UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño y desarrollo de software para aprender guitarra

TESIS

Que para obtener el titulo de

Ingeniero en computación

P R E S E N T A

Rodrigo Díaz Maldonado

DIRECTOR DE TESIS

Ing. Alberto Templos Carbajal

Ciudad Universitaria, Cd. Mx., 2016

Agradecimientos

A mis padres Leticia Maldonado Rodríguez y Guillermo Díaz Castañeda por brindarme su apoyo para concluir mi formación profesional.

Al ingeniero Alberto Templos Carbajal por ser un gran profesor y una gran persona, le doy un agradecimiento especial por su ayuda para la selección del tema y apoyo para que el trabajo se acabara en tiempo.

A Claudia Córdova Cruz por su cariño y apoyo para la revisión de este trabajo y aclaración de algunos puntos de la investigación.

A mis familiares por su apoyo durante el desarrollo del sistema y las fases de prueba, en especial a Mario Alberto por haber colaborado en la realimentación para un mejor diseño.

Índice

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

Hoy en día el conocimiento se puede obtener de varias formas desde leer un libro hasta interactuando con una aplicación de móvil o software para ordenador, existen diversas aplicaciones que enseñan distintas disciplinas por ejemplo matemáticas, inglés, programación, incluso música. Si analizamos las aplicaciones que existen para la instrucción de música, en concreto para la enseñanza de guitarra nos podemos dar cuenta que la mayoría son estáticas, es decir no existe una interacción entre el usuario y la aplicación, en este trabajo se propone un diseño diferente a las aplicaciones existentes en el mercado, sin perder de vista el objetivo principal, el cual es que el usuario aprenda o mejore sus habilidades con el instrumento musical.

El tema de investigación tratado en esta tesis involucra la propuesta de diseño y desarrollo de un programa para ordenador y móvil, el cual guiará al usuario a través de ejercicios sencillos que le permitirán mejorar su técnica y aprender las escalas más empleadas en la guitarra y en otros instrumentos musicales, así mismo el diseño de esta aplicación estará orientado a despertar el interés del usuario para continuar practicando con la guitarra o poner a prueba sus habilidades con otros instrumentos.

En el pasado los videojuegos fueron una forma de entretenimiento más, hoy día es una de las industria más grande de entretenimiento con presupuestos de desarrollo que van desde los 60 a 250 millones de dólares (solo para producciones grandes)[1], es por ello que resulta factible aprovechar su tecnología y ciertos atributos para usarlos con otros fines, los serious games son una de esas herramientas que aprovechan dichos atributos pues desde el 2008 están siendo utilizados en diversas áreas como la medicina, el entrenamiento militar, la música y la enseñanza de programación entre otras, a pesar de que han tenido un impacto positivo en países como España y Estados Unidos uno de los mayores retos que enfrentan los serious games es la escasa investigación pedagógica y la poca homogenización entre las diversas aplicaciones desarrolladas para poder clasificarlos adecuadamente y estandarizar su implementación, en México la industria de desarrollo de videojuegos está en fase de desarrollo pues según el estudio realizado por The Competitive Intelligence Unit el 45% de desarrolladores de videojuegos dentro del país tiene un segundo trabajo debido a que no cuentan con una remuneración suficiente en este mercado[2], más allá de los problemas de la industria nacional la intención de este trabajo de investigación es aportar ideas, métodos y conocimientos a la industria del videojuego nacional.

1.1.-Objetivos

El principal objetivo de este proyecto es el diseño y desarrollo de un software que permita al usuario aprender y entender las bases más importantes de la teoría musical y como se relaciona con una guitarra acústica convencional de forma entretenida y divertida. El diseño abordará conceptos de programación y pedagógicos para hacer de un simple software una herramienta que pueda ser valorada como una guía para el aprendizaje y motivación. Como objetivos secundarios se tienen los siguientes puntos:

- Demostrar que el diseño del sistema es usable, funcional, y diferente a sistemas similares.

- Poner a prueba una metodología de diseño de serious games dando ejemplos a lo largo del desarrollo de un prototipo que servirán de guía para trabajos futuros que aborden un tema similar al propuesto.

- A partir del prototipo final se darán propuestas para la mejora y se evaluara el desempeño del mismo así como su nivel de exactitud.

- Se hará especial énfasis en aplicar y mejorar un algoritmo de reconocimiento de notas y acordes musicales en tiempo real.

1.2.-Planteamiento del problema y descripción de la propuesta

En los últimos años se ha observado un aumento exponencial en la oferta de aplicaciones para cualquier dispositivo para aprender alguna habilidad cognitiva (Rosseta Stone, Duolingo, Math vs zombies, Binary Game**,** Code Combat, etc.), centrándonos en el tema del aprendizaje de guitarra uno se puede dar cuenta que hay diversas aplicaciones que explican la teoría de guitarra acústica a través de texto y ejercicios propuestos (ezChords, iJam, Coach guitar, etc.), sin embargo no todas son interactivas ya que el hacer una aplicación interactiva resulta bastante cara y difícil de desarrollar, por eso que en este trabajo de investigación se propone una forma para desarrollar una aplicación para aprender guitarra de forma interactiva y atractiva con recursos limitados y con una ventaja adicional la cual consiste en la implementación y mejora de un algoritmo capaz de reconocer tonos individuales o polifónicos de una guitarra acústica convencional y se transcribirán a una forma textual y gráfica en tiempo real (parecido a un afinador digital) proporcionando una mayor interactividad entre el usuario y la aplicación.

Una de las razones principales por las cuales se decidió desarrollar el sistema basado en las características de serious game fue porque explorando diversos sistemas interactivos con funcionalidades similares al propuesto en este trabajo, uno se puede dar cuenta que tienen una cosa en común: una tablatura (concepto que se verá más adelante) dinámica correspondiente a una melodía seleccionada de un listado de canciones en su mayoría en inglés, el problema con ese diseño es que si se dispone de pocos recursos económicos es casi imposible incorporar canciones sujetas a derecho de autor, es por ello que se buscó una alternativa flexible con la que se pudiera realizar un sistema capaz de mejorar o igualar a los sistemas ya existentes con recursos limitados, la conclusión a la que se llego fue que el desarrollo basado en las características de un serious game sería la mejor opción pues es lo bastante flexible para incluir recursos interesantes, ya que al unir la pedagogía, el arte y tecnología el resultado final será un sistema capaz de trasmitir conocimientos significativos de una forma estratégica a través de recursos como: símbolos, sonidos, colores, gráficos y

animaciones, dichos recursos pueden ser aplicados a un conjunto de actividades lúdicas que responden a los objetivos y escenarios pedagógicos.

Características principales del sistema:

- ✓ Diseño e integración de actividades diferente a productos semejantes.
- ✓ Aplicación interactiva con el usuario.
- ✓ Los periféricos de entrada serán: el micrófono, teclado o la pantalla del dispositivo donde se ejecute la aplicación.
- ✓ El sistema tendrá un módulo de realimentación y motivación.
- ✓ Existirá una metáfora ligada al propósito del sistema y los objetivos establecidos durante la fase de diseño.
- ✓ Tendrá diferentes escenarios con arte en dos dimensiones minimalista y mecánicas. (las reglas del juego) dependientes a las actividades propuestas.
- ✓ Existirán retos con ejercicios diferentes con varios niveles de dificultad.

1.3.- Resultados esperados

Se espera desarrollar un software que cumpla como objetivo trasmitir una enseñanza a través de un conjunto de actividades lúdicas que permitirá al usuario familiarizarse con la teoría básica musical y entender cómo se relaciona con la guitarra y poderla aplicar a mas instrumentos (los demás instrumentos también hacen uso de temas que se tratarán: escalas, notación de notas, tono de las notas y acordes).

Capítulo 2: Estado del arte

2.1.-Los serious games

Existen diversas definiciones para los "serious game", la mayoría de autores los catalogan como aquellas herramientas electrónicas, métodos o técnicas diseñadas con la intención de trasmitir alguna enseñanza o habilidad en diversos ámbitos sociales por ejemplo: entrenamiento militar, educación de la salud, programación, matemáticas etc.

Una de las mayores ventajas de los serious games es que el usuario no tiene nada que ganar ni nada que perder, pues se abordan situaciones simuladas por consiguiente no existe presión para reaccionar de un modo u otro ante una situación predefinida, la idea proviene del método de ensayo-error, el usuario podrá cometer cualquier cantidad de errores y en cada iteración actuara de una mejor forma que la anterior. Otra ventaja notable es que se puede implementar un sistema de puntuación cuya función principal es atrapar la atención del usuario y motivarlo a continuar hasta el final.

A diferencia de un juego convencional los serious games tienen historia, arte, software pero involucran actividades pedagógicas que instruyen para adquirir un conocimiento o una habilidad esta adición hace de un juego un serious game[3].

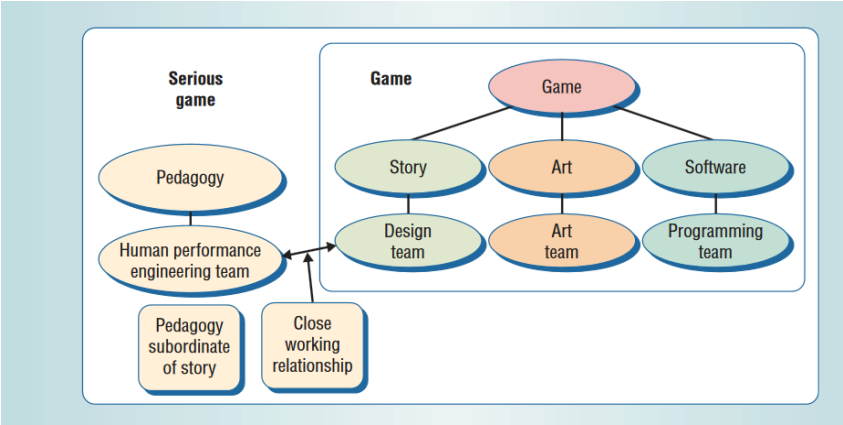


Figura 2.1 Del juego al serious game

2.2.-Serious games para el aprendizaje de música

Investigaciones recientes han demostrado que los videojuegos pueden ser usados en contextos educativos, los serious games también son usados para entretener a los usuarios aunque su objetivo principal es trasmitir un conocimiento o habilidad, en el mundo de la música las investigaciones e implementaciones no se limitan solo a las actividades prácticas para aprender a tocar un instrumento musical si no también se pueden ocupar de diversas habilidades musicales tales como: la composición musical, simulación de instrumentos, entrenamiento del oído etc[4].

Una de las tareas más aburridas y difíciles para quien estudia música es el reconocimiento de notas musicales pues es una tarea repetitiva, esta actividad se podría integrar en un serious games que tenga como objetivo pedagógico el reconocimiento de notas musicales de tal forma que en cada nivel aumente su dificultad, otro tema interesante que se puede abordar en un serious game es la notación musical ya que para algunas personas suele ser frustrante si no están familiarizados con conceptos matemáticos.

Para desarrollar un serious game efectivo se debe proveer al jugador una interfaz gráfica con una interactividad sencilla para abarcar a la mayor parte del público, los niveles deben estar diseñados de acuerdo al contexto de la habilidad o conocimiento que se va a trasmitir.

2.3.-Métodos para el diseño de serious games

En la tesis doctoral "Metodología para el diseño de videojuegos educativos sobre una arquitectura para el análisis del aprendizaje colaborativo" presentada por Natalia Padilla Zea en el capítulo "Procesos de Diseño para Videojuegos Educativos y Arquitecturas de Soporte"[5] se presentan una serie de metodologías que son consideradas por la autora como las mejores que pueden ser utilizadas para el diseño de un serious game, algunas de esas metodologías son las siguientes:

Metodología de EMERGO[6]

La metodología EMERGO (Efficent Method for Developing Experimental Education) está basada en el modelo ADDIE (Análisis, Diseño, Desarrollo, Implementación y Evaluación) el cual está enfocado a diseñar y crear material y contenido pedagógico, sin embargo EMERGO está adaptada para el desarrollo de serious games y está compuesta por cinco fases que se describen a continuación.

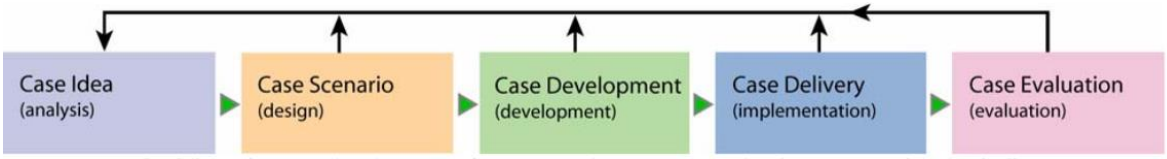


Figura 2.2 Fases de la metodología EMERGO

Análisis: En esta primera fase el equipo de desarrollo debe discutir diversas cuestiones relacionadas con el proyecto de tal forma que se tenga una visión clara y realista de que es lo que se quiere y que es lo que se va a poder llevar a cabo, para lograr dicha visión la metodología propone resolver un subconjunto adecuado de preguntas incluidas en la tabla 2.1.

Aspecto	Preguntas
Contexto	¿Para qué curso, currículo o instituciones se va usar? ¿Es un material aislado o se usará con otros materiales? ¿Qué carga de estudio e intervalo de tiempo se le suponen? ¿Cuántos créditos gana el estudiante cuando lo supera?
Contenido	¿Cuál es la principal habilidad cognitiva que se requiere? ¿Se necesitan otras habilidades? ¿Qué conocimiento previo y habilidades necesitan los estudiantes? ¿Qué es lo fundamental (por ejemplo: el paciente, el equipo, etc.)? ¿Cuáles son las localizaciones (por ejemplo: mapear espacios virtuales)? ¿Los estudiantes necesitan seguir un proceso establecido para realizarlo? ¿Qué tipo de actividades necesitan realizar los estudiantes para adquirir las habilidades cognitivas esperadas?

	¿Hay un orden estricto en las tareas obligatorias? ¿Hay tareas obligatorias y voluntarias y que determina que sean de un tipo o de otro? ¿La información suministrada es estrictamente necesaria o hay redundancias? ¿Cómo de realista y autentico es el caso? ¿Si los estudiantes pueden repetir un caso, será el mismo caso o habrá variaciones? ¿Los estudiantes pueden deshacer sus decisiones previas? ¿Hay diferentes rutas de aprendizaje y tareas diferentes para estudiantes? ¿Los estudiantes tienen diferentes características en sus casos? ¿Los estudiantes tienen roles activos? ¿Los profesores tienen roles activos? ¿Qué aspectos inducen y mantienen el interés y la motivación? ¿Qué circunstancias imprevistas se han incorporado? ¿Existe competición? ¿Cómo obtienen los estudiantes recompensas por su comportamiento?
Progreso de los estudiantes	¿Cómo descubren los estudiantes que todavía no han adquirido las habilidades cognitivas esperadas? ¿Cómo pueden los estudiantes monitorizar su progreso? ¿Cómo se comprueba que los estudiantes han adquirido las habilidades cognitivas esperadas? ¿Se ha incluido evaluación aditiva? ¿Se usa el resultado para la evaluación? ¿Qué cifras de progreso de los alumnos se van a usar para informar a los profesores durante la ejecución?
Contacto con compañeros	¿Se debe promover el contacto entre compañeros? ¿Deben ver los alumnos si sus compañeros están conectados? ¿Los estudiantes pueden comparar su progreso con los compañeros?
Uso de medios	Si se ha usado el material, ¿se necesita material nuevo? ¿Qué recursos se utilizan (entrevistas, películas, animaciones)?

Entrega	¿Qué medios se necesitan y cuál es su coste? ¿Está restringido el número de estudiantes en cada ejecución? ¿Cuándo puede un estudiante conectarse al juego? ¿Es posible cambiar el caso una vez se ha iniciado?
Soporte	¿Cómo se proporcionara el soporte técnico? ¿Cómo se proporcionara el soporte para la adquisición de habilidades cognitivas?
Coste	¿Cuál es el coste de desarrollo por estudiante? ¿Cuál es el ratio esperado de profesores/estudiantes durante la explotación?
Derechos de propiedad intelectual	¿Está permitido que otros utilicen el caso? ¿Hay materiales incorporados por otros autores y cuáles son sus derechos de estos materiales?

Tabla 2.1 Preguntas de la fase de Análisis (traducido de Nadolski, 2008)

Diseño: En esta fase se genera un documento de escenario detallado el cual describe las actividades exhaustivamente, la descripción de cada actividad se realiza distinguiendo de las actividades ordenadas y no ordenadas, obligatorias y no obligatorias, la descripción de cada actividad deberá responder preguntas como: ¿Qué hacen los estudiantes?, ¿Cómo van a actuar?, ¿Con quién interactúan?, ¿Qué recursos y herramientas necesitan?, y ¿Qué soporte utilizan?, ¿Las tareas resultan de un producto?, etc.

Desarrollo: En esta fase se usa el documento que se generó en la fase anterior a manera de guía para introducir la información en las herramientas que provee la metodología EMERGO.

Implementación: En esta fase los estudiantes y profesores podrán acceder al juego y deben cumplirse las siguientes suposiciones:

1) El estudiante puede elegir el juego desde su propio entorno
2) Un profesor puede elegir el juego desde su propio entorno
3) La entrada de datos del juego ha sido comprobada y es aceptable
4) El gestor de ejecución de juegos se ha usado para preparar el juego.

Si todo ocurre así y se tienen datos de autorización, los estudiantes y profesores pueden descargar los juegos desde la web de EMERGO.

Evaluación: La evaluación se mide con base en las demandas iniciales de la fase del análisis.

Metodología EDoS[7]

EDoS es un entorno para la creación de serious games, EDoS proporciona un método para crear serious games de una forma rápida, estructurada y formal, el entorno funciona como un mediador entre los expertos en el área de pedagogía y los desarrolladores con el fin que no existan confusiones que pudieran surgir durante el desarrollo.

Figura 2.3 Pestañas funcionales de EDoS

El entorno EDoS tiene seis fases de producción como se puede observar en la figura 2.3, la fase de diseño se inicia después de haber reunido los requerimientos del cliente, dentro de la fase de diseño se especifica la información de las subcategorías que se describirán a continuación.

Modelo de objetivos pedagógicos: Se definen los objetivos pedagógicos, un objetivo pedagógico responde a la pregunta ¿Qué conocimiento se desea trasmitir?, para la definición de dichos objetivos el entorno EDoS provee a los desarrolladores una herramienta que les sirve como guía.

Escenario pedagógico: En el escenario pedagógico se establecen las relaciones entre los objetos que va a tener el serious game y los objetivos pedagógicos, es decir, se debe establecer una relación entre las actividades y los recursos que se disponen con el fin de cumplir los objetivos que se propusieron, para ello se debe especificar qué tipo de serious game se va a emplear, todo lo anterior se debe hacer sin perder de vista que cada actividad tiene que tener su lado lúdico.

Descripción de pantallas y game-play: Este punto hace referencia a aspectos propios del juego, se detallan las actividades y se busca un mecanismo visual que vaya de la mano con las actividades propuestas por los docentes esto con el propósito de evitar malos entendidos con los desarrolladores.

Proceso de diseño en seis pasos de Marfisi-Schottman (2010)[8]

En el congreso europeo de aprendizaje basado en juegos realizado en 2010 (European Conference N game- Based Learning, EGBL 2010) se presentó una metodología basada en seis pasos para el diseño de serious games (Marfisi-Schottman, 2010). Los pasos que se proponen no tienen que seguirse en un orden estricto pues tienen como finalidad guiar a los desarrolladores en los proceso de diseño del serious game.

Paso 1) Al igual que en el método anterior se especifican los objetivos pedagógicos.

Paso 2) Elegir el modelo del juego, en este paso se elige el tipo de juego y se adapta al diseño del escenario pedagógico, ejemplos de tipos de juego: puzle, plataformas, rpg, etc.

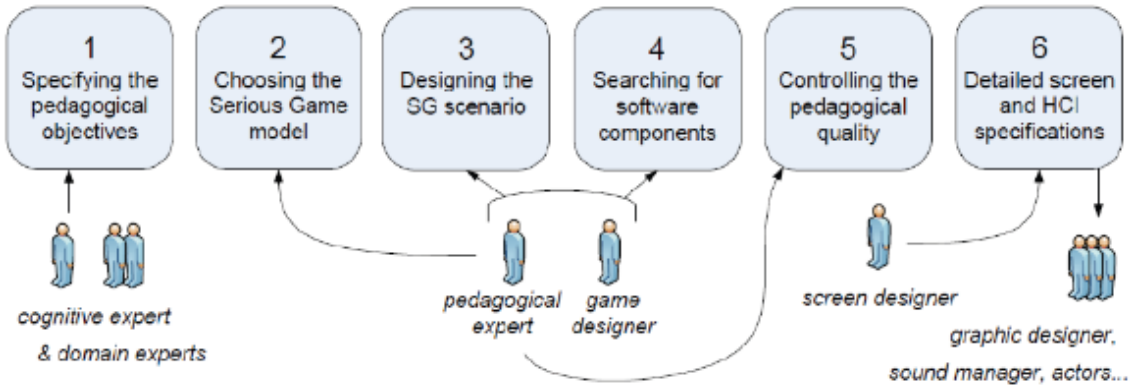


Figura 2.4 Seis pasos para el diseño de juegos serios (Marfisi-Schottman, 2010)

Paso 3) Diseño del escenario del serious game, en este paso se conjuntan los elementos multimedia (imágenes, animaciones, sonidos, etc.) del juego y los objetivos pedagógicos de forma tal que el balance entre la parte lúdica y de aprendizaje sea de una forma apropiada.

Paso 4) Búsqueda de componentes de software, en caso de que se tuvieran scripts o recursos multimedia se pueden volver a usar si encajan dentro del contexto del proyecto que se está desarrollando.

Paso 5) Controlar la calidad pedagógica, esta fase sirve como una especie de "testeo" para comprobar que si se está trasmitiendo conocimiento significativo al usuario, esto se hace con el fin de evitar el volver a diseñar elementos deficientes dentro del juego.

Paso 6) Detallar pantallas y especificaciones HCI (Human Computer Interactions), en este paso se ilustra cada escena detallando las interacciones que se desean incluir en el juego.

La metodología que se utilizó como guía para el desarrollo del sistema fue la propuesta de Marfisi-Schottman, ya que las metodologías EMERGO y EDoS están más enfocadas al diseño de juegos web y a un ambiente estudiantil mientras que la metodología de los seis pasos resulta más flexible en ciertos aspectos y es más fácil de comprender y aplicar.

Capítulo 3: Antecedentes

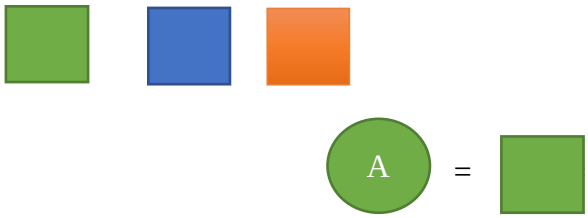
Al ser uno de los objetivos principales de este trabajo desarrollar un software con contenido teórico musical, lúdico y pedagógico, en este capítulo se darán a conocer algunos conceptos de programación, pedagógicos, y de teoría musical imprescindibles para entender mejor la naturaleza del proyecto.

3.1.-El método inductivo de enseñanza en la música

Para controlar el nivel de calidad pedagógica que va a tener el sistema final se propuso el uso del método inductivo de entre varios métodos de enseñanza ya que en el método inductivo el receptor juega un papel protagonista en la recepción del conocimiento, el aprendizaje inductivo se basa en proponer un reto del cual previamente se conoce poco para su resolución y una vez descubierta la forma de resolverlo se presenta otro reto similar que es posible de resolver usando un razonamiento similar al ejercicio anterior, si aplicamos el método inductivo para la enseñanza de escalas musicales resulta ser efectivo[9], pues al estar basado en el razonamiento inductivo se puede dar un enfoque práctico implementando actividades relacionadas entre sí en forma de retos pequeños con dificultad variable y cada reto forma parte de un conocimiento mayor, en pocas palabras lo que se busca es trasmitir conocimientos particulares que al unirlos generan un conocimiento más general, un ejemplo de lo anterior mencionado es el siguiente:

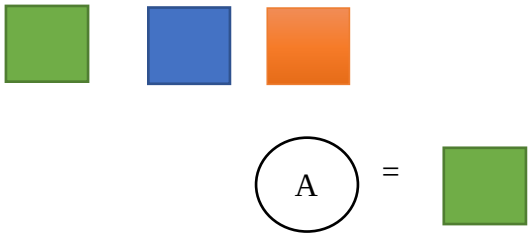
Ejercicio 1 (E1):

En este ejercicio la nota de La (A) es representada por un color entonces se tiene que escoger el color adecuadamente.



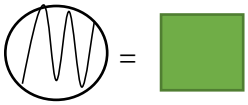
Ejercicio 2 (E2):

En este ejercicio se vuelve a presentar la nota de La (A) y se tiene que elegir el color que le corresponde, que sería el anterior aprendido.



Ahora imaginemos se le presenta en formato de audio (previamente se le presentaron los formatos textuales), se cometerán varios errores antes de dar con la respuesta correcta y cuando se elija correctamente se estará asociando un tono a un color y es así como se aprendió gráficamente, textualmente (notación) y auditivamente la nota de La, esto se pudo comprobar en la fase de pruebas con menores de edad quienes pudieron reconocer el audio que produce una nota y ligarla al color que se propuso para la representación de dicha nota (modo sin guitarra, para más información consultar apéndice B).

Ejercicio 3 (E3)



El aprendizaje que se obtuvo se puede representar como una secuencia de conocimientos particulares que se tuvieron que adquirir para llegar a uno general, Simbólicamente la relación entre los ejercicios se podría representar así:

0->E1 (en el primer ejercicio no se conocía a que el grafico verde era igual la nota de La)
E1->E2 (en el segundo ejercicio se llega a la conclusión que La es verde por que se aprendió previamente en el ejercicio 1)

Nota: la interpretación simbólica no representa proposiciones lógicas, solo sirve para ejemplificar como se relacionan los ejercicios.

Y es así que se pueden formular enseñanzas más complejas, por ejemplo escalas musicales, acordes, reconocimiento de notas, reconocimiento de acordes, patrones musicales etc. La creación del contenido pedagógico de este proyecto estará basado en este método.

3.2.- Conceptos básicos de teoría musical

Para tener una visión general de cómo se procesará el audio dentro del programa, debemos entender algunos conceptos de teoría musical, pues no se puede desarrollar ningún software de calidad sin entender los conceptos teóricos involucrados para su desarrollo, los conceptos más importantes tratados en este proyecto son: los acordes, las notas musicales, las escalas y la forma en que se relacionan entre sí pues se debe encontrar una representación adecuada para un lenguaje de programación.

3.2.1.- Las notas musicales

La mayoría de las melodías están hechas a partir de un conjunto de 12 notas musicales las cuales forman parte de la escala cromática, las notas poseen las siguientes características:

Tono: El tono de una nota describe qué tan alto o qué tan bajo se escucha dicha nota, el tono depende de la frecuencia de la nota, cuanto mayor sea la frecuencia de dicha nota mayor será el tono.

Duración: Describe la duración que suena una nota.

Timbre: El timbre es una característica que puede variar entre diferentes instrumentos musicales, por ejemplo, si alguien toca una flauta una nota de C (DO) y alguien una guitarra una nota de C, el oído humano puede distinguir entre los dos sonidos producidos aunque tengan el mismo tono.

Octava: El término octava se refiere a que una nota puede sonar con un tono diferente, pero sigue siendo la misma nota, esto quiere decir que una nota puede tener tonos más agudos o más graves.

Armónico: Un armónico es un múltiplo de la frecuencia fundamental de una nota, así por ejemplo la nota de A (LA), su segunda octava tiene una frecuencia de 110 Hz, sus respectivos armónicos serán 220Hz, 440 Hz, 880 Hz, etc.

3.2.2.-Las escalas musicales

Antes de empezar cabe mencionar que por simplicidad en todo el trabajo se empleará la notación americana para las notas musicales, en esta notación se usan las siete primeras letras del alfabeto para representar una nota musical: C=DO, C#=DO sostenido, D=RE, D#=RE sostenido, E=MI, F=FA, F#=FA sostenido, G=SOL, G#=SOL sostenido, A=LA, A#=LA sostenido, B=SI. También se debe entender el concepto de "grado", un grado es la posición en la que se encuentra una nota musical en una escala musical, por ejemplo la escala mayor de C el grado 1 de la escala mayor de C será la misma C, el grado 2 es D y así sucesivamente.

1	1# 2b	2	2# 3b	3	4	4# 5b	5	5# 6b	6	6# 7b	7	8
C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C

Tabla 3.1 Representación de la escala de "C"

Los grados que tienen de sufijo b representan que la nota tiene un semitono de diferencia con la siguiente nota y cuando tiene de sufijo # quiere decir que tiene un semitono de diferencia con la anterior, las notas que se encuentren en las casillas que tengan números con sufijo no pertenecerán a la escala mayor, también se pueden construir más escalas a partir de este concepto, por ejemplo, la escala mayor de D será:

1	1# 2b	2	2# 3b	3	4	4# 5b	5	5# 6b	6	6# 7b	7	8
D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D

Tabla 3.2 Representación de la escala de "D"

Para la construcción de las escalas convencionales se toma como referencia escala mayor de C aunque se pueden tomar de referencia las demás escalas pero ya estaríamos hablando de los "modos griegos". Las escalas y los modos griegos al igual que los acordes, son un subconjunto de notas de la escala cromática la diferencia radica en que para el caso de las escalas el subconjunto de notas no se tocan al mismo tiempo como es el caso de los acordes.

Existen varias escalas que se utilizan para componer una melodía, pero las más usuales son las mostradas en la tabla 3.3, las cuales están expresadas a partir de fórmulas basadas en la escala de "C" (ver tabla 3.1) los números de cada formula representan a los grados ordenados convenientemente.

Tipo de escala	Formula
Modo Jónico o Escala Mayor	1-2-3-5-6-7
Modo Dórico	1-2-3b-4-5-6-7b
Modo Frigio	1-2b-3b-4-5-6b-7b
Modo Lidio	1-2-3-5b-5-6b-7b
Modo Mixolidio	1-2-3-4-5-6-7b
Modo Eólico o Escala Menor	1-2-3b-4-5-6b-7b
Modo Locrio	1-2b-3b-4-5b-6b-7b
Pentatónica Menor	1-3b-4-5-7b
Pentatónica Mayor	1-2-3-5-6

Tabla 3.3 Formulas para generar escalas musicales

3.2.3.-Los acordes

Los acordes son un subconjunto de notas de la escala cromática que suenan al mismo tiempo, para armar un acorde se tiene que seguir ciertas reglas, la primer regla y más importante es saber qué tipo de acorde se quiere tocar pues hay varios tipos: mayores, menores, en séptima, en quinta, en cuarta o en novena y la segunda es escoger la nota tónica (nota principal del acorde su función es dar el tono adecuado) del acorde. A continuación se listan y describen los más importantes.

a) Acordes Mayores:

Un acorde mayor también se le conoce como triada ya que se compone de 3 notas las cuales están dadas por la fórmula: 1-3-5, si queremos hacer el acorde C tenemos que coger la escala mayor de C y tomar las notas que aparecen en los grados 1, 3, 5, es decir, C, E y G, observar que la primer nota es la que le da el tono al acorde, es decir la nota tónica es C y en consecuencia nuestro acorde es C mayor.

b) Acordes menores:

Los acordes menores son similares a los mayores pues también están compuestos de tres notas, la diferencia radica en el segundo grado, la fórmula es: 1-3b-5.

c) Acordes en quinta:

Las notas en quinta son los acordes con menor número de notas pues solo se componen de dos notas, la fórmula es: 1-5.

d) Acordes en séptima

Los acordes en séptima son una derivación de los acordes mayores la diferencia radica en que en este tipo de acordes se le agrega el grado 7b, la formula seria: 1-3-5-7b. En resumen cualquier acorde se puede generar un conjunto de fórmulas basadas en la escala de "C".

