}_{M^a_h}, \vec{x}_{M^a_h} = \mathcal{D}_{M_h}.$$

GUI update: h updates her/his game mechanics design space via the game editor panel

$$\vec{x}_{M^a_h} \rightarrow \hat{x}_{M^a}.$$

The designer follows a very similar GUI workflow as in the formulation process (See GUI update in Sections 6.1.1.3 and 6.1.2.2). That is, they may set a new game board shape and size, modify the layers and aliases, and modify the number and structure of game rules and win conditions (See Figs. 6.10 to 6.13, 6.15, 6.17 and 6.18).

6.2 Evaluation

At this stage of the collaborative CCDS development, only the evaluation of the user interface (GUI) is considered. Incorporating an evaluation of either the computational agent's creative capacity (perhaps when the agent acquires greater responsibilities in the creative process) or of the creative process itself is pending.

6.2.1 GUI evaluation

With the collaboration of a professional user interface (UI)/UX/IxD designer, an evaluation tool was defined based on the ten heuristics (Table 6.1) proposed by Jakob Nielsen (1994, 2020) for assessing usability, interaction, and user experience. These heuristics has been developed and improved since 1990 within the field of HCI as general principles or broad rules (rather than specific usability guidelines) for interaction design.

Table 6.1: The ten Nielsen's interaction design heuristics as published in (ibid.).

Visibility of system status	Designs should keep users informed about what is going on, through appropriate, timely feedback.
Match between system and the real world	The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order.
User control and freedom	Users often perform actions by mistake. They need a clearly marked ”emergency exit” to leave the unwanted action without having to go through an extended process.
Consistency and standards	Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions.
Error prevention	Good error messages are important, but the best designs carefully prevent problems from occurring in the first place. Either eliminate error-prone conditions, or check for them and present users with a confirmation option before they commit to the action.
Recognition rather than recall	Minimize the user's memory load by making elements, actions, and options visible. The user should not have to remember information from one part of the interface to another. Information required to use the design (e.g. field labels or menu items) should be visible or easily retrievable when needed.
Flexibility and efficiency of use	Shortcuts — hidden from novice users — may speed up the interaction for the expert user such that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

Aesthetic and minimalist design	Interfaces should not contain information which is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.
Help users recognize, diagnose, and recover from errors	Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.
Help and documentation	It is best if the system doesn't need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.

The concrete elements of each assessment item were formulated during an interview with the UI/UX designer to discuss the system's purpose. Then, after defining the elements of each evaluation item, an external evaluation of the system's GUI was carried out in the actual tool's GUI (Figs. 6.31 to 6.34).

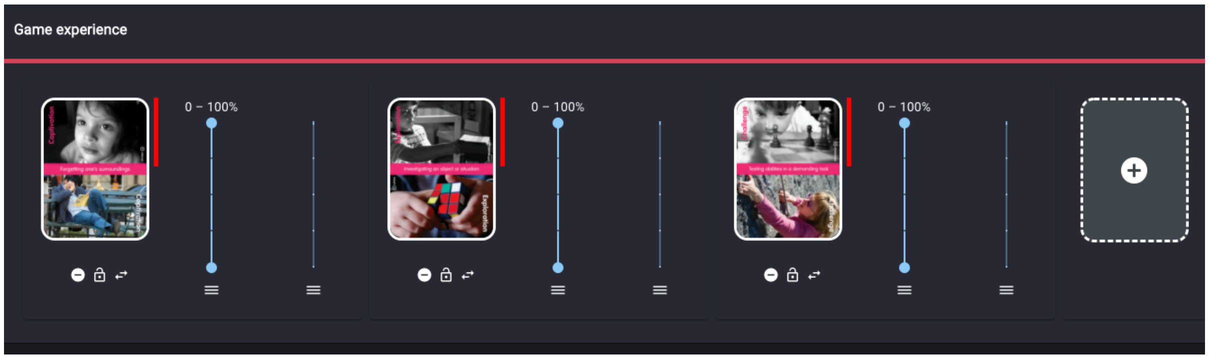


Figure 6.31: Collaborative CCDS with computational 'assistant' - actual GUI game experience panel screenshot.

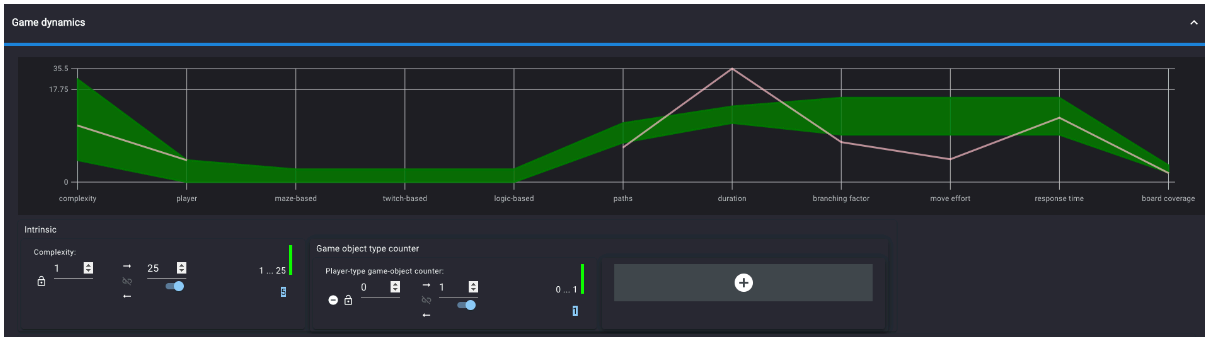


Figure 6.32: Collaborative CCDS with computational 'assistant' - actual GUI game dynamics panel screenshot depicting the automatic plot to compare expected with actual game dynamics design concepts.

Each item is evaluated on a 0 to 2 scale, where 0 indicates that the system does not comply with the minimum necessary, 1 indicates that the system complies with the minimum necessary,

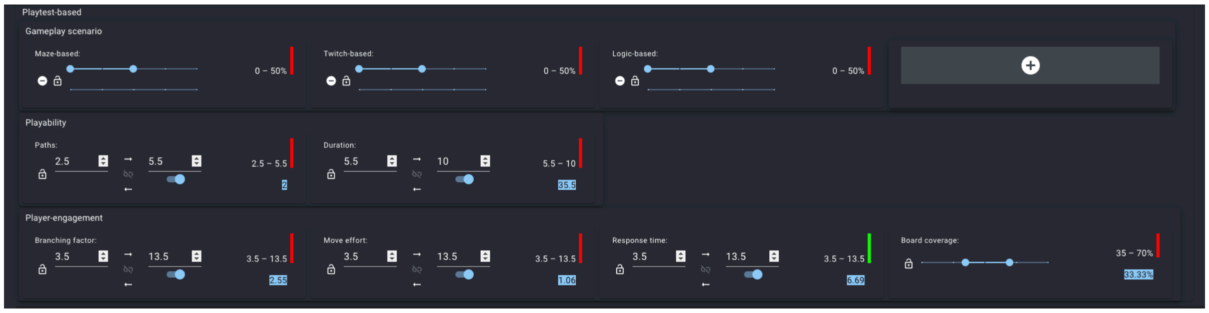


Figure 6.33: Collaborative CCDS with computational ‘assistant’ - actual GUI game dynamics panel screenshot (2).

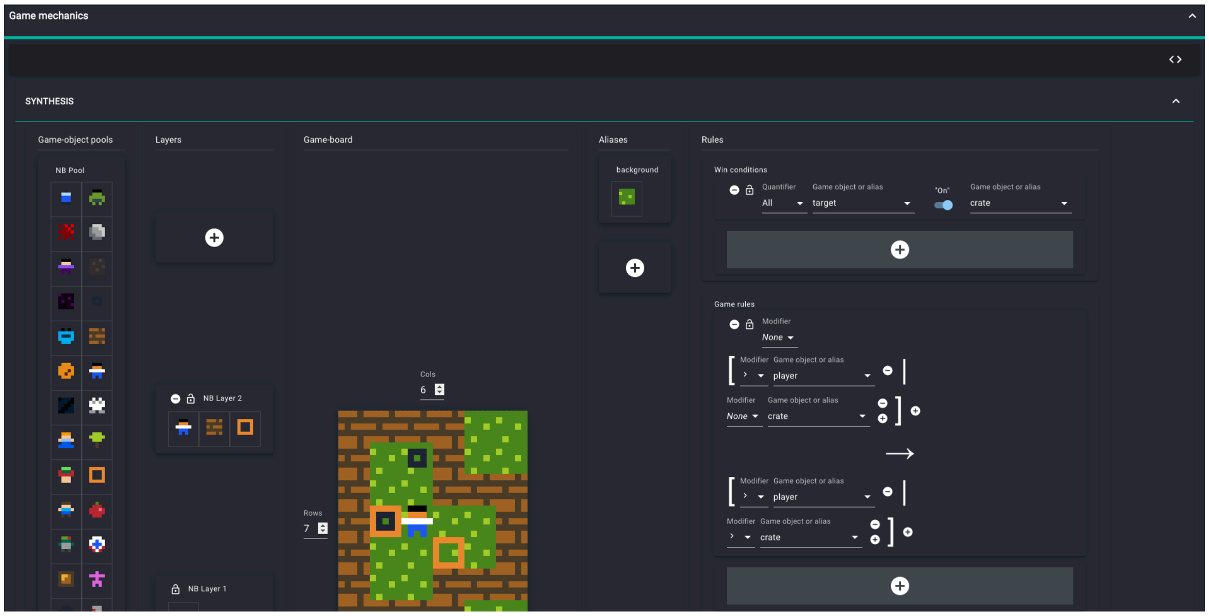


Figure 6.34: Collaborative CCDS with computational ‘assistant’ - actual GUI game mechanics panel screenshot.

and 2 indicates that the system satisfactorily complies with the evaluated item. Once each element has been evaluated, the global evaluation is computed to assess the overall compliance with the heuristics Fig. 6.35 and Table 6.2).

Table 6.2: Collaborative CCDS with computational ‘assistant’ - GUI detailed evaluation.

Heuristic/item	Value	%
Visibility of system status		
The system provides textual, graphical, auditory or sensory (vibration) prompts when the system requires time to process information before displaying it.	1	

Continued on next page

Table 6.2 – *Continued from previous page*

Heuristic/item	Value	%
Interactive elements (e.g., buttons, inputs, controls) provide feedback to the user when the user manipulates them.	2	
Feedback from interactive elements (e.g., buttons, inputs, controls) is clear to the user.	2	
The system uses textual, graphical, auditory, or sensory (vibrations) warnings whether the tasks have been performed successfully or not.	1	
Subtotal	6	75.00%
Match between system and real world		
The system has a language or terminology that is familiar to the user.	1	
There is a hierarchy in the information presented (e.g., title, subtitle, text box, images, icons).	2	
The icons used are clear to the user.	1	
Icons that are not clear to the user have descriptive support labels.	2	
The labels of interactive elements correspond with the functions they perform or request.	2	
Field labels are descriptive concerning the information requested.	2	
The system uses metaphors for navigation or interaction elements (e.g., tabs, accordion, bookmarks).	1	
The transition between screens presents a logical sense of navigation (e.g., right-to-left, up-to-down, down-to-up, fade-in, fade-out, rotate).	2	
Subtotal	13	81.25%
User control and freedom		
When the user performs a task by mistake, the system presents undo and or redo options.	0	
When the user wants to edit data in a previously filled-in field, the system allows her/him to do it easily.	2	
The fields allow for easy entry of information.	1	
Subtotal	3	50.00%
Consistency and standards		
In multi-step tasks, the user is shown which step she/he is in and how many steps are left to complete the task.	2	
The native elements of each operating system or platform are used appropriately.	2	
Appropriate interaction elements and controls are used for each action.	2	

Continued on next page

Table 6.2 – *Continued from previous page*

Heuristic/item	Value	%
For interactive elements, the use of standard colors and styles (e.g., links, green for accept, red for cancel) is followed.	2	
The visual aspect and style of graphical elements and interactive elements are consistent throughout the system.	2	
The way of functioning of the interactive elements is the same throughout the system.	2	
The language and terminology are the same throughout the system.	2	
Subtotal	14	100.00%
Error prevention		
Tasks that remove or modify sensitive information request confirmation from the user before being executed.	1	
Validation systems are in place to prevent the user from submitting/entering erroneous information (e.g., checking form fields on the fly).	1	
The system prevents the user from entering invalid characters (e.g., numeric fields do not allow entering alphabetic or special characters).	2	
Subtotal	4	66.67%
Recognition rather than recall		
The affordance of interactive elements is evident.	2	
There is no affordance in non-interactive elements.	2	
Field labels are always visible.	2	
The main interaction elements have a predominant color in contrast with the rest of the interface elements.	2	
The main interaction elements are visible when they are required.	2	
In multi-step tasks, the system alleviates the user from having to remember decisions previously made.	1	
Subtotal	11	91.67%
Flexibility and efficiency of use		
The system is flexible enough that novice users can interact with it without any problem.	1	
The system has “accelerators” which allow frequent actions to be performed more quickly.	2	
The system has advanced options that are displayed on demand by advanced users.	2	
The user has customization options within the system.	1	
Subtotal	6	75.00%

Continued on next page

Table 6.2 – *Continued from previous page*

Heuristic/item	Value	%
Aesthetic and minimalism design		
The system avoids information overload, displaying only the necessary information for each section.	2	
The system has an interface free of visual noise, avoiding displaying decorative elements that do not support the message transmitted in each section.	2	
The system has a visual resting space between elements.	2	
The system uses different levels of detail to display the content.	2	
Subtotal	8	100.00%
Recognize, diagnose, and recover from errors		
When the system is faulty, it informs the user with textual or graphical messages.	2	
Error messages have a clear description of the problem and offer a possible solution.	1	
Fields requesting complex or unusual information have clear contextual help.	2	
The fields that the user does not fill in correctly show clear contextual help.	2	
Subtotal	7	87.50%
Help and documentation		
The system has a help button or link to information regarding the operation of the system.	0	
The help button or link is located in a visible and standard area.	0	
The help information is focused on the user's tasks. It is displayed as a list with a series of steps to be performed.	0	
The system has a mini-tour, which briefly explains the main elements of the system.	0	
Subtotal	0	0.00%
	Total	72.71%

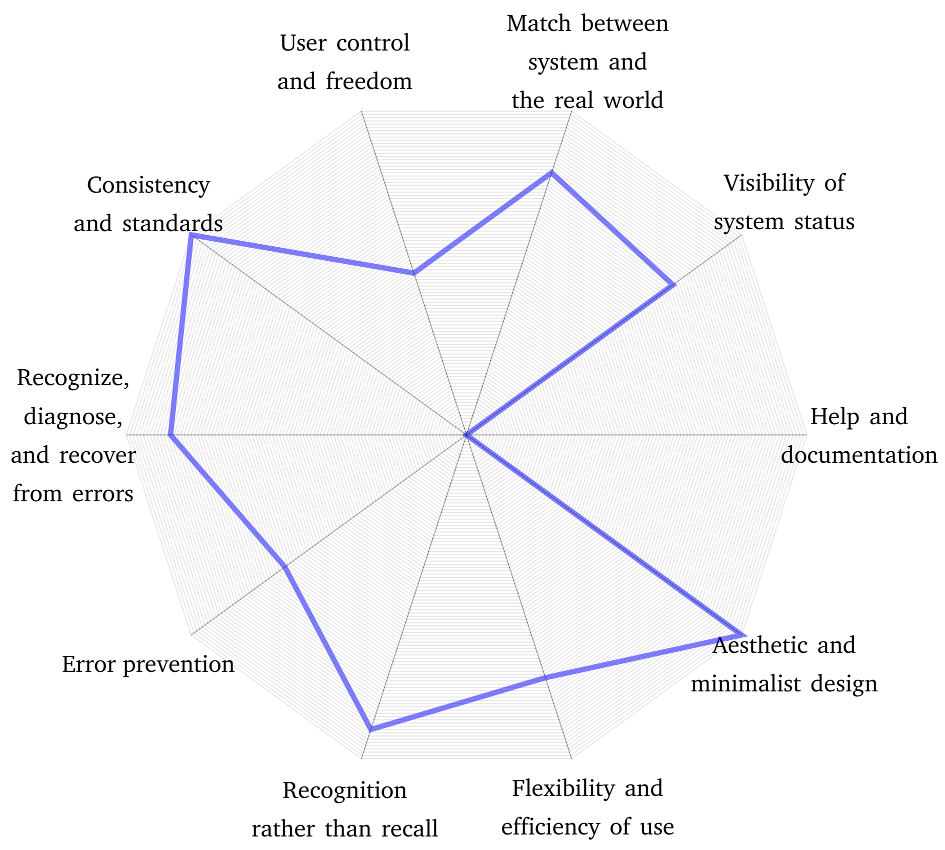


Figure 6.35: Collaborative CCDS with computational ‘assistant’ - GUI global evaluation.

7 Building a collaborative CCDS: Apprentice assistant computational agent

In this system version, the computational agent c keeps performing its previous roles and, additionally, starts learning how to perform specific processes based on the observation of human (h agent) performing (supervised learning). This version does not require major modifications to the tool's GUI.

The computational agent c should be able to:

- Learn the rules to produce expected game dynamics design concepts that the human designer would likely produce when trying to satisfy particular game experiences (the human designer PD synthesis encoded rationale). The agent should learn $X \mapsto Y$ rules, where X are game experience design concepts, and Y are game dynamics design concepts.
- Learn the rules to derive actual game experience design concepts from actual game dynamics design concepts (the encoded dynamics$\mapsto$experience mapping described in the MDA framework). The agent should learn $X \mapsto Y$ rules, where X are game dynamics design concepts, and Y are game experience design concepts.
- Learn the rules to produce games that the human designer would likely produce when trying to satisfy particular game dynamics (the human designer DM synthesis encoded rationale). The agent should learn $X \mapsto Y$ rules, where X are game dynamics design concepts, and Y are games.
- Learn the rules to derive actual game dynamics design concepts from actual games (the encoded mechanics$\mapsto$dynamics mapping described in the MDA framework). The agent should learn $X \mapsto Y$ rules, where X are games, and Y are game dynamics design concepts.

The construction of this version of the system requires adding new strategies to the existing rulesets of the participant agents. The formalization of the system is only slightly affected at the high level in the game experience–dynamics $\boldsymbol{PD}$ layer to incorporate the $\boldsymbol{I}$ *interaction* category (non-design category) of dimensions:

$$\Big\langle \dots, \Big\langle \langle \Delta_{PD} \cup \Delta_{I}, \mathcal{U}_{PD} \cup \mathcal{U}_{I} \rangle, \langle \mathring{\Delta}_{PD_h} \cup \mathring{\Delta}_{I_h} \cup \mathring{\Delta}_{PD_c}, \mathring{\mathcal{U}}_{PD_h} \cup \mathring{\mathcal{U}}_{I_h} \cup \mathring{\mathcal{U}}_{PD_c} \rangle \Big\rangle, \dots \Big\rangle.$$

In the following section, we first describe the generalities of the computational implementation of the new strategies, followed by the details of each of them within their corresponding processes.

7.1 Design process update

This system version involves a computational agent capable of learning concept derivation rules (synthetical $\Mapsto$ and analytical $\mapsto$) between two pairs of design levels $\langle \boldsymbol{P}, \boldsymbol{D} \rangle$, and $\langle \boldsymbol{D}, \boldsymbol{M} \rangle$. In terms of the framework, this new capability requires to add two types of chained reflection $\Rrightarrow$ strategies to the computational agent perception rulesets $\mathscr{P}_{PD_c}$ and $\mathscr{P}_{DM_c}$:

a) A reflection strategy (conversion units) to convert the internal representations of concepts (already interpreted by the assistant agent) into specialized representations suitable for a learning algorithm $\big\{ \check{X}_{L^+}, \check{X}_{L} \big\} \Rrightarrow \big\{ \check{X}_{L^+}', \check{X}_{L}' \big\}$.

b) A second chained reflection strategy (learning units) to build the corresponding learning models and formation of learning memory $\big\{ \check{X}_{L^+}, \check{X}_{L} \big\} \Rrightarrow \big\{ \check{X}_{L^+}'', \check{X}_{L}'' \big\}$.

Computationally, second-level reflection strategies are implemented (in a first revision) with a backpropagation algorithm to train a regression multilayer perceptron (MLP) with a single hidden layer. This algorithm's selection leads to implementing the first level reflection strategies as conversion units to convert interpreted representations of concepts into specialized representations in the form of vectors that will be the input data for the second level reflection learning units.

Learning strategies are performed online (i.e., each learning model is updated each time the apprentice is presented with a pair of design concepts of different levels to learn). However, it is not strictly incremental since when the agent is presented with an example and is requested to learn it, it must execute its learning strategy considering the new example plus the concepts already stored in its memory as a training set. This situation, however, may not be a significant issue as long as the size of the training set is small.

Specialized representations of concepts (input vectors) produced by reflection strategies of the type $\big\{ \check{X}_{L^+}, \check{X}_{L} \big\} \Rrightarrow \big\{ \check{X}_{L^+}', \check{X}_{L}' \big\}$ are static; they are limited to a specific subset of design dimensions, and in some cases, their ranges are also limited to a subset. That is, in the case of dimensions having unbounded intervals as ranges, the range is converted to a bounded interval within -100 and 100.

In the case of numerical dimensions, two vector positions are used to encode the subset of the range of the dimension being learned, and in the case of categorical dimensions, a single vector position is used to encode a design dimension.

In all cases, the hyperbolic tangent function **tanh** which is defined in the range $(-1,\ 1)$ is used as MLP activation function to allow encoding both the presence or absence of a dimension and its value if it is present in a submitted design concept:

- Categorical dimension values are encoded in one vector position by mapping a set of n categorical values indexed by $I \coloneqq \{1..n\}$ to numerical values in the interval $\left[\frac{i-1}{n},\ \frac{i}{n}\right)$ for each i^{th} category.
- On the other hand, numerical dimension values are normalized to the interval $[0,\ 1]$ and encoded in two vector positions, each in the interval $[0,\ 1]$ that establish a subregion or subset (as a percentage) of the original range of the design dimension.
- A negative value in one or two vector positions would indicate that the dimension is absent in the concept (it is not part of the current design space).

Example 7.1.1.

- Given the *paths* dimension $\dot{\delta}_{D_{pt}} \coloneqq \langle\{0..10\},\ D^c\rangle$ and a design concept $x_D = \langle \ldots,\ \{1..5\},\ \ldots\rangle$, the *paths* dimension value $\{1..5\}$ would be transformed into two vector positions $[\ldots,\ 0.1,\ 0.5,\ \ldots]$ in the vector representation of the concept. The value of the left side of the dimension value 1 corresponds to the 10% of the original range, and the right side of the dimension value 5 corresponds to the 50% of the original range.
- If the *maze-based* dimension $\left(\dot{\delta}_{D_{gs}}\right)_{mz} \coloneqq \langle I,\ D^c\rangle$ is not present in a design concept x_D but it is defined in the learning model algorithm, the dimension in the transformed concept would be represented by two vector positions with negative one $[\ldots,\ -1.0,\ -1.0,\ \ldots]$.
- Given the *win condition game object* dimension $\dot{\delta}_{M_{wo}} \coloneqq \left\langle C_{ots} \cup C_{an},\ D^H \right\rangle$, the set $C_{ots} \cup C_{an} = \{\text{background, player, wall, target, crate}\}$, and a design concept $x_M = \langle \ldots,\ \{\text{player}\},\ \ldots\rangle$, the dimension value $\{\text{player}\}$ in the transformed concept would be $[\ldots,\ 0.2,\ \ldots]$.

In all cases the mean squared error (MSE) is used as the cost function to be minimized by the backpropagation learning algorithm.

7.1.1 Synthesis

7.1.1.1 Game dynamics synthesis

$$\left\{ \ldots,\ \check{x}_{{De}_h} \dashrightarrow \check{x}_{I_h},\ \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h} \right\} \subseteq \mathscr{P}_{{PD}_h},\ \left\{ \ldots,\ \vec{x}_{I_h} \rightarrow \hat{x}_{I_h} \right\} \subseteq \mathscr{T}_{{PD}_h},$$

$$\Big\{ \dots, \hat{x}_{I_h} \mathrel{\underrightarrow{\subset}} \check{x}_{I_c}^{h}, \{\check{x}_{Pe_h}, \check{x}_{De_h}\} \mathrel{\underrightarrow{\subset}} \{\check{x}_{Pe_c}^{h}, \check{x}_{De_c}^{h}\}, \{\check{x}_{Pe_c}^{h}, \check{x}_{De_c}^{h}\} \dashrightarrow \{\check{x}_{Pe_c}^{h\prime}, \check{x}_{De_c}^{h\prime}\},$$
$$\{\check{X}_{Pe_c}^{h\prime}, \check{X}_{De_c}^{h\prime}\} \dashrightarrow \check{x}_{PD_c}^{h}, \check{x}_{I_c}^{h} \dashrightarrow \check{x}_{I_c}, \check{x}_{PD_c}^{h} \dashrightarrow \check{x}_{I_c}, \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \Big\} \subseteq \mathscr{P}_{PD_c},$$

$$\Big\{ \dots, \vec{x}_{I_c} \rightarrow \hat{x}_{I_c} \Big\} \subseteq \mathscr{T}_{PD_c}.$$

…

Learning request elaboration (h): After having synthesized the expected game dynamics design concept (or at any time), if the human designer h deems the concept sufficiently complete and that it is worthwhile for their computational apprentice c to learn the derivation rules that generated it, then she/he generates the intention to call the apprentice's attention and request it to learn the derivation rules for this state of the design

$$\check{x}_{De_h} \dashrightarrow \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h}.$$

GUI update (h): h externalizes their request via an action button (Fig. 7.1)

$$\vec{x}_{I_h} \rightarrow \hat{x}_{I_h}.$$

Figure 7.1: Game experience-dynamics synthetical derivation learning request button.

Learning request interpretation (c): c interprets the human's request to watch out for the expected concepts of experience and dynamics and learn the synthetical derivation rules that relate them

$$\hat{x}_{I_h} \mathrel{\underrightarrow{\subset}} \check{x}_{I_c}^{h}.$$

Learning start notification (c): In response to the human's request, the computational agent c places a looping progress bar widget labeled as 'Learning' in the tool's GUI to notify that the c agent's learning task has commenced (Fig. 7.2)

$$\check{x}_{I_c}^{h} \dashrightarrow \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

Figure 7.2: Learning-in-progress notification.

Expected experience & dynamics interpretation (c): c interprets the human-sourced expected game experience and dynamics design concepts

$$\left\{\check{x}_{Pe_h},\ \check{x}_{De_h}\right\} \rightrightarrows \left\{\check{x}_{Pe_c^h},\ \check{x}_{De_c^h}\right\}.$$

Expected experience & dynamics transformation (c): c transforms the interpreted representations of expected game experience and dynamics design concepts into specialized representations suitable for the learning algorithm

$$\left\{\check{x}_{Pe_c^h},\ \check{x}_{De_c^h}\right\} \dashrightarrow \left\{\check{x}_{Pe_c^{h\prime}},\ \check{x}_{De_c^{h\prime}}\right\}.$$

The new, specialized representations are vectors that will be the input data for the learning algorithm according to the procedure explained at the beginning of Section 7.1. This strategy covers the complete working set of game experience design dimensions Δ'_P and a subset of the working set of game dynamics design dimensions Δ'_D including 36 *game-object-type-counter* design dimensions for each game object available in the tool, and omitting the *response time* and any additional *gameplay-scenario* design dimension (other than *maze-based*, *twitch-based*, and *logic-based*). That is, 22 dimensions that correspond to 22 PLEX playfulness categories and 45 dynamics dimensions that are transformed into an experience vector of 44 positions and a dynamics vector of 90 positions; that is, two positions per design dimension.

Expected experience–dynamics derivation learning (c): As a second-level chained reflection strategy, c builds or updates a learning model of experience-dynamics synthetical derivation rules

$$\left\{\check{X}_{Pe_c^{h\prime}},\ \check{X}_{De_c^{h\prime}}\right\} \dashrightarrow \check{x}_{PD_c^h}, \text{where } \check{x}_{Pe_c^{h\prime}} \in \check{X}_{Pe_c^{h\prime}},\ \check{x}_{De_c^{h\prime}} \in \check{X}_{De_c^{h\prime}}.$$

The MLP architecture consists of an input layer of 44 units (game experience), a hidden layer of 8 units (determined empirically by experimentation), and an output layer of 90 units (game dynamics) (Fig. 7.3).

The backpropagation algorithm that creates or updates the learning model is executed for 100 epochs with a learning rate of 0.1, both parameters determined empirically by some experimentation.

