UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Programa de Maestría y Doctorado en Música

Escuela Nacional de Música
Centro de Ciencias Aplicadas y Desarrollo Tecnológico
Instituto de Investigaciones Antropológicas

APRENDER JUGANDO. UNA INTERFAZ INTERACTIVA DE APOYO AL APRENDIZAJE DEL VIOLÍN.

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRÍA EN MÚSICA (Tecnología Musical)

PRESENTA:
KATYA ALEJANDRA ALVAREZ MOLINA

TUTOR
M EN I ANTONIO PÉREZ LÓPEZ CCADET UNAM

JURADO:
DR. FELIPE ORDUÑA BUSTAMANTE, CCADET (UNAM)
DR. PABLO PADILLA LONGORIA, IIMAS (UNAM)
DR. ROBERTO MORALES MANZANARES, Programa de Maestría y Doctorado en Música
DR. IR FONS VERBEEK, Programa de Maestría y Doctorado en Música
M EN I. ANTONIO PÉREZ LÓPEZ, CCADET (UNAM)

MÉXICO, D. F. OCTUBRE 2014

Agradecimientos

Primeramente agradezco al Posgrado en Música de la UNAM por el esfuerzo y empeño que realiza cada profesor, personal administrativo, el Dr. Kolb y el comité académico para que cada alumno que ingrese cuente con las herramientas necesarias para su área de estudio.

Quiero agradecer al programa de Movilidad Estudiantil del Posgrado de la UNAM por su apoyo para realizar una estancia de investigación en la Universidad de Leiden; también agradecer al programa PAEP por su apoyo para la compra del material necesario para la elaboración de este proyecto de investigación.

Por último quiero agradecer a los miembros del jurado por su apoyo y buenos consejos que me dieron para poder hacer este trabajo de la mejor manera y cumplir con las exigencias del posgrado.

Al Dr. Roberto Morales y al Dr. Pablo Padilla, que además de tomar de las mejores clases del posgrado y compartir sus conocimientos con los alumnos, me permitieron participar de alguna manera en sus proyectos.

Al Dr. Orduña por su interés y dedicación en esta tesis y su apoyo en la parte de programación. A mi tutor el M en I. Antonio por su apoyo con las cuestiones técnicas de mi proyecto y con esta tesis; y que junto con el Dr. Orduña me brindaron no sólo un espacio de trabajo sino una grupo de compañeros dentro del CCADET que hicieron mucho mas ameno este trabajo.

Al Dr. Fons por ser mi supervisor durante mi estancia en Leiden y compartir su experiencia dentro de esta área conmigo. Todos y cada uno de ellos me han apoyado para seguir desarrollándome en el campo de la tecnología musical

Agradecimientos a título personal.

Gracias a Dios por brindarme salud y dejar terminar otra etapa de mi vida, por darme momentos buenos y malos que me ayudaron a crecer.

Primeramente y como recuerdo a ella quiero agradecer a Tere, que para mi fue siempre mi segunda mamá, por sólo recibir de ella cosas buenas, por pedir por mi cada noche, por cuidarme, apoyarme incondicionalmente y siempre darme una sonrisa y su paz. Tere, ya no pudiste ver este trabajo terminado, pero sé que donde quiera que estés, te dará mucho gusto.

Quiero agradecer a mi mamá por siempre apoyarme en todas mis decisiones por muy extrañas que las considere; por alentarme a ser mejor en todos los aspectos de mi vida. Por darme sus consejos pero sobretodo su amor.

A mi hermana que me ha puesto la meta muy alta: alcanzarla a ella. Por siempre estar conmigo pese a las distancias y compartirlo todo. Porque sé que nunca vamos a estar solas mientras estemos unidas.

A Tadeo que se ha convertido en la personita más importante, y que aunque él aún no lo sepa, le ha dado un hermoso giro a mi familia.

A mi abuelo Pato por preocuparse y querer siempre lo mejor para mi, por siempre estar ahí apoyándonos y compartiéndome su experiencia.

A Matilde por darle emoción a mi vida y ser mi compañera fiel por todos estos años.

Al Dr. Fons que no sólo es un gran profesor, sino un gran amigo; por creer en este proyecto y apoyarme para seguir adelante dentro del área; por hacerme parte de su grupo de alumnos y no sólo intercambiar ideas sino también hacer grandes amigos. Gracias por haber hecho mi estancia en Leiden una gran experiencia tanto académica como personal.

Quiero agradecer a Hanna Schraffenberger por su tiempo y apoyo en la realización del videojuego, por compartir conmigo sus conocimientos con los códigos y su experiencia en el área de Media Technology.

A mi amiga Lu Cao por sus consejos en el diseño del videojuego y por hacer más placentera mi estancia en Leiden, pero sobretodo por la amistad que me ha brindado todo este tiempo.

A mi amiga Aline por compartir sus conocimientos; amiga, has sido un gran apoyo para este trabajo y todas mis ideas, gracias por siempre tener tiempo para mis cosas, por las largas pláticas y por cada vez enseñarme a disfrutar la vida.

A mis compañeros de la maestría y del CCADET por compartir sus experiencias e intercambiar ideas, pero sobretodo por compartir su tiempo conmigo.

Al foro de Processing y Arduino, por su interés en mi proyecto y apoyo con la programación.

Dedicada y en memoria
a mi papá

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN. 1
DEFINICIÓN DEL PROYECTO. 1
OBJETIVOS GENERALES. 3
OBJETIVOS PARTICULARES. 3
DESCRIPCIÓN. 4
CAPÍTULO 2. ESTADO DEL ARTE. 5
PRINCIPIOS BÁSICOS DEL VIOLÍN. 5
Posición de la mano izquierda. 5
Posición básica o natural del violín 9
Características de la partitura de principiantes. 10
ASPECTOS TECNOLÓGICOS. 11
El violín como instrumento de performance tecnológico. 11
Los Videojuegos 16
Videojuegos Casuales 17
ASPECTOS PSICOMOTRICES 19
ASPECTOS PEDAGÓGICOS 21
Diseño de videojuegos para el aprendizaje 24
ASPECTOS DE DISEÑO. 27
Estructura de los sistemas interactivos. 27
Cuatro puntos básicos para el diseño de sistemas interactivos. 28
Procesos para el diseño de sistemas interactivos. 31
Escenarios 32
Usabilidad 33
Visualización. 35
CAPÍTULO 3. DESCRIPCIÓN DEL PROYECTO. 37
DESCRIPCIÓN GENERAL. 37
DESARROLLO. 38
INTERFAZ ELECTRÓNICA. 40
Construcción. 40
Microcontrolador 48
INTERFAZ GRÁFICA 52
Aspectos Visuales 52
Pantallas 55
Sliders 57
Listas 58
Botones 58
Pantalla de la partitura (SongScreen) 59
Pantalla de Información 60
Pantalla de Inicio 60
Pantalla Principal 61
Partitura y sonido 65
Instrucciones 70
CAPITULO 4. EVALUACIÓN Y ANÁLISIS. 73
CAPÍTULO 5. CONCLUSIONES. 81
BIBLIOGRAFÍA 83
ANEXOS A CÓDIGO DE PROGRAMACIÓN INTERFAZ GRÁFICA EN PROCESSING2.0 85

Pantalla videogameG4P_18 85
Pantalla Estrellita 90
Pantalla Lightly_Row.....93
Pantalla Wind 97
Pantalla Help Screen 100
Pantalla Info Screen.....102
Pantalla Inicio Screen 103
Pantalla Setting Screen.....104
Pantalla Song Screen 106

ANEXO B CÓDIGO DE PROGRAMACIÓN ARDUINO NANO..... 109

Tabla de Figuras.

Fig. 1. Ejercicio de violín del método del Profesor Raymundo Moro (1985) 2

Fig. 2. posición incorrecta del ángulo que deben de tener los dedos de la mano izquierda. 7

Fig. 3. posición correcta del ángulo que deben de tener los dedos de la mano izquierda 7

Fig. 4. Posición correcta de la mano izquierda (Suzuki, 1978).....8

Fig. 5. A la izquierda el ejercicio 1 que propone Ramos (1947). A la derecha un ejemplo de la mano izquierda realizando el ejercicio de Ramos (1947). 9

Fig. 6. Partitura Lightly row del método Suzuki (1978).....11

Fig. 7. El violin overtone.....12

Fig. 8. A la izquierda el puente del violín impreso en una tarjeta PCB con sensores ópticos. A la derecha el guante con un acelerómetro.....13

Fig. 9. A la izquierda el R-Bow o arco con los 2 sensores FSR, los 2 acelerómetros y el microcontrolador para enviar los datos. Del lado derecho el Bonge por donde se deslizará el R-Bow o arco con las 4 esponjas que contienen los sensores. 14

Fig. 10. A la izquierda el diapasón conformado por el sensor posicional lineal y los 4 sensores FSR. Del lado derecho una forma de utilizar el diapasón utilizándolo como flauta cuando no se utiliza el R-Bow. 14

Fig. 11. A la izquierda el BoSSA (bowed-sensor-speaker-array). A la derecha el diagrama con algunos de los sensores que contiene el BoSSA. 14

Fig. 12. Componentes electrónicos instalados en el acrílico del violín vivisi 16

Fig. 13. El violín Vivisi 16

Fig. 14. Motivaciones detrás del juego vs ganar (Blaine, 2005).....23

Fig. 15. Implicaciones sociales del juego vs ganar (Blaine, 2005). 23

Fig. 16. Esquema de diseño conceptual del sistema interactivo.....39

Fig. 17. Pistas para plantilla de LEDs con tinta conductora.....42

Fig. 18. Prueba de LEDs sobre las pistas de tinta conductora y acetato ...43

FIG. 19. RESISTENCIAS DE 330Ω Y LEDS SOBRE PISTAS DE TINTA CONDUCTORA EN ACETATO 44
FIG. 20. COMPARATIVA DE TAMAÑO ENTRE LÁPIZ Y PLANTILLA 45
FIG. 21. PRUEBA DE LEDS CON VOLTAJE 46
FIG. 22. DIMENSIONES DE GROSOR DE LA PLANTILLA 46
FIG. 23 PRUEBA DE LA PLANTILLA SOBRE LEDS 47
FIG. 24. IZQUIERDA PLANTILLA CONECTADA AL ARDUINO. DERECHA PLANTILLA SOBRE EL VIOLÍN 47
FIG. 25. PLACA ARDUINO DUEMILANOVE 48
FIG. 26 PLACA ARDUINO NANO 50
FIG. 27 MONTAJE DE PLANTILLA Y ARDUINO SOBRE EL VIOLÍN 51
FIG. 28. BORRADOR DE POSIBLES PANTALLAS 52
FIG. 29. BORRADOR DE PANTALLAS DE INICIO Y PARTITURA 53
FIG. 30. BORRADOR DE PANTALLA DE OPCIONES Y AYUDA 54
FIG. 31. BORRADOR DE PANTALLA DE OPCIONES 54
FIG. 32. DIAGRAMA DE FLUJO DE ACCESO A LAS DIFERENTES PANTALLAS 55
FIG. 33. PANTALLA DE INICIO DEL VIDEOJUEGO 74
FIG. 34. PANTALLA DE OPCIONES 75
FIG. 35. PANTALLA DE INFORMACIÓN DEL SISTEMA. 76
FIG. 36. PANTALLA MOSTRANDO LA AYUDA DEL JUEGO 77
FIG. 37. PANTALLA DE LA PARTITURA 78

Capítulo 1. Introducción.

Este capítulo en su inicio explica una de las problemáticas con que se encuentran los niños que inician sus estudios del violín, y el porqué y cómo se pretende ayudar a los ejecutantes con este problema de una forma dinámica y divertida. Menciona los objetivos generales y particulares del proyecto. Al final del capítulo se hace una breve descripción general de cómo se desarrollará el prototipo para obtener los resultados deseados y cumplir con los objetivos planteados, para dar pie al capitulo 2 que hablará de las investigaciones recientes en los diferentes campos que se involucran.

Definición del proyecto.

El aprendizaje del violín comienza cada vez a menor edad; sin embargo, cualquiera que se acerca por primera vez a este instrumento se encuentra con problemas o retos que en muchas ocasiones desaniman al alumno y lo hace abandonar su intento de aprender. Dentro de los problemas a que se enfrentan estos alumnos en sus inicios están aprender una posición correcta para sostener el violín, aprender a sostener el arco, el movimiento de la muñeca derecha con el arco, el movimiento de los dedos en las diferentes cuerdas a lo largo del diapasón del violín, etc.

En este proyecto se aborda específicamente uno de estos problemas: asociar la posición de los dedos en un punto específico del diapasón del violín.

Para ayudar al aprendizaje, en niños y principiantes, los maestros generalmente recurren a marcar con diferentes materiales (barniz, cinta adhesiva u otros) la posición básica en que deben colocarse los dedos a lo largo del diapasón con el fin de que los alumnos tengan una guía visual del lugar en donde se ubicarán los dedos, y con esto, poco a poco tengan una percepción del espacio sobre el diapasón. De forma paralela se lee la partitura, asociando a cada nota la posición de un dedo sobre la cuerda correspondiente. Muchos profesores recurren a métodos muy sencillos e ingeniosos para que el alumno logre la asociación motora y de espacio de la mano sobre el diapasón

con la partitura. Como se puede observar en la figura 1 se muestra la relación entre las notas y la posición de los dedos asignando un número romano para determinada cuerda (I=Mi, II=La, III=Re, IV=Sol) y un número arábigo para cada dedo (0=cuerda al aire, 1=índice, 2=medio, 3=anular, 4=meñique).

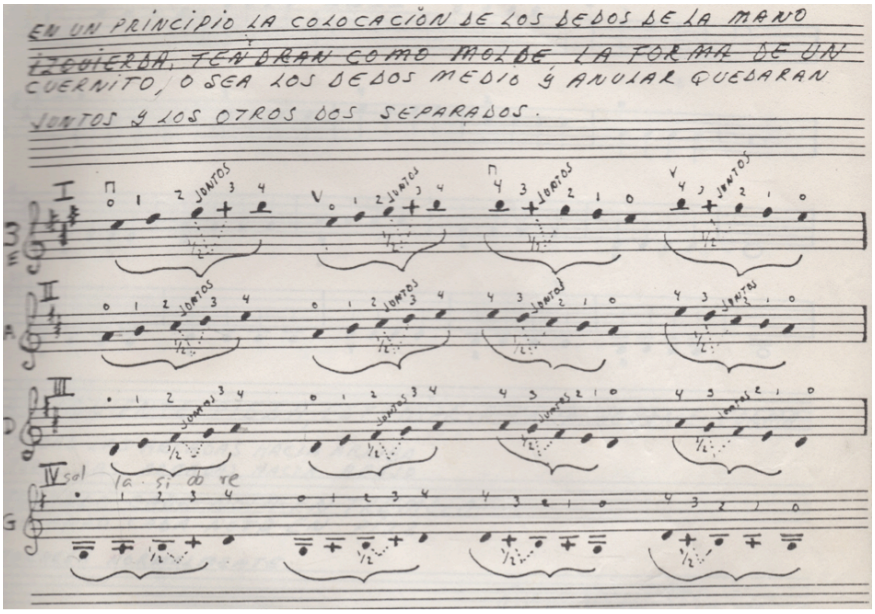

FIG. 1. EJERCICIO DE VIOLÍN DEL MÉTODO DEL PROFESOR RAYMUNDO MORO (1985)

Buscando ayudar a la solución de este problema, se plantea la implementación de una interfaz que asocie la parte visual, espacial y motora del alumno con el violín, como se explica más adelante.

Objetivos Generales.

- Diseñar, construir e implementar una interfaz electrónica que permita asociar la colocación de los dedos sobre las cuerdas del violín, a lo largo de su diapasón, con la representación gráfica de la partitura y esto, a su vez, con las características propias de un video juego.

Objetivos Particulares.

- Que la implementación de la interfaz adaptada a un violín sea accesible.
- Crear una interfaz gráfica en la que se emule un videojuego y que, a su vez, simule la interpretación de la música con un violín.
- Sincronizar visualmente la interfaz electrónica o controlador adaptado en el violín con la interfaz gráfica mostrada en la pantalla de una computadora.
- Que la implementación de esta interfaz reúna las características necesarias para que el aprendizaje del violín sea más divertido y dinámico.

Descripción.

Este proyecto busca hacer más ameno el aprendizaje del violín. Para esto se ha conceptualizado un juego de imitación para que los niños y las personas que lo deseen puedan interactuar con él mediante el uso de un dispositivo electrónico añadido al instrumento, que en este caso será el violín, y un videojuego.

Para la parte didáctica se desarrolla un videojuego con el fin de que los niños puedan jugar mientras ejercitan o practican con el violín. Actualmente, los niños encuentran divertidas estas aplicaciones, con las que pueden pasar un tiempo considerable jugando con ellas. Este videojuego tiene cinco pantallas que llevan a diferentes escenarios: la presentación del juego, la configuración, la información del mismo, una pantalla para ayuda y la pantalla del desarrollo del juego.

En la pantalla donde se desarrolla el videojuego, se muestra una partitura que va marcando la nota que se debe ir tocando, mediante un color diferente y un cambio en el tamaño de la misma. Al mismo tiempo, si se desea, se va escuchando la melodía que sirve como guía auditiva al juego.

El dispositivo electrónico se puede incorporar a cualquier violín y se considera que se pueda remover sin dañar el instrumento; contendrá luces o marcas, las cuales indican la posición sobre el diapasón en la que se deben de colocar cada uno de los dedos. Este dispositivo irá colocado entre el diapasón y las cuerdas.

Las luces o marcas indicarán cuál es el dedo que se debe ir posicionando en el diapasón conforme a la nota que se va mostrando en la partitura del videojuego. Con esto se busca que el jugador tenga una asociación entre la mano izquierda del violín y la partitura.

Capítulo 2. Estado del arte.

Se inicia este capítulo con información sobre la técnica de la posición del brazo izquierdo en el violín. Se menciona porqué, generalmente, se usa la armadura de La mayor en las partituras para principiantes. Después de los aspectos musicales, el capítulo continúa con la explicación de los aspectos tecnológicos enfocados al violín, los desarrollos que han surgido en estos años y su visualización como interfaz de expresión. La sección referente a la tecnología menciona los videojuegos y su diseño, para posteriormente enfocarse en el aspecto pedagógico de su uso. El capítulo termina con la parte psicomotriz y educativa que se genera a partir de las tecnologías mencionadas.

Principios básicos del violín.

Técnica para la posición de la mano izquierda.

Galamian (1962) señala que “el camino hacia el dominio del violín es largo y arduo, y se requiere gran dedicación y perseverancia para alcanzar el objetivo. El talento ayuda a facilitar el camino, pero no puede en sí mismo ser un sustituto del trabajo duro de práctica. Incluso el trabajo duro será de poco beneficio si es del tipo que no da resultados, ya que hay tanto buena práctica como mala, desafortunadamente la mala es más común que la buena. No hay nada más valioso para un instrumentista que la habilidad de trabajar eficientemente –saber cómo lograr los máximos resultados benéficos usando el mínimo tiempo”.

La posición de cuerpo es fundamental para el desarrollo de la técnica ya que permite una respiración eficiente, menos tensión y mejora la funcionalidad corporal, permitiendo una mejor ejecución (Barrón, 2009).

Algunos de los elementos corporales en los que Galamian (1962) y Barrón (2009) han puesto énfasis involucran los pies, tronco, brazos, muñeca, cabeza, dedos, etc. Algunos de estos elementos mencionados por Barrón (2009) y Galamian (1962) son:

Brazo izquierdo.

Se debe buscar la óptima posición del codo de manera que se tenga un mejor desempeño en los dedos. Por lo general se sitúa en línea vertical por debajo del cuello de violín. Galamian (1962) comenta que los dedos van a indicar la posición más cómoda, natural y optima para cada ejecutante.

Cabeza.

La posición de la cabeza es una de las menos estéticas ya que se altera el ángulo de giro con frecuencia hacia la cabeza del violín – Barrón (2009), hace referencia a quienes llegan incluso a dirigir su nariz más a la izquierda de la cabeza del violín o voluta- Cuando se ve hacia la cabeza del violín o voluta, uno puede prestar atención a los movimientos técnicos del arco y de la mano izquierda, y también se puede leer una partitura si ésta se encuentra en línea con la voluta.

Mano izquierda.

Es la encargada de afinar y digitar las notas, así como de la afinación de cada una de ellas. También es la encargada del vibrato para tener un sonido más agradable dependiendo de la interpretación.

- Muñeca.- la mano y el antebrazo forman aproximadamente una línea recta. Esto permite la mayor naturalidad en el movimiento de los dedos. En algunas ocasiones, la muñeca se flexiona más para facilitar la ejecución de algunos trinos, extensiones, notas dobles y acordes.

- Dedos

a) Elevación y ángulo.- En una buena posición, las articulaciones de los dedos 1, 2, 3 y 4 (índice, medio, anular y meñique) están de forma flexionada formando una curvatura de medio circulo aproximadamente; si la curvatura es pequeña, al presionar los dedos sobre la cuerda, lo harán de forma plana y con la parte no central de la yema, provocando sonidos desfavorables, así como incomodidad e ineficiencia en el desempeño de los dedos (Ramos, 1947). Respecto al índice y pulgar; el

cuello del violín se apoya suevamente en el lado interno del pulgar, conectando también la parte baja del dedo índice. El contacto con el índice no es obligatoriamente fijo todo el tiempo, se puede separar para permitir ciertas acciones técnicas. No se debe recargar el cuello del violín en la cuenca que se forma entre el dedo pulgar y el índice, ya que ocasiona que el pulgar sobresalga excesivamente del diapasón y que los demás dedos queden a un nivel muy elevado; además, limita la libertad en general de la muñeca. Tampoco se debe flexionar la muñeca hacia atrás de manera que el cuello se recargue parcialmente en la palma de la mano en la primera posición; esto impide el libre tránsito hacia otras posiciones.

FIG. 2. POSICIÓN INCORRECTA DEL ÁNGULO QUE DEBEN DE TENER LOS DEDOS DE LA MANO IZQUIERDA.

FIG. 3. POSICIÓN CORRECTA DEL ÁNGULO QUE DEBEN DE TENER LOS DEDOS DE LA MANO IZQUIERDA

b) Balance.- un balance eficiente permite que todos los dedos (1, 2, 3 y 4) tengan la misma cercanía a las cuerdas; esto se logra con un ligero giro de la muñeca en el sentido horario de las manecillas del reloj. Con esto se puede compensar el tamaño del dedo meñique acercándolo así a la cuerda, de lo contrario quedaría en una situación de desventaja.

FIG. 4. POSICIÓN CORRECTA DE LA MANO IZQUIERDA (SUZUKI, 1978)

c) Sensibilidad.- es esencial que el acomodo de la mano y su contacto con el cuello y el diapasón promuevan condiciones de libertad, relajación y sensibilidad, permitiendo la acción de los dedos y el cambio de posición. Como menciona Barrón (2009): “la sensibilidad es necesaria para desarrollar hábitos favorables de movimiento y ajuste, indispensables para lograr una buena afinación. Un contacto ligero promueve una mejor sensibilidad. Con la guía del oído, la visualización mental de las distancias e intervalos y un tacto suave y sensible, la mano y los dedos se orientan más fácilmente en el diapasón. La orientación se ve favorecida por el contacto, tanto de las partes laterales del índice y pulgar, así como la parte inferior de la palma de la mano. Con esto el violinista es capaz de ir estableciendo útiles referencias sensoriales y mentales de ubicación”.

Posición básica o natural de los dedos aplicada en el violín

La separación natural o primaria de los dedos (Ramos, 1947) en la posición básica del violín es de: un tono del primero al segundo, un semitono del segundo al tercero, y un tono del tercero al cuarto dedo. Esta posición es la resultante de la contracción activa del supinador[1] del antebrazo que actúa sobre los flexores[2] de los dedos.

Ramos (1947) propone ejercicios para ejercitar los músculos del supinador y crear memoria muscular como el ejercicio 1: Tomar el acorde y ejecutar una presión simultánea con los dedos, anulando el esfuerzo al tocar con la cuerda en la tastiera, y permitiendo que ésta (la cuerda) vuelva sin impedimento a la posición anterior.

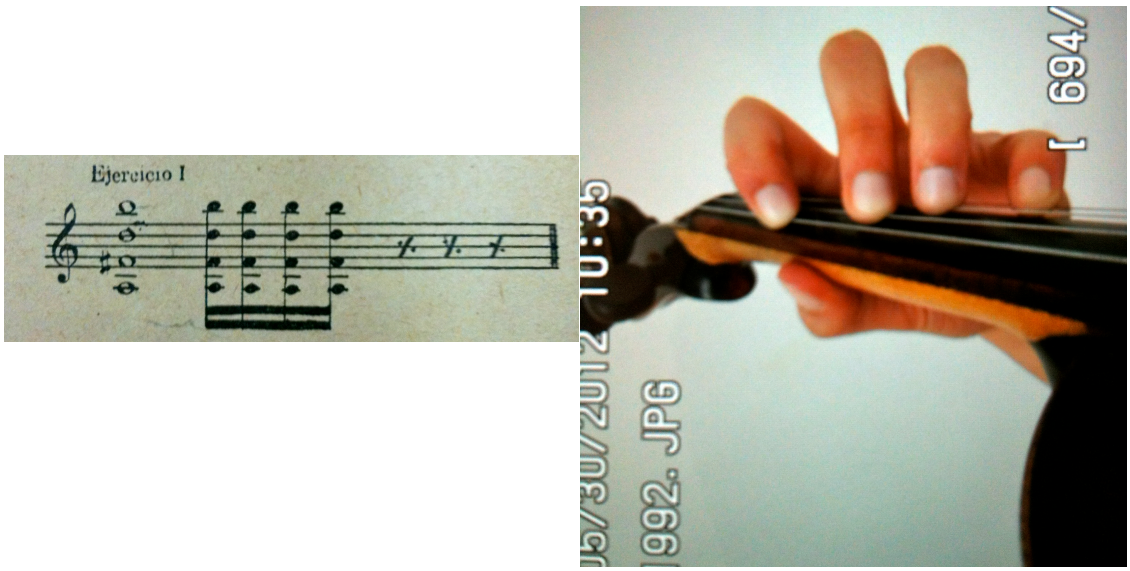

FIG. 5. A LA IZQUIERDA EL EJERCICIO 1 QUE PROPONE RAMOS (1947). A LA DERECHA UN EJEMPLO DE LA MANO IZQUIERDA REALIZANDO EL EJERCICIO DE RAMOS (1947).

También hace referencia a los ejercicios de mano izquierda sin arco, que los incorpora oficialmente Karl Flesch en su “Urstudien”, en 1911, y Francesco Sfilio en su “Corso di alta cultura de tecnica violinistica”, publicado en 1937. Sfilio mediante postulados originales, logra en primer término, el apoyo leve y la soltura de los dedos sobre las cuerdas.

1. Según el diccionario de la Real Academia Española supinación (del lat. supinatĭo, -ōnis) f. Movimiento del antebrazo que hace girar la mano de dentro a fuera, presentando la palma.
2. Según el dirccionario de la Real Academia Española flexor (del lat. Flexus) adj. Que dobla o hace que algo se doble con movimiento de flexión. *Músculo flexor.*

Los ejercicios de Ramos (1947) además de la administración de la energía muscular, establecen los distintos grados de distención de los dedos entre sí. Fundamenta en el cromatismo su método para principiantes, Sfilio elude la colocación de los dedos sobre la cuerda según las normas anatomo-fisiológicas, de ahí que el acorde en base al cual realiza la gimnasia de los dedos (la-fa-re-si) fue registrado en su libro con fa natural. Emilio Pelaia, revisor y traductor de la “la nueva escuela violinística italiana” del mismo autor (Ed. Ricordi Americana, Buenos Aires, 1950), ha elevado un semitono cromático dicha nota, haciendo un aporte conceptual a la obra del maestro Sfilio. Esta disposición del “acorde básico” introdujo a Joachim-Moser a adoptar la escala de Re mayor para los estudios iniciales en su “Traité du violón”.