Tipo de acorde	Formula
Acorde mayor	1-3-5
Acorde menor	1-3b-5
Acorde en quinta	1-5
Acorde en séptima	1-3-5-7b

Tabla 3.4 fórmulas para la construcción de acordes

Las fórmulas que se obtuvieron son de gran importancia para este trabajo, ya que más adelante se aplicaran para desarrollar un reconocedor de audio polifónico.

3.3.- La relación entre la física y la música

Durante el proceso de diseño y desarrollo de la aplicación se tuvieron que tomar en cuenta ciertos conceptos de física para comprender mejor su implementación, así como para representar los datos de entrada y las actividades.

3.3.1.-La importancia del dominio de la frecuencia

El sonido es una perturbación u onda que se propaga por un medio y produce variaciones en la presión atmosférica, matemáticamente el movimiento de una onda se puede representar como:

$$y = A * sen(\omega t)$$

Donde:

$\boldsymbol{\omega} = 2\pi f$
$\boldsymbol{f}$ $(Frecuencia)$: Número de veces que se repite la vibración por unidad de tiempo
A (amplitud): La amplitud es la distancia que hay entre el valle y el punto medio

Obviamente expresar el tono de un sonido externo a un dispositivo sería demasiado complejo trabajarlo como ondas sinusoidales continuas pues se necesitaría más poder de procesamiento y una memoria muy grande, así como de un algoritmo complejo que calcule múltiples frecuencias involucradas en el sonido producido a lo largo del tiempo, por esta razón se prefiere trabajar en el dominio de la frecuencia y no del tiempo.

Cuando hablamos del dominio de la frecuencia es necesario hablar del concepto de la transformada de Fourier ya que permite transformar una función en el dominio del tiempo al dominio de la frecuencia.

En resumen el sonido físicamente se interpreta como una sumatoria de funciones sinusoidales de las cuales nos interesa únicamente su frecuencia la cual la podemos obtener haciendo uso de la transformada de Fourier, por lo que es menester analizar más a profundidad el comportamiento del sonido en términos de su frecuencia e intensidad ya que facilita muchos cálculos y cuestiones técnicas que se verán más adelante que corresponden algunos requerimientos funcionales del sistema.

3.3.2.- La transformada discreta de Fourier

Toda señal periódica sin importar que tan complicada sea, puede ser reconstruida a partir de sinusoides cuyas frecuencias son múltiplos enteros de una frecuencia fundamental, eligiendo las amplitudes y fases adecuadas. (Matemático francés Joseph Fourier,1768-1830).

La transformada de Fourier consiste en transformar una función periódica en una equivalente no periódica más fácil de analizar, una de las mayores ventajas de la transformada de Fourier es que al descomponer una señal periódica en varias sinusoides las cuales nos permite estudiar y clasificar sonidos en un plano espectral.

Para poder analizar la transformada discreta primero se introducirá la definición de la transformada continua de Fourier, la cual se puede definir como:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft}\, dt$$

t : tiempo
f: frecuencia en Hz
x(t): señal de prueba
$e^{-j2\pi ft}$: fasor de sondeo
X (f): espectro en función de la frecuencia f

Se puede observar que la formula anterior presupone que la función está en el dominio del tiempo y por lo tanto la función es continua, sin embargo para el análisis de una señal en un ordenador es imposible procesar una onda continua pues se requeriría de una memoria infinita, por eso se usa la transformada discreta de Fourier.

El equivalente de la transformada anterior en tiempo discreto se expresa de la siguiente manera:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-2pij}{N} kn}$$

N: Número de muestras en x[n]
x[n]: señal de prueba discreta con un índice n
X[k]: Espectro en función de la frecuencia discreta
$e^{-\frac{2\pi j}{N} kn}$: Fusor de sondeo discreto

Lo que nos da como resultado la ecuación de arriba son muestras que se toman cada cierto tiempo y se van añadiendo en un arreglo X[] de tamaño N.

Capítulo 4: Diseño del sistema

Antes de empezar a programar cada una de las partes que conforman a todo sistema en este capítulo se presentará una visión general del diseño de cada subsistema así como del sistema completo, se aplicará la metodología de los seis pasos de Marfisi Schottman para la creación y unión del contenido pedagógico y las mecánicas del juego, también se expondrá y describirá la arquitectura que se empleara para el desarrollo del sistema.

4.1.- Requerimientos del sistema

A continuación se listan los requerimientos del sistema basados en la naturaleza del sistema y en la arquitectura Educational Game Architecture (Hu, 2010).

4.1.1.- Requerimientos Funcionales

Id requisito	**Nombre del requisito**	**Descripción del requisito**
RF-01	Registrar el inicio de partida	El sistema permitirá a cualquier usuario poder registrar una partida nueva (hasta 3 perfiles diferentes)
RF-02	Guardar parámetros y estado de partida	Cuando se inicia una nueva partida y se termina el sistema deberá guardar los parámetros alterados durante la partida
RF-03	Borrar datos de partida	Si un usuario desea borrar el progreso de su partida el sistema deberá eliminar todo dato referente al progreso de su partida
RF-04	Leer micrófono	El sistema deberá ser capaz de leer y procesar el audio proveniente del micrófono
RF-05	Ajuste de parámetros	Con base en ciertos criterios provenientes del requerimiento RF-02 deberá actualizar sus parámetros internos para proponer al usuario una nueva actividad basada en el progreso
RF-06	Procesamiento de diferentes sonidos	El sistema será capaz de categorizar los sonidos provenientes de una guitarra acústica, es decir, mostrará en pantalla si se trata de un acorde ya sea acorde Mayor,

		menor, quinta o séptima y una nota individual
RF-07	Pausar partida	Si el usuario así lo desea, podrá pausar el estado de su partida
RF-08	Salir de aplicación	Existirá un botón de salir en el menú de pausa o en el menú principal
RF-09	Representación del contenido	Se incluirá el contenido teórico dentro de la lógica del juego mediante elementos metafóricos, tutoriales, animaciones y gráficos
RF-10	Actividades	Se incluirán una serie de actividades interactivas relacionadas con el sistema de juego y las actividades propuestas
RF-11	Realimentación	Existirá un módulo de realimentación simplificado que muestre información relevante durante la ejecución de cada actividad
RF-12	Sistema de puntación	Se incluirá un módulo de recompensa y castigo dependiendo de la efectividad de ejecución de cada actividad
RF-13	Tutoriales	Existirán tutoriales que guíen al usuario al inicio del sistema
RF-14	Monitorización	Deberá incluirse un módulo de monitorización que observe como se resuelven las actividades

Tabla 4.1 Requerimientos funcionales

4.1.2.- Requerimientos no funcionales

Id requisito	Descripción del requisito
RNF-01	Los gráficos deberán ser homogéneos, es decir deben de estar dentro del contexto del estilo visual que se va a emplear
RNF-02	La interfaz tendrá botones y ventanas que se despliegan dentro de la misma ventana donde está corriendo el sistema global

RNF-03	El programa deberá ser desarrollado mediante un motor gráfico que utilice lenguaje orientado a objetos capaz de procesar audio digital proveniente del micrófono de cualquier dispositivo
RNF-04	La interfaz gráfica y tutoriales solo deberá contener la información más relevante de alguna función del sistema
RNF-05	En cada actividad debe existir un nivel de reto mayor al anterior
RNF-06	El sistema final deberá ser capaz de inmergir al usuario
RNF-07	El sistema despertara la fascinación del usuario final

Tabla 4.2 Requerimientos no funcionales

4.2.-Arquitectura del sistema

La arquitectura presentada en este proyecto se diseñó a base de la arquitectura Educational Game Architecture (EGA) y la arquitectura de tres capas (a esta última se le agrego la capa de núcleo), se decidió usar una arquitectura propia ya que la arquitectura EGA tiene más módulos de los que se requerían para este sistema. En la arquitectura propuesta se unen los subsistemas actividades y tareas por el subsistema "Gestor de actividades", el subsistema nombrado como "Lógica del serious game" es la sustitución del subsistema "Contenido" de la arquitectura EGA, se agregó un subsistema de "Monitorización" que está relacionado con el subsistema de "Presentación" a su vez este subsistema tendrá incluido el subsistema llamado "Realimentaciones" en la arquitectura EGA y por último se agruparon los subsistemas en cuatro capas: datos, núcleo, contenido y presentación, esto con el fin de agrupar a un nivel alto de abstracción con base a las funciones que cumplen cada uno de los subsistemas.

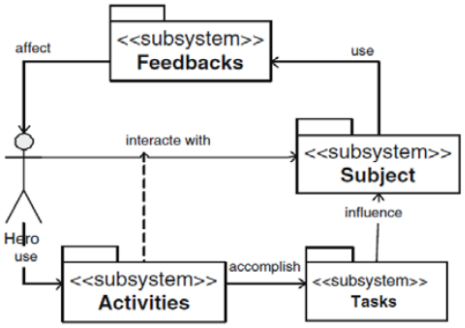


Figura 4.1 Relaciones entre los sub-sistemas de la arquitectura EGA (Hu, 2010)

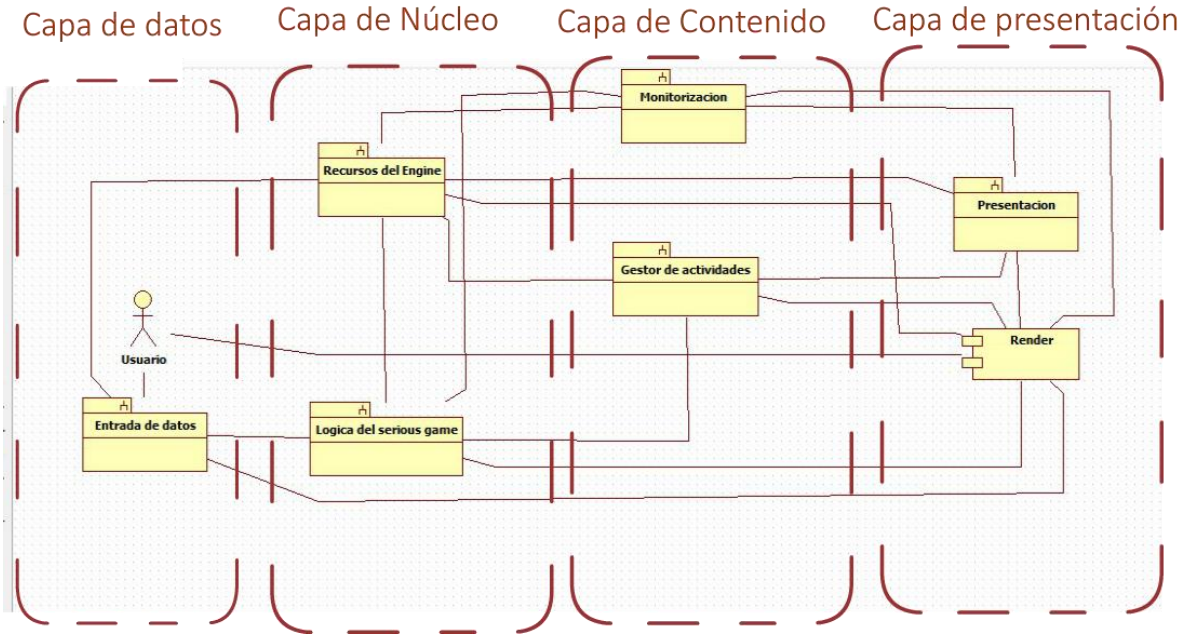


Figura 4.2 Arquitectura del sistema

En la figura 4.2 se ilustra la arquitectura que se usó para desarrollar el sistema a continuación se describe cada uno de sus módulos.

Subsistema entrada de datos: Este subsistema está directamente relacionado con las acciones que puede hacer el usuario respetando a las reglas del serious game y cumple la función de procesar el audio del exterior y representarlo de forma legible para el sistema y mandar dicha información al subsistema "Lógica del serious game", también estará encargado de manejar las funciones cuando se hace clic sobre algún botón que está relacionado directamente con las mecánicas del juego.

Recursos del Engine: Son todos los recursos que nos ofrece el motor gráfico como lo pueden ser: manejo de los recursos de audio, físicas, renderizado de gráficos en 2d y 3d, animaciones etc. Está relacionado con todos los demás subsistemas pues las clases pertenecientes a dichos subsistemas hacen uso de uno o más recursos del motor gráfico.

Subsistema de la lógica del juego: Es el conjunto de objetos gráficos que se controlan mediante scripts y cumplen las reglas propuestas por los diseñadores del sistema de juego a su vez dichas reglas están relacionadas con las actividades es por ese motivo que este subsistema está conectado con el subsistema gestor de actividades.

Subsistema de Monitorización: Es el encargado de llevar un registro de ciertos eventos que acontecen en el subsistema lógica del juego, también está relacionado con el subsistema de presentación ya que este último inicializa sus parámetros.

Subsistema Gestor de actividades: Es el encargado de ejecutar las actividades que va a hacer el usuario con una dificultad adecuada, así como de mantener una comunicación con

la lógica del juego pues deberá mantener una relación con los objetos en la escena y las reglas propuestas por los diseñadores.

Subsistema de Presentación: Es el encargado de mostrar las interfaces de usuario con una resolución adecuada en la pantalla del dispositivo donde se ejecutara la aplicación, así como de almacenar, cargar, mostrar y modificar parámetros que permitan ofrecerle al usuario una realimentación relevante según lo que observo el subsistema de monitorización durante cada partida.

Render: Es la aplicación ejecutable que conjunta a todos los subsistemas y su principal interfaz de salida es el monitor del dispositivo donde se ejecuta.

4.3.- Game design

Se decidió incluir este tema ya que el game design se trata de dar idea general a una persona ajena al proyecto y también es una forma simple de documentar el proyecto en términos de las mecánicas, reglas, historia, arte y gameplay. La documentación del game design normalmente se hace a través de un documento llamado game design document (GDD) y sirve como guía a los desarrolladores.

No existe una regla estricta ni una metodología para el uso o desarrollo del GDD, así que en este trabajo se usara como una guía para resaltar y describir algunos atributos importantes del sistema (funciones, mercancías, genero etc.), a continuación se presentan algunos segmentos más usuales en los GDD con el fin de ejemplificar y mostrar las características principales de la aplicación.

1.- Requerimientos del sistema:

Plataforma: PC y móvil
Engine: Unity 3d
Género y objetivo del juego: Tower Defense, el objetivo principal es impedir que las unidades enemigas ataquen a tu torre.

Características principales:

a) Capacidad de reconocer notas y acordes de una guitarra acústica.
b) Arte gráfico armónico.
c) Mecánicas inusuales.
d) Dos modalidades de juego: con y sin guitarra.

Requerimientos para ejecutar la aplicación[10]:

- Escritorio:

- Sistema operativo: Windows XP SP2+, Ubuntu 12.04+
- Tarjeta gráfica: DX9 (modelo de shader 3.0) o DX11 con capacidades de funciones de nivel 9.3.

- CPU: compatible con el conjunto de instrucciones SSE2.

- Android: OS 2.3.1 o posterior; CPU ARMv7 (Cortex) con tecnología NEON o CPU Atom; OpenGL ES 2.0 o posterior.

2.- Especificación funcional:

a) Mecánicas:

Las mecánicas (reglas del juego) están basadas en torno al género del juego (Tower Defense), también están basadas en los objetivos pedagógicos (capitulo 3). Las principales mecánicas son las siguientes:

a) Emisión de onda según el tono: Dependiendo del tono (resultado del subsistema entrada de datos) se genera un objeto tipo "Onda" de un color particular que al colisionar con una unidad enemiga del mismo color la podrá desactivar.

b) Disparo parabólico: Cuando se han ganado 300 puntos de experiencia se desbloquea esta acción, la cual consiste en la capacidad de disparar proyectiles que describen una trayectoria parabólica que al colisionar con una unidad enemiga de cualquier color lo podrá desactivar.

c) Disparo dirigido: Cuando se ganaron 500 puntos de experiencia se desbloquea esta mecánica, consiste en lo mismo que la anterior con la diferencia que la trayectoria que describe depende a la posición a la unidad a desactivar.

d) Disparo buscador: Cuando se ganaron 800 puntos de experiencia se podrá desbloquear esta mecánica, es parecida a la anterior con la diferencia que es más precisa al buscar la posición a la unidad a desactivar.

3.-Interfaz de usuario:

La interfaz de usuario incluida en el sistema se puede escalar a todas las resoluciones de los diferentes dispositivos y muestra la información relacionada con el perfil del usuario (ver figura 4.3), la información que es relevante a la partida se mostrara sobre los objetos que van a interaccionar en la escena (ver figura 4.4).

Figura 4.3 Interfaz perfil de usuario

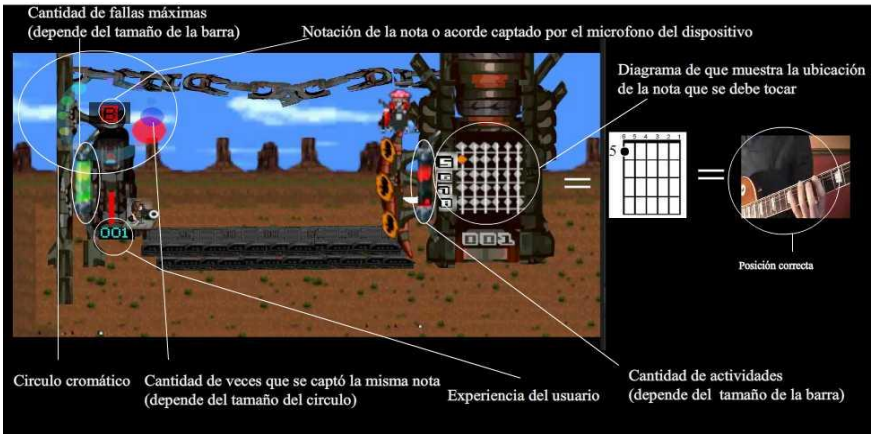


Figura 4.4 Información relevante de la partida

4.-Arte conceptual:

El arte conceptual va acorde a la metáfora incluida en la aplicación, el arte está inspirado principalmente en comics de ciencia ficción, ilustraciones medievales y en paisajes desérticos de México.

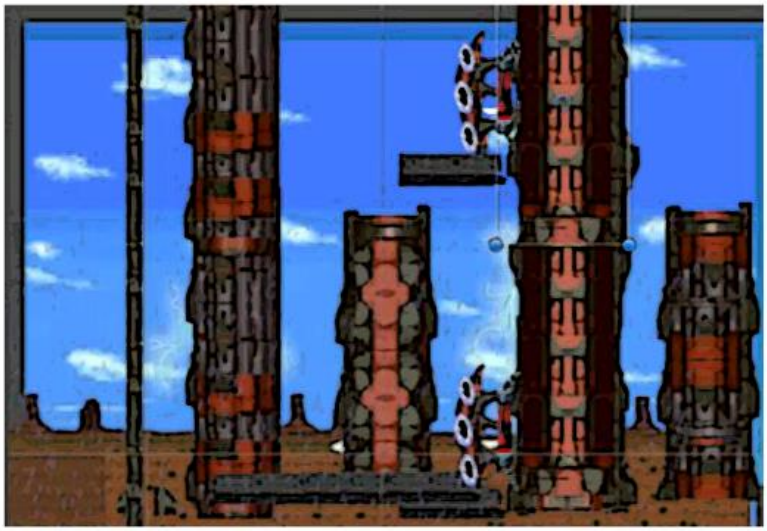


Figura 4.5 Arte conceptual

5.-Historia:

*Es el año 7000 después de la caída del O.Z, en la tierra habitan "**Echo-walkers**" (monstruos) y seres humanos sobrevivientes, las leyendas cuentan que en el año 2040 la **torre diapasón*** ubicada *en "El yermo de las sombras" es la responsable de la creación de los Echo-walkers, pues se alimentó de las **frecuencias** resonantes de la naturaleza y engendro a dichos monstruos. El **ingeniero** Screw G. ha inventado una maquina capaz de **reinterpretar los tonos** de la "**Guitar**-Gear" y convertirlas en **ondas poderosas** capaces de corromper a los monstros generados por la torre, ¿Será capaz de regresarle a la naturaleza la armonía que reinaba en los tiempos antiguos?, **el destino está en tus manos**.*

6.-Pantallas del juego a nivel usuario y tareas que se realizaron para la generación del prototipo final:

Con motivos de ilustrar mejor la navegación entre las pantallas y las tareas que se tuvieron que hacer para llegar a un prototipo que tuviera una visión general del sistema final se decidió agregar el apéndice A y B donde se muestran aspectos importantes del desarrollo del prototipo y el link de un video de la última compilación del sistema donde se muestran aspectos importantes del gameplay entre otras cuestiones relevantes.

4.4.-Diseño del subsistema de presentación

El subsistema de presentación deberá cubrir las funciones de: mostrar, almacenar, cargar y modificar parámetros relevantes de un perfil de usuario, para tener una mejor idea de cómo estará organizado y relacionado el subsistema presentación ver la figura 4.6.

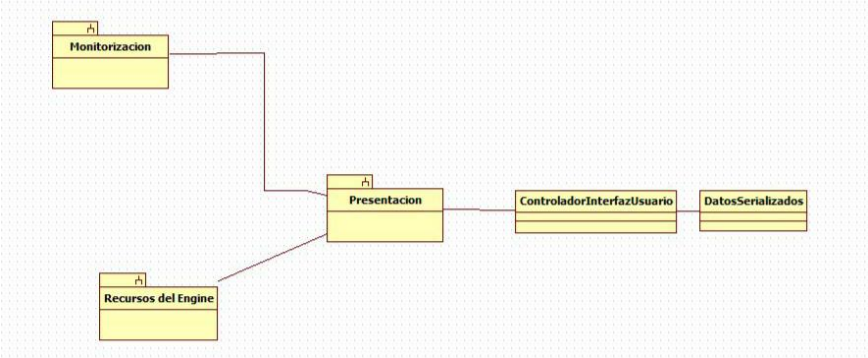


Figura 4.6 Relaciones del subsistema “Presentación”

Como se puede observar en la figura 4.6, este subsistema solo está formado de dos clases pero en este tema solo nos ocuparemos de analizar el cómo se podría diseñar una clase que controle la interfaz de usuario, la información que se muestra en la interfaz de usuario es dependiente a su perfil y de acuerdo al requerimiento RF-01 (ver tabla 4.1) se desea almacenar hasta tres perfiles diferentes los cuales tendrán la siguiente información:

a) Nombre del perfil.
b) Experiencia del usuario.
c) Numero de partidas.
d) Calificaciones o Rankings.
e) Nivel actual.
f) Partidas perdidas.
g) ¿El usuario tiene guitarra?

Las principales funciones serán:

a) Crear perfil
b) Cargar perfil
c) Guardar perfil
d) Borrar perfil
e) Actualizar datos
f) Mostrar resultados
g) Mostrar estadísticas
h) Pedir nombre del perfil
i) Pedir información si se tiene guitarra o no
j) Mostrar mensajes en pantalla (como “Start”, “You Win”, “You Lose”, etc.)

Resumiendo, el subsistema de presentación es uno de los subsistemas que resulto más fácil de diseñar ya que comprende funciones básicas de control de información y mostrarlas en pantalla, sin embargo se debe tener cuidado en cómo se implementa el guardado y el cargado de datos pues puede que los datos sean inconsistentes entre las distintas pantallas, por experiencia propia suele pasar que no se contemplen todas las posibles condiciones para cargar, mostrar o guardar datos, por eso antes de implementar este subsistema se hizo un diagrama que muestra las diferentes pantallas que componen a la aplicación y que funciones se ejecutan según que botón se presione (figura 4.7).

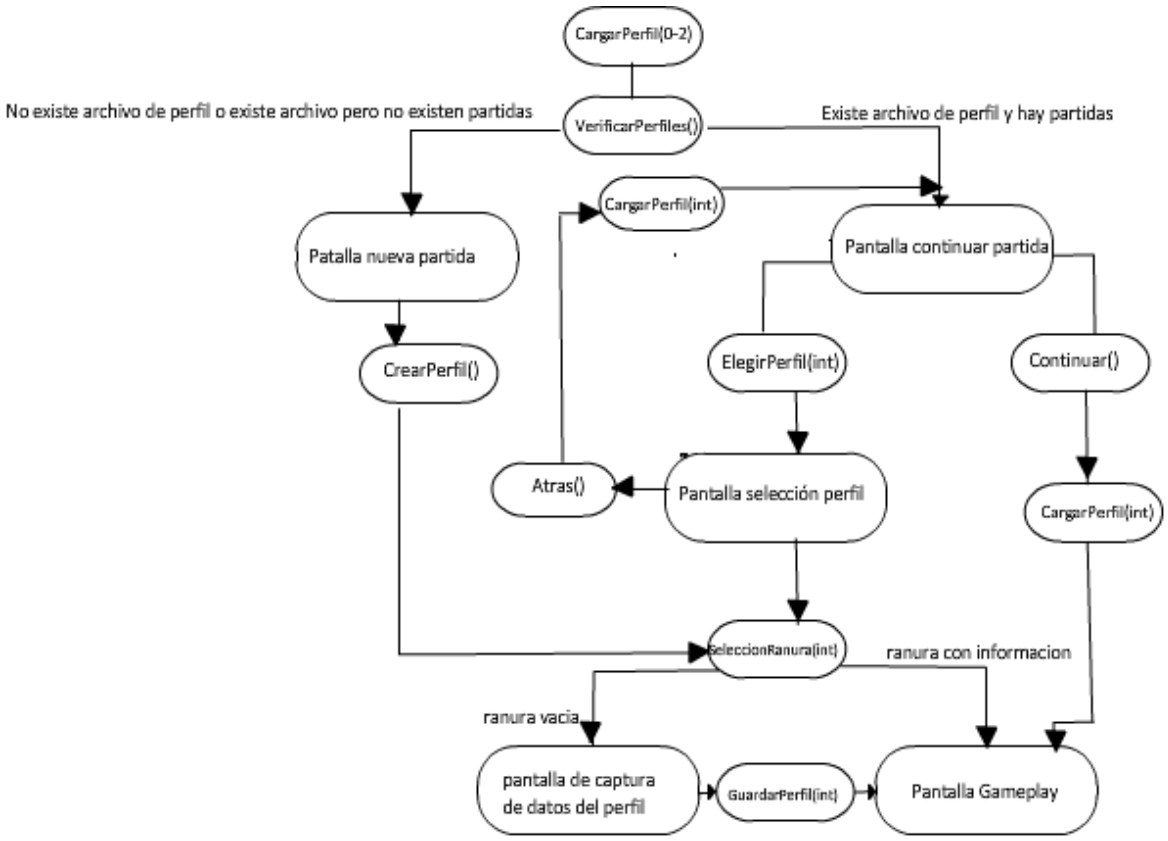


Figura 4.7 Mapa de navegación

Se recomienda ver el apéndice B para la visualización de las distintas pantallas que componen todo el sistema.

Nota: La función ElegirPerfil se manda a llamar cuando se presiona botón "seleccionar perfil" esto provoca que se activen los objetos "ranura" para que el usuario cambie de perfil o genere uno nuevo y la función CargarPerfil carga los datos según el argumento de la función.

4.5.- Diseño del subsistema Entrada de datos

El subsistema ”Entrada de datos” estará encargado del procesamiento del audio externo al programa y transcribirlos a forma gráfica, si el usuario no tiene guitarra se encargara de llamadas a funciones a través de botones de la interfaz de usuario.

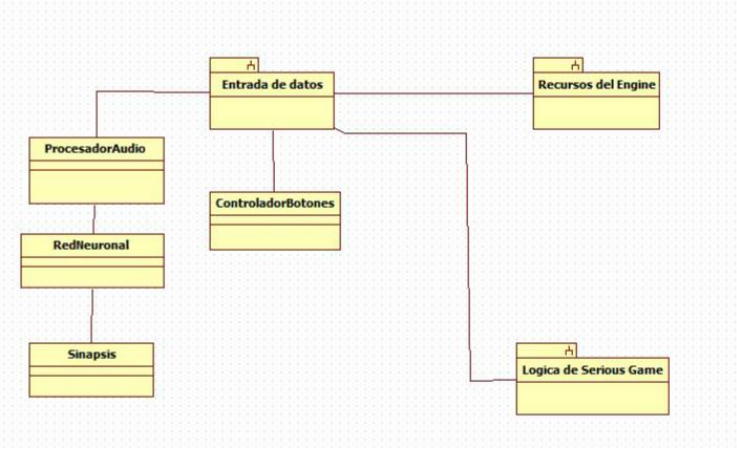


Figura 4.8 Relaciones del subsistema “Entrada de datos”

Como se puede observar en la figura 4.8 el subsistema está constituido por cuatro clases, en este tema solo se describirá el procedimiento para el diseño de la red neuronal ya que las clases que se relacionan con esta surgen a partir de los elementos que constituyen a la red neuronal, se dejará de lado el controlador de botones ya que aunque es importante no requiere de ningún análisis tan exhaustivo para su implementación.

4.5.1.- Análisis del espectro de frecuencias del mástil

De acuerdo al RF-06 (ver tabla 4.1) se requiere de un módulo que sea capaz de procesar el audio externo y lo convierta a un tipo de dato legible para una computadora y poderlo representar gráficamente y textualmente, antes de implementarlo en el motor gráfico, se tuvo que investigar varias alternativas para llegar a la selección final del algoritmo que cumpliera con el requerimiento funcional, también se hizo un análisis de las frecuencias del mástil de una guitarra acústica.

Figura 4.9 Las octavas en un mástil de una guitarra acústica

Como se estudió en el capítulo 3 el tono de un sonido se puede expresar en términos de su frecuencia, la obtención de la frecuencia de una nota específica se puede hacer matemática mediante la siguiente formula:

$$f(n,o) = 440 * 2^{(o-4)+\frac{(n-10)}{12}}$$

Donde **n** representa el número de nota.

1	2	3	4	5	6	7	8	9	10	11	12
C	C#	D	D#	E	F	F#	G	G#	A	A#	B

Tabla 4.3 Notas musicales con número

Y la letra **o** se refiere al número de octava, a continuación se muestra una la tabla 4.4 con las frecuencias medidas en Hz de las 12 notas (la ecuación se formuló tomado como referencia la frecuencia natural de A de su cuarta octava, si sustituimos n=10 y o=4 da como resultado $440*2^0 = 440$)[20].

Octava	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
1	33	35	37	39	41	44	46	49	52	55	58	62
2	65	69	73	78	82	87	92	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	831	880	932	988

Tabla 4.4 Frecuencias de las notas musicales y sus octavas

Si en la figura 4.9 en vez de ponerle notas (C2, A2, B3, etc.) las sustituimos por las frecuencias de la tabla 4.10, obtendríamos algo así:

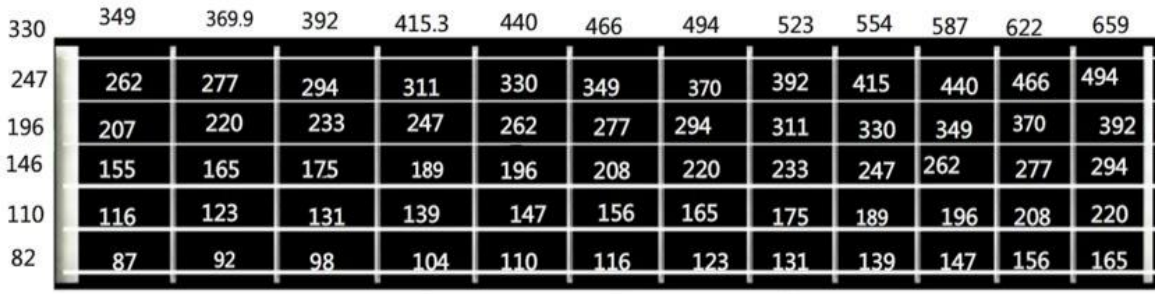


Figura 4.10 El mástil con las frecuencias de sus octavas

Nota: todas las cifras mostradas en la tabla 4.4 y la figura 4.10 tienen unidad de Hz y las frecuencias **están redondeadas** (si el decimal es mayor a 0.5 se redondea a la siguiente cifra).