Learning finish notification (c): Having the learning model built or updated, the computational agent c removes the *learning* looping progress bar widget from the tool's GUI to notify the human agent h that her/his request has been completed

$$\check{x}_{PD_c^h} \dashrightarrow \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

7.1.1.2 Game mechanics synthesis

$$\left\{\ldots,\ \check{x}_{Ma_h} \dashrightarrow \check{x}_{I_h},\ \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h}\right\} \subseteq \mathscr{P}_{DM_h},\ \left\{\ldots,\ \vec{x}_{I_h} \rightarrow \hat{x}_{I_h}\right\} \subseteq \mathscr{T}_{DM_h},$$

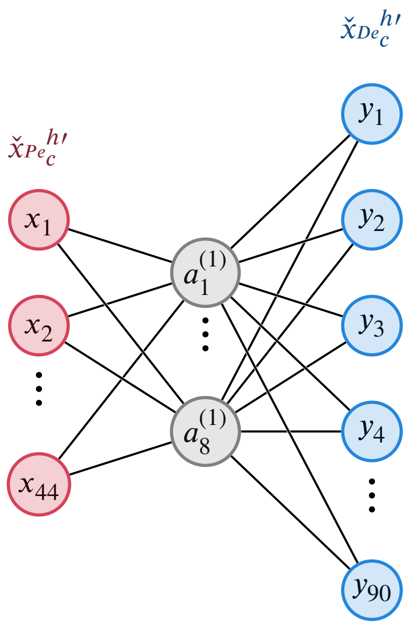


Figure 7.3: MLP architecture for expected experience-dynamics derivation learning.

$$\Big\{ \ldots,\ \hat{x}_{I_h} \subsetneq \check{x}_{I_c}^{h},\ \{\hat{x}_{De_h},\ \hat{x}_{Ma}\} \subsetneq \{\check{x}_{De_c}^{h},\ \check{x}_{Ma_c}\},\ \{\check{x}_{De_c}^{h},\ \check{x}_{Ma_c}\} \overset{\dashleftarrow}{\dashrightarrow} \{\check{x}_{De_c}^{h\prime},\ \check{x}_{Ma_c}'\},$$
$$\{\check{X}_{De_c}^{h\prime},\ \check{X}_{Ma_c}'\} \overset{\dashleftarrow}{\dashrightarrow} \check{x}_{DM_c}^{h},\ \check{x}_{I_c}^{h} \overset{\dashleftarrow}{\dashrightarrow} \check{x}_{I_c},\ \check{x}_{DM_c}^{h} \overset{\dashleftarrow}{\dashrightarrow} \check{x}_{I_c},\ \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \Big\} \subseteq \mathscr{P}_{DM_c},$$

$$\Big\{ \ldots,\ \vec{x}_{I_c} \rightarrow \hat{x}_{I_c} \Big\} \subseteq \mathscr{T}_{DM_c}.$$

…

Learning request (h): After having synthesized the actual game mechanics design concept (or at any time), if the human designer h deems the concept sufficiently complete and that it is worthwhile for their computational apprentice c to learn the derivation rules that generated it, then she/he requests to its computational apprentice c, via an action button (Fig. 7.4), to pay attention to both expected game dynamics and actual game mechanics design concepts and learn the synthetical derivation rules

$$\check{x}_{Ma_h} \overset{\dashleftarrow}{\dashrightarrow} \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h} \rightarrow \hat{x}_{I_h}.$$

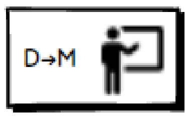


Figure 7.4: Game dynamics-mechanics synthetical derivation learning request button.

Learning request interpretation (c): c interprets the human's request to watch out for the expected game dynamics and actual game mechanics design concepts, and learn the synthetical derivation rules that relate them

$$\hat{x}_{I_h} \mathrel{\subseteq\!\!\!\rightarrow} \check{x}_{I_c}^{h}.$$

Learning start notification (c): In response to the human's request, the computational agent c places a looping progress bar widget labeled as 'Learning' in the tool's GUI to notify that the c agent's learning task has commenced (Fig. 7.2)

$$\check{x}_{I_c}^{h} \mathrel{\subseteq\!\!\!\dashrightarrow} \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

Expected dynamics & actual mechanics interpretation (c): c interprets the human-sourced expected game dynamics and actual game mechanics design concepts

$$\left\{\hat{x}_{De_h},\, \hat{x}_{Ma}\right\} \mathrel{\subseteq\!\!\!\rightarrow} \left\{\check{x}_{De_c}^{h},\, \check{x}_{Ma_c}\right\}.$$

Expected dynamics & actual mechanics transformation (c): c transforms the interpreted representations of expected game dynamics and actual game mechanics design concepts into specialized representations suitable for the learning algorithm

$$\left\{\check{x}_{De_c}^{h},\, \check{x}_{Ma_c}\right\} \mathrel{\subseteq\!\!\!\dashrightarrow} \left\{\check{x}_{De_c}^{h\prime},\, \check{x}_{Ma_c}^{\prime}\right\}.$$

This strategy covers a subset of the working set of game dynamics design dimensions Δ'_D omitting the *response time* and any additional *gameplay-scenario* design dimension, and a subset of the working set of game mechanics design dimensions Δ'_M as explained as follows:

- A game board consisting of up to 22×22 cells (or smaller), each containing up to 4 stacked game objects for a total of $22 \times 22 \times 4 = 1936$ base dimensions.
- A background-exclusive alias (labeled as 'background') with up to 4 game object types (4 dimensions).
- Up to 3 additional aliases labeled as 'player', 'alias1', and 'alias2 containing up to 4 game object types each (12 dimensions).
- Up to 2 win conditions composed of a quantifier, a game object or alias, and an 'on' game object or alias (6 dimensions).
- Up to 5 game rules, each consisting of a rule modifier, a single pattern composed of up to 5 cells; each cell composed of a modifier and a game object type or alias for a total of 105 dimensions.

A total of 45 dynamics dimensions and 2063 mechanics dimensions are transformed into a dynamics vector of 90 positions and a mechanics vector of 2063 positions; that is, two positions per dynamics design dimension and one position per mechanics design dimension.

Expected dynamics–actual mechanics derivation learning (c): As a second-level chained reflection strategy, c builds or updates a learning model of dynamics-mechanics synthetical derivation rules

$$\left\{\check{X}_{De_c}^{h\prime},\, \check{X}_{Ma_c}^{\prime}\right\} \mathrel{\subseteq\!\!\!\dashrightarrow} \check{x}_{DM_c}^{h}, \text{ where } \check{x}_{De_c}^{h\prime} \in \check{X}_{De_c}^{h\prime},\, \check{x}_{Ma_c}^{\prime} \in \check{X}_{Ma_c}^{\prime}.$$

The MLP architecture consists of an input layer of 90 units (game dynamics), a hidden layer of 6 units, and an output layer of 2063 units (game mechanics) (Fig. 7.5). The backpropagation algorithm that creates or updates the learning model is executed for 100 epochs with a learning rate of 0.1.

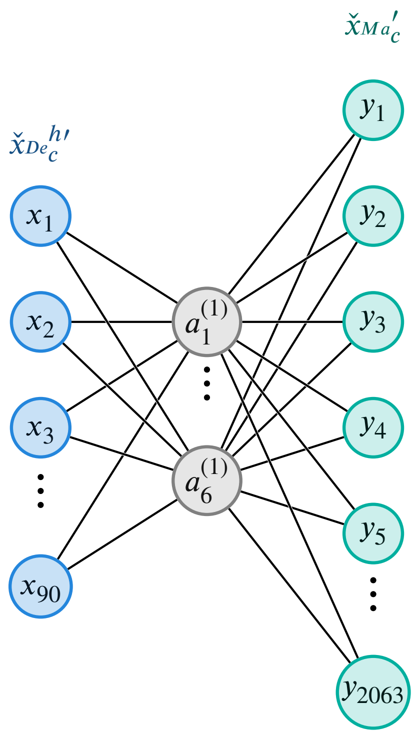


Figure 7.5: MLP architecture for expected dynamics – actual mechanics derivation learning.

Learning finish notification (c): Having the learning model built or updated, the computational agent c removes the *learning* looping progress bar widget from the tool's GUI to notify the human agent h that her/his request has been completed

$$\check{x}_{DM}{}_c^{h} \Rrightarrow \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \to \hat{x}_{I_c}.$$

7.1.2 Analysis

7.1.2.1 Game mechanics analysis

Human-based partial analysis

$$\left\{\ldots, \check{x}_{Da_h} \Rrightarrow \check{x}_{I_h}, \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h}\right\} \subseteq \mathscr{P}_{DM_h}, \left\{\ldots, \vec{x}_{I_h} \to \hat{x}_{I_h}\right\} \subseteq \mathscr{T}_{DM_h},$$

$$\left\{\begin{array}{c} \dots,\ \hat{x}_{I_h} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{I_c}^{h},\ \left\{\hat{x}_{Ma},\ \hat{x}_{Da_h}\right\} \mathrel{\underset{\rightarrow}{\subset}} \left\{\check{x}_{Ma_c},\ \check{x}_{Da_c}^{h}\right\},\ \left\{\check{x}_{Ma_c},\ \check{x}_{Da_c}^{h}\right\} \mathrel{\underset{\dashrightarrow}{\subset}} \left\{\check{x}_{Ma'_c},\ \check{x}_{Da_c}^{h\prime}\right\}, \\ \left\{\check{X}_{Ma'_c},\ \check{X}_{Da_c}^{h\prime}\right\} \mathrel{\underset{\dashrightarrow}{\subset}} \check{x}_{MD_c}^{h},\ \check{x}_{I_c}^{h} \mathrel{\underset{\dashrightarrow}{\subset}} \check{x}_{I_c},\ \check{x}_{MD_c}^{h} \mathrel{\underset{\dashrightarrow}{\subset}} \check{x}_{I_c},\ \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \end{array}\right\} \subseteq \mathscr{P}_{DM_c},$$

$$\left\{\ \dots,\ \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}\right\} \subseteq \mathscr{T}_{DM_c}.$$

...

Learning request (h): After having produced its partial actual game dynamics concept by analyzing the actual game mechanics through playtesting, the human designer h requests to its computational apprentice c, via an action button (Fig. 7.6), to pay attention to both actual game mechanics and human-sourced game dynamics design concepts and learn the analytical derivation rules

$$\check{x}_{Da_h} \mathrel{\underset{\dashrightarrow}{\subset}} \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h} \rightarrow \hat{x}_{I_h}.$$

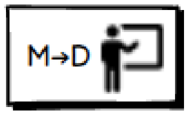


Figure 7.6: Game dynamics-experience analytical derivation learning request button.

Learning request interpretation (c): c interprets the human's request to watch out for the actual game mechanics and human-sourced partial game dynamics design concepts, and learn the analytical derivation rules that relate them

$$\hat{x}_{I_h} \mathrel{\underset{\rightarrow}{\subset}} \check{x}_{I_c}^{h}.$$

Learning start notification (c): In response to the human's request, the computational agent c places a looping progress bar widget labeled as 'Learning' in the tool's GUI to notify that the c agent's learning task has commenced (Fig. 7.2)

$$\check{x}_{I_c}^{h} \mathrel{\underset{\dashrightarrow}{\subset}} \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

Actual mechanics & partial dynamics interpretation (c): c interprets the human-sourced actual game mechanics and partial dynamics design concepts

$$\left\{\hat{x}_{Ma},\ \hat{x}_{Da_h}\right\} \mathrel{\underset{\rightarrow}{\subset}} \left\{\check{x}_{Ma_c},\ \check{x}_{Da_c}^{h}\right\}.$$

Actual mechanics & partial dynamics transformation (c): c transforms the interpreted representations of human-sourced actual game mechanics and partial dynamics design concepts into specialized representations suitable for the learning algorithm

$$\left\{\check{x}_{Ma_c},\ \check{x}_{Da_c}^{h}\right\} \mathrel{\underset{\dashrightarrow}{\subset}} \left\{\check{x}_{Ma'_c},\ \check{x}_{Da_c}^{h\prime}\right\}.$$

This strategy covers the same subset of the working set of game mechanics design dimensions Δ'_M described in Section 7.1.1.2 and a subset of the working set of game dynamics design dimensions Δ'_D omitting any computer-analyzed dimension. That is, 2063 mechanics dimensions and 6 dynamics dimensions transformed into vectors.

Actual mechanics–human-analyzed-dynamics derivation learning (c): As a second-level chained reflection strategy, c builds or updates a learning model of mechanics-dynamics analytical derivation rules

$$\left\{\check{X}_{Ma}{}'_c,\ \check{X}_{Da}{}^{h\prime}_c\right\} \Rrightarrow \check{x}_{MD}{}^{h}_c, \text{ where}$$

$$\check{x}_{Ma}{}'_c \in \check{X}_{Ma}{}'_c,\ \check{x}_{Da}{}^{h\prime}_c \in \check{X}_{Da}{}^{h\prime}_c.$$

The MLP architecture consists of an input layer of 2063 units (game mechanics), a hidden layer of 4 units, and an output layer of 6 units (game dynamics) (Fig. 7.7). The backpropagation algorithm that creates or updates the learning model is executed for 100 epochs with a learning rate of 0.1.

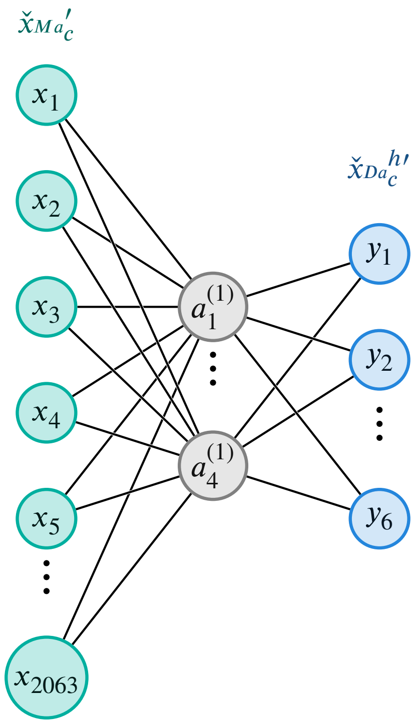


Figure 7.7: MLP architecture for actual mechanics–human-analyzed-dynamics derivation learning.

Learning finish notification (c): Having the learning model built or updated, the computational agent c removes the *learning* looping progress bar widget from the tool's GUI to notify the human agent h that her/his request has been completed

$$\check{x}_{MD}{}^{h}_c \Rrightarrow \check{x}_{I\,c} \rightleftarrows \vec{x}_{I\,c} \rightarrow \hat{x}_{I\,c}.$$

Computer-based partial analysis (c)

$$\Big\{\ \ldots,\ \check{x}_{Da\,c} \Rrightarrow \check{x}_{I\,c},\ \left\{\check{x}_{Ma\,c},\ \check{x}_{Da\,c}\right\} \Rrightarrow \left\{\check{x}_{Ma}{}'_c,\ \check{x}_{Da}{}'_c\right\},$$
$$\left\{\check{X}_{Ma}{}'_c,\ \check{X}_{Da}{}'_c\right\} \Rrightarrow \check{x}_{MD\,c},\ \check{x}_{MD\,c} \Rrightarrow \check{x}_{I\,c},\ \check{x}_{I\,c} \rightleftarrows \vec{x}_{I\,c}\Big\} \subseteq \mathscr{P}_{DM\,c},$$

$$\left\{ \ldots,\ \vec{x}_{I_c} \rightarrow \hat{x}_{I_c} \right\} \subseteq \mathscr{T}_{DM_c}.$$

…

Learning start: After having produced the partial actual game dynamics concept by analyzing the human-sourced actual game mechanics through automatic playtesting, the computational agent c places a looping progress bar widget labeled as ‘Learning’ in the tool’s GUI to notify that it is starting a learning task to learn the analytical rules that made it producing the game dynamics concept (Fig. 7.2)

$$\check{x}_{Da_c} \Rrightarrow \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

Actual mechanics & dynamics transformation: c transforms the already interpreted representations of human-sourced actual game mechanics and its own partial game dynamics design concepts into specialized representations suitable for the learning algorithm

$$\left\{\check{x}_{Ma_c},\ \check{x}_{Da_c}\right\} \Rrightarrow \left\{\check{x}_{Ma'_c},\ \check{x}_{Da'_c}\right\}.$$

This strategy covers the same subset of the working set of game mechanics design dimensions Δ'_M described in Section 7.1.1.2, and a subset of the working set of game dynamics design dimensions Δ'_D omitting the *response time* and any human-analyzed design dimension. That is, 2063 mechanics dimensions and 42 dynamics dimensions transformed into vectors.

Actual mechanics–dynamics derivation learning: As a second-level chained reflection strategy, c builds or updates a learning model of mechanics-dynamics analytical derivation rules

$$\left\{\check{X}_{Ma'_c},\ \check{X}_{Da'_c}\right\} \Rrightarrow \check{x}_{MD_c}, \text{ where}$$

$$\check{x}_{Ma'_c} \in \check{X}_{Ma'_c},\ \check{x}_{Da'_c} \in \check{X}_{Da'_c}.$$

The MLP architecture consists of an input layer of 2063 units (game mechanics), a hidden layer of 4 units, and an output layer of 84 units (game dynamics) (Fig. 7.8). The backpropagation algorithm that creates or updates the learning model is executed for 100 epochs with a learning rate of 0.1.

Learning finish notification: c removes the *learning* looping progress bar widget from the tool’s GUI to notify that its learning task has ended

$$\check{x}_{MD_c} \Rrightarrow \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

7.1.2.2 Game dynamics analysis

$$\left\{ \ldots,\ \check{x}_{Pa_h} \Rrightarrow \check{x}_{I_h},\ \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h} \right\} \subseteq \mathscr{P}_{PD_h},\ \left\{ \ldots,\ \vec{x}_{I_h} \rightarrow \hat{x}_{I_h} \right\} \subseteq \mathscr{T}_{PD_h},$$

$$\Big\{ \ldots,\ \hat{x}_{I_h} \subseteq \check{x}_{I_c}^{h},\ \left\{\hat{x}_{Da_h},\ \hat{x}_{Pa_h}\right\} \subseteq \left\{\check{x}_{Da_c}^{h},\ \check{x}_{Pa_c}^{h}\right\},\ \left\{\check{x}_{Da_c},\ \check{x}_{Da_c}^{h},\ \check{x}_{Pa_c}^{h}\right\} \Rrightarrow \left\{\check{x}_{Da_c}^{ch\prime},\ \check{x}_{Pa_c}^{h\prime}\right\},$$

$$\left\{\check{X}_{Da_c}^{ch\prime},\ \check{X}_{Pa_c}^{h\prime}\right\} \Rrightarrow \check{x}_{DP_c}^{ch},\ \check{x}_{I_c}^{h} \Rrightarrow \check{x}_{I_c},\ \check{x}_{DP_c}^{h} \Rrightarrow \check{x}_{I_c},\ \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \Big\} \subseteq \mathscr{P}_{PD_c},$$

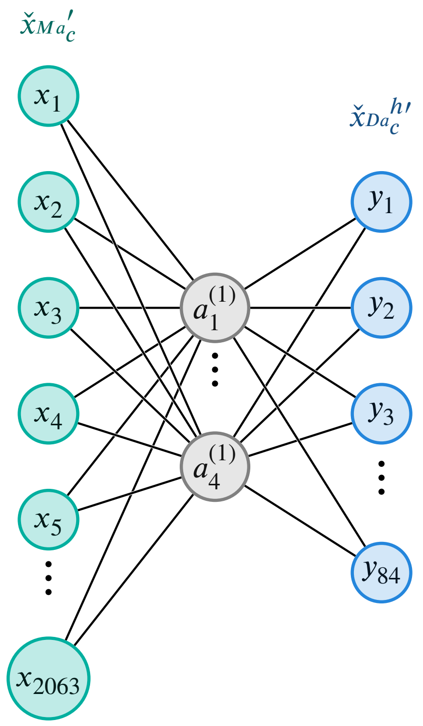


Figure 7.8: MLP architecture for actual mechanics–computer-analyzed-dynamics derivation learning.

$$\left\{ \ldots, \vec{x}_{I_c} \rightarrow \hat{x}_{I_c} \right\} \subseteq \mathscr{T}_{PD_c}.$$

...

Learning request (h): After having produced the actual game experience by analyzing the actual game dynamics, the human designer h requests to its computational apprentice c, via an action button (Fig. 7.9), to pay attention to both actual game experience and dynamics design concepts and learn the analytical derivation rules

$$\check{x}_{Pa_h} \mathrel{\subseteq\!\!\!\rightrightarrows} \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h} \rightarrow \hat{x}_{I_h}.$$

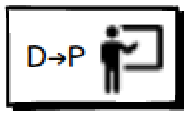


Figure 7.9: Game dynamics-experience analytical derivation learning request button.

Learning request interpretation (c): c interprets the human's request to watch out for the actual game dynamics and experience design concepts, and learn the analytical derivation rules that relate them

$$\hat{x}_{I_h} \mathrel{\subseteq\!\!\!\rightrightarrows} \check{x}_{I_c}^{h}.$$

Learning start notification (c): In response to the human's request, the computational agent c places a looping progress bar widget labeled as 'Learning' in the tool's GUI to notify that the c agent's learning task has commenced (Fig. 7.2)

$$\check{x}_{I_c}^{h} \dashrightarrow \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

Actual dynamics & experience interpretation (c): c interprets the human-sourced actual game dynamics and experience design concepts

$$\left\{\check{x}_{Da_h}, \check{x}_{Pa_h}\right\} \underrightarrow{\subset} \left\{\check{x}_{Da_c}^{h}, \check{x}_{Pa_c}^{h}\right\}.$$

Actual dynamics & experience transformation (c): c transforms the interpreted representations of actual game dynamics (both human and computer sourced) and experience design concepts into specialized representations suitable for the learning algorithm

$$\left\{\check{x}_{Da_c}, \check{x}_{Da_c}^{h}, \check{x}_{Pa_c}^{h}\right\} \dashrightarrow \left\{\check{x}_{Da_c}^{ch\prime}, \check{x}_{Pa_c}^{h\prime}\right\}.$$

This strategy covers the same subsets of working sets of game dynamics Δ'_D and game experience Δ'_P design dimensions described in Section 7.1.1.1 but backwards. 45 dynamics dimensions and 22 game experience dimensions are transformed into an experience vector of 44 positions and a dynamics vector of 90 positions.

Actual dynamics–experience derivation learning (c): As a second-level chained reflection strategy, c builds or updates a learning model of dynamics-experience analytical derivation rules

$$\left\{\check{X}_{Da_c}^{ch\prime}, \check{X}_{Pa_c}^{h\prime}\right\} \dashrightarrow \check{x}_{DP_c}^{ch}, \text{ where}$$

$$\check{x}_{Da_c}^{ch\prime} \in \check{X}_{Da_c}^{ch\prime}, \check{x}_{Pa_c}^{h\prime} \in \check{X}_{Pa_c}^{h\prime}.$$

The MLP architecture consists of an input layer of 90 units (game dynamics), a hidden layer of 11 units (determined empirically by experimentation), and an output layer of 44 units (game experience) (Fig. 7.10). The backpropagation algorithm that creates or updates the learning model is executed for 100 epochs with a learning rate of 0.1, both parameters determined empirically by some experimentation.

Learning finish notification (c): Having the learning model built or updated, the computational agent c removes the *learning* looping progress bar widget from the tool's GUI to notify the human agent h that her/his request has been completed

$$\check{x}_{DP_c}^{ch} \dashrightarrow \check{x}_{I_c} \rightleftarrows \vec{x}_{I_c} \rightarrow \hat{x}_{I_c}.$$

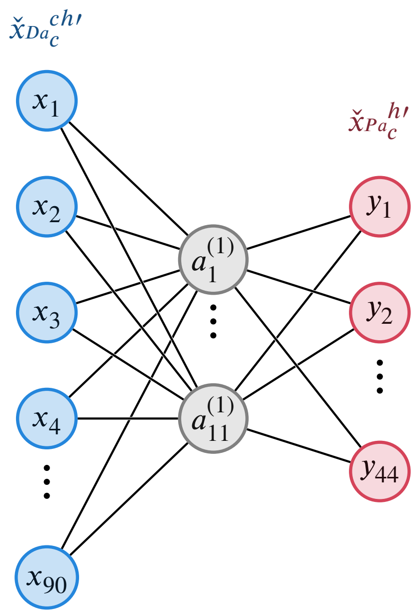


Figure 7.10: MLP architecture for actual dynamics-experience derivation learning.

7.2 Computational results

7.2.1 Test designs set

A small set of six puzzle video games was developed both to functionally and comprehensively test the PoC tool and to have a way to systematically test and tune the learning strategies introduced in this version of the computational agent (i.e., different MLP architectures and parameters).

The games were designed, implemented, and analyzed directly in the tool with the help of a puzzle video game hobbyist and casual gamer (not a professional game designer) with access to PuzzleScript's video game gallery as an inspiring set. For this task, no significant constraints were imposed on the non-professional designer other than the limitations imposed by the characteristics of the learning strategies (such as restrictions on the number of or ranges of specific dimensions) and the express request to try to create interesting but straightforward games (games for experimentation). Before and during the design process, the designer was instructed about the meaning of some of the design dimensions, especially the game dynamics level dimensions that were somewhat unintuitive to a person who is not immersed in the formal terminology of video game design theory. Although the set of game objects remains static in the tool, the objects requested by the designer were introduced to the system to be available in the tool interface.

During the video game design exercise, the designer expressed the need to incorporate a mechanism to name and draw game objects at game mechanics and dynamics design levels. "It is possible to design games with generic game objects or a static game object palette; however, their specific names and appearances help create a theme or atmosphere directly related to ludic experiences evocation.” This request (which may be satisfied in future work) reflects the importance of the user interface of a creative support tool or creative-domain collaboration system and evidences the cause-effect relationship between the different design levels.

The following is a brief description of the designs created:

Design 1 - ‘Alice’: It is a canonical maze-based game in which the player’s avatar (Alice) must meet the ‘mad hatter’ to win the game. It did not require modeling any game rules as it uses the implicit movement rules of the PuzzleScript engine, which allows the player to move his avatar freely on the board as long as his path is not blocked by some other object located on the same layer on which the player’s avatar is located. In order to make the labyrinth more interesting, a large board was chosen.

The game experience design concept was designed with only two dimensions: ‘challenge’ and ‘completion’ (Fig. 7.11a). Two dynamics design dimensions were added into the dynamics default design space: ‘alice’ and ‘strippedwall’ counter dimensions (Figs. 7.11b and 7.11c). The game mechanics were designed with a 21×22 game space, two non-background layers, one ‘player’ alias (additional to the default background alias) to cast the ‘alice’ object as the ‘player’ object, and one win condition `'All alice On madhatter'` to produce the playable game depicted in Fig. 7.12.

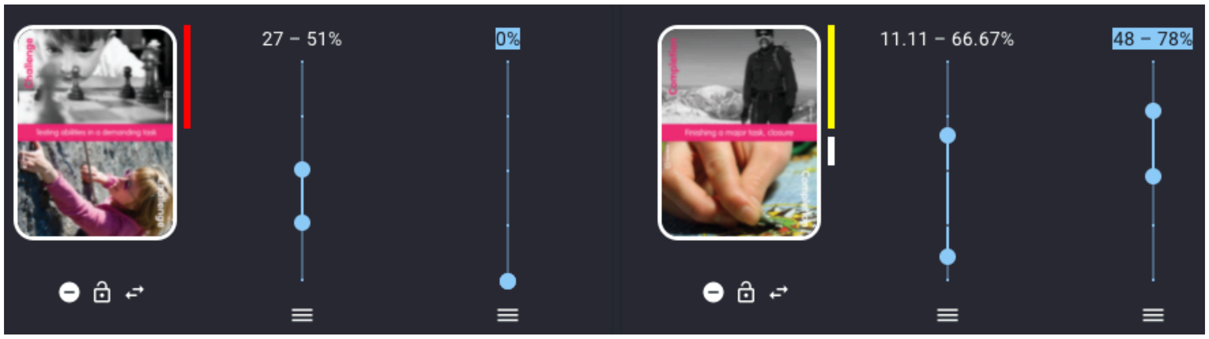
Design 2 - ‘CookieMonster’: In this game, the cookie monster (the player’s avatar) must eat all the cookies that appear in the game space. The cookie monster can only move from its current position to the position of a cookie placed directly in front of it (orthogonally), free of obstacles in between and no matter how many cells are between the monster and the cookie. The movement restrictions make the player run the risk of getting stuck if it eats the cookies in the wrong order.

The game experience was modeled with two dimensions: ‘challenge’ and ‘nurture’ (Fig. 7.13a). Two dynamics design dimensions were added into the dynamics default design space: ‘cmonster’ and ‘cookie’ counter dimensions (Figs. 7.13b and 7.13c). The playable game (Fig. 7.14) was produced by synthetizing the game mechanics with a small 7×8 game space, a single non-background layer, one ‘player’ alias to cast ‘cmonster’ as ‘player’, one simple win condition `'No cookie'`, and the following pair of game rules:

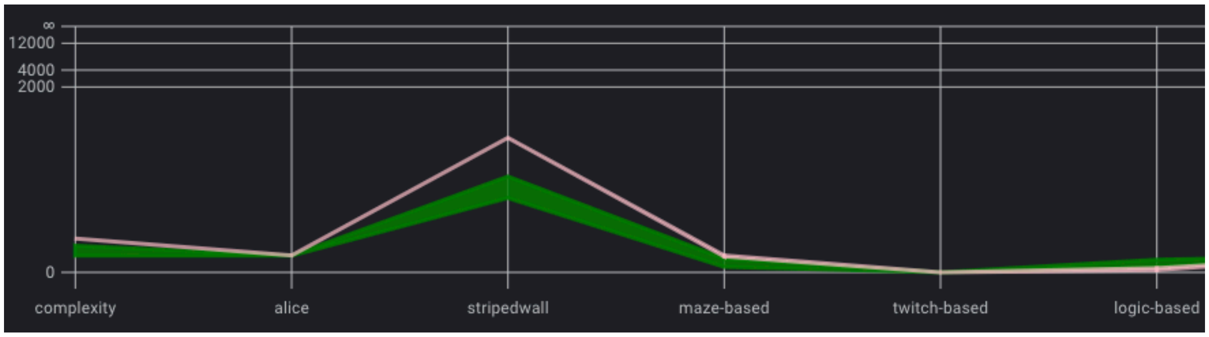
```
[ > cmonster | ... | cookie ] -> [ | ... | cmonster ]
[ > cmonster ] -> [ cmonster ]
```

Design 3 - ‘Hulk’ It is a game in which the player (hulk) must move rocks to make his way to the heart token.