Para Ramos (1947) estas técnicas tienen por objeto, desarrollar la función dinámica de toda la mano, ofreciendo al control directo de la voluntad. El dominio de dicha dificultad no consiste en su práctica continua (al menor síntoma de cansancio debe suspenderse el ejercicio, bajando la mano en reposo para que la circulación normal se restablezca) sino en la fijación perfecta de la sensación de cada movimiento de los dedos con relación a la actitud de toda la mano. La presión de los dedos y la elasticidad de cada movimiento, quedará así supeditada a las exigencias de la ejecución, y no a la intervención causal y arbitraria de un estado nervioso o inconsciente del violinista (Ramos, 1947).

Características de las partituras de principiantes.

En diferentes métodos de violín o libros para principiantes (Moro, 1980; Suzuki, 1978) se asigna un número arábigo a cada nota y un número romano a cada cuerda, con el fin de asociar la nota del pentagrama con el dedo que debe de ser utilizado; ésta asignación será de la siguiente forma en la mano izquierda: índice número 1, medio número 2, anular número 3 y meñique número 4. Esto generará (Altenmüller, 2006) una asociación visual y motora con el ejecutante.

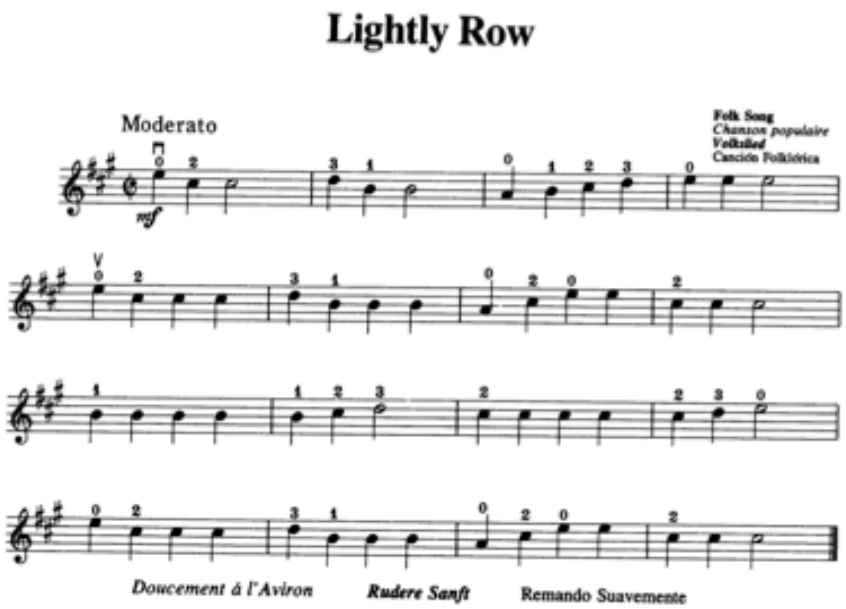

FIG. 6. PARTITURA LIGHTLY ROW DEL MÉTODO SUZUKI (1978)

Aunado a lo anterior, algunos profesores de violín optan por colocar alguna marca o referencia sobre el diapasón del violín, indicando el lugar correcto del dedo sobre la cuerda.

Aspectos tecnológicos.

El violín como instrumento de performance tecnológico.

Existen diferentes trabajos sobre uso de la tecnología para crear nuevos medios de expresión, sobretodo en universidades e institutos. Estos trabajos nos pueden dar una idea sobre los elementos o componentes electrónicos que se han usado para la construcción de interfaces y de cómo podría ser el procesamiento de señales.

Desde hace algunos años, los diseñadores se han esforzado por integrar la tecnología computacional al violín, centrando estos desarrollos en el procesamiento de señales mediante sensores para generar nuevos sonidos y crear nuevas formas de performance.

Existen diferentes instrumentos acústicos en los que la forma en la que son ejecutados contiene mayor grado de interacción ejecutante/instrumento; uno de ellos es el violín, el cual requiere cierta habilidad motora y de coordinación para

poder ser tocado; aunado a esto, se puede decir que se tienen dos elementos principales: el arco y la caja acústica. El uso de estos dos elementos hace mucho más dinámica la interacción que puede lograr el ejecutante. Por estas razones, el violín o más bien la forma en que es ejecutado, ha sido elegido por muchos desarrolladores para crear nuevas interfaces sonoras basándose en su forma.

Ahora bien, el diseño de nuevas interfaces musicales ha dejado a un lado el uso de la caja acústica, que ha sido reemplazada por sensores y componentes electrónicos, originando que la estética de este instrumento se haya visto modificada para alojar estos componentes dentro de diferentes tipos de materiales, como se muestra en la figura 7 y 11, lo que ha ocasionado nuevas formas en las que éste instrumento puede ser interpretado.

El uso de sensores y otros dispositivos electrónicos han extendido la gama de sonidos que son generados por el violín debido al procesamiento digital de señales y no sólo por su caja acústica, como el caso del violín OVERTONE de Dan Overhold (2004) en el que se conserva la forma en como es ejecutado el violín; es decir, con un arco y una simulación de la caja acústica. Esta caja no es hueca, contiene sensores para cada cuerda teniendo en cuenta que este violín tiene 6 cuerdas (Fig. 8) y botones para variar los sonidos emitidos, igualmente el arco contiene sensores para medir su posición, velocidad y desplazamiento.

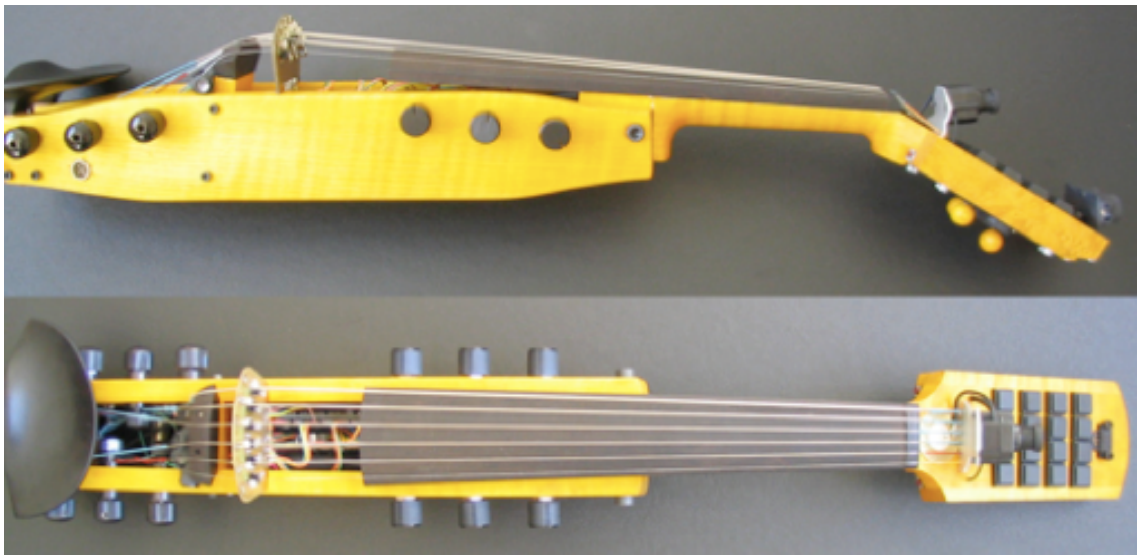

FIG. 7. EL VIOLIN OVERTONE

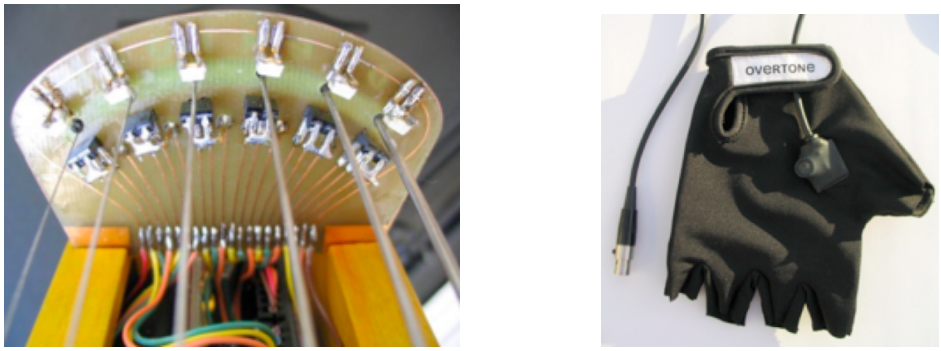

FIG. 8. A LA IZQUIERDA EL PUENTE DEL VIOLÍN IMPRESO EN UNA TARJETA PCB CON SENSORES ÓPTICOS. A LA DERECHA EL GUANTE CON UN ACELERÓMETRO

Otro ejemplo en el que se ven modificadas totalmente las dimensiones del violín y que proporciona más de una opción para ser ejecutado, es el violín BoSSA (*Bowed Sensor Speaker Array*) creado por Trueman y Coock en 1998, el cual reemplaza la caja acústica por una base rectangular larga y delgada en donde se montan cuatro sensores de presión y un sensor posicional por donde se puede deslizar el dedo. El arco igualmente contiene sensores de presión y movimiento, así como acelerómetros y un puente o área en donde se recorre el arco hecho también de sensores de presión. Este violín puede ser tocado también como flauta -si es que no se usa el arco- accionando tanto los cuatro sensores de presión con una mano y el sensor de posición con la otra. Otro aspecto importante es la salida de la señal, que es mediante un arreglo de doce bocinas montadas en un dodecaedro como se muestra en las figuras 4, 5, 6.

Como menciona Dan Overhold (2012), existe un nuevo campo de estudio como el *Musical Interface Technology Design Space* (MITDS) que involucra mayormente tres áreas: el performance musical, la interacción humano-computadora, y la incorporación de modernas tecnologías en sensores y el procesamiento digital de señales con el fin de crear avanzados instrumentos musicales.

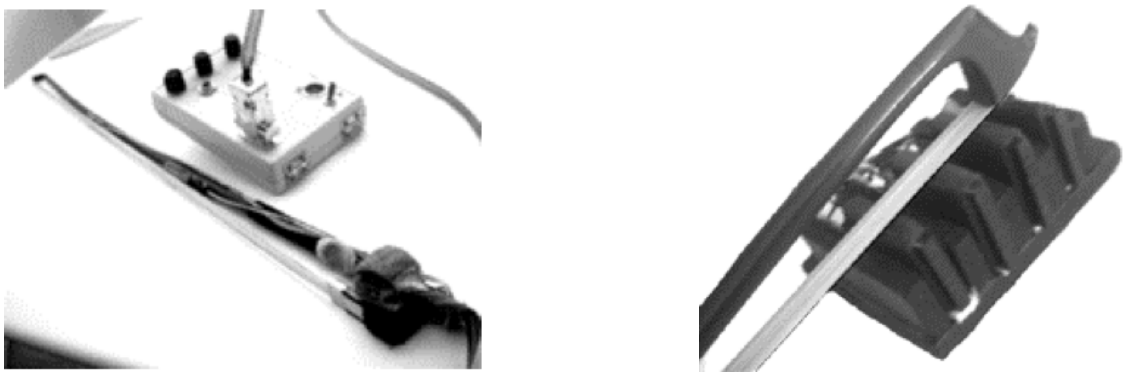

FIG. 9. A LA IZQUIERDA EL R-BOW O ARCO CON LOS 2 SENSORES FSR, LOS 2 ACELERÓMETROS Y EL MICROCONTROLADOR PARA ENVIAR LOS DATOS. DEL LADO DERECHO EL BONGE POR DONDE SE DESLIZARÁ EL R-BOW O ARCO CON LAS 4 ESPONJAS QUE CONTIENEN LOS SENSORES.

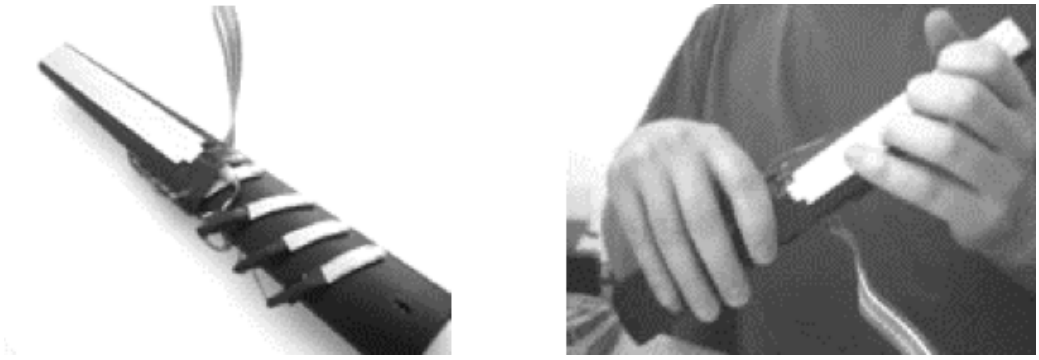

FIG. 10. A LA IZQUIERDA EL DIAPASÓN CONFORMADO POR EL SENSOR POSICIONAL LINEAL Y LOS 4 SENSORES FSR. DEL LADO DERECHO UNA FORMA DE UTILIZAR EL DIAPASÓN UTILIZÁNDOLO COMO FLAUTA CUANDO NO SE UTILIZA EL R-BOW.

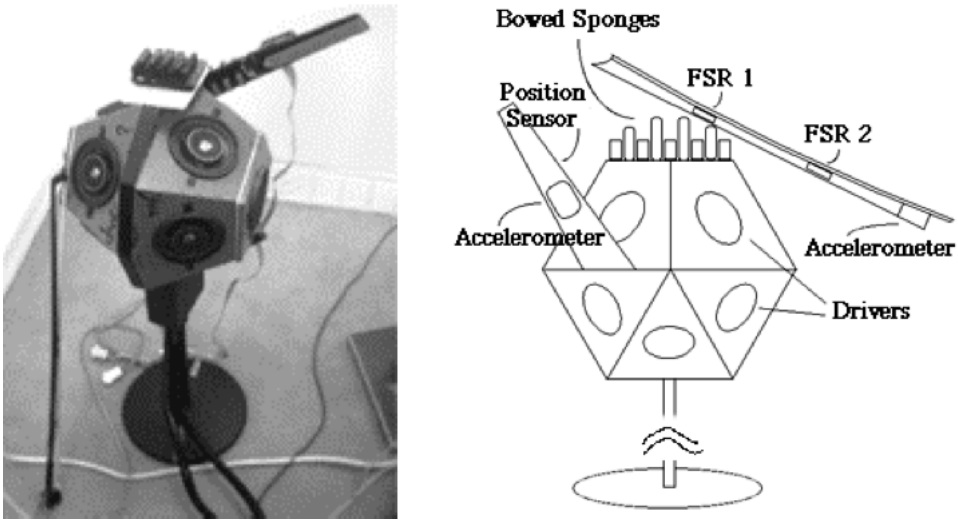

FIG. 11. A LA IZQUIERDA EL BOSSA (BOWED-SENSOR-SPEAKER-ARRAY). A LA DERECHA EL DIAGRAMA CON ALGUNOS DE LOS SENSORES QUE CONTIENE EL BOSSA.

Se puede hablar de una clasificación de estas nuevas interfaces musicales inspiradas en el violín abarcando desde su desarrollo como controladores gracias a los métodos de *Human Computer Interaction* (HCI). Wanderley (2001) distingue 3 categorías que a su vez en el MITDS se dividen en subcategorías, coincidiendo en los controladores alternos utilizados en la propuesta de Paradiso (1997):

1) Controladores como instrumento, que son interfaces que se asemejan a instrumentos ya existentes (fig. 7).
    a) Controladores simulando instrumentos, que reflejan técnicas de ejecución.
    b) Controladores inspirados en instrumentos, que provienen de técnicas abstractas derivadas.
2) Controladores aumentados como instrumentos tradicionales aumentados con sensores.
    a) Aumentados utilizando las técnicas tradicionales.
    b) Aumentados a través de técnicas extendidas.
3) Controladores alternativos, que son interfaces no semejantes a instrumentos existentes.
    a) Controladores touch que requieren contacto físico con la superficie.
    b) Controladores sin contacto o de gestos libres – rango de detección limitado.
    c) Controladores portátiles que tienen detección del movimiento del cuerpo, sobre todo en las extremidades.
    d) Controladores prestados, que son interfaces de realidad virtual, *gamepads,* etc.

Dentro de esta clasificación se encuentra el violín relacionado con HCI esencialmente en 3 de las subcategorías: controladores inspirados en instrumentos (categoría 1b), en los que a menudo la forma del violín no se mantiene. Sin embargo, varios gestos del ejecutante y la técnica sí; por ejemplo, el violín BoSSA de la figura 11, controladores aumentados capturando

las técnicas tradicionales (categoría 2a) haciendo uso del término tradicional refiriéndose a cualquier instrumento que tenga cuerdas y que puedan ser interpretados de manera convencional como el violín VIVISI de la figura 12 y 13.

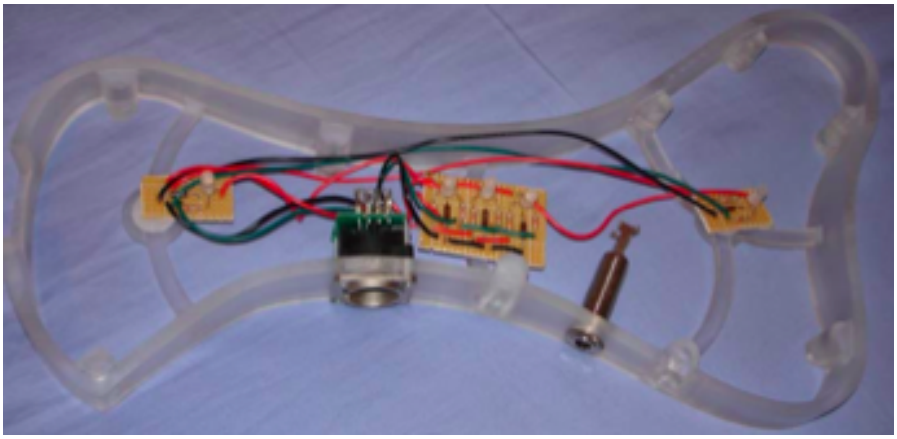

FIG. 12. COMPONENTES ELECTRÓNICOS INSTALADOS EN EL ACRÍLICO DEL VIOLÍN VIVISI

FIG. 13. EL VIOLÍN VIVISI

Tipos de Videojuegos usados actualmente.

Los videojuegos son los agentes de nuevas formas de alfabetización; aquí jugar puede significar leer, escribir y/o programar en un ambiente donde el consumo y producción de texto están inmediata y consecuentemente conectados (Anderson, 2007). El juego enlaza el consumo y producción de texto, alineando el lenguaje y el aprendizaje en un camino que se "ajusta" a cómo el cerebro humano está diseñado para aprender (Prensky, 2001). Los juegos deben ser intrínsecamente motivacionales, en especial los videojuegos

a los que se les puede agregar cierto elemento de fantasía para causar mayor efecto en el aprendizaje de los jugadores.

Videojuegos Casuales

Actualmente el videojuego no es algo *cool*, es algo normal, estos nuevos videojuegos no hacen que los jugadores se adapten al juego, sino que el juego se adapte a sus horarios y vidas. En un principio, los videojuegos fueron creados para una audiencia general y no para crear grupos de expertos. Actualmente, estamos en una era donde la simplicidad de los videojuegos está siendo redescubierta (Juul, 2010) y están siendo reinventados. Cualquiera puede ser un jugador de videojuegos. El querer saber cómo va a terminar una situación o qué va a pasar nos empuja a seguir dentro de ella. En los videojuegos pasa lo mismo, uno quiere saber cómo terminar el juego. Si uno no sabe qué se tiene que hacer, cómo se tiene que iniciar o el objetivo de un juego, por lo general no le llamará la atención.

Existe una nueva ola de juegos que están volviendo a dar empuje a los videojuegos y recobrando audiencia: los juegos fáciles de aprender, los llamados juegos casuales. Juul (2010) clasifica a los juegos casuales en dos vertientes: los juegos con interfaces miméticas y juegos casuales descargables. Los juegos con interfaces miméticas son aquellos en donde la actividad física que el jugador realiza, imita lo que el videojuego muestra en la pantalla. Ejemplos de estos juegos son los creados por Nintendo Wii, en donde se puede jugar un partido de tenis haciendo un movimiento de los brazos como en un juego real, el *Rock Band* o *Dance Revolución.*

Los juegos casuales descargables son adquiridos en línea y se juegan en periodos cortos; generalmente, no requieren algún conocimiento previo sobre la historia del videojuego para poder jugarlo. Un ejemplo de este tipo es *Cake Mania 3.*

Esta clase de videojuegos han incrementado el número de usuarios, así como el rango de edades de los jugadores. Existe más gente que juega videojuegos de la que no lo hace.

Los videojuegos casuales son más populares que los videojuegos profesionales o dedicados a jugadores expertos. El concepto de juegos casuales o jugadores casuales inicia en el 2000 (Juul, 2010). Son completamente distintos a los videojuegos y jugadores “profesionales”, quienes prefieren videojuegos de fantasía, ciencia ficción, zombies, etc., y juegan diversos videojuegos de mayor dificultad, además de pasar largas horas jugándolos. Por el contrario, los jugadores casuales prefieren juegos positivos y placenteros, que les tome poco tiempo jugarlos y que sean sencillos de jugar.

Un jugador es alguien que interactúa con el juego, y el juego es algo que interactúa con el jugador (Juul, 2010). Mientras los desarrolladores de videojuegos “profesionales” son criticados por diseñar para ellos mismos, los desarrolladores de juegos casuales son reconocidos por diseñar juegos para la audiencia, sin tener que ser parte del videojuego. Los juegos de estrategias son claros para los que usualmente los juegan, saben de qué se tratan y están acostumbrados a las reglas y trucos que se presentan; aunque son una barrera para los que quieren acercarse o iniciarse en ellos. Los videojuegos casuales tienen historias pequeñas que contar para jugadores casuales.

En los videojuegos casuales los gráficos en 3D no son necesariamente importantes, lo esencial es diseñar para el jugador, tomando en consideración su tiempo y uso de su espacio en diferente manera que en los juegos en 3D.

Existen 2 diferentes espacios que involucran al videojuego y el jugador, el espacio en donde está sentado o parado, que es llamado espacio del jugador, el espacio físico frente a la pantalla. La pantalla en sí misma es una superficie plana y es llamada el espacio de pantalla; y los juegos en 3D presentan un tercer espacio o dimensión llamado espacio 3-D (Juul, 2010). El movimiento en el espacio de la pantalla y en el espacio del jugador son esenciales para los juegos casuales. Los juegos casuales descargables generalmente son en dos dimensiones; es decir, se desarrollan en el espacio de la pantalla. Mientras que algunos juegos casuales con interfaces miméticas son en 3 dimensiones y dan mayor importancia a la interacción entre el jugador y el espacio del jugador. Con los videojuegos casuales el diseño en 2 dimensiones ha vuelto y concretamente ha desarrollado el diseño sobre el espacio del jugador.

Los videojuegos de imitación generan la acción en el espacio del jugador, pero muchos de ellos se juegan en el contexto social como parte de un espectáculo, haciéndolos más entretenidos.

Aspectos psicomotrices en la ejecución del violín.

Los músicos aprenden habilidades complejas motoras y auditivas. Algunos estudios han encontrado evidencia de las diferentes estructuras en la región del cerebro que está directamente ligada con la percepción, integración multimodal, control motriz y ejecución musical. Tocar un instrumento requiere leer notación musical y traducirla dentro de una secuencia, actividad motriz bi-manual, el desarrollo fino de las habilidades motrices con retroalimentación multi-sensorial, memorizar largos pasajes musicales, e improvisar música. Estas estructuras no nacen con el hombre, sino que con entrenamiento y práctica se puede desarrollar, sobre todo a edad temprana.

Tocar el violín requiere de habilidades excepcionales sobre todo con el arco y la digitación en la ejecución, esto implica un control y precisión temporal, espacial y motora mayor. Generalmente, esto se logra con años de práctica e iniciando a edades tempranas. La digitación está caracterizada por un mecanismo serial-paralelo, además del tiempo y la nota (la precisión espacial del contacto entre el dedo y la cuerda) son usualmente determinadas por la acción combinada de dos dedos de la mano izquierda (Altenmüller, 2006). Esta acción tiene un retardo de tiempo de 50ms que no es perceptible para el humano. Así como existe este retardo, podemos decir que existen muchos más; por ejemplo, los movimientos con el arco, el cambio de posiciones de la mano, el cambio de los dedos entre las cuerdas, el arco hacia arriba o hacia abajo, etc., todo lleva un retardo imperceptible.

Los ejecutantes deben saber el espacio correcto del intervalo sobre el diapasón, este corresponde a intervalos de medio tono de forma no lineal y sin trastes, como lo es en el caso de la guitarra.

En el violín, se toma como base la posición básica que es de: un tono del primero al segundo, un semitono del segundo al tercer, y un tono del tercero al

cuarto dedo. Esto tomando en cuenta la separación natural de los dedos debida a la contracción activa del supinador[1] del antebrazo que actúa sobre los flexores de éstos (Barrón, 2009).

Algunos autores han propuesto ejercicios para fortalecer los músculos de la mano izquierda antes de tratar de tocar con el arco, como tomar un acorde haciendo presión simultanea con los dedos y dejando las cuerdas libres de las pisadas, y con sus variantes en la posición de los dedos. Esto con el objetivo de desarrollar la función dinámica de la mano, es decir, la presión de los dedos y la elasticidad de cada movimiento quedará guardada en la memoria muscular del ejecutante (Barrón, 2009).

La percepción auditiva procede de un excelente nivel de habilidades motoras y de control. Existe una relación muy estrecha entre la percepción y la acción, todos al escuchar música tenemos una reacción; es una reacción que se tiene en la estructura cerebral y redes neuronales independientemente de si se es músico o no.

La diferencia radica en cómo es escuchada; por ejemplo, un músico puede identificar más fácilmente la diferencia entre una nota tocada con una guitarra y la misma nota tocada con un piano, en comparación con alguien que no es músico.

Otro aspecto es el de leer la partitura, el músico debe de traducir la nota que está en el papel y asociarla al apropiado comando motriz para generar un movimiento específico, adicionalmente debe de conseguir una estética. En los niveles básico o de principiantes, existe una traducción directa entre una nota con un movimiento (Altenmüller, 2006). Conforme se avanza en las habilidades, el músico deja de leer nota por nota y sus ojos se van anticipando a la que sigue dentro de un rango similar a como leemos palabras. Según estudios, no ha sido un factor para determinar quién es más virtuoso, si aquel que abarca un mayor rango de captación visual o aquel que no, se cree que tiene que ver con la decodificación de lo que se está leyendo, el mensaje que

1. Según el diccionario de la Real Academia Española supinación (del latín supinatĭo, -ōnis) f. Movimiento del antebrazo que hace girar la mano de dentro a fuera, presentando la palma.

se está enviando y la acción sensomotora que se genera.

Aspectos Pedagógicos. El juego y el videojuego como nuevas herramientas de aprendizaje.

Para Huizinga (1972) el juego es más viejo que la cultura, y la característica principal del juego es que éste es libre, es libertad. Con ella se relaciona directamente una segunda, el juego es desinteresado.

Huizinga plantea lo siguiente: El niño juega con una seriedad perfecta, juega y sabe jugar. El violinista siente una emoción sagrada, vive un mundo más allá del habitual y; sin embargo, sabe qué está ejecutando o, como se dice en muchos idiomas, está "jugando" (Gómez, 2003) .

Lo esencial de toda actividad musical es el juego (Huizinga, 1972). Anderson (2007) hace referencia sobre lo que Squire y Jankins (2003) toman como punto de partida sobre las propiedades y buen diseño motivan a los estudiantes a buscar libros de texto con una nueva actitud hacia adquirir conocimiento.

En un estudio realizado por Aranda y Sánchez-Navarro (2009) encontraron que los estudiantes no temen participar frente a los demás compañeros cuando se está jugando, ni temen cometer errores ya que se encuentran en un contexto de videojuego, lo que les permite probar nuevos métodos y formas de ejecutar las funciones implicadas en el videojuego sin temor a equivocarse, lo que genera un progreso significativo.

Los videojuegos son capaces de crear espacios de libertad y comunicación. Se cree que los videojuegos son individualistas y te hacen encerrar en el mundo propio, pero la realidad es que algunos de los videojuegos que cuentan con algún controlador o interfaz electrónica han generado comunidades de juego y mayor socialización por parte de los jugadores.