Como se vio en el capítulo 3, al aplicar la DFT (transformada discreta de Fourier) se obtenía un arreglo de tamaño igual al número de muestras entonces representación de los

sonidos podría hacerse con un arreglo de tamaño igual a las muestras que contengan a la frecuencia más grande del mástil de una guitarra convencional.

Por ejemplo:

Sea el arreglo **f** de tamaño 660 con los siguientes valores almacenados en las posiciones mostradas en la tabla 3.3:

0-164	165	166-219	220	220-276	277	278-659
0	1	0	1	0	1	0

Tabla 4.5 valores del arreglo f

Gráficamente se representaría como:

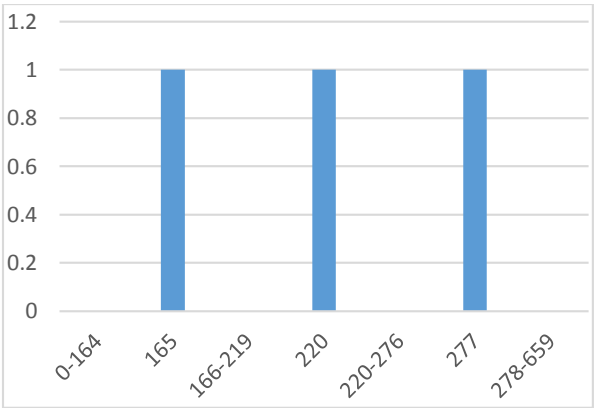


Figura 4.11 Espectro de frecuencia del acorde A

Las frecuencias de la tabla 4.5 que tienen valor de 1 corresponden a las octavas A3, E3, C4# que se obtuvieron al aplicar la fórmula para acordes mayores (ver tabla 3.4), es decir estamos hablado del espectro de frecuencias del acorde ideal "A mayor", le llamamos ideal ya que los valores de las demás frecuencias rara vez son 0 y se usaron octavas para su construcción con fines ilustrativos, si repetimos el mismo procedimiento para construir el espectro de cualquier acorde usando las formulas de la tabla 3.4 se obtendrán un conjunto de histogramas que es lo mismo a tener un conjunto de valores diferentes a través del tiempo en un arreglo de tamaño igual o mayor al número de muestras que se deseé.

4.5.2.- Diseño de la red neuronal reconocedora de notas y acordes, basándose en el algoritmo PCP (Pitch Class Profiles)

Cabe mencionar que para programar el algoritmo PCP previamente a la función del audio a analizar se le debió aplicar la transformada discreta de Fourier. Antes de explicar cómo se diseñó la red neuronal que se utilizó para desarrollar el reconocedor de notas y acordes, se ha de explicar a profundidad el algoritmo PCP.

Del ejemplo expuesto anteriormente del arreglo **f** se puede decir que el espectro de frecuencias se puede modelar como si fuera un vector que es el que se obtiene al aplicar la DFT, que en el caso del proyecto es de dimensión 4096 pues aunque la frecuencia más grande de una guitarra es 659 en el motor gráfico (Unity 3d versión 4) donde se desarrolló la aplicación la frecuencia que representa cada índice está dada por la formula **índice*24000/muestras** (donde 24000 es la mitad de la frecuencia de muestreo de un ordenador y **muestras** es el tamaño del arreglo y debe ser un múltiplo de dos en este caso es $2^{12} = 4096$), en pocas palabras el resultado de la formula sin la variable **índice** es la resolución, en el caso de este trabajo caso es de 5.85 es decir entre el índice 0 del arreglo y el 1 hay una diferencia de 5.85 Hz, el número de muestras se eligió después de haber hecho varias pruebas balanceando la rapidez del cálculo y la exactitud (tomando de referencia la frecuencia natural de la nota A (440 Hz)), una vez aclarado lo anterior al tener un arreglo de tamaño 4096 y se hubiera diseñado una red neuronal basándose en esa dimensión tendrían que haber por lo menos 4096 neuronas en la capa de entrada y cada una conectada a 12 neuronas de la capa intermedia (que corresponden a las 12 notas de la escala cromática) lo que se convierte en 4096*12=49152 conexiones, lo que se traduce a tener una matriz de 4096x12 que alberga 49152 pesos sinápticos, lo cual resultaría perjudicial para el programa, es por ello que se tuvo que aplicar el algoritmo PCP previamente al diseño de la red neuronal.

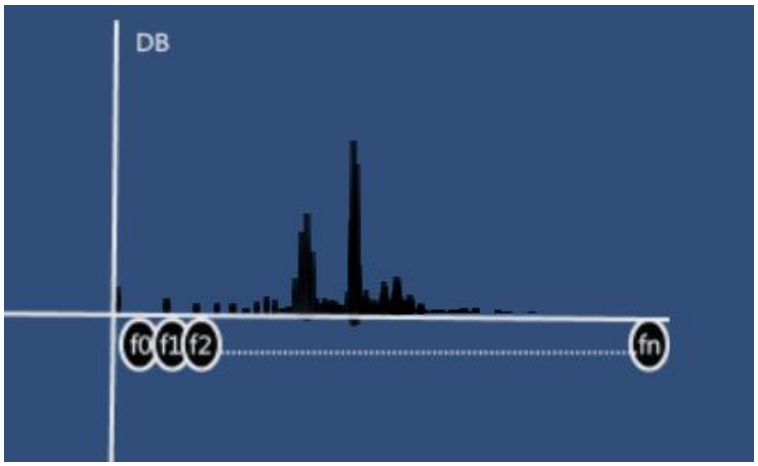


Figura 4.12 Espectro de frecuencias

Una vez entendido el porque la dimensión del arreglo y todos los inconvenientes de diseñar una red neuronal debido a su gran tamaño, la conclusión a la que se llego fue que se debía reducir el tamaño de datos de entrada pero sin que afectara el resultado final, si recordamos el concepto de "armónico" que se expuso en el tema 3.3 y observamos la tabla 4.4, nos damos cuenta que las octavas de una nota son múltiplos de dos de la octava anterior, es decir las octavas son los armónicos de una frecuencia fundamental (dicha frecuencia es la de una octava misma) y aunque tengan diferente tono pertenecen a la misma nota, entonces se puede expresar la intensidad sonora de cada notas en términos de la sumatoria de las frecuencias de las octavas es decir:

$$C = C1 + C2 + C3 + C4 + C5$$

$$C\# = C\#1 + C\#2 + C\#3 + C\#4 + C\#5$$

$$D = D1 + D2 + D3 + D4 + D5$$

$$D\# = D\#1 + D\#2 + D\#3 + D\#4 + D\#5$$

$$E = E1 + E2 + E3 + E4 + E5$$

$$F = F1 + F2 + F3 + F4 + F5$$

$$F\# = F\#1 + F\#2 + F\#3 + F\#4 + F\#5$$

$$G = G1 + G2 + G3 + G4 + G5$$

$$G\# = G\#1 + G\#2 + G\#3 + G\#4 + G\#5$$

$$A = A1 + A2 + A3 + A4 + A5$$

$$A\# = A\#1 + A\#2 + A\#3 + A\#4 + A\#5$$

$$B = B1 + B2 + B3 + B4 + B5$$

De esa forma se reduce la dimensión del espacio vectorial del espectro de frecuencias que inicialmente era de una dimensión de 4096, se reduce a un espacio vectorial de dimensión 12, a esa suma de intensidades se le conoce como el algoritmo del "Pitch Class Profile" (PCP) ,existen otros algoritmos para resolver este problema y problemas similares para reducir el tamaño de vectores, el más interesante es el de "Análisis de componentes principales" (PCA), el objetivo de dicho algoritmo es reducir una serie de variables X={x1,x2,x3,x4…,xn} a un subconjunto C={c1,c2,…cn} más pequeño que el conjunto X, esto también se puede ver desde la perspectiva de algebra lineal como "matriz de transformación", pero se escogió el algoritmo del PCP porque está más enfocado a música que es lo que interesa a este trabajo.

El conjunto de variables resultantes al aplicar el algoritmo PCP se les aplica una ecuación matemática definida como:

$$PCP(p) = \sum_{M(l)=p} ||X(l)||^2$$

$$p = 0 \dots \dots 11$$

M(l) es una tabla que mapea una espiga del espectro de frecuencias al índice del vector PCP, esta tabla puede ser inicializada como:

$$M(l) = \begin{cases} -1 & l \leq 0 \\ redondea(12log_2(f_s * \frac{1}{N})/f_{ref}) & l > 0 \dots . N/2 - 1 \end{cases}$$

La función $redondea(12log_2(f_s * \frac{1}{N})/f_{ref})$ es el resultado de despejar la variable **n** de la ecuación de la página 33 (pero está más generalizada ya que no tiene el parámetro **o**), así es como se define matemáticamente el algoritmo, sin embargo esta definición resulta ser un tanto compleja e ilegible para un programador, así que después de analizar las formulas anteriores se llegó a la conclusión que es una simple sumatoria de la intensidad de las espigas asociadas a las octavas de una nota, por eso **p** varia de 0 a 11 (son 12 notas) y **M(l)** es el índice de **p**, es decir, la letra **l** representa una frecuencia determinada obtenida de la transformada discreta de Fourier, **X(l)** es el valor en decibeles de dicha frecuencia.

Como vimos anteriormente, el resultado del algoritmo PCP son vectores de dimensión 12, que a su vez pueden programarse como arreglos de dimensión 12, el valor de las componentes de estos vectores varían en función de la entrada. Por ejemplo el vector del acorde C idealmente es el siguiente:

$$v_C = \{1,0,0,0,1,0,0,1,0,0,0,0\)$$

Pero como en la realidad nada es ideal otros vectores que representa al mismo acorde podrían ser:

$$v_C = \{0.4,0,0,0,0.6,0,0,1,0,0,0,0\)$$

$$v_C = \{0.7,0,0,0,1,0,0,0.6,0,0,0,0\)$$

Y gráficamente se podrían ver como un histograma:

Figura 4.13 Graficas de los vectores del acorde C

Para un ser humano conocedor del tema y que ha leído esta tesis resulta muy sencillo reconocer de que cualquiera de las gráficas mostradas en la figura 4.13 corresponde al acorde C, pero para una computadora no lo es, si nos detenemos y analizamos el proceso que ocurre en nuestro cerebro podemos dar cuenta que con base en un conocimiento previo de la teoría musical se podría percatar de que existe un patrón en las gráficas de la figura 4.3 y se hace una conjetura para saber de qué acorde se podría tratar según un histograma dado, entonces a partir de esta reflexión se pudo reconocer que algoritmo se podía usar para que la computadora pudiera reconocer de que acorde o nota se trata, la conclusión a la que se llego fue el uso de una red neuronal competitiva.

La red neuronal competitiva usará el algoritmo LVQ para su entrenamiento, en este algoritmo se tienen que definir los vectores de entrada que se van a usar y como se van a clasificar los vectores a la salida de la capa oculta.

Para esta red neuronal se necesitarán: vectores de clasificación, vectores prototipos, matriz conformada por los vectores prototipos y una segunda matriz que estará conformada por los vectores de clasificación, así como el vector de entrada.

Para ser más precisos se describirá la notación a continuación:

$$\overline{c1}, \overline{c2}, etc - vectores\ de\ clasificacion$$
$$\overline{w1}, \overline{w2}, etc - vectores\ prototipos$$
$$W^1 - matriz\ con\ los\ vectores\ prototipos$$
$$W^2 - mariz\ con\ los\ vectores\ de\ clasificacion$$
$$\bar{v} - vector\ de\ entrada$$

El reto para diseñar este tipo de redes neuronales es: ¿Cómo se van a definir los vectores?, pues ya sabiendo los requerimientos se debe elegir una representación apropiada, en nuestro caso se decidió que la red neuronal solo va a reconocer notas individuales, acordes mayores, menores, en quinta y en séptima, entonces una vez teniendo esa información se decidió que los vectores de clasificación para la red neuronal fueran los siguientes:

$$\overline{c1} = (1,0,0,0,0)\ vector\ clase\ notas$$
$$\overline{c2} = (0,1,0,0,0)\ vector\ clase\ acorde\ mayor$$
$$\overline{c3} = (0,0,1,0,0)\ vector\ clase\ acorde\ menor$$
$$\overline{c4} = (0,0,0,1,0)\ vector\ clase\ acorde\ quinta$$
$$\overline{c5} = (0,0,0,0,1)\ vector\ clase\ acorde\ septima$$

Ahora, como se decidió que solo hubieran 5 clases diferentes y en cada clase existen 12 posibles variedades porque dentro de la clase notas hay 12 notas, dentro de la clase acorde mayor hay 12 acordes mayores, en la clase acorde menor hay 12 acordes menores y así sucesivamente, entonces como resultado final se definieron 60 vectores, ya que 5 clases por 12 variedades son 60 vectores posibles, por simplicidad los vectores de clasificación y los 60 vectores tienen componentes que varían del 0 al 1, por ejemplo:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
C	C#	D	D#	E	F	F#	G	G#	A	A#	B

Los números de la parte superior representan la componente del vector y la parte de abajo las notas, ahora bien, como dije anteriormente el vector de entrada es el resultado del algoritmo PCP, el cual agrupa las frecuencias y después lo normaliza, es decir, las componentes del vector tendrán valor menor o igual a 1, por ejemplo, imaginemos que alguien toca la nota de A en la guitarra, el resultado sería algo como esto:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
C	C#	D	D#	E	F	F#	G	G#	A	A#	B

El componente 9 que corresponde a la nota A tendrá un color más intenso aproximado al azul y sus vecinos también porque las frecuencias que agrupamos tienen armónicos que pueden no ser tan precisos, ahora si los representamos numéricamente tendríamos algo como esto:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0	0	0	0	0	0	0	0	0.25	0.6	0.25	B

Finalmente para ejemplificar como se vería un acorde vectorialmente, como definimos anteriormente un acorde es un conjunto de notas bajo ciertas reglas y se definieron algunas fórmulas para no complicar su representación en términos de notas, por ejemplo el acorde C mayor está compuesto por las notas:

1	**1#** **2b**	**2**	**2#** **3b**	**3**	**4**	**4#** **5b**	**5**	**5#** **6b**	**6**	**6#** **7b**	**7**	**8**
C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C

Siguiendo el mismo ejemplo la representación numérica sería algo así:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0.7	0.1	0	0.1	0.8	0.1	0	0.8	0.1	0.1	0.1	0

Una vez comprendida la idea se tienen que definir los vectores prototipos que estarán basados en los vectores de entrada pero con la diferencia que serán idealizados y el valor de sus componentes más grandes serán la nota o conjunto de notas que va a representar en el caso de los acordes ejemplo:

Vector prototipo de la nota A:

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0.1	0	0

Vector prototipo del acorde C mayor:

	0	1	2	3	4	5	6	7	8	9	10	11
0.1	0	0	0	0	0.1	0	0	0.1	0	0	0	0

Siguiendo la misma idea se obtienen todos los vectores prototipo.

Nota: se escogió como valor inicial 0.1 debido a que durante la programación de la red neuronal se notó que cuando se ponía un valor más grande algunas neuronas se morían o lo que es lo mismo, ya no se modificaba su valor y por lo tanto eran inservibles.

Vectores de notas:

w1=(0.1,0,0,0,0,0,0,0,0,0,0,0)
w2=(0,0.1,0,0,0,0,0,0,0,0,0,0)
w3=(0,0,0.1,0,0,0,0,0,0,0,0,0)
w4=(0,0,0,0.1,0,0,0,0,0,0,0,0)
w5=(0,0,0,0,0.1,0,0,0,0,0,0,0)
w6=(0,0,0,0,0,0.1,0,0,0,0,0,0)
w7=(0,0,0,0,0,0,0.1,0,0,0,0,0)
w8=(0,0,0,0,0,0,0,0.1,0,0,0,0)
w9=(0,0,0,0,0,0,0,0,0.1,0,0,0)
w10=(0,0,0,0,0,0,0,0,0,0.1,0,0)
w11=(0,0,0,0,0,0,0,0,0,0,0.1,0)
w12=(0,0,0,0,0,0,0,0,0,0,0,0.1)

Vectores acordes mayores:

w13=(0.1,0,0,0,0.1,0,0,0.1,0,0,0,0)
w14=(0,0.1,0,0,0,0.1,0,0,0.1,0,0,0)
w15=(0,0,0.1,0,0,0,0.1,0,0,0.1,0,0)
w16=(0,0,0,0.1,0,0,0,0.1,0,0,0.1,0)
w17=(0,0,0,0,0.1,0,0,0,0.1,0,0,0.1)
w18=(0.1,0,0,0,0,0.1,0,0,0,0.1,0,0)
w19=(0,0.1,0,0,0,0,0.1,0,0,0,0.1,0)
w20=(0,0,0.1,0,0,0,0,0.1,0,0,0,0.1)
w21=(0.1,0,0,0.1,0,0,0,0,0.1,0,0,0)

w22=(0,0.1,0,0,0.1,0,0,0,0,0.1,0,0)
w23=(0,0,0.1,0,0,0.1,0,0,0,0,0.1,0)
w24=(0,0,0,0.1,0,0,0.1,0,0,0,0,0.1)

Vectores acordes menores:

w25=(0.1,0,0,0.1,0,0,0,0.1,0,0,0,0)
w26=(0,0.1,0,0,0.1,0,0,0,0.1,0,0,0)
w27=(0,0,0.1,0,0,0.1,0,0,0,0.1,0,0)
w28=(0,0,0,0.1,0,0,0.1,0,0,0,0.1,0)
w29=(0,0,0,0,0.1,0,0,0.1,0,0,0,0.1)
w30=(0.1,0,0,0,0,0.1,0,0,0.1,0,0,0)
w31=(0,0.1,0,0,0,0,0.1,0,0,0.1,0,0)
w32=(0,0,0.1,0,0,0,0,0.1,0,0,0.1,0)
w33=(0,0,0,0.1,0,0,0,0,0.1,0,0,0.1)
w34=(0.1,0,0,0,0.1,0,0,0,0,0.1,0,0)
w35=(0,0.1,0,0,0,0.1,0,0,0,0,0.1,0)
w36=(0,0,0.1,0,0,0,0.1,0,0,0,0,0.1)

Vectores acordes en quinta:

w37=(0.1,0,0,0,0,0,0,0.1,0,0,0,0)
w38=(0,0.1,0,0,0,0,0,0,0.1,0,0,0)
w39=(0,0,0.1,0,0,0,0,0,0,0.1,0,0)
w40=(0,0,0,0.1,0,0,0,0,0,0,0.1,0)
w41=(0,0,0,0,0.1,0,0,0,0,0,0,0.1)
w42=(0.1,0,0,0,0,0.1,0,0,0,0,0,0)
w43=(0,0.1,0,0,0,0,0.1,0,0,0,0,0)
w44=(0,0,0.1,0,0,0,0,0.1,0,0,0,0)
w45=(0,0,0,0.1,0,0,0,0,0.1,0,0,0)
w46=(0,0,0,0,0.1,0,0,0,0,0.1,0,0)
w47=(0,0,0,0,0,0.1,0,0,0,0,0.1,0)
w48=(0,0,0,0,0,0,0.1,0,0,0,0,0.1)

Vectores acordes en séptima:

w49=(0.1,0,0,0,0.1,0,0,0.1,0,0,0.1,0)
w50=(0,0.1,0,0,0,0.1,0,0,0.1,0,0,0.1)
w51=(0.1,0,0.1,0,0,0,0.1,0,0,0.1,0,0)
w52=(0,0.1,0,0.1,0,0,0,0.1,0,0,0.1,0)
w53=(0,0,0.1,0,0.1,0,0,0,0.1,0,0,0.1)
w54=(0.1,0,0,0.1,0,0.1,0,0,0,0.1,0,0)
w55=(0,0.1,0,0,0.1,0,0.1,0,0,0,0.1,0)
w56=(0,0,0.1,0,0,0.1,0,0.1,0,0,0,0.1)
w57=(0.1,0,0,0.1,0,0,0.1,0,0.1,0,0,0)
w58=(0,0.1,0,0,0.1,0,0,0.1,0,0.1,0,0)
w59=(0,0,0.1,0,0,0.1,0,0,0.1,0,0.1,0)

w60=(0,0,0,0.1,0,0,0.1,0,0,0.1,0,0.1)

Por último se definirán las matrices conformadas por los vectores prototipos. La matriz W^1 está definida por renglones conformados por los vectores w1 hasta el vector w60.

$$W^1 = \begin{bmatrix} w1 \\ . \\ : \\ w60 \end{bmatrix}$$

La dimensión de la matriz resultante es 60x12.

Ahora, para definir la matriz W^2, observamos cuantas clases tenemos y de que dimensión son los renglones de la matriz W^1, ya que para que sea consistente la multiplicación del vector resultante de la capa oculta (W^1) el número de columnas W^2 debe ser el mismo número de renglones que dicho vector resultante, es decir, 60 y el número de renglones de la matriz W^2 es igual al número de vectores clase, lo que da como resultado una matriz de 5x60.

Otra forma de verlo es iterar y poner en forma de columna los vectores de clase tantas veces como la dimensión de los vectores "w" de cada clase, es decir:

$$c1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, c2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, c3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, c4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, c5 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Iteramos 12 veces el vector c1 y ponemos en forma de columna en la matriz W^2

$$W^2 = \begin{bmatrix} 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \ldots\ldots \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix}$$

Siguiendo la misma idea, completamos la matriz:

$$W^2 = \begin{bmatrix} 111111111111000000000000000000000000000000000000000000000000 \\ 000000000000111111111111000000000000000000000000000000000000 \\ 000000000000000000000000111111111111000000000000000000000000 \\ 000000000000000000000000000000000000111111111111000000000000 \\ 000000000000000000000000000000000000000000000000111111111111 \end{bmatrix}$$

Gráficamente se vería de la siguiente manera:

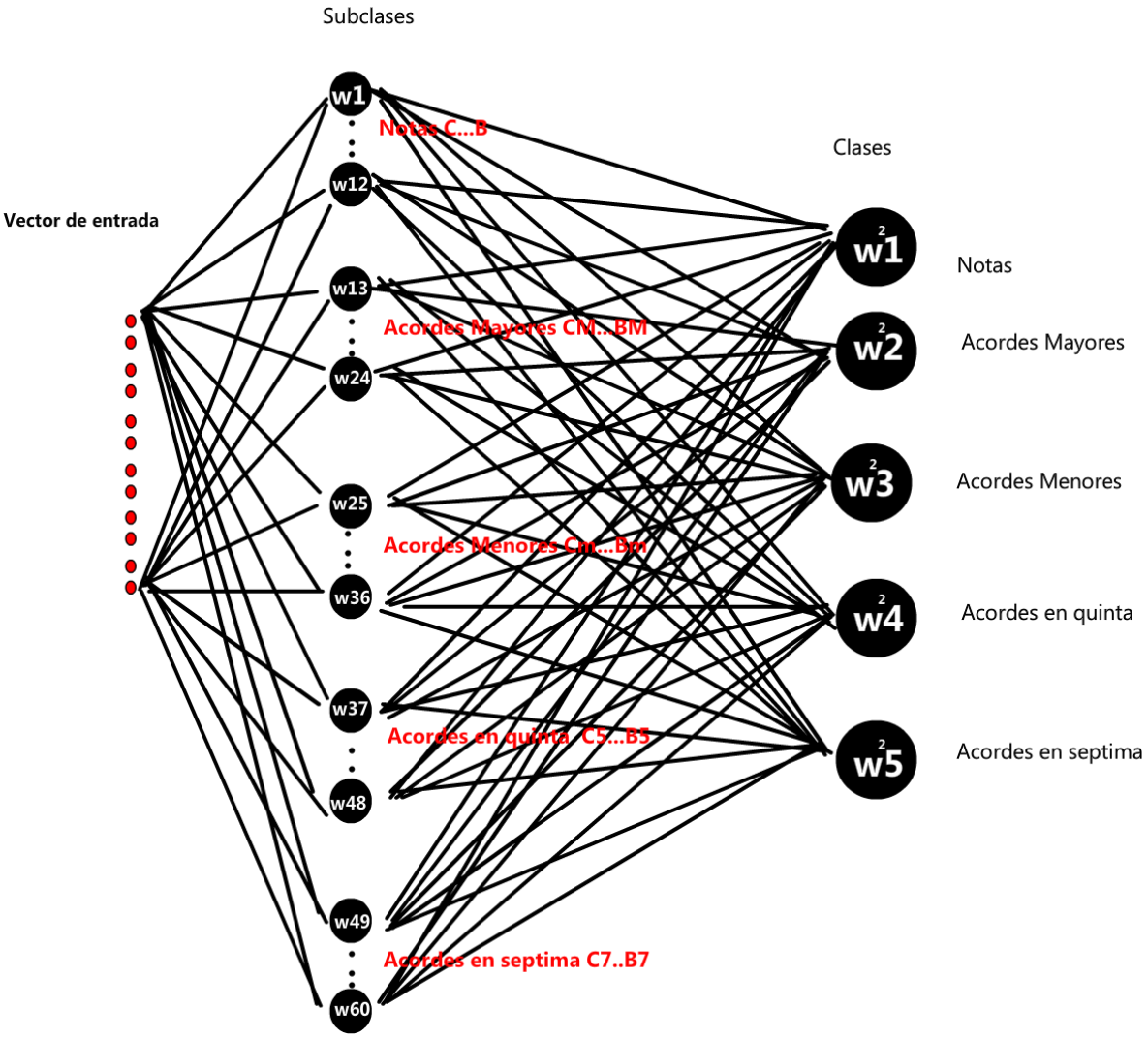


Figura 4.14 Red neuronal reconocedora de notas y acordes

4.6.-Diseño del subsistema gestor de actividades

El subsistema de gestor de actividades será el encargado de ejecutar las actividades programadas, así como de mantener un nivel de dificultad adecuado, las clases que conforman a este subsistema son los que se muestran en la figura 4.15.

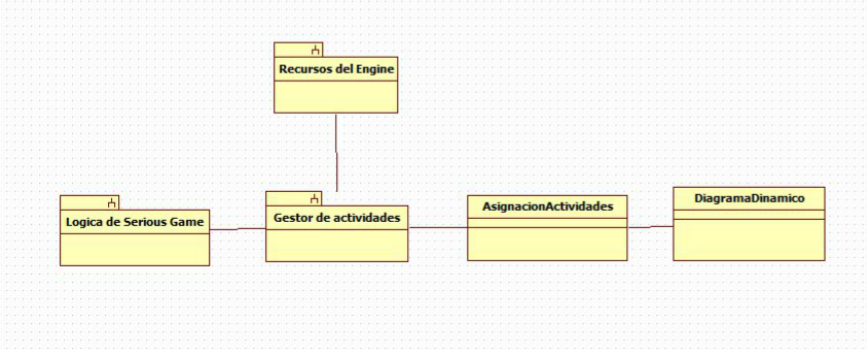


Figura 4.15 Relaciones del subsistema "Gestor de actividades"

Para el diseño de este subsistema se hará uso de la metodología de Marfisi Schottman por esta razón primero se definirá el modelo del juego, después se especificarán los objetivos pedagógicos y al final se diseñarán las actividades y se generarán los escenarios pedagógicos que en este proyecto llamamos cómo: "contenido del serious game", en el siguiente capítulo se unirá el contenido con los recursos del subsistema "Lógica del juego" para implementar las clases mostradas en la figura 4.5.

4.6.1.- El modelo del juego

A continuación se muestra una tabla con todos los atributos destacables del juego propuesto.

Atributo	**Valor**
identificador	Juego01
Nombre	Guitar Tower
Área de conocimiento	Música, Guitarra básica
Edades para jugarlo	10 años en adelante
Dificultades	Normal
Interacción	Con ayuda de una guitarra o sin una guitarra el jugador podrá cumplir los retos propuestos en el juego, el objetivo principal consiste en impedir que sus enemigos ataquen a su torre.
Tipo	Tower Deffense
Dispositivos	PC y móvil
Multimedia	Incluye gráficos, animaciones, cinemáticas, sonidos, música, manual y tutoriales

Tabla 4.6 Modelo del juego

4.6.2.-Especificación de los objetivos pedagógicos

Atributo	Valor
Identificador	OP01
Nombre general	Afinar guitarra
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Familiarizarse con la guitarra y aprender el procedimiento de afinación
Modelo de objetivos educativos	Conocer la nota principal de cada cuerda

Atributo	Valor
Identificador	OP02
Nombre general	Digitalización Básica
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Hacer ejercicios de digitalización basados en patrones melódicos sencillos
Modelo de objetivos educativos	objetivo de tipo informativo, nivel: conocer la posición de las notas en una porción de diapasón

Atributo	Valor
Identificador	OP03
Nombre general	Digitalización de posición 1 Escala pentatónica menor
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Aprender cómo se representa la posición 1 de la escala pentatónica
Modelo de objetivos educativos	objetivo de tipo informativo, nivel: reconocer la posición 1 de la escala pentatónica

Atributo	Valor
Identificador	OP04
Nombre general	Digitalización de posición 2 Escala pentatónica menor
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Aprender cómo se toca la posición 2 de la escala pentatónica menor

Modelo de objetivos educativos	objetivo de tipo informativo, nivel: reconocer la posición 2 de la escala pentatónica

Atributo	**Valor**
Identificador	OP05
Nombre general	Digitalización de posición 3 Escala pentatónica menor
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Aprender cómo se puede representar la posición 3 de la escala pentatónica
Modelo de objetivos educativos	objetivo de tipo informativo, nivel: reconocer la posición 3 de la escala pentatónica

Atributo	**Valor**
Identificador	OP07
Nombre general	Digitalización de posición 1 Escala mayor
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Aprender cómo se puede representar la posición 1 de la escala mayor
Modelo de objetivos educativos	objetivo de tipo informativo, nivel: reconocer la posición 1 de la escala mayor

Atributo	**Valor**
Identificador	OP08
Nombre general	Digitalización de posición 2 Escala mayor
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Aprender cómo se puede representar la posición 2 de la escala mayor
Modelo de objetivos educativos	objetivo de tipo informativo, nivel: reconocer la posición 2 de la escala mayor

Atributo	**Valor**
Identificador	OP09

Nombre general	Digitalización de posición 3 Escala mayor
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Aprender cómo se puede representar la posición 3 de la escala mayor
Modelo de objetivos educativos	objetivo de tipo informativo, nivel: reconocer la posición 3 de la escala mayor

Atributo	**Valor**
Identificador	OP10
Nombre general	Progresión I-IV-V
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Aprender a tocar la progresión I-IV-V con diferentes tonos
Modelo de objetivos educativos	objetivo de tipo informativo, nivel: reconocer la progresión I-IV-V de forma particular

Atributo	**Valor**
Identificador	OP11
Nombre general	Progression i-vi-ii-V
Área de conocimiento	Música
Área trasversal	Guitarra básica
Contenido	Aprender a tocar la progresión i-vi-ii-V con diferentes tonos
Modelo de objetivos educativos	objetivo de tipo informativo, nivel: reconocer la progresión i-vi-ii-V de forma particular

Tabla 4.7 Objetivos pedagógicos

4.6.3-Diseño de actividades y contenido

Como los objetivos giran en torno a aprender a tocar o reconocer notas o acordes de una guitarra, las actividades serán cien por ciento prácticas y estarán representadas por tablaturas, una tablatura es una forma de representar las notas que componen a una canción y está compuesta por seis líneas horizontales y números que indican el número de traste que se desea tocar, en una tablatura también se pueden representar acordes pero en la mayoría de melodías se prefiere usar la notación americana con letras mayúsculas que van desde la "A" a la "G" añadiendo un # si tiene sostenido (descrito en el capítulo 3), para la definición de las actividades se usarán las letras A-G son para representar notas y las letras A-G pero con un sufijo como: AM (acorde mayor), Am (acorde menor), A5 (acorde en quinta), A7 (acorde en séptima), representaran un tipo de acorde .