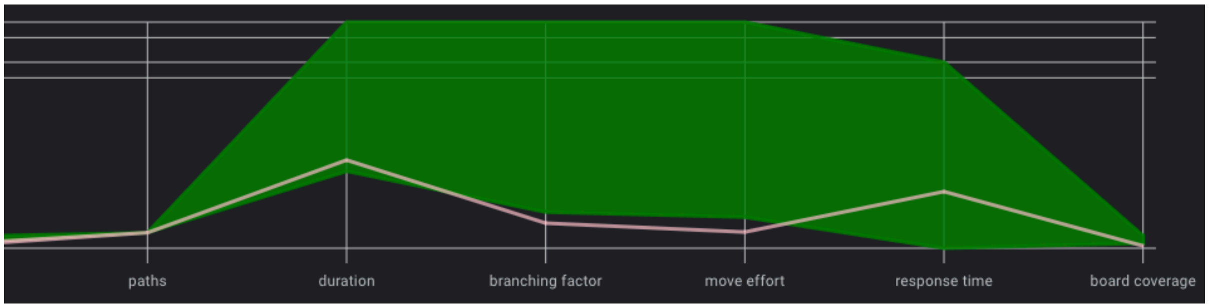
The game experience was modeled with three dimensions: ‘captivation’, ‘exploration’, and ‘fantasy’ (Fig. 7.15a). Three dynamics design dimensions were added into the dynamics default design space: ‘rock’, ‘tree’, and ‘rockwall’ counter dimensions (Figs. 7.15b and 7.15c), although ‘tree’



(a) Game experience



(b) Game dynamics (1/2)



(c) Game dynamics (2/2)

Figure 7.11: Design 1 - Game experience & dynamics.

game object was not used in the actual game mechanics design. The playable game (Fig. 7.16) was produced by synthetizing the game mechanics design concept with a rectangular 5×10 game space, two non-background layers, a ‘player’ alias to cast ‘hulk’ as ‘player’, a ‘alias1’ alias to create a synonym for ‘rockwall’ (static object) and ‘rock’ (movable object) objects, a ‘alias2’ alias to create a synonym for ‘rockwall’ and ‘step’ objects, one win condition `'All hulk On heart'`, and five somewhat complex game rules:

```
vertical [ > hulk ] -> [ > hulk ]
[ > hulk | no alias1 ] -> [ > hulk | ]
horizontal [ > hulk | rock | ... | no alias1 | alias2 ] ->
[ > hulk | > rock | ... | | alias2 ]
horizontal [ > rock | ... | rock ] -> [ > rock | ... | > rock ]
```

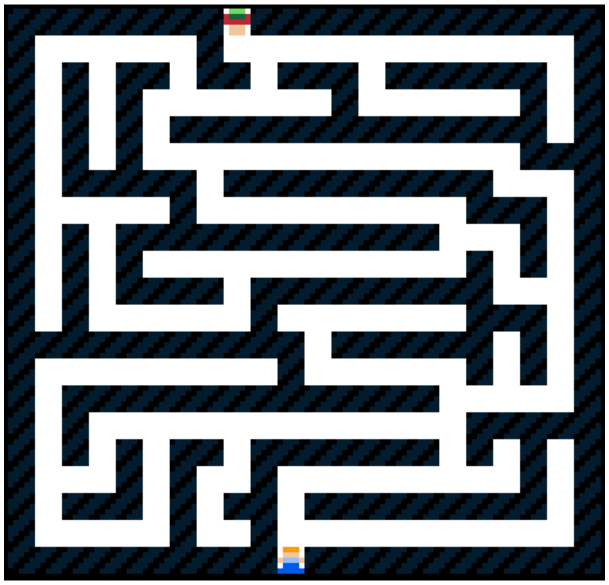


Figure 7.12: Design 1 - Game mechanics (playable game).

```
horizontal [ rock | ... | > rock ] -> [ > rock | ... | > rock ]
```

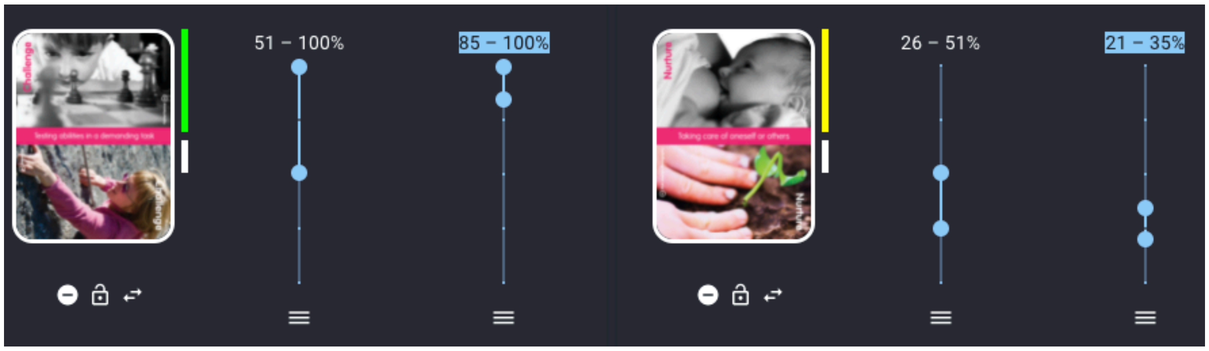
Design 4 - 'Zombie' In this game, the player's avatar is a zombie, and the objective is to turn all the humans that appear in the game space into zombies. When a human token becomes a zombie, it joins the zombie horde, which moves as a whole. If any of the horde members pass over a bomb, the game becomes unable to be won.

The game experience was modeled with two dimensions: 'exploration' and 'humor' (Fig. 7.17a). Two dynamics design dimensions were added into the dynamics default design space: 'bomb' and 'zombie' counter dimensions (Figs. 7.17b and 7.17c). The playable game (Figs. 7.18a and 7.18b) was produced by synthetizing the game mechanics design concept with a rectangular 7×9 game space, two non-background layers, a 'player' alias to cast the 'zombie' as 'player', a pair of win conditions `'No human2'`, `'No skull'`, and the following pair of game rules:

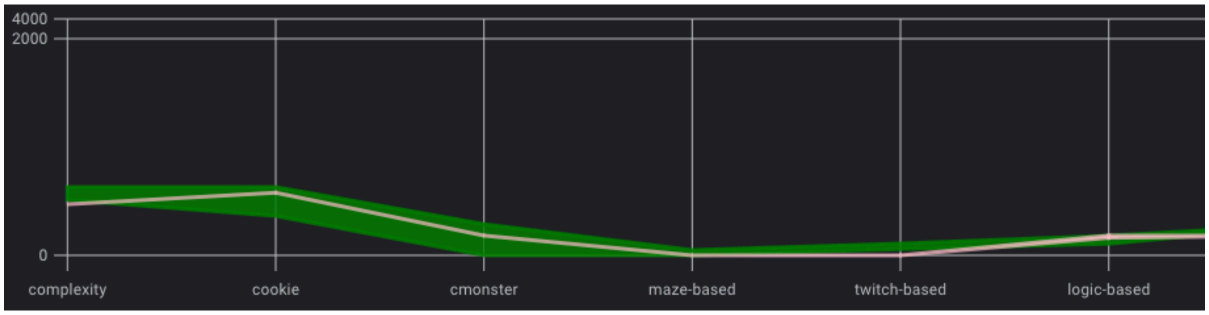
```
late [ zombie | human2 ] -> [ zombie | zombie ]
[ > zombie | bomb ] -> [ | skull]
```

Design 5 - 'Zombiekiller' It is a game that adds some complexity to a typical maze-based puzzle (like design 1). In this case, the player, in addition to finding the labyrinth's exit, must exterminate all the zombies in the game space.

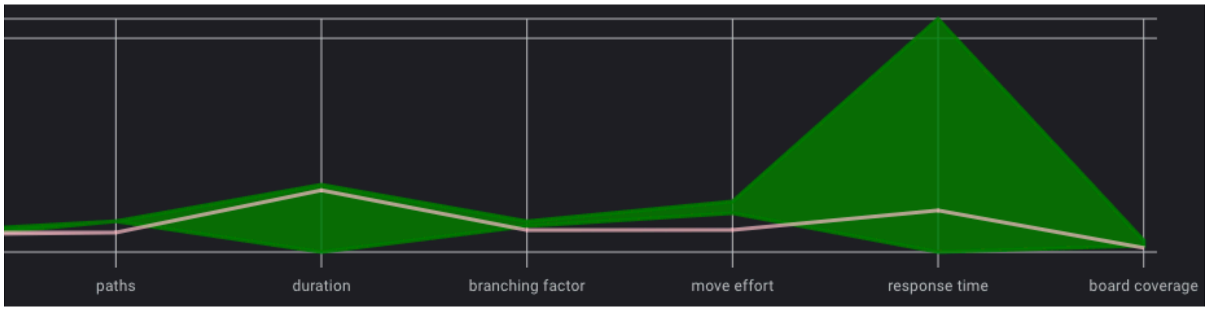
The game experience was modeled with two dimensions: 'cruelty' and 'exploration' (Fig. 7.19a). The 'zombie' counter dimension was added to the dynamics default design space: (Figs. 7.19b and 7.19c). The playable game (Fig. 7.20) was synthesized as a small 5×5 square game space, three non-background layers, a 'player' alias to cast the 'human1' object as 'player' object, a pair


Challenge
testing abilities in a demanding task
51 – 100%
85 – 100%
Nurture
Taking care of oneself or others
26 – 51%
21 – 35%

(a) Game experience



(b) Game dynamics (1/2)



(c) Game dynamics (2/2)

Figure 7.13: Design 2 - Game experience & dynamics.

of win conditions `'No zombie'`, `'All human1 On bottle'`, and the single game rule `[ action human1 | zombie ] -> [ human1 | blood ]`.

Design 6 - 'Zombiekiller2' It is the same game as design 5 but within a much larger game space.

The 'challenge' experience was added into the experience design space of the design 5 (Fig. 7.21a). The playable game (Fig. 7.22) was synthesized as a larger 21×22 square game space, keeping the same layers, aliases, win conditions, and game rules as the design 5.

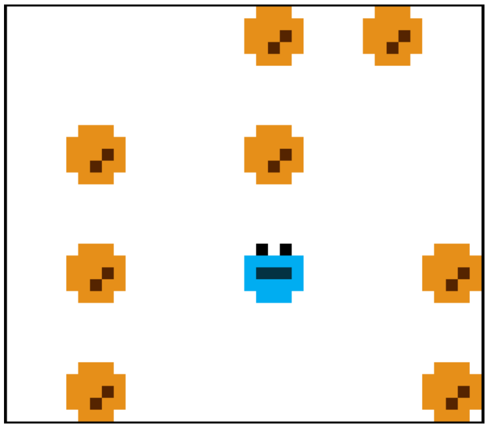
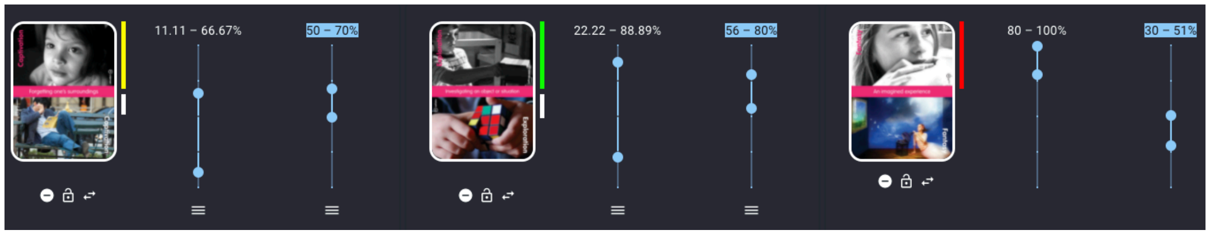
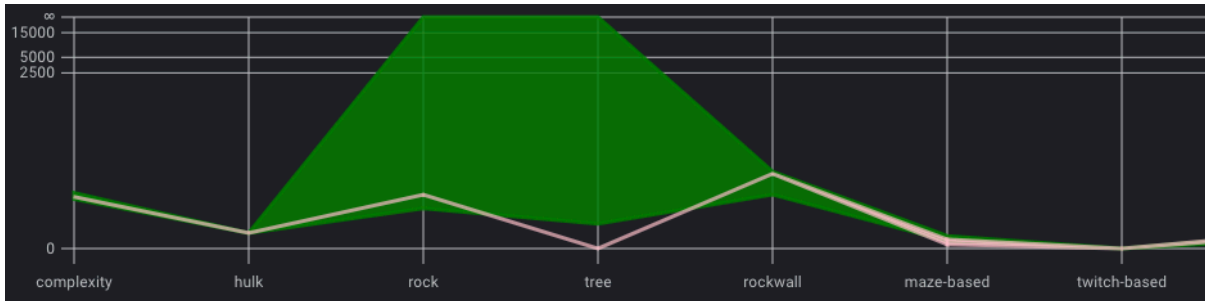


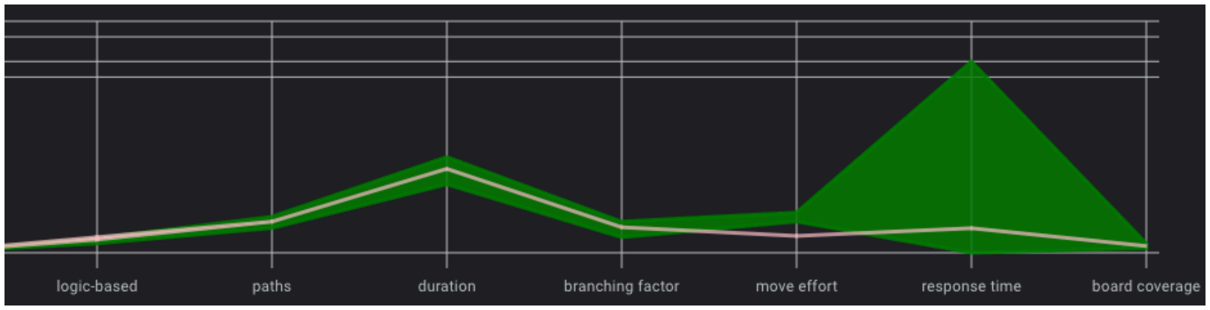
Figure 7.14: Design 2 - Game mechanics (playable game).



(a) Game experience



(b) Game dynamics (1/2)



(c) Game dynamics (2/2)

Figure 7.15: Design 3 - Game experience & dynamics.

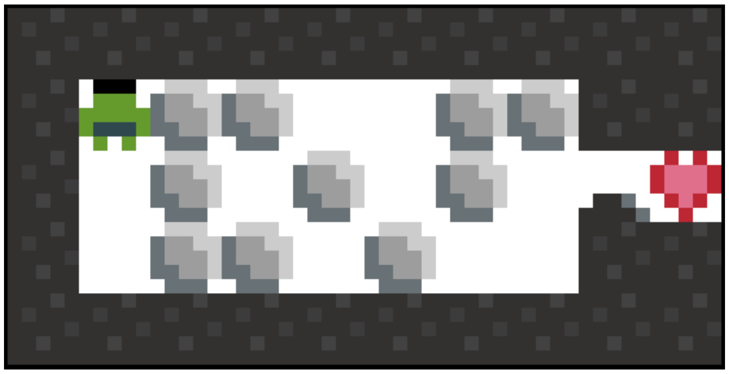
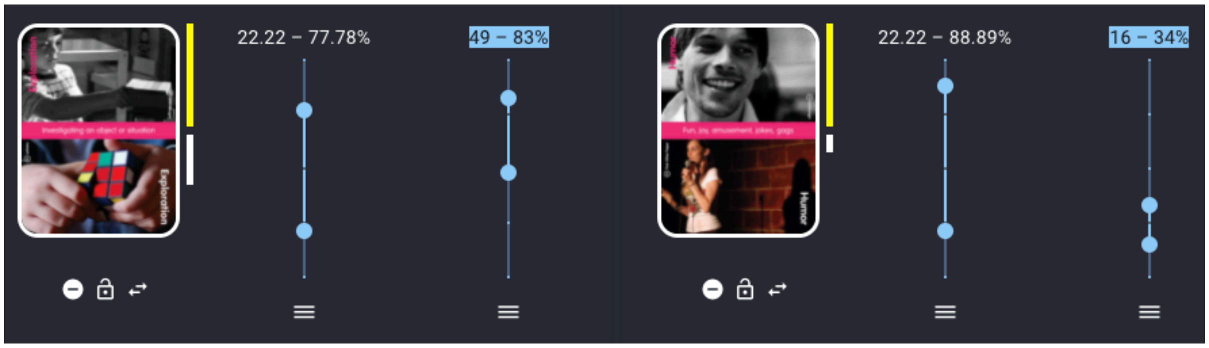
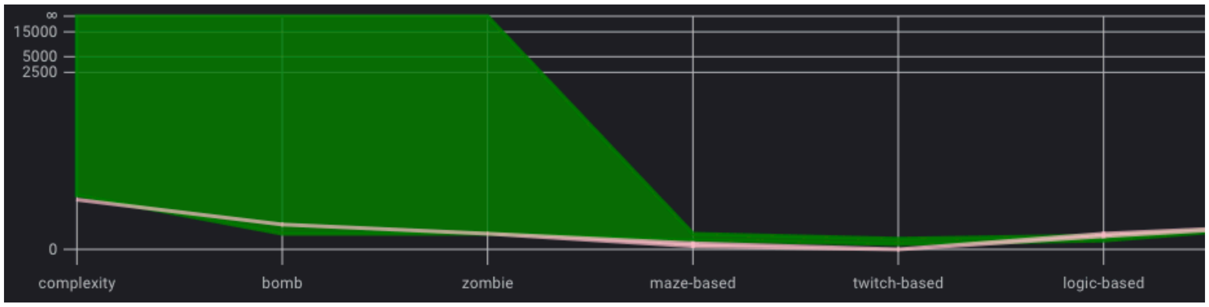


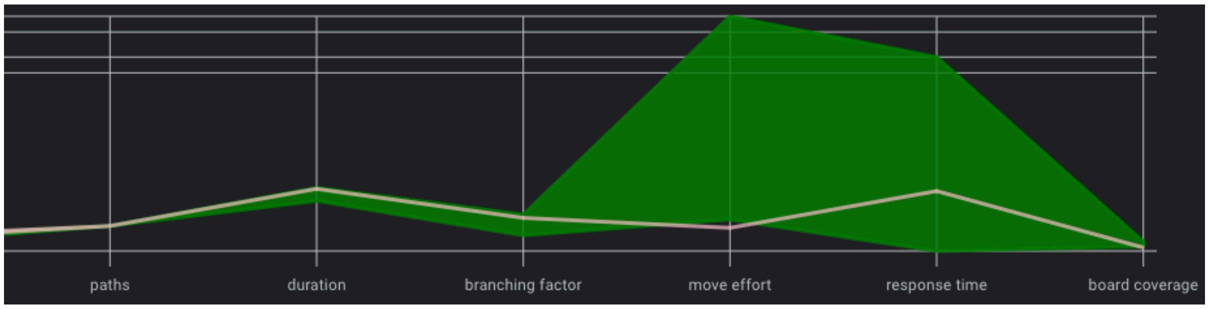
Figure 7.16: Design 3 - Game mechanics (playable game).



(a) Game experience

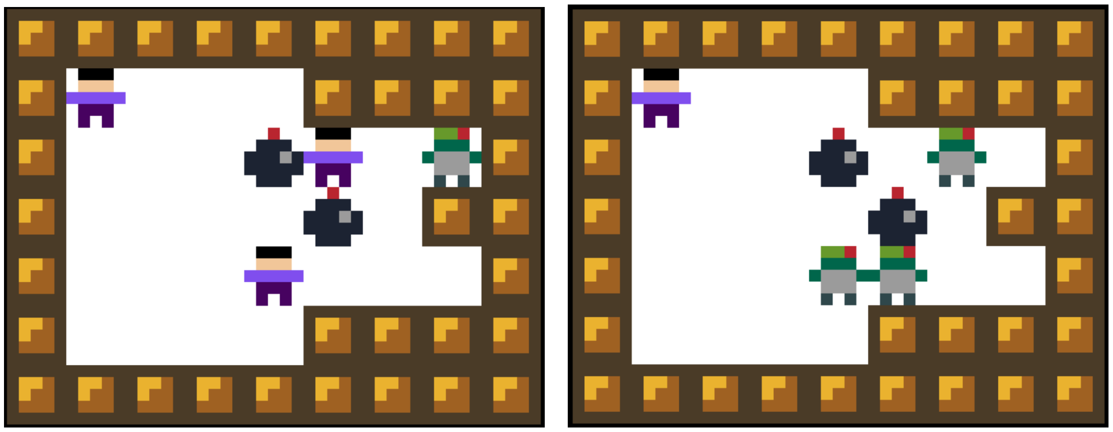


(b) Game dynamics (1/2)



(c) Game dynamics (2/2)

Figure 7.17: Design 4 - Game experience & dynamics.



(a) Initial game state (b) The zombie player becomes a zombie horde player

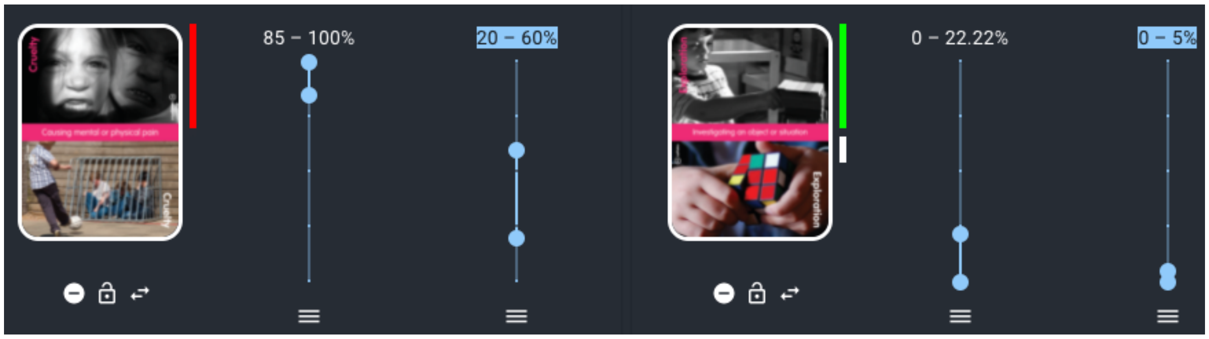
Figure 7.18: Design 4 - Game mechanics.

7.2.2 Strategies performance

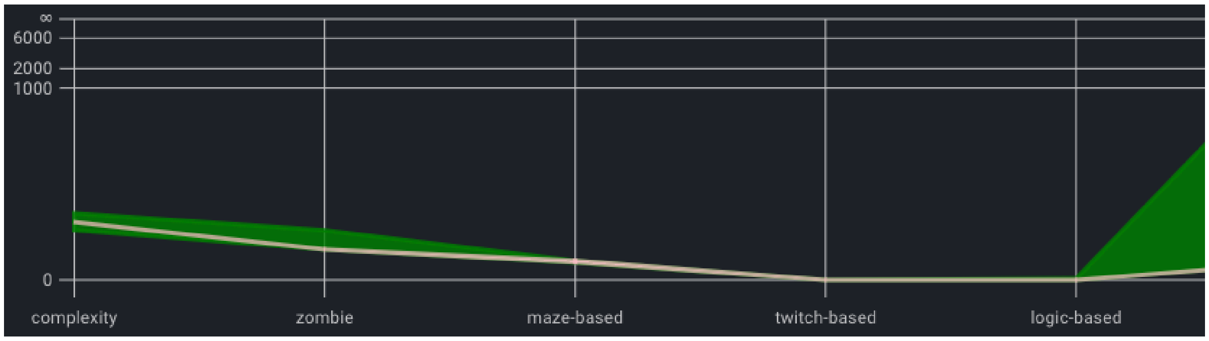
The set of 6 designs was used to experiment, tune, and fix the architecture and parameters of the different learning strategies with two objectives in mind: to achieve an acceptable error and, at the same time, to avoid the agent taking too much time to execute each strategy. Once the architectures and parameters were fixed, the following performance was achieved after a 20-executions round (Table 7.1):

Table 7.1: Computational 'assistant' learning strategies performance.

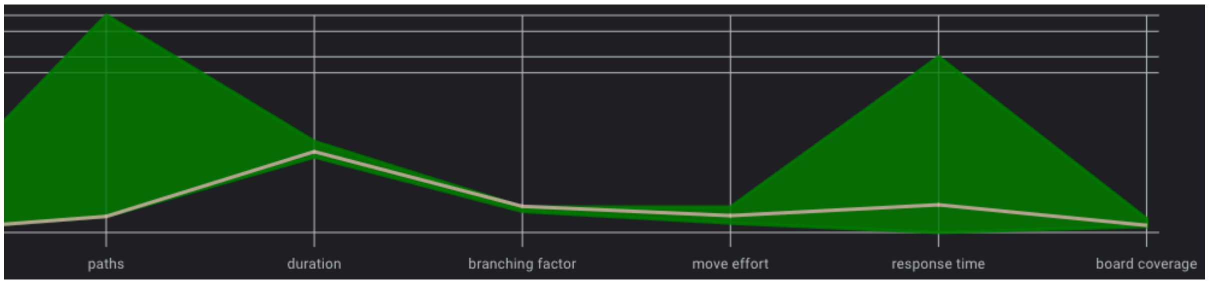
Second-level reflection strategy	MLP architecture	Epochs	Learning rate	MSE
$\left\{\check{X}_{Pe}{}_c^{h\prime}, \check{X}_{De}{}_c^{h\prime}\right\} \rightrightarrows \check{x}_{PD}{}_c^{h}$	$44 \times 8 \times 90$			0.03
$\left\{\check{X}_{De}{}_c^{h\prime}, \check{X}_{Ma}{}_c^{\prime}\right\} \rightrightarrows \check{x}_{DM}{}_c^{h}$	$90 \times 6 \times 2063$			0.10
$\left\{\check{X}_{Ma}{}_c^{\prime}, \check{X}_{Da}{}_c^{h\prime}\right\} \rightrightarrows \check{x}_{MD}{}_c^{h}$	$2063 \times 4 \times 6$	100	0.1	0.03
$\left\{\check{X}_{Ma}{}_c^{\prime}, \check{X}_{Da}{}_c^{\prime}\right\} \rightrightarrows \check{x}_{MD}{}_c$	$2063 \times 4 \times 84$			0.11
$\left\{\check{X}_{Da}{}_c^{ch\prime}, \check{X}_{Pa}{}_c^{h\prime}\right\} \rightrightarrows \check{x}_{DP}{}_c^{ch}$	$90 \times 11 \times 44$			0.01



(a) Game experience



(b) Game dynamics (1/2)



(c) Game dynamics (2/2)

Figure 7.19: Design 5 - Game experience & dynamics.

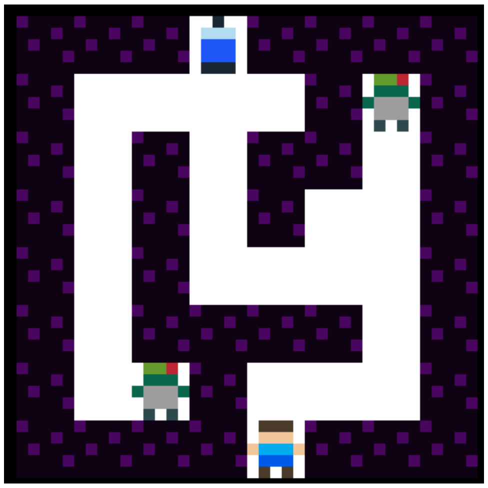
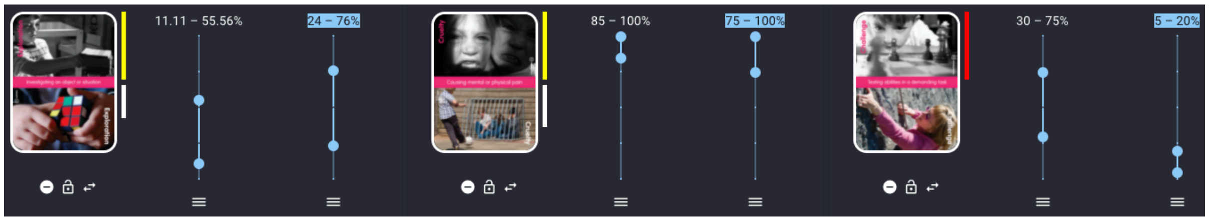
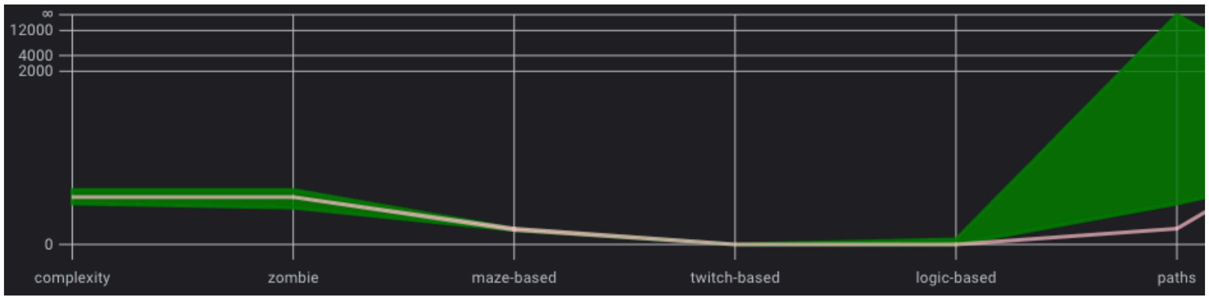


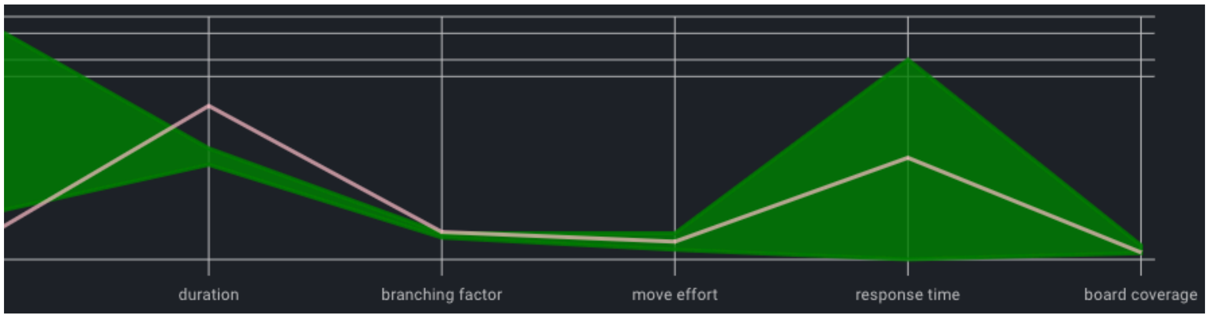
Figure 7.20: Design 5 - Game mechanics (playable game).