Existen diferentes estudios sobre la influencia de la tecnología en el aprendizaje de jóvenes y niños como el realizado por Jenson y De Castell (2009) sobre la reciente revolución de los controladores para videojuegos y el

aprendizaje que pueden ofrecer, el cual representa un significativo cambio epistemológico.

Jenson y De Castell (2009) mencionan que el aprendizaje de los videojuegos no está relacionado con el contenido o temática de éste, sino que surge a través de la activación y asociación de las imágenes del jugador con objetos, situaciones y eventos del juego.

Las personas estamos más familiarizadas en las ventajas que los controladores de videojuegos de simulación nos pueden dar, como por ejemplo los simuladores de vuelo para entrenamiento de pilotos. Sin embargo, los controladores han cambiado radicalmente permitiendo tener una relación entre el acto que hace el jugador con lo que hace el avatar en un juego, provocando una imitación, por ejemplo, el controlador de batería del videojuego Rock Band.

Mientras que en la simulación se emplea el “como si” fuera real, en la imitación se emplea “al igual que”. En cuestiones educativas, se ha demostrado que los niños aprenden más del ejemplo de sus padres o maestros, tratando de imitar su comportamiento o actitudes ante situaciones (Jensen y De Castell, 2009).

Los nuevos controladores facilitan un comportamiento imitado más real que los controladores clásicos. Estos controladores nuevos preparan los músculos y la percepción visual y del espacio.

Para Blaine (2005) la aplicación de un controlador es más utilizado para ganar un juego que pensado como para ser interpretado como un instrumento musical, es algo contrario a la noción que se tiene para el diseño de una interfaz musical.

Los niños tienen su primer encuentro con la música en los juguetes, videojuegos y otros medios, más que con un instrumento musical real.

La implicación es clara en jugadores donde la motivación por el perfeccionamiento del tiempo y la rítmica se alcanza para ganar un juego, considerando que el desarrollo de competencia con un instrumento para transmitir el resultado musicalmente hablando, suele ser la motivación detrás de tocar un controlador musical (Blaine, 2005) (Fig. 14).

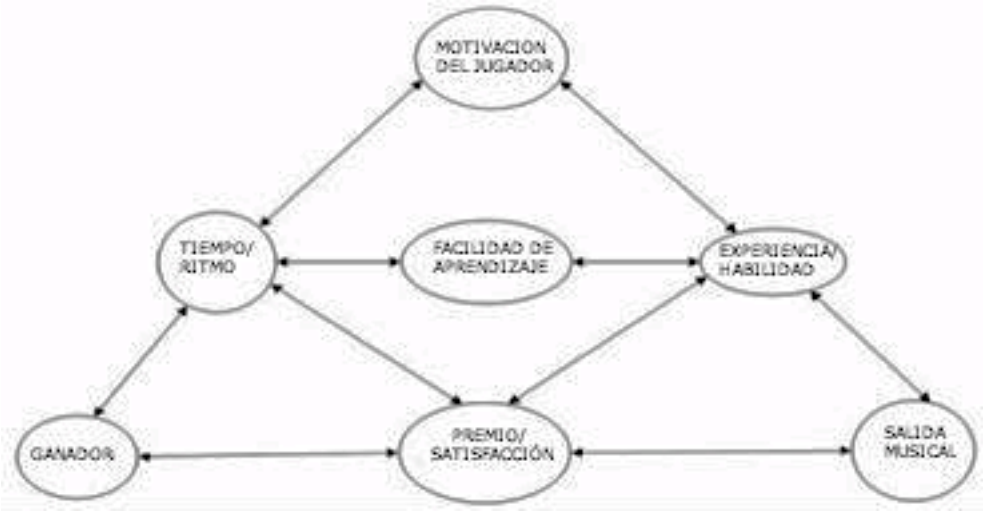

FIG. 14. MOTIVACIONES DETRÁS DEL JUEGO VS GANAR (BLAINE, 2005).

Otro aspecto es la motivación social, Blaine (2005) hace referencia a lo que Levitin sugiere: “que los instrumentos musicales deben de encontrar el justo equilibrio entre el desafío, la frustración y el aburrimiento. Dispositivos que son muy simples tienden a no proveer una experiencia enriquecedora, y dispositivos que son muy complejos apartan al usuario antes de que puedan beneficiarse de ellos”.

Para Blaine (2005) estos son los mismos principios del diseño de videojuegos para el aprendizaje y son fundamentales para el nivel de diseño usado en la construcción de la curva de interés de los jugadores (Fig. 15).

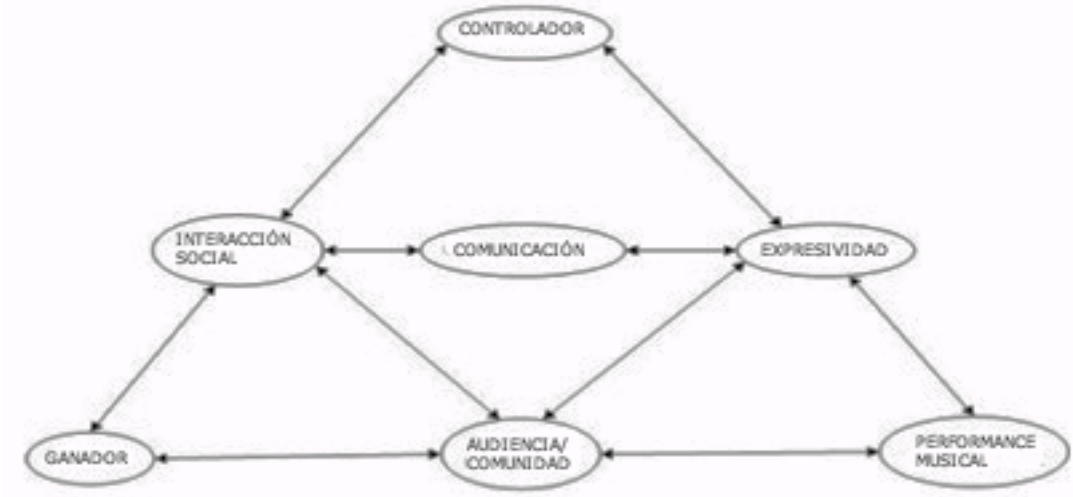

FIG. 15. IMPLICACIONES SOCIALES DEL JUEGO VS GANAR (BLAINE, 2005).

Consideraciones para un buen diseño de videojuegos aplicados al aprendizaje

Los videojuegos, se pueden dividir en función de su propósito, una de estas divisiones son los videojuegos serios. El propósito de un videojuego serio es dar a los usuarios la interacción con aplicaciones de Tecnologías de la Información (TI) que combinen aspectos tutoriales, de enseñanza, de entrenamiento, de comunicaciones y de información, con los elementos recreativos y tecnológicos derivados de los videojuegos (Alvarez, 2008).

Para Alvarez (2008) esta combinación de elementos tiene como objetivo ser práctico, con contenido útil (serio) y disfrutable (juego). Lo anterior, se consigue con el desarrollo de escenarios que son al mismo tiempo cumplen con estos objetivos.

Los juegos serios son empleados en diferentes áreas:

- Defensa
- Enseñanza y entrenamiento
- Publicidad
- Información y comunicación
- Salud
- Cultura
- Activismo

El sector menos desarrollado de los videojuegos serios es el cultural. Los inicios de los juegos serios datan de los años 80´s con videojuegos enfocados a la milicia para crear mayor interés en los jóvenes y poder reclutar más soldados.

Antes de 1973, se creó el juego *The Oregon Trail* dentro de los *edugames* por *Minnesota Educational Computing Consortium* (MECC) que trataba de enseñar sobre la colonización de América.

En su estudio sobre el uso de los videojuego para reforzar el aprendizaje adquirido, durante la clase teórica de música, Avery concluye que un buen

diseño de un videojuego para niños debe de tener: poco texto, botones grandes y coloridos -ya que son más llamativos para ellos-, mientras menos texto mejor, sobre todo para los niños con edades alrededor de los 5 años. Entre otros puntos destaca:

- No deben de contener mucha información.
- No deben de tener instrucciones largas y ambiguas.
- No deben de tener tareas repetitivas que los puedan aburrir.
- No debe de tener tiempos musicales rápidos y difíciles de seguir.

Los factores que deben de potencializarse durante el diseño de un videojuego según Padilla et al. (2009) para hacer el aprendizaje más efectivo son: motivación (Malone, 1987), atención (Keller y Kopp, 1987), concentración (Csiksentmihályi, 1990) y emoción (Norman, 2004), y el objetivo principal según Padilla et al., debe ser jugar.

La jugabilidad como la define Padilla et al. (2009), es el conjunto de propiedades que describen la experiencia del jugador ante un sistema de juego determinado, cuyo principal objetivo es divertir y entretener “de forma satisfactoria y creíble”, ya sea solo o en compañía.

Los atributos que proponen para caracterizar la jugabilidad son:

- Satisfacción: Agrado o complacencia del jugador al juego.

- Aprendizaje: Facilidad para comprender el sistema y la mecánica del videojuego: objetivos, reglas y formas de interaccionar con el videojuego.

- Eficiencia y Efectividad: Tiempo y recursos necesarios para lograr los objetivos propuestos en el videojuego.

- Inmersión: Capacidad del videojuego para creerse lo que se está jugando e integrarse en el mundo virtual mostrado en el videojuego.

- Motivación: Característica del videojuego que mueva a la persona a realizar determinadas acciones y persistir en ellas para su culminación.

- Emoción: Impulsos involuntarios originados como respuesta a los estímulos del videojuego que induce sentimientos y desencadena conductas de reacción automáticas.

- Socialización: Atributos y elementos del juego que fomentan el factor social o la experiencia en grupo.

Padilla et al. (2009) realizan una descomposición basada en las facetas de la jugabilidad, ya que el análisis en un videojuego es un proceso complejo, lo que permite identificar distintos atributos de la jugabilidad:

1. Jugabilidad intrínseca.- Es la jugabilidad medida en la propia naturaleza del juego; es decir, en las reglas, objetivos, retos, y cómo estos se proyectan al jugador. Está ligada al diseño del *gameplay* y a la implementación del *game machine*, analizando cómo se representan las reglas, objetivos, ritmo y mecánica del videojuego.

2. Jugabilidad mecánica.- Es la asociada a la calidad del videojuego como sistema (software). Está ligada al *game engine*, haciendo hincapié en características como la fluidez de las escenas cinemáticas, la correcta iluminación, sonido, movimientos gráficos y comportamiento de los personajes del juego y del entorno, sin olvidar los sistemas de comunicación en videojuegos *multiplayer*.

3. Jugabilidad Interactiva.- Es la faceta asociada a todo lo relacionado a la interacción con el usuario, diseño de interfaz de usuario, mecanismos de diálogo y sistemas de control. Está fuertemente unida al *game interfaz*.

4. Jugabilidad artística.- Asociada a la calidad y adecuación artística y estética de todos los elementos del videojuego a la naturaleza de éste. Entre ellos están la calidad gráfica y visual, los efectos sonoros, banda sonora y melodías del juego; la historia y la forma de narración de ésta, así como la ambientación realizada de todos estos elementos del videojuego.

5. Jugabilidad intrapersonal.- Esta faceta tiene como objeto estudiar la percepción que tiene el propio usuario del videojuego y los sentimientos que a éste le producen. Tiene un valor subjetivo.

6. Jugabilidad interpersonal.- Muestra las sensaciones o percepciones de los usuarios y la conciencia de grupo, que aparecen cuando se juega en compañía.

Aspectos de diseño.

Estructura de los sistemas interactivos.

Existen diferentes tipos de sistemas interactivos. Dentro de éstos, cuando se habla de diseño de software y diseño de hardware, no se habla sólo de diseño de sistemas interactivos, sino del diseño de producto. Las claves que se consideran en el diseño de un sistema interactivo (ISD por sus siglas en inglés) según Benyon (2009) son:

- Diseño.- Qué se diseña y cómo debe ser hecho.

- Tecnologías.- Sistemas interactivos, productos, dispositivos y componentes.

- Gente.- Quiénes usarán el sistema y en qué mejorarán sus vidas con estos diseños.

Actividades y contextos.- Lo que la gente quiere hacer y el contexto en el que se realizan estas actividades.

"¿Qué es el diseño? Es lo que se entiende en dos palabras –el mundo tecnológico y el mundo de la gente y propósitos humanos- y uno trata de poner las dos juntas" (Kapor, 1991).

El diseño lo podemos entender como la visualización de un objeto, su plan para crearlo y la manera en que se va a representar. Por un lado, se puede hablar de la parte del diseño técnico en el que se ven todos los aspectos científicos y especificaciones técnicas del objeto; y por otro lado, el diseño creativo o

artístico en donde se involucra más a la imaginación y a las ideas conceptuales.

Los sistemas interactivos son dispositivos y sistemas que responden a acciones de la gente, procesando la información mediante componentes, software o hardware.

En un buen diseño se tiene que tomar en cuenta el punto de vista humano y no el mecánico, se debe diseñar tomando en cuenta cómo piensa el hombre y no la máquina (Benyon, 2010).

Como Benyon menciona: la interfaz forma parte de un sistema interactivo con el cual se puede tener contacto física, perceptiva y conceptualmente.

- Físicamente deben interactuar con un dispositivo por medio de presionar botones, mover niveles, etc., y el sistema interactivo debe responder mediante una retroalimentación a partir de éstos.

- Perceptualmente el dispositivo debe mostrar en la pantalla cosas que el usuario pueda ver o generar sonido para que pueda escuchar o ver.

- Conceptualmente interactuar con un dispositivo tratando de trabajar con éste sobre qué es y qué debería de ser. El dispositivo provee mensajes diseñados para ayudar a entenderlo.

Los diseñadores de sistemas interactivos están conectando personas a través de dispositivos y sistemas; ellos deben de considerar todo el ambiente que están creando (Benyon, 2010).

Cuatro puntos básicos para el diseño de sistemas interactivos.

La tecnología es el soporte para que un amplio rango de personas desarrollen varias actividades en diferentes contextos.

Para hacer un buen diseño tecnológico interactivo debemos entender las variables inherentes a este proceso y que deben ser tomadas en cuenta.

1. Usuarios.

1.1. Diferencias Físicas.- En este punto uno de los aspectos más estudiados y cuidados es la ergonomía, además del aspecto físico de los usuarios, es decir, altura, talla, peso, capacidades especiales, etc.

1.2. Diferencias Psicológicas.- La habilidad especial que cada usuario tenga; por ejemplo, la capacidad de memorizar imágenes, la percepción del espacio, la asociación de objetos con conceptos, etc.

1.3. Modelos mentales.- Un sistema debe ser fácil de entender y de comunicarse con la gente que usa el sistema, debe de lograr una clara concepción o crear un mapa mental del sistema en cada usuario.

1.4. Diferencias sociales.- Se debe tomar en cuenta hacia que usuarios va dirigido el diseño, si son grupos reducidos o grandes, etc.

2. Actividades.

2.1. Aspectos temporales en todo diseño. Qué tan frecuente o no es el uso del dispositivo. Por ejemplo; un celular es un dispositivo que se utiliza frecuentemente durante el día y a diario, por lo que hacer una llamada y aprender el método de cómo se marca nos resulta después de cierto tiempo de uso muy fácil de realizar, mas sin embargo, cambiar la batería es una actividad que no se suele realizar frecuentemente por lo que no sabremos hacerlo cuando llega el momento.

2.2. Tiempo de respuesta del dispositivo. Como regla general, un usuario espera un tiempo de respuesta alrededor de 100 milisegundos que es captado por el ojo antes de que la mano realice alguna actividad, y un segundo para la relación causa-efecto de presionar un botón y pase algo. Algo más de 5 segundos puede hacernos sentir frustrados o confundidos.

2.3. Los datos requeridos para cada actividad. Analizar qué actividad se va a desarrollar y qué datos va a requerir para poder definir o implementar un dispositivo que colecte esa cantidad de información de manera eficaz.

3. Contexto.

La actividad y el contexto comúnmente van ligadas dependiendo de las circunstancias.

3.1. Ambiente físico.- El ambiente en el cual se va a desarrollar la actividad, por ejemplo: si es húmedo, ruidoso, con mucha luz, la ubicación geográfica, etc.

3.2. Contexto social.- En ocasiones las normas sociales marcan la aceptación o rechazo de un nuevo diseño.

3.3. Contexto organizacional.

4. Tecnologías.

Es el medio que los diseñadores de sistemas interactivos utilizan para trabajar en sus diseños. Los diseñadores necesitan entender los materiales con los que trabajan, deben de tener en cuenta varias posibilidades para diferentes entradas, salidas, comunicación y contenido.

4.1. Entradas.- Se concentran en cómo los usuarios ingresan datos o instrucciones a un sistema. Los más comunes son interruptores y botones porque son fáciles de usar pero, en la mayoría de los casos, utilizan mucho espacio por lo que se tiene que ser cuidadoso de qué se va a controlar con ellos.

4.2. Salidas.- Las tecnologías para mostrar información a los usuarios se enfocan en tres aspectos de percepción visual, auditiva y táctil; el dispositivo más utilizado ha sido una pantalla. La háptica se refiere a la sensación de tocar; nos permite estar en contacto con dispositivos y medios interactivos en un modo directo e inmediato, lo que nos da en muchas ocasiones una retroalimentación de sentir como si realmente estuvieran sucediendo ciertas cosas.

4.3. Comunicación.- La comunicación entre los dispositivos y los usuarios es esencial en el diseño. La conectividad actual de los dispositivos como el ancho de banda o la velocidad se han vuelto un reto.

4.4. Contenido.- Se refiere al tipo de datos que tiene el sistema y la forma en que están estructurados. En algunos dispositivos, el contenido trata de un todo; por ejemplo, las páginas web que usualmente contienen mucha información relativa a la página. Otras tecnologías se concentran más en el funcionamiento; por ejemplo, un control remoto de televisión. Algunas otras tecnologías tienen una mezcla de ambas.

El objetivo de diseñar un sistema interactivo es poder combinar todos estos aspectos del PACT (*People, Activities, Context, Technologies*). Para la gente, los diseñadores deberían pensar sobre las diferencias físicas, psicológicas y sociales y cómo estas diferencias cambian dependiendo de las circunstancias (Beyon, 2010).

Procesos para el diseño de sistemas interactivos.

Como primer paso, se debe entender que tiene que hacer el sistema, cómo debe verse y cómo debe interactuar con otros objetos. Los requerimientos funcionales conciernen con lo que el sistema debe ser capaz de hacer y sus limitaciones.

Se puede considerar un diseño conceptual en el que se debe tomar en cuenta qué información y funciones debe requerir el sistema para lograr sus objetivos, decidir qué herramientas debe conocer el usuario y definir una clara idea de la solución para la interfaz y su comunicación con el usuario.

Dentro de las técnicas para obtener una idea más clara de lo que se quiere, está plasmar todas las ideas en dibujos, bosquejos o diagramas, enfocándose al “qué” sobre el “cómo”.

También se debe hacer un diseño físico que involucra el cómo van a trabajar los dispositivos, qué aspecto tendrán y la idea estética del producto final. Es

trasladar un concepto a algo concreto. Existen tres aspectos fundamentales en el diseño físico según Beyon (2010):

- Diseño operativo.- Especificar cómo debe de trabajar todo y cómo está compuesto estructuralmente. Los eventos o acciones que se deben de activar para generar otro tipo de estado del dispositivo.

- Representaciones.- Concerniente a la parte estética y de estilo que es importante, ya que provoca en la gente sentimientos y primeras impresiones.

- Diseño interactivo.- Se refiere a la designación de funciones entre las personas y las tecnologías y la secuencia que deben seguir los usuarios para utilizarlas.

Se debe tener una clara idea visual para hacer más fácil el diseño y poder planear estrategias de pruebas.

Escenarios

Un buen diseño está pensado para las personas y no sólo para el diseñador. Cuando se diseña se identifican algunas mejoras que se aplican a determinadas circunstancias, lo que se llama escenarios concretos; éstos se enfocan específicamente en el diseño de una interfaz y en la designación de funciones específicas entre la persona y el dispositivo.

Se pueden utilizar casos de aplicación que describen la interacción entre los usuarios y el dispositivo; es decir, desarrollan qué hacen los usuarios y lo que debería hacer el dispositivo o sistema hacia quienes va dirigido. Las especificaciones de los casos de aplicación informan sobre procesos de tareas y funciones designadas.

Durante todos estos procesos de análisis sobre el diseño de la interfaz deben ser identificados varios problemas y deben ser establecidos como una lista de requisitos a evaluar.

Usabilidad

De acuerdo a un compilado internacional expuesto por el Centro de Diseño Internacional del Colegio de Diseño de la Universidad del Norte de Carolina, en Estados Unidos, expresa lo siguiente:

- Uso equitativo.- El diseño no excluye ni estigmatiza a ningún grupo de usuarios.
- Flexibilidad de uso.- El diseño aloja un amplio rango de preferencias individuales y habilidades.
- Uso simple e intuitivo.- Su uso debe ser sencillo y fácil, independientemente de la experiencia del usuario, su conocimiento, habilidades de lenguaje o nivel de concentración.
- Información perceptiva.- Debe comunicar información eficientemente al usuario, independientemente de las condiciones en el medio o las habilidades sensoriales del usuario.
- Tolerancia al error.- El diseño minimiza riesgos y consecuencias desfavorables de accidentes o acciones no intencionales.
- Esfuerzo físico moderado.- El diseño puede ser usado eficientemente y cómodamente, con un mínimo de fatiga.
- Tamaño y espacio para el acceso de usuario.- Un tamaño y espacio apropiado para el alcance, manipulación y uso, independientemente de la complexión de cuerpo del usuario, postura o movilidad.

La usabilidad se enfoca en que el usuario haga cosas con el mínimo esfuerzo, que las aplicaciones tengan el contenido y funciones apropiadas y que éstas sean lógicamente organizadas, que sean fáciles de recordar a pesar de no hacerla cotidianamente, y sobretodo que sean seguras para los usuarios.

Otro factor importante en un diseño es la aceptabilidad; que se mide -a diferencia de la usabilidad- en el contexto del uso. Los usuarios aceptan el diseño si los convence, si no los obliga a nada, si es económico o su valor monetario es justo para su uso.

Beyon (2010) se refiere a 12 principios del diseño propuestos por Norman (1998), Niels (1993) y otros, los cuales son:

1. Visibilidad.- Asegurarse de que los objetos son visibles, que el usuario se dé cuenta visualmente que debe de hacer una función y que se está ejecutando en el sistema.

2. Consistencia.- Ser congruente en hacia dónde va enfocado el diseño, la compatibilidad con sistemas similares y en la forma de trabajar. Conceptual y físicamente es importante.

3. Familiaridad.- Usar símbolos o lenguajes con los que el usuario se identifique. Sino es posible porque el sistema no lo permite, buscar analogías que ayude a los usuarios a asociar los nuevos símbolos o lenguajes con los ya conocidos.

4. *Affordance* (Permisividad).- Se refiere a las propiedades que tiene un objeto o que se perciben del mismo, y cómo éstas se relacionan para que el usuario las pueda utilizar.

5. Navegación.- Facilita a los usuarios moverse alrededor de las partes del sistema.

6. Control.- Dejar claro qué o quién tiene el mando y permitir al usuario que tenga el dominio. El control es mejor si está claro lo que se tiene que hacer. También es necesario dejar clara la relación entre que debe hacer el sistema y qué pasará dentro o fuera de él.

7. Retroalimentación.- Dar información al usuario para que conozca qué efecto causan sus acciones. Una constante retroalimentación causa en los usuarios la sensación de control.

8. Recuperación.- Reponerse de acciones, principalmente de errores y problemas, rápida y efectivamente.

9. Restricciones.- Son las acciones que los usuarios no deben de tratar de hacer o son inapropiadas. Los usuarios deben ser prevenidos de errores u operaciones riesgosas.

10. Flexibilidad.- Permite al usuario múltiples formas de hacer las cosas o acceder a diferentes niveles de experiencias o intereses del sistema. Permite el avance hacia otro ambiente o personalizar el sistema.

11. Estilo.- El diseño debe ser estético y atractivo.

12. Convivencia.- Los sistemas interactivos deben ser amigables y generalmente placenteros, no deben contener mensajes agresivos o irrupciones abruptas.

Los puntos 1 a 4 se enfocan al fácil acceso de los usuarios al aprendizaje y memorización del sistema. Los puntos 5 a 7 hacen referencia a dar la sensación de control, sabiendo qué hacer y cómo hacerlo. Los punto 8 a 9 hablan sobre la seguridad y estabilidad del sistema. Y los puntos 10 a 12 a la forma más conveniente de hacer el diseño flexible, placentero, amigable y adaptable para cada determinado tipo de usuario.

Visualización.

La visualización ayuda a los diseñadores a observar las cosas desde diferentes perspectivas. También a generar diferentes representaciones de las ideas de diferente forma. Por ejemplo, a diferentes usuarios puede enriquecer más un diseño y dejarlo más claro que otro. Escoger la mejor forma de representar una idea para determinados usuarios es parte de las habilidades de un diseñador.

Entre las diferentes formas de representación se encuentran los bocetos y fotos como primer acercamiento, guiones gráficos, mapas de navegación, prototipos, etc.; cualquier forma de comunicar y dejar más clara la idea.

Un prototipo es una representación o implementación concreta pero parcial de un diseño; es útil para ver detalles y valorar muchos de los elementos del diseño (contenido, visuales, interactividad, funcionalidad), con una evaluación

afirmativa se puede proceder a construir o implementar el producto final o definitivo.

Capítulo 3. Descripción del proyecto.

El capítulo 3 inicia con una descripción general de los elementos con los que se desarrolla el prototipo, para después hacer una descripción de cada una de las partes esenciales que lo componen: la parte visual y la parte electrónica. La parte electrónica habla del cómo se construyó y sus elementos que la conforman, por último se menciona la parte visual propia del videojuego, en la que describe cómo se programó y consideró cada uno de los escenarios que lo conforman.

Descripción general.

Para iniciar con el desarrollo de este proyecto, y una vez planteada la idea y objetivo al que se quiere llegar, se inicia con la parte del videojuego. Para esta etapa se realizan bosquejos del diseño que deberá tener cada pantalla del juego. Una vez que se tienen los diseños en papel, se plasma el diseño del dibujo en la computadora mediante algunos programas de edición fotográfica. Obtenidos los diseños de los fondos de pantalla y los elementos que están en cada pantalla (botones, sliders, etc.), se consignan las funciones de cada uno de éstos con el programa de las partituras y la música para que puedan ser modificados.

Se realiza un programa mediante el lenguaje de programación Processing2.0 en el cual se muestra la partitura que va a ser utilizada por el jugador, este programa general se toma como base para la elaboración de las demás partituras, modificando así sólo las alturas y duraciones de las notas para cada una de ellas, con esto el programa generará una partitura nueva dependiendo de los parámetros dados.

Al mismo tiempo, se hace otro programa en Processing2.0 que genera tonos y ritmos dependiendo de los parámetros dados. Al igual que el programa que dibuja la partitura, este es general para todas las canciones, por lo que sólo se modifican los parámetros de las alturas y las duraciones de las notas para generar una nueva melodía. Este programa muestra la nota que debe ser

tocada por el jugador sobre la partitura previamente dibujada, la cual debe coincidir con la nota que se escucha.

Una vez obtenido esto, se hace la programación que genera una salida en el puerto serial que transmite datos a un microcontrolador conectado a la tableta de pruebas con los LEDs. Esta salida serial depende de las alturas y duraciones de cada nota, con lo que se obtiene una sincronización entre la lectura de la partitura en la pantalla y los LEDs

A cada LED de la tableta de pruebas -y posteriormente a la plantilla- se le asigna un número que corresponde a una nota de la partitura. Este número depende de la nota que va en turno y es enviado a la salida serial para ser manipulado por el microcontrolador y mandar esa señal a cada LED, el cual corresponde a una digitación determinada en el violín dependiendo de la nota. La asignación del número se realiza mediante la programación de un microcontrolador.

Para el diseño de la interfaz electrónica se toma en cuenta el confort del ejecutante, así como el lugar que corresponde a la pisada en el violín de cada dedo. Esta interfaz electrónica indica qué posición o dedo se debe usar, dependiendo de la nota correspondiente en la partitura, en el tiempo exacto en el que se va desplazando la nota.