En el capítulo 3 se explicó el método de enseñanza inductiva y se resumió que consiste en ir de un conocimiento particular a uno más general, por consiguiente para ilustrar mejor la idea se procederá a mostrar cómo se diseñaron algunos ejercicios los cuales están basados en patrones melódicos y fueron creados especialmente para este proyecto.

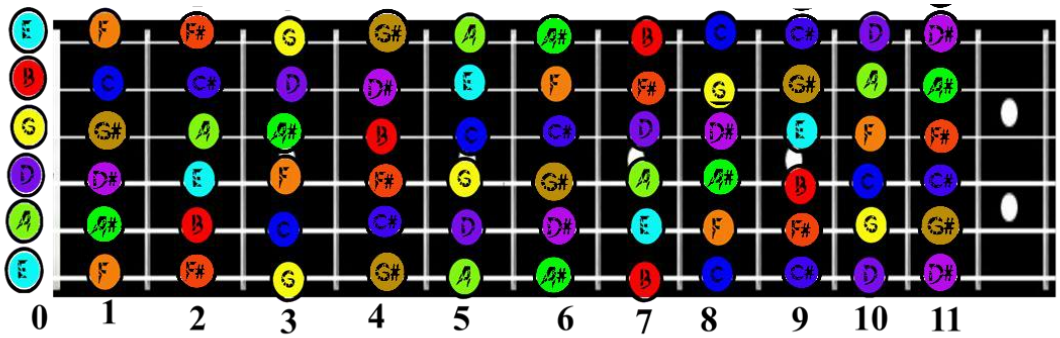


Figura 4.16 diagrama completo con notas representadas por un color específico

En la figura 4.16 se expone el diagrama más general de un mástil con las notas de la escala cromática, partiendo de este diagrama y de acuerdo a la definición de escala mayor es posible afirmar que es un subconjunto de la escala cromática, por ende se obtiene un diagrama más limpio en el mástil de la guitarra que se ve de la siguiente manera:

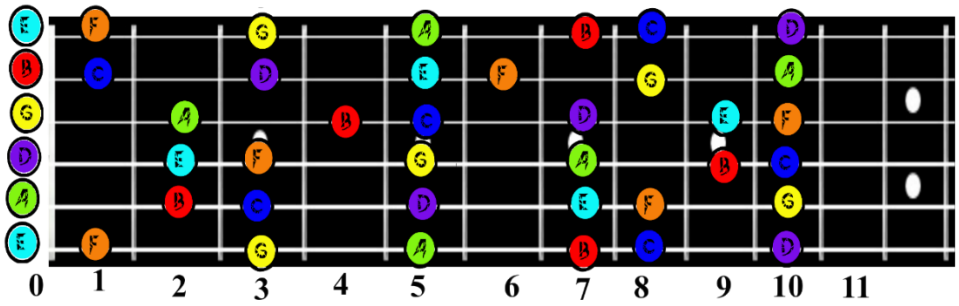


Figura 4.17 diagrama escala mayor de C

Sin embargo existe otra escala más sencilla compuesta de cinco notas conocida como la escala pentatónica menor en tono de A (la más usual para hacer licks).

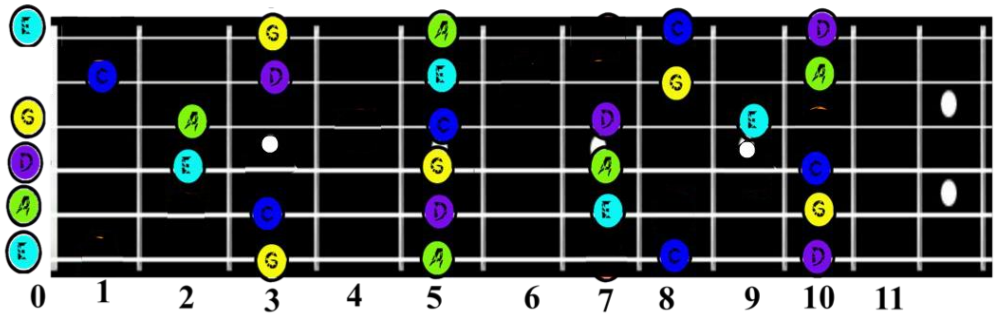


Figura 4.18 Diagrama de la escala pentatónica

Los dos diagramas mostrados en la figura 4.17 y 4.18 son las escalas más fáciles de estudiar y aprender son la base para aprender a improvisar, sin embargo aunque son pocas notas las escalas de la figura 4.18 y 4.17 se prefiere estudiarlas por separado a estas divisiones se les conocen como "posiciones de la escala mayor, pentatónica o cualquier otra". Para tener un mejor orden en la implementación de las actividades se sugiere organizar las actividades en una base de datos de tal forma que cualquier persona ajena a temas de programación pueda agregar o modificar actividades, y después el programador las pueda implementar al sistema adecuadamente. En la figura 4.19 se muestra un diagrama de entidad-relación propuesto para este sistema.

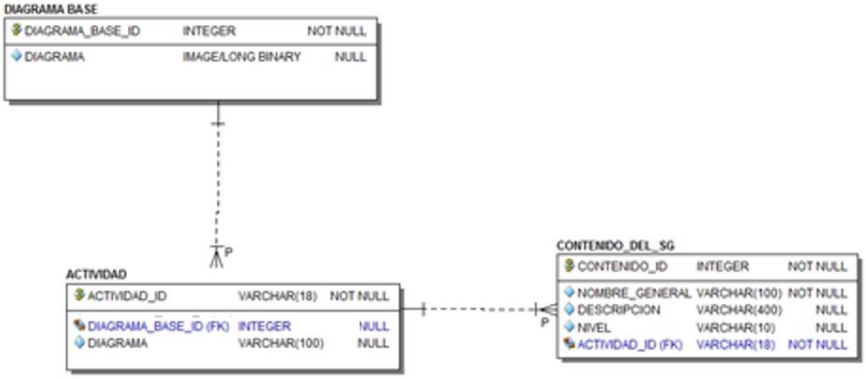


Figura 4.19 Diagrama entidad-relación

El diagrama de la figura 4.19 tiene 3 entidades, suponiendo que hay un experto para la enseñanza de un área específica (en este caso guitarra acústica), la persona que diseñara las actividades solo vera el contenido de las entidades "DIAGRAMA BASE" y "ACTIVIDAD" pues los ejercicios y diagramas almacenados en estas entidades serán legibles para la persona y cualquier persona que conozca un poco de la lectura de tablaturas o pentagramas si es el caso. La entidad "CONTENIDO_DEL_SG" tiene como registros los retos en cada nivel y la relación con las actividades definidas por el diseñador de las actividades, de esa forma se puede unir el área pedagógica con el área de programación.

En la siguiente tabla se muestran los diagramas base que se usaron para generar los patrones melódicos en la tabla “ACTIVIDADES”.

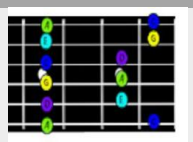
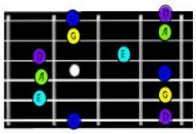
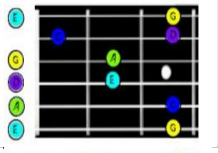
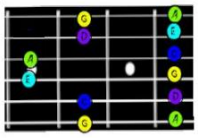
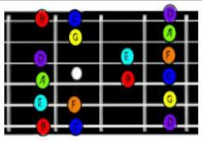
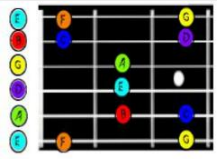
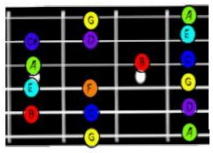
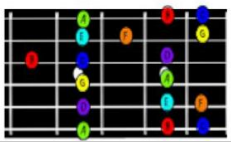
DIAGRAMA_BASE_ID	DIAGRAMA
01	
02	
03	
04	
05	
06	
07	
08	

Tabla 4.8 Diagramas base

En la columna “DIAGRAMA” de la tabla siguiente se muestra la tablatura de cada actividad y en la columna “DIAGRAMA BASE_ID” muestra el identificador de la posición de la cual se obtuvo dicha tablatura, en el caso de que una actividad sea una progresión o secuencia de acordes no se tendrá relación con la tabla “DIAGRAMA BASE”.

ACTIVIDAD:ID	DIAGRAMA_BASE_ID	DIAGRAMA
P01-01	01	1|----------5-8----------| 2|----------5—8----------| 3|----------5-7----------| 4|----------5-7----------| 5|----------5-7----------| 6|----5—8----------|
P01-02	01	1|---8-5----------| 2|-------8-5----------| 3|----------7-5----------| 4|----------7-5----------| 5|----------7-5----------| 6|----------8-5----------|
P01-03	01	1|----------| 2|----------| 3|----------| 4|----------| 5|----------| 6|----------8-8---5-5------|
P11-04	01	1|----------| 2|----------| 3|----------| 4|----------| 5|-------7-7—5-5----------| 6|----------|
P11-05	01	1|----------| 2|----------| 3|----------| 4|----------5-7----------| 5|-------5-7----------| 6|----------|
P11-06	01	1|----------| 2|----------| 3|----------| 4|-----5----------| 5|----------7----------| 6|----------|
P11-07	01	1|----------| 2|----------| 3|--------5----------| 4|----7--------7-----5—7-----| 5|----------| 6|----------|
P01-08	01	1----------| 2----------5----------| 3-------5-----5-------5---------8----| 4----------7----------| 5----------| 6----------|
P01-09	01	1----------| 2----------| 3----------| 4-----5-7-5--------5----------| 5----------7--------7----------| 6----------|
P01-10	01	1----------| 2----------| 3----------5----------| 4----------7----------| 5--------5-7----------| 6----------|
P01-11	01	1----------|

		2----------| 3----------5---------| 4-----5-7-5--------5----------7------| 5--------------7--------7------5-7----------| 6----------|
P01-12	01	1-----5—8-5-----5----------| 2--------------8----------| 3----------| 4----------| 5----------| 6----------|
P01-04	01	1|----------| 2|----------| 3|----------| 4|----------| 5|------7-7—5-5----------| 6|----------|
. . .	…….	……
P02-01	02	1----------8—10-----| 2----------8----10----------| 3----------7-9----------| 4----------7-10----------| 5----------7—10----------| 6------8-10----------|
P02-02	02	1------10—8----------| 2--------------10—8----------| 3----------9—7----------| 4----------10-7----------| 5----------10 —7----------| 6----------10---8---------|
P02-03	02	1----------| 2----------8----------| 3----------7—9-------9----------| 4-----10----------| 5----------| 6----------|
……	……	……..
P03-01	03	1----------0—3-----| 2----------1---3----------| 3----------0—2----------| 4----------0-2----------| 5----------0-3----------| 6------0-3----------|-|
P03-02	03	1----3—0----------| 2----------3—1----------| 3----------2—0----------| 4----------2—0----------| 5----------3—0----------| 6----------3—0------|
P03-03	03	1----------| 2----------| 3--------0—2—0------0---| 4----------2------| 5----------| 6----------|
. . .		|
P04-01	04	1----------3-5---| 2----------3—5----------| 3----------2-5----------| 4----------2-5----------| 5----------3-5----------| 6----3—5----------|
P04-02	04	1-----5-3----------| 2----------5-3----------|

		3----------5-2----------| 4----------5-2----------| 5----------5-3----------|| 6----------5-3----------|
P04-03	04	1----------| 2----------| 3----------| 4----------| 5----3----------| 6-------5-3-5-----|
. . .	……	…….
M01-01	05	1----------7-8-10--| 2----------8-10----------| 3----------7-9-10----------| 4----------7—9-10----------| 5----------7-8-10----------| 6------8—10----------|
M01-02	05	1---------10-8-7----------| 2----------10-8----------| 3----------10—9-7----------| 4----------10-9-7----------| 5----------10-8-7----------| 6----------10—8---|
M01-03	05	1----------| 2----------| 3----------| 4----------| 5----------7-8-10---------| 6------8-10 ----------|
…….	…….	…….
M02-01	06	1----------0---1—3----| 2----------0-1-3----------| 3----------0-2----------| 4----------0—23----------| 5----------0---2-3----------| 6-----0—1—3----------|
M02-02	06	1------3—1—0----------| 2----------3—1—0----------| 3----------2-0----------| 4----------3---2-0----------| 5----------3—2-0----------| 6----------3-10----|
M02-03	06	1----------| 2----------| 3----------| 4----------| 5----------| 6-----0-0----1---0-0---3--------|
……..	……	…….
M03-01	07	1----------3—5----| 2----------3---5---6----------| 3----------2—4-5----------| 4----------2—3—5----------| 5----------2—3—5----------| 6------3—5----------|
M03-02	07	1----------5—3----------| 2----------6—5-----3----------| 3----------5—4—2----------| 4----------5—3---2----------| 5----------5----3—2--------| 6----------5—3-|
M03-03	07	1----------| 2----------| 3----------| 4----------| 5----------2---------2---3---5--------| 6-------5—3--------5----------|

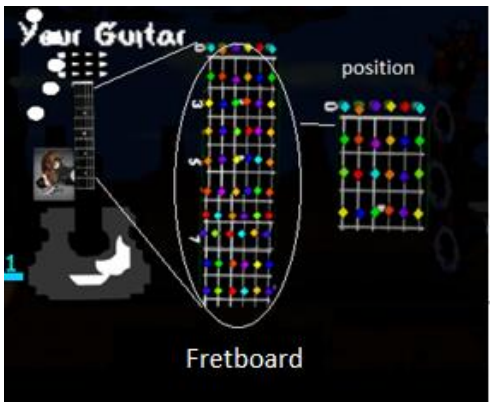
……..	……..	…….
M04-01	08	1----------5—7-8---| 2----------5—6—8----------| 3----------4---5—7----------| 4----------5—7----------| 5----------5—7-8----------| 6-----5—7—8----------|
M04-02	08	1—8-7—5----------| 2----------8-6—5----------| 3----------7-5-4----------| 4----------7-5----------| 5----------8—7-5----------| 6----------8-7—5----------|
M04-03	08	1----------| 2----------| 3----------| 4----------| 5----------5---------6---------3-----| 6-------5-5-5-----5-5-5----5-5-5-------|
. . .		
A01-00		C-D-E-F—G-A-B
A01-01		C#m-Dm-Em-F#m-G#m-Am-A#m-Bm
A01-02		C5-C#5-D5-D#5-E5-F5-F#5-G5-G#5-A5-A#5-B5
A01-03		C7-D7-E7-F7-G7-A7-B7
. . .		
A02-00		C-F-G
A02-01		E-A-B
A02-02		G-C-D
A02-03		A-D-E
. . .		
A03-00		C5-F5-G5
A03-01		E5-A5-B5
A03-02		G5-C5-D5
A03-03		A5-D5-E5
. . .		
A04-00		C-F7-G
A04-01		E-A7-B
A04-02		G-C7-D
A04-03		A-D7-E
. .		
A05-00		Dm-G-C
A05-01		Em-A-D
A05-02		F#m-B-E
A05-03		C#m-F#-B
. . .		
A06-00		C-Am-Dm-G
A06-01		D-Bm-Em-A
A06-02		E-C#m-F#m-B
. . .		
A07-00		G-Em-Am-D
A07-01		A-F#m-Bm-E
A07-02		B-G#m-C#m-F#

Tabla 4.9 Actividades

La última tabla muestra varias columnas que podrían resultar confusas por eso se describirán las más importantes:

- Nombre general: nombre del reto
- Descripción: que va a contener el reto
- Nivel: el nivel donde se va a aplicar el reto
- Relación: la relación que tiene el contenido actual con el contenido previo

En cada actividad se hizo uso de recursos gráficos para la ejecución de cada ejercicio, por ejemplo para el tutorial se usó una animación:



Para las demás actividades se usaron recursos de audio, gráficos y animaciones (ver figura 4.4, 5.3, a la 5.7).

CONTENIDO_ID	NOMBRE_GENERAL	DESCRIPCION	NIVEL	RELACION	ACTIVIDAD_ID
01	¿Cómo se juega?	Se ofrece un tutorial al jugador donde se presenta el diagrama completo basado en la actividad P01-01	Nivel 1.0	0	01
02	Primer reto	Se ejecuta el ejercicio P01-01 siempre mostrando el diagrama	Nivel 1.0	01->02	02
03	Ocultando diagrama	Se ejecuta el ejercicio P01-02, si el usuario acierta 10 veces entonces suceden los siguientes eventos: a)si tiene guitarra: se oculta el diagrama b)si no se tiene guitarra: los colores de los botones para generar las ondas se vuelven color blanco	Nivel 1.1	0	03
04	Ocultando diagrama	Se ejecuta el ejercicio P01-03, tendrá 1 tiempo por cada	Nivel 1.2	0	04

		nota y 4 tiempos de repetición, si el usuario acierta 10 veces entonces suceden los siguientes eventos: a)si tiene guitarra: se oculta el diagrama b)si no se tiene guitarra: los colores de los botones para generar las ondas se vuelven color blanco			
05	Reto base 05	Se ejecuta el ejercicio P01-04, tendrá 3 tiempos por cada nota y 4 tiempos de repetición, si el usuario acierta 9 veces entonces suceden los siguientes eventos: a)si tiene guitarra: se oculta el diagrama b)si no se tiene guitarra: los colores de los botones para generar las ondas se vuelven color blanco	Nivel 1.3	0	05
06	Reto base 06	Se ejecuta el ejercicio P01-05, tendrá 2 tiempos por cada nota y 2tiempos de repetición, si el usuario acierta 9 veces entonces suceden los siguientes eventos: a)si tiene guitarra: se oculta el diagrama b)si no se tiene guitarra: los colores de los botones para generar las ondas se vuelven color blanco	Nivel 1.4	0	06
07	Unión del reto base 05 y reto base 06	Se ejecuta el ejercicio P01-06, tendrá 2 tiempos por cada nota y 2tiempos de repetición, si el usuario acierta 9 veces entonces suceden los siguientes eventos: a)si tiene guitarra: se oculta el diagrama	Nivel 1.5	05,06->07	07

		b)si no se tiene guitarra: los colores de los botones para generar las ondas se vuelven color blanco			
08	Reto base 08	Se ejecuta el ejercicio P01-08, tendrá 3 tiempos por cada nota y 4 tiempos de repetición, si el usuario acierta 9 veces entonces suceden los siguientes eventos: a)si tiene guitarra: se oculta el diagrama b)si no se tiene guitarra: los colores de los botones para generar las ondas se vuelven color blanco	Nivel 1.6	0	08
09	Reto base 09	Se ejecuta el ejercicio P01-09, tendrá 2 tiempos por cada nota y 2tiempos de repetición, si el usuario acierta 9 veces entonces suceden los siguientes eventos: a)si tiene guitarra: se oculta el diagrama b)si no se tiene guitarra: los colores de los botones para generar las ondas se vuelven color blanco	Nivel 1.7	0	09
10	Reto base 09	Se ejecuta el ejercicio P01-10, tendrá 3 tiempos por cada nota y 4 tiempos de repetición, si el usuario acierta 9 veces entonces suceden los siguientes eventos: a)si tiene guitarra: se oculta el diagrama b)si no se tiene guitarra: los colores de los botones para generar las ondas se vuelven color blanco	Nivel 1.8	0	P01-10
11	Reto base 08	Se ejecuta el ejercicio P01-11, tendrá 2tiempos por cada nota y 1 tiempo de	Nivel 1.9	0	P01-11

		repetición, si el usuario acierta 9 veces entonces suceden los siguientes eventos: a)si tiene guitarra: se oculta el diagrama b)si no se tiene guitarra: los colores de los botones para generar las ondas se vuelven color blanco			
12	Reto fin del nivel 1	Se generan notas aleatorias según el diagrama base por id 01	Nivel 1.10	09,10,11->12	
…..					

Tabla 4.10 Contenido del serious game

En la tabla 4.8 se puede observar que el registro cuyo atributo “CONTENIDO_ID” lleva por valor el número 12 se puede observar que el atributo “RELACION” contiene “09,10, 11 ->12” esto significa que el contenido actual está relacionado con el contenido 09, 10 y 11, es decir el contenido 12 es la unión de las actividades, esto ya se había descrito en el capítulo 3 en el tema “método de enseñanza inductivo en la música”.

Capítulo 5.- Prototipo e Implementación de los subsistemas

Usando el método de los seis pasos de Schottman (para más información ver apéndice A) y según la arquitectura propuesta en este trabajo, se hicieron varios prototipos y cada uno fue evaluado por usuarios de diferentes edades (mayores a 10 años) hasta llegar al prototipo más adecuado, en este capítulo se presentara como se programaron las partes más importantes del prototipo y como se empezó a desarrollar.

5.1.- Desarrollo del prototipo

La primera actividad para el desarrollo del prototipo fue el diseño del arte ya que al ser un requerimiento no funcional no juega un papel importante en la programación de cada elemento pero sirve como guía para programar la interacción entre los distintos objetos en escena, primero se realizaron gráficos de figuras geométricas simples, después se hicieron dibujos propios hasta que se llegó a la conclusión de que era más conveniente usar arte libre de derechos de autor (royalty free) en la figura 5.1 se puede observar la evolución del arte visual, algunos gráficos fueron convertidas a hojas de sprites, otros fueron modificados para atender los requerimientos RNF-01, RNF-04, RNF-06 y RNF-07, los principales objetos en escena se muestran en las figuras 5.2, 5.3, 5.4, 5.5, 5.6 y 5.7.

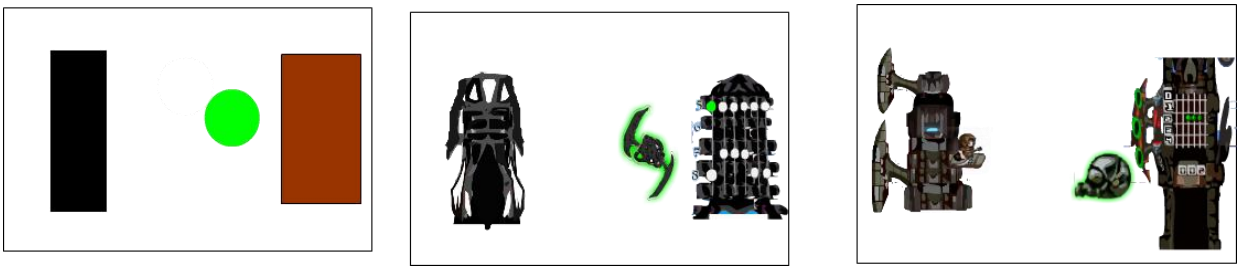


Figura 5.1 Evolución visual

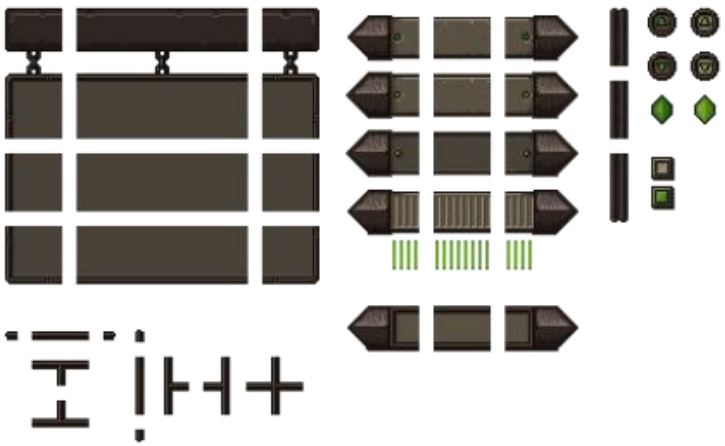


Figura 5.2 Interfaz de usuario

Figura 5.3 Hoja de sprites de la torre del jugador con diferentes variaciones

Torres de retos:

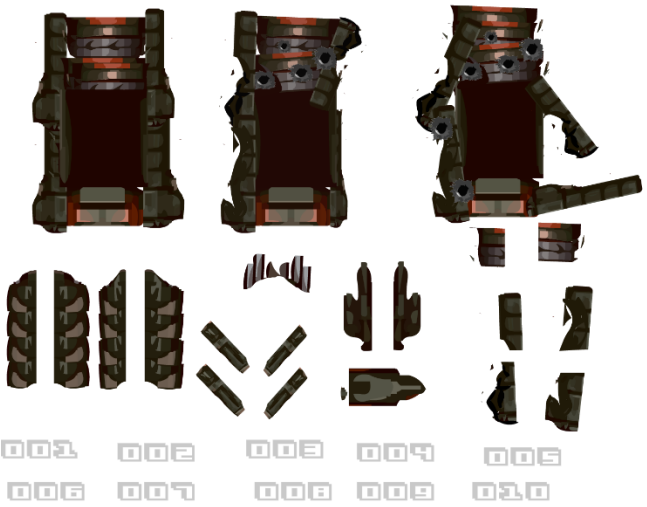


Figura 5.4.Torre de retos con complementos

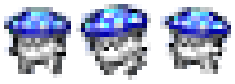


Figura 5.5 Hongo azul animado

Figura 5.6 Murciélago animado

Figura 5.7 Espectro animado

El siguiente paso durante la fase de desarrollo del prototipo fue el diseño de los niveles el cual corresponde en parte a la generación de los escenarios pedagógicos, el diseño de cada nivel y cada reto en el nivel es dependiente a las actividades propuestas en el subtema 4.6.3 ya que se deben ligar con los recursos que se van a usar, el diseño de niveles puede variar según la o las personas que estén involucradas y depende de su experiencia, en este caso como el contenido estuvo ligado a actividades prácticas con la guitarra (como tocar progresiones y notas de una escala) se decidió poner en cada nivel una "torre" que generara instancias graficas (en la figura 5.8 son los círculos azules y verdes) siguiendo la lógica del gestor de actividades, cada instancia estaría identificada por un color y una etiqueta dentro del código, la primera idea que se tuvo se puede observar en la figura 5.8, sin embargo después de hacer algunas pruebas se descubrió que les resultaba difícil tocar las notas que corresponden a las etiquetas de dos instancias diferentes (con diferente etiqueta y diferente color), y además no se mostraba ninguna otra información más que los colores aunque en el manual se decía que representaba cada uno, entonces se tuvo que modificar la generación de retos y se llegó a la conclusión que era mejor poner un diagrama sobre la torre para guiar al usuario y se forzó a que solo hubieran "n" instancias de un solo color como se muestra en la figura 5.9.

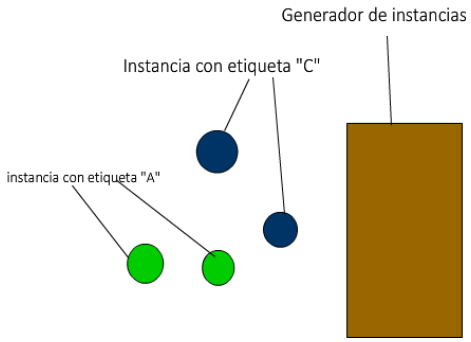


Figura 5.8 Primer diseño de la lógica de los retos

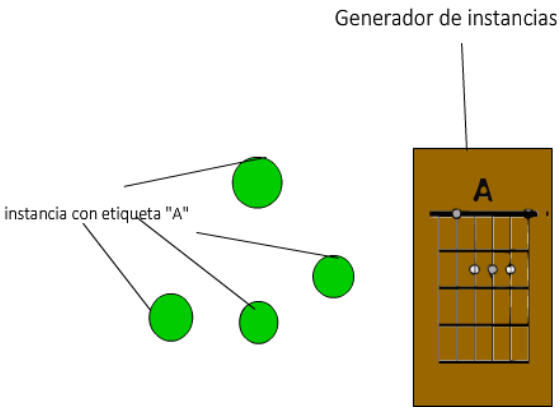


Figura 5.9 Segundo diseño de la lógica de los retos

Una vez diseñados los retos que tendrían cada nivel con ayuda del gestor de actividades y el generador de instancias (ControladorTorres) se diseñaron los niveles en forma de una torre de varios pisos, en cada piso se pone un reto mayor al anterior que siguen la lógica del gestor de actividades, en la figura 5.10 se puede observar cómo se acomodaron las torres que internamente tienen programado los parámetros para la generación de cada reto.

Figura 5.10 diseño de niveles

Durante todo el desarrollo se fue probando diferentes caminos para poner en escena las mecánicas, al final se definió con una animación parecida a una onda de sonido grafica que cambia de color según la nota o acorde que tocara el usuario, esto se puede ver más claramente en la figura 5.11, en la primera imagen se muestra la nota y la posición en el diagrama que esta incrustado en la torre también se puede observar una instancia de NPC (Non-player character) con un color de fondo verde, entonces al tocar la nota en el traste 5 se genera una onda que se puede ver en la segunda imagen, lo que provoca que el NPC desaparezca y reste número de ejercicios al objeto de la torre.

Figura 5.11 mecánica principal

En resumen, para la generación de cada prototipo primero se hicieron gráficos para programar todo lo que corresponde a los subsistemas: gestor de actividades, lógica del juego y el subsistema entrada de datos que eran los más extensos de programar, después se tuvo que programar todos los demás subsistemas dependientes a estos y mientras se hacía esto se fue probando cada prototipo para que no existieran bugs en alguna rutina.

5.2.-Implementación del subsistema de Presentación

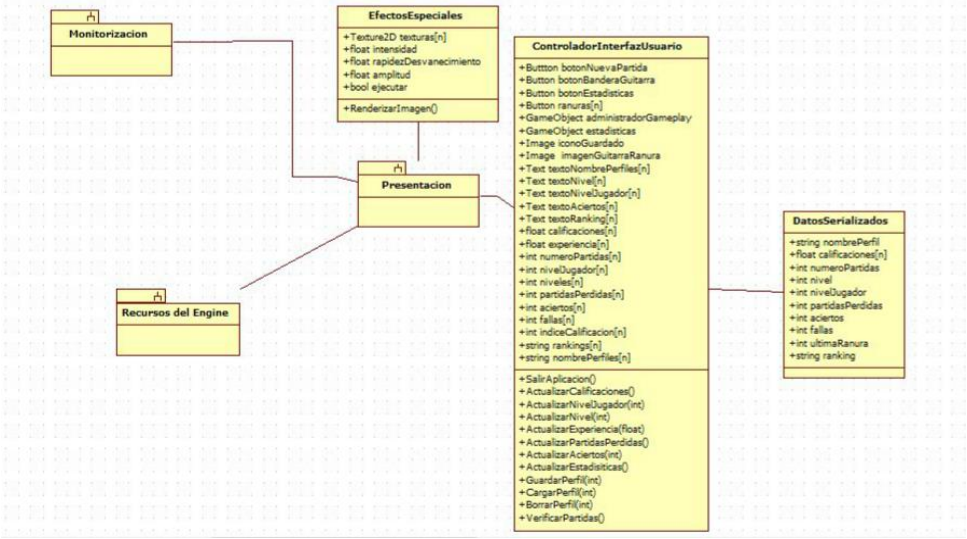


Figura 5.12 Subsistema “Presentación” con clases

Clase	Descripción
EfectosEspeciales	Su función es efectuar efectos de “temblar” y de “distorsión” para agregarle valor a los requerimientos no funcionales
ControladorInterfazUsuario	La función de esta clase es el buen manejo información del perfil de un usuario mediante elementos visuales de interfaz de usuario en pantalla, como lo son: botones, texto e iconos.
DatosSerializados	En esta clase se guardan los datos de un usuario en su perfil correspondiente.