(a) Game experience



(b) Game dynamics (1/2)



(c) Game dynamics (2/2)

Figure 7.21: Design 6 - Game experience & dynamics.

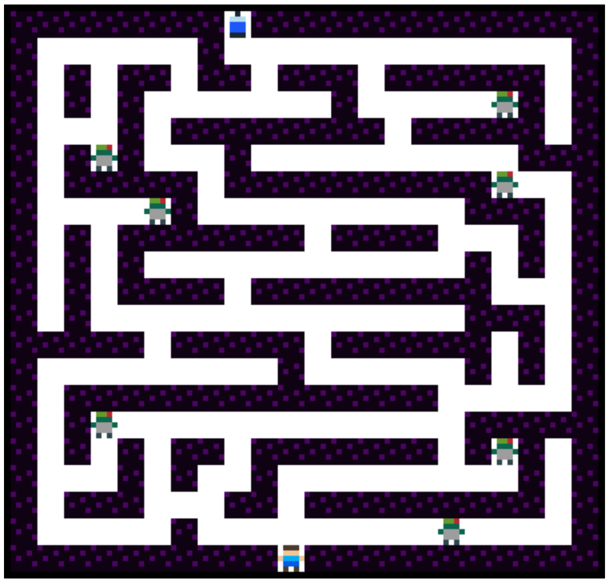


Figure 7.22: Design 6 - Game mechanics (playable game).

8

Building a collaborative CCDS: “Creative” assistant computational agent

In this chapter, we describe the next version of the collaborative CCDS. In addition to its previous roles, the computational agent c that we call in this version ‘creative’ assistant, provides design ideas to the human designer h based on the current design state.

In particular, a scenario similar to a brainstorming session is established where the human designer asks (when some inspiration is required) the computational agent to produce a set of video-game-level ideas (arrangements of game objects in the game space) based on the human designer’s current game-level design. At the same time, through the previously implemented learning strategies, the computational agent can keep collecting valuable knowledge that could later be used to implement capabilities such as providing analysis estimates and thus speeding up the design process and saving computational resources by avoiding costly evaluations[1].

[1]After building the previous version of the CCDS, the next step in its evolution was to turn the computational agent from an assistant/apprentice into a ‘creative’ assistant. Learning capabilities were provided to the agent so that it could learn (via multilayer perceptrons) derivation rules for synthesis and analysis. The initial improvement strategy was to leverage the agent learning capabilities (and acquired knowledge) to generate concepts from this learning. However, although this approach could be suitable for estimation (generating approximate concepts) in analysis processes, it did not seem to be very useful for concept generation by synthesis because the creative potential for a given input concept is limited (and fixed) to a single output concept (the multilayer perceptron will always give the same output for the same input) and what we are interested in is that the agent produces not only a single concept but a set of concepts from an expected concept of a higher level of design. We tried a variational autoencoder (instead of a multilayer perceptron) to give another chance to the connectionist paradigm to learn and generate variant concepts. Although the implementation worked, the concepts produced (variants of a seed concept) were highly similar since our test set was probably minimal (for a connectionist paradigm). However, we believe this option could still be feasible if a more extensive set is available. We then decided to explore other

A new section and widgets were crafted and incorporated into the GUI on which the artificial agent externalizes its ideas (including their analysis and evaluations) and the human designer manages the brainstorming session and selects an idea provided by her/his assistant to be promoted into the new state of the design.

We also document a formative and summative qualitative evaluation of the brainstorming scenario conducted by a user interface designer and puzzle video game enthusiast. The evaluation results may suggest that the human designer could have benefited technically and creatively after participating in collaborative design sessions with the artificial assistant.

8.1 Design process update

The construction of this system version requires adding new strategies to the existing perception, action and evaluative rulesets of the participant agents, more specifically, into the synthesis, analysis, and evaluation processes for designing. Computationally, the "creative" assistant utilizes evolutionary computation (EC) techniques as the basis for its synthesis strategy. In particular, two ways of implementing these strategies (selectable by the human designer) are implemented in the system, one based on a direct-representation GA with fitness-proportional mutation and the other based on a custom $(\mu/\rho, \lambda)$-type (Beyer and Schwefel 2002) evolution strategy (ES). Both algorithms were modified and implemented as interactive evolutionary algorithms (EAs), so the human designer always controls the evolution of the brainstorming session (one generation at a time).

For this CCDS version, the computational agent c should be able to:

- Build an internal representation of the current human-led design at both expected game dynamics $\hat{x}_{De_h}$ and actual game mechanics $\hat{x}_{Ma}$ levels.
- Produce partial game mechanics design concepts $\hat{X}_{M\underline{a}_c}$ as design suggestions to satisfy expected game dynamics besides being based on the current human designer's design ($\boldsymbol{M}$ synthesis).
- Derive game dynamics produced by each game level idea ($\boldsymbol{M}$ analysis).
- Assess derived game dynamics for each generated idea against both expected and current design's actual game dynamics ($\boldsymbol{D}$ evaluation).

On the other hand, the human designer h should be able to:

- Request creative assistance of the computational agent c.
- Assess ideas $\hat{X}_{M\underline{a}_c}$ produced by her/his assistant ($\boldsymbol{M}$ analysis and $\boldsymbol{D}$ evaluation).

computational alternatives to incorporate more creative behavior in the agent, taking advantage of the CCDSF's modular architecture.

- Pick an idea $\hat{x}_{M\underline{a}_c}$ and set it as her/his current design $\hat{x}_{Ma}$.
- Parametrize and manage the flow of the brainstorming session, thus affecting the "creative" behavior of her/his assistant.
- Restore the design to its previous state $\hat{x}_{Ma_{(t-1)}}$.

In the following sections, we describe the new strategies, their computational implementation within their corresponding synthesis, analysis, and evaluation (already operational) processes for designing, and the GUI related additions. Three scenarios are described and implemented: the initial c-creative turn, the subsequent h-turn(s), and the subsequent c-creative turn(s) of a brainstorming session.

8.1.1 Brainstorming session initial c-creative turn

This process models the scenario where the human designer requests (and obtains) the creative assistance of the artificial agent for the first time. The 'first time' refers to the point after the human designer has altered the game's design in terms of game mechanics or dynamics levels, either by formulation, synthesis, or reformulation (i.e., when the assistant's current design spaces are no longer compatible with the human ones) for which a new brainstorming session can be initiated (triggered by the human) based on the new state of the design.

Both human and computational agents are provided with new perception $\mathscr{P}$ and action $\mathscr{T}$ strategies embedded into the synthesis and analysis processes at the game mechanics level and the evaluation process at the game dynamics level as follows:

$$\begin{aligned}\Big\{ \ldots,\ & \hat{x}_{Ma} \mathrel{\subseteq\!\!\!\rightarrow} \check{x}_{Ma_h} \dashrightarrow \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h}, \\ & \left\{\hat{X}_{M\underline{a}_c}, \hat{X}_{M\underline{v}_h}, \hat{X}_{D\underline{a}_c}, \hat{X}_{D\underline{v}_c}\right\} \mathrel{\subseteq\!\!\!\rightarrow} \left\{\check{X}_{M\underline{a}_h^c}, \check{X}_{M\underline{v}_{h(t)}}, \check{X}_{D\underline{a}_h^c}, \check{X}_{D\underline{v}_h^c}\right\} \Big\} \subseteq \mathring{\mathscr{P}}_{DM_h},\end{aligned}$$

$$\Big\{ \ldots, \vec{x}_{I_h} \rightarrow \hat{x}_{I_h} \Big\} \subseteq \mathring{\mathscr{T}}_{DM_h},$$

$$\begin{aligned}\Big\{ \ldots,\ & \left\{\hat{x}_{I_h}, \hat{x}_{De_h}, \hat{x}_{Ma}\right\} \mathrel{\subseteq\!\!\!\rightarrow} \left\{\check{x}_{De_c^h}, \check{x}_{Ma_c}\right\}, \check{x}_{Ma_c} \dashrightarrow \check{x}_{Ma_c''}, \\ & \left\{\check{x}_{De_c^h}, \check{x}_{Ma_c}, \check{x}_{Ma_c''}\right\} \rightleftarrows \left\{\vec{x}_{De_c^h}, \vec{x}_{Ma_c}, \vec{x}_{Ma_c''}\right\}, \\ & \Big\{\vec{x}_{De_c^h}, \vec{x}_{Ma_c}, \vec{x}_{Ma_c''}, \vec{X}_{M\underline{a}_c}\Big\} \rightleftarrows \left\{\check{x}_{De_c^h}, \check{x}_{Ma_c}, \check{x}_{Ma_c''}, \check{X}_{M\underline{a}_c}\right\}, \\ & \hat{x}_{Da_c} \mathrel{\subseteq\!\!\!\rightarrow} \check{x}_{Da_c}, \left\{\check{x}_{De_c^h}, \check{x}_{Da_c}\right\} \rightleftarrows \left\{\vec{x}_{De_c^h}, \vec{x}_{Da_c}\right\}, \\ & \left\{\check{X}_{M\underline{a}_c}, \check{X}_{D\underline{a}_c}, \check{X}_{D\underline{v}_c}\right\} \rightleftarrows \Big\{\vec{X}_{M\underline{a}_c}, \vec{X}_{D\underline{a}_c}, \vec{X}_{D\underline{v}_c}\Big\} \Big\} \subseteq \mathring{\mathscr{P}}_{DM_c},\end{aligned}$$

$$\begin{aligned}\Big\{ \ldots,\ & \left\{\vec{x}_{De_c^h}, \vec{x}_{Ma_c}, \vec{x}_{Ma_c''}\right\} \Mapsto \vec{X}_{M\underline{a}_c}, \\ & \Big\{\vec{X}_{M\underline{a}_c}, \vec{X}_{D\underline{a}_c}, \vec{X}_{D\underline{v}_c}\Big\} \rightarrow \left\{\hat{X}_{M\underline{a}_c}, \hat{X}_{D\underline{a}_c}, \hat{X}_{D\underline{v}_c}\right\} \Big\} \subseteq \mathring{\mathscr{T}}_{DM_c},\end{aligned}$$

$$\left\{ \ldots, \left(\left\{\vec{x}_{De_c^h}, \vec{x}_{Da_c}\right\} \leftrightarrow \check{X}_{D\underline{a}_c}\right), \left\{\check{x}_{De_c^h}, \check{x}_{Ma_c}, \check{x}_{Ma_c''}, \check{X}_{M\underline{a}_c}\right\} \mapsto \check{X}_{D\underline{a}_c} \right\} \subseteq \mathring{\mathscr{E}}_{DM_c}$$

1) **Brainstorming session initiation request (h):** After (re)formulation or at any time during the synthesis process at the game mechanics level, the human designer may require creative support to continue developing her/his idea or, perhaps, to discover new possibilities. If so, the designer h asks the computational assistant c to provide a set of game-level design ideas based on her/his current design, whether the level is empty (she/he has only formulated the design space) or already in progress

$$\hat{x}_{Ma} \underrightarrow{\subseteq} \check{x}_{Ma_h} \underset{\dashrightarrow}{\subset\!\!\dashrightarrow} \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h}.$$

2) **GUI update (h):** h externalizes her/his request via an action button identified by a 'bulb' (idea) icon in a new GUI container widget (Fig. 8.1) located next to the main design game board widget of the game editor panel (Fig. 6.12)

$$\vec{x}_{I_h} \rightarrow \hat{x}_{I_h}.$$

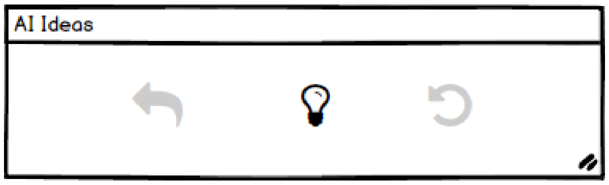


Figure 8.1: Game-level-design ideas widget ready to start brainstorming session. The 'undo' button (only enabled when an assistant's idea is currently applied as the main design) resets the design to the state prior to applying one of the ideas provided by the assistant. With the 'bulb' (idea) button, the human designer requests the computational assistant to start a brainstorming session or to move to a new round of ideas within the same session. With the 'restart' button (only enabled during an active session), the human designer requests the computational assistant to generate a new brainstorming session from scratch.

3) **Brainstorming session starting up (c):** c interprets the human request to participate in a brainstorming session and triggers the interpretation of the current human-designed concept at the game mechanics level and the expected game dynamics

$$\left\{\hat{x}_{I_h}, \hat{x}_{De_h}, \hat{x}_{Ma}\right\} \underrightarrow{\subseteq} \left\{\check{x}_{De_c^h}, \check{x}_{Ma_c}\right\}.$$

4) **Reference design transformation (c):** c extracts a partial concept from the whole current design concept at the game mechanics level and encodes it as a reference genotype $\check{x}_{Ma_c''}$ to be used in its EAs-based synthesis strategy

$$\check{x}_{Ma_c} \underset{\dashrightarrow}{\subset\!\!\dashrightarrow} \check{x}_{Ma_c''}.$$

The genotype is a direct representation of a video game level encoded into a fixed length vector of integers (its length is fixed relative to the current formulated design space: game objects and the number of layers) where the first two positions encode the size of the game space (rows $\times$ columns) between a minimum of 4×4 and a maximum of 10×10, and the remaining positions encode a flattened three-dimensional matrix where each position corresponds to a game object (encoded with a string lookup table) placed in a three-dimensional coordinate (row, column, layer). The genotype always encodes a 10×10 game space; however, the first two control genes express the actual size. In the initialization stage of the EA, the size of the reference design game board may be smaller than the 10×10 maximum size, in which case the genetic information is completed by repeating the first ($10\times10\times$ the number of layers) $-$ (rows $\times$ columns $\times$ the number of layers) placed game objects of the level. Otherwise (the reference game board is bigger than 10×10 in any dimension), the board is truncated to 10×10.

For the ES underlying algorithm, one extra gene (position) is appended to the genotype to encode the mutation probability endogenous parameter, which evolves along with its 'domain' genes and is randomly generated during initialization within a range of minimum and maximum mutation probabilities (an exogenous parameter).

5) **Reference concepts gathering (c):** c collects the reference concepts at game dynamics and mechanics levels required to produce design ideas including the EA's seed genotype

$$\left\{\check{x}_{De_c^h},\ \check{x}_{Ma_c},\ \check{x}_{Ma_c''}\right\} \rightleftarrows \left\{\vec{x}_{De_c^h},\ \vec{x}_{Ma_c},\ \vec{x}_{Ma_c''}\right\}.$$

6) **EA-population-initialization-based synthesis (c):** c produces the first set of design ideas (game levels) through the population initialization routine of the underlying EA

$$\left\{\vec{x}_{De_c^h},\ \vec{x}_{Ma_c},\ \vec{x}_{Ma_c''}\right\} \Mapsto \vec{X}_{M\underline{a}_c}.$$

Each new member of the initial population is created by repeatedly applying two custom operators:

a) **Fluctuating mutation operator:** This custom operator works with two parameters, a minimum and maximum mutation probabilities. A random probability within the [minimum, maximum] range is computed for each new individual and taken as the mutation probability for a standard uniform mutation operator that is applied to the reference concept genotype, thus creating a reference concept mutant.

   This operator aims to obtain a population composed of individuals with different levels of similarity to the reference concept, some remarkably similar and others not so similar but still preserving some common ancestral features. The operator attempts to model the influence of the reference concept (inspiration concept) from the game mechanics design level.

b) **"Booster" operator:** This custom operator attempts to model the influence of the expected game dynamics concept by randomly removing objects placed in excess (by previous variation operators) in the level for each game-object-type-counter game dynamics dimension.

   A reason for incorporating this operator in the processing pipeline was to compensate for the slow convergence obtained as a side effect of transforming the algorithm from its traditional form to an interactive one, thus introducing early, hopefully beneficial, features and repairs to the population.

7) **Self-analysis preparation (c):** c collects the reference concepts along with the produced concepts (level ideas) in preparation for analysis as complete games

$$\left\{\vec{x}_{De_c}^{h},\, \vec{x}_{Ma_c},\, \vec{x}_{Ma_c}'',\, \vec{X}_{M\underline{a}_c}\right\} \rightleftarrows \left\{\check{x}_{De_c}^{h},\, \check{x}_{Ma_c},\, \check{x}_{Ma_c}'',\, \check{X}_{M\underline{a}_c}\right\}.$$

8) **Self-analysis (c):** following the EA's logic, the computational agent c derives a game dynamics concept for each generated idea (through its ability, it already had to play and perform an intrinsic analysis of the game)

$$\left\{\check{x}_{De_c}^{h},\, \check{x}_{Ma_c},\, \check{x}_{Ma_c}'',\, \check{X}_{M\underline{a}_c}\right\} \mapsto \check{X}_{D\underline{a}_c}.$$

Since, in this process stage, the computational agent must analyze several ideas (EA's population size) instead of just one, the analysis is performed using only one search algorithm: Greedy Best-first search. This algorithm was selected after reviewing the performance of all the search algorithms (Section 6.1.3.1) playing the test set of games previously created (Section 7.2.1), finding that this algorithm offered the highest efficiency (the shortest solutions in the fewest number of iterations). The maximum number of iterations was also reduced at this stage of the process by 40% since it was also found that, at least for the test set, the selected algorithm could solve the games within a smaller number of iterations.

9) **Current dynamics interpretation (c):** c interprets the external representation of the actual game dynamics derived from the current game (which it previously obtained using its existing capabilities) in order to contrast it with the actual game dynamics concepts derived for each of its game level ideas

$$\hat{x}_{Da_c} \subseteq\!\!\rightrightarrows \check{x}_{Da_c}.$$

10) **Self-evaluation preparation (c):** c selects the current design's expected and actual game dynamics concepts

$$\left\{\check{x}_{De_c}^{h},\, \check{x}_{Da_c}\right\} \rightleftarrows \left\{\vec{x}_{De_c}^{h},\, \vec{x}_{Da_c}\right\}.$$

11) **Self-evaluation (c):** c compares the actual game dynamics (derived by analysis) of each of the ideas to the expected and actual game dynamics of the reference design by using a modified comparison strategy to determine the distances between each idea's dynamics and the reference expected and actual dynamics and, in terms of EC, the fitness of each individual (ideas)

$$\check{X}_{D\underline{v}_c} = \left(\left\{\vec{x}_{De_c}^{h},\, \vec{x}_{Da_c}\right\} \leftrightarrow \check{X}_{D\underline{a}_c}\right).$$

Because the complexity dimension $\dot{\delta}_{D_c}$ is not affected by the arrangement of game objects in the game space, it is suppressed from the analysis and evaluation of ideas. The fitness computation of each individual is performed in three stages:

I) For each actual game dynamics (derived from each idea), a categorical assessment (see Eq. 5.2) is computed for each dimension. The categorical value is converted to a numerical value according to the following equivalences:

out-of-range	0
range-partially-exceeded	1/3
range-exceeded	2/3
in-range	1

Then, the obtained values are grouped and averaged into three fitness values: 'intrinsic game dynamics' (*game-object-type-counters*), 'player engagement' (*branching factor, move effort, response time, board coverage*), and 'playability' (*paths, duration*). Finally, the three fitnesses are combined into a single fitness value through linear scalarization with the following weights:

'intrinsic dynamics'	0.5
'player engagement'	0.3
'playability'	0.2

II) For each idea's actual game dynamics concept, the statistical mode of its categorical assessments is computed, and a global categorical assessment is assigned: *mostly-out-of-range* if the mode in the group is *out-of-range*, and *rarely-out-of-range* otherwise. In this same stage, a feasibility checklist test is performed based on the generated game's correctness and the duration dimension's value. Through this test, each individual is assigned a feasibility category as follows:

   a) *Uncompilable* if the generated game does not compile in the PuzzleScript engine.
   b) *Unsolvable* if the game is playable in the PuzzleScript engine, although the search algorithm could not find a solution.
   c) *Irrelevant* if the game has a less than three moves solution.
   d) *Relevant* if the game has a duration greater than three moves.

III) Finally, the fitness of each individual computed in the first stage is penalized according to its feasibility and overall categorical assessment:

uncompilable	100% (full penalization)
unsolvable — mostly-out-of-range	95%
unsolvable — rarely-out-of-range	80%
irrelevant — mostly-out-of-range	75%
irrelevant — rarely-out-of-range	50%
relevant — mostly-out-of-range	20%
relevant — rarely-out-of-range	0% (no penalization)

12) **Ideas curation (c):** c integrates the ideas produced with their corresponding actual game dynamics and evaluation concepts and selects the four[2] best ideas according to the computed fitness

$$\left\{\check{X}_{M\underline{a}_c},\, \check{X}_{D\underline{a}_c},\, \check{X}_{D\underline{v}_c}\right\} \rightleftarrows \left\{\vec{\bar{X}}_{M\underline{a}_c},\, \vec{\bar{X}}_{D\underline{a}_c},\, \vec{\bar{X}}_{D\underline{v}_c}\right\}.$$

13) **GUI update (c):** c presents its ideas' selection to the human designer by rendering them on the GUI's game-level-design-ideas widget (Fig. 8.2) together with the results of the analysis and evaluation

$$\left\{\vec{\bar{X}}_{M\underline{a}_c},\, \vec{\bar{X}}_{D\underline{a}_c},\, \vec{\bar{X}}_{D\underline{v}_c}\right\} \to \left\{\hat{X}_{M\underline{a}_c},\, \hat{X}_{D\underline{a}_c},\, \hat{X}_{D\underline{v}_c}\right\}.$$

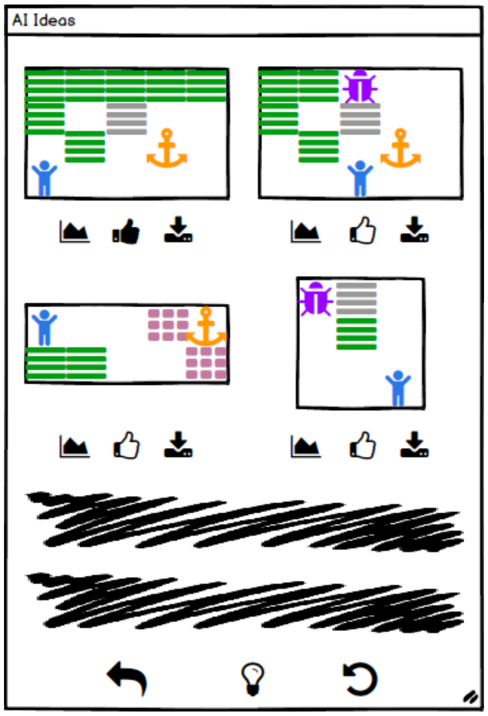


Figure 8.2: Game-level-design-ideas populated widget. The container is divided into three sections. In the upper section, each of the four ideas curated by the computational assistant is rendered in detail (albeit in a smaller size than the main design). Three buttons are provided for each rendered idea: a) a button to review the results of the analysis $\hat{x}_{D\underline{a}_c}$ and evaluation $\hat{x}_{D\underline{v}_c}$ of the idea through a visual representation (chart) of the distances between expected, current actual, and idea's actual game dynamics concepts similar to the chart in the game dynamics main panel (Figs. 6.25 and 6.26), b) a button to apply the idea (promote it into current design) $\hat{x}_{M\underline{a}} = \hat{x}_{M\underline{a}_c}$, and c) a 'toggle' button to mark as 'liked' (or remove the mark) the idea, thus setting a human-based evaluation $\hat{x}_{D\underline{v}_h}$. The general action buttons are now placed into the lowest part of the container.

14) **Game-level-ideas interpretation (h):** The human designer interprets the ideas proposed by the assistant, their corresponding analysis and evaluation results, and the current 'liked'

[2]It was experimentally determined that 4 was a reasonable number of options to offer to the human designer since, on the one hand, it does not represent a heavy cognitive load for the designer, and on the other hand, it reduces expensive rendering resources.

status of the ideas $\check{X}_{M\underline{v}_{h(t-1)}}$ previously set by her/himself (if any)

$$\left\{\hat{X}_{M\underline{a}_c},\ \hat{X}_{M\underline{v}_h},\ \hat{X}_{D\underline{a}_c},\ \hat{X}_{D\underline{v}_c}\right\} \subsetneqq \left\{\check{X}_{M\underline{a}_h^c},\ \check{X}_{M\underline{v}_{h(t)}},\ \check{X}_{D\underline{a}_h^c},\ \check{X}_{D\underline{v}_h^c}\right\}.$$

8.1.2 Brainstorming session subsequent h-turn(s)

Once the game-level ideas produced by the computational assistant c are presented in the GUI $|\hat{X}_{M\underline{a}_c}| = |\hat{X}_{D\underline{a}_c}| = |\hat{X}_{D\underline{v}_c}| = 4$, the human designer h can take her/his turn in the collaborative process to perform any of the following potential actions on or concerning some idea or ideas:

$$\left\{\ \ldots,\ \check{X}_{M\underline{v}_{h(t+1)}} \rightleftarrows \vec{X}_{M\underline{v}_{h(t+1)}},\ \check{x}_{M\underline{a}_h^c} \dashrightarrow \check{x}_{I\underline{a}_h} \rightleftarrows \vec{x}_{I\underline{a}_h},\ \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h}\right\} \subseteq \mathring{\mathscr{P}}_{DM_h},$$

$$\left\{\ \ldots,\ \vec{X}_{M\underline{v}_{h(t+1)}} \rightarrow \hat{X}_{M\underline{v}_h},\ \vec{x}_{I\underline{a}_h} \rightarrow \hat{x}_{I\underline{a}_h},\ \vec{x}_{I_h} \rightarrow \hat{x}_{I_h}\right\} \subseteq \mathring{\mathscr{T}}_{DM_h},$$

$$\begin{aligned}\Big\{\ \ldots,\ &\left(\left\{\vec{x}_{De_h},\ \vec{x}_{Ma_h}\right\}\leftrightarrow\left\{\check{x}_{M\underline{a}_h^c},\ \check{X}_{M\underline{v}_{h(t)}},\ \check{X}_{D\underline{a}_h^c},\ \check{X}_{D\underline{v}_h^c}\right\}\right),\\ &\left(\left\{\vec{x}_{De_h},\ \vec{x}_{Ma_h}\right\}\dot{\leftrightarrow}\left\{\check{x}_{M\underline{a}_h^c},\ \check{X}_{M\underline{v}_{h(t)}},\ \check{X}_{D\underline{a}_h^c},\ \check{X}_{D\underline{v}_h^c}\right\}\right),\\ &\left(\left\{\vec{x}_{De_h},\ \vec{x}_{Ma_h}\right\}\dot{\leftrightarrow}\left\{\check{x}_{M\underline{a}_h^c},\ \check{X}_{M\underline{v}_{h(t)}},\ \check{X}_{D\underline{a}_h^c},\ \check{X}_{D\underline{v}_h^c}\right\}\right)\Big\} \subseteq \mathring{\mathscr{E}}_{DM_h},\end{aligned}$$

$$\begin{aligned}\Big\{\ \ldots,\ &\left\{\hat{x}_{I\underline{a}_h},\ \hat{x}_{Ma}\right\} \subsetneqq \left\{\check{x}_{I\underline{a}_c^h},\ \check{x}_{Ma_c}\right\},\ \check{x}_{Ma_c} \dashrightarrow \check{x}_{Ma_{c(t-1)}},\ \check{x}_{I\underline{a}_c^h} \dashrightarrow \check{x}_{M\underline{a}_c} \rightleftarrows \vec{x}_{M\underline{a}_c},\\ &\hat{x}_{I_h} \subsetneqq \check{x}_{I_c^h} \dashrightarrow \check{x}_{Ma_{c(t-1)}} \rightleftarrows \vec{x}_{Ma_{c(t-1)}}\Big\} \subseteq \mathring{\mathscr{P}}_{DM_c},\end{aligned}$$

$$\left\{\ \ldots,\ \vec{x}_{M\underline{a}_c} \rightarrow \hat{x}_{Ma},\ \vec{x}_{Ma_{c(t-1)}} \rightarrow \hat{x}_{Ma}\right\} \subseteq \mathring{\mathscr{T}}_{DM_c}$$

8.1.2.1 Sets/updates 'liked' status of one or more ideas

'Liked' status update (h): By contrasting the ideas provided by the assistant c with her/his expected design at the game dynamics level and her/his actual design at the game mechanics level, the human designer h determines (or updates) which ideas the assistant provided are to her/his liking

$$\check{X}_{M\underline{v}_{h(t+1)}} = \left(\left\{\vec{x}_{De_h},\ \vec{x}_{Ma_h}\right\}\leftrightarrow\left\{\check{x}_{M\underline{a}_h^c},\ \check{X}_{M\underline{v}_{h(t)}},\ \check{X}_{D\underline{a}_h^c},\ \check{X}_{D\underline{v}_h^c}\right\}\right),$$
$$\check{X}_{M\underline{v}_{h(t+1)}} \rightleftarrows \vec{X}_{M\underline{v}_{h(t+1)}}.$$

GUI update (h): The human designer externalizes her/his current 'like' status through the GUI's 'toggle' button of each affected idea (Fig. 8.3)

$$\vec{X}_{M\underline{v}_{h(t+1)}} \rightarrow \hat{X}_{M\underline{v}_h}.$$

The GUI allows the human designer to either mark an idea as 'liked' or unmark it. This mechanism does not force the human designer to "rate" every idea but will function to reinforce a trend in the assistant's ideas generation process that the human designer deems appropriate.