Para esta interfaz es necesario llevar a cabo pruebas con diferentes materiales para hacer una interfaz electrónica adaptable, ergonómica y de fácil montaje que se adhiera al diapasón del violín, sin que se vean modificadas las dimensiones del instrumento o altere su ejecución normal.

Desarrollo.

Como primer paso se crea un diseño conceptual sobre el funcionamiento de la interfaz y qué alcances deber de tener este proyecto. Partiendo de algunos diseños existentes creados por otras universidades, se analizan las diferentes opciones tecnológicas para iniciar con el diseño. La ergonomía de los materiales a utilizar es esencial en el diseño, por lo que es necesario buscar

materiales y componentes que no entorpezcan la ejecución del instrumento, sobre todo, los que pueden ser colocados entre el diapasón y las cuerdas del violín. Considerando algunos materiales que pueden funcionar, se realiza un diseño conceptual de la interfaz electrónica.

En cuanto a la interfaz gráfica, existen diferentes librerías y programas para poder hacer la parte de la partitura. Con estas librerías se hacen varios bosquejos o esquemas que se pueden utilizar; como por ejemplo, el que se muestra en la figura 16. En éste se indica la conexión entre la partitura, la lectura de datos, la salida de datos, la conexión entre la interfaz electrónica y la gráfica; y el cómo la información se puede desplegar en la pantalla de la computadora.

De la idea plateada en el esquema se parte para realizar el diseño de cada uno de los elementos que conforman el videojuego.

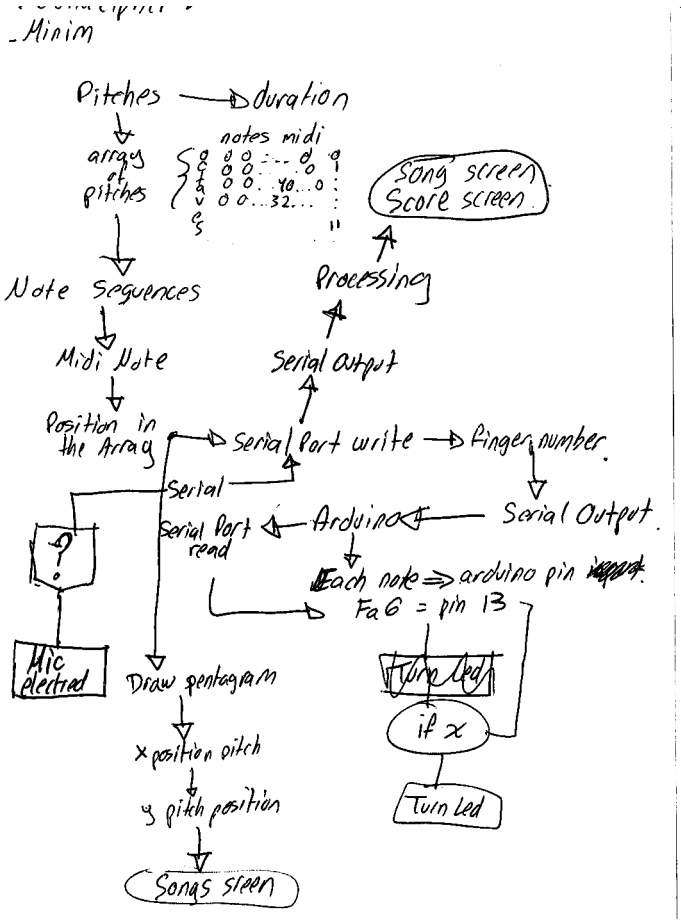

FIG. 16. ESQUEMA DE DISEÑO CONCEPTUAL DEL SISTEMA INTERACTIVO

Diseño de la interfaz electrónica.

Construcción.

La idea del uso de los materiales surge a partir de un proyecto de arte electrónico realizado en el MediaLab del Instituto Tecnológico de Massachusetts (MIT, por sus siglas en inglés) con tinta conductora sobre papel bond, por lo que se eligen materiales similares a los ocupados en dicho proyecto. Lo primero a considerar para la elección de los materiales es que éstos sean económicos. Debido a que el especio entre el diapasón y la cuerda es muy estrecho, del orden de 0.8mm a 1.0mm, se considera el uso de LEDs SMD que tienen dimensiones menores al resto de los LEDs (3.2mm largo x 1.6mm ancho y 0.75mm de alto) por lo que no interfieren con las cuerdas del violín.

Otra característica importante es que la cantidad de corriente eléctrica que circula por cada LED es muy baja, por lo que una pista dibujada con tinta conductora y la salida del microcontrolador pueden soportar la demanda de dicha corriente.

Se considera el uso de tinta conductora para mantener la plantilla lo más delgada posible, evitando que ésta interfiriera con las cuerdas del violín y altere la vibración propia de la cuerda.

Como siguiente paso, se analizan diferentes materiales en los que se puede dibujar con la tinta conductora y al mismo tiempo colocar los LEDs. Viendo las necesidades, y teniendo como principal punto de referencia el uso de la tinta, se recurre a la idea de las impresoras láser o de inyección, las cuales utilizan tinta y se puede imprimir en diferentes materiales como hojas de papel mate, papel brillante de diferentes grosores, y acetatos. Estos últimos debido a su textura y flexibilidad, presentan ventajas para el diseño ya que son más resistentes que el papel común además de ser igualmente manejables.

Se realizan diferentes bosquejos para ver la forma más óptima que debe tener el acetato sobre el diapasón, y cómo se debe sujetar sobre éste. La mejor opción es considerar el largo de medio diapasón y colocar tres pestañas para

tener una mejor sujeción y así evitar un posible movimiento debido a los cables.

Debido a que el diseño conceptual está pensado para que la plantilla se pueda poner y quitar fácilmente del violín, se utiliza velcro ya que es un material sencillo de usar que cumple con el propósito, ya que además de ser fácil de utilizar, mantiene con mayor firmeza la plantilla en la posición necesaria y no daña el barniz del violín.

Una vez determinadas las dimensiones y forma de la plantilla con el acetato, se dibujan las líneas de tinta conductora sobre éste, a manera de pistas de circuito impreso. Posteriormente, se realizaron pruebas de conductividad entre las terminales de cada pista utilizando un multímetro digital, lo que nos permite saber si existe o no conductividad entre los extremos de las pistas dibujadas. Una vez realizadas estas pruebas con la tinta, y observado que no se corre y tiene buena conductividad, se realizan varios bosquejos a lápiz para ver la trayectoria que deben recorrer las pistas y no rebasar las dimensiones de la plantilla. Así mismo, se tiene que cuidar la separación entre las pistas considerando el ancho del punto de la pluma de la tinta conductiva, como se muestra en la figura 17.

La mejor opción para la disposición de las pistas es considerar la tierra del circuito impreso del lado más angosto del diapasón -ya que la tierra es común para todos los LEDs- lo que nos simplifica su diseño a cuatro pistas que se unen en un punto para lograr una sola tierra. Las otras dieciséis pistas se dirigen hacia la parte más gruesa del diapasón para tener mayor separación entre pistas.

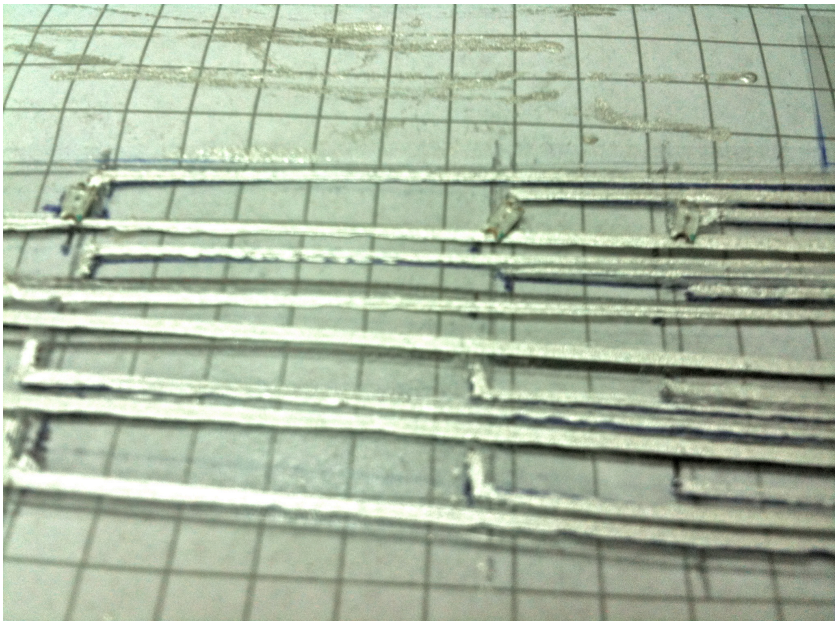

FIG. 17. PISTAS PARA PLANTILLA DE LEDS CON TINTA CONDUCTORA

Después se trabaja con pruebas de fijación del LED sobre el papel acetato. Primero se trató de soldar directamente el LED sobre el acetato, estañando las terminales del LED y acercándolo al acetato con tinta pero en algunas ocasiones el acetato se quemó. Con esta problemática se recurre a otra forma de colocar el LED haga contacto con las pistas. Lo anterior se logra colocando una pequeña gota de pegamento en el centro del LED sin que toque sus terminales ánodo-cátodo para que estas, al presionar el LED sobre el acetato haga un contacto eficaz con la tinta conductiva y pueda conducir la corriente.

En algunos casos se reforzó el contacto eléctrico con soldadura de estaño o tinta conductiva entre las terminales de los LEDs y las pistas para hacer un mejor contacto.

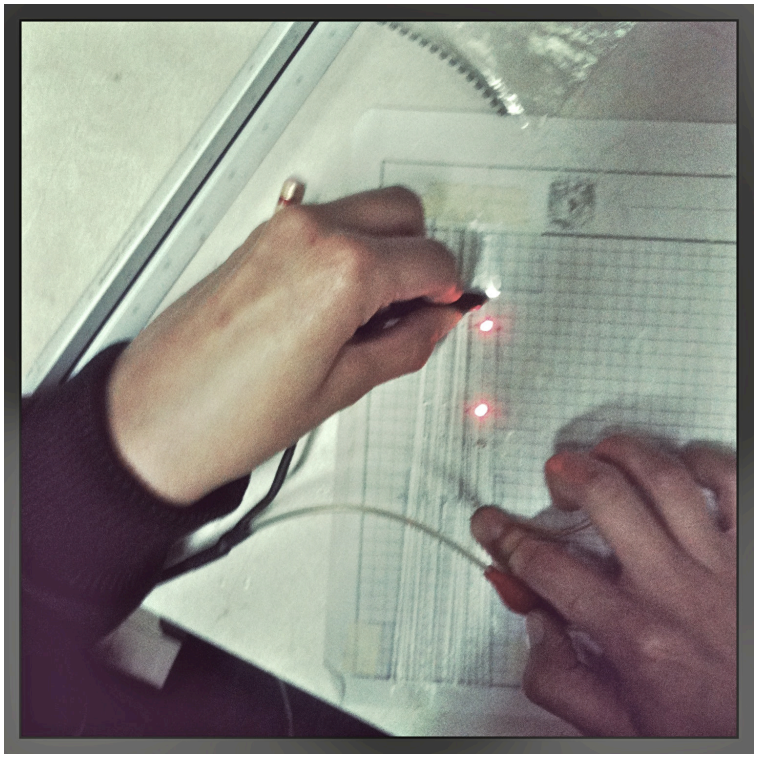

FIG. 18. PRUEBA DE LEDS SOBRE LAS PISTAS DE TINTA CONDUCTORA Y ACETATO

Se prueba la continuidad, con la ayuda de un multímetro, cómo se muestra en la Fig. 18. Para ver el voltaje que soportan estos LEDs se conectó uno de ellos a una fuente de poder de corriente directa y se le suministro voltaje a partir de 2V y al llegar a 3V el LED se fundió. El objetivo de esta prueba fue determinar la corriente que soporta el LED antes de fundirse, la cual fue de 0.020A.

Se sabe que el Arduino suministra a cada Pin de salida un voltaje de 5V por lo que es necesario limitar principalmente la corriente que circula por el LED sin perder la luminosidad, para realizar esto se agrega una resistencia para controlar el voltaje suministrado por el Arduino.

Para saber el valor de la resistencia limitadora que se debe utilizar se recurre a la Ley de Ohm utilizando los datos de voltaje suministrado y corriente obtenidos en las mediciones. Por lo que

$$V = RI$$

Despejando R

$$R = V/I$$

$$R = 5V / 0.020A = 250\Omega$$

No existen valores comerciales de 250Ω y los valores comerciales más próximos son de 220Ω o 330Ω; debido a que la corriente que limita la resistencia de 220Ω excede la máxima que soporta el LED se utilizan resistencias de 330Ω. Estas resistencias son igualmente que los LEDs de montaje en superficie y se colocan al inicio de cada pista; se fijan con pegamento y se refuerzan las terminales aplicando un punto de tinta conductiva sobre las terminales como se muestra en la figura 19.

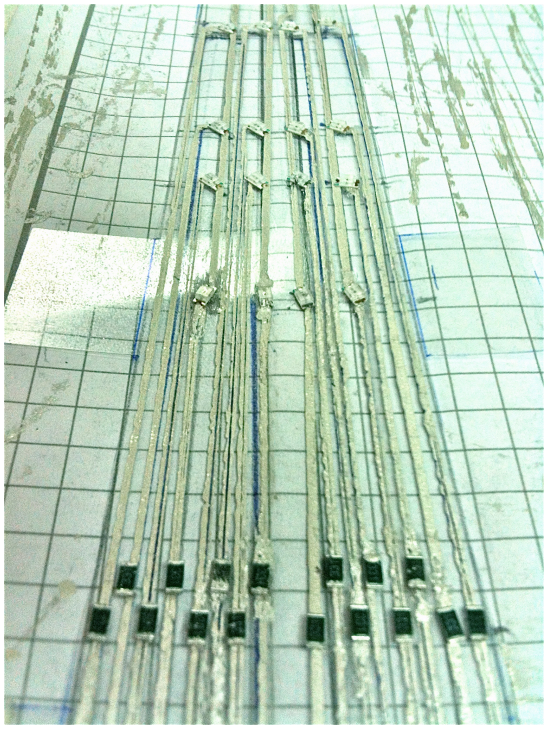

FIG. 19. RESISTENCIAS DE 330Ω Y LEDS SOBRE PISTAS DE TINTA CONDUCTORA EN ACETATO

En la figura 20, se aprecia el tamaño de la plantilla comparado con la dimensión de un lápiz.

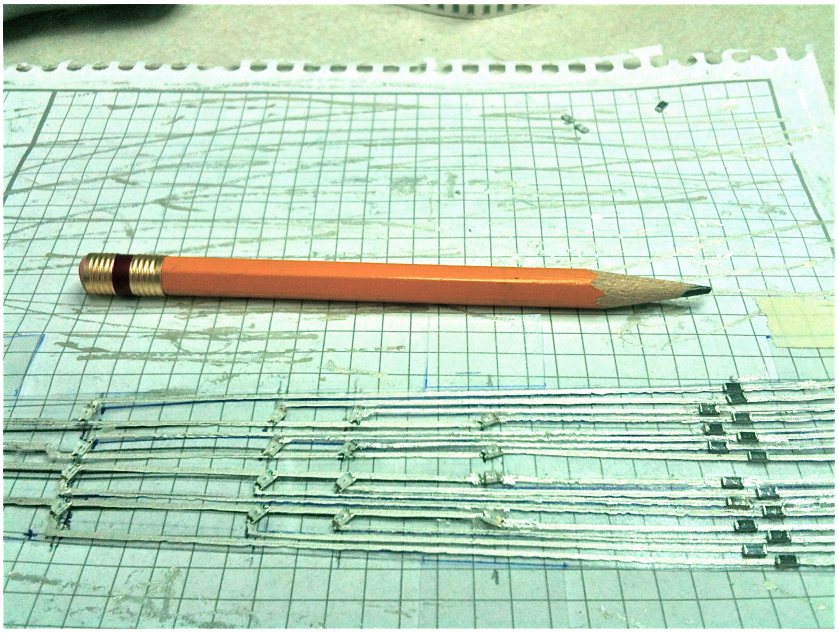

FIG. 20. COMPARATIVA DE TAMAÑO ENTRE LÁPIZ Y PLANTILLA

Para la conexión de las pistas de plata con el microcontrolador es necesario utilizar cables por lo que después de analizar diferentes opciones se opta por cables planos flexibles que son usados comúnmente en los rieles de las impresoras o dispositivos con movimiento debido a su flexibilidad y dimensiones. Estos cables por su grosor no interfieren ni aumenta las dimensiones pensadas para la interfaz electrónica además de venir agrupados en un mismo empaque lo que se evita tener cables sueltos que puedan enredarse unos con otros.

Estos cables se separan sólo en las orillas para poder soldarse a las pistas y colocarse en el microcontrolador mediante el uso de conectores tipo *headers*. Se parte la tira de cable a lo largo a la mitad para separar 8 pistas de un lado y 8 pistas del otro, también se considera un cable para la tierra.

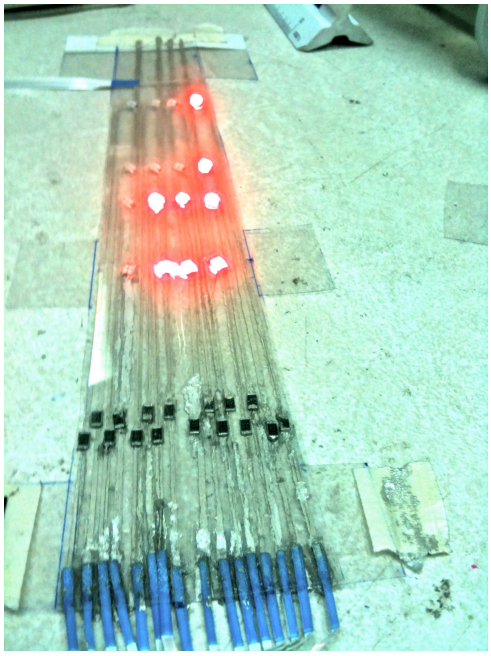

FIG. 21. PRUEBA DE LEDS CON VOLTAJE

Una vez ya armados los LEDs y antes de conectarse al microcontrolador, se hicieron pruebas punta a punta para verificar que todos los LEDs encendieran e identificar cada cable en el pin del microcontrolador que debe de ir.

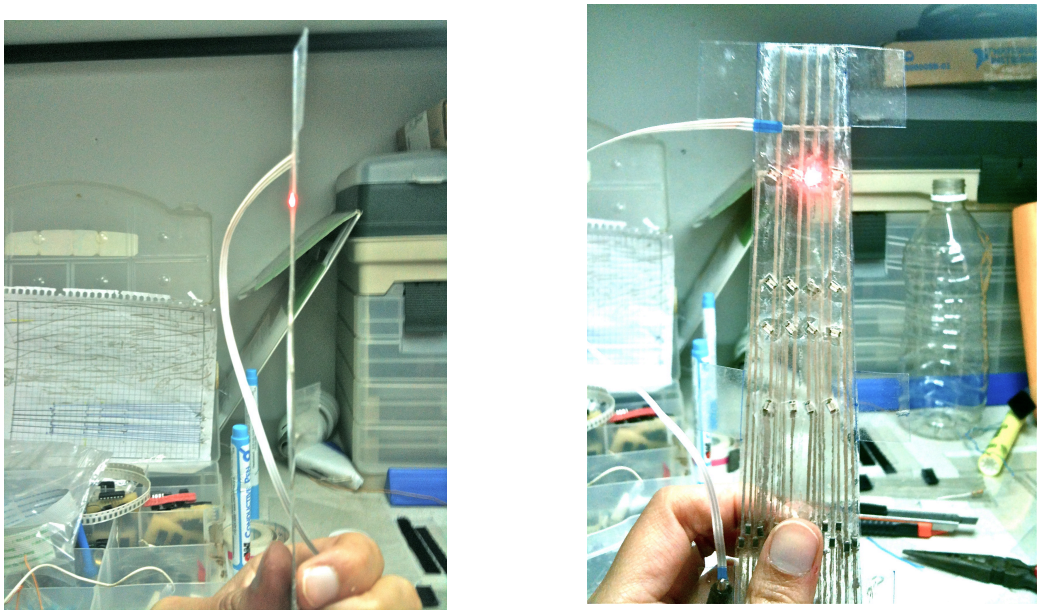

FIG. 22. DIMENSIONES DE GROSOR DE LA PLANTILLA

Una vez probados los LEDs, se hacen ensayos con la plantilla montada en el violín y se verifica que las dimensiones fueran las correctas y que nada interfiera con la ejecución del instrumento.

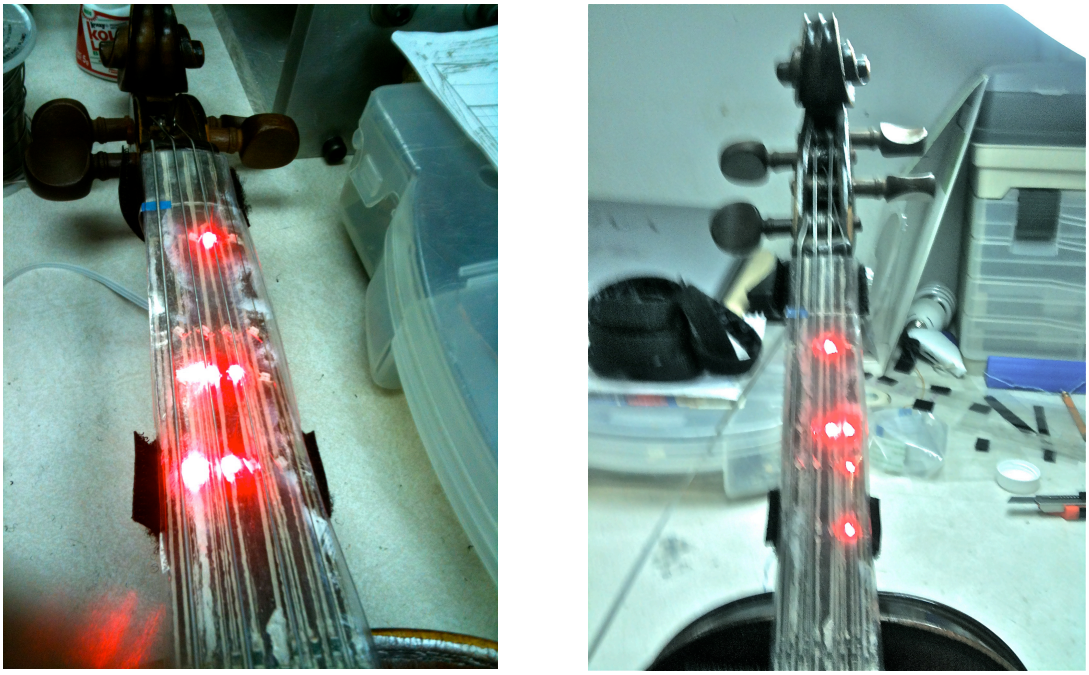

FIG. 23 PRUEBA DE LA PLANTILLA SOBRE LEDS

Como siguiente paso se conecta el microcontrolador, previamente programado y se vuelven a realizar pruebas para ver que los LEDs correspondan al pin adecuado del microcontrolador. Posteriormente, se busca la mejor forma de fijar el microcontrolador al violín, así como la mejor ubicación. La opción óptima es debajo del diapasón justo detrás del cuerpo del violín.

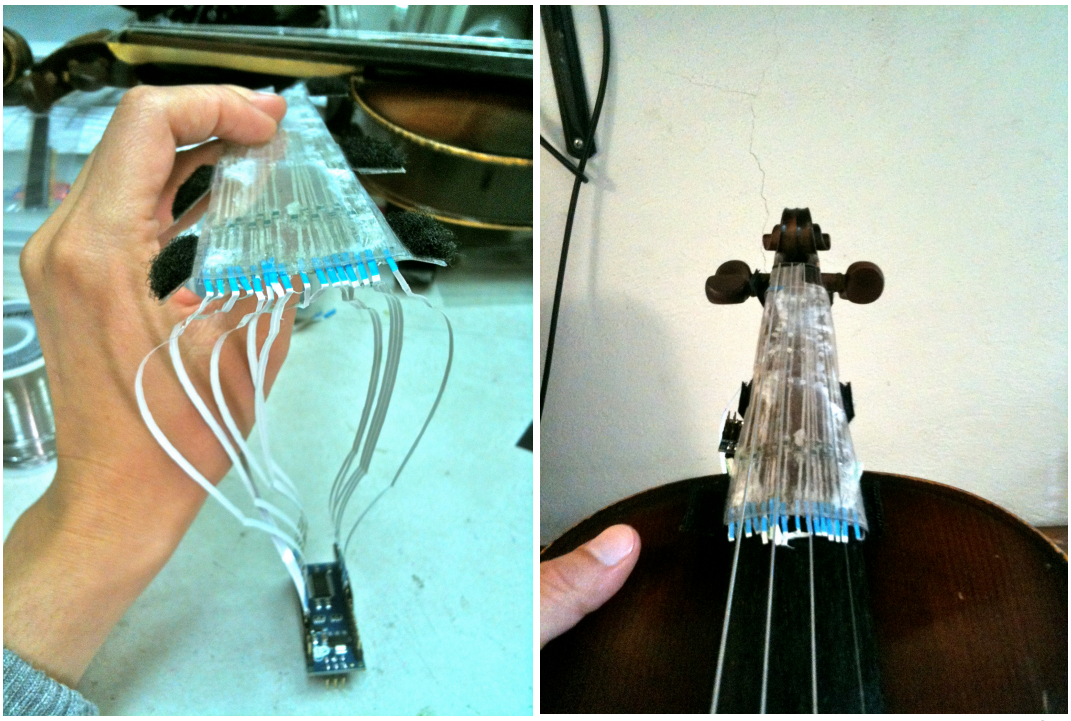

FIG. 24. IZQUIERDA PLANTILLA CONECTADA AL ARDUINO. DERECHA PLANTILLA SOBRE EL VIOLÍN

Microcontrolador

De acuerdo a la aplicación desarrollada, la mejor alternativa tanto en precio como en compatibilidad, es el microcontrolador Arduino. Una de sus principales ventajas es que ofrece compatibilidad con programas de software libre como Processing2.0, entre otros.

Dentro de la familia Arduino se consideraron diferentes opciones, en primer lugar el Arduino Mega por la cantidad de entradas/salidas que tiene, pero el tamaño es bastante grande para el diseño pensado de la interfaz, además de tener una capacidad que excede a nuestras necesidades. Por lo tanto, se optó por el desarrollo con el Arduino Duemilanove ATmega328 que contiene 14 pines con entradas/salidas digitales y 6 entradas analógicas con conexión USB y funciona a 5V. Cada pin digital se puede emplear como entrada o salida dependiendo de su uso, en este caso, se usa como pin de salida y operan a 5V y 40mA. Por tal motivo, se tiene que disponer de resistencia en los LEDs, además de proporcionar la corriente necesaria para alimentarlos.

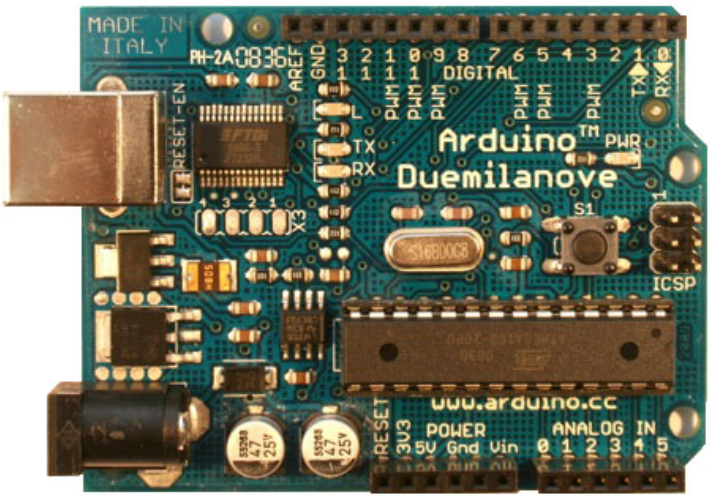

FIG. 25. PLACA ARDUINO DUEMILANOVE

Los pines 0 (Rx) y 1(Tx) no se pueden utilizar ya que por esos pines se da la recepción/transmisión de datos con el puerto serial de la computadora. También proporciona un puerto virtual en la computadora que incluye un

monitor de puerto serie que permite enviar y recibir datos textuales a la placa Arduino.