Tabla 5.1 Descripción de las clases del subsistema “Presentación”

Para la implementación del subsistema de presentación se usó el diagrama que se muestra en la figura 5.12, se puede observar que este sistema está relacionado con el de monitorización, esta relación se debe a que estos dos subsistemas están en comunicación al inicio y al final de cada partida, los mensajes que reciben y envían se pueden observar en el diagrama de secuencia de la figura 5.13.

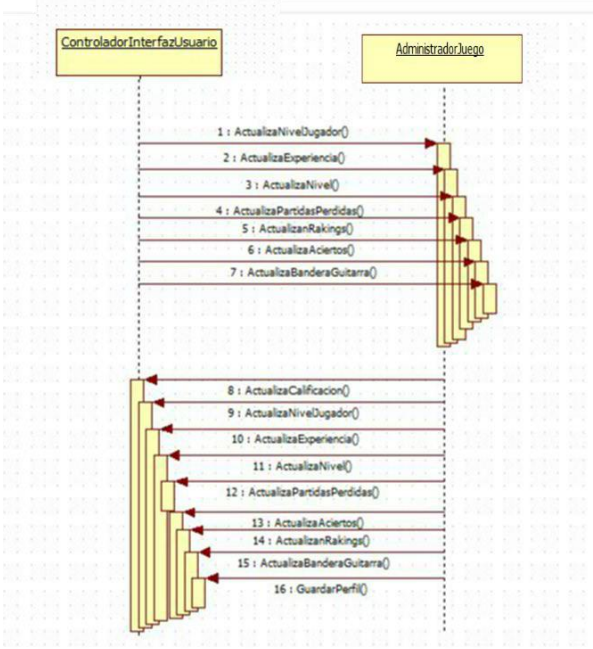


Figura 5.13 Diagrama de secuencia

Cabe mencionar que también se añadió una clase dedicada a efectos especiales usando shaders esto para mejorar el requerimiento RNF-06 (ver tabla 3.2), sin embargo se omitirá la forma de cómo se programó ya que si se quita o se mantiene no altera ninguna funcionalidad del sistema final, para la programación de la clase controlador partidas se utilizó el siguiente seudocódigo (nota: no se incluyeron todos los procedimientos, solo los más relevantes e importantes).

/* Los objetos del tipo **Text, Image** o **Button**, ya están declarados en la clase Canvas, que son uno de los recursos que ofrece el motor gráfico*/
#NUMERO_PERFILES 3
SpriteRenderer iconoGuardado
Button botonNuevaPartida
Button botonEstadisticas
Button botonBanderaGuitarra
Button ranuras[NUMERO_PERFILES]
GameObject administradorJuego
GameObject estadísticas
Image iconoGuardado
Image ImagenGuitarraRanura
Text textoNombrePerfiles[NUMERO_PERFILES]

```
Text   textoNivel[NUMERO_PERFILES]
Text   textoNivelJugador[NUMERO_PERFILES]
Text   textoAciertos[NUMERO_PERFILES]
Text  textoRanking[NUMERO_PERFILES]
float calificaciones[NUMERO_PERFILES][]
float experiencia[NUMERO_PERFILES]
int numeroPartidas[NUMERO_PERFILES]
int nivelJugador[NUMERO_PERFILES]
int niveles[NUMERO_PERFILES]
int partidasPerdidas[NUMERO_PERFILES]
int aciertos[NUMERO_PERFILES]
int fallas[NUMERO_PERFILES]
int indiceCalificacion[NUMERO_PERFILES]
int ultimaRanura
int ranuraActual
string rankings[NUMERO_PERFILES]
string nombrePerfiles[NUMERO_PERFILES]

PROCEDIMIENTO Inicializar() INICIAR
        ultimaRanura=-1
        LLAMAR VerificarPartidas();
FIN PROCEDIMIENTO

PROCEDIMIENTO InicializarPerfil(string nombre) INICIAR
        Archivo archivo=Archivo.Crear(nombre)
        DatosSerializables datos=nuevo DatosSerializables();
        datos=archivo.LeerDatos();
        datos.nombrePerfil=”Libre”
        datos.nivel=0
        datos.nivelJugador=0
        datos.partidasPerdidas=0
        datos.aciertos=0
        datos.experiencia=0
        datos.ultimaRanura=-1
        datos.indiceCalificacion=0
        archivo.Escribir(datos)
        archivo.Cerrar()

FIN PROCEDIMIENTO

PROCEDIMIENTO VerificarPartidas() INICIAR
        bool hayPartidas=verdad
        PARA i=0 hasta NUMERO_DEPERFILES HACER
                CargarPerfil(i)
                    FIN PARA
        PARA CADA s en nombresPerfiles HACER
                SI s==”Libre” y s!=”” ENTONCES
```

```
                    hayPartidas=falso
                FIN SI
            FIN PARA
            SI (Archivo.Existe(“UltimaRanura”) y hayPartidas) ENTONCES
                Archivo archivo=Archivo.Abrir (“UltimaRanura”)
                DatosSerializables datos=nuevo DatosSerializables();
                datos=archivo.LeerDatos();
                ultimaRanura=datos.ultimaRanura;
                SI ultimaRanura!=-1 ENTONCES
                    administradorJuego.EnviarMensaje()
                    administradorJuego.EnviarMensaje
("ActualizaNivelJugador",nivelJugador[ultimaRanura]);
                    administradorJuego.EnviarMensaje
("ActualizaExperiencia",experienciaJugador[ultimaRanura]);
                    administradorJuego.EnviarMensaje
("ActualizaNivel",nivel[ultimaRanura]);
                    administradorJuego.EnviarMensaje
("ActualizaPartidasPerdidas",partidasPerdidas[ultimaRanura]);
                    administradorJuego.EnviarMensaje
("ActualizanRakings",rankings[ultimaRanura]);
                    administradorJuego.EnviarMensaje
("ActualizaAciertos",aciertos[ultimaRanura]);
                    administradorJuego.EnviarMensaje
("ActualizaBanderaGuitarra",banderaGuitarra[ultimaRanura]);
                SINO ENTONCES
                    LLAMAR InicializarPerfil(”UltimaRanura”)
                FIN SINO
            FIN SI
            SINO ENTONCES
                LLAMAR InicializarPerfil(”UltimaRanura”)
            FIN SINO
FIN PROCEDIMIENTO

PROCEDIMIENTO SalirAplicacion INICIAR
        Application.Quitar();
FIN PROCEDIMIENTO

PROCEDIMIENTO ActualizarCalificacion(float c) INICIAR
        float calificación=0
        SI indiceCalifiacion[ranuraActual]>9 ENTONCES
            indiceCalifiacion[ranuraActual]=0
        FIN SI
        Califiaciones[ranuraActual][indiceCalifiacion[ranuraActual]]=c
        indiceCalifiacion[ranuraActual]++
        PARA i=0 HASTA índiceCalifiacion[ranuraActual] HACER
            calificacion=califiaciones[ranuraActual][ranuraActual][i]
        FIN HACER
```

```
        SI(calificacion>=1) ENTONCES
                rankings[ranuraActual]="A++";
        FIN SI
        SI(calificacion>=0.8&&calificacion<1) ENTONCES
                rankings[ranuraActual]="A+";
        FIN SI
        SI(calificacion>=0.6&&calificacion<0.8) ENTONCES
                rankings[ranuraActual]="A";
        FIN SI
        SI(calificacion>=0.5&&calificacion<0.6) ENTONCES
                rankings[ranuraActual]="B";
        FIN SI
        SI(calificacion>=0.3&&calificacion<0.5) ENTONCES
                rankings[ranuraActual]="C";
        FIN SI
        SI(calificacion>=0.2&&calificacion<0.3) ENTONCES
                rankings[ranuraActual]="D";
        FIN SI
        SI(calificacion>0&&calificacion<0.2) ENTONCES
                rankings[ranuraActual]="E";
        FIN SI
        SI(calificacion<=0) ENTONCES
                rankings[ranuraActual]="F";
        FIN SI
FIN PROCEDIMIENTO



PROCEDIMIENTO CargarPerfil(Int numeroRanura) INICIAR
        SI (Archivo.Existe("Ranura"+numeroRanura)) ENTONCES
                Archivo archivo=Archivo.Abrir ("Ranura"+numeroRanura)
                DatosSerializables datos=nuevo DatosSerializables();
                datos=archivo.LeerDatos()
                nombrePerfiles [numeroRanura] = datos.nombrePerfiles
                nivelJugador [numeroRanura] = datos.nivelJugador
                experiencia [numeroRanura]=datos.experiencia
                nivel [numeroRanura] = datos.nivel
                partidasPerdidas [numeroRanura] = datos.partidasPerdidas
                aciertos [numeroRanura] = datos.aciertos
                rankings [numeroRanura] = datos.rankings
                banderaGuitarra[numeroRanura]=datos.banderaGuitarra
                indiceCalificacion[numeroRanura]=datos.indiceCalificacion
                numeroPartidas[numeroRanura]=datos.numeroPartidas
                PARA  i=0 HASTA 10 HACER
                        calificaciones[numeroRanura][i]=datos.calificaciones[i];
                FIN HACER
                archivo.Cerrar()
```

```
        FIN SI
        SI NO  ENTONCES
                LLAMAR InicializarPerfil(“Ranura”+numeroRanura)
        FIN SINO
        textoNivelJugador[numeroRanura].texto=nivelJugador[numeroRanura]
        textoNivel[numeroRanura].texto=nivel[numeroRanura]
        textoPartidasPerdidas[numeroRanura].texto=muertes[numeroRanura]
        textoAciertos[numeroRanura].texto=enemigosAniquilados[numeroRanura]
        textoRanking[numeroRanura].texto=rankings[numeroRanura]
        textoNombrePerfiles[numeroRanura].texto=nombrePerfiles[numeroRanura]
FIN PROCEDIMIENTO

PROCEDIMIENTO GuardarPerfil(Int numeroRanura) INICIAR
        SI (Archivo.Existe(“Ranura”+numeroRanura)) ENTONCES
                Archivo archivo=Archivo.Abrir(“Ranura”+numeroRanura)
                DatosSerializables datos=nuevo DatosSerializables();
                datos=archivo.LeerDatos()
        FIN SI
        SINO ENTONCES
                Archivo archivo=Archivo.Crear (“Ranura”+numeroRanura”)
                DatosSerializables datos=nuevo DatosSerializables();
                datos=archivo.LeerDatos()
        FIN SINO
        datos.nombrePerfiles =nombrePerfiles [numeroRanura]
        datos.nivelJugador =nivelJugador [numeroRanura]
        datos.experiencia =experiencia [numeroRanura]
        datos.nivel =nivel [numeroRanura]
        datos.partidasPerdidas =partidasPerdidas [numeroRanura]
        datos.aciertos =aciertos [numeroRanura]
        datos.rankings =rankings [numeroRanura]
        datos.banderaGuitarra =banderaGuitarra[numeroRanura]
        datos.indiceCalificacion =indiceCalificacion[numeroRanura]
        datos.numeroPartidas =numeroPartidas[numeroRanura]
        PARA  i=0 HASTA 10  HACER
                datos.calificaciones[i]=calificaciones[numeroRanura][i]
        FIN HACER
        archivo.Cerrar()
        FIN SI NO
FIN PROCEDIMIENTO
```

5.3.- Implementación del subsistema de Entrada de datos

El diagrama de clases que se usó para implementar el subsistema Entrada de datos se puede observar en la figura 5.14 donde se muestran todos los atributos y operaciones más importantes de cada clase.

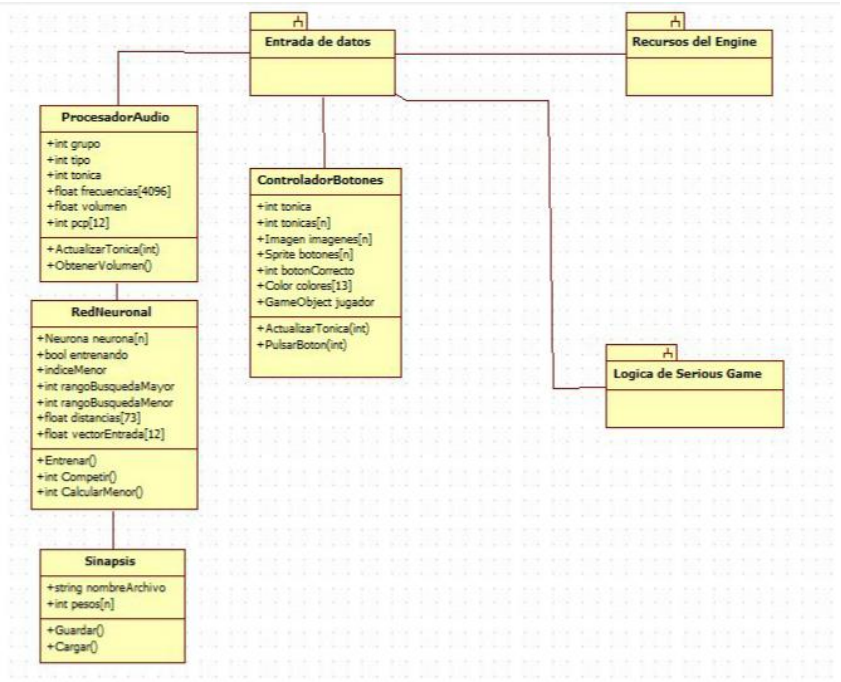


Figura 5.14 Subsistema "Entrada de datos" con clases

Clase	Descripción
ProcesadorAudio	Se encarga de aplicarle al audio de entrada la transformada discreta de Fourier (ya está implementada en el motor gráfico), también se encarga de ejecutar el algoritmo de PCP para generar el vector de entrada que va a reconocer la clase RedNeuronal, también tiene definidos los parámetros de grupo, tipo y tónica, para saber qué entrada se va a leer, si el grupo es 0 son notas si el grupo es igual a más de 1 son acordes, en ese caso si tipo es 0 son acordes mayores, si el tipo es 1 son acordes menores, si es 2 entonces son acordes en quinta y si es 3 son acordes en séptima, la variable tónica va a ser el resultado de la clase RedNeuronal.
Red Neuronal	En esta clase están programados los métodos y definidas las variables que

	constituyen el algoritmo de una red neuronal competitiva.
Sinapsis	En esta clase se guardan los pesos sinápticos de la red neuronal, esto sucede durante la fase de entrenamiento de la Red Neuronal, también se encarga de cargar todos los pesos modificados en el último entrenamiento cuando se ejecuta el programa.
ControladorBotones	Cuando no existe ningún micrófono en el ordenador o móvil o simplemente se escoge jugar sin guitarra, su función es recibir los mensajes de la clase AsignacionActividades y generar botones aleatorios de diferentes colores en los cuales en alguno de ellos está la respuesta correcta.

Tabla 5.2 Descripción de las clases del subsistema "Entrada de datos"

En este subsistema, las clases más relevantes son las de "RedNeuronal" y "ProcesadorAudio", en este tema se hará especial énfasis a los algoritmos que se usaron para la programación de la clase RedNeuronal, pues para el procesamiento del audio dependerá del programa que se esté utilizando para su desarrollo.

5.3.1.-Implementación del algoritmo PCP (Pitch Class Profiles)

En el capítulo anterior se mencionó que se debe escoger una representación apropiada para el audio de entrada, también se llegó a la conclusión de que la forma más adecuada para su representación serian vectores de dimensión 12. El resultado del algoritmo PCP se puede observar en la figura 5.15, es claro que al aplicarse la transformada discreta de Fourier se generaran cierto número de muestras que se guardan en el arreglo **Frecuencias**, este arreglo sirve de entrada para algoritmo PCP y su salida es un arreglo de tamaño 12 normalizado que representa la intensidad de las 12 notas de la escala cromática.

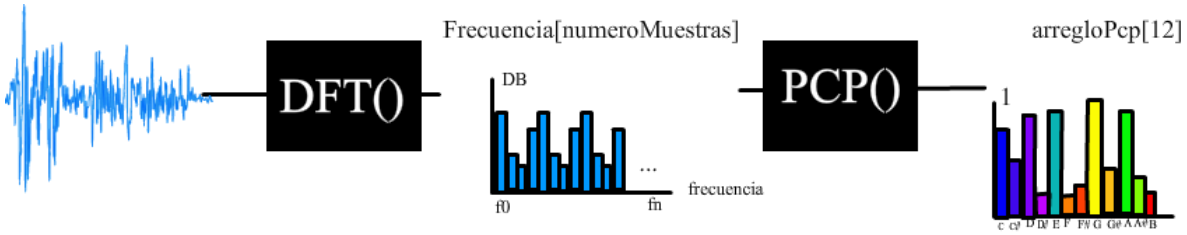


Figura 5.15 Entrada y salida del algoritmo PCP

Recordando las formulas descritas en el subtema 4.5.1.

$$PCP(p) = \sum_{M(l)=p} ||X(l)||^2$$

$$p = 0 \ldots \ldots 11$$

M(l) es una tabla que mapea una espiga del espectro de frecuencias al índice del vector **PCP**, ésta tabla puede ser inicializada como:

$$M(l) = \begin{cases} -1 & l \leq 0 \\ redondea(12log_2(f_s * \frac{1}{N})/f_{ref}) & l > 0 \ldots . N/2 - 1 \end{cases}$$

Aplicando las formulas se genera la tabla M (sin considerar valores negativos).

Octava	C =0	C#=1	D=2	D#=3	E=4	F=5	F#=6	G=7	G#=8	A=9	A#=10	B=11
1	33	35	37	39	41	44	46	49	52	55	58	62
2	65	69	73	78	82	87	92	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	831	880	932	988

Tabla 5.3 Tabla M

La función del algoritmo es mapear una frecuencia **l** al identificador de cada nota médiate fórmula:

$$M(l) = redondea(12log_2(f_s * \frac{1}{N})/f_{ref})$$

Es decir, si la frecuencia vale por ejemplo 110, el valor de **M(110)** será igual a 9 que corresponde a la nota *A*, esto significa que el valor en decibeles se le sumará a **PCP(9)**, cabe mencionar que se debe considerar que cuando no existe una frecuencia por ejemplo **l=112 M(112)** el resultado será -1, lo que significa que no se debe sumar al vector **PCP**, a pesar de que la definición matemática contempla todas las frecuencias, no considera que podrían haberse redondeado las frecuencias (como es el caso de este trabajo) ya que al ser índices de un arreglo son números enteros y esto podría dar a lugar que los de frecuencias sean -1 pues la función de logaritmo trabaja con variables de tipo flotante, otra cosa que no considera la fórmula es que la resolución podría ser diferente es decir que cada índice representa otra frecuencia, por ejemplo en el caso de este trabajo la resolución es de 5.85 ósea que el índice 110 no es la frecuencia 110 si no la 644.53, si no se tiene en cuenta lo anteriormente mencionado se podrían cometer errores, por eso se decidió implementar el algoritmo con una ligera variación.

El algoritmo que se trabajó resulta que cada elemento es el resultado de las octavas asociadas a la nota, matemáticamente podría verse como una sumatoria:

PCP(0)=$\sum_{i=1}^{i=5} espectro(Ci) = espectro(C1) + espectro(C2) + espectro(C3) + espectro(C4) + espectro(C5)$

.

.

.

PCP(11)=$\sum_{i=1}^{i=5} espectro(Bi) = espectro(B1) + espectro(B2) + espectro(B3) + espectro(B4) + espectro(B5)$

O lo que es lo mismo (apoyándose de la tabla **M**):

```
float resolución=24000/4096 // depende del IDE donde se esté programando
PARA  i=0 HASTA 11 HACER
        PARA o=0 HASTA  5 HACER
                PCP(i)+=espectro(M(o)(i)/resolucion)
        FIN HACER
FIN HACER
```

En la definición se consideró a PCP(i) como un vector que varía de i=0…11, las octavas ("C1","C2", etc.) representan a la frecuencia de **las octavas o el índice** del arreglo **espectro**, notar que solo se usaron cinco octavas para cada nota ya que el trabajo solo se centra a las octavas presentes en una guitarra acústica.

No hace falta ser músico ni matemático para darse cuenta que el resultado al aplicar el algoritmo PCP es mucho más sencilla que antes de aplicarse, gráficamente la salida del algoritmo PCP se puede representar como un histograma de 12 barras.

5.3.2.-Implementación del algoritmo de cuantificación vectorial (Learning Vector Quantization).

La función principal del algoritmo LVQ consiste combinar el aprendizaje no supervisado y supervisado para la clasificación de patrones, para el aprendizaje supervisado la red requiere de vectores ejemplos y el vector clase a la que pertenece dicho vector:

$$\{v1, c1\}, \{v2, c2\} \ldots.$$

Los vectores de entrada deben estar normalizados y las componentes de los vectores clase deben estar compuestas por 0 o 1, un ejemplo podría ser:

$$\{\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right), (1,0)\}$$

Antes del aprendizaje, cada neurona en la primera capa es asignada a una neurona de salida, esto genera la matriz W^2, el número de columnas de dicha matriz está en función del número de neuronas en la capa oculta y el número de renglones es el número de clases que el diseñador propuso, una vez definida nunca es modificada, existe otra matriz llamada W^1 cuyos valores serán modificados según la regla de Kohonen.

El proceso de aprendizaje LVQ es el siguiente:

1.-Se presenta el vector **p** a la red
2.- Por cada vector prototipo que se tenga se debe calcular la distancia euclidiana entre **p** y el vector prototipo, formando un nuevo vector llamado a^1
3.-Las neuronas ocultas compiten hasta que resulta una ganadora (i*)
4.-Al vector a^1 se le aplica una regla de competitividad
5.-El vector a^1 se multiplica por la matriz W^2 formando otro vector a^2
6.- Se aplica la regla de Kohonen si:

Si el vector **p** fue clasificado correctamente, los pesos de la neurona ganadora es actualizado entorno a dicho vector:

$$w^1(t)_{i*} = w^1(t-1)_{i*} + \alpha(p - w^1(t-1)_{i*})$$

Si el vector **p** fue clasificado incorrectamente, movemos los pesos de la neurona ganadora de forma que la alejamos de **p**:

$$w^1(t)_{i*} = w^1(t-1)_{i*} - \alpha(p - w^1(t-1)_{i*})$$

El algoritmo se ejecuta únicamente durante la fase de entrenamiento de la red, el resultado del entrenamiento son los vectores **w**, los cuales se van aproximando a los vectores de entrada que fueron clasificados correctamente, si uno es buen observador se dará cuenta que el grado de exactitud de este algoritmo depende de los vectores prototipos pues se usan para comparar la entrada y decir que tanto se parece a tal "vector prototipo" con base en la distancia euclidiana. Debido a que el algoritmo resultó confuso al programarlo, se propone otra forma para expresarlo de una forma más clara.

Si se va entrenar la red neuronal programar los siguientes pasos repitiéndolos durante un tiempo "t":

$$se\ presenta\ la\ tupla\ \{p, W^2(k)\}\ a\ la\ red\ neuronal$$

$$a^1 = competir \begin{pmatrix} ||p - W^1(0)|| \\ ||p - W^1(1)|| \\ . \\ . \\ ||p - W^1(W^1.longitud - 1)|| \end{pmatrix}$$

$$i = buscarIndiceMayorACero(a^1)$$

Si $k == i$ Entonces

$$W^1(k) = W^1(i) + \alpha\left(p - W^1(i)\right)$$

Sino entonces

$$W^1(k) = W^1(i) - \alpha\left(p - W^1(i)\right)$$

Una vez entrenada la red neuronal, el algoritmo para el reconocimiento de patrones es el siguiente:

$$se\ presenta\ el\ vector\ \boldsymbol{p}\ a\ la\ red\ neuronal$$

$$a^1 = competir\begin{pmatrix} \left|\left|p - W^1(0)\right|\right| \\ \left|\left|p - W^1(1)\right|\right| \\ \cdot \\ \cdot \\ \left|\left|p - W^1(W^1.longitud - 1)\right|\right| \end{pmatrix}$$

$$a^2 = W^2 a^1$$

El siguiente seudocódigo muestra de una forma más clara como se ve el algoritmo en un lenguaje de programación.

```
#MUESTRAS 4096
string notaAEntrenar
int indiceNotaEntrenar
float  W1[][]
float  W2[][]
float a1[]
float a2[]
int indiceMayorCero
int indiceMenor
float distancias[73]
float pcp[12]
float espectro[MUESTRAS]
float promedio
bool entrenar
int tónica
/*Multiplica un arreglo bidimensional por un arreglo unidimensional y regresa el producto en forma de arreglo unidimensional*/
float[] PROCEDIMIENTO Multiplicar(float W[][],float a[])  INICIO
        float producto[][]
        PARA i=0 HASTA i<w.longitud HACER
                PARA j=0 HASTA j<w[i].longitud HACER
                        PARA k=0 HASTA k<a.longitud HACER
                                producto[i][j]+=W[i][k]*a[k]
```

```
                    FIN HACER
                FIN HACER
            FIN HACER
FIN PROCEDIMIENTO
/*El método aplica el algoritmo PCP */
PROCEDIMIENTO ObtenerPcp() INCIAR
        espectro=DFT(Microfono.audio,MUESTRAS)
        float resolución=24000/4096
        PARA  i=0 HASTA 11 HACER
            PARA o=0 HASTA  5 HACER
                pcp(i)+=espectro(M(o)(i)/resolucion)
            FIN HACER
        FIN HACER
        pcp.Normalizar()
FIN PROCEDIMIENTO
/*Este método regresa el índice de la neurona del vector prototipo de una nota a partir de
una cadena de caracteres  */
Int PROCEDIMIENTO ObtenerIndiceNota(string nota) INICIAR
        string
notas={"C","C#","D","D#","E","F","F#","G","G#","A","A#","B","CM","C#M","DM","D#
M","EM","FM","F#M","GM","G#M","AM","A#M","BM","Cm","C#m","Dm","D#m","E
m","Fm","F#m","Gm","G#m","Am","A#m","Bm","C5","C#5","D5","D#5","E5","F5","F#5
","G5","G#5","A5","A#5","B5","C7","C#7","D7","D#7","E7","F7","F#7","G7","G#7","A7
","A#7","B7","Cm7","C#m7","Dm7","D#m7","Em7","Fm7","F#m7","Gm7","G#m7","Am
7","A#m7","Bm7","X"};
        SI (nota.logitud>0) ENTONCES
            PARA  i=0 HASTA i<notas.longitud HACER
                SI (nota.Igual (notas [i]))
                    Regresa i;
            FIN HACER
        FIN SI
        Regresa -1;
FIN PROCEDIMIENTO
/*Este método calcula las distancias entre los vectores de la matriz W1 y el arreglo
obtenido  al aplicar el algoritmo PCP*/
float[] PROCEDIMIENTO Competir() INICIAR
        PARA j=0 HASTA j<distancias.longitud HACER
            PARA i=0 HASTA i<pcp.longitud HACER
                distancias[j]+=⟦(pcp[i]-W1[j][i])⟧^2
            FIN HACER
            distancias[j]= √(distancias[j])
            a1[j]=distancias[j]
        FIN HACER
FIN PROCEDIMIENTO

Int PROCEDIMIENTO BuscarIndiceMayorACero(int a[])  INICIAR
        Int mayor=a[0]
```

```
        Int indiceMayor=0
        PARA  i=0 HASTA i<notas.longitud HACER
                SI (mayor<a[i]) ENTONCES
                        indiceMayor=i
                        mayor=a[i]
                FIN SI
        FIN HACER
        Regresa indiceMayor
FIN PROCEDIMIENTO
/*Este método como su nombre lo dice entrena a la red neuronal según el algoritmo
presentado anteriormente*/
PROCEDIMIENTO Entrenar() INICIAR
        SI (no(notaAEntrenar.Igual (""))) ENTONCES
                indiceNotaAEntrenar = ObtenerIndiceNota(notaAEntrenar);

                PARA  i=0 HASTA  i<12 HACER
                        SI (indiceMenor == indiceNotaAEntrenar) ENTONCES
                                W1 [indiceNotaAEntrenar] [i] += 0.4f * (-W1
[indiceNotaAEntrenar] [i] + pcp [i])
                                SI (indiceMenor != indiceNotaAEntrenar) ENTONCES
                                        W1 [indiceNotaAEntrenar] [i] += 0.4f * (-W1
[indiceNotaAEntrenar] [i] + pcp [i])
                                        W1 [indiceNotaAEntrenar] [i] += 0.4f * (W1
[indiceMenor] [i] - pcp [i])

                                FIN SI
                                Sinapsis.Guardar(W1)
                        FIN SI
        FIN HACER
FIN PROCEDIMIENTO
/*Se mandan a llamar los procedimientos ordenados según el algoritmo propuesto en la
página 73 */
LOOP
        pcp=ObtenerPCP()
        a1=Competir()
        a2=Multiplicar(W2,a1)
        tónica= BuscarIndiceMayorACero (a1)
        ProcesadorAudio.ActualizarTonica(tonica)
        Si (entrenar) ENTONCES
                Entrenar()
        FIN SI
FIN LOOP
```

5.4.-Implementación del subsistema Gestor de actividades

Para la implementación de este subsistema se apoyó del diagrama de clases mostrado en la figura 5.16.

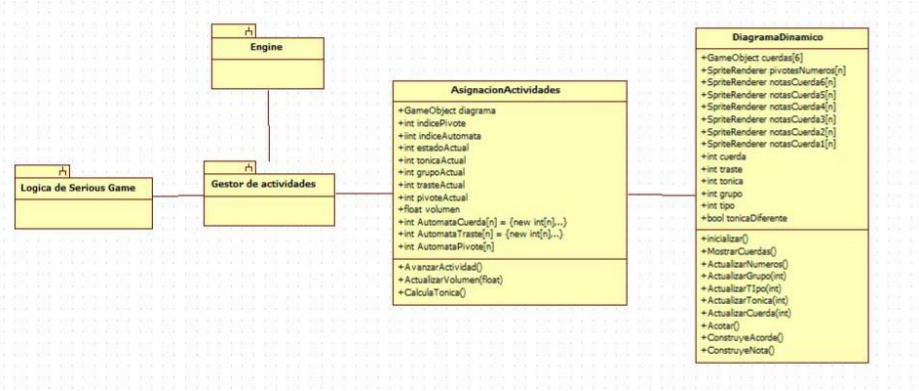


Figura 5.16 diagrama de clases del subsistema Gestor de actividades

Clase	Descripción
AsignacionActividades	Su función es albergar el contenido que se definió en el subtema 4.6.3 cambiando de nota o acorde según la lógica del juego
DiagramaDinamico	Su función es mostrar el diagrama de la nota o acorde actual según la clase "AsignacionActividades"

Tabla 5.4 Descripción de las clases del subsistema "Gestor de actividades"

De acuerdo al contenido y a las actividades que se diseñaron en el tema 4.6 se podría decir que debe existir una secuencia de notas o acordes y según las reglas descritas en el tema 5.1 donde se describe como se diseñaron los retos, no debe haber instancias con etiquetas diferentes y si el usuario acierta y destruye a todas las instancias en la escena se debe generar las instancias que corresponden a la siguiente nota o acorde, es decir, estamos hablando de un conjunto de autómatas, cada autómata representa un actividad diferente y el estado del autómata representara una nota o acorde.