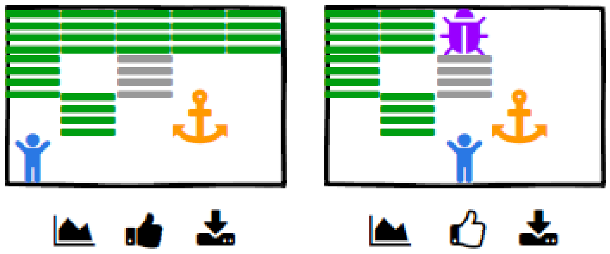


Figure 8.3: Game-level-design-ideas 'liking'. An idea marked as 'liked' (on the left) and an unmarked one (on the right).

8.1.2.2 Promote an idea into current design

Idea selection (h): The human designer h compares the ideas supplied by the assistant c with her/his expectations and current results and selects one of the ideas to be promoted into current design of the video game level

$$\check{x}_{M\underline{a}_h^c} = \left(\left\{\vec{x}_{De_h}, \vec{x}_{Ma_h}\right\} \dashleftarrow\!\!\dashrightarrow \left\{\check{x}_{M\underline{a}_h^c}, \check{X}_{M\underline{v}_{h(t)}}, \check{X}_{D\underline{a}_h^c}, \check{X}_{D\underline{v}_h^c}\right\}\right),$$

$$\check{x}_{M\underline{a}_h^c} \dashrightarrow \check{x}_{I\underline{a}_h} \rightleftarrows \vec{x}_{I\underline{a}_h}.$$

Promotion request (h): The human designer indicates his selection through the GUI's 'apply idea' button (See Fig. 8.2), thus triggering the replacement of the current design representation with the design of the selected idea

$$\vec{x}_{I\underline{a}_h} \rightarrow \hat{x}_{I\underline{a}_h}.$$

Promotion initiation (c): The computational assistant interprets the human designer's request along with the actual design of the video game level

$$\left\{\hat{x}_{I\underline{a}_h}, \hat{x}_{Ma}\right\} \subseteq \left\{\check{x}_{I\underline{a}_c^h}, \check{x}_{Ma_c}\right\}.$$

Previous design memorization (c): The computational assistant c secures the previous design (already interpreted) to restore it later if necessary

$$\check{x}_{Ma_c} \dashrightarrow \check{x}_{Ma_{c(t-1)}}.$$

Promotion preparation (c): The computational assistant pulls from the human designer's request the design idea selected by the human designer and prepares to perform the replacement of the current design

$$\check{x}_{I\underline{a}_c^h} \dashrightarrow \check{x}_{M\underline{a}_c} \rightleftarrows \vec{x}_{M\underline{a}_c}.$$

GUI update (c): The assistant c realizes the idea promotion by replacing the current video-game-level design with its human-selected idea

$$\vec{x}_{M\underline{a}_c} \rightarrow \hat{x}_{Ma}.$$

8.1.2.3 Restore design

Restoration decision (h): The human designer h compares the ideas supplied by the assistant c with her/his expectations and current results and decides to restore the design to the state previous to applying an idea provided by the computational agent c

$$\check{x}_{I_h} = \left(\left\{\vec{x}_{De_h},\, \vec{x}_{Ma_h}\right\} \dot{\leftrightarrow} \left\{\check{x}_{M\underline{a}^c_h},\, \check{X}_{M\underline{v}_{h(t)}},\, \check{X}_{D\underline{a}^c_h},\, \check{X}_{D\underline{v}^c_h}\right\}\right),$$

$$\check{x}_{I_h} \rightleftarrows \vec{x}_{I_h}.$$

There could be several reasons why the human designer decides to restore her/his design; however, a frequent use case could be when previously, the human designer applied an idea supplied by the assistant in order to be able to generate the playable version of the game and perform a more detailed playtesting-based analysis and then return to the previous state.

Restoration request (h): The human designer asks her/his assistant to restore the design state prior to the last idea applied through the GUI's 'undo' button (See Fig. 8.2)

$$\vec{x}_{I_h} \rightarrow \hat{x}_{I_h}.$$

Previous design recall (c): The computational assistant interprets the human designer's request, recalls the previously backed-up design and prepares to override the current design

$$\hat{x}_{I_h} \underset{\rightarrow}{\subset} \check{x}^h_{I_c} \overset{\dashrightarrow}{\dashrightarrow} \check{x}_{Ma_{c(t-1)}} \rightleftarrows \vec{x}_{Ma_{c(t-1)}}.$$

GUI update (c): The assistant c restores the previous game-level design into the GUI game editor

$$\vec{x}_{Ma_{c(t-1)}} \rightarrow \hat{x}_{Ma}.$$

8.1.3 Brainstorming session subsequent c-creative turn(s)

$$\begin{aligned}\Big\{ \ldots,\, &\left\{\hat{x}_{Ma},\, \hat{X}_{M\underline{a}_c},\, \hat{X}_{M\underline{v}_h},\, \hat{X}_{D\underline{a}_c},\, \hat{X}_{D\underline{v}_c}\right\} \underset{\rightarrow}{\subset} \left\{\check{x}_{Ma_h},\, \check{X}_{M\underline{a}^c_h},\, \check{X}_{M\underline{v}_h},\, \check{X}_{D\underline{a}^c_h},\, \check{X}_{D\underline{v}^c_h}\right\},\\ &\left\{\check{x}_{Ma_h},\, \check{X}_{M\underline{a}^c_h},\, \check{X}_{M\underline{v}_h},\, \check{X}_{D\underline{a}^c_h},\, \check{X}_{D\underline{v}^c_h}\right\} \overset{\dashrightarrow}{\dashrightarrow} \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h} \Big\} \subseteq \mathring{\mathscr{P}}_{DM_h},\end{aligned}$$

$$\Big\{ \ldots,\, \vec{x}_{I_h} \rightarrow \hat{x}_{I_h} \Big\} \subseteq \mathring{\mathscr{T}}_{DM_h},$$

$$\begin{aligned}\Big\{ \ldots,\, &\left\{\hat{x}_{I_h},\, \hat{x}_{De_h},\, \hat{x}_{Ma},\, \hat{X}_{M\underline{v}_h}\right\} \underset{\rightarrow}{\subset} \left\{\check{x}^h_{I_c},\, \check{x}^h_{De_c},\, \check{x}_{Ma_c},\, \check{X}_{M\underline{v}^h_c}\right\},\\ &\left\{\check{X}_{M\underline{v}^h_c},\, \check{X}_{D\underline{v}_c},\, \check{X}_{M\underline{a}_c}\right\} \overset{\dashrightarrow}{\dashrightarrow} \check{X}_{D\underline{v}'_c},\\ &\left\{\check{x}^h_{De_c},\, \check{x}_{Ma_c},\, \check{X}_{M\underline{a}_c},\, \check{X}_{D\underline{v}'_c},\, \check{x}^h_{I_c}\right\} \rightleftarrows \Big\{\vec{x}^h_{De_c},\, \vec{x}_{Ma_c},\, \vec{X}_{M\underline{a}_c},\, \vec{X}_{D\underline{v}'_c},\, \vec{x}^h_{I_c}\Big\} \Big\} \subseteq \mathring{\mathscr{P}}_{DM_c},\end{aligned}$$

$$\Big\{ \ldots,\, \Big\{\vec{x}^h_{De_c},\, \vec{x}_{Ma_c},\, \vec{X}_{M\underline{a}_c},\, \vec{X}_{D\underline{v}'_c},\, \vec{x}^h_{I_c}\Big\} \Rightarrow \vec{X}_{M\underline{a}_{c(t+1)}} \Big\} \subseteq \mathring{\mathscr{T}}_{DM_c}$$

1) **Brainstorming session next round request (h):** At any time during a brainstorming session, the human designer h may ask the computational assistant c to provide a new set of game-level design ideas

$$\left\{\hat{x}_{Ma},\ \hat{X}_{M\underline{a}_c},\ \hat{X}_{M\underline{v}_h},\ \hat{X}_{D\underline{a}_c},\ \hat{X}_{D\underline{v}_c}\right\} \subseteq \left\{\check{x}_{Ma_h},\ \check{X}_{M\underline{a}_h^c},\ \check{X}_{M\underline{v}_h},\ \check{X}_{D\underline{a}_h^c},\ \check{X}_{D\underline{v}_h^c}\right\} \dashrightarrow \check{x}_{I_h} \rightleftarrows \vec{x}_{I_h}.$$

2) **GUI update (h):** h externalizes her/his request via the ‘bulb’ action button (Fig. 8.2)

$$\vec{x}_{I_h} \to \hat{x}_{I_h}.$$

Beginning on the second time the human designer requests the assistant to provide a batch of ideas, the $\hat{x}_{I_h}$ request now includes parameter values intended to influence (‘tweak’) the assistant’s creative behavior. These values are, in terms of EC, exogenous parameters of the specific underlying algorithms that can be set by the human designer through controls provided by the GUI (Fig. 8.4) depending on the current underlying EA (globally set in the system settings). From the EC perspective, this exogenous parameters regulation mechanism is crucial to compensate for the reduction in evolutionary speed that results after adapting the algorithms to be interactive.

3) **Brainstorming session current state interpretation (c):** c interprets the human request to generate the next batch of ideas and triggers the interpretation of reference concepts (actual design at the game mechanics level and the expected game dynamics) and the human-based evaluations $\hat{X}_{M\underline{v}_h}$ (if any)

$$\left\{\hat{x}_{I_h},\ \hat{x}_{De_h},\ \hat{x}_{Ma},\ \hat{X}_{M\underline{v}_h}\right\} \subseteq \left\{\check{x}_{I_c^h},\ \check{x}_{De_c^h},\ \check{x}_{Ma_c},\ \check{X}_{M\underline{v}_c^h}\right\}.$$

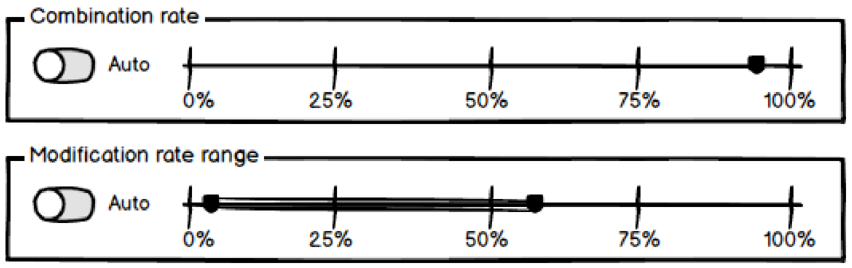
4) **Human-based evaluation integration (c):** c re-computes the evaluation (fitness) of all the ideas it generated last time (not only the ones it submitted to the human designer) through a weighted sum of its self-evaluation and the evaluation of the human designer. For the human-based evaluation, the nominal value (before weighting) is 1 when the idea is marked as ‘liked’ or 0 otherwise. In our experiments, 0.85 and 0.15 were used as weights for self- $\check{X}_{D\underline{v}_c}$ and human-based- $\check{X}_{M\underline{v}_c^h}$ evaluation, respectively. In terms of its underlying EAs, the assistant is just re-computing its current population fitnesses before applying its selection and variation operators. This mechanism has a penalizing effect (in this case, by 15%) on the assistant’s self-evaluation if an idea has not been marked as ‘liked’ by the human designer or has not even been selected to be presented

$$\left\{\check{X}_{M\underline{v}_c^h},\ \check{X}_{D\underline{v}_c},\ \check{X}_{M\underline{a}_c}\right\} \dashrightarrow \check{X}_{D\underline{v}_c'}.$$

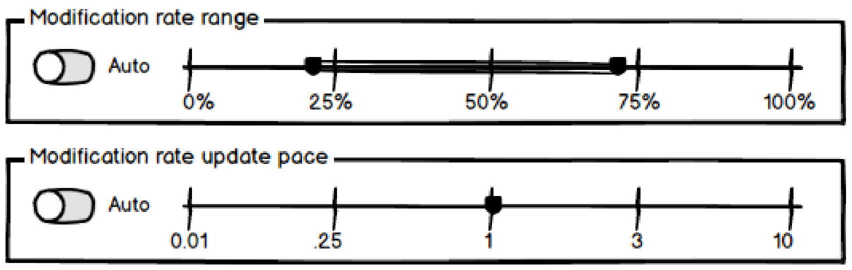
The resulting $\check{X}_{D\underline{v}_c'}$ set of concepts is, in terms of EC, the vector of the population evaluations (fitnesses).

5) **New ideas production preparation (c):** c collects the reference concepts at game dynamics and mechanics levels in addition to, in terms of its underlying EAs, the current population $\check{X}_{M\underline{a}_c}$, evaluations $\check{X}_{D\underline{v}_c'}$, and specific algorithm exogenous parameters embedded into $\check{x}_{I_c^h}$

$$\left\{\check{x}_{De_c^h},\ \check{x}_{Ma_c},\ \check{X}_{M\underline{a}_c},\ \check{X}_{D\underline{v}_c'},\ \check{x}_{I_c^h}\right\} \rightleftarrows \left\{\vec{x}_{De_c^h},\ \vec{x}_{Ma_c},\ \vec{X}_{M\underline{a}_c},\ \vec{X}_{D\underline{v}_c'},\ \vec{x}_{I_c^h}\right\}.$$



(a) GA tweaking controls. The human designer can 1) set the 'combination rate' that corresponds to the crossover probability value employed by the one-point crossover operator of the GA; and 2) the 'modification rate range' that sets the minimum and maximum values of the mutation probability employed by the fitness-proportional mutation operator of the GA.



(b) ES tweaking controls. The human designer can set the 'modification rate range' and the 'modification rate update pace' that set the minimum and maximum values of the mutation probability and the learning rate, respectively, of the auto-adaptive mutation operator of the ES.

Figure 8.4: Game-level-design-ideas - tweaking controls. The GUI provides the human designer with controls (placed right between the ideas and general buttons areas) for influencing ('tweaking') the assistant's creative behavior for the next batch of ideas (subsequent execution of its synthesis strategies). It is not entirely required for the human designer to manipulate these controls every time she/he requests the assistant for a new batch of ideas, as by default, they are set to automatic mode, thus making the underlying algorithm use default parameter values. The human designer can return these controls to automatic mode at any time (through the switch widgets), enabling the scenario where the human designer needs the assistant to return to a more stable behavior after having influenced, perhaps in excess, the assistant's behavior.

6) **EA-loop-iteration-based synthesis (c):** c produces a new set of video-game-level ideas by applying its underlying EAs selection and variation operators to the current set of ideas (including those previously omitted from externalization)

$$\left\{ \vec{x}_{De}{}_c^h,\ \vec{x}_{Ma}{}_c,\ \vec{X}_{M\underline{a}}{}_c,\ \vec{X}_{D\underline{v}}{}_c^{\prime},\ \vec{x}_I{}_c^h \right\} \Rightarrow \vec{X}_{M\underline{a}}{}_{c(t+1)}.$$

From the EC perspective, this step is equivalent to creating the next generation (i.e., a single loop iteration):

a) **GA-based synthesis:**
   - New ideas (individuals) are created through a one-point crossover operator from two ancestors selected through the fitness proportional selection (FPS) method. Through

the exogenous crossover probability parameter ('combination rate' for the human designer [Fig. 8.4a]), the human designer controls how likely each new idea is generated by combining two existing ideas (or is a clone if not combined). The FPS method helps to increase the probability of selecting highly fitted ancestors, expecting (without guarantee) that the combined idea will also obtain appropriate fitness.

- A fitness proportional uniform mutation operator is then applied to each new individual to introduce small (or significant) changes. The number of changes introduced depends on a generation-wise probability computed inversely proportional to the average generation fitness (i.e., the fitter the generation, the steeply declining probability of mutation, and vice versa) within the range defined by the 'modification rate range' exogenous parameter (Fig. 8.4a), also controllable by the human designer.
- As in population-initialization-based synthesis (step 6 described in Section 8.1.1), a "booster" operator is applied to each individual to make them more suitable to comply with the requirements embedded into the expected game dynamics concept.
- Finally, the new generation is assembled by taking the best (elitist selection) ideas from the current generation and a set of randomly selected new ideas.

b) **Synthesis based on (μ/ρ, λ)-ES:**
- A pool of ancestors is created by selecting the μ fittest individuals (ideas) from the current λ-size generation.
- Each of the λ new individuals is created through the 'dominant' and 'intermediate' recombination operators (Beyer and Schwefel 2002) as follows:
  - A pool of ρ distinct recombinants is sampled from the μ-size ancestors pool.
  - The part of the new individual's genotype that encodes the video game level (all but one gene) is assembled by copying each i^{th} gene from a randomly picked recombinant ('dominant recombination').
  - The gene that encodes the individual's mutation probability endogenous parameter is computed by averaging all recombinant pool's mutation probability gene values ('intermediate recombination').
- The newly created individuals are modified through an auto-adaptive method according to the canonical specification of the (μ/ρ, λ)-ES (ibid.). The part of the genotype that encodes the video game level is altered through a uniform mutation operator where the mutation probability is first determined for each individual (endogenous parameter) through a discrete logistic transformation function (Eq. 8.1, [Li et al. 2013])

$$\varrho' = \frac{1}{1 + \frac{1-\varrho}{\varrho}\exp(-\tau N(0,\ 1))} \tag{8.1}$$

(normalized to the human-controllable $\left[\varrho_{min},\ \varrho_{max}\right]$ 'modification rate range' exogenous parameter [Fig. 8.4b]), that computes the new mutation probability ϱ' from the current mutation probability ϱ and the adaptability rate or 'modification rate update pace' exogenous parameter τ (also set by the human designer [Fig. 8.4b]).

- Finally, as in GA-based synthesis, the "booster" operator is applied to each new idea.

This step is followed by steps 7–14 described in Section 8.1.1. Subsequently, either a new c-creative turn described in this section or one or more of the alternatives described in Section 8.1.2 are performed.

When implementing the the new system capabilities, some of the parameters of the underlying EAs were empirically determined and set for the subsequent user testing and evaluation of the new features:

a) **Common parameters:**
   - Population size (λ in ES): 17
   - Initialization fluctuating mutation range: $[.02, .25]$

b) **GA:**
   - Population size: 17
   - Elitism size: 1 (guarantees preserving the best idea)
   - Crossover probability ('combination rate' when set to 'Auto'): 0.9
   - Mutation probability range ('modification rate range' when set to 'Auto'): $\left[10^{-4},\ 0.6\right]$

c) $(\mu/\rho,\ \lambda)$-**ES:**
   - Ancestors pool size μ: 4
   - Recombinants pool size ρ: 4
   - Mutation probability range $\left[\varrho_{min},\ \varrho_{max}\right]$ ('modification rate range' when set to 'Auto'):

$$\left[\frac{1}{3 \cdot \text{genotype length}},\ 0.5\right] \quad \text{(as suggested in [Li et al. 2013])}$$

   - Adaptability rate τ ('modification rate update pace' when set to 'Auto'): 1

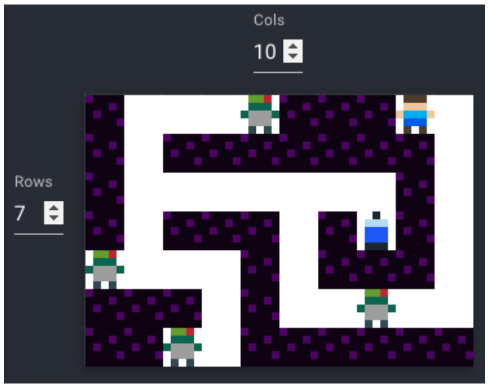
8.2 User evaluation

In order to assess the new system features, a qualitative evaluation was conducted by a professional UI/UX designer and puzzle-video-game enthusiast.

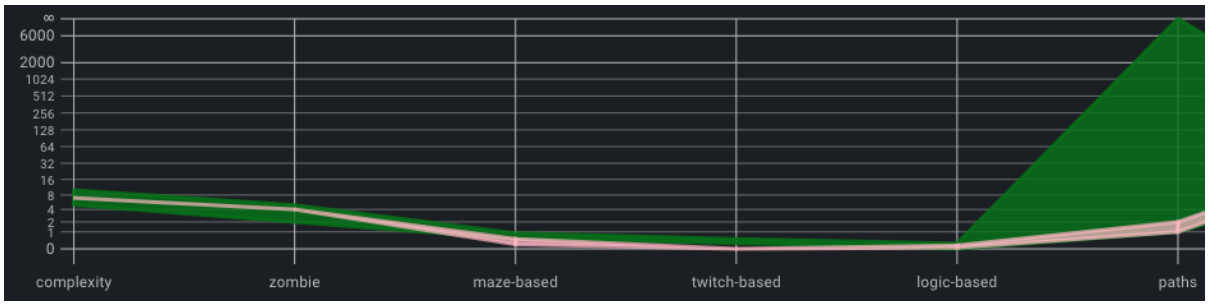
The evaluation consisted of conducting five design sessions, a free design session (familiarization and exploration), and four structured ones:

8.2.1 Structured evaluation sessions' specific results

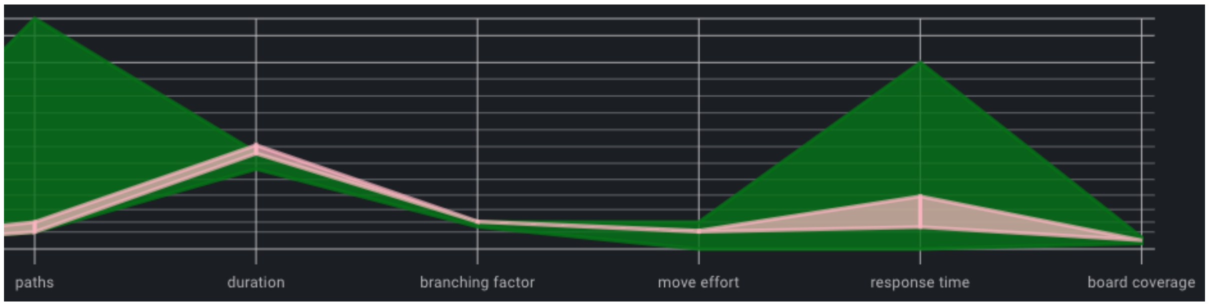
1. **Unassisted session starting from an empty game space (blank game-level layout).** In this session, the designer started from the existing game design 'Zombiekiller' (Design 5 in Section 7.2.1) with an empty game level, establishing the scenario in which both the design spaces and the expected design concepts are fully formulated/synthesized at the experience and game dynamics levels and partially formulated/synthesized at the game mechanics level. Upon completion of the session, the designer produced the design in Fig. 8.5. During the session and the following moments after its end, the human designer



(a) Video game level (game mechanics design level)



(b) Expected vs actual game dynamics (1/2)



(c) Expected vs actual game dynamics (2/2)

Figure 8.5: Session 1 - Final video game design.

made some remarks and brief explanations regarding her game-level construction process.

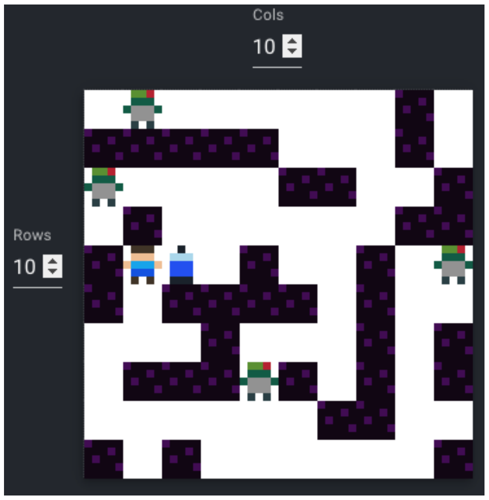
Concerning the laying out (design decisions) sequence, the designer mostly worked in layers from back to front. The designer explained that she always starts by imagining the maze shape, then she starts thinking about the position of the start and end objects, trying to put them as far as possible, drawing the most intricate possible path between them, which usually triggers adjustments in the maze structure. She then relocates the bottle (end object) to a more confined position. Finally, she decides on the number and positions of the zombie objects in a way that forces the player to make intricate and long runs to find a way to the bottle. The designer stated that the maximum size limitation prevented making a more complex labyrinth, and because of this constraint, framing the maze was avoided, although it would typically have been done.

2. **Creatively assisted session starting from an empty game space.** In this session, the human designer also started from the existing game design 'Zombiekiller' with an empty game board. This time, the designer requested and received creative assistance from the GA-based computational agent to produce the design depicted in Fig. 8.6.

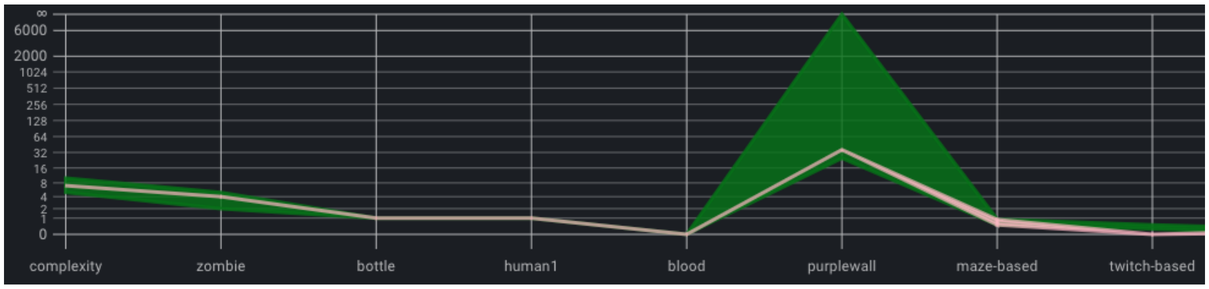
   In this session, the human designer reported that at the beginning, she was surprised to see many players (start tokens), bottles (end tokens), and blood objects in the ideas provided by the assistant while she was expecting to see one start, one end, and no blood objects. However, after some analysis, she solved the problem by adding and properly restricting some dimensions in the game dynamics level (more on this later in this chapter).

   After three turns, she also observed very few walls on the boards, which she immediately identified as a problem that would be solved by adding and restricting the appropriate dimension (wall counter) at the dynamics level. Once the corrections were made (she set the minimum number of walls to 25), the designer perceived a positive change after three turns. However, she perceived that between one turn and the next, there were no significant changes (she did not get the amount of wall she expected), so she increased the 'modification rate' (mutation probability range) to a [.7, 1] range (something that could be considered very high in a standard, no-interactive, evolutionary algorithm). She obtained the desired result with this action and immediately returned the assistant's behavior to 'Auto' mode (during the familiarization session, the designer realized that just one shift could be enough to get the agent out of a possible impasse or to steer it in a different direction). Only at this point did the designer use the liking feature and mark as 'liked' two design ideas. Then gave one more turn to the artificial agent, selected one of the proposed ideas from this final turn, and promoted it into the main design, making only a few final manual adjustments.

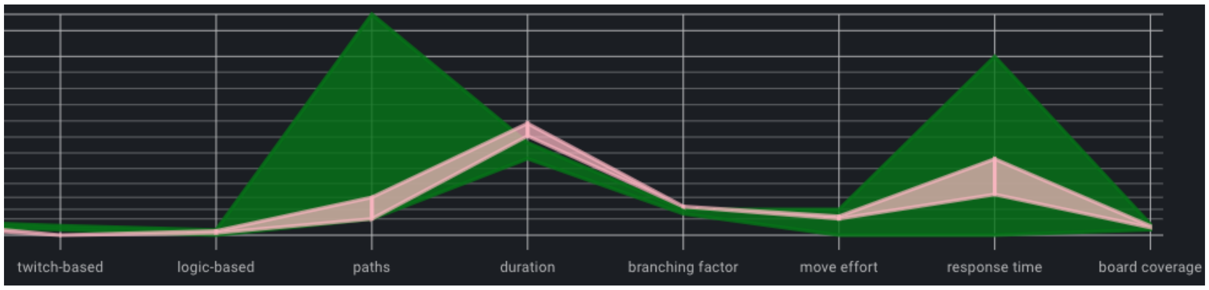
   The human designer explained that when dealing with labyrinths, one has the established idea that mazes have very obvious or marked trajectories (when visualized from the top in a 2D view) and that when designing them, one tends to frame the labyrinth and leave only the entrance and exit cells free. However, after examining the ideas proposed by the assistant, she realized that making the trajectories less obvious (with more spaces) increases the complexity of the maze, something that was not obvious or usual to observe (a contribution of the computational agent to the human designer mindset). The human designer stated that after studying and understanding this behavior, she purposely exploited the computational assistant's "technique" to create even more confusing mazes with less



(a) Video game level (game mechanics design level)



(b) Expected vs actual game dynamics (1/2)



(c) Expected vs actual game dynamics (2/2)

Figure 8.6: Session 2 - Final video game design.

obvious trajectories. She also mentioned that designing mazes following this technique is difficult when starting from a blank board precisely because the trajectories are not evident for the designer herself.