Si bien sólo quedan libres 12 pines digitales para encender o apagar los LEDs, se usaron los pines analógicos ya que proporcionan una resolución de 10 bits (1024 valores) y con esto se completaron los 16 pines de salida necesarios para manipular los LEDs.

El Arduino es programado desde su propio software gratuito, mediante la conexión USB, por lo que no es necesario usar hardware externo. Para la programación se dan de alta las variables int asignándoles el nombre de la nota que corresponde y tomando en cuenta lo que se programó en Processing como más adelante se verá. Se asigna un pin en orden, por ejemplo el pin 14 se le asigna la nota Si5, al pin 13 la nota Do6 y así sucesivamente hasta completar las 16 notas.

```
int Si5 = 14; // the pin that the LED is attached to 21
int Do6 = 13; // the pin that the LED is attached to 22
```

Después, dentro de la función void setup() se da de alta el puerto serial y se le asigna la velocidad en baudios con que se transmitirán los datos. La velocidad debe de ser la misma que se selecciona en el puerto virtual de la computadora y la que se utilizará en el código que se genere en Processing. Después de inicializado el puerto serial, se dan de alta los pines como salidas mediante pinMode(), en el cual se especifica qué pin se usará y si este pin será salida o entrada.

```
pinMode(Fa6, OUTPUT);
pinMode(Sol6, OUTPUT);
```

Después en la función void loop() -que se ejecuta continuamente- se lee el puerto serial y se dan de alta las condicionales de cada pin, por ejemplo

```
 if (incomingByte == 11) {
    digitalWrite(Fa6, HIGH);
  }
  else
```

```
{
  digitalWrite(Fa6, LOW);
}
```

que es si (If) en el puerto serial se recibe el numero 11, manda la salida a nivel alto; en otras palabras se enciende el LED del pin designado por las letras Fa6 que en este caso es el pin 18; si no (else) recibes ese dato mándalo a estado bajo o apagado.

Y así sucesivamente con cada uno de los pines que se utilizarán, sean pines análogos o digitales. Una vez hechas las pruebas con el Arduino Duemilanove ATmega328 y una vez acabada la plantilla electrónica se realizaron pruebas para ver el funcionamiento con un Arduino de menor tamaño ya que este Arduino a pesar de ser menor que el Arduino Mega sigue siendo de un tamaño mayor al que se tuvo en mente. Con esto y teniendo el programa ya funcionando, se buscó la opción de usar el Arduino Nano que básicamente tenía las mismas funciones que el Arduino Duemilanove ATmega328 pero, con dimensiones menores.

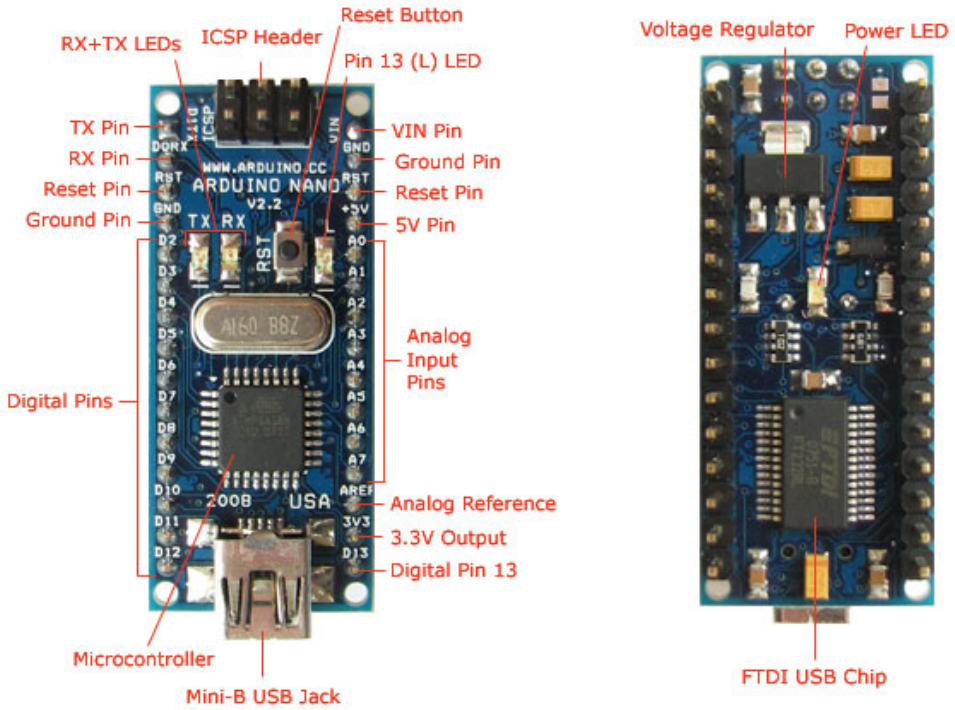

FIG. 26 PLACA ARDUINO NANO

El Arduino Nano está basado en el Atmega328, no posee conector para alimentación externa y funciona con un cable USB mini-B en lugar del cable estándar. Es producido por Gravitech. Opera a 5V y 40mA y tiene 14 pines digitales y 8 entradas analógicas, y mide 18.5mm x 43.2mm por lo que fue perfecto para nuestro diseño. La comunicación serial se realiza de la misma forma que con el Arduino Duemilanove por lo que, el mismo código que se utiliza para este último se puede emplear en Arduino Nano y sin necesidad de hacerle alguna modificación.

FIG. 27 MONTAJE DE PLANTILLA Y ARDUINO SOBRE EL VIOLÍN

Interfaz gráfica

Aspectos Visuales

De acuerdo al diseño conceptual de la interfaz se tienen consideradas 5 diferentes pantallas que contendrían toda la información necesaria para el videojuego.

Primero se debe tomar en cuenta el diseño de los videojuegos casuales y sobre ese concepto realizar el diseño. Se toman solo figuras y formas sencillas. Se realizan varios bosquejos de lo que podría tener cada pantalla así como de la función que cada control tendrá, tomando en cuenta las recomendaciones dadas tanto en la literatura como por los profesores.

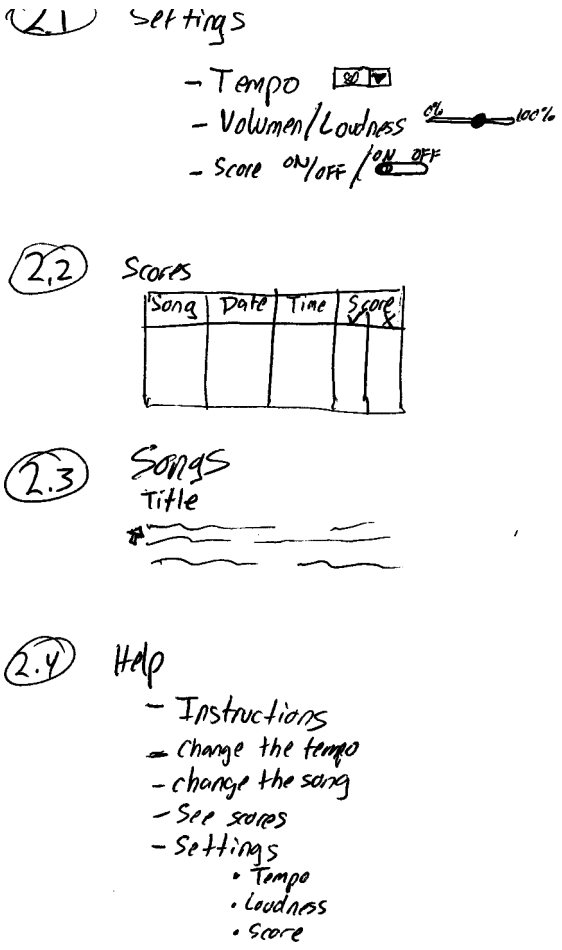

FIG. 28. BORRADOR DE POSIBLES PANTALLAS

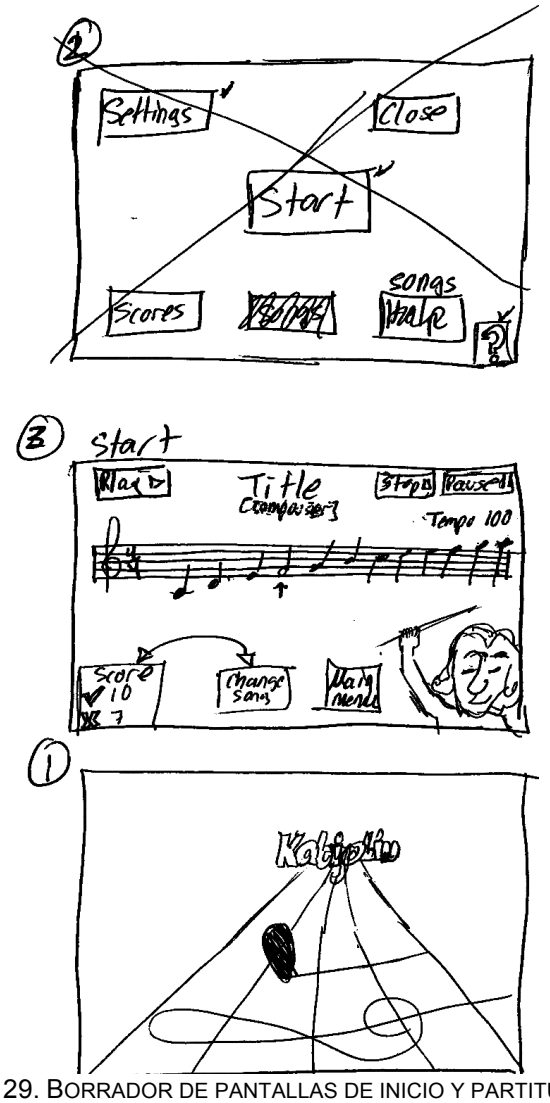

FIG. 29. BORRADOR DE PANTALLAS DE INICIO Y PARTITURA

Por ejemplo, se consideró para la pantalla de inicio un diseño que se modificó dado que para una ventana de inicio contenía demasiada información y no daba una presentación formal al videojuego. Posteriormente en la misma hoja se hizo otro diseño tomando en cuenta la presentación del videojuego.

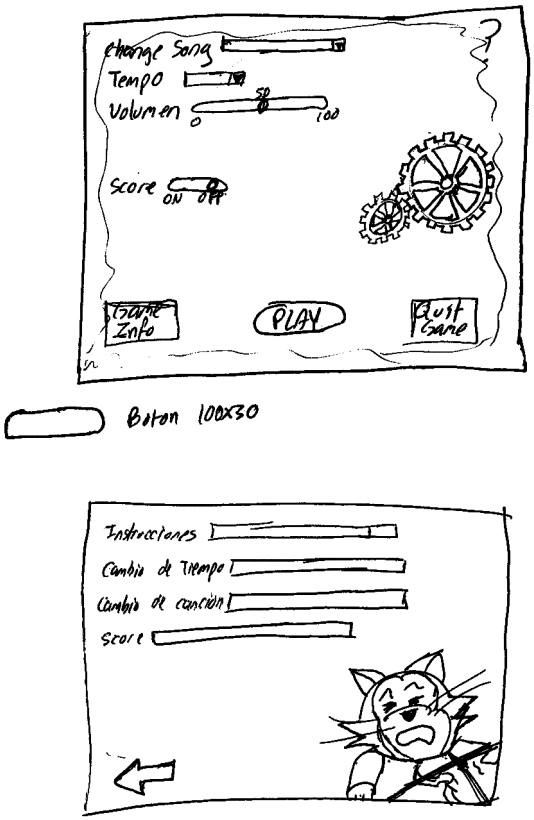

FIG. 30. BORRADOR DE PANTALLA DE OPCIONES Y AYUDA

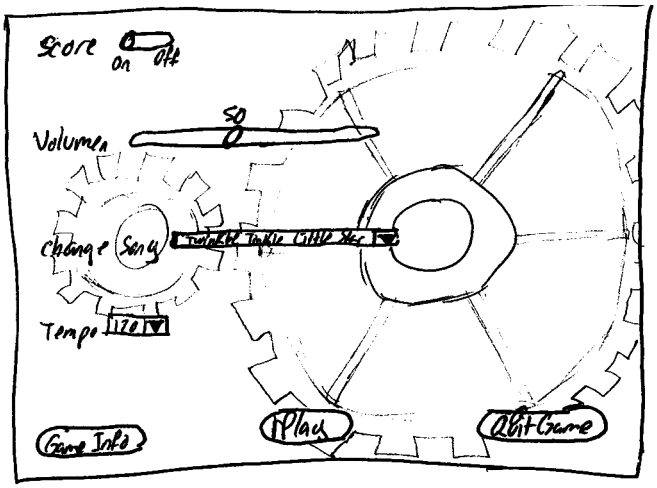

FIG. 31. BORRADOR DE PANTALLA DE OPCIONES

La pantalla de Configuración o Setting Screen en un inicio se había considerado con dos engranes en movimiento lo cual quitaba visibilidad a la parte principal de los controles por lo que es mejor opción colocar una imagen fija. Después de hacer una reubicación de los controles se llega a la pantalla deseada. Así mismo, se agrupan los controles en cajas por la función que

realizan dentro del videojuego para tener una mejor percepción y ubicación, por lo que el diseño que se muestra en la figura anterior cambió con el diseño final.

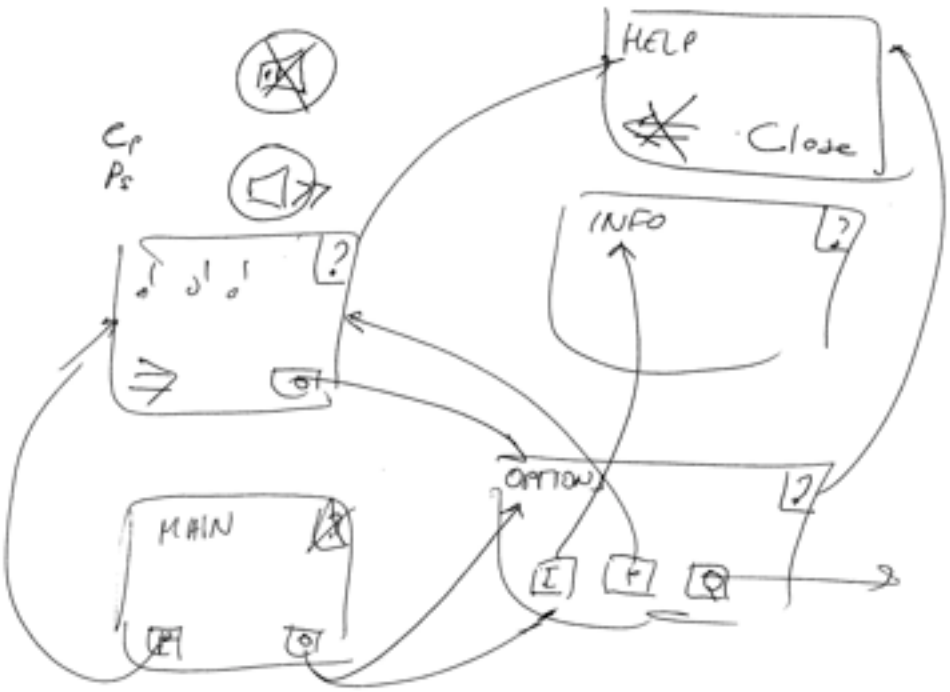

FIG. 32. DIAGRAMA DE FLUJO DE ACCESO A LAS DIFERENTES PANTALLAS

En el esquema anterior se muestra el flujo que deberían de tener las pantallas dependiendo de qué control se activaría.

En cada pantalla con excepción de la pantalla de Inicio se considera dejar un botón de ayuda o Help para ayuda o dudas que se tuvieran de la funcionalidad del videojuego para el ejecutante. Se considera dejar este botón en el mismo lugar en todas las pantallas para que sea fácil su ubicación.

Pantallas

Para la creación de los botones y controladores en las pantallas de configuración (Settings Screen), de ayuda (Help Screen) , de información (Info Screen) y de la partitura (Song Screen), se utiliza la librería G4P, la cual provee una vasta colección de controles 2D GUI para Processing utilizando la siguiente estructura.

Primero se da de alta para cada pantalla la función void *pantalla*_screen() – *pantalla* = nombre de la pantalla, ej., *pantalla*_screen() = song_screen()- que hace la función de un setup interno para cada una y que será llamado desde el setup() de la pantalla principal, por lo tanto contiene sub funciones void como void *pantalla*_screen_visible(true) en la cual tiene valor booleano para hacer visibles o no los botones mediante la clase *botón*.setVisible(visible) la cual mediante la variable *visible* asumirá el valor booleano que se designó en el void *pantalla*_screen(). Ejemplo:

- Void *pantalla*_screen();
    - *pantalla*_screen_visible(true);
    - Se programan los demás elementos de dibujo que lleve la pantalla, ej. fondos de pantalla, animaciones, colores, etc.

Después se crea una función void make_*pantalla*_screen() el cual carga las funciones donde se crean e inicializan los controles y botones y se llama una vez desde el setup(). Los botones son generados con imágenes hechas y editadas en PixlEditor en formato png para evitar que se vean los fondos blancos. Una vez creados los archivos se guardan en la misma carpeta de datos en el que se encuentra el programa en Processing para poder ser llamados mediante la función en la cual se dieron de alta todos los botones, sliders, cuadros con listas desplegables, etc. y se llaman en la función void setup() principal del programa.

Para los controles se utiliza un arreglo de variables String[] y se dan de alta las clases propias de la librería G4P para cada tipo de control. La asignación de las tareas que cada control debe de cumplir se asigna en la pantalla principal del sketch.

Después de la función make_*pantalla*_screen() se crea la función void *pantalla*_screen_visible(boolean visible). Esta función puede correrse en cualquier otra parte del programa (después de haber creado los controles).

Para hacer visibles/invisibles los controles en la pantalla en la cual se ejecutarán los botones, sliders o listas que se quieran ver en la pantalla se

asigna la variable booleana “visible”, por ejemplo, sdrVolumen.setVisible(visible). Esta función se llama dentro de la función void *pantalla*_screen(true) como se puso anteriormente con la variable booleana true para hacer los botones visibles cuando la función *pantalla*_screen() sea llamada en la función void setup() principal del programa, mientras tanto si no es llamada se mantendrá como *pantalla*_screen_visible(false) para ocultar los botones del resto de las pantallas y evitar que se encimen los botones de cada pantalla.

Controles de barra deslizadora (Sliders).

Los slider o barras son objetos dentro de una GUI (Graphical User Interface) con la cual se selecciona un valor moviendo un indicador dentro de cierto rango de valores en la barra. Ocupamos la clase de GCustom Slider de la librería G4P para crear sliders en la pantalla de Configuración cómo se explica a continuación de forma genérica.

sdr*nombredeslider* = new GCustomSlider(this, 200, y+160, 40, 50, null); // que nos da las coordenadas, tamaño y ubicación dentro de la pantalla del control.

sdr*nombredeslider*.setShowDecor(false, true, true, true); // da formato al control mediante variables booleanas (boolean opacidad, boolean divisiones, boolean valor, boolean limites).

sdr*nombredeslider*.setLimits(1, 1, 2); // da el valor inicial y el rango de valores que contendrá el control (int valor inicial, int rango menor, int rango mayor).

sdr*nombredeslider*.setTickLabels(tickLabels); // muestra el texto del valor máximo y mínimo del rango de valores.

lbl*nombredeslider*. = new GLabel(this, 100, y+160, 80, 50, "Nombre del slider"); // con este método se dan la ubicación y tamaño del texto con el que se llamará y aparecerá el Slider.

lbl*nombredeslider*.setTextAlign(GAlign.RIGHT, null); // Da la alineación horizontal y vertical del texto)

lbl*nombredeslider*.setTextBold(); // Hace todos los caracteres en negritas.

Listas desplegables.

Las listas desplegables son igual que los sliders elementos de una GUI (Graphical User Interface) la cual permite escoger un valor dentro de una lista de opciones. Cuando la lista está activa muestra todas las opciones de valores, y cuando está inactiva muestra solamente el valor inicial asignado , cuando un valor es seleccionado éste se muestra en el recuadro para saber que ese valor fue seleccionado. Dentro del programa se usan listas en diferentes pantallas usando la clase DropList de la librería G4P como a continuación se presenta de forma general .

lbl*nombredelista* = new GLabel(this, 100, y+300, 80, 19, “nombre de lista”); // nos da el nombre, las coordenadas y el tamaño que tendrá la lista.

lbl*nombredelista*.setTextBold(); // pone en negritas el nombre de la lista

String[] ítems = new String { “lo que aparecerá en la lista o cuadro por renglón, cada renglón se puede seleccionar y se separa por “,” ” }

dpl*nombredelista* = newDropList(this, 200, y+300, 250, 96, 5); // da el tamaño y coordenadas del cuadro que contiene los elementos de la lista.

dpl*nombredelista*.setItem(ítems, 0); // indica cuál es el renglón que tendrá el valor inicial.

Control mediante botones.

Los botones tienen dos estados los cuales se crearon en PixlEditor , una imagen para el estado encendido y otra imagen para el estado apagado, estos archivos con extensión png se guardan en la misma carpeta en donde se guarda el sketch y archivo de processing para que puedan ser llamados desde el setup() y cargados en la interfaz. Igualmente que los controles anteriores estos son creados con la librería G4P.

Files = new String[]{ “nombre del archivo de la imagen contenido en la misma carpeta que el sketchbook” }; //llama al archivo de la imagen que se quiere usar como imagen del botón.

btnBack2 = new GImageButton(this, 20, 40, files); da las coordenadas en donde será colocado el botón.

Pantalla de la partitura (SongScreen)

Para el caso particular de la pantalla Canción (Song Screen), se diseña de forma que esta pantalla sea lo más limpia posible, es decir que no contenga mas que sólo los elementos necesarios para poder manipular sólo los controles básicos como *play, stop*, pausa, volumen, y un botón de regreso a la pantalla principal. Esto con el fin de evitar distractores cuando se esté leyendo la partitura y así lograr centrar la vista del jugador en la nota que se debe de ir tocando en lugar de fijar la vista en animaciones de segundo plano.

Para poder hacer un cambio de partitura en la pantalla utilizamos la estructura switch dentro de la función void *pantalla*_screen() para obtener diferentes niveles para evaluar los datos contenidos en la estructura void de cada partitura, tanto para *partitura*partitura() como para *partitura*musica()

```
switch(cancion){
case 0:
   estrellitapartitura();
   estrellitamusica();
   break;
case 1:
   rowpartitura();
   rowmusica();
   break;
case 2:
   windpartitura();
   windmusica();
   break;
```

```
 }
```

Pantalla de Información

Para esta pantalla se sigue la misma estructura que el resto de las pantallas con excepción de que se genera la función void drawType(float x) para poder escribir texto con formato, ya que en esta pantalla no era necesario la interacción más que con el botón de regreso. Esta función se llama desde la función void info_screen(). El botón de regreso se genera igual que el resto de los botones de las demás pantallas dentro de la función make_info_screen().

```
void drawType(float x) {
  fill(255);
  text("Este programa se realizo", x, 100);
  text("mediante Processing2.0, Arduino1.0.4", x, 150);
  text("panit.Net y Pixlr Editor.", x, 200);
  text("Mexico DF, 2013", x, 250);
}
```

Pantalla de Inicio

En esta pantalla se consideró hacer una presentación breve del videojuego y colocar los botones que llevan a las diferentes pantallas principales. Se sigue la misma estructura que con el resto de las pantallas. El *framerate* que es el encargado de la velocidad a la que se mueven las imágenes fue de 40, se cargaron dos imágenes *space* y *space1* las cuales se van desplazando de derecha a izquierda una seguida de la otra.

```
void inicio_screen(){
    inicio_screen_visible(true);
    frameRate(40);
     image(space,bx1,by);
     image(space2,bx2,by);
     bx1 -= accel;
     bx2 -= accel;
     if (bx1+bw < 0) {
         bx1 = bx2 + bw;
```

```
    }
    if (bx2+bw< 0) {
        bx2 = bx1 + bw;
    }
}
```

Pantalla Principal

Finalmente se unen o controlan todos los datos desde una pantalla principal en la que se dan de alta todas las variables y se inicializan las librerías que se utilizan para todas las pantallas.

En la función void setup() se programan todos los elementos que se cargarán en el programa una sola vez. Aquí se da de alta el puerto serial asignando la velocidad en baudios (numero de símbolos que pueden ser de uno o mas bits por segundo en una transmisión digital) con la que se comunica el puerto con el microcontrolador.

```
println(Serial.list());
String Puerto = Serial.list()[1];
mipuerto = new Serial(this, Puerto, 9600);
```

También se genera la dimensión que tiene la pantalla que en este caso es de 1024 x 768 pixeles.

Después se llaman a todas la funciones con las que se crearon los controles en cada pantalla para cargar cada control en el programa una sola vez.

```
make_inicio_screen();
make_settings_screen();
make_song_screen();
make_info_screen();
```

Después se agregan las funciones que nos permiten hacer visibles los controles o hacerlos invisibles.

```
inicio_screen_visible(false);
```

```
settings_screen_visible(false);
song_screen_visible(false);
info_screen_visible(false);
help_screen();
```

y se cargan todas las imágenes que se ocuparan en el programa.

Dentro de este setup() también se establece el comando addCallbackListener() que llama a las subfunciones enviadas de la clase score. El tiempo (velocidad) del score es de 180 beats por minuto. Durante es ejecutada la clase score, el tiempo para cada subfunción en la clase score es alcanzada por el método handleCallbacks() y es activada. Este método verifica que el valor de la sub función sea -1 y si es así la subfunción se vuelve a ejecutar y se toca la nota.

```
 score = new SCScore();
 score.addCallbackListener(this);
 score.addCallback(0.5,1);
 score.play(-1);
 redraw();
```

En la función draw() se utiliza de nueva cuenta la estructura switch para llamar cada pantalla con sus botones correspondientes; a este switch se le asigna la variable *mode* que se asignará al valor de cada control como más adelante se explica. Se usan 5 casos para las 5 diferentes pantallas por ejemplo en el caso 1 se tiene:

```
void draw() {  // this is run repeatedly.
 switch(mode){
   case 1:
     inicio_screen_visible(true);
     settings_screen_visible(false);
     song_screen_visible(false);
     info_screen_visible(false);
     //help_screen_visible(false);
     inicio_screen();
     break;
```

en donde se llama a la pantalla inicio_screen_visible(true) lo que hace los botones contenidos en esta función visibles, mientras que el resto de las funciones aparecen con false lo que no los hace visibles.

Se utiliza nuevamente la estructura switch para seleccionar entre volumen o silencio asignando a switch la variable *mute*.

```
switch(mute){
   case 6:
      //int j = Integer.parseInt(sdrVolumen.getValueS());
      sdrVolumen.setValue(v);
      break;
   case 7:
      //int j = Integer.parseInt(sdrVolumen.getValueS());
      sdrVolumen.getValueS();
      sdrVolumen.setValue(h);
      break;
    }
```

donde v = float v= 0.0 y h = float h= 50.0 y que se asignarán al control del botón de silencio o mute.

Para los botones se utiliza una función void handleButtonEvents(GImageButton button, GEvent event) en la cual se asignan los estados que debe tomar cada botón. Por ejemplo, para la pantalla de inicio se considera un botón llamado *Play* que nos va a llevar directo a la pantalla de la canción y partitura o *Song Screen* por lo que su asignación tomando la estructura switch(mode) es

```
if(button == btnPlay1){
   mode = 3;
 }
```

Para el caso del botón Play de la pantalla de Song Screen se utiliza lo siguiente

```
if(button == btnPlay3){
   loop();
   score.play(-1);
```

}

que contiene un loop() que corre continuamente la función draw() por lo que inician nuevamente los procesos que generan tanto la música como la nota que va en turno. Aunado a esto se utiliza score.play(-1) que se programa con addCallback por lo que después de haber llamado al botón *Stop* la sub función es llamada al ser presionado el botón *Play* y comienza la música desde la primer nota al igual que la nota en turno inicia desde el principio de la partitura.