Para diseñar la forma en que se programarían los autómatas, previamente se hizo un análisis en donde se tomó de referencia la forma en que se representaron las actividades.

Para representar una secuencia de notas usamos una tablatura:

1|----------0---1—3-----
2|----------0-1-3----------
3|----------0-2----------
4|----------0—2-3----------
5|----------0---2-3----------
6|-----0—1—3----------

Para representar una secuencia de acordes usamos la notación americana:

CM, Fm, G5, A7

Recordando un poco las partes de una tablatura, tenemos que está conformada por los dígitos que están hasta el inicio que representan el número de cuerda de la guitarra, después tenemos los dígitos que están entre la línea punteada que representan los trastes, por otra parte los acordes son un conjunto de notas y hay varios tipos de acordes: mayores, menores, en quinta, novena, en séptima, en cuarta etc. A pesar de que los acordes son un conjunto de notas y las notas solo son una en sí misma tienen algo en común y es que ambos se pueden expresar en función de su tónica o nota principal, por ejemplo a la tablatura que se muestra en la parte inferior se le puede aplicar una función que obtenga la tónica a partir del traste y la cuerda es decir nota=CalculaTonica(6,0)=3, para el caso de los acordes basta con declarar un parámetro adicional que nos diga qué tipo de acorde se trata.

En resumen tenemos cuatro diferentes conjuntos de autómatas que representarán a las actividades y un autómata adicional que solo se usara para el diagrama dinámico, un conjunto controlará las tónicas de cada actividad, otro los trastes, otro las cuerdas y otro el tipo de acorde y el autómata adicional controlará en donde empieza el diagrama, a este autómata le llamamos "autómata de pivotes", también se usara un parámetro que se nombró como "grupo" que sirve para decirnos si la tónica pertenece a una nota, acorde o escala. La definición de los autómatas nombrados se presenta a continuación.

Autómata para las tónicas:

El conjunto de estados es el siguiente:

$$Q = \{tonica0, tonica1, \ldots\ldots, tonican\}$$

El estado inicial siempre será q0=tonica0

Y el conjunto de los estados finales solo estará definido por un solo elemento y siempre será:

F={tonican}

El alfabeto está definido por:

$$\sum = \{0,1,2,3,4,5,6,7,8,9,10,11\}$$

Donde cada número representa en general todos los acordes y todas las notas posibles, este alfabeto puede variar conforme al autómata de tipo y a la variable grupo, de esta forma se reduce el alfabeto de 60 (12 por cada nota y 12 por cada tipo de acorde) a 12 elementos. En la tabla 5.5 se muestra que nota, acorde o escala se está representando por algún número definido en el alfabeto en función de la variable tipo y grupo.

Grupo	Tipo	tónica	Representación
0	X	0	Nota de C
0	X	1	Nota de C#
0	X	2	Nota de D
0	X	3	Nota de D#
0	X	4	Nota de E
0	X	5	Nota de F
0	X	6	Nota de F#
0	X	7	Nota de G
0	X	8	Nota de G#
0	X	9	Nota de A
0	X	10	Nota de A#
0	X	11	Nota de B
1	0	0	Acorde mayor C
1	0	1-10	Acorde mayor C#-A#
1	0	11	Acorde mayor B
1	1	0-11	Acorde menor C-B
1	2	0-11	Acorde quinta C-B
1	3	0-11	Acorde séptima C-B
2	0	0-11	Escala jónica(mayor) C-B
2	1	0-11	Escala dórica C-B
2	2	0-11	Escala frigia C-B
2	3	0-11	Escala lidia C-B
2	4	0-11	Escala mixolidia C-B
2	5	0-11	Escala eólica (menor)C-B
2	6	0-11	Escala locria C-B
2	7	0-11	Pentatónica menor C-B
2	8	0-11	Pentatónica mayor C-B

Tabla 5.5 Nota, acorde o escala en función de la variable grupo y tipo

La función de transición es la siguiente:

$$\delta(\mathrm{i}, \mathrm{tonicaj}) = \mathrm{tonicak}$$

Donde **i** representa el identificador de un elemento del alfabeto, **tonicaj** es la salida de la clase "ProcesadorAudio" y **tonicak** es el siguiente estado.

Autómata para las cuerdas:

*El alfabeto representa el número de cuerda.

$$Q = \{cuerda0, cuerda1, \ldots\ldots, cuerdan\}$$

$$\sum = \{1,2,3,4,5,6\}$$

$$\delta(\mathrm{i}, \mathrm{cuerdaj}) = \mathrm{cuerdak}$$

q0=cuerda0

F={cuerdan}

Autómata para el control de trastes:

*El alfabeto son los números del traste en el mástil de la guitarra.

$$Q = \{traste0, traste1, \ldots\ldots, trasten\}$$

$$\sum = \{0,1,2,3,4,5,6,7,8,9,10,11,12\}$$

$$\delta(\mathrm{i}, \mathrm{trastej}) = \mathrm{trastek}$$

q0=traste0

F={trasten}

Autómata para los tipos de acorde:

*El alfabeto son los números de tipo de acorde.

$$Q = \{tipo0, tipo1, \ldots\ldots, tipon\}$$

$$\sum = \{0,1,2,3\}$$

$$\delta(\mathrm{i}, \mathrm{tipoj}) = \mathrm{tipok}$$

q0=tipo0

F={tipon}

Autómata para los pivotes del diagrama dinámico:

El alfabeto son todos los posibles pivotes que se pueden usar.

$$Q = \{pivote0, pivote1, \ldots\ldots, pivoten\}$$

$$\sum = \{0,1,2,3,4,5,6,7,8,9,10,11,12\}$$

$$\delta(\mathrm{i}, \mathrm{pivotej}) = \mathrm{pivotek}$$

q0=pivote0

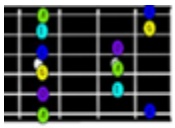
F={pivoten}

Con los autómatas definidos se podrá implementar cualquier tablatura o sucesión de acordes de cualquier actividad, estableciendo el número de actividad por el número de autómata, es decir, el autómata de cuerdas, pivotes y tónicas número 0 será el que represente la actividad P01-01, el autómata de cuerdas, pivotes y tónicas número 81 será el que represente la actividad A01-00 (ver tabla 4.7),

A continuación se ofrecen un par de ejemplos que muestran cómo se declararon los autómatas de las actividades P01-08 y A05-00 definidas en la tabla 4.7.

Para la actividad P0-08:

El diagrama base de la actividad es el siguiente



Y el diagrama de la actividad es:

1|-----------------------------------
2|----------5---------------------------
3|-------5-----5-------5---------8----
4|-------------------7-----------------
5|-----------------------------------
6|-----------------------------------

Los autómatas para representar la actividad son los siguientes:

```
int[][] AutomatasCuerda=new int[][] {
                .
                .
                .
                // P01-08
                new int[] {3,2,3,4,3,3},
                .
                .
                .
}

int[][] AutomatasTraste=new int[][] {
                .
                .
                .
                // P01-08
                new int[] {5,5,5,7,5,8},
                .
                .
                .
}
int[] AutomataPivotes=new int[] {…..,5,….}
```

Para la actividad A05-00:

La actividad es la siguiente:

Dm-G-C

Los autómatas para representar la actividad serán los siguientes:

```
int[][] AutomatasTonica=new int[][] {
                .
                .
                .
                // A05-00
                new int[] {2,7,0},
                .
                .
                .
}

int[][] AutomatasTipo=new int[][] {
                .
                .
                .
```

```
// A05-00
new int[] {1,0,0},
.
.
.
```

}

5.4.2.-Implementación del diagrama dinámico

Para la implementación del diagrama dinámico primero se identificó que elementos y restricciones debería tener, para ello se hizo la siguiente lista:

- Debe tener 6 cuerdas
- Máximo debe tener 4 trastes
- Debe mostrar información del número de los trastes a partir de una referencia por ejemplo si la referencia (pivote) empieza en 5 los siguientes deben ser 6, 7 y 8
- Cuando se va a tocar una nota debe mostrar la posición exacta de la cuerda y del traste donde se encuentra la nota en el mástil de la guitarra
- Cuando se trata de un acorde se muestran varias notas localizadas en diferentes trastes y cuerdas

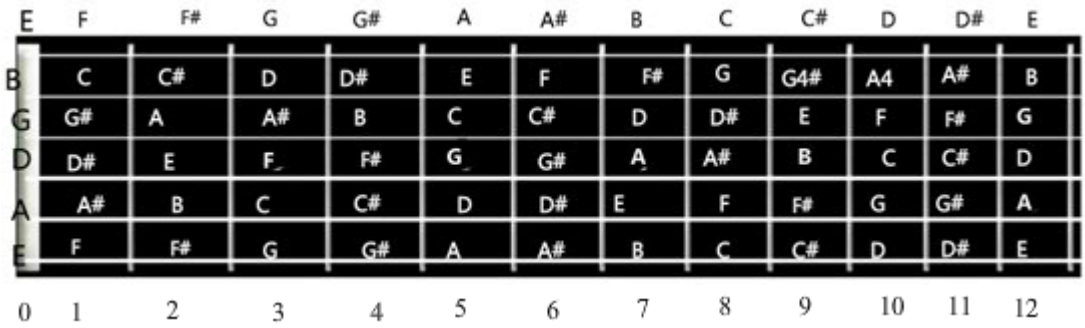


Figura 5.17 Mástil de una guitarra con notas

Si observamos el diagrama nos podemos dar cuenta que a partir del traste número 12 las notas se empiezan a repetir, otra cosa que se puede observar es que la primera nota de entre cada cuerda está desfasada 5 semitonos de la cuerda anterior, es decir, en el traste 5 de la cuerda 6 la nota es la misma que aparece en la primer nota de la cuerda 5 y esto se cumple para todas las cuerdas excepto en la cuerda 3, si tomamos como referencia la escala mayor de C se podría decir que en la primer y sexta cuerda la escala mayor de C está desfasada 4 semitonos, en la quinta cuerda está desfasada 9 semitonos, en la cuarta 2 semitonos en la tercera 7 semitonos y en la segunda 10 semitonos.

Entonces para diseñar el diagrama dinámico se decidió representar cada elemento del mástil por una variable, para representar las notas que contiene cada traste se creó un arreglo

de tipo entero para cada cuerda, el valor de la variable pivote funciona como una referencia por ejemplo si el pivote vale 1 se van a mostrar los trastes 1, 2, 3 y 4, si el pivote vale 2 se mostraran los trastes 2, 4, 5 y 6. Los elementos de un arreglo **cuerda** tendrán como valor 0 o 1 que dependen del pivote y de una función que le llamaremos "FuncionMapeadora", un 1 representa que en el diagrama se va a mostrar una imagen de la nota exacta que se desea tocar, un 0 representa que no se muestra nada.

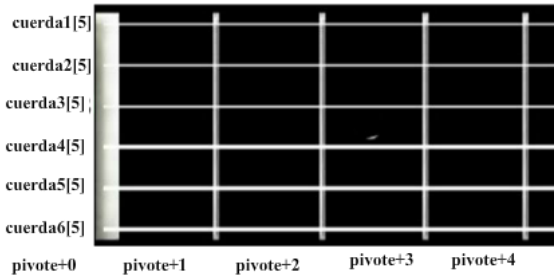


Figura 5.18 Diagrama del mástil recortado

0	1	2	3	4	5	6	7	8	9	10	11
C	C#	D	D#	E	F	F#	G	G#	A	A#	B

Tabla 5.6 Notas con identificadores

Como el diagrama dinámico está ligado con la clase "AsignacionActividades" se usarán las mismas variables que usan los autómatas y tendrá la misma representación según el valor de las variables de la tabla 5.4.

0	1	2	3	4	5	6	7	8	9	10	11	12
D	D#	E	F	F#	G	G#	A	A#	B	C	C#	D

Tabla 5.7 Escala de D mayor con identificador

0	1	2	3	4	5	6	7	8	9	10	11	12
E	F	F#	G	G#	A	A#	B	C	C#	D	D#	E

Tabla 5.8 Escala de E mayor con identificador

0	1	2	3	4	5	6	7	8	9	10	11	12
C	C#	D	D#	E	F	F#	G	G#	A	A#	B	C

Tabla 5.9 Escala de C mayor con identificador

En las tablas 5.7 y 5.8 se observa que se movió **la nota tónica de la escala** al índice 0 para generar la escala de **D** en el primer caso y la escala de **E** en el segundo caso y eso provocó que se movieran las demás notas de posición, sin embargo, los identificadores se quedaron en el mismo lugar, si generalizamos el comportamiento para generar las demás escalas nos podemos dar cuenta rápidamente que se comporta como si fuera un registro de corrimiento a la izquierda o derecha**,** depende de donde se localice la nota tónica, para verlo de una forma más sencilla tomaremos de referencia la escala mayor de C (tabla 5.9) y diremos que la nota tónica está representada por el identificador correspondiente (por ejemplo el identificador para la tónica de la escala A será el 9) y le sumaremos la tónica más el grado (concepto definido en el subtema 3.2.2) de la nota que conforma alguna escala o acorde (se dará un ejemplo más adelante), cabe mencionar que hay que tener cuidado ya que el arreglo tiene solo 12 elementos y no se puede sobrepasar, lo que estamos haciendo con esto es recorrer las notas tantas veces como sea necesario para obtener el tono deseado. Para programar más cómodamente se generaron las tablas 5.10 y 5.11 a partir de las tablas 3.3 y 3.4 (construcción de acordes y escalas), la única diferencia es que se sustituyeron los grados por los identificadores según la tabla 5.9.

Por ejemplo, imaginemos que queremos construir el acorde mayor de D, la tónica tendrá el valor de tónica=2 (recordemos que nuestro sistema de referencia es la escala mayor de C), la fórmula para construir un acorde mayor es 0-4-7 (ver tabla 5.9), entonces el conjunto con que se construye el acorde A mayor será la suma del número que conforma la fórmula más la tónica, es decir {0+2,4+2,7+2}, que es lo mismo que {2,6,9}={D,F#,A}, para demostrar que es válida esta construcción observar la tabla 5.7 que corresponde a la escala mayor D, se toma la formula 0-4-7 y se sustituyen los números por las notas según el identificador del conjunto de números de la tabla 5.7 y el resultado es:{0,4,7}={D,F#,A}.

Tipo de acorde	**Formula**
Acorde mayor	0-4-7
Acorde menor	0-3-7
Acorde en quinta	0-7
Acorde en séptima	0-4-7-11

Tabla 5.10 Formulas para generar acordes con identificadores

Tipo de escala	**Formula**
Modo Jónico, Escala Mayor	0-2-4-7-9-11
Modo Dórico	0-2-3-5-7-9-10
Modo Frigio	0-1-3-5-5-8-10
Modo Lidio	0-2-4-6-7-8-10
Modo Mixolidio	0-2-4-5-7-9-10
Modo Eólico, Escala Menor	0-2-3-5-7-8-10
Modo Locrio	0-1-3-5-6-8-10
Pentatónica Menor	0-3-5-7-10
Pentatónica Mayor	0-2-4-7-9

Tabla 5.11 Tabla con fórmulas para generar escalas con identificadores

Una vez aclaradas las cosas, el seudocódigo es uno de los mas grandes y es el siguiente:

```
int vectorBinario[12] // este arreglo almacena valores de 0 o 1 según las notas que componga al acorde o escala (si es una nota solo tendrá un elemento en 1)
Text textoTraste[6]
Int grupo // las variables grupo, tipo, tónica, cuerda y pivote son proporcionadas por la clase "AsignacionActividades"
Image imagenNotas[12] // tiene las imágenes a cada nota correspondiente
Int tipo
Int tonica
Int cuerda
Int pivote
/*Este método sirve para calcular el índice de la nota perteneciente a alguna escala o acorde según su tónica y según su posición en la escala mayor de C (su grado) */
int PROCEDIMIENTO Acota( tonica, grado) INICIAR
        SI (tonica + grado > 11 ENTONCES
                regresa |tonica + grado – 12|
        SI NO
                regresa tonica + grado;

FIN PROCEDIMIENTO
/*Este método calcula todas las notas pertenecientes a una escala o acorde representando cada nota perteneciente con un valor de 1 en la posición exacta de dicha nota en el arreglo "n" */
int[] PROCEDIMIENTO FuncionMapeadora(int grupo,int tipo,int tonica) INCIIAR

        int n[]=null

        textoTraste [0].text = (pivote)
        textoTraste [1].text = (pivote+1)
        textoTraste [2].text = (pivote+2)
        textoTraste [3].text = (pivote+3)
        textoTraste [4].text = (pivote+4)
        SI (grupo == 0) ENTONCES //NOTAS
                n[Acota(tonica,0)]=1
        SI (grupo == 1) ENTONCES
                        SI(tipo==0)ENTONCES//MAYORES
                                n[Acota(tonica,0)]=1
                                n[Acota(tonica,4)]=1
                                n[Acota(tonica,7)]=1
                        FIN SI
                        SI(tipo==1)ENTONCES//MENORES
                                n[Acota(tonica,0)]=1
```

```
                n[Acota(tonica,3)]=1
                n[Acota(tonica,7)]=1
        FIN SI
        SI(tipo==2)ENTONCES//QUINTA
                n[Acota(tonica,0)]=1
                n[Acota(tonica,7)]=1

        FIN SI
        SI(tipo==3)ENTONCES//SEPTIMA
                n[Acota(tonica,0)]=1
                n[Acota(tonica,4)]=1
                n[Acota(tonica,7)]=1
                n[Acota(tonica,10)]=1

        FIN SI
FIN SI
SI (grupo == 2) ENTONCES
        SI(tipo==0)ENTONCES//JONICO
                n[Acota(tonica,0)]=1
                n[Acota(tonica,2)]=1
                n[Acota(tonica,4)]=1
                n[Acota(tonica,5)]=1
                n[Acota(tonica,7)]=1
                n[Acota(tonica,9)]=1
                n[Acota(tonica,11)]=1


        FIN SI
        SI(tipo==1)ENTONCES//DORICO
                n[Acota(tonica,0)]=1
                n[Acota(tonica,2)]=1
                n[Acota(tonica,3)]=1
                n[Acota(tonica,5)]=1
                n[Acota(tonica,7)]=1
                n[Acota(tonica,9)]=1
                n[Acota(tonica,10)]=1


        FIN SI
        SI(tipo==2)ENTONCES//FRIGIO
                n[Acota(tonica,0)]=1
                n[Acota(tonica,1)]=1
                n[Acota(tonica,3)]=1
```

```
        n[Acota(tonica,5)]=1
        n[Acota(tonica,7)]=1
        n[Acota(tonica,8)]=1
        n[Acota(tonica,10)]=1

FIN SI
SI(tipo==3)ENTONCES//LIDIO
        n[Acota(tonica,0)]=1
        n[Acota(tonica,2)]=1
        n[Acota(tonica,4)]=1
        n[Acota(tonica,6)]=1
        n[Acota(tonica,7)]=1
        n[Acota(tonica,9)]=1
        n[Acota(tonica,11)]=1
FIN SI
SI(tipo==4)ENTONCES//MIXOLIDIO
        n[Acota(tonica,0)]=1
        n[Acota(tonica,2)]=1
        n[Acota(tonica,4)]=1
        n[Acota(tonica,5)]=1
        n[Acota(tonica,7)]=1
        n[Acota(tonica,9)]=1
        n[Acota(tonica,10)]=1
FIN SI
SI(tipo==5)ENTONCES//EOLICO-MENOR
        n[Acota(tonica,0)]=1
        n[Acota(tonica,2)]=1
        n[Acota(tonica,3)]=1
        n[Acota(tonica,5)]=1
        n[Acota(tonica,7)]=1
        n[Acota(tonica,8)]=1
        n[Acota(tonica,10)]=1
FIN SI
SI(tipo==6)ENTONCES//LOCRIO
        n[Acota(tonica,0)]=1
        n[Acota(tonica,1)]=1
        n[Acota(tonica,3)]=1
        n[Acota(tonica,5)]=1
        n[Acota(tonica,6)]=1
        n[Acota(tonica,8)]=1
        n[Acota(tonica,10)]=1
```

```
                    FIN SI

                    SI(tipo==7)ENTONCES//PENTATONICA MENOR
                        n[Acota(tonica,0)]=1
                        n[Acota(tonica,3)]=1
                        n[Acota(tonica,5)]=1
                        n[Acota(tonica,7)]=1
                        n[Acota(tonica,10)]=1
                    FIN SI
                    SI(tipo==8)ENTONCES//PENTATONICA MAYOR
                        n[Acota(tonica,0)]=1
                        n[Acota(tonica,2)]=1
                        n[Acota(tonica,4)]=1
                        n[Acota(tonica,7)]=1
                        n[Acota(tonica,9)]=1
                    FIN SI
                FIN SI
FIN PROCEDIMIENTO
/*Este método dibuja en un diagrama de cuatro trastes las posiciones exactas de las notas
que pertenecen a un acorde o escala o de la misma nota individual*/
PROCEDIMIENTO ActualizarDiagrama() INICIAR
        vectorBinario=FuncionMapeadora(grupo,tipo,tonica)

        SI (grupo==0) ENTONCES // es una nota
            SI (cuerda==1) ENTONCES
                cuerda1.Activar()
            SINO ENTONCES
                cuerda1.Desactivar()
            SI (cuerda==2) ENTONCES
                cuerda2.Activar()
            SINO ENTONCES
                cuerda2.Desactivar()
            SI (cuerda==3) ENTONCES
                cuerda3.Activar()
            SINO ENTONCES
                cuerda3.Desactivar()
            SI (cuerda==4) ENTONCES
                cuerda4.Activar()
            SINO ENTONCES
                cuerda4.Desactivar()
            SI (cuerda==5) ENTONCES
                cuerda5.Activar()
            SINO ENTONCES
```

```
                cuerda5.Desactivar()
        SI (cuerda==6) ENTONCES
                Cuerda6.Activar()
        SINO ENTONCES
                Cuerda6.Desactivar()

FIN SI
SINO ENTONCES //es una escala o un acorde
        cuerda1.Activar()
        cuerda2.Activar()
        cuerda3.Activar()
        cuerda4.Activar()
        cuerda5.Activar()
FIN SINO
PARA i=0 HASTA 4 HACER
        SI (vectorBinario[Acota(pivote,4+i)]==1 y cuerda==6) ENTONCES
                cuerda6[i].imagen=imagenNotas[Acota(pivote,4+i)]
        FIN SI
        SINO ENTONCES
                cuerda6[i].imagen=imagenNotas[12]
        FIN SINO
         Si (vectorBinario[Acota(pivote,9+i)]==1 y cuerda==5) ENTONCES
                cuerda5[i].imagen=imagenNotas[Acota(pivote,9+i)]
        FIN SI
        SINO ENTONCES
                cuerda5[i].imagen=imagenNotas[12]
        FIN SINO
        SI (vectorBinario[Acota(pivote,2+i)]==1 y cuerda==4) ENTONCES
                Cuerda4[i].imagen=imagenNotas[Acota(pivote,2+i)]
        FIN SI
        SINO ENTONCES
                cuerda4[i].imagen=imagenNotas[12]
        SINO ENTONCES
        SI (vectorBinario[Acota(pivote,7+i-temp)]==1 y cuerda==3) ENTONCES
                Cuerda3[i].imagen=imagenNotas[Acota(pivote,7+i)]
        FIN SI
        SINO ENTONCES
                cuerda3[i].imagen=imagenNotas[12]
        FIN SINO
        SI (vectorBinario[Acota(pivote,11+i)]==1&&cuerda==2) ENTONCES
                        Cuerda2[i].imagen=imagenNotas[Acota(pivote,11+i)]
        FIN SI
        SI NO ENTONCES
```

```
            cuerda2[i].imagen=imagenNotas[12]
        FIN SINO
        SI (vectorBinario[Acota(pivote,4+i)]==1 y cuerda==1) ENTONCES
            Cuerda1[i].imagen=imagenNotas[Acota(pivote,4+i)]
        FIN SI
        SINO ENTONCES
            cuerda1[i].imagen=imagenNotas[12]
        FIN SINO
FIN PROCEDIMIENTO
```

El resultado visual al programar el seudocódigo se puede observar en la figura 5.17, el algoritmo sirve para mostrar notas individuales que pertenecen a una escala, así como para representar acordes y las posiciones de las escalas.

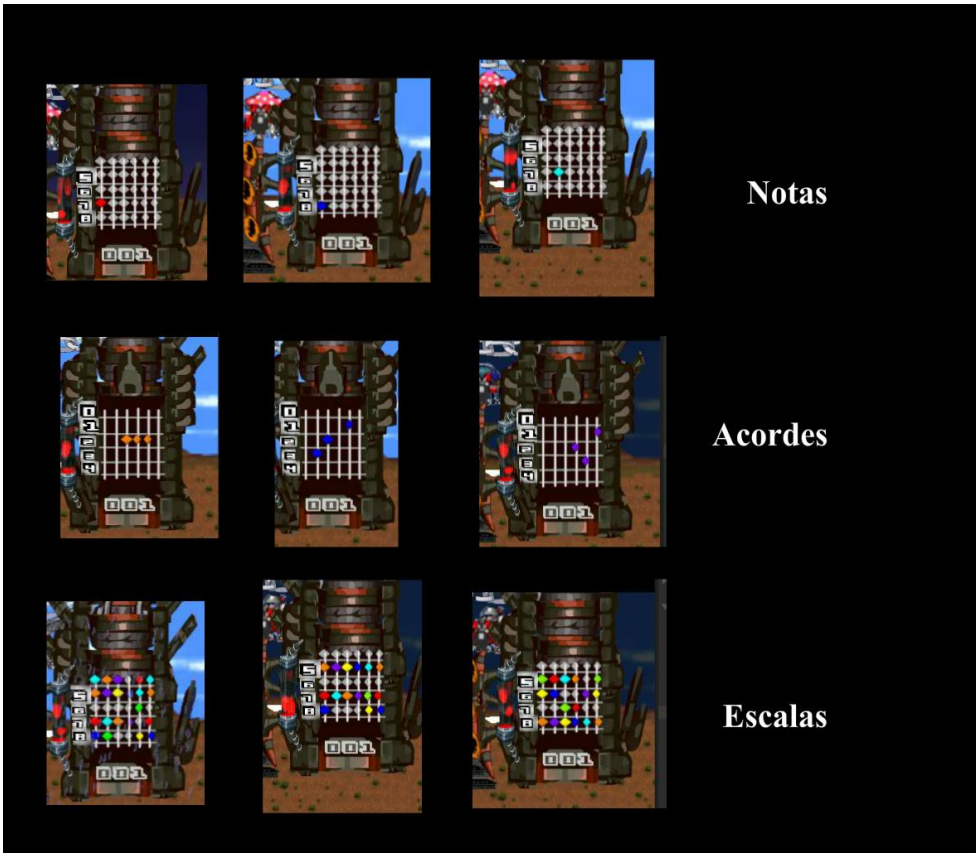


Figura 5.19 Algoritmo en ejecución

5.5.-Implementación del subsistema de monitorización

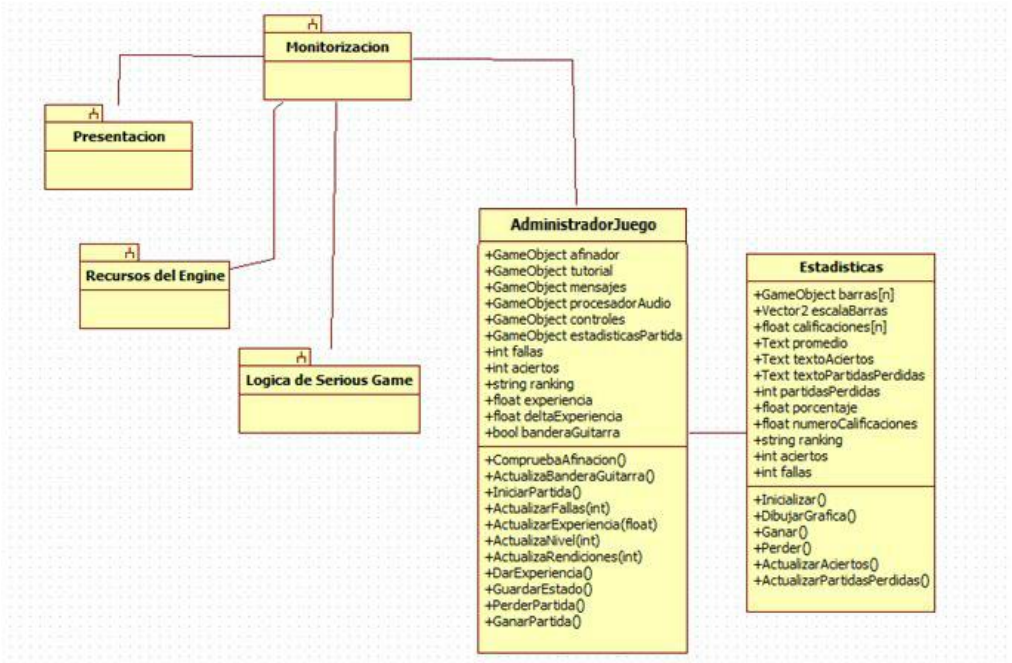


Figura 5.20 Diagrama del subsistema "Monitorización" con clases

Clase	Descripción
AdministradorJuego	Cumple la función de recibir todos los mensajes de los objetos en escena y ejecutar los métodos adecuados para mantener coherencia con la lógica del juego
Estadisticas	Tiene como función principal dar una realimentación al usuario apoyándose de los recursos del subsistema de "Presentación"

Tabla 5.12 Descripción de las clases del subsistema de "Monitorización"

El subsistema de monitorización consiste en dos clases las cuales tienen como función principal monitorear lo que está pasando en el subsistema "Lógica del serious game" durante la ejecución del programa, estas clases reciben mensajes de los demás objetos y van actualizando el valor de sus variables a lo largo del tiempo, en el diagrama de colaboración mostrado en la figura 5.19 se muestra algunos de estos mensajes.

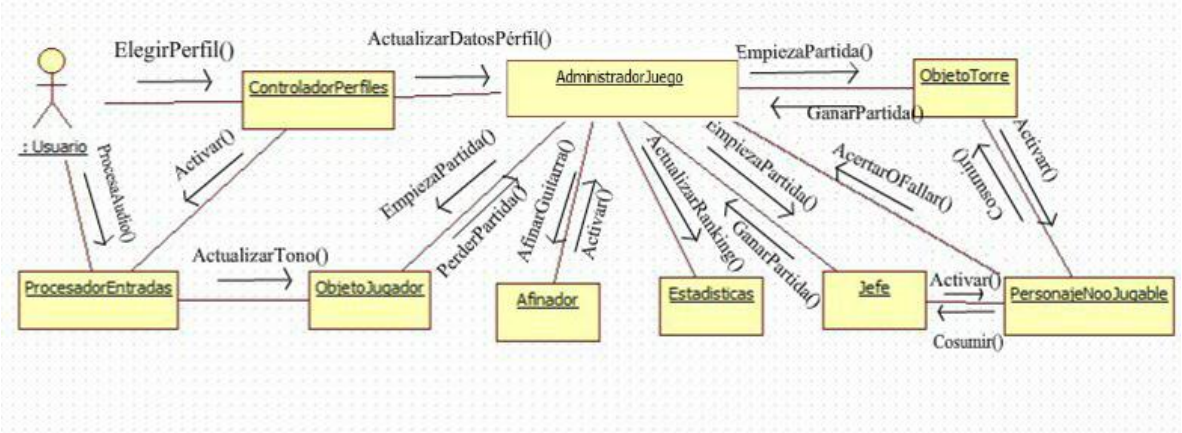


Figura 5.21 Diagrama de colaboración

Es fácilmente notar la importancia de este subsistema ya que sin la clase "AdministradorJuego" se romperían todas las reglas del juego y sería un desastre todo el sistema, no se podrían contabilizar aciertos ni se podría avanzar entre los niveles, también se puede observar en la figura 5.21 que sirve como moderador para activar o desactivar ciertos objetos, así como de recibir y mostrar información.