In an idea proposed by the computational assistant, it placed the start and end objects right next to each other. According to the human designer, at first glance, this would seem silly (even incompetent), but given the rules of the game, this scenario would make the player perform even more moves because she/he/it first had to go and eliminate the zombies (spread along the board) and would always be forced to return to the same point to solve the puzzle (again more moves).

The human designer informed that the computational assistant sometimes places objects under other objects, thus hiding them (to a human player). Although not incorrect from the point of view of the rules, this could interfere with the game experience; for example, according to the layers visible to the player, the player could judge that the current visible state should be the ending state, but because there are hidden objects, the end conditions may not be activated, leading the player to believe that there is something else to do and making her/him feel frustrated for not finding anywhere the missing move (which does not exist in reality). Regarding this situation, the human designer stated that as a future improvement to the system's tool, the human designer should be allowed to introduce more complex structural rules (not necessarily required by PuzzleScript).

3. **Unassisted session starting from an existing design (nonempty game space).** In this session, the human designer was requested to try to improve the original design of the 'CookieMonster' game (Design 2 in Section 7.2.1) without the creative assistance of the computational agent. Upon completion of the session, the human designer produced the design in Figs. 8.7 and 8.8.

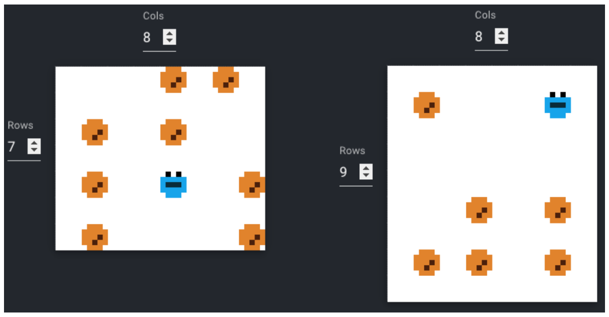
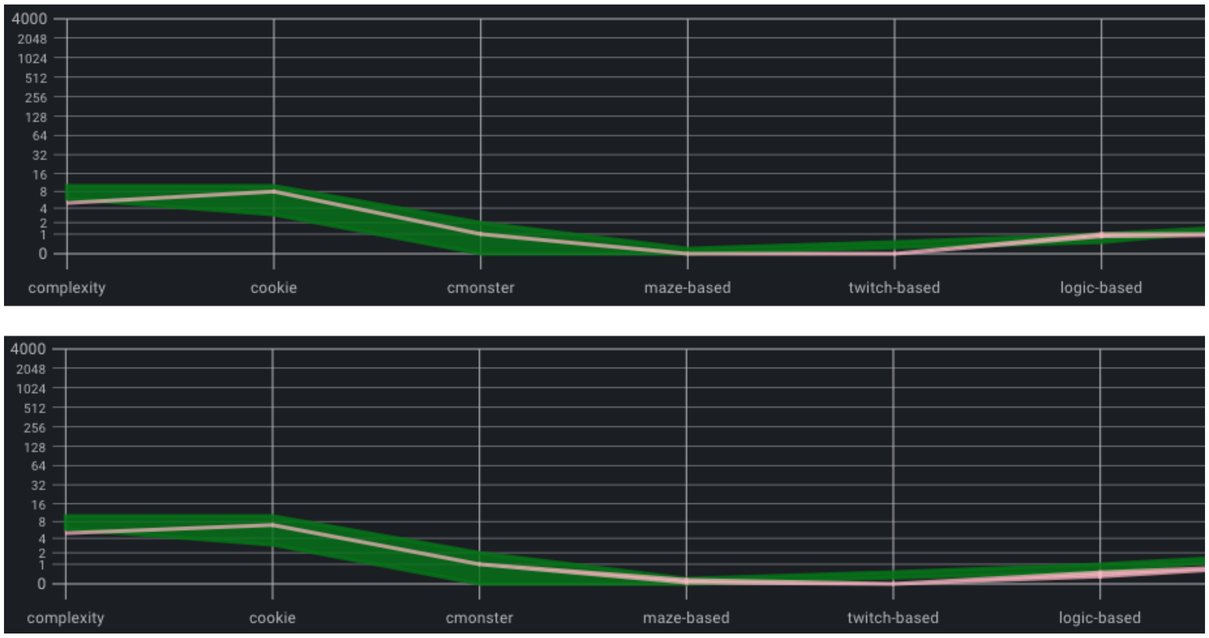


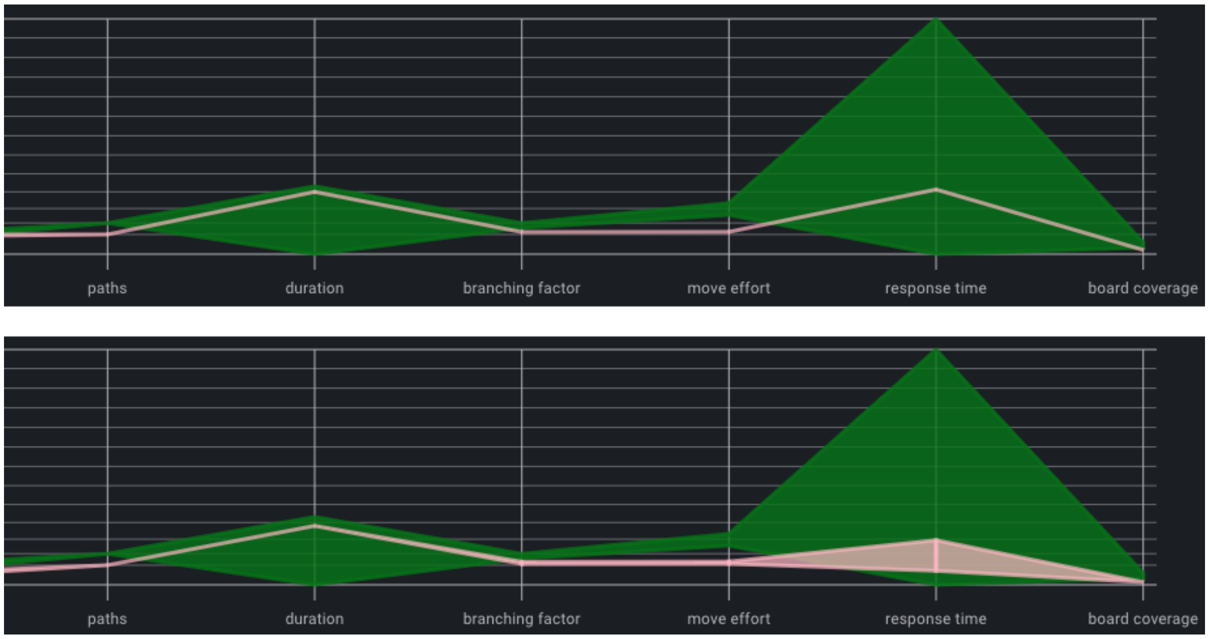
Figure 8.7: Session 3 - Original vs redesigned video game (game mechanics).

According to the human designer, it was a complicated task to try to obtain a more interesting and complex game than the original, especially not to make the solution to the puzzle so obvious. Notably, in this particular game, the human designer resorted more to reviewing the analysis and evaluation conducted by the assistant (pre-existing capabilities) and playing the game herself, whether to find a solution or to grasp the obtained game dynamics and experience. The designer stated that she used this technique more because the solutions for this game are difficult to visualize mentally and that after making the first modifications, she judged that the game would have a solution when it did not, and she had to modify the design again. Finally, the human designer expressed that she had not achieved a better game than the original (already the 'optimal' version she had come up with) only by modifying the game level, but she probably could reach a better design with more time and effort.

4. **Creatively assisted session starting from an existing design.** In this session, the human designer tried again redesigning (for better) the 'CookieMonster' game, this time with the ideas provided by the computational agent (in its $(\mu/\rho,\ \lambda)$-ES mode). Upon completion of



(a) Expected vs actual game dynamics. Original (top) vs redesigned (bottom) (1/2)



(b) Expected vs actual game dynamics. Original (top) vs redesigned (bottom) (2/2)

Figure 8.8: Session 3 - Original vs redesigned video game (game dynamics).

the session, the human designer, with the computational agent's creative support, produced the design depicted in Figs. 8.9 and 8.10.

During this last testing session, the human designer observed that almost no ideas had a solution. From previous experience, the designer reviewed the design at the game dynamics level and detected an error in the specification of the 'duration' dimension (it was defined in the range [0, 10]), obviously discovering that a 0 value made no sense (it makes

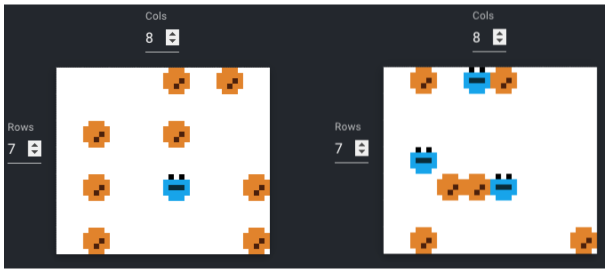
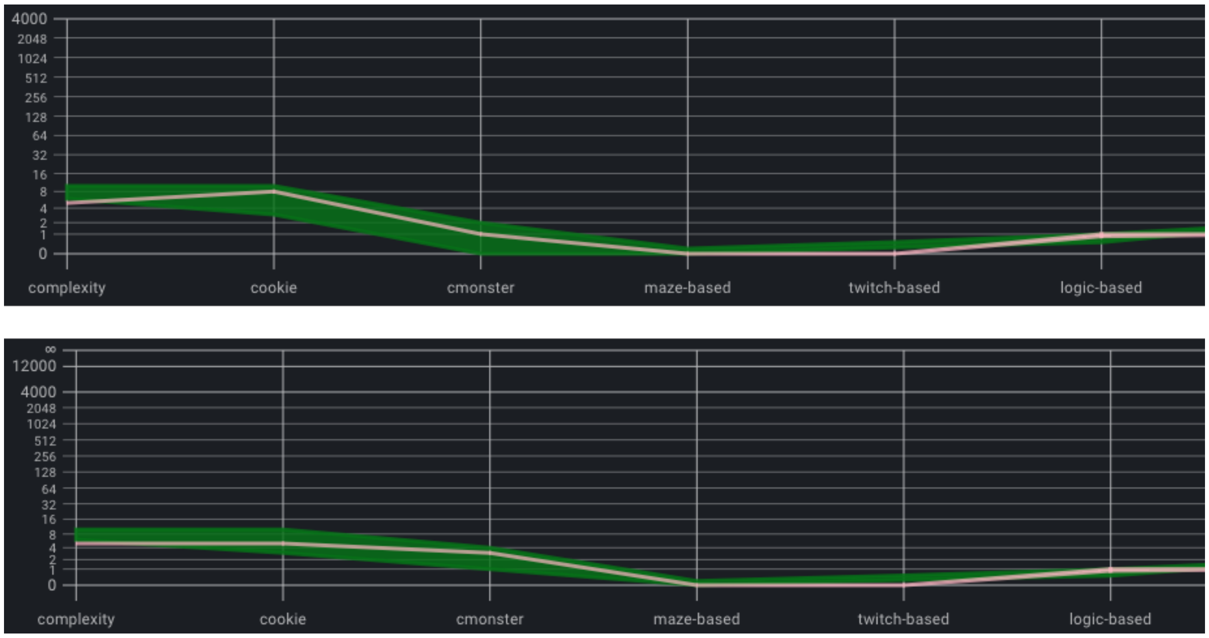


Figure 8.9: Session 4 - Original vs redesigned video game (game mechanics).

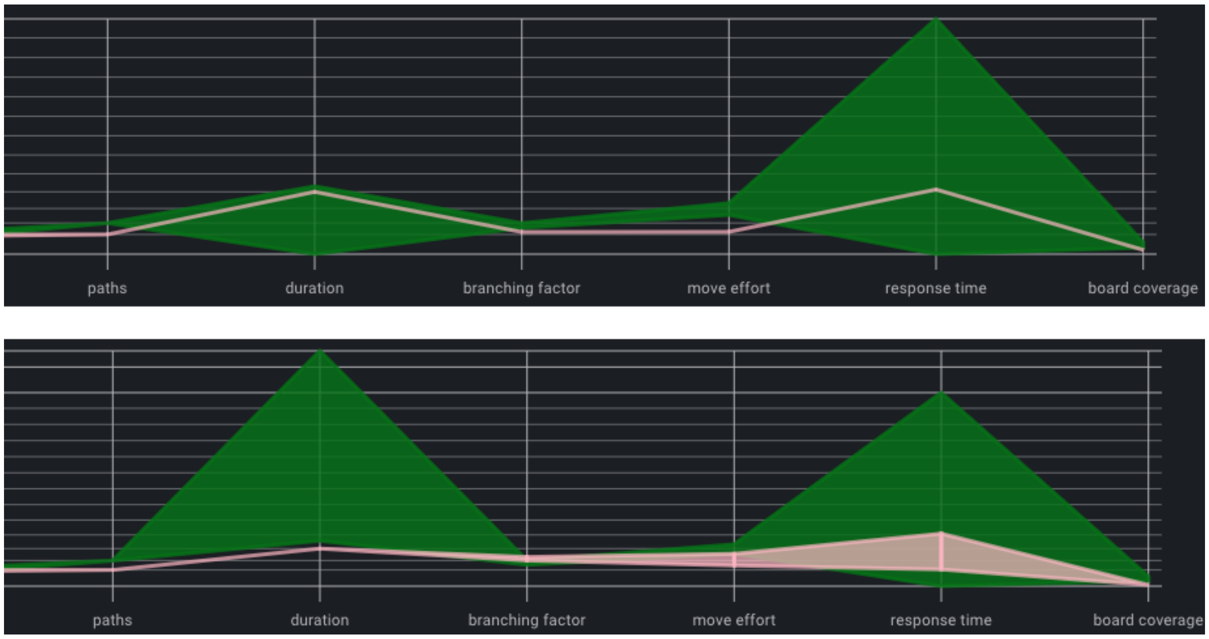
no sense for a game to have no solution) so she again corrected the situation from a higher design level. After evolving the design a few more turns, the human designer noticed ideas with up to 2 monster-type objects. Having two player avatars on the board fascinated the human designer as it opened up the possibility of the player having not only one object but a “team” under the player’s control. The human designer said that she had specified up to 2 player avatars in the original game design but had never really tried this possibility and had “automatically” (an unconscious decision) placed only one avatar at the beginning of the game.

As she found this ‘control a team’ feature interesting, she decided to redefine the design to allow ≥ 1 monster-type objects to see how far this idea could go. A few turns later, the human designer marked as ‘liked’ two ideas with three avatars on the board. After a couple more turns, however, level ideas with too many avatars (> 4) began to appear, which seemed to the designer excessive, so she again modified the design at the game dynamics level, leaving the number of avatars constrained to the range $[1, 4]$. At that moment, she decided to apply the computational agent’s idea to be able to play it by herself and experience the dynamics of controlling a team (Fig. 8.11). The human designer expected that this feature would allow the player to perform simultaneous movements, i.e., more than one or even all avatars would move at the same time; however, although this behavior was indeed present in some ideas (if their object layout allowed it), it was not always the case. Sometimes, many avatars did not move while only one moved, or even sometimes, some avatars did not move at all (they were completely useless). Even with this situation, the designer stated that without a doubt, her –seeing multiple player avatars moving simultaneously– expectation came from the idea generated by the computational agent, and it was just a matter of selecting one of the designs she found most interesting and modifying it by hand in order to provoke as many simultaneous movements as possible.

At this point, the designer decided to continue with the brainstorming process, marked ideas that she liked, and continued for three more turns until the computational agent came up with an idea with four avatars in place that she liked and promoted into the actual design. The human designer explained that she had difficulties not only visualizing but finding by herself (by playing) the solutions that, according to the analysis of the computational agent, a game level had (in some cases, she had to play a level up to 7 times to find the shortest solutions). In order to continue experimenting, the designer decided to continue evolving the design some more turns until the process reached a design that was to her liking, for which, however, she could not find the shortest solution reported by the



(a) Expected vs actual game dynamics. Original (top) vs redesigned (bottom) (1/2)



(b) Expected vs actual game dynamics. Original (top) vs redesigned (bottom) (2/2)

Figure 8.10: Session 4 - Original vs redesigned video game (game dynamics).

computational agent. As in the previous session, the human designer also reviewed the analysis and evaluation generated by the computational agent more frequently since she explained that in this particular game, it is more difficult to visualize the solutions and the board's coverage.

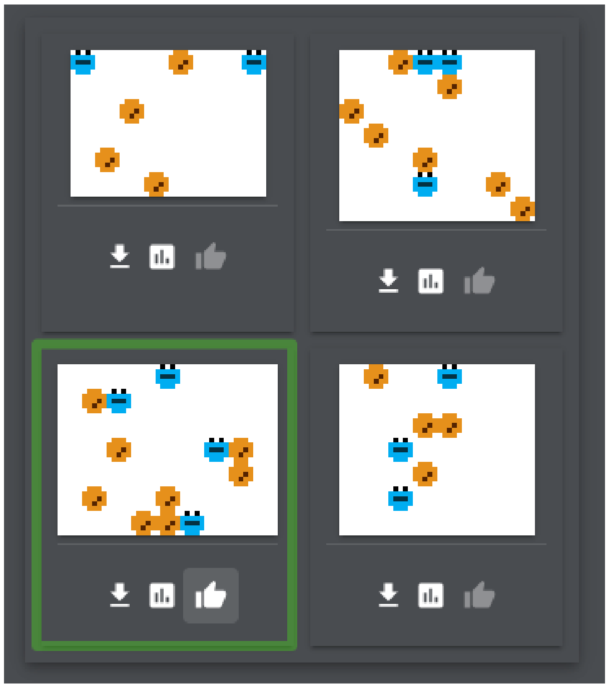


Figure 8.11: Session 4 - Selected idea to test 'controlling a team'.

8.2.2 Overall evaluation results

Throughout the collaborative sessions with the artificial agent (including the first familiarization session), some unexpected patterns and behaviors emerged that led the human designer to make design decisions not only at the level of game mechanics but also at the level of game dynamics and to become aware of some essential aspects for the design process that are typically taken into account by the designer but unconsciously or automatically, which although help to streamline the process, can bring as a disadvantage the possible generation of design fixations or conventional practices that end up limiting her creative scope.

For example, according to the game rules established by the human designer in the 'Zombie' game, when a zombie object stands on a bomb object, the zombie is replaced by a skull-type object (to represent the dead zombie visually). This same situation occurs in the 'ZombieKiller' action maze game, where a zombie killed by direct player action leaves a bloodstain as a trace (a blood object replaces the zombie object). Because of these rule patterns, an unexpected behavior emerged when the ability to generate and contribute level ideas was incorporated into the artificial agent. The human designer observed that some of the ideas contributed by the artificial agent included skull-type (or blood-type) objects placed in the game space (initially). This situation does not make sense (according to the mental concept expected by the human designer) since skull- and blood-type objects should appear as a consequence of the execution of the game rules and not at the beginning of the game (game board initial state). The human designer then reviewed the specifications of the expected game dynamics concept and could realize that there was no constraint preventing the artificial agent from placing skull- and blood-type objects on the board, i.e., the design concept did not have the counter dimension for the skull- and blood-type object. Then, the human designer added the appropriate counter-type dimension(s) to the expected dynamics concept and restricted its/their value(s) to precisely 0 (zero) instances. With this, the human designer automatically constrained the design space at the game mechanics level by modifying the design at the next higher level (game dynamics). As a result, the artificial agent generated level ideas without placing unexpected objects in the initial game space (Fig. 8.12). While a constraint like this would seem evident to the human designer (not placing skull-type objects in the startup), it was not to the computational assistant. The human designer realized the importance of externalizing her design concepts (communicating design decisions) through the GUI as accurately and thoroughly as possible, thus helping herself to get conscious (literally passing information from the unconscious to the conscious part of her mind) about crucial design aspects.

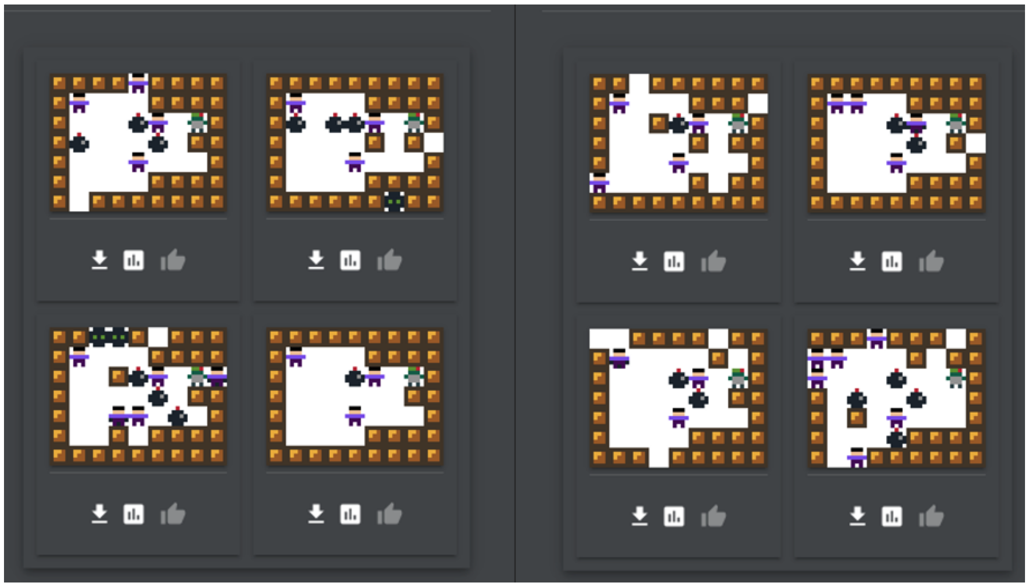


Figure 8.12: Free session - Mechanics level design based on dynamics level. Unexpected skull-type objects (left) vs expected design pattern (right) after modifying the expected game dynamics concept.

Another unexpected behavior emerged when the artificial agent put several objects (> 1) on the game board that the human designer had considered as avatars of the player or as unique tokens that triggered a game win condition. In general, this situation caused the human designer to update the design of the expected dynamics and the game dynamics design space by including counter dimensions that limited these object types to precisely one (or any other required specific number). However, in the case of the 'CookieMonster' game, the human designer realized that having more than one player avatar was very interesting as it brought to the game the feature that the player could control not just one character but a team of characters and have to take advantage of that behavior. This case is an example where the human designer explicitly mentioned that the ideas provided by the artificial agent allowed her to be more creative by discovering new gameplay possibilities. In her mind, she was fixed on considering only one player avatar, and the artificial agent's ideas helped her to break this fixation (go out of the box as she explicitly stated).

A further example of how the ideas generated by the computational agent allowed the human designer to rethink pre-conceived ideas about the structure of a game was presented in maze-like games. Within the evolution of ideas proposed for the 'Alice'[3] game, the human designer observed that the artificial agent sometimes evolved its ideas into board layouts with short, widely dispersed walls (Fig. 8.13). Initially, this situation seemed counter-intuitive to the human designer per her maze design procedure, which consisted of placing continuous lines of wall objects from which precise trajectories emerge. However, upon reviewing the comparative evaluation of these ideas, she observed that, in some cases, the number of steps (duration) to solve these mazes exceeded the number of steps required to solve the base or inspiration maze. After analyzing the situation, the human designer realized that having small and scattered walls made it difficult for the player to visualize continuous trajectories, thus increasing the difficulty not only in solving the maze but also in designing them using that technique (because it is also difficult for the designer to see trajectories). This behavior allowed the human designer to incorporate this new knowledge to design more interesting and complex mazes.

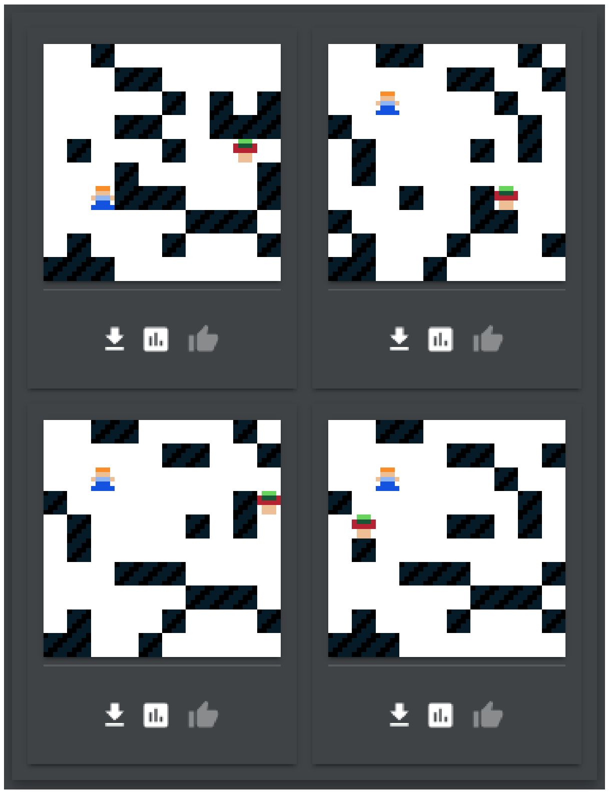


Figure 8.13: Free session - Computational agent's maze design.

The human designer was able to identify that, in some types of games, it is challenging at first glance to estimate whether a level has a solution or not, regardless of whether it looks exciting or not. In these cases, the analysis and evaluation provided by the artificial agent were of great value.

The human designer generally borrowed ideas provided by the artificial agent that she found interesting. She did not seem to want to find perfect ideas, but those that she found interesting and different from what she was used to getting for herself (surprise). Once these ideas were

[3]A reduced version of the original game was created to comply with the maximum size restriction.

applied, she would modify them to achieve her purpose. This behavior could exemplify that the collaborative scheme could have influenced the creative performance of the human designer and that some signs of creative teamwork were present (the contribution of one agent influenced the conceptualization of the other, ideas were supplemented or augmented, and solutions were built with contributions from both).

The overall opinion of the human designer was that working with the computational agent's assistance effectively allowed her to obtain results that she could not obtain without the assistance. According to her, the collaborative sessions allowed her to understand better how her designs relate and are affected from one design level to another (as described in the framework), (i.e., how a specification at a higher level affects the outcome at the lower level and vice versa). Finally, the human designer tried to explain (in her terms) the artificial agent's behavior as a strategy to grasp why the assistant had arrived at one idea or another. Regarding the idea generation strategies, the human designer could understand roughly how they worked, but mainly she could interpret some aspects of the algorithms' behavior. (e.g., to induce ”divergent thinking” by increasing the modification rate, to detect periods of ”lack of inspiration” or ”fixation” –several turns with no apparent progress–, to figure out how to get the computational agent out of that situation, to induce algorithm stabilization to allow evolving a promising idea).

9 Building an autonomous CCDS

9.1 Design levels

For purposes of demonstrating the framework's capability to describe and model not only collaborative but also autonomous systems, we present an autonomous system designing schema to design new game rules (an essential type of game mechanics) for existing games. For this schema we are only considering two design levels, game dynamics D and game mechanics (game rules) M levels to originate the dynamics-mechanics design layer DM.

In this designing schema, new game rules are created from expected game dynamics. Then, actual game dynamics are derived through automated playtesting on a puzzle video game built from an existing game whose game rules have been replaced with the newly created rules. (i.e., a new variant of the original game) (Fig. 9.1).