Para el caso del botón Stop de la pantalla Song Screen es necesario detener el proceso tanto de la música como de la nota que va en turno dibujándose, por lo que se ocupa

```
if(button == btnStop){
   noLoop();
   score.stop();
   a = 0;
   xLoc3 = 0;
   yLoc2 = -150;
```

en la que se utiliza un noLoop() para detener los procesos generados para la música y se asigna a la variable a de la partitura y música el valor de 0 para que reinicie desde ahí cuando se accione el botón Play, también se reasignan los valores de xLoc y yLoc a los valores de inicio.

Para los sliders se utiliza la función handleSliderEvents(GValueControl slider, GEvent event) propia de la librería G4P en la que se le asignan los valores deseados a cada slider, por ejemplo

```
if (slider == sdrTempo);
}
```

donde sdrTempo está definido en cada pantalla de la *partitura* dentro de la función *partitura*musica() mediante

```
int k = Integer.parseInt(sdrTempo.getValueS());
```

donde k se define en el frameRate como

```
  frameRate(k/60);
```

Para las listas desplegables se utiliza la función handleDropListEvents(GDropList, GEvent event) en donde se designa la lista y se selecciona el valor que se desea

```
public void handleDropListEvents(GDropList list, GEvent event){
if(list == dplSong);
  dplSong.getSelectedIndex();
}
```

Para el caso del botón de silencio o *mute* se utiliza la estructura que anteriormente se había defino de switch(mute) dado que el botón tomará dos estados (encendido y apagado) por lo que varía un poco en la estructura de los otros botones, por ejemplo

```
public void handleToggleButtonEvents(GImageToggleButton button, GEvent event){
  if(button == btnSpeaker){
    if(btnSpeaker.stateValue() == 0){
    mute = 7;}
  else
    mute = 6;
  }
```

en donde si el botón toma el estado 0 tomará el nivel de 7 en el que el volumen es de 50.0 que es la mitad del volumen total, y si mute toma el nivel de 6 tomará el valor v que es igual a 0 del volumen general.

Partitura y sonido

Como primer paso se trabaja en diferentes formas de poder crear una partitura que se sincroniza con la parte audible y que a su vez genere los audios. Para el sonido se busca un sonido lo más parecido al piano sin pretender agregar mas instrumentos

Se hizo una búsqueda de diferentes librerías contenidas en Processing y se encuentra que la más conveniente para desarrollar este programa es la librería SoundCipher desarrollada especialmente para música en Processing y combinarlo con la parte visual ya sea para animaciones, juegos o interfaces de control.

Utiliza el sintetizador JavaSound para reproducir archivos de audio y comunicación MIDI; permite también asignar tiempos y ritmos, así como código para generar partituras complejas. También se puede integrar a la librería Minim para audios más sofisticados, y también se puede integrar a la librería OscP5 para comunicación vía open sound control.

Esta Librería es libre y opensource bajo la licencia de GNU GPL, tiene comandos o códigos preestablecidos que sincronizan música y visuales.

Se utiliza la clase playNote dentro de la estructura que crea notas MIDI sencillas que suenan inmediatamente y que se utilizaron los parámetros básicos que maneja esta clase que son:

Pitch: La nota MIDI la cual será tocada (0 – 127)
Dynamic: el volumen, velocidad MIDI de la nota (0-127)
Duration: El tiempo que durará sonando cada nota en beats

Para crear todas las notas de la partitura y que estas fueran sonando una a una es necesario un array de floats de todas las notas de la partitura –*partitura* = nombre de la partitura, ej., *partitura*nota = estrellitanota- tanto para el pitch como para las duraciones.

```
float[] partituranota = {69, 69, 76, 76, 78, 78, 76, 74, 74, 73, 73, 71, 71, 69,
                         76, 76, 74, 74, 73, 73, 71, 76, 76, 74, 74, 73, 73, 71,
                         69, 69, 76, 76, 78, 78, 76, 74, 74, 73, 73, 71, 71, 69,
                         69, 69, 76, 76, 78, 78, 76, 74, 74, 73, 73, 71, 71, 69,
                         76, 76, 74, 74, 73, 73, 71, 76, 76, 74, 74, 73, 73, 71,
                         69, 69, 76, 76, 78, 78, 76, 74, 74, 73, 73, 71, 71, 69};
float[] partituradynamics = new float[partituranota.length];
float[] partituradur = {1.0,1.0,1.0,1.0,1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
```

```
                1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
                1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
                1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
                1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
                1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0};
if (a >= partitura nota.length-1){
   a = 0;
   }
```

y se irá recorriendo el array de nota en nota

```
part1.playNote(partitura nota[a], j, partitura dur[a]);
  a += 1;
```

La variable j se asigna para controlar el slider del volumen pero posteriormente hablaremos e ella.

El método setup() carga todo una sola vez en el programa y la función draw() carga cada comando y repite las líneas de código cada determinado tiempo asignado por un *framerate* que es el número de cuadros para ser mostrados por segundo, el valor por default en processing para el *framerate* es de 60 cuadros (*frames*) por segundo.

```
  frameRate(k/60);
```

Se considera la variable k que está asignada al slider del tempo, el cuál asigna un valor para cada k tal que k/60[s] = tempo de la partitura

Para cada partitura se genera una función que contiene la clase playNote, así como variables independientes entre partituras.

Se utiliza la función ellipse() para crear las notas con los siguientes parámetros:

A: float: cordenadas x
B: float: cordenadas y
C: float: ancho de la elipse
D: Alto de la elipse

En esta parte se contempla la variable xLoc3 como el parámetro A el cual se refiere a la posición de la nota dentro de la pantalla que se configuró de 1024 x 768 pixeles por lo que se hizo un ajuste sumando 75 pixeles para que el dibujo de la nota comenzara en la posición x+75 y poder dibujar cada nota sin rebasar el ancho de la pantalla y conservar los márgenes establecidos.

También se agregó la variable lineDistY la cual nos entrega la distancia de las notas dentro del eje Y con un ajuste en la pantalla de -288 para poder centrar la partitura, por lo que ocupamos las siguientes funciones

```
if (75+xLoc3 >= width){
   xLoc3 = 0;
   yLoc2 = yLoc2 + lineDistY;
   }
 ellipse(xLoc3+75, height-3*partitura nota[a]+yLoc2-288, 10, 10);
 xLoc3 += 20.0;
```

con una separación en x de 20 pixeles entre notas.

Se crea una función partitura_handleCallbacks(int callbackID) la cual crea una sub función que cuando es llamada se ejecuta también la función principal. Dentro de esta subfunción se crea un beat que va a depender de la nota que vaya tocando

```
beat += partitura nota[n];
```

y dentro de la clase score de la librería SoundCipher se agrega esta subfunción para que cada nota sea llamada a la vez dentro de esta clase para poder asignarla a los botones de *stop, play* y pausa.

```
score.addCallback(windnota[n], 1);
score.play();
```

Para la parte para enviar los datos al puerto serial de la computadora es necesaria una matriz o array que incluye todas las notas MIDI y asignar un numero –que representa el primer dígito la cuerda del violín (Sol-4, Re-3, La-2,

Mi-1) y el segundo dígito el dedo que se debe de utilizar (1, 2, 3, 4) Simulando los códigos escritos en las partituras de algunos métodos de violín para niños.

```
int[] pitch_to_LED = {
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 0, pitch 0 a 11
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 1, pitch 12 a 23
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 2, pitch 24 a 35
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 3, pitch 36 a 47
   0, 0, 0, 0, 0, 0, 0, 40, 0, 41, 0, 42, // octava 4, pitch 48 a 59
   43, 0, 44, 0, 31, 0, 32, 33, 0, 34, 0, 21, // octava 5, pitch 60 a 71
   0, 22, 23, 0, 24, 0, 11, 0, 12, 13, 0, 14, // octava 6, pitch 72 a 83
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 7, pitch 84 a 95
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 8, pitch 96 a 107
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 9, pitch 108 a 119
   0, 0, 0, 0, 0, 0, 0 //, 0, 0, 0, 0, 0, // octava 10, pitch 120 a 127
 };
```

Con este array, cuando en la clase playNote se toque una nota MIDI por la posición de esa nota, el array convertirá ese valor de la nota MIDI en un valor numérico que enviará al puerto serial de la computadora. Esto se hace asignando el array de pitches a un nuevo array de enteros (int) para que la función mipuerto.write pueda leerlo como enteros y escribir el dato en el serial.

Para la parte de la partitura se dibujan las funciones dentro de un void llamado *nombredelacanción*partitura() que aparece inmediatamente cuando se selecciona la partitura que se quiere interpretar.

Al igual que para tocar cada nota, se asigna una variable n la cual va a tomar cada valor del array pitches.

```
if (n >= partituranota.length-1){
   n = 0;
   }
```

y se asigna otro valor de xLoc para la separación sobre el eje x de las notas.

```
if (75+xLoc >= width){
```

```
  xLoc = 0;
  }
  xLoc2 = 0;
```

Primero se diferenciaron las notas con plica hacia arriba para las notas de Do4 a Sol5 y la plica hacia abajo para las notas de La5 a Si6. Esto se hace mediante un ciclo for e if para elegir si la nota es menor que la nota MIDI 70 la plica va hacia arriba, y si la nota es mayor que la nota MIDI 70 la plica va hacia abajo.

```
for(n = 0; n< partituranota.length; n++){
if(75+n*20<width){
  if(partituranota[n]>70){
    line(xLoc-3.5+75, height-3*partituranota[n]-438, xLoc-3.5+75, 40+(height-3*partituranota[n]-438));
    }
  if(partituranota[n]<70){
    line(xLoc+4+75, height-3*partituranota[n]-438, xLoc+4+75, (height-3*partituranota[n]-438)-40);
    }
```

Para el caso de distinguir entre notas negras de un tiempo o blancas de 2 tiempos, se sigue el mismo procedimiento que con las plicas, sólo que ahora se toma en consideración para el ciclo if si la duración de la nota en el array *partitura* es mayor o menor de 1 tiempo mediante la función fill(rgb) ; donde rgb= la escala de colores siendo de 0 al máximo 255 el valor más alto.

Instrucciones

El videojuego funciona de la siguiente forma:

se coloca la plantilla de LEDs en el violín de 4/4 y se ajustan los velcros para asegurar la posición. Igualmente, se fija con velcro el controlador y se conectan los cables de la plantilla de LEDs al Arduino. Se conecta el cable USB del Arduino a la computadora. Se corre el programa. La primer pantalla que se presenta contiene 4 botones con los cuales se puede ir a tres diferentes

pantallas o salir del juego. Si se oprime el botón de opciones, éste llevará a la pantalla de configuración donde se puede escoger la partitura que se quiere tocar mediante una lista desplegable. Con un slider, se elige el volumen de la música y con el otro slider se escoge el tiempo al que se desea sea tocado. Un botón del lado superior izquierdo llamado "*back*" nos regresa a la pantalla principal.

Si en la pantalla principal se presiona el botón de Info, nos llevará a la pantalla que contiene la información del sistema sobre qué programas se emplearon y que versiones, así como las diferentes entidades que colaboraron. Igualmente tiene un botón de "*back*" para regresarnos a la pantalla principal.

Si en la pantalla principal se presiona el botón "play" éste nos llevará a la pantalla de la partitura en donde iniciará la música indicando la nota que se debe de ir tocando. En esta pantalla se tiene un botón de pausa el cual al ser presionado detiene la música y el avance de la nota en turno, si se quiere continuar se tiene que presionar de nueva cuenta el botón que se encuentra junto de "play".

También se encuentra un botón de "stop" el cual detiene por completo la música y la nota en turno y regresa al inicio todos los parámetros; si se quiere iniciar de nuevo la canción se tiene que presionar nuevamente el botón de "play". Se tiene también un botón de "mute" o silencio con el cual se puede silenciar el sonido de la música aunque la nota en turno siga avanzando; si se vuelve a oprimir volverá a escucharse la música.

Todas las pantallas tienen en la esquina superior derecha una pestaña con la leyenda "help", la cual despliega una pantalla semitransparente sobre la pantalla en que se encuentra el usuario y aparecen las instrucciones generales del videojuego, así como la función de los diferentes controles en forma de lista desplegable. Al igual que en las pantallas anteriores, se tiene el botón de "back" para regresar a la pantalla principal.

Por último está el botón de "Quit Game", el cual al ser presionado, se cierra el juego.

Capitulo 4. Evaluación y análisis.

En este capítulo se hace una descripción de los resultados que se obtuvieron en cada uno de los escenarios del videojuego, así como de la interfaz electrónica.

En cuanto a la parte visual, tenemos 5 pantallas las cuales fueron diseñadas pensando en los principios de usabilidad y de interacción máquina-humano; se consiguió hacer que estas fueran visualmente simples y que contuvieran lo importante de cada sección.

En la pantalla de inicio (fig. 33) se observan colores que resaltan del fondo y formas que se refieren a la parte musical del juego. Estas se van desplazando a lo largo de la pantalla de derecha a izquierda, lo que genera un movimiento simple y evita tener sólo un dibujo fijo plasmado en un fondo.

Se observan también 4 botones, los cuales, 3 están agrupados hacia la izquierda y uno está hacia la derecha; esto se hizo con el fin de que el jugador pueda identificar o diferenciar los botones que lo introducen al juego del botón que lo saca del él. Dentro del grupo de los 3 botones, se ve un botón en color diferente a los demás, es el botón de *PLAY* –que se encuentra en color azul- con el fin de que el jugador visualice cual será el botón con que inicie el juego; los otros botones son relevantes pero no lo llevarán al objetivo principal, que es jugar.

Se consideró importante, en la pantalla de inicio, los botones de “opciones” e “información del juego”, ya que el de opciones nos lleva a la configuración del juego, en el que se puede cambiar de partitura, tiempos, etc. Y el de información contiene -como su nombre lo indica- información sobre la creación del videojuego.

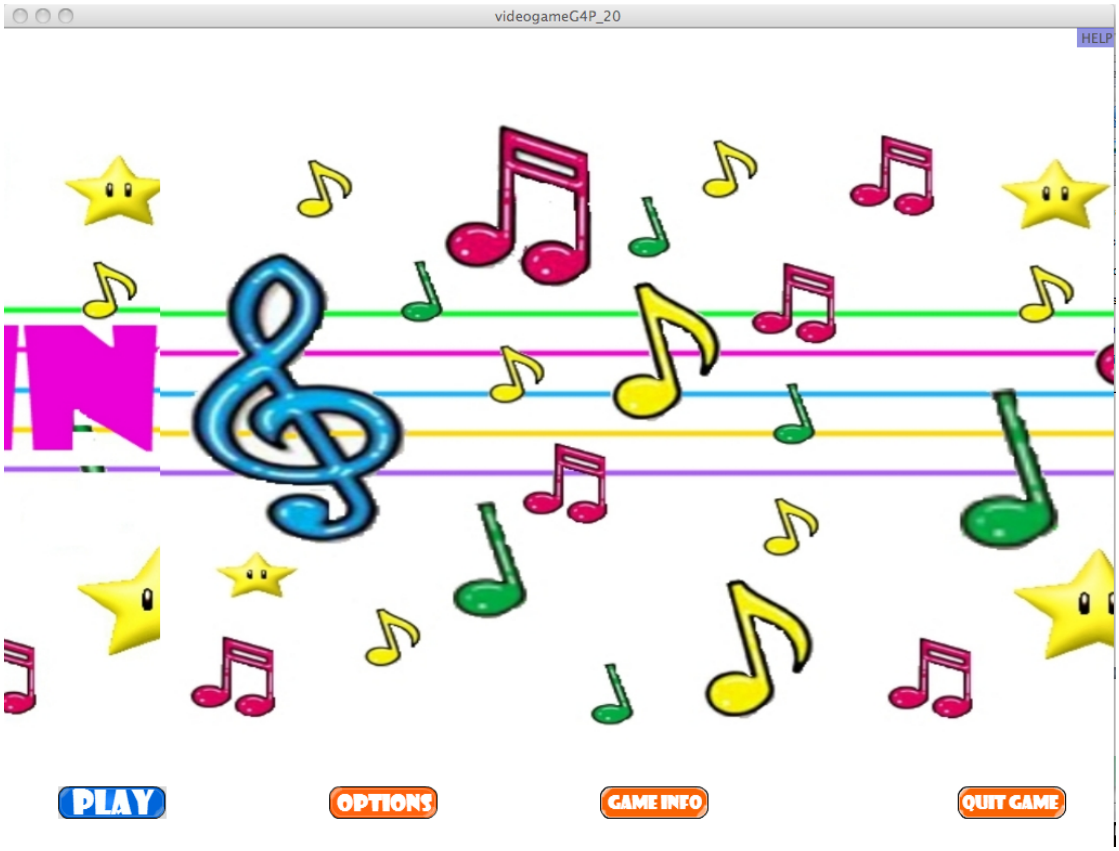

FIG. 33. PANTALLA DE INICIO DEL VIDEOJUEGO

En la pantalla de opciones, como se muestra en la figura 34, podemos encontrar la configuración del juego. En esta pantalla se consideraron los cambios más significativos que se pudieran realizar a la partitura en cuanto a su ejecución. Por ejemplo, en esta pantalla podemos elegir qué partitura se desea tocar de entre las 5 con que cuenta.

También se puede ajustar el volumen de la melodía de la partitura cuando se está jugando; es decir, si se desea escuchar el violín sobre la melodía o viceversa. Se consideró para esto un *slider* o barra deslizadora, ya que se identifica mas con los diseños existentes en las diferentes aplicaciones que existen en el mercado; en donde el rango mas bajo de volumen es el nivel 1 y el más alto es el 100.

Otro botón en forma de barra deslizadora que se considera es el botón del tiempo; con este se puede modificar la velocidad a la que se desea tocar la melodía y la velocidad con la que se va a recorrer o mostrar la nota que se

debe ir tocando; al mismo tiempo también afectará la velocidad con la que se prenden los LEDs en la plantilla electrónica.

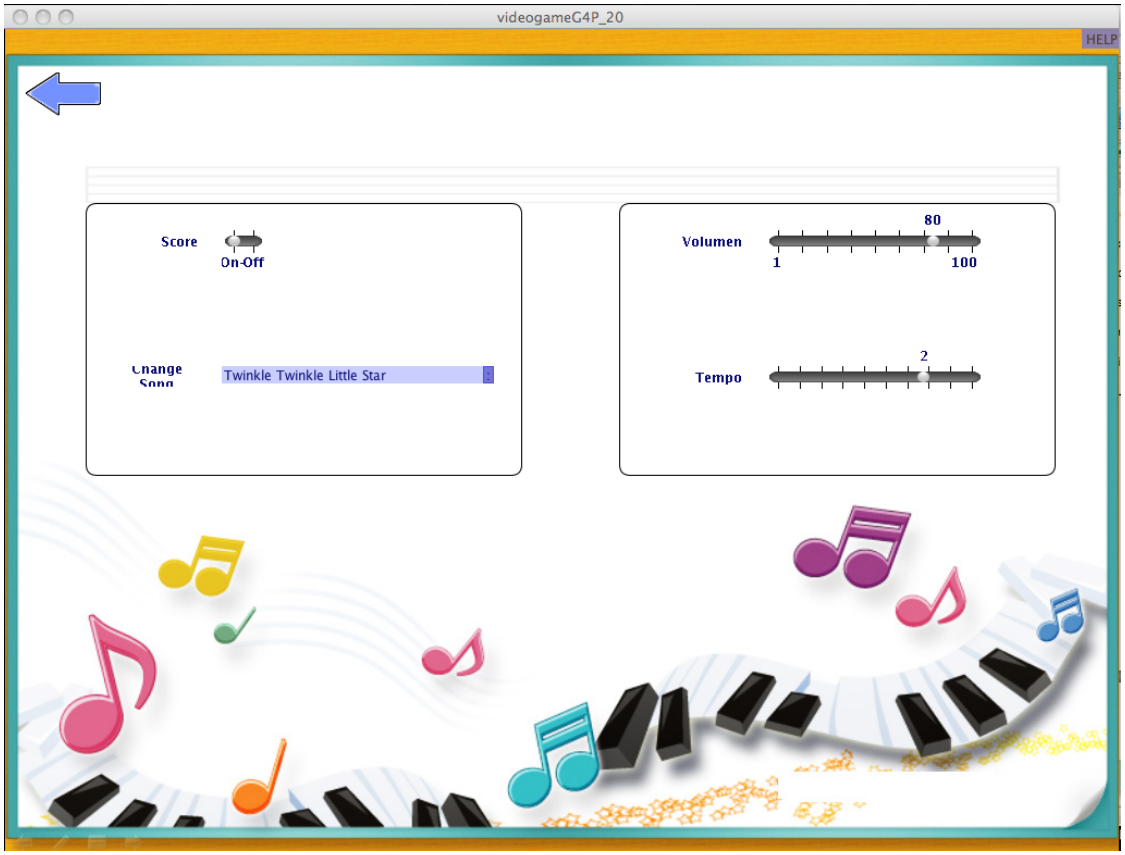

FIG. 34. PANTALLA DE OPCIONES

La pantalla de información del juego (Fig. 35) menciona los diferentes software con los que se realizó tanto la parte de programación, como la parte para el diseño de las pantallas; así como las versiones de cada uno que se utilizó. También se muestra el escudo o logo de cada una de las instituciones involucradas en el desarrollo de este proyecto.

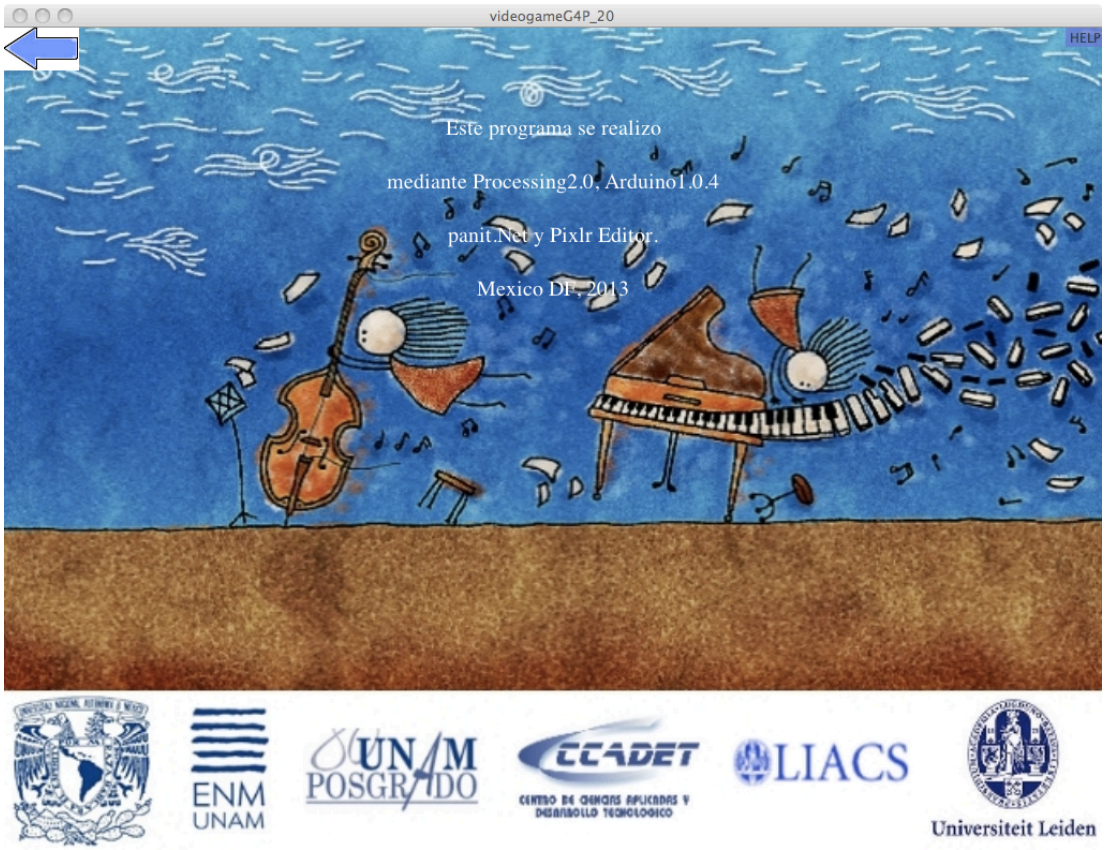

FIG. 35. PANTALLA DE INFORMACIÓN DEL SISTEMA.

La pantalla de ayuda (Fig. 36) es elemental en cada diseño de una aplicación, en ella se muestra el uso básico del juego; debe de estar presente siempre en cada uno de los escenarios de un videojuego, ya sea de forma visual o mediante el accionamiento de un botón en una interfaz electrónica. Se colocó en la esquina superior derecha de cada pantalla ya que es importante que esté colocada siempre en el mismo lugar para que sea fácil su ubicación.

Esta opción de ayuda se despliega sobre la pantalla en que se encuentre el videojuego en forma translúcida, y muestra cómo activar o desactivar las funciones básicas del videojuego como son el volumen, cambiar el tiempo y cambiar la partitura; también puede mostrar unas breves y sencillas instrucciones de cómo se debe de jugar el videojuego.

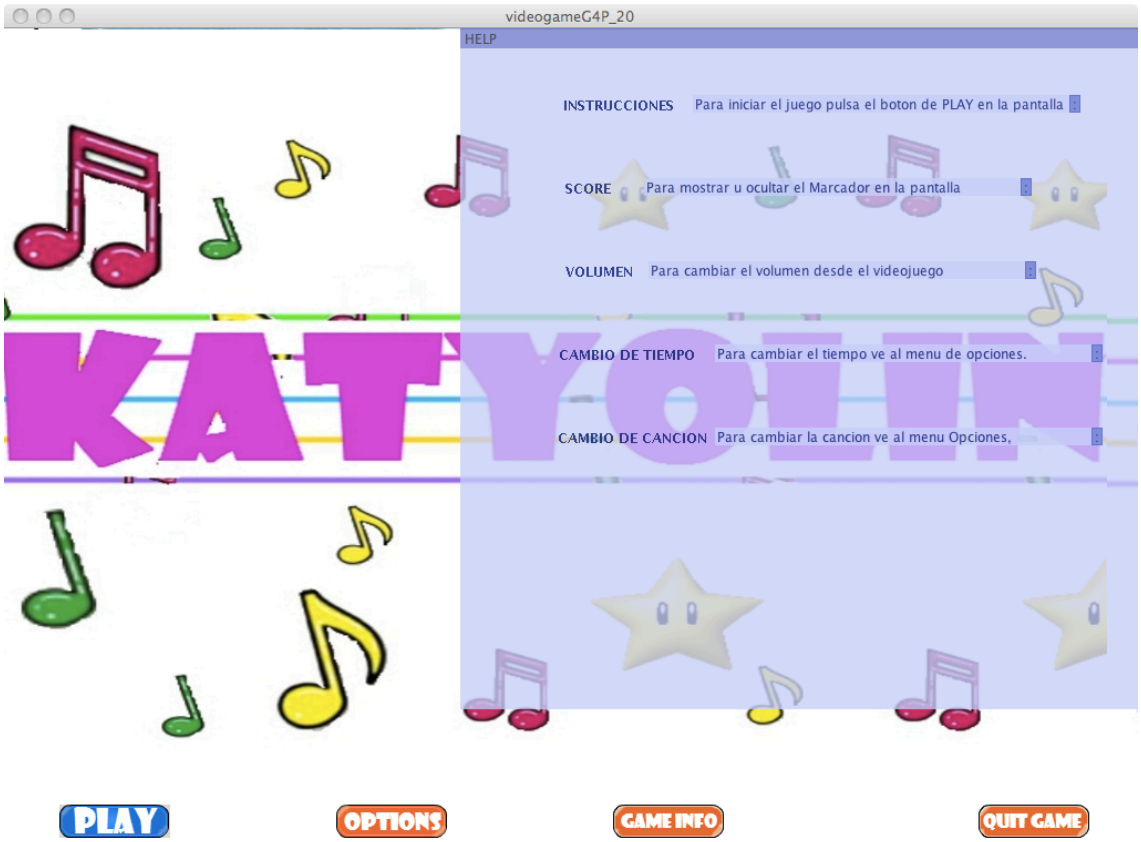

FIG. 36. PANTALLA MOSTRANDO LA AYUDA DEL JUEGO

La pantalla en donde se muestra la partitura se diseñó sin tener muchos elementos visuales que pudieran distraer al jugador, por lo que sólo se centra en la parte de la partitura, como se observa en la figura 37. Se agregaron botones para tener control de la ejecución de la partitura: el botón de *stop, play,* pausa y *mute*. Se eligió ponerlos en la parte superior de la pantalla para tenerlos presentes y localizarlos con facilidad, igualmente el tamaño de los botones y los colores son importantes ya que visualmente los asociamos con acciones cotidianas.