5.5.1.-Implementación del gestor del juego

El seudocódigo que se muestra a continuación muestra los métodos más importantes los cuales reciben los mensajes de otros objetos, también tiene referencias a otros objetos de la escena para mandarles mensajes.

```
GameObject afinador
GameObject tutorial
GameObject mensajes
GameObject procesadorAudio
GameObject controles
GameObject estadisticasPartida
int fallas
int aciertos
int rendiciones
string ranking
float experiencia
float deltaExperiencia
bool banderaGuitarra
/* Los procedimientos que tienen de prefijo "Actualizar" sirven para recibir mensajes los demás su mismo nombre explica la función que va a realizar*/
PROCEDIMIENTO CompruebaAfinacion(int bandera) INICIAR
        SI (bandera==1) ENTONCES
                Brodcast.EmpezarPartida()
        FIN SI
        SINO
```

```
            afinador.Activar()
        FIN SINO

FIN PROCEDIMIENTO

PROCEDIMIENTO ActualizarBanderaGuitarra(bool bandera)  INICIAR
        banderaGuitarra=bandera
FIN PROCEDIMIENTO

PROCEDIMIENTO IniciarPartida() INICIAR
        Broadcast.IniciarPartida()
FIN PROCEDIMIENTO

PROCEDIMIENTO ActualizarFallas(int f) INICIAR PROCEDIMIENTO
        fallas=f
FIN PROCEDIMIENTO

PROCEDIMIENTO ActualizarExperiencia(int e) INICIAR PROCEDIMIENTO
        experiencia=e
        jugador.ActualizarExperiencia(e)
FIN PROCEDIMIENTO

PROCEDIMIENTO ActualizarNivel(int n) INICIAR PORCEDIMIENTO
        nivel=n
        jugador.ActualizarNivel(n)
FIN PROCEDIMIENTO

PROCEDIMIENTO ActualizarRendiciones(int x) INICIAR PROCEDIMIENTO
        rendiciones=x
FIN PROCEDIMIENTO

PROCEDIMIENTO RecibeExperiencia() INICIAR PROCEDIMIENTO
        experiencia´+=deltaExperiencia
        SI  experiencia>=100 ENTONCES
                experiencia=0
                nivelJugador+=1
                mensaje.texto=”Subio de nivel!”
                mensaje.Activar(4segundos)
                jugador.ActualizarNivel(nivelJugador)
        FIN SI

FIN PROCEDIMIENTO

PROCEDIMIENTO AlmacenarEstado() INICIAR PROCEDIMIENTO
        float rank= aciertos/(aciertos+fallas)
        controladorPartidas. ActualizaNivelJugador"(nivelJugador)
        controladorPartidas. ActualizaExperiencia(experiencia)
```

```
        controladorPartidas.ActualizaPartidasPerdidas(partidasPerdidas);
        controladorPartidas.ActualizaAciertos(aciertos);
        estadísticas.Activar()
        aciertos=0
        fallas=0

FIN PROCEDIMIENTO

PROCEDIMIENTO PerderPartida() INICIAR PROCEDIMIENTO
        controles.Desactivar()
        procesadorAudio.Desactivar()
        partidasPerdidas++
        estadísticas.Perder()
        pantallaPerderJuego.ActualizarAciertos(aciertos)
        pantallaPerderJuego.ActualizanRaking(rank)
        pantallaPerderJuego.ActualizaPartidasPerdidas(partidasPerdidas)
        controladorPartidas.GuardarPerfil(-1)
        AlmacenarEstado()
FIN PROCEDIMIENTO


PROCEDIMIENTO GanarJuego() INICIAR PROCEDIMIENTO
        controles.Desactivar()
        procesadorAudio.Desactivar()
        jugador.AvanzarNivel()
        estadísticas.Ganar()
        pantallaPerderJuego.ActualizarAciertos(aciertos)
        pantallaPerderJuego.ActualizanRaking(rank)
        pantallaPerderJuego.ActualizaPartidasPerdidas(partidasPerdidas)
        controladorPartidas.GuardarPerfil(-1)
        AlmacenarEstado()
FIN PROCEDIMIENTO
```

El seudocódigo anterior se puede resumir en funciones que van a llamar los demás objetos quienes a través de los parámetros va actualizando el valor de las variables de la clase "AdministradorJuego".

5.5.2.-Estadísticas del usuario

El objeto estadísticas tiene como función mostrar mediante la interfaz de usuario información clave de ciertos parámetros que se modificaron durante una partida, la programación se hizo basándose en el siguiente seudocódigo.

```
GameObject estadisticasPartida //esta variable es objeto de interfaz que contiene los objetos "barras" y "texto promedio"
GameObject estadisticiasJuego // es un objeto de interfaz que contiene a todos los objetos tipo Text
```

```
GameObject[] barras // son objetos visuales que representan las barras de un histograma
Vector2[] escalaBarras
Text textoPromedio
Text textoAciertos
Text textoFallas
Text textoPartidasPerdidas
Text textoRank
int partidasPerdidas
float porcentaje
float numeroCalificaciones
string Rank
int ciertos
int fallas
float califica
float promedio

PROCEDIMIENTO Inicializar() INICIAR
        PARA CADA escala en barras.escala HACER
                escalaBarras[i]=escala
        FIN HACER
FIN PROCEDIMIENTO
/*Dibuja un histograma según basado en el desempeño del usuario*/
PROCEDIMIENTO DibujarGrafica() INICIAR
        promedio=0
        estadisticasJuego.Activar()
        numeroCalificaciones=0
        PARA i=0 HASTA i<calificaciones.longitud HACER
                SI (calificaciones[i]>0) ENTONCES

        barras.escala=Vector2(escalaBarra[i].x,escalaBarra[i].y*calificaciones[i])
                        promedio+=calificaciones[i]
                        numeroCalifiaciones++
                FIN SI
                SI  (numeroCalificaciones > 0) ENTONCES
                        promedio /= numeroCaIicaciones

                        SI  (promedio >= 1) ENTONCES
                                rank = "A++"
                        FIN SI
                        SI  (promedio >= 0.8 && promedio < 1) ENTONCES
                                rank = "A+"
                        FIN SI
                        SI (promedio >= 0.6 && promedio < 0.8) ENTONCES
                                rank = "A"
                        FIN SI
                        SI (promedio >= 0.5 && promedio < 0.6) ENTONCES
                                rank = "B"
```

```
                    FIN SI
                    SI (promedio >= 0.3 && promedio < 0.5) ENTONCES
                        rank = "C"
                    FIN SI
                    SI (promedio >= 0.2 && promedio < 0.3) ENTONCES
                        rank = "D"
                    FIN SI
                    SI (promedio >= 0 && promedio < 0.2) ENTONCES
                        rank = "E"
                    FIN SI
                    SI (promedio <= 0) ENTONCES
                        rank = "F"
                    FIN SI
                    caritaPromedio.enabled=true
                    caritaPromedio.sprite = icaritas [(int)(promedio * 10)] //Se tiene un
arreglo de imágenes de caras tristes y felices según la calificación que se obtuvo
                    textoPromedio.text = rank
;
                FIN SI
            FIN HACER
FIN PROCEDIMIENTO

PROCEDIMIENTO Ganar() INICIAR
        estadisticasPartida.Activar()
FIN PROCEDIMIENTO

PROCEDIMIENTO Perder() INICIAR
        estadisticasPartida.Activar()
        promedio.texto=(aciertos/(aciertos+fallas))
        textoAciertos.texto=aciertos
        textoFallas.texto=fallas
        textoPartidasPerdidas.texto=partidasPerdidas
FIN PROCEDIMIENTO

PROCEDIMIENTO ActualizarAciertos(int aciertos) INICIAR
        aciertos=a
FIN PROCEDIMIENTO

PROCEDIMIENTO ActualizarFallas(int f) INICIAR
        fallas=f
FIN PROCEDIMIENTO


PROCEDIMIENTO ActualizarPartidasPerdidas(int pp) INICIAR
        partidasPerdidas=pp
FIN PROCEDIMIENTO
```

```
/*Los siguientes métodos son llamados por botones en la interfaz de usuarios*/
PROCEDIMIENTO Aceptar() INICIAR
        SI estadisticasPartida.activado ENTONCES
                estadisticasPartida.Desactivar()
        FIN SI
        SI estadisiticasJuego.activado ENTONCES
                estadisticasJuego.Desactivar()
        FIN SI
FIN PROCEDIMIENTO

PROCEDIMIENTO Reiniciar() INICIAR
        estadisticasPartida.Desactivar()
        Brodcast.EmpezarPartida()
FIN PROCEDIMIENTO
```

5.6.- Implementación del subsistema “Lógica del serious game”

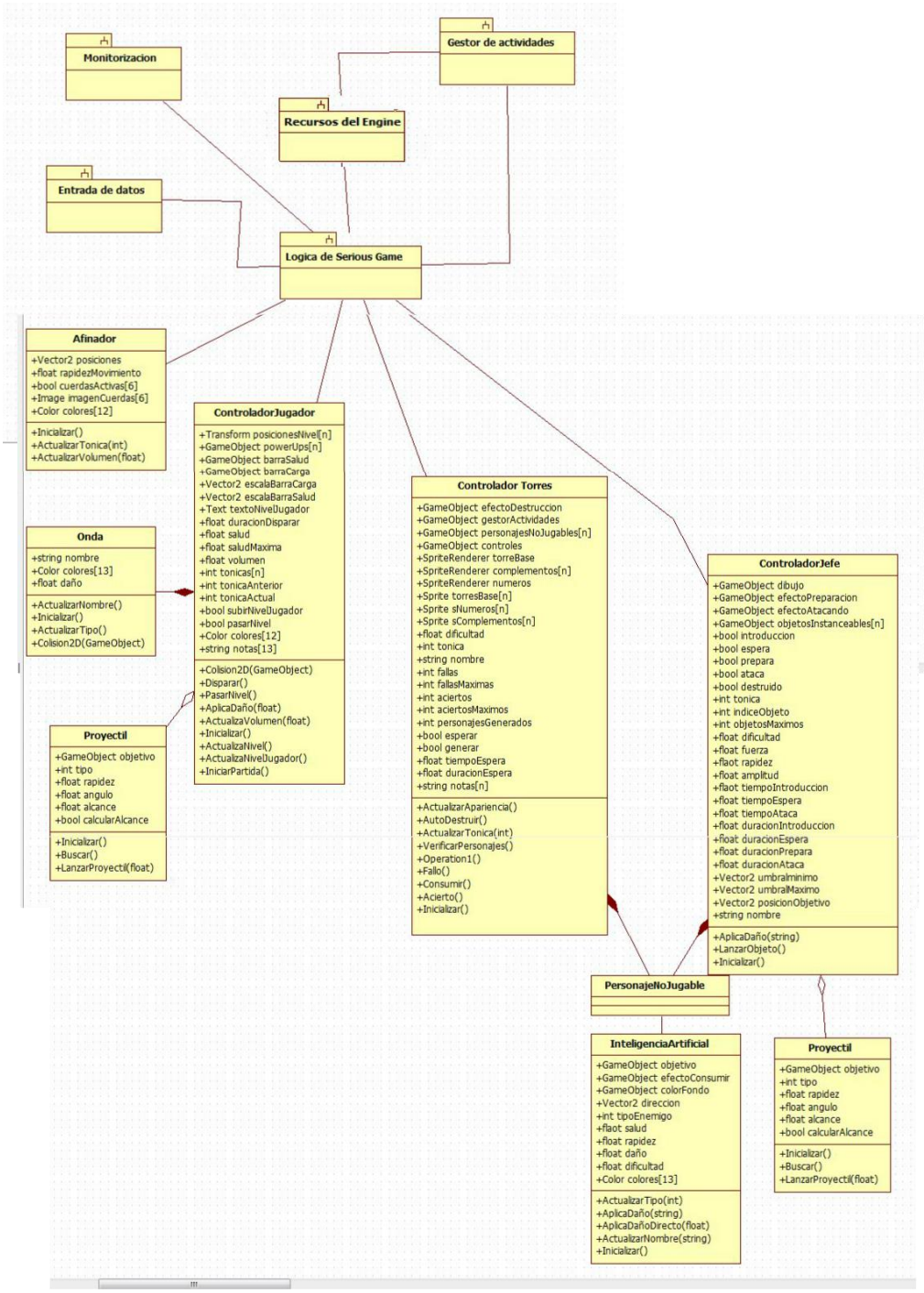


Figura 5.22 Diagrama subsistema “Lógica del serious game” con clases

Clase	Descripción
Afinador	Tiene todas las funciones de un afinador digital para guitarra
Onda	Tiene como función recibir mensajes del ControladorJugador para cambiar sus propiedades, también puede enviar mensajes a los objetos del tipo NPC
ControladorJugador	Cumple la función de minimizar el ruido e interpretar las acciones del usuario apropiadamente
ControladorTorres	Su función principal es generar instancias de unidades enemigas con propiedades diferentes según el estado de los autómatas definidos en el susbsistema "Gestor de actividades", también debe mantener un tiempo apropiado para la instanciación de dichas unidades respondiendo a una dificultad adecuada
ControladorJefe	Tiene la misma función del ControladorTorres con el detalle que la generación de unidades es aleatoria según una escala o progresión musical
InteligenciaArtificial	Su función es hacer que los objetos NPC aparenten tener inteligencia propia
Proyectil	Tiene definidos métodos que utilizan los recursos de físicas del mismo motor para otorgarles a los objetos tipo proyectil trayectorias parabólicas, dirigidas, horizontales o sinusoidales.

Tabla 5.13 Descripción de las clases del subsistema "Lógica del serious game"

El subsistema "Lógica del juego" responde a la mayor parte de los requerimientos no funcionales del sistema, constituye todos los objetos que van interactuar durante una partida, así mismo está conectado con otros subsistemas importantes como lo son: el gestor de actividades, monitorización y entrada de datos.

5.6.1.- Implementación de un afinador para guitarra básico

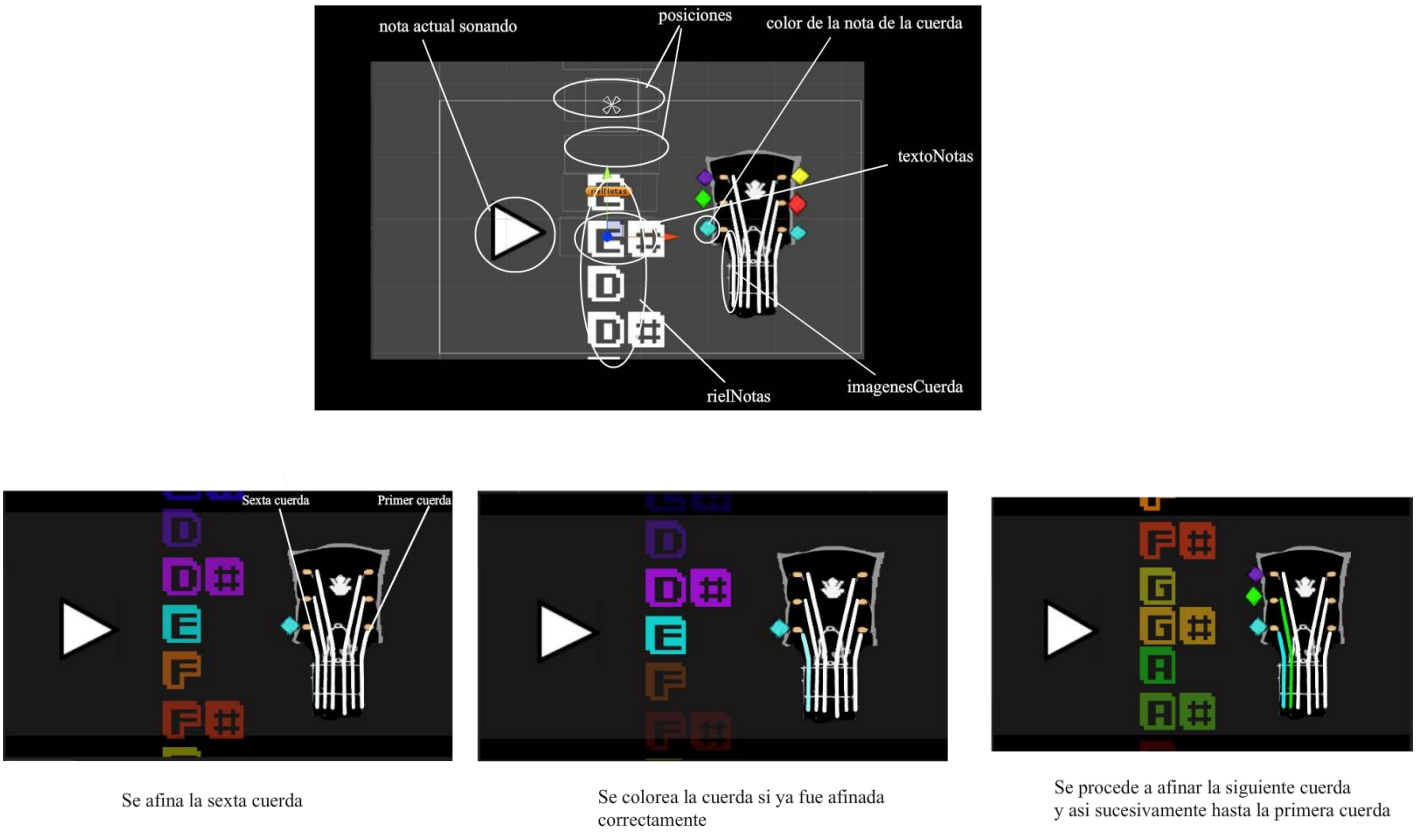


Figura 5.23 Variables gráficas y funcionamiento del algoritmo

Una de las cuestiones más importantes para que el usuario use de forma apropiada todo el sistema se requiere que su guitarra este bien afinada, por eso se implementó un afinador básico, para su programación se utilizaron algunos métodos de las clases "ProcesadorAudio" y "RedNeuronal". A continuación se muestra el algoritmo en forma de seudocódigo adicionalmente se presenta la figura 5.23 que muestra los objetos que contienen algunas de las variables del código.

```
GameObhect rielNotas
Vector 2[] posiciones
float rapidezmovimiento
bool cuerasActivas[6]
Image imagenCuerdas[6]
Text textoNotas[12]
Color colores[12]
float volumen
float tónica
float pcp[12]
Vector2 posicionTemporal
```

```
float tiempoRetencion
float umbralAceptacion=30
int cuerdaActual=6
float tonicasCuerda[6]
Color coloresObjetivo[6]
bool finalizar
// Se asignan los colores definidos para cada nota en código rgba
ROCEDIMIENTO Inicializar() INICIAR
        colores [0] = nuevo Color (0.01, 0.01 0.92,1)
        colores [1] = nuevo Color (0.25, 0, 0.99,1)
        colores [2] = nuevo Color (0.49, 0.03, 0.93,1)
        colores [3] = nuevo Color (0.72, 0.03, 0.95,1)
        colores [4] = nuevo Color (0.04, 0.97, 0.97,1)
        colores [5] = nuevo Color (0.96, 0.51, 0.03,1)
        colores [6] = nuevo Color (0.94, 0.26, 0.03,1)
        colores [7] = nuevo Color (0.98, 0.98, 0.03, 1)
        colores [8] = nuevo Color (0.97, 0.74, 0.05, 1)
        colores [9] = nuevo Color (0.03, 0.97, 0.03,1)
        colores [10] = nuevo Color (0.5, 0.96, 0.05,1)
        colores [11] = nuevo Color (0.95, 0.01, 0.01,1)
        coloresObjetivo[0]=nuevo Color (0.04, 0.97, 0.97,1)
        coloresObjetivo[1]=nuevo Color (0.03, 0.97, 0.03,1)
        coloresObjetivo[2]=nuevo Color (0.49, 0.03, 0.93,1)
        coloresObjetivo[3]=nuevo Color (0.98, 0.98, 0.03, 1)
        coloresObjetivo[4]=nuevo Color (0.95, 0.01, 0.01,1)
        coloresObjetivo[5]=nuevo Color (0.04, 0.97, 0.97,1)
        tonicasCuerda[0]=4
        tonicasCuerda[1]=11
        tonicasCuerda[2]=7
        tonicasCuerda[3]=2
        tonicasCuerda[4]=9
        tonicasCuerda[5]=4
        PARA i=0 HASTA i<12 HACER
                textoNotas.color=colores[i]
        FIN HACER
FIN PROCEDIMIENTO
// aquí se recibe el mensaje de la clase ProcesadorAudio
PROCEDIMIENTO ActualizaTonica(int x) INICIAR
                        tonica = x;
FIN PROCEDIMIENTO
PROCEDIMIENTO ActualizaVolumen(float x) INICIAR
                volumen = x;
FIN PROCEDIMIENTO
LOOP
        SI (no(finalizacion)) ENTONCES
                pcp.ProcesadorAudio.GetPcp();
```

```
            //Como la variable rielMetro es un objeto gráfico y el arreglo de posiciones
ya ha sido inicializado el rielMetro se va a mover según la tónica que este sonando
actualmente y el vector posiciones será el que tenga índice según la tónica
            SI (Vector2.Distancia(nuevo
        Vector2(rielMetro.posicion.x,posiciones[tonica].y),posicionTemporal)>0.1)
        ENTONCES
                    PosicionTemporal.x=metro.posicion.x;
                    PosicionTemporal.y=rapidezMovimiento*-
ObtenerSigno(metro.posicion.y-posiciones[tonica].posicion.y)
            FIN SI
            // En esta parte se comprueba que el audio externo  sea igual a la tónica de la
        cuerda que se va afinar
            SI
            (no(cuerdaActiva[cuerdaActual])&&tónica==tonicasCuerda[cuerdaActual])
        ENTNCES
                        tiempoRetencion+=0.1
                FIN SI
                SINO
                        tiempoRetencion-=0.1
                FIN SINO
            // En esta parte se va coloreando el dibujo de la cuerda actual que se está
afinando, se va a colorear mas según la variable  tiempoRetencion
            SI (no(cuerdas[cuerdaActual])) ENTONCES
                    imagenCuerdas[cuerdaActual].color=coloresObjetivo[cuerdaActual]-
Color.Blanco*((tiempoRetencion-umbralAceptacion)/umbralAceptacion);
            FIN SI
            SI (timpoRetencion>=umbralAceptacion) ENTONCES
                    cuerdasActivas[cuerdaActual]=true
                    tiempoRetencion=0
                    SI (cuerdaActual==5) ENTONCES
                        Finalizar=true
                    FIN SI
                    SINO
                        cuerdaActual++
                    FIN SINO
            FIN SI
        FIN SI
FIN LOOP
```

5.6.2.-Controlador del objeto “jugador”

Como se mencionó en el tema 4.3 correspondiente al Game Design que una de las mecánicas más importantes era generar una especie de onda de un color diferente dependiendo que nota o acorde que toque el usuario con una guitarra acústica, también se mencionó que esta onda cumplía la función de destruir las unidades enemigas del mismo color, desde el punto de vista de un lenguaje de programación las ondas son un objeto con una etiqueta (variable string con valor) que identifica exclusivamente al objeto, si el objeto colisiona con un objeto que representa una unidad enemiga entonces le manda un mensaje para comparar sus etiquetas si son las mismas entonces se aplica la función de Destroy() de lo contrario lo dejara intacto. Lo complicado no es generar objetos de tipo “onda” sino más bien generarlos de forma apropiada pues debido al ruido se generan demasiados objetos en escena, lo cual además de ser perjudicial para el desempeño del programa también es poco atractivo para un usuario, entonces la clase “ControladorJugador” sirve para manejar tanto parámetros del estado del usuario pero a su vez funciona como un amortiguador de ruido para el subsistema entrada de datos en específico para las clases que se encargan del reconocimiento de notas o acordes, y es un amortiguador de ruido pues reduce en un grado notable los inconvenientes del ruido.

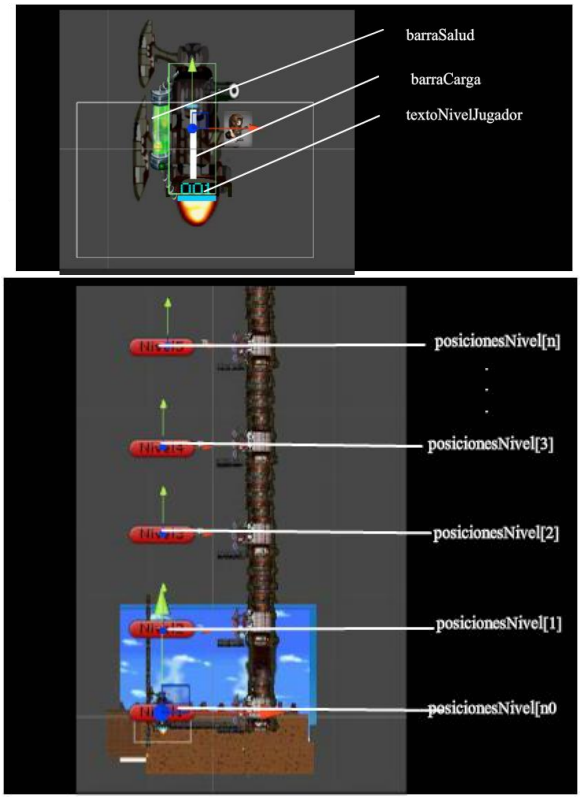


Figura 5.24 Variables del ControladorJugador

A continuación se muestra el seudocódigo del controlador jugador que se utilizó para programar la clase.

```
#NUMERO_NIVELES 100
Transform posicionesNivel[NUMERO_NIVELES]
GameObject powerUps[]
GameObject barraSalud
GameObject barraCarga
GameObject onda
Vector2 escalaBarraCarga
Vector2 escalaBarraSald
Text textoNivelJugador
float duracionDisparar
float tiempoDisparar
float salud
float saludMaxima
float volumen
float tiempoRetencionMaxInicial
float tiempoRetencionMaxima
int tonicas[12]
int tonicaAnterior
int tonicaActual
bool subirNivelJugador
bool esperarDisparar
bool perderPartida
bool pasarNivel
Color colores[12]
string notas[12]
int tonicaAnterior
float tónica
float tonicasRecibidas[]
float promedioTonicas[12]
// Se asignan los colores definidos para cada nota en código rgba y los vectores escalaCarga
y escalaBarra con las escalas de los objetos barraVida y barraCarga
PROCEDIMIENTO Inicializar() INICIAR
        tiempoRepeticionMaxInicial = tiempoRetencionMaxima;
        salud = saludMaxima;
        tonicasRecibidas=nuevo int[notasIgualesMaximas];
        colores [0] = nuevo Color (0.01, 0.01 0.92,1)
        colores [1] = nuevo Color (0.25, 0, 0.99,1)
        colores [2] = nuevo Color (0.49, 0.03, 0.93,1)
        colores [3] = nuevo Color (0.72, 0.03, 0.95,1)
        colores [4] = nuevo Color (0.04, 0.97, 0.97,1)
        colores [5] = nuevo Color (0.96, 0.51, 0.03,1)
        colores [6] = nuevo Color (0.94, 0.26, 0.03,1)
        colores [7] = nuevo Color (0.98, 0.98, 0.03, 1)
        colores [8] = nuevo Color (0.97, 0.74, 0.05, 1)
        colores [9] = nuevo Color (0.03, 0.97, 0.03,1)
        colores [10] = nuevo Color (0.5, 0.96, 0.05,1)
        colores [11] = nuevo Color (0.95, 0.01, 0.01,1)
```

```
        escalaCarga = barraCarga.escala
        escalaBarra = barraVida.escala

FIN PROCEDIMIENTO

PROCEDIMIENTO ActualizarTonica(int t) INICIAR
                tonica = t
FIN PROCEDIMIENTO
// Este método instancia un objeto tipo onda con una etiqueta
 PROCEDIMIENTO Disparar() INICIAR
        SI (!esperaDisparar&&no(pasarNivel)) ENTONCES
                        GameObject onda=nueva Onda()
                        onda.ActualizarNombre("Etiqueta"+notas[tonica])
                        onda.ActualizarTipo(tonica)
                        tiempoEsperaDisparar=Tiempo.tiempo+duracionDisparar;

        FIN SI
FIN PROCEDIMIENTO
// Si el usuario se equivocó recibe daño  a través de un mensaje proporcionado por la
instancia que toco el objeto onda
PROCEDIMIENTO AplicaDaño(float x) INICIAR
        Si (no(perderPartida)) ENTONCES
                salud-=x
                SI (salud<=0) ENTONCES
                        perderPartida=true
                        Administrador.PerderPartida()

                FIN SI
                barraVida.escala=Vector2(escalaBarra.x,
escalaBarra.y*(salud/saludMaxima))
        FIN S
FIN PROCEDIMIENTO
// Los métodos con prefijo Actualizar son mensajes que se reciben de diferentes objetos y
son parámetros importantes
PROCEIDMIENTO ActualizarNivel(int n) INICIAR
        nivel=n
        this.posicion= ivotesNivel [n].posicion

FIN PROCEDIMIENTO

PROCEvIMIENTO AcutliazarVolumen(float v) INICIAR
        volumen=v
FIN PROCEDIMIENTO
PROCEDIMIENTO ActualizarNivelJugador(int n) INICIAR
        SI  (n < 10)  ENTONCES
                textoNivel.text="00"+n
        FIN ENTONCES
```

```
        SI  (not> 10&&not<100) ENTONCES
                textoNivel.text="0"+n
        FIN SI
        SI (not>=100) ENTONCES
                textoNivel.text=n
        FIN SI
        float nivelJugadorDiezDiez = not/10;
        SI (nivelJugadorDiez >= 2 && nivelJugadorDiez <= 3) ENTONCES
                powerUps[0].Activar()
                powerUps[1].Desactivar()
                powerUps[2].Desactivar()
                saludMaxima=40;

        FIN SI
        SI (nivelJugadorDiez >= 4 && nivelJugadorDiez <= 5) ENTONCES
                powerUps[0].Activar()
                powerUps[1].Activar()
                powerUps[2].Desactivar()
                saludMaxima=60;
        FIN SI
        SI (nivelJugadorDiez >= 7 ) ENTONCES
                powerUps[0].Activar()
                powerUps[1].Activar()
                powerUps[2].Activar()
                saludMaxima=nivelJugadorDiez*10;

        FIN SI
        PARA i=0 HASTA i<powerUps.longitud  ENTONCES
                SI(powerUps[i].estaActivo)
                        powerUps[i]. ActualizaNivelPowerUp(nivelJugadorDiez)

        FIN PARA

FIN PROCEDIMIENTO

LOOP
        SI (no(perderPartida)) ENTONCES
                SI (pasarNivel) ENTONCES
                        SI (Vector2.Distance (this.posicion, posicionesNivel [nivel].posicion)
> 1f)
                                this.velocidad =Vector2 (0, 20)
                        SINO ENTONCES
                                posicionesNivel[nivel.Desactivar()
                                pasarNivel=true
                                AdministradoGameplay.EmpezarJuego()
                                this.velocidad =Vector2 (0, 0);
                        FIN SINO
```

```
        FIN SI
        SI (esperaDisparar) ENTONCES
                barraCarga.transPARAm.localScale = new Vector2 (escalaCarga.x *
-(((-tiempoEsperaDisparar + Time.time) / duracionDisparar)), escalaCarga.y)

        FIN
        SINO ENTONCES
                barraCarga.transPARAm.localScale = new Vector2 (escalaCarga.x
*tiempoRetencion/tiempoRetencionMaxima, escalaCarga.y)

        FIN SI
        // En esta parte se va contabilizando que nota o acorde se tocó con mayor
frecuencia  en un tiempo t razonable, es decir se está asegurando de que se está
tocando la guitarra  y no sea ruido
        SI(tonica==tonicaAnterior&&tonica!=13&&volumen>=2&&!esperaDispara
r) ENTONCES
                promedioTonicas[tonicaAnterior]++
                float prom=0
                prom=(float)(1/tiempoRetencionMaxima)
                int tonicaGrande=promedioTonicas[0]
                PARA  i=0 HASTA i<12 ENTONCES
                        SI(promedioTonicas[i]>=tonicaGrande)ENTONCES
                                tonicaGrande=promedioTonicas[i]
                                indiceTonica=i
                        FIN SI
                FIN SI
                iempoRetencion++
                SI(tiempoRetencion>=tiempoRetencionMaxima)ENTONCES
                        tiempoRetencion=0
                        tiempoRetencionMaxima=Mathf.Round(tiempoRepeticionMa
                        xInicial+intensidadVolumen*2.5f)
                        tonicaGrande=promedioTonicas[0]
                        indiceTonica=0
                        PARA  i=0 HASTA i<12 HACER
                                SI(promedioTonicas[i]>tonicaGrande)ENTONCES
                                        tonicaGrande=promedioTonicas[i]
                                        indiceTonica=i
                                FIN SI
                        FIN HACER

                        PARA  i=0 HASTA i<12 ENTONCES

                                promedioTonicas[i]=0
                        FIN SI
                        Disparar ()
```

```
                FIN SI
            FIN SI
            SI(tonica!=tonicaAnterior&&tiempoRetencion>0&&tonicaAnterior!=13)EN
        TONCES
                tiempoRetencion--
                promedioTonicas[tonicaAnterior]--
                float prom=(float)(1/tiempoRetencionMaxima)

            FIN SI
            // Se hace un pequeño retraso para que no se generen demasiados objetos
        onda en escena
            SI (esperaDisparar && Time.time > tiempoEsperaDisparar) ENTONCES
                esperaDisparar = false
            FIN SI
            SI (disparar) ENTONCES
                Disparar ()
            FIN SI
            tonicaAnterior=tonica
        FIN SI
FIN LOOP
```

Capítulo 6.- Problemas, propuestas para mejoras y evaluación

Este capítulo estará dedicado a comentar algunos inconvenientes (la mayoría ya se corrigieron) que se presentaron a lo largo del desarrollo del sistema, algunos fueron en gran medida por la falta de experiencia en algunos campos, otros se presentaron por limitantes del alcance del proyecto, los errores más notables fueron de diseño y de programación, al final de la descripción de cada error se dará una serie de propuestas para la mejora para trabajos similares a este.