9.1.1 Game dynamics D level

Lim and Harrell (2014) developed a method to evaluate the playability, feasibility, and validity of rulesets (game mechanics) of PuzzleScript games. Playability is assessed via a set of intrinsic dynamic design dimensions called ruleset heuristics, while both feasibility and validity are playtest-based design dimensions. These dimensions, together with the *duration* dimension inherited from the primary schema, constitute the working set of dynamic design dimensions for modeling expected and actual design concepts produced at this level:

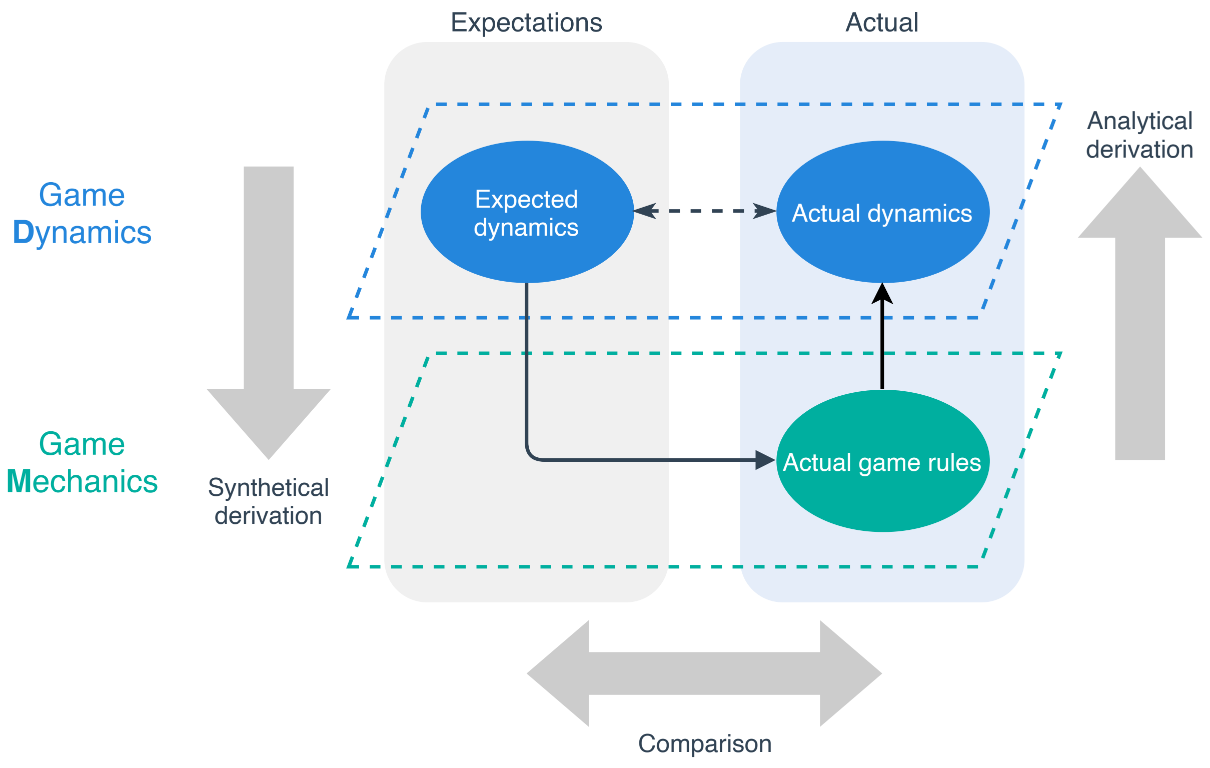


Figure 9.1: Puzzle video game design autonomous CCDS designing schema. Depicts the main design concepts production flow.

$$\Delta'_{\mathrm{D}} := \left\{ \ddot{\delta}_{\mathrm{D}_{rh}},\ \dot{\delta}_{\mathrm{D}_{rf}},\ \dot{\delta}_{\mathrm{D}_{rv}},\ \dot{\delta}_{\mathrm{D}_{d}} \right\} \cup \left\{ \dot{\delta}_{\mathrm{D}_{pr}},\ \dot{\delta}_{\mathrm{D}_{or}},\ \dot{\delta}_{\mathrm{D}_{pl}},\ \dot{\delta}_{\mathrm{D}_{pm}},\ \dot{\delta}_{\mathrm{D}_{ud}} \right\}.$$

***Ruleset heuristics $\ddot{\delta}_{\mathrm{D}_{rh}}$*:** A compound design dimension that represents the weighted average of scores from each ruleset heuristic[1]

$$\ddot{\delta}_{\mathrm{D}_{rh}} := \left\langle \left\langle \dot{\delta}_{\mathrm{D}_{pr}},\ \dot{\delta}_{\mathrm{D}_{or}},\ \dot{\delta}_{\mathrm{D}_{pl}},\ \dot{\delta}_{\mathrm{D}_{pm}},\ \dot{\delta}_{\mathrm{D}_{ud}} \right\rangle,\ L^2 \right\rangle,$$

where L^2 is the Euclidean distance representing the distance between games from the perspective of the playability enabled directly from the game ruleset.

***Player in ruleset $\dot{\delta}_{\mathrm{D}_{pr}}$*:** An indicator for the presence of a *player* game object in the game ruleset

$$\dot{\delta}_{\mathrm{D}_{pr}} := \langle \{0,\ 1\},\ L^1 \rangle,$$

where L^1 is the Manhattan distance. The absence of player-type game objects in the ruleset prevents interaction (at all) with the human or computational player.

[1]Design dimensions in italics are based on or borrowed from the set of dimensions described in (Lim and Harrell 2014)

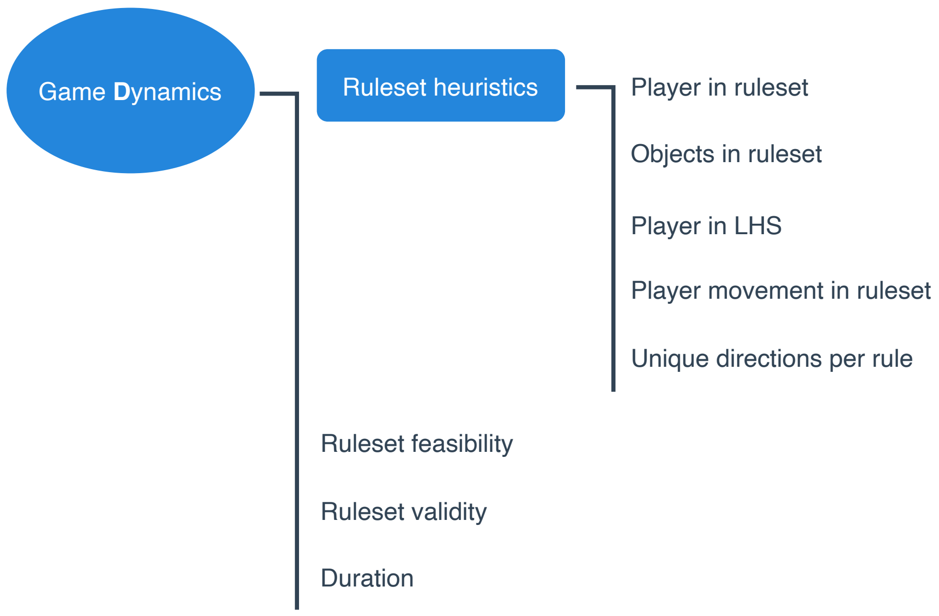


Figure 9.2: Game dynamics design dimensions of subsidiary designing schema

Objects in ruleset $\dot{\delta}_{D_{or}}$**:** The proportion of non-player game objects appearing in its ruleset

$$\dot{\delta}_{D_{or}} \coloneqq \langle I,\ L^2 \rangle.$$

Lim and Harrell hypothesize that rulesets affecting more objects in the game should increase playability; however, this might not be necessarily true for pure, simple maze-based puzzles where non-player game objects are positioned just as static obstacles for the player.

Player in LHS $\dot{\delta}_{D_{pl}}$**:** An indicator for the presence of a player-type game object precisely on the left-hand side of a game rule

$$\dot{\delta}_{D_{pl}} \coloneqq \langle \{0,\ 1\},\ L^1 \rangle.$$

The presence of a player game object on the left side of a game rule is a sign that the player can perform actions and, perhaps, modify the current game state.

Player movement in ruleset $\dot{\delta}_{D_{pm}}$**:** An indicator for the presence of player movements in the game ruleset

$$\dot{\delta}_{D_{pm}} \coloneqq \langle \{0,\ 1\},\ L^1 \rangle.$$

According to Lim and Harrell, player movements appearing in rules increase the likelihood of the player both performing and receiving actions.

Unique directions per rule $\dot{\delta}_{D_{ud}}$**:** The reciprocal of the proportion of unique directional moves per rule

$$\dot{\delta}_{D_{ud}} \coloneqq \langle I,\ L^2 \rangle.$$

As stated in (Lim and Harrell 2014), playability could be negatively affected by the presence of erratic directional changes.

Ruleset feasibility $\dot{\delta}_{D_{rf}}$**:** A marker indicating whether or not a game generated from a ruleset and other game mechanics has a solution

$$\dot{\delta}_{D_{rf}} := \langle \{0, 1\}, L^1 \rangle.$$

Ruleset validity $\dot{\delta}_{D_{rv}}$**:** A marker indicating whether or not a game generated from a ruleset and other game mechanics produce runtime errors in the PuzzleScript game engine

$$\dot{\delta}_{D_{rv}} := \langle \{0, 1\}, L^1 \rangle.$$

Duration $\dot{\delta}_{D_d}$**:** (See 5.1.2).

The previous design dimensions define the structure of the $\mathcal{D}_D$ design space that works as the evaluation space for the DM layer:

$$\mathcal{D}_D = \mathcal{E}_{DM} := \left\langle \left\langle \underbrace{\dot{\delta}_{D_{pr}}, \dot{\delta}_{D_{or}}, \dot{\delta}_{D_{pl}}, \dot{\delta}_{D_{pm}}, \dot{\delta}_{D_{ud}}}_{\ddot{\delta}_{D_{rh}}} \right\rangle, \dot{\delta}_{D_{rf}}, \dot{\delta}_{D_{rv}}, \dot{\delta}_{D_d} \right\rangle, \mathcal{D}_D \subseteq \vec{X}_D.$$

9.1.2 Game mechanics M level

The working set of design dimensions in this schema comprises a modified, simplified subset of the primary schema's:

$$\Delta'_M := \left\{ \ddot{\delta}_{M_r} \right\} \cup \left\{ \dot{\delta}_{M_{rm}} \right\} \cup \left\{ \left(\ddot{\delta}_{M_{rc}} \right)_{s,v,w} \right\}_{\substack{s \in \{\text{left}, \text{right}\}, \\ v \in \mathbb{N}, \\ w \in \mathbb{N}}} \cup \left\{ \dot{\delta}_{M_{rcm}}, \ddot{\delta}_{M_{rco}} \right\}.$$

Game rule $\ddot{\delta}_{M_r}$**:** A compound dimension representing a single game rule conformed by two sides, left and right, each side composed of one pattern, each pattern constituted by exactly two rule cells composed of t cell items

$$\ddot{\delta}_{M_r} := \left\langle \dot{\delta}_{M_{rm}}, \left\langle \left(\ddot{\delta}_{M_{rc}} \right)_{s,v,w} \right\rangle_{\substack{s \in \{\text{left}, \text{right}\}, \\ v \in \{1..2\}, \\ w \in \{1..t \in \mathbb{N}\}}} \right\rangle.$$

Game rule modifier $\dot{\delta}_{M_{rm}}$**:** (See Section 5.1.3).

w^{th} **cell item of** v^{th} **cell of** s **side** $\left(\ddot{\delta}_{M_{rc}} \right)_{s,v,w}$**:** A compound dimension representing a single game rule cell item

$$\left(\ddot{\delta}_{M_{rc}} \right)_{s,v,w} := \left\langle \dot{\delta}_{M_{rcm}}, \dot{\delta}_{M_{rco}} \right\rangle.$$

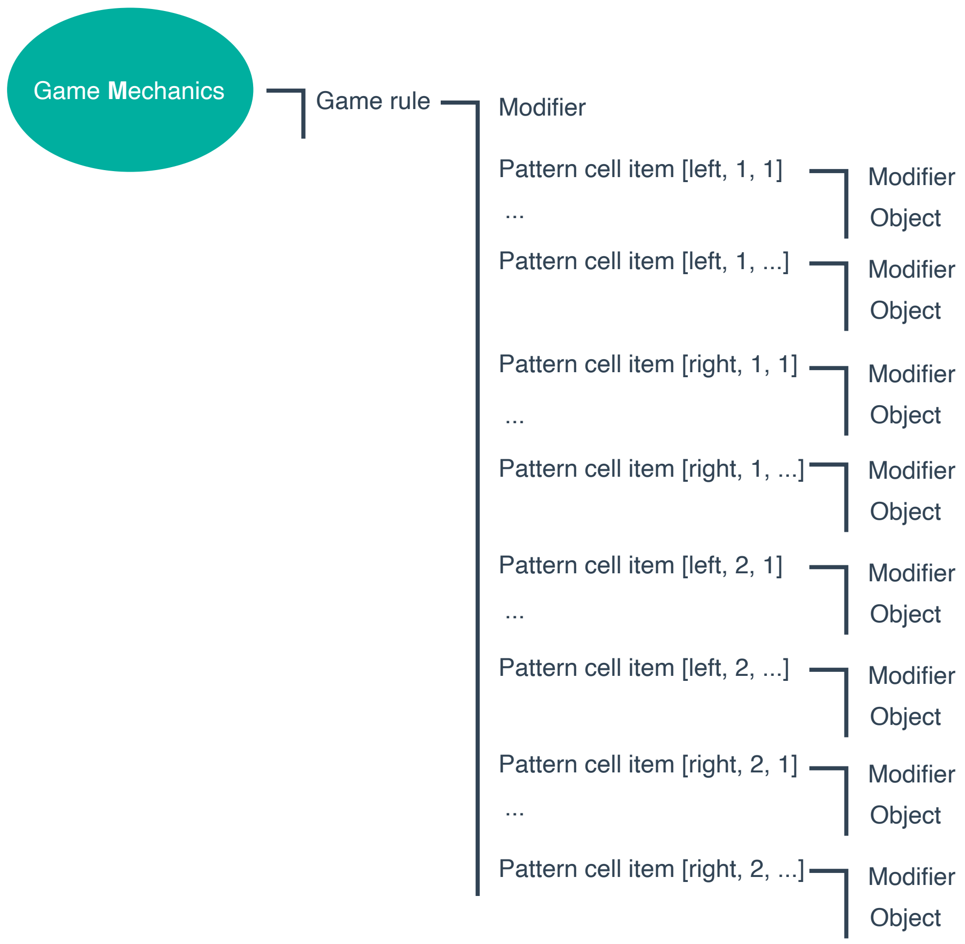

Game Mechanics
Game rule
Modifier
Pattern cell item [left, 1, 1]
...
Pattern cell item [left, 1, ...]
Pattern cell item [right, 1, 1]
...
Pattern cell item [right, 1, ...]
Pattern cell item [left, 2, 1]
...
Pattern cell item [left, 2, ...]
Pattern cell item [right, 2, 1]
...
Pattern cell item [right, 2, ...]
Modifier
Object

Figure 9.3: Game mechanics design dimensions of subsidiary designing schema

Cell item modifier $\dot{\delta}M_{rcm}$: (See Section 5.1.3).
Cell item object type $\dot{\delta}M_{rco}$: (See Section 5.1.3).

These design dimensions give rise to the structure of the $\mathcal{D}_M$ design space that corresponds to the solution space of the DM layer:

$$\mathcal{D}_M = \mathcal{S}_{DM} := \Bigg\langle \underbrace{\Bigg\langle \underbrace{\Big\langle \dot{\delta}M_{rcm}, \dot{\delta}M_{rco} \Big\rangle_{s,v,w}}_{(\ddot{\delta}M_{rc})_{s,v,w}} \Bigg\rangle_{\substack{s \in \{\text{left}, \text{right}\},\\ v \in \{1..2\},\\ w \in \{1..t \in \mathbb{N}\}}}}_{\ddot{\delta}M_r} \Bigg\rangle.$$

9.2 CCDS definition

The autonomous CCDS composed of a single design layer is formally described as follows:

$$\left\langle \mathcal{L},\ \left\langle \langle \Delta_{DM},\ \mathcal{U}_{DM} \rangle,\ \langle \mathring{\Delta}_{DM},\ \mathring{\mathcal{U}}_{DM} \rangle \right\rangle,\ [\![\cdot,\ \cdot]\!],\ \langle\!\langle \cdot,\ \cdot,\ \cdot,\ \cdot \rangle\!\rangle,\ \mathscr{R}_{DM},\ \mathscr{E}_{DM} \cup \mathring{\mathscr{E}}_{DM},\ \mathscr{P}_{DM},\ \mathscr{T}_{DM} \right\rangle,$$

$$\mathscr{R}_{DM} = \left\{ \Delta'_{M},\ \hat{x}_{Mg} \right\},\ \mathscr{E}_{DM} = \left\{ \Delta'_{D},\ \hat{x}_{De} \right\},$$

where

- $\left\langle \langle \Delta_{DM},\ \mathcal{U}_{DM} \rangle,\ \langle \mathring{\Delta}_{DM},\ \mathring{\mathcal{U}}_{DM} \rangle \right\rangle$ is the universe represented by both the *whole domain* universe $\langle \Delta_{DM},\ \mathcal{U}_{DM} \rangle$, and the designer's universe $\langle \mathring{\Delta}_{DM},\ \mathring{\mathcal{U}}_{DM} \rangle$;
- $\mathscr{R}_{DM}$ is the definitional ruleset consisting of the working set of game mechanics design dimensions Δ'_{M}, and an existing game design (game mechanics) $\hat{x}_{Mg}$;
- $\mathscr{E}_{DM} \cup \mathring{\mathscr{E}}_{DM}$ is the evaluative ruleset composed of both the common $\mathscr{E}_{DM}$ consisting of the working set of game dynamics design dimensions Δ'_{D} and the game dynamics 'optimal' expected design concept $\vec{x}_{De} = \langle \langle 1,\ 1.0,\ 1,\ 1,\ 1.0 \rangle,\ 1,\ 1,\ 65 \rangle$ (See Section 9.3.3.2 for details), and the designer's individual ruleset $\mathring{\mathscr{E}}_{DM}$;
- $\mathscr{P}_{DM}$ is the the computational agent perception ruleset;
- $\mathscr{T}_{DM}$ is the the computational agent action ruleset.

9.3 Design process

In the following sections, we break down the fundamental processes for designing into smaller steps and provide a detailed description and definition in terms of our framework with mappings to the components of a direct representation genetic algorithm (GA).

9.3.1 Formulation & reformulation

$$\left\{ \hat{x}_{De} \underrightarrow{\subset} \check{x}_{De},\ \check{x}_{De} \rightleftarrows \vec{x}_{De},\ \hat{x}_{Mg} \underrightarrow{\subset} \check{x}_{Mg},\ \check{x}_{Mg} \overset{\dashrightarrow}{\dashrightarrow} \check{x}_{Mp},\ \check{x}_{Mp} \rightleftarrows \vec{x}_{Mp} \right\} \subseteq \mathscr{P}_{DM}.$$

The design task starts with the computational agent receiving, as arguments, both the expected dynamics design concept $\hat{x}_{De}$ and the existing game mechanics design concept $\hat{x}_{Mg}$ (an element of a inspiring set)

$$\hat{x}_{De} \underrightarrow{\subset} \check{x}_{De},\ \hat{x}_{Mg} \underrightarrow{\subset} \check{x}_{Mg}.$$

The agent removes the game rule dimensions of the existing game to set a game mechanics design concept template

$$\check{x}_{Mg} \Rrightarrow \check{x}_{Mp}.$$

With these two interpreted design concepts, the agent formulates its game dynamics and mechanics design spaces

$$\check{x}_{De} \rightleftarrows \vec{x}_{De},\ \check{x}_{Mp} \rightleftarrows \vec{x}_{Mp}.$$

9.3.2 Synthesis

$$\left\{ \vec{x}_{De} \Mapsto \vec{X}_{Mr},\ \vec{X}_{Mr} \rightarrow \hat{X}_{Mr} \right\} \subseteq \mathscr{T}_{DM}.$$

In the first design iteration (population initialization stage of the GA), the agent randomly produces a set of new game rules (partial game mechanics design concepts) $\vec{X}_{Mr}$, taking into account the game objects already present in its game mechanics design concept template $\vec{x}_{Mp}$.

In subsequent design iterations (inside the main evolution loop of the GA), the agent produces new sets of game rules by applying its uniform mutation and uniform recombination operators to members of its current set of produced game rules selected by the method FPS (fitness proportional selection) according to their evaluations

$$\vec{x}_{De} \Mapsto \vec{X}_{Mr}.$$

At the end of each design iteration, the agent outputs its synthesized designs via logging

$$\vec{X}_{Mr} \rightarrow \hat{X}_{Mr}.$$

9.3.3 Analysis & evaluation

9.3.3.1 Game mechanics analysis

$$\left\{ \left\{ \vec{X}_{Mr},\ \check{x}_{Mp} \right\} \rightleftarrows \left\{ \check{X}_{Mr},\ \check{x}_{Mp} \right\},\ \left\{ \check{X}_{Mr},\ \check{x}_{Mp} \right\} \Rrightarrow \check{X}_{Ma} \right\} \subseteq \mathscr{P}_{DM},$$

$$\left\{ \check{X}_{Ma} \Mapsto \check{X}_{Da} \right\} \subseteq \mathscr{E}_{DM}.$$

In order to analyze its designs, the agent first retrieve its game mechanics template and the set of synthesized rules

$$\left\{ \vec{X}_{Mr},\ \check{x}_{Mp} \right\} \rightleftarrows \left\{ \check{X}_{Mr},\ \check{x}_{Mp} \right\}.$$

Next, the agent produces a set of complete playable games (game mechanics representations)

$$\left\{ \check{X}_{Mr},\ \check{x}_{Mp} \right\} \Rrightarrow \check{X}_{Ma},$$

and derives a set of actual game dynamics concepts through automatic playtesting and intrinsic analysis (See 'Actual design partial analysis (c)' in Section 6.1.3.1)

$$\check{X}_{Ma} \mapsto \check{X}_{Da}.$$

9.3.3.2 Game dynamics evaluation

$$\left\{ \check{X}_{Dd} \rightleftarrows \vec{X}_{Dd} \right\} \subseteq \mathscr{P}_{DM}, \left\{ \vec{x}_{De} \leftrightarrow \check{X}_{Da} \right\} \subseteq \mathscr{E}_{DM}, \left\{ \vec{X}_{Dd} \rightarrow \hat{X}_{Dd} \right\} \subseteq \mathscr{T}_{DM}.$$

Fitness computation: The agent computes the fitness (as evaluation value) for each of the derived game dynamics concepts through a combined weighted metric and compares them with its expected game dynamics concept to determine if the design task is completed or further design iterations may be required (stop criteria in the GA)

$$\check{X}_{Dd} = \left(\vec{x}_{De} \leftrightarrow \check{X}_{Da} \right).$$

For each $\check{x}_{Da}$ actual game dynamics design concept:

a) the agent computes the fitness of the *duration* dimension value according to the function

$$f_d(x) := \begin{cases} 0 & \text{if } (x < 2) \vee (x > 150) \\ \dfrac{(x-2)}{63} & \text{if } x \leq 65 \\ \dfrac{(151-x)}{86} & \text{ow.} \end{cases}$$

This function disfavors games requiring less than two or more than 150 moves to fire its win conditions and considers a 65-moves game to have the highest fitness. The fitness of games requiring more than 65 moves decreases with a steeper slope than games requiring less than 65 moves (Fig. 9.4).

b) computes the fitness of the *ruleset heuristics* dimension as a weighted sum of its base dimensions' actual values

$$f_h(x) := \sum_{i=1}^{|\ddot{\delta}_{D_{rh}}|} w_i x_i$$

for $w = [0.20, 0.15, 0.30, 0.25, 0.10]$,

c) and computes the final design concept fitness $\check{x}_{Dd}$ as the weighted sum of the complete set of game dynamics fitness values

$$f(x) := \sum_{i=1}^{|\mathcal{D}_D|} w_i x_i$$

for $x = \left[f_h\left(\check{x}_{Da_{(0)}}\right), \check{x}_{Da_{(1)}}, \check{x}_{Da_{(2)}}, f_d\left(\check{x}_{Da_{(3)}}\right) \right]$, and $w = [0.10, 0.30, 0.30, 0.30]$.

Fitness logging: At the end of each design iteration, the agent outputs its computed evaluation via logging

$$\vec{X}_{Dd} \rightarrow \hat{X}_{Dd}.$$

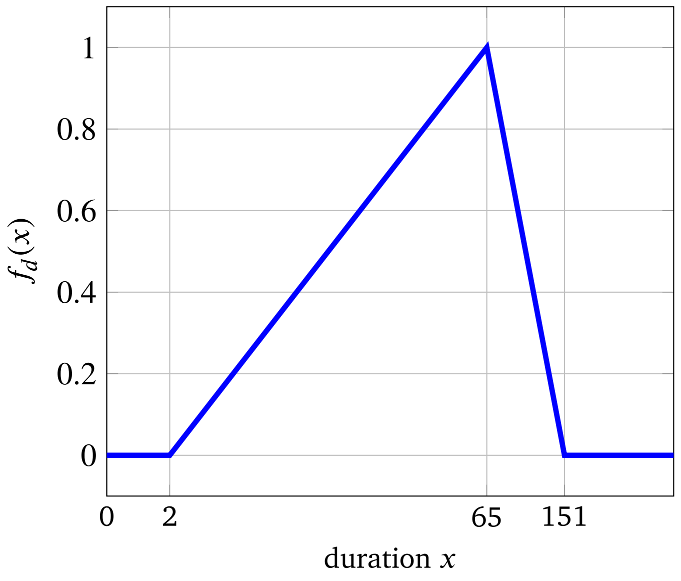


Figure 9.4: Autonomous CCDS - *duration* dimension fitness function

9.4 Experimental results

The overall activity of the agent was implemented as a direct representation genetic algorithm (GA) with the following parameters:

- Mutation probability: 3% (determined empirically through experimentation)
- Swapping probability: 90% (determined empirically through experimentation)
- Elitism: 1 individual
- Number of generations: 50

After 27 executions (Figs. 9.5 and 9.6), the autonomous CCDS was able to come up with the original game rule and several new ones (Fig. 9.7) that would account for new variants of the original game.

For example, the original rule `[ > player | create ] -> [ > player | > crate ]`, interpreted as “if the player moves towards an adjacent crate then the player moves and the crate also moves in the same direction”, enables the player to push crates on the board and thus be able to place them in a position to fire a win condition. One of the rules designed by the autonomous system was `[ crate | < player ] -> [ < crate | > player ]`, interpreted as “if the player moves towards an adjacent crate then the player’s avatar moves in the opposite direction to its original direction and the crate moves in the player’s original direction”, enables the player to push crates on the board bouncing when doing so, thus introducing a variant to the game, a new condition that the human or computational player must consider if she/he/it wants to trigger the win condition.

The Fig. 9.5 shows the fitness curve for the experiment:

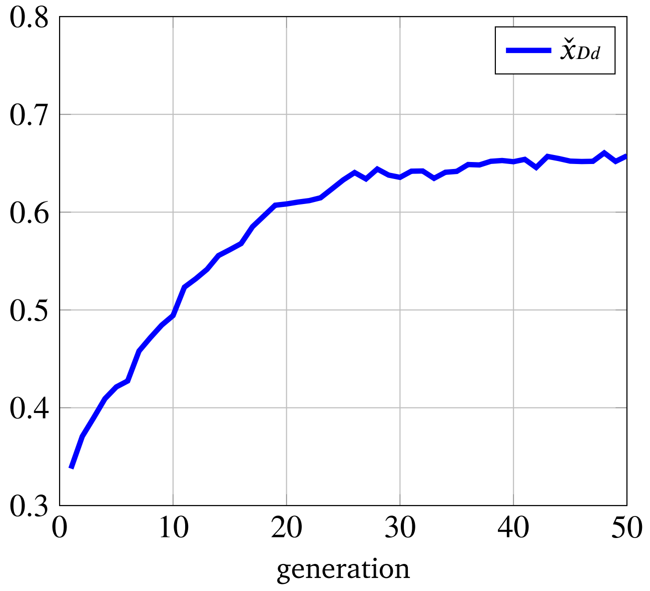


Figure 9.5: Autonomous CCDS designing activity convergence.

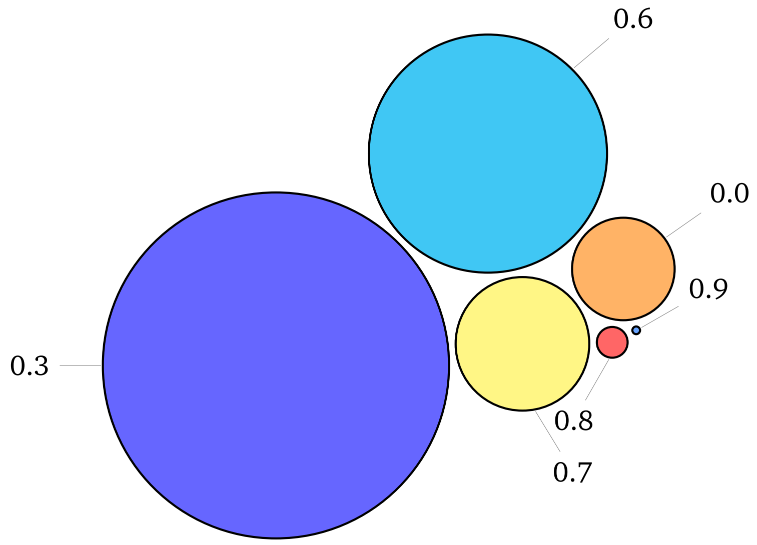


Figure 9.6: Autonomous CCDS - produced game rulesets by fitness.

Adding the *duration* dimension into the set of dimensions described in Lim and Harrel's work was very beneficial because it increased the evolutionary pressure of the genetic algorithm to find rules that caused more interesting (longer) games than just games with valid rules.

We initially implemented the agent's overall behavior through grammatical evolution (GE) to allow the agent to design rulesets of any size and complexity (open-ended design). However, we decided to switch to a direct and fixed representation GA. After running some experiments,

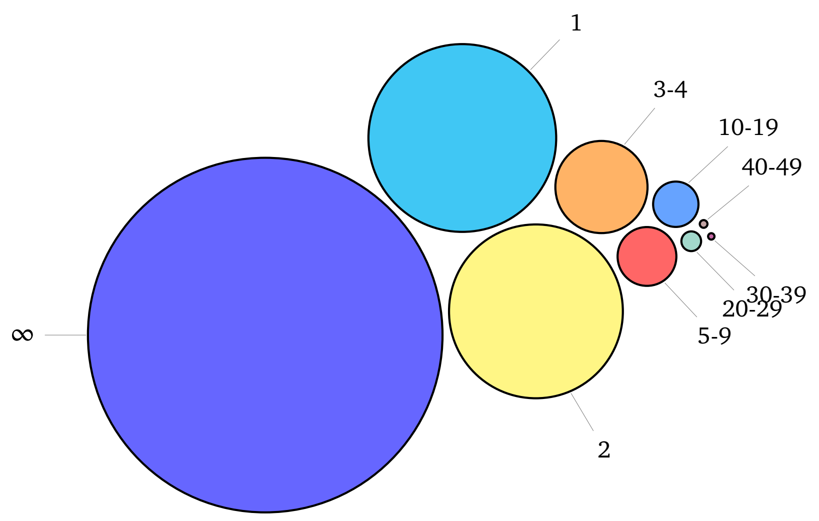


Figure 9.7: Autonomous CCDS - produced game rulesets by *duration*. An infinite number of moves means that none of the simulated players used during analysis could find a solution for the game within 1000 iterations of its search algorithm.

we detected too much overrepresentation and underrepresentation of gens that diminished the effect of the variation (synthesis) operators.