Conforme la nota que tiene que ser tocada va avanzando se va mostrando cada pentagrama, esto para evitar llenar la pantalla de información y que el jugador se confunda entre pentagramas.

Como parte dinámica se muestra una figura con movimiento que va marcando el tiempo en que se debe tocar la melodía y que simula a un director de orquesta.

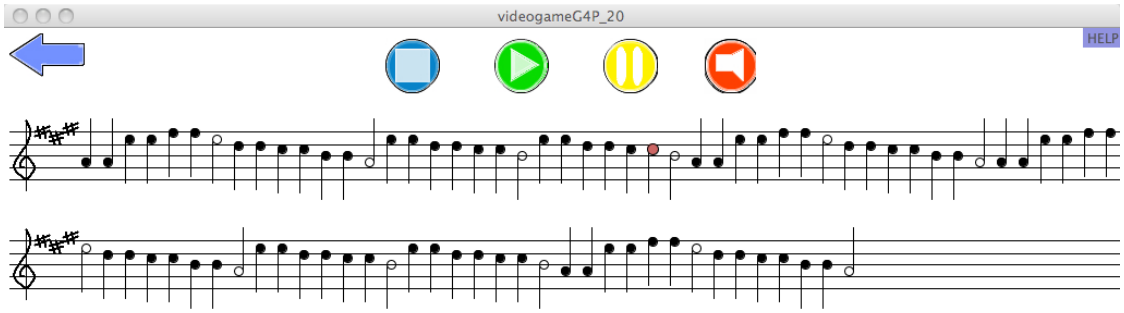

FIG. 37. PANTALLA DE LA PARTITURA

En cuanto a la parte de la interfaz electrónica, se consiguió tener una plantilla con indicadores de la posición de los dedos muy delgada y que no interfiere con la forma de sostener el violín ni con su ejecución. Esta plantilla se hizo con materiales de bajo costo que era uno de los principales objetivos del proyecto. Se utilizaron materiales de fácil adquisición como acetato, velcro, LEDs, resistencias y tinta conductora de plata.

Esta última es la que implica mayor costo pero puede ser utilizada para la elaboración de varias plantillas.

Los LEDs, como se mencionó en el capítulo del desarrollo, son del tipo SMD que son LEDs muy pequeños y planos por lo que no interfieren o rozan con las cuerdas del violín. La tinta conductora permite descartar el uso de cables para la conexión de los LEDs entre los acetatos. Y la conexión hacia el microcontrolador hecha con un cable plano facilitó el diseño y permitió que la plantilla electrónica fuera lo mas delgada posible, por lo que se cumplió uno de los objetivos del diseño del dispositivo electrónico.

Se puede apreciar perfectamente la pisada o lugar en el que se debe de colocar el dedo con la luz emitida por los LED, desde la posición de los ojos hacia el diapasón, sin que se confunda por la emisión de una luz mas intensa que podría lastimar o cansar la vista.

La plantilla se puede reutilizar en diferentes violines ya que se puede quitar y poner las veces que sea necesario sin maltratar el instrumento, su fijación es mediante cintas velcro las cuales nos permiten abrochar y desabrochar el dispositivo electrónico fácilmente.

La sincronización entre la parte visual del videojuego y la parte visual del dispositivo electrónico, que en este caso es la plantilla de LEDs, se da sin ningún retardo debido a que la cantidad de variables o datos no implica un procesamiento complejo para el controlador y la computadora, además de no necesitar varios procesos para procesar estos datos.

Este prototipo nos limita solamente al uso de ciertas partituras en la parte de la programación ya que no se detalla claramente el uso de corcheas o semicorcheas, así como también por el diseño de la plantilla electrónica, que sólo se basa en la posición básica y natural de la mano izquierda del violín por lo que sólo se toman en cuenta partituras que estén basadas en la escala de La mayor.

Capítulo 5. Conclusiones.

En el capitulo 5 se da una retrospectiva de lo que se realizó y lo que se obtuvo con la elaboración de este proyecto.

Con este trabajo se buscó aplicar el uso de la tecnología en un caso práctico: poder diseñar una interfaz como apoyo a la enseñanza del violín de una manera más dinámica y divertida. Los resultados obtenidos cumplen con los objetivos planteados al inicio del proyecto ya que los datos se muestran de la forma en que se diseñaron conceptualmente.

Esta interfaz es única en el mundo y puede ser la primera de varios desarrollos tecnológicos aplicados al arte que se pueden generar en México. La colaboración que se tuvo con la Universidad de Leiden, específicamente con LIACS es importante ya que este tema de tesis se externó a la comunidad de Leiden y generó interés por parte de ellos hacia un desarrollo netamente mexicano y universitario.

El uso de herramientas opensource hace que el desarrollo de este tipo de sistemas sea de bajo costo ya que no se tienen que adquirir licencias o permisos para cargar algún software en una computadora. Estas herramientas son accesibles para todos, por lo que en la parte de uso de recursos para programación no se invirtió más que horas hombre.

Para diseños a futuro se podrían utilizar otros materiales como los accesorios Lilypad. Por ejemplo, el uso de LEDs LilyPad que son de tamaño similar a los LEDs SMD usados en el diseño de la interfaz electrónica pero, tienen la ventaja de tener adaptaciones para insertar ajuga e hilo en sus extremos. Así mismo, usar hilo conductor de acero inoxidable para simular las pistas nos permitiría tener mayor flexibilidad en la plantilla que la tinta conductiva no proporciona.

El problema con la tinta conductiva como se mencionó es que al doblarse el acetato puede llegar a rasgar una pista, las pistas son de un grosor muy estrecho por lo que cualquier rasgadura ya no permite la continuidad entre pistas.

En caso de querer diseñar la interfaz con tinta conductiva, lo recomendable sería usar una pluma como la desarrollada por el grupo de Investigación de Materiales de la Universidad de Illinois. Es una pluma tipo gel, por lo que no se recorre ni mancha cuando se dibuja, por lo que se pudo haber evitado el uso de la navaja para separar las pistas.

Para un futuro trabajo, se tiene pensado poder hacer una medición de la frecuencia a la que se está tocando el instrumento para compararla con la frecuencia de la nota que se debe de tocar, y así indicar si la afinación es correcta o no. Esto se puede realizar utilizando el mismo micrófono de la computadora ya que al usar la interfaz gráfica se necesita forzosamente el uso de una computadora, que por lo general y actualmente la mayoría ya cuentan con un micrófono integrado.

Algunos precios de los materiales que se adquirieron para la elaboración de este proyecto fueron:

Tira de LEDs SMD $ 1.73

Resistencia de superficie 330 ohms $ 1.97

Cable plano flexible 18 pistas $35.00

Pluma de tinta conductora $1000.00 / 4 plantillas = $250.00

Arduino Nano $ 220.00

Cable USB – micro USB $ 57.00

Materiales otros por plantilla (velcro, acetato, etc.) $ 20.00

Total materiales por plantilla: $ 590.00 aproximadamente.

Bibliografía

- Adnré, Paul; Teevan, Jaime; Dumais, Susan. Discovery is Never by Chance: Designing for (Un) Serendipity. ACM New York, EUA, 2009
- Altenmüller, Eckart; Wiesendanger, Mario; Kesselring, Jurg. Music, Motor Control and the Brain. Oxford University Press, 2006.
- Barr, Pippin. VideoGame Values: Play as Human-Computer Interaction. Thesis. Victoria University of Wellington, NZ, 2008
- Barrón Corvera, Jorge. Violín, viola, violonchello y piano: Procesos de aprendizaje de enseñanza-aprendizaje. 1era ed. Edit. Universidad Autónoma de Zacatecas
- Blaine, Tine. The Convergence of Alternate Controllers and Musical Interface in Interactive Entertainment. Carnegie Mellon University, EUA, 2005
- Etxeberria, , Xabier. 1998. "Videojuegos y educación". Comunicar, num. marzo,
- Etxeberria, , Xabier. Videojuegos y educaciónComunicar [en linea] 1998, (marzo) : [Fecha de consulta: 14 de octubre de 2013] Disponible en:<http://www.redalyc.org/articulo.oa?id=15801026> ISSN 1134-3478
- Foster Daniel, Phillips. Illuminating Music: Research and product design study applying synesthesia and ambient peripherial display theory to violin. Auburn University, Alabama, EUA, 2008
- Foster Daniel, Phillips. Illuminating Music: Research and product design study applying synaesthesia and ambient peripheral display theory to violin. Auburn University, Alabama, EUA, 2008
- Frazer, Alex; Argles David; Wills Gary. Demystifying the Educational Benefits of Different Gaming Genres. University of Southampton, UK, 2008
- Habgood, MP and Ainsworth, Shaaron. Motivating children to learn effectively: exploring the value of intrinsic integration in educational games. Sheffield Hallam University, UK, 2011
- Jenson, Jennifer & De Castell, Suzanne. From Simulation to Imitation: New Controllers, New forms of play. Brunel University, UK, 2009
- Jenson, Jennifer; Taylor, Nicholas, Fischer, Stephanie. Critical review and analysis of the issue of “Skill, Technology and Learning”. Faculty of education, York University, Canada.
- Jenson, Jennifer; Taylor, Nicholas; Fischer, Stephanie. Critical review and analysis of the issue of “Skill, Technology and Learning”. Faculty of education, York University, Canada.
- McPherson, Gary. The Child as Musician: A handbook of musical development. Oxford University Press, 2006
- Overholt, Dan. The overtone violin: A new computer music instrument. Create-Centre for research in electronic art technology-U.C, STEIM-Studio for electro instrumental music, Amsterdam, The Netherlands, 2005.
- Overholt, Dan. The overtone violin: A new computer music instrument. Create-Centre for research in electronic art technology-U. C, STEIM-Studio for electro instrumental music, Amsterdam, The Netherlands.

- Prensky, Marc. The digital game-based learning revolution. MacGraw-Hill, 2001
- Ramos Mejía, Carlos. La dinámica del Instrumentista. 2ª. Ed. Edit. Ricordi Americana. Buenos Aires, Argentina
- Rideout, Victoria; Andewater, Elizabeth; Wartella, Ellen. Henry J. Kaiser. Zero to six: Elctronic media in the lives of infants, toddlers ans preschoolers. Family Foundation, CA. 2003
- Rideout, Victoria; Andewater, Elizabeth; Wartella, Ellen. Henry J. Zero to six: Elctronic media in the lives of infants, toddler's and preschoolers. Kaiser Family Foundation, CA. 2003
- Trueman, Dan & Cook, Perry. BoSSA: The deconstructed violin reconstructed. Princeton University, EUA, 1998
- Trueman, Dan (Department of Musica, Princeton University, NJ) y Cook, Perry (Department of Comuter Science, Princeton University, NJ) BoSSA: The deconstructed violin reconstructed, 1998
- Young, Phyllis. Playing the stringgame. Strategies for teaching cello and strings. 5ª.ed. Edit University of Texas press, Austin, USA, 1993

- Young, Phyllis. Playing the stringgame. Strategies for teaching cello and strings. 5ª.ed. Edit University of Texas press, Austin, USA, 1993

Anexo A. Código de programación interfaz gráfica en Processing2.0.

Pantalla videogameG4P_18

```
import arb.soundcipher.*;
import arb.soundcipher.constants.*;
import processing.serial.*;
import g4p_controls.*;

Serial mipuerto;
SoundCipher part1 = new SoundCipher(this);
SCScore score;
//Minim minim;
GImageToggleButton btnSpeaker;
GImageButton btnPlay, btnOptions1, btnOptions2, btnQuit_game, btnGame_info,
btnPlay1, btnPlay2, btnPlay3, btnStop, btnPause, btnBack1, btnBack2, btnBack3,
btnHelp1, btnHelp2, btnHelp3, btnHelp4;
GLabel lblOut, lblVolumen, lblScore, lblTempo, lblSong, lblInstrucciones,
lblCambiotiempo, lblCambiocancion, lblActivacionScore, lblVolumen1;
GCustomSlider sdrVolumen, sdrScore, sdrTempo, sdr;
GDropList dplSong, dplInstrucciones, dplCambiotiempo, dplCambiocancion,
dplActivacionScore, dplVolumen;
GPanel help_screen;

long timer;
String[] files;

int j;
int i = 0;
int mode = 1;
int mute = 7;

float v = 0.0;
float h = 50.0;

int a = 0;
int n = 0;
int c = 0;
int g = 0;
float beat;
float xLoc = 0;
float xLoc2 = 0;
float xLoc3 = 0;
float newyLoc = 438; // if I put this don't works
float yLoc2 = -150;
float lineDistY = 100;
color blancas = color(255);
color negras = color(0);
color nota = color(204,102,150);
  //Del siguiente arreglo las unidades indican dedo, decenas indican cuerda
```

```
int[] pitch_to_LED = {
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 0, pitch 0 a 11
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 1, pitch 12 a 23
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 2, pitch 24 a 35
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 3, pitch 36 a 47
    0, 0, 0, 0, 0, 0, 0, 40, 0, 41, 0, 42, // octava 4, pitch 48 a 59
    43, 0, 44, 0, 31, 0, 32, 33, 0, 34, 0, 21, // octava 5, pitch 60 a 71
    0, 22, 23, 0, 24, 0, 11, 0, 12, 13, 0, 14, // octava 6, pitch 72 a 83
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 7, pitch 84 a 95
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 8, pitch 96 a 107
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // octava 9, pitch 108 a 119
    0, 0, 0, 0, 0, 0, 0 //, 0, 0, 0, 0, 0, // octava 10, pitch 120 a 127
  };

void setup() {  // this is run once.
  println(Serial.list());
  String Puerto = Serial.list()[1];
  mipuerto = new Serial(this, Puerto, 9600);
  //minim = new Minim(this);
  //out = minim.getLineOut();

    size(1024, 768);
    cursor(CROSS);
    make_inicio_screen();
    make_settings_screen();
    make_song_screen();
    make_info_screen();
   // make_help_screen();
    inicio_screen_visible(false);
    settings_screen_visible(false);
    song_screen_visible(false);
    info_screen_visible(false);
    //help_screen_visible(false);
    help_screen();

    bx1 = width ;
    bx2 = bx1+bw;
    by = height /480;
    smooth();
    //frameRate(30);
    space = loadImage("diseno.jpg");
    space2 = loadImage("diseno3.jpg");

    prueba = loadImage("notasborder.jpg");
    cursor(CROSS);

    directorarriba = loadImage("director1.png");
    directorenmedio = loadImage("director2.png");
    directorabajo = loadImage("director3.png");
    director = directorarriba;

    infofondo = loadImage("infofondo.jpg");
    f=createFont("Serif", 20);
    textFont(f);
```

```
    musicborder = loadImage("fondobarda.jpg");

    //noLoop();
  score = new SCScore();
  score.addCallbackListener(this);
  score.addCallback(0.5,1);
  score.play(-1);
  redraw();
}

void draw() {  // this is run repeatedly.
  switch(mode){
    case 1:
      inicio_screen_visible(true);
      settings_screen_visible(false);
      song_screen_visible(false);
      info_screen_visible(false);
      //help_screen_visible(false);
      inicio_screen();
      break;
    case 2:
      settings_screen_visible(true);
      inicio_screen_visible(false);
      song_screen_visible(false);
      info_screen_visible(false);
      //help_screen_visible(false);
      settings_screen();
      break;
    case 3:
      song_screen_visible(true);
      //partitura(true);
      inicio_screen_visible(false);
      settings_screen_visible(false);
      info_screen_visible(false);
      //help_screen_visible(false);
      song_screen();
      break;
    case 4:
      info_screen_visible(true);
      song_screen_visible(false);
      inicio_screen_visible(false);
      settings_screen_visible(false);
      //help_screen_visible(false);
      info_screen();
      break;
    case 5:
      //help_screen_visible(true);
      info_screen_visible(false);
      song_screen_visible(false);
      inicio_screen_visible(false);
      settings_screen_visible(false);
      help_screen();
      break;
  }
  switch(mute){
```

```
    case 6:
       //int j = Integer.parseInt(sdrVolumen.getValueS());
       sdrVolumen.setValue(v);
       break;
    case 7:
       //int j = Integer.parseInt(sdrVolumen.getValueS());
       sdrVolumen.getValueS();
       sdrVolumen.setValue(h);
       break;
     }
}

void handleButtonEvents(GImageButton button, GEvent event){
  /////Botones InicioScreen
  if(button == btnPlay1){
    //println("song_screen");
    mode = 3;
  }
  if(button == btnOptions1){
    //println("options_screen");
    mode = 2;
  }
  if(button == btnHelp3){
    //println("help_screen");
    mode = 5;
  }
  if(button == btnQuit_game){
    //println("quit_game");
    exit();
  }
  if(button == btnGame_info){
    //println("info_screen");
    mode = 4;
  }
  //////BotonesSettingsScreen

  if(button == btnBack2){
    //println("inicio_screen");
    mode = 1;
  }
  if(button == btnHelp2){
    //println("help_screen");
    mode = 5;
  }
  ////Botones SongScreen
  if(button == btnPlay3){
    //println("Play");
    loop();
    score.play(-1);
  }

  if(button == btnStop){
    //println("Stop");
    noLoop();
    score.stop();
```

```
      //if (a >= estrellitanota.length-1){
    a = 0;
    xLoc3 = 0;
    yLoc2 = -150;

    //yLoc2 = height-3*estrellitanota[a]+yLoc2-576;
    //yLoc2 = 0 + lineDistY;
    //}
  }
  if(button == btnPause){
    //println("Pause");
    noLoop();
    score.stop();
  }

  /*if(button == btnSpeaker){
    println("Speaker");
    loop();
    sdrVolumen.setLimits(1, 1, 100);
    }
    else if (button == btnSpeaker){
    loop();
    //sdrVolumen.setLimits(50, 1, 100);
    int j = Integer.parseInt(sdrVolumen.getValueS());
    }*/

  if(button == btnBack3){
    //println("inicio_screen");
    loop();
    mode = 1;
  }
  if(button == btnHelp1){
    //println("help_screen");
    loop();
    mode = 5;
  }
  ////Botones info_screen
  if(button == btnBack1){
    //println("options_screen");
    mode = 1;
  }
  if(button == btnHelp4){
    //println("help_screen");
    mode = 5;
  }
  ////Botones HelpScreen
  /*if(button == btnBack2){
    println("help_screen");
    mode = 2;
  }*/
}

void handleSliderEvents(GValueControl slider, GEvent event){
  /////Slider InicioScreen
  //print("integer value:" + slider.getValueI() + "float value:" + slider.getValueF());
```

```
  ////Slider SettingScreen
  if (slider == sdrVolumen);
    //println(sdrVolumen.getValueS() + " " + event);
    sdrVolumen.getValueS();
  if (slider == sdrScore);
    //println(sdrScore.getValueS()+ " " + event);
  if (slider == sdrTempo);
    //println(sdrTempo.getValueS()+ " " + event);
}

public void handleDropListEvents(GDropList list, GEvent event){
  //System.out.println("Selected " + dplSong.getSelectedIndex() + "" +
dplSong.getSelectedText() + "");
  //System.out.println("Selected " + dplSong.getSelectedIndex() + "" +
dplSong.getSelectedText() + "");
  if(list == dplSong);
  dplSong.getSelectedIndex();
}

public void handleToggleButtonEvents(GImageToggleButton button, GEvent event){
  if(button == btnSpeaker){
    if(btnSpeaker.stateValue() == 0){
    mute = 7;}
  else
    mute = 6;

  }
  //println(button + "State: " + button.stateValue());
}
```

Pantalla Estrellita

```
float[] estrellitanota = {69, 69, 76, 76, 78, 78, 76, 74, 74, 73, 73, 71, 71, 69,
                    76, 76, 74, 74, 73, 73, 71, 76, 76, 74, 74, 73, 73, 71,
                    69, 69, 76, 76, 78, 78, 76, 74, 74, 73, 73, 71, 71, 69,
                    69, 69, 76, 76, 78, 78, 76, 74, 74, 73, 73, 71, 71, 69,
                    76, 76, 74, 74, 73, 73, 71, 76, 76, 74, 74, 73, 73, 71,
                    69, 69, 76, 76, 78, 78, 76, 74, 74, 73, 73, 71, 71, 69};
float[] estrellitadynamics = new float[estrellitanota.length];

float[] estrellitadur = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
                     1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
                     1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
                     1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
                     1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
                     1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0};

void estrellitamusica(){

  int k = Integer.parseInt(sdrTempo.getValueS());
  int j = Integer.parseInt(sdrVolumen.getValueS());

  stroke(0);
  frameRate(k/60);
```

```
  for (int i=0; i<estrellitanota.length; i++) {
    estrellitadynamics[i] = random(40) + 70;
  }

  if (a >= estrellitanota.length-1){
    a = 0;
    }

  if (75+xLoc3 >= width){
    xLoc3 = 0;
    yLoc2 = yLoc2 + lineDistY;
    }

  fill(nota);
  ellipse(xLoc3+75, height-3*estrellitanota[a]+yLoc2-288, 10, 10);
  part1.playNote(estrellitanota[a], j, estrellitadur[a]);
  a += 1;
  xLoc3 += 20.0;

  int newNotas = int(estrellitanota[a]);

  mipuerto.write(pitch_to_LED[newNotas]);
  println(pitch_to_LED[newNotas]);
}

void estrellita_handleCallbacks(int callbackID) {
  //redraw();
  //int newestrellitadur = int(estrellitadur[n]);
  beat += estrellitanota[n];
   //if(estrellitanota[a]>=estrellitanota.length-1){
     //   /*ore.stop();
  //}
  score.addCallback(estrellitanota[n], 1);
  score.play();
}

void estrellitapartitura(){
  background(255);

  if (n >= estrellitanota.length-1){
    n = 0;
    }
  if (75+xLoc >= width){
    xLoc = 0;
    }

    xLoc2 = 0;

 for(n = 0; n<estrellitanota.length; n++){
     //println(n);
     //println(estrellitanota[n]);
    if(75+n*20<width){
    if(estrellitanota[n]>70){
```

```
    line(xLoc-3.5+75, height-3*estrellitanota[n]-438, xLoc-3.5+75, 40+(height-
3*estrellitanota[n]-438));
    }
   if(estrellitanota[n]<70){
    line(xLoc+4+75, height-3*estrellitanota[n]-438, xLoc+4+75, (height-
3*estrellitanota[n]-438)-40);
    }
   if(estrellitadur[n]<=1){
    fill(negras);
    ellipse(xLoc+75, height-3*estrellitanota[n]-newyLoc, 8 , 8);
    }
    if(estrellitadur[n]>1){
    fill(blancas);
    ellipse(xLoc+75, height-3*estrellitanota[n]-newyLoc, 8 , 8);
    }

    //Pentagrama
      line(5, 139.5, width, 139.5);
      line(5, 128.5, width, 128.5);
      line(5, 117.5, width, 117.5);
      line(5, 107, width, 107);
      line(5, 95.5, width, 95.5);
    //Clave Sol
      noFill();
      bezier(20,117.5,20,117.5,7,128.5,20,128.5);
      bezier(20,139.5,30,128.5,30,128.5,20,117.5);
      bezier(20,139.5,5,128.5,5,128.5,20,107.5);
      bezier(20,107.5,25,95.5,25,95.5,20,85.5);
      bezier(20,149.5,20,95.5,20,95.5,20,85.5);
    //Armadura
      line(28,99,41,94);
      line(28,95,41,90);
      line(33,101,30,90);
      line(38,101,35,90);

      line(41,100,54,96);
      line(41,104,54,100);
      line(46,106,42,95);
      line(51,106,47,95);

      line(54,96,67,91);
      line(54,92,67,87);
      line(59,98,56,87);
      line(64,98,61,87);

  }
 xLoc += 20.0;
  if(75+n*20>=width){
        if(estrellitanota[n]>70){
          line(xLoc2-3.5+75, height-3*estrellitanota[n]-338, xLoc2-3.5+75, 40+(height-
3*estrellitanota[n]-338));
          }
        if(estrellitanota[n]<70){
          line(xLoc2+4+75, height-3*estrellitanota[n]-338, xLoc2+4+75, (height-
3*estrellitanota[n]-338)-40);
```

```
            }
          if(estrellitadur[n]<=1){
                fill(negras);
            ellipse(xLoc2+75, height-3*estrellitanota[n]-newyLoc+100, 8 , 8);
            }
          if(estrellitadur[n]>1){
                fill(blancas);
            ellipse(xLoc2+75, height-3*estrellitanota[n]-newyLoc+100, 8 , 8);
            }
        xLoc2 += 20.0;

        //Pentagrama
        line(5, 239.5, width, 239.5);
        line(5, 228.5, width, 228.5);
        line(5, 217.5, width, 217.5);
        line(5, 207, width, 207);
        line(5, 195.5, width, 195.5);
        //Clave de Sol
        noFill();
        bezier(20,217.5,20,217.5,7,228.5,20,228.5);
        bezier(20,239.5,30,228.5,30,228.5,20,217.5);
        bezier(20,239.5,5,228.5,5,228.5,20,207.5);
        bezier(20,207.5,25,195.5,25,195.5,20,185.5);
        bezier(20,249.5,20,195.5,20,195.5,20,185.5);
        //Armadura
        line(28,199,41,194);
        line(28,195,41,190);
        line(33,201,30,190);
        line(38,201,35,190);

        line(41,200,54,196);
        line(41,204,54,200);
        line(46,206,42,195);
        line(51,206,47,195);

        line(54,196,67,191);
        line(54,192,67,187);
        line(59,198,56,187);
        line(64,198,61,187);
    }
    }

}
```

Pantalla Lightly_Row

```
float[] rownota= {76, 73, 73, 74, 71, 71, 69, 71, 73, 74, 76, 76, 76,
                  76, 73, 73, 73, 74, 71, 71, 71, 69, 73, 76, 76, 73,73,73,
                  71, 71, 71, 71, 71, 73, 74, 73, 73, 73, 73, 73, 74, 76,
                  76, 73, 73, 73, 74, 71, 71, 71, 69, 73, 76, 76, 73, 73, 73};
float[] rowdynamics = new float[rownota.length];
float[] rowdur = {1.0,1.0,2.0,1.0,1.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,
                   1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,
```

```
                1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,
                1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0};

void rowmusica(){
  int k = Integer.parseInt(sdrTempo.getValueS());
  int j = Integer.parseInt(sdrVolumen.getValueS());
  //println(rownota.length);
  //println(k/60);
  stroke(0);
  frameRate(k/60);

  for (int i=0; i<rownota.length; i++) {
    rowdynamics[i] = random(40) + 70;
  }

  if (a >= rownota.length-1){
    a = 0;
    }

  if (75+xLoc3 >= width){
    xLoc3 = 0;
    yLoc2 = yLoc2 + lineDistY;
    }

  fill(nota);
  ellipse(xLoc3+75, height-3*rownota[a]+yLoc2-288, 10, 10);
  part1.playNote(rownota[a], j, rowdur[a]);
  a += 1;
  xLoc3 += 20.0;

  int newNotas = int(rownota[a]);

  mipuerto.write(pitch_to_LED[newNotas]);
  println(pitch_to_LED[newNotas]);
  //println(rownota.length);

}

void row_handleCallbacks(int callbackID) {
  //redraw();
  //int newrowdur = int(rowdur[n]);
  beat += rownota[n];
   //if(rownota[a]>=rownota.length-1){
        //score.stop();
  //}
  score.addCallback(rownota[n], 1);
  score.play();
}

void rowpartitura(){
  background(255);

  if (n >= rownota.length-1){
    n = 0;
    }
```

```
 if (75+xLoc >= width){
   xLoc = 0;
   }

   xLoc2 = 0;

 for(n = 0; n<rownota.length; n++){
    //println(n);
    //println(rownota[n]);
   if(75+n*20<width){
   if(rownota[n]>70){
     line(xLoc-3.5+75, height-3*rownota[n]-438, xLoc-3.5+75, 40+(height-3*rownota[n]-
438));
     }
   if(rownota[n]<70){
     line(xLoc+4+75, height-3*rownota[n]-438, xLoc+4+75, (height-3*rownota[n]-438)-
40);
     }
   if(rowdur[n]<=1){
     fill(negras);
     ellipse(xLoc+75, height-3*rownota[n]-newyLoc, 8 , 8);
     }
    if(rowdur[n]>1){
     fill(blancas);
     ellipse(xLoc+75, height-3*rownota[n]-newyLoc, 8 , 8);
     }

     //Pentagrama
       line(5, 139.5, width, 139.5);
       line(5, 128.5, width, 128.5);
       line(5, 117.5, width, 117.5);
       line(5, 107, width, 107);
       line(5, 95.5, width, 95.5);
     //Clave Sol
       noFill();
       bezier(20,117.5,20,117.5,7,128.5,20,128.5);
       bezier(20,139.5,30,128.5,30,128.5,20,117.5);
       bezier(20,139.5,5,128.5,5,128.5,20,107.5);
       bezier(20,107.5,25,95.5,25,95.5,20,85.5);
       bezier(20,149.5,20,95.5,20,95.5,20,85.5);
     //Armadura
       line(28,99,41,94);
       line(28,95,41,90);
       line(33,101,30,90);
       line(38,101,35,90);

       line(41,100,54,96);
       line(41,104,54,100);
       line(46,106,42,95);
       line(51,106,47,95);

       line(54,96,67,91);
       line(54,92,67,87);
       line(59,98,56,87);
       line(64,98,61,87);
```

```
   }
  xLoc += 20.0;
    if(75+n*20>=width){
            if(rownota[n]>70){
               line(xLoc2-3.5+75,   height-3*rownota[n]-338,   xLoc2-3.5+75,   40+(height-
3*rownota[n]-338));
               }
            if(rownota[n]<70){
               line(xLoc2+4+75,      height-3*rownota[n]-338,      xLoc2+4+75,      (height-
3*rownota[n]-338)-40);
               }
            if(rowdur[n]<=1){
                  fill(negras);
              ellipse(xLoc2+75, height-3*rownota[n]-newyLoc+100, 8 , 8);
              }
            if(rowdur[n]>1){
                  fill(blancas);
              ellipse(xLoc2+75, height-3*rownota[n]-newyLoc+100, 8 , 8);
              }
       xLoc2 += 20.0;

       //Pentagrama
       line(5, 239.5, width, 239.5);
       line(5, 228.5, width, 228.5);
       line(5, 217.5, width, 217.5);
       line(5, 207, width, 207);
       line(5, 195.5, width, 195.5);
       //Clave de Sol
       noFill();
       bezier(20,217.5,20,217.5,7,228.5,20,228.5);
       bezier(20,239.5,30,228.5,30,228.5,20,217.5);
       bezier(20,239.5,5,228.5,5,228.5,20,207.5);
       bezier(20,207.5,25,195.5,25,195.5,20,185.5);
       bezier(20,249.5,20,195.5,20,195.5,20,185.5);
       //Armadura
       line(28,199,41,194);
       line(28,195,41,190);
       line(33,201,30,190);
       line(38,201,35,190);

       line(41,200,54,196);
       line(41,204,54,200);
       line(46,206,42,195);
       line(51,206,47,195);

       line(54,196,67,191);
       line(54,192,67,187);
       line(59,198,56,187);
       line(64,198,61,187);
   }

}
}
```

Pantalla Wind

```
float[] windnota = {69, 71, 73, 74, 76, 76, 76, 76, 78, 74, 81, 78, 76,
                    78, 74, 81, 78, 76, 76, 74, 74, 74, 74, 73, 73, 73, 73, 71, 71, 71,
                    69, 73, 76, 76, 74, 74, 74, 74, 73, 73, 73, 73, 71, 71, 71, 69,
                    69, 71, 73, 74, 76, 76, 76, 76, 78, 74, 81, 78, 76,
                    78, 74, 81, 78, 76, 76, 74, 74, 74, 74, 73, 73, 73, 73, 71, 71, 71,
                    69, 73, 76, 76, 74, 74, 74, 74, 73, 73, 73, 73, 71, 71, 71, 69};

float[] winddynamics = new float[windnota.length];

float[] winddur = {0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 2.0,
                   0.5, 0.5, 0.5, 0.5, 2.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
                   0.5, 0.5, 1.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 2.0,
                   0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 2.0,
                   0.5, 0.5, 0.5, 0.5, 2.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
                   0.5, 0.5, 1.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 2.0};

void windmusica(){

  int k = Integer.parseInt(sdrTempo.getValueS());
  int j = Integer.parseInt(sdrVolumen.getValueS());

  //println(k/60);
  stroke(0);
  frameRate(k/60);

  for (int i=0; i<windnota.length; i++) {
    winddynamics[i] = random(40) + 70;
  }

  if (a >= windnota.length-1){
    a = 0;
    }

  if (75+xLoc3 >= width){
    xLoc3 = 0;
    yLoc2 = yLoc2 + lineDistY;
    }

  fill(nota);
  ellipse(xLoc3+75, height-3*windnota[a]+yLoc2-288, 10, 10);
  part1.playNote(windnota[a], j, winddur[a]);
  a += 1;
  xLoc3 += 20.0;

  int newNotas = int(windnota[a]);

  mipuerto.write(pitch_to_LED[newNotas]);
  println(pitch_to_LED[newNotas]);
}

void wind_handleCallbacks(int callbackID) {
  //redraw();
```

```
  //int newwinddur = int(winddur[n]);
  beat += windnota[n];
   //if(windnota[a]>=windnota.length-1){
     //   /*score.stop();
  //}
  score.addCallback(windnota[n], 1);
  score.play();
}

void windpartitura(){
  background(255);

  if (n >= windnota.length-1){
    n = 0;
    }
  if (75+xLoc >= width){
    xLoc = 0;
    }

    xLoc2 = 0;

 for(n = 0; n<windnota.length; n++){
     //println(n);
     //println(windnota[n]);
    if(75+n*20<width){
    if(windnota[n]>70){
      line(xLoc-3.5+75,        height-3*windnota[n]-438,        xLoc-3.5+75,        40+(height-
3*windnota[n]-438));
      }
    if(windnota[n]<70){
      line(xLoc+4+75, height-3*windnota[n]-438, xLoc+4+75, (height-3*windnota[n]-438)-
40);
      }
    if(winddur[n]<=1){
      fill(negras);
      ellipse(xLoc+75, height-3*windnota[n]-newyLoc, 8 , 8);
      }
     if(winddur[n]>1){
      fill(blancas);
      ellipse(xLoc+75, height-3*windnota[n]-newyLoc, 8 , 8);
      }

      //Pentagrama
        line(5, 139.5, width, 139.5);
        line(5, 128.5, width, 128.5);
        line(5, 117.5, width, 117.5);
        line(5, 107, width, 107);
        line(5, 95.5, width, 95.5);
      //Clave Sol
        noFill();
        bezier(20,117.5,20,117.5,7,128.5,20,128.5);
        bezier(20,139.5,30,128.5,30,128.5,20,117.5);
        bezier(20,139.5,5,128.5,5,128.5,20,107.5);
        bezier(20,107.5,25,95.5,25,95.5,20,85.5);
        bezier(20,149.5,20,95.5,20,95.5,20,85.5);
```

```
    //Armadura
      line(28,99,41,94);
      line(28,95,41,90);
      line(33,101,30,90);
      line(38,101,35,90);

      line(41,100,54,96);
      line(41,104,54,100);
      line(46,106,42,95);
      line(51,106,47,95);

      line(54,96,67,91);
      line(54,92,67,87);
      line(59,98,56,87);
      line(64,98,61,87);

   }
  xLoc += 20.0;
   if(75+n*20>=width){
          if(windnota[n]>70){
            line(xLoc2-3.5+75,  height-3*windnota[n]-338,  xLoc2-3.5+75,  40+(height-
3*windnota[n]-338));
            }
          if(windnota[n]<70){
            line(xLoc2+4+75,     height-3*windnota[n]-338,     xLoc2+4+75,     (height-
3*windnota[n]-338)-40);
            }
          if(winddur[n]<=1){
               fill(negras);
           ellipse(xLoc2+75, height-3*windnota[n]-newyLoc+100, 8 , 8);
           }
          if(winddur[n]>1){
               fill(blancas);
           ellipse(xLoc2+75, height-3*windnota[n]-newyLoc+100, 8 , 8);
           }
      xLoc2 += 20.0;

      //Pentagrama
      line(5, 239.5, width, 239.5);
      line(5, 228.5, width, 228.5);
      line(5, 217.5, width, 217.5);
      line(5, 207, width, 207);
      line(5, 195.5, width, 195.5);
      //Clave de Sol
      noFill();
      bezier(20,217.5,20,217.5,7,228.5,20,228.5);
      bezier(20,239.5,30,228.5,30,228.5,20,217.5);
      bezier(20,239.5,5,228.5,5,228.5,20,207.5);
      bezier(20,207.5,25,195.5,25,195.5,20,185.5);
      bezier(20,249.5,20,195.5,20,195.5,20,185.5);
      //Armadura
      line(28,199,41,194);
      line(28,195,41,190);
      line(33,201,30,190);
      line(38,201,35,190);
```

```
      line(41,200,54,196);
      line(41,204,54,200);
      line(46,206,42,195);
      line(51,206,47,195);

      line(54,196,67,191);
      line(54,192,67,187);
      line(59,198,56,187);
      line(64,198,61,187);
   }
   }

}

/*float[] windnota = {69, 71, 73, 74, 76, 76, 76, 76, 78, 74, 81, 78, 76,
                    78, 74, 81, 78, 76, 76, 74, 74, 74, 74, 73, 73, 73, 73, 71, 71, 71,
                    69, 73, 76, 76, 74, 74, 74, 74, 73, 73, 73, 73, 71, 71, 71, 69};
float[] winddynamics = new float[windnota.length];

float[] winddur = {0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 2,
                   0.5, 0.5, 0.5, 0.5, 2, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
                   0.5, 0.5, 1, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 2};*/
```

Pantalla Help Screen

```
PImage musicborder;

void help_screen(){

   //G4P.setGlobalColorScheme(G4P.GOLD_SCHEME);
    help_screen = new GPanel(this, 990, 0, 612, 614, "HELP");

  //background(musicborder);
//}

//void make_help_screen(){

  int x = width, y = 10;

  lblInstrucciones = new GLabel(this, 75, y+50, 135, 19, "INSTRUCCIONES");
  lblInstrucciones.setTextBold();
  String[] items = new String[]{
   "Para iniciar el juego pulsa el boton de PLAY en la pantalla",
   "de inicio, cuando estes en la pantalla con la partitura ",
   "pulsa el boton de play o de triangulo",
   "para que la melodia y la imagen empiecen a tocar;",
   "al iniciar, la nota que va en turno cambiara de color",
   "y en la interfaz del violin se encendera un led que indica",
   "el lugar en donde debes colocar el dedo"
  };
  dplInstrucciones = new GDropList(this, 210, y+50, 350, 96, 5);
```

```
dplInstrucciones.setItems(items, 0);

lblActivacionScore = new GLabel(this, 75, y+125, 81, 19, "SCORE");
lblActivacionScore.setTextBold();
String[] score = new String[]{
 "Para mostrar u ocultar el Marcador en la pantalla",
 "con la partitura ir al menu Opciones y junto a la",
 "palabra Score seleccionar On u Off segun sea el caso.",
 "Para regresar al juego da clock en el boton",
 "PLAY para regresar a la pantalla de la partitura,",
 "da play y listo.",
};
dplActivacionScore = new GDropList(this, 166, y+125, 350, 96, 5);
dplActivacionScore.setItems(score, 0);

lblVolumen1 = new GLabel(this, 75, y+200, 100, 19, "VOLUMEN");
lblVolumen1.setTextBold();
String[] volumen = new String[]{
 "Para cambiar el volumen desde el videojuego",
 "ir al menu Opciones. Una vez en la pantalla de ",
 "opciones mover la bolita del slider en la posicion",
 "que se requiera, siendo 0 el volumen mas bajo y ",
 "100 el volumen mas alto. Para regresar al juego da",
 "click en el botón PLAY para regresar a la",
 "pantalla de la partitura, da play y listo.",
};
dplVolumen = new GDropList(this, 170, y+200, 350, 96, 5);
dplVolumen.setItems(volumen, 0);

lblCambiotiempo = new GLabel(this, 75, y+275, 150, 19, "CAMBIO DE TIEMPO");
lblCambiotiempo.setTextBold();
String[] tiempo = new String[]{
 "Para cambiar el tiempo ve al menu de opciones.",
 "Una vez en la pantalla de Opciones, da click",
 "en la flecha que esta junto a la palabra TIEMPO",
 "y se mostraran varios valores. Selecciona",
 "el valor que desees y listo.",
 "Para regresar al juego da click en el boton",
 "PLAY para regresar a la pantalla de la partitura,",
 "da play y listo.",
};
dplCambiotiempo = new GDropList(this, 230, y+275, 350, 96, 5);
dplCambiotiempo.setItems(tiempo, 0);

lblCambiocancion = new GLabel(this, 75, y+350, 160, 19, "CAMBIO DE CANCION");
lblCambiocancion.setTextBold();
String[] cancion = new String[]{
 "Para cambiar la canción ve al menu Opciones,",
 "da click en la flecha junto a Chnage Song y se",
 "mostraran las diferentes melodías, selecciona la ",
 "que desees, da click en el boton de PLAY para regresar",
 "a la pantalla con la partitura nueva y listo"
};
dplCambiocancion = new GDropList(this, 230, y+350, 350, 96, 5);
dplCambiocancion.setItems(cancion, 0);
```

```
  lblOut = new GLabel(this, 10, 190, 560, 20, "");
  lblOut.setTextAlign(GAlign.CENTER, null);
  timer = millis() - 5000;

  help_screen.addControl(dplInstrucciones);
  help_screen.addControl(lblInstrucciones);
  help_screen.addControl(lblActivacionScore);
  help_screen.addControl(dplActivacionScore);
  help_screen.addControl(lblVolumen1);
  help_screen.addControl(dplVolumen);
  help_screen.addControl(lblCambiotiempo);
  help_screen.addControl(dplCambiotiempo);
  help_screen.addControl(lblCambiocancion);
  help_screen.addControl(dplCambiocancion);
  help_screen.addControl(lblOut);
  help_screen.setAlpha(200, true);

}

/*void help_screen_visible(boolean visible){
  lblInstrucciones.setVisible(visible);
  dplInstrucciones.setVisible(visible);
  lblCambiotiempo.setVisible(visible);
  dplCambiotiempo.setVisible(visible);
  lblCambiocancion.setVisible(visible);
  dplCambiocancion.setVisible(visible);
  lblActivacionScore.setVisible(visible);
  dplActivacionScore.setVisible(visible);
  lblVolumen1.setVisible(visible);
  dplVolumen.setVisible(visible);
  //btnBack2.setVisible(visible);
}*/
```

Pantalla Info Screen

```
PImage infofondo;
PFont f;

void info_screen(){
  info_screen_visible(true);
  background(infofondo);
  textAlign(CENTER);
  drawType(width*0.5);
}

void drawType(float x) {
  fill(255);
  text("Este programa se realizo", x, 100);
  text("mediante Processing2.0, Arduino1.0.4", x, 150);
  text("panit.Net y Pixlr Editor.", x, 200);
  text("Mexico DF, 2013", x, 250);
}
```

```
void make_info_screen(){

  int x = width, y = 10;

  files = new String[]{
    "backOff.jpg", "backOn.jpg"
  };
  btnBack1 = new GImageButton(this, 0, 0, files);

}

void info_screen_visible(boolean visible){
  btnBack1.setVisible(visible);
}
```

Pantalla Inicio Screen

```
PImage space = null;
PImage space2 = null;
//PImage Play;

//int pos_x = 200;
//int pos_y = 200;

int bx1 = 0;
int bx2 = 0;
int by = 0;
int bw = 1024;
float accel = 4; // Velocidad de la imagen

//float ex = random(1024);
//float ey = random(768);
//float extimer = 0;
//float eytimer = 0;
//float ex2 = 0;
//float ey2 = 0;

void inicio_screen(){
    inicio_screen_visible(true);
    frameRate(40);
     image(space,bx1,by);
     image(space2,bx2,by);
     bx1 -= accel;
     bx2 -= accel;
     if (bx1+bw < 0) {
         bx1 = bx2 + bw;
    }
    if (bx2+bw< 0) {
        bx2 = bx1 + bw;
    }

}
```

```
void make_inicio_screen(){
  int x = width, y = 10;
  files = new String[]{
    "playOff.jpg", "playOn.jpg"
  };
  btnPlay1 = new GImageButton(this, 50, 700, files);

  files = new String[]{
    "optionsOff.jpg", "optionsOn.jpg"
  };
  btnOptions1 = new GImageButton(this, 300, 700, files);

  files = new String[]{
    "game_infoOff.jpg", "game_infoOn.jpg"
  };
  btnGame_info = new GImageButton(this, 550, 700, files);

  files = new String[]{
    "quit_gameOff.jpg", "quit_gameOn.jpg"
  };
  btnQuit_game = new GImageButton(this, 880, 700, files);

}

void inicio_screen_visible(boolean visible){
  btnPlay1.setVisible(visible);
  btnOptions1.setVisible(visible);
  btnGame_info.setVisible(visible);
  btnQuit_game.setVisible(visible);
}

/*void handleButtonEvents1(GImageButton button, GEvent event){
  if(button == btnPlay1){
    println("inicio_screen");
    //song_screen();
  }
  if(button == btnOptions1){
    println("options_screen");
    //settings_screen();
  }
  //G4P.setGlobalAlpha(0);
}*/
```

Pantalla Setting Screen

```
int numFrames=4;
int frame = 0;
PImage[] images = new PImage[numFrames];
PImage[] engraneverde = new PImage[numFrames];
PImage prueba;

void settings_screen(){
```

```
  settings_screen_visible(true);
  background(prueba);
  frameRate(5);

  //stroke(280, 150, 50);
  //strokeWeight(5);
  rect(75, 160, 400, 250, 10);
  fill(255);
  rect(565, 160, 400, 250, 10);
  fill(255);
}

void make_settings_screen(){

  int x = width, y = 10;

  String[] tickLabels = new String[]{"On-", "Off"};
  sdrScore = new GCustomSlider(this, 200, y+160, 40, 50, null);
  sdrScore.setShowDecor(false, true, true, true);
  sdrScore.setLimits(1, 1, 2);
  sdrScore.setTickLabels(tickLabels);
  lblScore = new GLabel(this, 100, y+160, 80, 50, "Score");
  lblScore.setTextAlign(GAlign.RIGHT, null);
  lblScore.setTextBold();

  sdrVolumen = new GCustomSlider(this, 700, y+160, 200, 50, null);
  sdrVolumen.setShowDecor(false, true, true, true);
  sdrVolumen.setNbrTicks(9);
  sdrVolumen.setLimits(50, 1, 100);
  lblVolumen = new GLabel(this, 600, y+160, 80, 50, "Volumen");
  lblVolumen.setTextAlign(GAlign.RIGHT, null);
  lblVolumen.setTextBold();

  lblSong = new GLabel(this, 100, y+300, 80, 19, "Change Song");
  lblSong.setTextBold();
  String[] items = new String[]{
    "Twinkle Twinkle Little Star",
    "Lightly Row",
    "Song of the Wind",
    "O Come, Little Children",
    "May Song",
    "Allegro",
    "Minuet 1 Bach",
    "Minuet 2 Bach",
    "Minuet 3 Bach",
    "The Happy Farmer",
    "Gavotte" };
  dplSong = new GDropList(this, 200, y+300, 250, 96, 5);
  dplSong.setItems(items, 0);

  sdrTempo = new GCustomSlider(this, 700, y+285, 200, 50, "Tempo");
  sdrTempo.setShowDecor(false, true, true, true);
  sdrTempo.setNbrTicks(10);
  sdrTempo.setLimits(100, 50, 300);
  lblTempo = new GLabel(this, 600, y+285, 80, 50, "Tempo");
```

```
  lblTempo.setTextAlign(GAlign.RIGHT, null);
  lblTempo.setTextBold();

  files = new String[]{
    "backOff.jpg", "backOn.jpg"
  };
  btnBack2 = new GImageButton(this, 20, 40, files);
}

void settings_screen_visible(boolean visible){
  sdrVolumen.setVisible(visible);
  lblVolumen.setVisible(visible);
  sdrScore.setVisible(visible);
  lblScore.setVisible(visible);
  sdrTempo.setVisible(visible);
  lblTempo.setVisible(visible);
  dplSong.setVisible(visible);
  lblSong.setVisible(visible);
  btnBack2.setVisible(visible);
}

/*void handleButtonEvents(GImageButton button, GEvent event){
  if(button == btnQuit_game){
    println("quit_game");
  }
  if(button == btnPlay2){
    println("songs_screen");
    song_screen();
  }
  if(button == btnGame_info){
    println("info_screen");
  }
}
void handleSliderEvents2(GValueControl slider, GEvent event){
  if (slider == sdrVolumen)
    println(sdrVolumen.getValueS() + " " + event);
  if (slider == sdrScore)
    println(sdrScore.getValueS()+ " " + event);
}

public void handleDropListEvents2(GDropList list, GEvent event){
  System.out.println("Selected " + dplSong.getSelectedIndex() + "" + 
dplSong.getSelectedText() + "");
  System.out.println("Selected " + dplTempo.getSelectedIndex() + "" + 
dplTempo.getSelectedText() + "");
}*/
```

Pantalla Song Screen

```
PImage director;
PImage directorarriba;
PImage directorenmedio;
PImage directorabajo;
```

```
int cancion;

void song_screen(){

  song_screen_visible(true);

  switch(cancion){
    case 0:
    estrellitapartitura();
    estrellitamusica();
    break;

    case 1:
    rowpartitura();
    rowmusica();
    break;

    case 2:
    windpartitura();
    windmusica();
    break;
  }

  int l = dplSong.getSelectedIndex();
  cancion = l;

  //rowmusica();
  //rowpartitura();
  smooth();

   // limit the number of frames per second

  image(director, 909, 638);
  i++;
  if(i % 2 == 0)
  {
    director = directorarriba;
  }
  else
  {
    director = directorabajo;
  }
}

void make_song_screen(){

  int x = width, y = 10;
  G4P.setCursor(ARROW);
  files = new String[]{
    "stopOff.jpg", "stopOn.jpg"
  };
  btnStop = new GImageButton(this, 350, 10, files);
  files = new String[]{
    "play1Off.jpg", "play1On.jpg"
  };
```

```
  btnPlay3 = new GImageButton(this, 450, 10, files);

  files = new String[]{
    "pauseOff.jpg", "pauseOn.jpg"
  };
  btnPause = new GImageButton(this, 550, 10, files);

  //files = new String[]{
    //"speakerOff.jpg", "speakerOn.jpg"
  //};
  btnSpeaker = new GImageToggleButton(this, 620, -12,"speaker.png", 2 , 1);

  files = new String[]{
    "backOff.jpg", "backOn.jpg"
  };
  btnBack3 = new GImageButton(this, 5, 5, files);

}

void song_screen_visible(boolean visible){
  btnPlay3.setVisible(visible);
  btnStop.setVisible(visible);
  btnPause.setVisible(visible);
  btnSpeaker.setVisible(visible);
  btnBack3.setVisible(visible);
}

/*void handleButtonEvents3(GImageButton button, GEvent event){
  if(button == btnPlay3){
    //lblOut.setText("Play");
    println("Play");
    //G4P.setGlobalAlpha(0);
  }
  if(button == btnStop){
    //lblOut.setText("Stop");
    println("Stop");
    //G4P.setGlobalAlpha(255);
  }
  if(button == btnPause){
    //lblOut.setText("Stop");
    println("Pause");
    //G4P.setGlobalAlpha(255);
  }
  if(button == btnSpeaker){
    //lblOut.setText("Stop");
    println("Speaker");
    //G4P.setGlobalAlpha(255);
  }
  if(button == btnOptions2){
    //lblOut.setText("Stop");
    println("option_screen");
    //G4P.setGlobalAlpha(255);
  }
}*/
```

Anexo B Código de programación Arduino Nano

```
int Si5 = 14; // the pin that the LED is attached to 11
int Do6 = 13; // the pin that the LED is attached to 12
int Re6 = 11; // the pin that the LED is attached to 13
int Mi6 = 10; // the pin that the LED is attached to 14
int La5 = 9; // the pin that the LED is attached to 21
int Sol5 = 8; // the pin that the LED is attached to 22
int Fa5 = 7; // the pin that the LED is attached to 23
int Mi5 = 6; // the pin that the LED is attached to 24
int Re5 = 5; // the pin that the LED is attached to 31
int Do5 = 4; // the pin that the LED is attached to 32
int Si4 = 3; // the pin that the LED is attached to 33
int La4 = 2; // the pin that the LED is attached to 34
int Si6 = 15; // the pin that the LED is attached to 41
int La6 = 16; // the pin that the LED is attached to 42
int Sol6 = 17; // the pin that the LED is attached to 43
int Fa6 = 18; // the pin that the LED is attached to 44
int incomingByte;      // a variable to read incoming serial data into

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  // initialize the LED pin as an output:
  pinMode(Fa6, OUTPUT);
  pinMode(Sol6, OUTPUT);
  pinMode(La6, OUTPUT);
  pinMode(Si6, OUTPUT);
  pinMode(Si5, OUTPUT);
  pinMode(Do6, OUTPUT);
  pinMode(Re6, OUTPUT);
  pinMode(Mi6, OUTPUT);
  pinMode(Mi5, OUTPUT);
  pinMode(Fa5, OUTPUT);
  pinMode(Sol5, OUTPUT);
  pinMode(La5, OUTPUT);
  pinMode(La4, OUTPUT);
  pinMode(Si4, OUTPUT);
  pinMode(Do5, OUTPUT);
  pinMode(Re5, OUTPUT);
}

void loop() {
  // see if there's incoming serial data:
  if (Serial.available() > 0) {
    // read the oldest byte in the serial buffer:
    incomingByte = Serial.read();
    // if it's a capital H (ASCII 72), turn on the LED:
    if (incomingByte == 11) {
      digitalWrite(Fa6, HIGH);
    }
    else
```

```
{
  digitalWrite(Fa6, LOW);
}
if (incomingByte == 12) {
  digitalWrite(Sol6, HIGH);
}
else
{
  digitalWrite(Sol6, LOW);
}
 if (incomingByte == 13) {
  digitalWrite(La6, HIGH);
}
else
{
  digitalWrite(La6, LOW);
}
    if (incomingByte == 14) {
  digitalWrite(Si6, HIGH);
}
else
{
  digitalWrite(Si6, LOW);
}
    if (incomingByte == 21) {
  digitalWrite(Si5, HIGH);
}
else
{
  digitalWrite(Si5, LOW);
}
    if (incomingByte == 22) {
  digitalWrite(Do6, HIGH);
}
else
{
  digitalWrite(Do6, LOW);
}
    if (incomingByte == 23) {
  digitalWrite(Re6, HIGH);
}
else
{
  digitalWrite(Re6, LOW);
}
    if (incomingByte == 24) {
  digitalWrite(Mi6, HIGH);
}
else
{
  digitalWrite(Mi6, LOW);
}
    if (incomingByte == 31) {
  digitalWrite(Mi5, HIGH);
}
```

```
    else
    {
      digitalWrite(Mi5, LOW);
    }
       if (incomingByte == 32) {
      digitalWrite(Fa5, HIGH);
    }
    else
    {
      digitalWrite(Fa5, LOW);
    }
    if (incomingByte == 33) {
      digitalWrite(Sol5, HIGH);
    }
    else
    {
      digitalWrite(Sol5, LOW);
    }
    if (incomingByte == 34) {
      digitalWrite(La5, HIGH);
    }
    else
    {
      digitalWrite(La5, LOW);
    }
     if (incomingByte == 41) {
      digitalWrite(La4, HIGH);
    }
    else
    {
      digitalWrite(La4, LOW);
    }
       if (incomingByte == 42) {
      digitalWrite(Si4, HIGH);
    }
    else
    {
      digitalWrite(Si4, LOW);
    }
       if (incomingByte == 43) {
      digitalWrite(Do5, HIGH);
    }
    else
    {
      digitalWrite(Do5, LOW);
    }
    if (incomingByte == 44) {
      digitalWrite(Re5, HIGH);
    }
    else
    {
      digitalWrite(Re5, LOW);
    }
  }
}
```