Another problem with grammatical evolution was that sometimes too large and complicated rules were generated (e.g., a ruleset composed of many rules, with some rules composed of many patterns, and patterns composed of many cells) that although valid, made it difficult to find a solution by automatic playtesting, or caused 'undetermined' compilation errors in the PuzzleScript engine.

10 Conclusions

We started this work by reviewing several theories, models, and systems in fields related to design research, computational creativity, and cognitive science, and we thoroughly studied the three fundamental theoretical models: Gero's FBS/sFBS, Wiggins' CSF, and Gärdenfors' CST.

Then, by unifying, adapting, and extending these reference models, we formally and comprehensively formulated the computational creative design systems framework (CCDSF) to describe, analyze and model autonomous and collaborative computational creative design systems (CCDSs), which we published (Pérez-Romero and Aguilar 2020) together with a description and analysis, in terms of the CCDSF, of an existing system in the domain of video game design ***«Secondary objective 1»***.

Our framework generalizes FBS and sFBS by introducing the possibility of modeling an unlimited number of design levels, enabling the description of different views or perspectives involved in a particular design process. It considers the design layers that form between levels as creative design systems in themselves (even if they belong to a more extensive system), with the advantage that each layer or system can be studied and described independently with its own universe, conceptual space, and rulesets for defining, evaluating and producing design objects. Another consequence of our layered structure is that the two-fold function of a design space (conceptual space) becomes evident: solution space and evaluation space. We extend CSF by introducing an additional set of rules to model different modes of perception (understood as cognitive processes), giving the possibility to distinguish, for example, between different agents using the same techniques for synthesizing objects of the same type.

Our framework consolidates the declarative and procedural knowledge of the design process in question through rulesets that not only encode the different specific operations of the design domain that each agent performs in the process but also group them into different categories of processes for designing (formulation, synthesis, analysis, evaluation, documentation, reformulation) and clearly identifies their defining, evaluating or generating nature according to the creative systems perspective, that is, it allows the definition or distinction of both design process responsibilities and creative responsibilities.

Our framework introduces the distinction between the idea of the universal conceptual space (universe) of a specific design domain and the universe of the designer (agent), the former being the space within which the latter has the opportunity to expand and intersect with the universes of other agents with whom it collaborates. This reasoning is replicated in the rulesets distinguishing between different types of shared rules (e.g., domain rules, cultural rules, rules established by requirements) and agent-internal rules.

The conceptual spaces theory (CST) was crucial to integrate the notion of conceptual space from Boden's conceptual theory of creativity with those of design space and variables of FBS, allowing to establish a representation scheme of design concepts through sets of (design) dimensions belonging to each of the levels involved that can be directly mapped to different concrete computational implementation mechanisms (e.g., vectors, maps, objects). The integration of CST also allowed our framework to propose that both design spaces and design objects themselves can be considered and studied as aggregates of dimensions at different levels of abstraction and that, as a consequence, the activity of designing could be considered an exercise of extension and restriction of the same space as a result of the execution of the different design operations.

We consider that if the system developed as a proof of concept had been built solely from the perspective of computational creativity (definition, evaluation, and exploration of a conceptual space), the resulting system would not have captured the tasks and responsibilities of the agents concerning the design process, with a more accentuated focus on the generation of objects that could be considered creative without emphasizing the requirements and constraints oriented nature of the activity of designing. On the contrary, if the system had been modeled only from the FBS perspective (and without its extensions), in principle, it would not have been possible to take as a basis the MDA model of video game design since FBS considers only the function, behavior, and structure levels. However, even omitting this limitation, it would not have been possible to clearly identify and group the concrete design operations that can potentially promote or directly trigger behaviors possibly deemed as creative (not limited to synthesis or generation), and perhaps, it would not have been possible to propose a flexible design space (or spaces), capable of contraction and expansion within other larger spaces (universes).

During the second part of this research work, we incrementally modeled and built a PoC consisting of a collaborative (human-computer) CCDS for designing puzzle-type video games for the PuzzleScript platform. Also, to illustrate the CCDSF's versatility, we created a second PoC in the same domain, this time as a small autonomous CCDS ***«Secondary objective 2»***.

When formulating the CCDSF and building the collaborative CCDS, we identified aspects inherent to the environment (also computational) where two agents, human and computational, would collaborate to some degree to produce designs. In this sense, although this work was carried out within the computational creativity and artificial intelligence fields, we addressed and applied, albeit minimally and superficially, aspects related to the design and assessment of the user interface, user experience, and interaction ***«Secondary sub-objectives 2.2 and 2.4»***. These elements are not usually considered relevant because traditionally, in the artificial intelligence field, algorithms and systems are built with minimal or no interaction with the user (since the purpose is usually the execution of an 'intelligent' task as effectively and efficiently as possible), but in our case, interactivity acquires greater relevance as it is essential to enable collaborative systems (of any type) and as an element that may foster creativity in the human agent.

To provide the GUI our PoC required to enable interaction between the assistant and the human designer, we applied, with the help of a professional user interface and user experience designer, the ten Nielsen interface usability heuristics, both for the conception of the interface and for its evaluation from the user's perspective. From these guidelines and as detailed in the chapters where PoC is developed, we conceptualized, in a user-story-like style, the GUI through mockups while outlining the applicable rulesets and their underlying strategies according to our framework.

The notions of collaborative agent and environment necessarily led us to develop the framework for systems and not just agents, which implies that, at least for a collaborative CCDS, the system will require not only the performance of an 'intelligent' task but an orchestration of several of them (and others perhaps not considered as 'intelligent' but still essential in designing), some even simultaneous or interdependent. Due to this reason and to demonstrate the features of our framework through the PoC, the efforts to model and build it were focused on completeness, that is, covering as many design operations as possible at as many levels as possible, and with the greatest possible positive impact on the human agent and not on the degree of 'intelligence' that they proved to have (traditional criteria in artificial intelligence).

A criterion we followed to implement the rulesets and underlying strategies of the computational agent ***«Secondary sub-objective 2.3»*** was to select algorithms and mechanisms that were relatively straightforward to implement and integrate with the PuzzleScript platform. The second important criterion was to maintain a balance between the effectiveness of the algorithms and the speed of their execution in order not to hinder the fluidity of the user experience, something that is crucial in an interactive tool; for example, in the first version, the computational agent played the human-lead designed games (to analyze them) simulating the behavior of 4 different players, however, when the computational agent incorporated synthesis strategies, it was necessary to review the four algorithms utilized and select the one that offered the best balance since now the agent would require an analysis strategy for its own productions, in addition to testing the game design led by the human. It was due to these balancing needs that we had to establish some restrictions, particularly on the size of the games and the number of dimensions used to design the games at the lowest level of game mechanics. Identifying, integrating, adapting, and implementing these specific computational mechanisms and artificial intelligence techniques helped to demonstrate one of the key elements of our framework, which is that a variety of interchangeable strategies instrument the capabilities of an agent and characterize its behavior. Although this modular-architecture-like feature is treated in various software engineering models and techniques, in our case, it is the one that occurs at the level of the rulesets and fundamental processes for designing, regardless of actual software engineering and coding techniques.

The reason for selecting the video game design field for our verification products was the multifaceted nature of this field (a wide variety of design concepts are produced), a feature that seemed ideal but challenging for building a PoC. Even though different types of design objects are involved in this domain, we attempted to cover the domain as holistically as possible to produce at least designs (games) fully playable by both computational and human players. To this end, we identified and studied theoretical elements of this domain ***«Secondary sub-objective 2.1»***, discovering with some ease a clear correspondence between specific elements of this domain with the components of our framework, a situation that may support the validity of the design-related aspects of our framework and count as further evidence of the generalization power of FBS/sFBS.

In essence, a puzzle-type video game presents the player with the challenge of finding a solution by means of one or more mechanisms such as logical reasoning, a dominant strategy, quick response movements, and maze-like environments, which in general may require the player to stop and think for a moment to deduce a sequence of movements (anticipate) in order to solve the situation. Based on the video game design model MDA, it is possible to approach the design process of this type of video game through three levels: (a) game mechanics (e.g., game objects and characters, movement rules, completion conditions, topologies of game boards or game spaces, level maps) that enable a player to "play" a video game, (b) game dynamics that emerge from both the particular composition of mechanics and the player's interaction with the game, and (c) the game (or playful) experience that the player experiences as a result of the emerged game dynamics.

The orchestration of puzzle video game mechanics can be thought of as a type of rule-based system that is executed when the player plays the video game (from a given game state and a movement or input of the player, the applicable rules are executed, and a new game state is obtained). Due to this feature, we took advantage of the existence of the Puzzlescript platform, which provides both a language for the specification of puzzle-like video game mechanics and an engine for their execution.

In the third stage of developing the collaborative CCDS, the computational agent was equipped with its first synthesis capabilities, and a brainstorming-like session was enabled between the two participant agents. A user interface designer and puzzle video game enthusiast conducted a formative and summative qualitative evaluation of this collaborative scenario ***«Secondary sub-objective 2.4»***. The evaluation results may suggest that the human designer could have benefited technically and creatively after participating in collaborative design sessions with the computational assistant. Some of the ideas provided by the assistant and the "techniques" displayed by it helped the human designer to identify and break certain design fixations and preconceived ideas that limited her creative potential. In addition to these benefits, the interaction with the computational agent helped the human designer improve her technical performance by making conscious of some aspects and design decisions that she usually managed automatically or unconsciously. Observing the computational agent's performance and attempting to understand its behavior indirectly caused the human designer to notice inaccuracies in defining requirement-like concepts. Besides the synthesis skills of the computational agent, the designer also recognized the advantage of having the assistance of an agent in tasks that, although they are not usually given a creative connotation, are essential in a creative system, such as the analysis and evaluation of the designs. Having this information available relatively immediately allowed her to experiment with more ideas.

After having developed the present research work and according to the results documented in the previous chapters, we can consider that our research hypothesis happens to be sustained since, through the formulated framework, we were able not only to describe an existing computational design system from the mixed perspective of the design process and computational creativity but to comprehensively and accurately model (also from the mixed design-creativity perspective) and incrementally build a collaborative computational design system in which the participating agents (human and computational) perform various tasks of the particular design domain and the computational agent (in particular) exhibits (incrementally) diverse levels of creative responsibility whether by the direct output of its tasks or by the impact of its tasks output on the human agent creative performance.

10.1 Summary of contributions

- We formally and comprehensively formulate and published the computational creative design systems framework (CCDSF), which we consider our main contribution.
- We built through the CCDSF a complete and robust collaborative (human-computer) CCDS as well as a small autonomous CCDS as PoCs in the field of puzzle-type video game design.
- We demonstrated the importance of interactivity and user experience design and evaluation as essential elements in building a human-computer collaborative CCDS.
- We showed that the collaborative CCDS we built as PoC could help a human designer to produce viable designs and improve her own technical and creative performance in different ways.
- When formulating the CCDSF, some collateral findings and ideas emerged that, although they do not directly respond to a research objective, they do represent, in our opinion, useful contributions:
  - We extended elements of the reference models to gain flexibility and greater generalization power (e.g., levels, designer's universe vs. universe, shared vs. individual rulesets).
  - We formulated a structural recursive construction that formally explains how a design space (or a conceptual space in its broadest sense) is constructed with base and composite dimensions and how multiple conceptual levels may be embedded, from base dimensions to the notion of universe.
- The effort to unify design with computational creativity in a formal model may contribute to dispelling the widespread notion of tightly relating computational creativity systems and sometimes even the general idea of creativity with the sphere of art.

10.2 Limitations

- Because we built the collaborative CCDS entirely from scratch, we could not develop the computational agent to the point where it could be deemed a colleague of the human designer, nor an environment that could be considered fully co-creative. Nevertheless, we believe we came to develop a creative support agent that collaborates (to some extent) with the human designer in a collaborative alternating-mode environment.
- Although no professional game designers were involved in this research, we had a valuable contribution from a professional user experience and interaction designer who was also a puzzle video game enthusiast.
- After completing the implementation of the second version of the collaborative computational agent (apprentice-assistant), we ran into the limitation of having a dataset too minimal to be exploited by the ML-based synthesis-oriented strategies, forcing us to adjust the direction (in terms of concrete strategies' underlying techniques) of the next version of the agent.
- Since we focused on implementing as many operations as possible in the collaborative computational agent, it was not possible to introduce more complex interpretation strategies that would allow us to simulate different forms of "perception."

10.3 Future work

- To employ the CCDSF for modeling and building systems in diverse design domains and to incorporate new strategies into an existing CCDS.
- When crafting the CCDSF, we proposed that the dimension- and concept-based representation system could be utilized to describe design concepts and potentially other types, such as interaction-related ones; further work could look into this potential in more detail.
- To make the necessary adjustments to the collaborative CCDS software product so that it can be released and perhaps used as a research platform (e.g., test different interpretation strategies to simulate complex cognitive processes, test synthesis strategies based on state-of-the-art generative techniques, test new analysis strategies) with the advantage that the rest of the strategies would already be implemented (there would be a base functional platform), in a way similar to working with cognitive architectures.
- We suggest building a relatively large dataset to use ML-based synthesis-related strategies, such as variational autoencoders. However, we know that the task is not easy when considering that our system defines three levels of design for each game. The PuzzleScript platform maintains a repository with more than 100 playable games (some with multiple game levels) designed by enthusiasts (and maybe some professionals). However, this data only accounts for the game mechanics level, so at least one more level (game dynamics) would have to be built to set the synthesis relationship.

References

Adams, E. and Dormans, J. (2012). *Game Mechacnis: Advanced Game Design*. Berkeley, CA, USA: New Riders Games. ISBN: 978-0-321-82027-3.

Arciszewski, T., Michalski, R. S., and Wnek, J. (1997). *Constructive Induction: The Key to Design Creativity*. Technical report MLI 95-6. Machine Learning and Inference Laboratory, George Mason University.

Arrasvuori, J., Boberg, M., and Korhonen, H. (2010). "Understanding Playfulness: An Overview of the Revised Playful Experience PLEX Framework." In: Seventh International Conference on Design and Emotion (Design & Emotion 2010). Chicago, IL, USA, pp. 1–12. DOI: `10.5281/zenodo.2596777`.

Bentley, P. J. and Wakefield, J. P. (1998). "Generic Evolutionary Design." In: *Soft Computing in Engineering Design and Manufacturing*. Part 6. Springer, London, pp. 289–298.

Beyer, H.-G. and Schwefel, H.-P. (May 2002). "Evolution Strategies - A comprehensive introduction." In: *Natural Computing* 1.1, pp. 3–52. ISSN: 1567-7818. DOI: `10.1023/A:1015059928466`.

Boberg, M., Karapanos, E., Holopainen, J., and Lucero, A. (2015). "PLEXQ: Towards a Playful Experiences Questionnaire." In: 2015 Annual Symposium on Computer-Human Interaction in Play CHI PLAY '15 (Oct. 2015). London, United Kingdom: Association for Computing Machinery (ACM), pp. 381–391. DOI: `10.1145/2793107.2793124`.

Boden, M. A. (1998). "Creativity and artificial intelligence." In: *Artificial Intelligence* 103.1–2, pp. 347–356.

Boden, M. A. (2004). *The creative mind. Myths and mechanisms*. 2nd ed. Routledge, London. 344 pp.

Brown, S. (Dec. 2010). *Likert Scale Examples for Surveys*. URL: `https://www.extension.iastate.edu/Documents/ANR/LikertScaleExamplesforSurveys.pdf` (visited on 08/04/2020).

Browne, C. (2011). *Evolutionary Game Design*. SpringerBriefs in Computer Science. Springer. 125 pp. ISBN: 978-1-4471-2179-4.

Browne, C. (2013). "Metrics for Better Puzzles." In: *Game Analytics. Maximizing the Value of Player Data*. Ed. by M. Seif El-Nasr, A. Drachen, and A. Canossa. Springer, London, pp. 769–800. ISBN: 978-1-4471-4769-5. DOI: `10.1007/978-1-4471-4769-5_34`.

Carbonell, J. R. (1970). "AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction." In: *IEEE Transactions on Man-Machine Systems* 11.4, pp. 190–202.

Cascini, G., Fantoni, G., and Montagna, F. (2011). “Beyond the Design Perspective of Gero’s FBS Framework.” In: *Fourth International Conference on Design Computing and Cognition (DCC ’10)*. Fourth International Conference on Design Computing and Cognition (DCC ’10) (2010). Stuttgart, Germany, pp. 77–96.

Cascini, G., Fantoni, G., and Montagna, F. (2013). “Situating needs and requirements in the FBS framework.” In: *Design Studies* 34.5, pp. 636–662.

Chakrabarti, A. and Blessing, L. (2015). “A Review of Theories and Models of Design.” In: *Journal of the Indian Institute of Science* 95.4, pp. 325–340.

Chakrabarti, A. and Blessing, L. T. M. (2014). “Theories and Models of Design: A Summary of Findings.” In: *An Anthology of Theories and Models of Design. Philosophy, Approaches and Empirical Explorations. Philosophy, Approaches and Empirical Explorations*. Springer-Verlag, London. Chap. 1, pp. 1–45.

Crawford, C. (1984). *The Art of Computer Game Design. Reflections Of A Master Game Designer*. Berkeley, CA, US: Osborne/McGraw-Hill. 113 pp. ISBN: 0-88134-117-7.

Crawford, C. (Dec. 1990). “Yet Another Definition of Game.” In: *The Journal of Computer Game Design* 4.2. URL: `https://www.erasmatazz.com/library/the-journal-of-computer/jcgd-volume-4/volume-4-number-2-december.html` (visited on 05/31/2020).

Eberle, B. (1996). *Scamper: Games for Imagination Development*. Prufrock Press. 42 pp.

Galle, P. (2009). “The ontology of Gero’s FBS model of designing.” In: *Design Studies* 30.4, pp. 321–339.

Game Design Ed (Aug. 15, 2015a). *How to Build a Maze*. URL: `https://youtu.be/WShnM3lPxhk` (visited on 03/30/2020).

Game Design Ed (Aug. 17, 2015b). *Puzzle Explorer*. URL: `https://www.youtube.com/playlist?list=PLlY144fsDBfL3Xszf0z1wutVvagIWtPCq` (visited on 03/30/2020).

Gärdenfors, P. (2004). *Conceptual Spaces. The Geometry of Thought*. MIT Press. 318 pp.

Gärdenfors, P. (2007). “Representing actions and functional properties in conceptual spaces.” In: *Body, Language and Mind, Volume 1: Embodiment*. Ed. by T. Ziemke, J. Zlatev, and R. M. Frank. Mouton de Gruyter, pp. 167–195.

Gärdenfors, P. (2014). *The Geometry of Meaning. Semantics Based on Conceptual Spaces*. MIT Press. 356 pp.

Gärdenfors, P. (2017). “Semantic Knowledge, Domains of Meaning and Conceptual Spaces.” In: *Knowledge and Action*. Ed. by P. Meusburger, B. Werlen, and L. Suarsana. Vol. 9. Knowledge and Space (KNAS). Springer, Cham. Chap. 12, pp. 203–219. ISBN: 978-3-319-44587-8.

Gärdenfors, P. and Warglien, M. (2012). “Using conceptual spaces to model actions and events.” In: *Journal of Semantics* 29.4, pp. 487–519.

Gero, J. S. (1990). “Design Prototypes: A Knowledge Representation Schema for Design.” In: *AI Magazine* 11.4 (Winter), pp. 26–36.

Gero, J. S. (2000). “Computational Models of Innovative and Creative Design Processes.” In: *Technological Forecasting and Social Change* 64.2–3, pp. 183–196.

Gero, J. S. and Kannengiesser, U. (2000). “Towards a Situated Function-Behaviour-Structure Framework as the Basis of a Theory of Designing.” In: *Workshop on Development and Application of Design Theories in AI in Design Research, Sixth International Conference on Artificial Intelligence in Design (AID ’00)*. Worcester, MA, USA.

Gero, J. S. and Kannengiesser, U. (2002). “The Situated Function—Behaviour—Structure Framework.” In: *Seventh International Conference on Artificial Intelligence in Design (AID ’02)*. Seventh International Conference on Artificial Intelligence in Design (AID ’02). Cambridge, UK, pp. 89–104.

Gero, J. S. and Kannengiesser, U. (2004). “The situated function–behaviour–structure framework.” In: *Design Studies* 25.4, pp. 373–391.

Gero, J. S. and Kannengiesser, U. (2014). “The Function-Behavior-Structure Ontology of Design.” In: *An Ontology of Theories and Models of Design. Philosophy, Approaches and Empirical Explorations*. Springer-Verlag, London. Chap. 13, pp. 263–283. ISBN: 978-1-4471-6337-4.

Gero, J. S., Kannengiesser, U., and Pourmohamadi, M. (2014). “Commonalities Across Designing: Empirical Results.” In: *Fifth International Conference on Design Computing and Cognition (DCC ’12)*. Texas A & M University, USA: Springer, Dordrecht, pp. 265–281.

Gero, J. S., Kannengiesser, U., and Williams, C. B. (2014). “Does designing have a common cognitive behavior independent of domain and task: a meta-analysis of design protocols.” In: *International Conference on Human Behavior in Design (HBiD 2014)*. Ascona, Switzerland.

GMW contributors (Oct. 6, 2011). *Level — Game Mechanics Wiki*. URL: `https://gamemechanics.fandom.com/wiki/Level` (visited on 04/02/2020).

Gómez de Silva Garza, A. (2017). “An introduction to and comparison of computational creativity and design computing.” In: *Artificial Intelligence Review* 51.1, pp. 61–76.

Hatchuel, A. and Weil, B. (2007). “Design as Forcing: deepening the foundations of C-K theory.” In: Sixteenth International Conference on Engineering Design ICED’07. Paris, France.

Howard, T. J., Culley, S. J., and Dekoninck, E. (2008). “Describing the creative design process by the integration of engineering design and cognitive psychology literature.” In: *Design Studies* 29.2, pp. 160–180.

Hunicke, R., Leblanc, M., and Zubek, R. (2004). “MDA: A Formal Approach to Game Design and Game Research.” In: Challenges in Game AI Workshop, Nineteenth National Conference of Artificial Intelligence (AAAI-04). San Jose, CA, USA, pp. 1–5.

Juul, J. (Nov. 2005a). *A Dictionary of Video Game Theory*. URL: `https://www.half-real.net/dictionary/` (visited on 05/25/2020).

Juul, J. (2005b). *Half-Real. Video Games between Real Rules and Fictional Worlds*. MIT Press. 244 pp. ISBN: 9780262101103.

Kan, J. W. T. and Gero, J. S. (2011). “Comparing designing across different domains: an exploratory case study.” In: *18th International Conference on Engineering Design (ICED 11). Impacting Society through Engineering Design. Vol. 2: Design Theory and Research Methodology*. Lyngby/Copenhagen, Denmark, pp. 194–203.

Kannengiesser, U. and Gero, J. S. (2015). “Is designing independent of domain? Comparing models of engineering, software and service design.” In: *Research in Engineering Design* 26.3, pp. 253–275.

Kantosalo, A. and Toivonen, H. (June 2016). “Modes for Creative Human-Computer Collaboration: Alternating and Task-Divided Co-Creativity.” In: Seventh International Conference on Computational Creativity (ICCC 2016). Paris, France: Association for Computational Creativity (ACC), pp. 77–84. ISBN: 9782746691551.

Kelly, N. and Gero, J. S. (2015). “Situated interpretation in computational creativity.” In: *Knowledge-Based Systems* 80, pp. 48–57.

Korhonen, H., Montola, M., and Arrasvouri, J. (2009). “Understanding playful user experience through digital games.” In: International Conference on Designing Pleasurable Products and Interfaces DPPI09. Compiegne, France, pp. 274–285.

Lavelle, S. (2013). *PuzzleScript*. URL: `https://www.puzzlescript.net/Documentation/about.html` (visited on 03/03/2020).

Li, R., Emmerich, M. T., Eggermont, J., Bäck, T., Schütz, M., Dijkstra, J., and Reiber, J. (Mar. 2013). “Mixed Integer Evolution Strategies for Parameter Optimization.” In: *Evolutionary Computation* 21.1, pp. 29–64. ISSN: 1063-6560, 1530-9304. DOI: `10.1162/EVCO_a_00059`.

Liapis, A., Yannakakis, G. N., and Togelius, J. (2013). “Sentient Sketchbook: Computer-Aided Game Level Authoring.” In: *8th International Conference on the Foundations of Digital Games (FDG)*. Chania, Crete, Greece.

Lim, C.-U. and Harrell, D. F. (2014). “An approach to general videogame evaluation and automatic generation using a description language.” In: 2014 IEEE Conference on Computational Intelligence and Games (CIG 2014). Dortmund, Germany: IEEE, pp. 1–8. DOI: `10.1109/CIG.2014.6932896`.

Lindley, C. (2004). “Narrative, Game Play, and Alternative Time Structures for Virtual Environments.” In: Second International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE 2004). Technologies for Interactive Digital Storytelling and Entertainment (June 2004). Ed. by S. Göbel, U. Spierling, A. Hoffmann, I. Iurgel, O. Schneider, J. Dechau, and A. Feix. Vol. 3105. Lecture Notes in Computer Science. Darmstadt, Germany, pp. 183–194. DOI: `10.1007/978-3-540-27797-2_25`.

Lucero, A. and Arrasvouri, J. (Sept. 2010). “PLEX Cards: A Source of Inspiration When Designing for Playfulness.” In: Third International Conference on Fun and Games (Fun and Games ’10). Leuven, Belgium: Association for Computing Machinery (ACM), pp. 28–37. DOI: `10.1145/1823818.1823821`.

Nielsen, J. (1994). “Enhancing the explanatory power of usability heuristics.” In: ACM Conference on Human Factors in Computer Systems (CHI94) (Apr. 1994). Ed. by B. Adelson, S. Dumais, and J. Olson. Boston, Massachusetts, USA: Association for Computing Machinery (ACM), pp. 152–158. DOI: `10.1145/191666.191729`.

Nielsen, J. (Nov. 15, 2020). *10 Usability Heuristics for User Interface Design*. URL: `https://nngroup.com/articles/ten-usability-heuristics` (visited on 01/03/2021).

Novick, D. G. and Sutton, S. (1997). *What is Mixed-Initiative Interaction?* Technical report SS-97-04. AAAI.

Pérez-Romero, J. and Aguilar, W. (Nov. 2020). “CCDSF: A Computational Creative Design Systems Framework.” In: *New Generation Computing* 38.4, pp. 673–711. ISSN: 0288-3635, 1882-7055. DOI: `10.1007/s00354-020-00109-9`.

Pustejovsky, J. (2013). “Foreword.” In: *Puzzles in Logic, Languages and Computation. The Red Book*. Ed. by D. Radev. Vol. 1. Recreational Linguistics. Springer-Verlag, p. vii. ISBN: 978-3-642-34378-0. DOI: `10.1007/978-3-642-34378-0`.

Quanz, B., Sun, W., Deshpande, A., Shah, D., and Park, J.-e. (2020). *Machine learning based co-creative design framework*. Vancouver, Canada. arXiv: `2001.08791 [cs.HC]`.

Rama Fiorini, S., Gärdenfors, P., and Abel, M. (2015). “Representing part–whole relations in conceptual spaces.” In: *Cognitive Processing* 15.2, pp. 127–142.

Reymen, I. M. M. J., Hammer, D. K., Kroes, P. A., Aken, J. E. van, Dorst, C. H., and Bax, M. F. T. (2006). “A domain-independent descriptive design model and its application to structured reflection on design processes.” In: *Research in Engineering Design* 16.4, pp. 147–173.

Salustri, F. (Feb. 10, 2015). *The Purpose-Function-Behaviour-Structure framework*. DesignWIKI: Fil Salustri's Design Site. URL: `https://deseng.ryerson.ca/dokuwiki/design:pfbs` (visited on 08/25/2018).

Schell, J. (2008). *The Art of Game Design. A Book of Lenses*. Morgan Kaufmann Publishers. 489 pp. ISBN: 978-0-12-369496-6.

Uflacker, M. and Zeier, A. (2008). "Extending the Situated Function-Behaviour-Structure Framework for User-Centered Software Design." In: *Third International Conference on Design Computing and Cognition (DCC '08)*. Third International Conference on Design Computing and Cognition (DCC '08). Atlanta, GA, USA, pp. 241–259.

Webb, A. (2002). "TRIZ: an inventive approach to invention." In: *Engineering Management* 12.3, pp. 117–124.

Wiggins, G. A. (Nov. 2006). "A preliminary framework for description, analysis and comparison of creative systems." In: *Knowledge-Based Systems* 19.7, pp. 449–458. ISSN: 0950-7051. DOI: `10.1016/j.knosys.2006.04.009`.

Wikipedia contributors (May 17, 2020). *Twitch gameplay — Wikipedia, The Free Encyclopedia*. URL: `https://en.wikipedia.org/w/index.php?title=Twitch_gameplay&oldid=957152091` (visited on 05/25/2020).

Yannakakis, G. N., Liapis, A., and Alexopoulos, C. (2014). "Mixed-Initiative Co-Creativity." In: *9th International Conference on the Foundations of Digital Games (FDG)*. Liberty of the Seas, Caribbean.

Yilmaz, S., Daly, S. R., Seifert, C. M., and Gonzalez, R. (2016). "Evidence-based design heuristics for idea generation." In: *Design Studies* 46, pp. 95–124.