6.1.- Problemas de diseño y propuestas de mejoras

A continuación se listan errores de índole visual y de game design

1.- Visuales:

Aunque este trabajo está más enfocado al aspecto de programación, también se usaron recursos gráficos y durante la fase de pruebas muchos de los usuarios se quejaron ya que el aspecto visual no era el más adecuado como se muestra en la figura 6.1, ya que la paleta de colores era de la escala de grises y los únicos colores eran para la interfaz de usuario, porque en un inicio se pesaba que así se iba a distinguir mejor los colores de las notas y por ende se entendería mejor el concepto, pero las pruebas mostraron que no fue así pues desde un punto de vista grafico estaba mal diseñado. Entonces se cambiaron varios gráficos que cubrían los requerimientos no funcionales, pero la funcionalidad se mantuvo y el prototipo final se muestra en la figura 6.2.

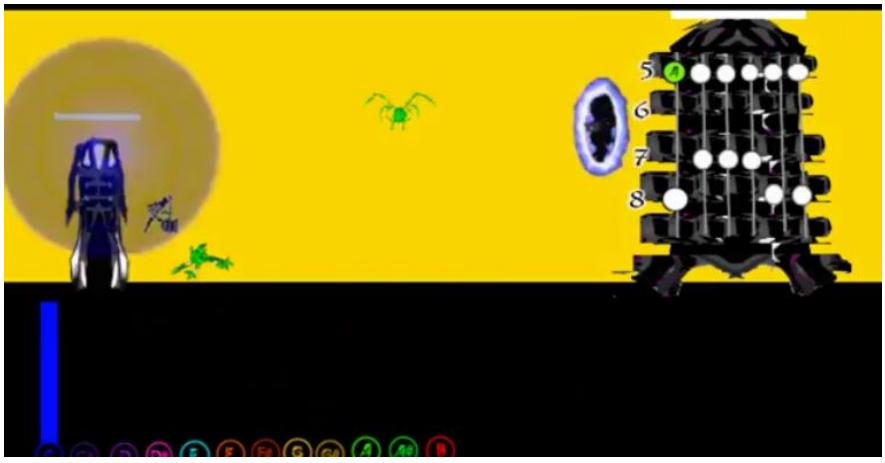


Figura 6.1 Prototipo inicial

Figura 6.2 Prototipo final

En las figuras 6.1 y 6.2 se puede apreciar la diferencia del diseño visual, la primera era muy pobre en colores v y en la segunda se ambiento de una forma más armónica con el fin que la mayoría de usuarios se sintiera a gusto con el concepto visual.

Durante la última fase de pruebas se detectó otro elemento visual que obstaculizó la buena utilización del programa fueron los colores que representan a las notas musicales, pues algunas de ellas tienen casi el mismo tono de color, pero esto es porque no existen 12 colores bien diferenciables, lo que se hizo para mejorar este detalle fue reacomodar la representación de cual nota o acorde está tocando el usuario, resulto que la forma circular fue mucho más cómoda que la del histograma (figura 6.3 y 6.4) .

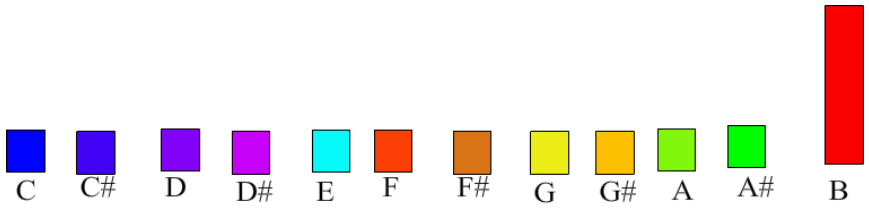


Figura 6.3 Representación de nota o acorde sonando en forma de histograma

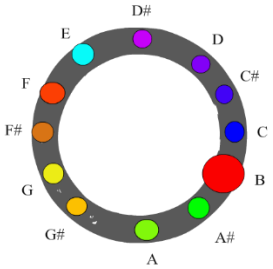


Figura 6.4 Representación de nota o acorde sonando en forma circular

Las propuestas para mejorar el diseño visual son las siguientes:

a) Consultar a un diseñador gráfico.
b) Pedir realimentación a los usuarios durante las fases de desarrollo y después de haber terminado (testeo).
c) Buscar referencias que se acomoden mejor a los requerimientos gráficos.
d) Antes de empezar con la elección de gráficos hacer el arte conceptual.

2.- Game Design

Como el proyecto iba de la mano con el diseño de videojuegos había ciertos temas que no se dominaban, uno de ellos era el diseño de mecánicas ya que depende mucho de la experiencia de la persona, el diseño suele ser subjetivo pues una de las características más importantes es que tiene que ser lo suficientemente divertido para el usuario y flexible para los requerimientos del sistema, al inicio del desarrollo se diseñaron diversas mecánicas, en el primer prototipo se había diseñado una pistola auto dirigida que generaban "proyectiles" de diferentes colores según el tono, el problema con esta mecánica es que como las instancias de las unidades enemigas se iban moviendo los "proyectiles" chocaban en otra parte como se puede observar en la figura 6.5.

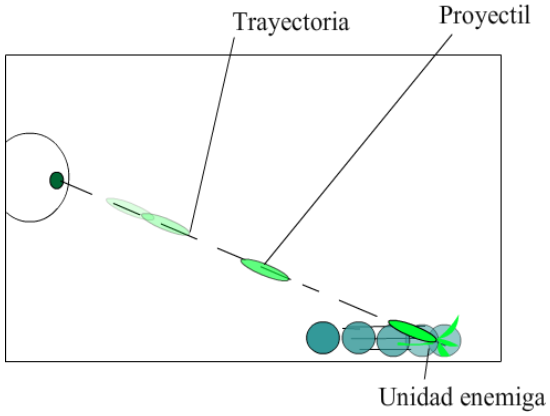


Figura 6.5 Primer idea de la mecánica principal

Después se hizo un segundo intento esta vez se cambió totalmente la mecánica, ahora en vez de ser "proyectiles" eran ondas como las que se forman cuando alguien tira un objeto en el agua, el tamaño de la onda era proporcional a la intensidad del sonido y el color dependía del tono, el problema fue que al ser de tamaño variable y como no todas las cuerdas producen la misma intensidad de sonido a veces no alcanzaba a colisionar con la unidad enemiga aunque se estuviera tocando la nota correcta lo que provocaba que el usuario se frustrara.

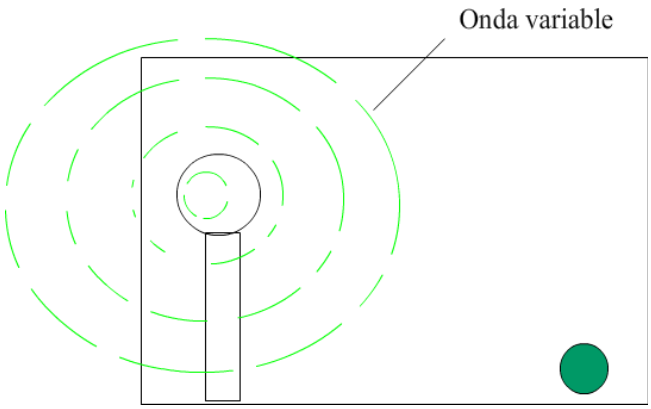


Figura 6.6 Segunda idea de la mecánica principal

La mecánica final fue la unión de las dos anteriores pues se concluyó que sería mejor tener ondas que a su vez fueran proyectiles que colisionaran con todo en la escena pero que solo pudieran destruir un objeto y es así como se balanceo la dificultad y se eliminaron los factores frustrantes.

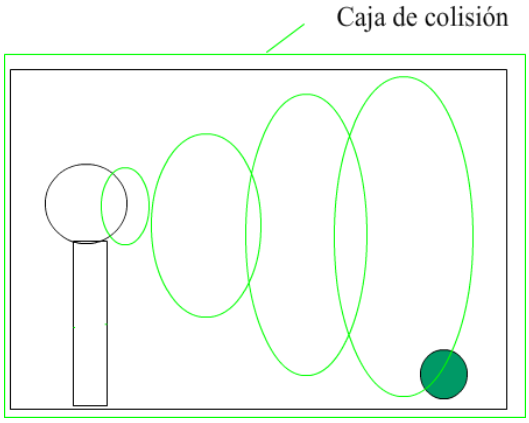


Figura 6.7 Tercer y última idea de la mecánica principal

Para mejorar el diseño de mecánicas se propone lo siguiente:

- Probar múltiples aplicaciones para darse una idea.
- Antes de programar hacer un bosquejo en papel y verificar que será entretenido, pensar que se está diseñando un nuevo juego de mesa.
- Intentar que las mecánicas no se vuelvan repetitivas y aburridas.
- No poner mecánicas o reglas que no sirvan para nada y solo sean un capricho del desarrollador o los desarrolladores.
- No ser abusivo con el usuario, no porque sea difícil es divertido.
- Ser realistas, si se es una sola persona involucrada en el desarrollo pensar que no se podrá lograr todo lo que se propone es por ello que se debe limitar lo que se desea.

6.2.- Problemas de implementación y propuestas de mejoras

En este tema se mostraran algunos errores y como se solucionaron y en algunos casos se proponen mejoras,

Red Neuronal

El primer error o más bien inconveniente fue debido a las limitantes de memoria y procesador en la computadora donde se probó el programa, se observó que el programa basado en el algoritmo mostrado en el tema 5.3.2 era demasiado lento al reconocer un tono de acorde o nota, se tardaba casi 0.6 segundos para dar una respuesta, así que se dispuso hacer un análisis grafico del funcionamiento del algoritmo como se muestra en los siguiente listado.

1.- Se le presenta un vector a la red neuronal el cual está representado por los 12 puntos de color rojo o verde si es verde quiere decir que el valor de la componente está más próxima a 1, si es rojo vale casi 0, adicionalmente se le presenta el vector de clasificación que es lo mismo que alguna de las neuronas de la salida.

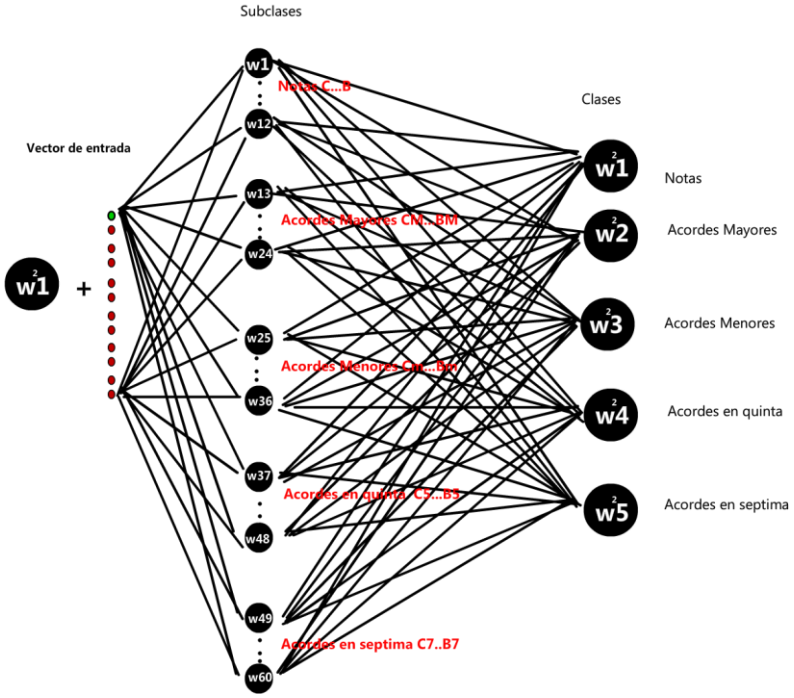


Figura 6.8 Red neuronal: fase de presentación

2.- Se calculan las distancias euclidianas, lo que se puede traducir a un flujo de información en la red.

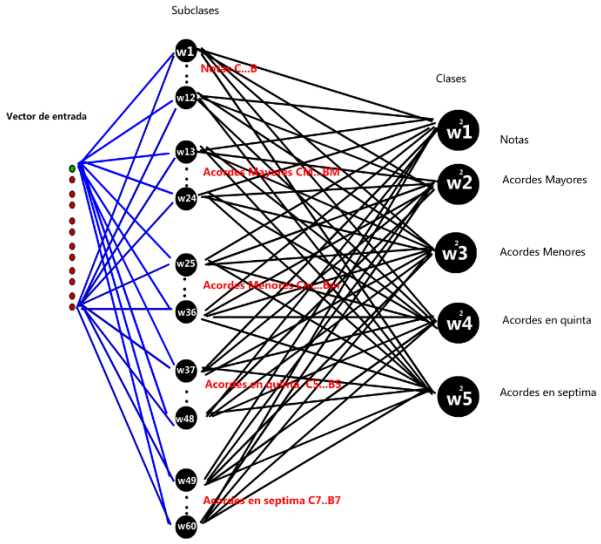


Figura 6.9 Red neuronal: fase del cálculo de distancias

3.- Después se enciende la neurona más cercana a la entrada:

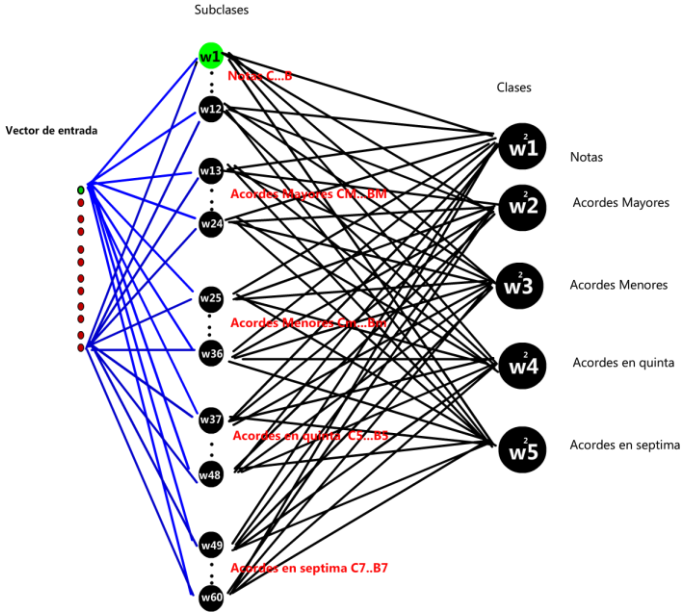


Figura 6.10 Red neuronal: fase del cálculo de la neurona ganadora

4.- Se clasifica la neurona.

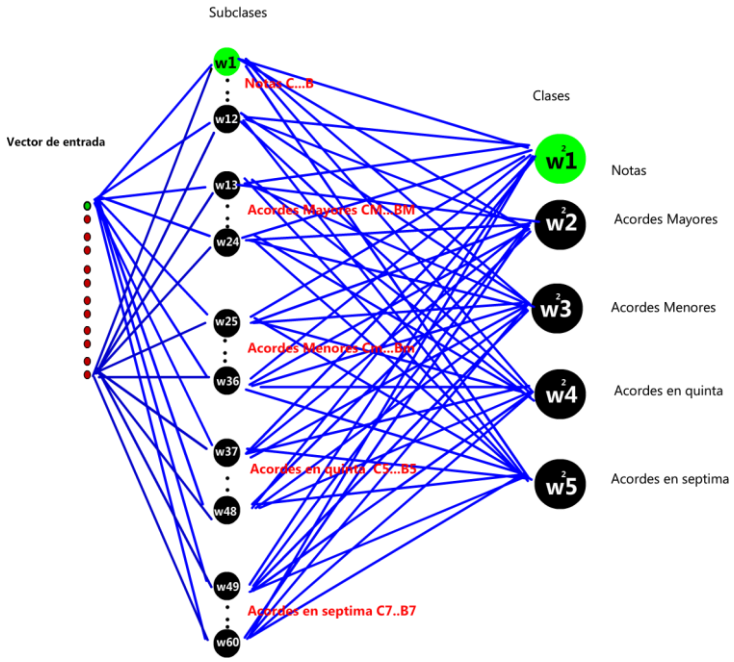


Figura 6.11 Red neuronal: fase de clasificación

Gráficamente es más claro que existen demasiadas conexiones y por cada conexión se debe realizar un cálculo (enlaces azules), desde el código el cálculo de cada conexión se interpreta a un cálculo dentro de uno o más ciclos **for**, para mejorar y corregir este inconveniente se decidió hacer otro algoritmo en él cual se omitió la multiplicación de la matriz W^2 y en vez de ello se sustituyo por un switch-case que nos dice si lo que se está reconociendo es un acorde mayor, menor, quinta, séptima o se trata de una nota individual, con ello se pudo eliminar la variable de **W2**, también se cambió el nombre a la matriz **W1** por **conexiones** y a la variable **a1** se le cambio el nombre por **distancias**, el seudocódigo es el siguiente:

```
#MUESTRAS 4096
string notaAEntrenar
int indiceNotaEntrenar
int  conexion[][]
int indiceMenor
float distancias[73]
float pcp[12]
string tipo
bool entrenar
int tónica
float promedio
```

```
float espectro[MUESTRAS]

PROCEDIMIENTO CalculaPcp() INCIAR
        espectro=DFT(Microfono.audio,MUESTRAS)
        PARA  i=0 HASTA 11 HACER
                PARA o=0 HASTA  5 HACER
                        pcp(i)+=espectro(M(o)(i))
                FIN HACER
        FIN HACER
        pcp.Normalizar()
FIN PROCEDIMIENTO

int PROCEDIMIENTO ObtenerIndiceNota(string nota) INICIAR
        string
notas={"C","C#","D","D#","E","F","F#","G","G#","A","A#","B","CM","C#M","DM","D#
M","EM","FM","F#M","GM","G#M","AM","A#M","BM","Cm","C#m","Dm","D#m","E
m","Fm","F#m","Gm","G#m","Am","A#m","Bm","C5","C#5","D5","D#5","E5","F5","F#5
","G5","G#5","A5","A#5","B5","C7","C#7","D7","D#7","E7","F7","F#7","G7","G#7","A7
","A#7","B7","Cm7","C#m7","Dm7","D#m7","Em7","Fm7","F#m7","Gm7","G#m7","Am
7","A#m7","Bm7","X"};
        SI (nota.logitud>0) ENTONCES
                PARA  i=0 HASTA i<notas.longitud HACER
                        SI (nota.Igual (notas [i]))
                                Regresa i;
                FIN HACER
        FIN SI
        Regresa -1;
FIN PROCEDIMIENTO
PROCEDIMIENTO CalcularDistancias() INICIAR
        PARA j=0 HASTA j<distancias.longitud HACER
                PARA i=0 HASTA i<pcp.longitud HACER
                        distancias[j]+=⟦(pcp[i]-conexion[j][i])⟧^2
                FIN HACER
                distancias[j]= √(distancias[j])
        FIN HACER
FIN PROCEDIMIENTO
Int PROCEDIMIENTO BuscaMenor(int a[])  INICIAR
        int menor=a[0]
        int indiceMenor=0
        PARA  i=0 HASTA i<notas.longitud HACER
                SI (menor>a[i]) ENTONCES
                        indiceMenor=i
                        menor=a[i]
                FIN SI
        FIN HACER
        SI (indiceMenor <= 11 && indiceMenor >= 0) ENTONCES
                tonica=indiceMenor
```

```
        FIN SI
        SI (indiceMenor <= 23 && indiceMenor >= 12) ENTONCES
                tonica=indiceMenor-12;

        FIN SI
        SI (indiceMenor <= 35 && indiceMenor >= 24)ENTONCES
                tipoAcorde = 24;
                tonica=indiceMenor-24;
        FIN SI
        SI (indiceMenor <= 47 && indiceMenor >= 36) ENTONCES
                tonica=indiceMenor-36;
        FIN SI
        SI (indiceMenor <= 59 && indiceMenor >= 48)ENTONCES
                tipoAcorde = 48;
                tonica=indiceMenor-48;
        FIN SI
        SI (indiceMenor <= 71 && indiceMenor >= 60)ENTONCES
                tonica=indiceMenor-60;
        FIN SI
FIN PROCEDIMIENTO


PROCEDIMIENTO Entrenar() INICIAR
        SI (no(notaAEntrenar.Igual ("""))) ENTONCES
                indiceNotaAEntrenar = ObtenerIndiceNota notaAEntrenar);
                distancias

                PARA  i=0 HASTA  i<12 HACER
                        SI (indiceMenor == indiceNotaAEntrenar) ENTONCES
                                        conexion[indiceNotaAEntrenar][i]+=0.4f*(conexión
                                                [indiceNotaAEntrenar]  [i]  + pcp [i])
                        SI (indiceMenor != indiceNotaAEntrenar) ENTONCES
                                        conexion[indiceNotaAEntrenar] [i] += 0.4f * (-
                                          conexion[indiceNotaAEntrenar] [i] + pcp [i])
                                          conexion[indiceNotaAEntrenar] [i] += 0.4f *
                                                (conexion[indiceMenor] [i] - pcp [i])

                        FIN SI
                        Sinapsis.Guardar(w)
                FIN SI
        FIN HACER
FIN PROCEDIMIENTO

LOOP
        pcp=obtenerPCP()
        CalcularDistancias()
        BuscaMenor (distancias)
```

```
        ProcesadorAudio.ActualizarTonica(tonica)
        SI (entrenar) ENTONCES
                Entrenar()
        FIN SI
FIN LOOP
```

Como se puede observar en el seudocódigo presentado, hay menos ciclos **for** que en el código del tema 5.3.2, sin embargo para el cálculo de las distancias aún quedaban dos ciclos **fo**r anidados los cuales hacen que el algoritmo sea de complejidad cuadrática y si se quisiera agregar una mayor cantidad de acordes el tiempo de respuesta seria más lenta, este problema se solucionó escribiendo el cálculo de las distancias sin ciclos como se puede observar en la siguiente captura de pantalla de la figura 6.9.

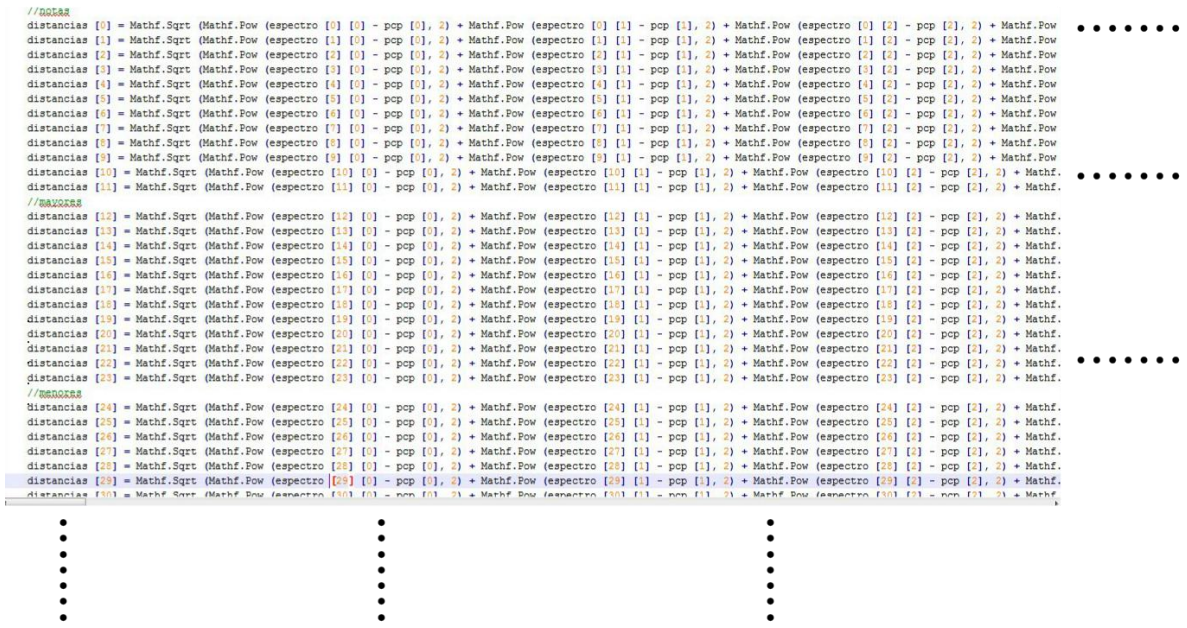


Figura 6.12 Calculo manual de las distancias

Algoritmo LVQ:

Otro problema que se encontró fue con respecto al algoritmo LVQ (Learning Vectorial Quantification) el cual tiene como función de reconocer 12 notas y 48 acordes cada uno conformado por diferentes neuronas con vectores prototipo de tamaño 12 esto provocaba que la red neuronal al no estar entrenada lo suficiente se confundiera con los acorde que está conformados, esto se puede interpretar a que los vectores de pesos estuvieran muy cercas unos de otros por ejemplo:

$$v_{Em} = \{0,0,0,0,0.1,0,0,0.1,0,0,0,0.1)$$
$$v_{Am} = (0.1,0,0,0,0.1,0,0,0,0,0.1,0,0)$$

Estos son los vectores de pesos antes de entrenar la red neuronal, si calculamos su distancia:

$$distancia = \sqrt{(0.1-0)^2+(0-0)^2+(0-0)^2+(0-0)^2+(0.1-0.1)^2+(0-0)^2+(0-0)^2+(0.1-0)^2+(0-0)^2+(0-0.1)^2+(0-0)^2+(0-0)^2} = \sqrt{0.2} = 0.44$$

Antes de la fase de entrenamiento se obtiene una distancia muy cercana entre las dos neuronas, pero después de entrenar la red neuronal se obtuvieron los siguientes vectores:

$$v_{Em} = (0.1,0.03,0.04,0.1608,0.825,0.253,0.0506,0.0904,0.108,0.169,0.141,0.234)$$
$$v_{Am} = (0.334,0.04,0.029,0.0413,0.50,0.0850,0.0648,0.0370,0.422,0.677,0.1540,0.129)$$

Si se calcula la distancia se obtiene como resultado una distancia de 0.703 lo cual teóricamente es buena pero según las pruebas en este trabajo para que se tenga un mejor desempeño en el reconocimiento de tonos la distancia deseable debería ser mayor de 0.85 ya que en la práctica no es muy óptima una distancia menor, pues retomando el ejemplo anterior cuando alguien toca el acorde por ejemplo el acorde **Am** la red se puede confundir por **Em** y viceversa, es por ello que se propone una solución para esta situación.

Como el sistema asigna actividades al usuario ya se sabe que nota o acorde se espera reconocer entonces lo que se hizo fue grabar todos los acordes que se van a reconocer y ponerlos en un arreglo de audios y antes de entrar a la fase de reconocimiento previamente se le debe presentar el audio del acorde que se espera reconocer, por ejemplo en este trabajo se grabaron los acordes que se van a reconocer por la red neuronal y se acomodaron en un arreglo de audios como se muestra en la 6.10 también se declaró una variable llamada **tonicaEsperada** la cual se recibe a través de un mensaje enviado por la clase "AsignacionActividades" y cuando se recibe esta variable se cambia el origen del audio del micrófono al elemento del arreglo de audios que tiene por índice la variable **tonicaEsperada** y se ejecuta la fase de entrenamiento por menos de 1 segundo y después se cambia el audio de entrada al micrófono.

Figura 6.13 Arreglo de audios

Otra mejora que se hizo para el algoritmo LVQ fue la reducción de algunos acordes ya que aunque existen 12 de cada tipo la realidad es que no se usan todos (como G# mayor, G7, F menor, etc.), para comprobar esta afirmación se hizo un estudio del número de concurrencias de acordes en el programa "Rapid Miner 5" (usado para minería de datos) donde la entrada fueron 1,289 archivos de texto (.pdf, .docx y .txt) que tienen acordes y letras de una o más melodías en inglés y español, el resultado se puede observar en la tabla 6.1.

Acorde	Concurrencias
G	14162
C	11875
D	11285
A	10205
E	9302
F	6474
Am	5281
Em	3667
B	2943
Bm	2543
Dm	2260
F#m	1578
A7	1193
G7	1122
D7	1053
F#	1008
B7	928
C#m	910
Gm	895
E7	874
C#	557
Am7	554
A5	538
G5	508
G#	468
D5	467
Cm	453
C7	437
Em7	388
B5	351
E5	350
Fm	341
C5	318
Dm7	305

G#m	299
F5	274
Bm7	227
D#	220
F#7	186
A#	177
A#	177
F7	155
Gm7	115
Cm7	105
D#m	97
F#5	82
C#7	81
G#5	71
C#5	58
Fm7	45
G#7	39
A#5	39
A#m	30
D#5	14
D#7	12
A#7	6
B#	4
B#m	1
B#7	1

Tabla 6.1 Numero de concurrencias por acorde

Nota: No se agregaron acordes extraños como: Maj, cuarta, novena, sexta, octava ni sus variaciones debido a su complejidad, pero si se puede ver su relación con respecto a los demás acordes en la gráfica de la figura 6.14.

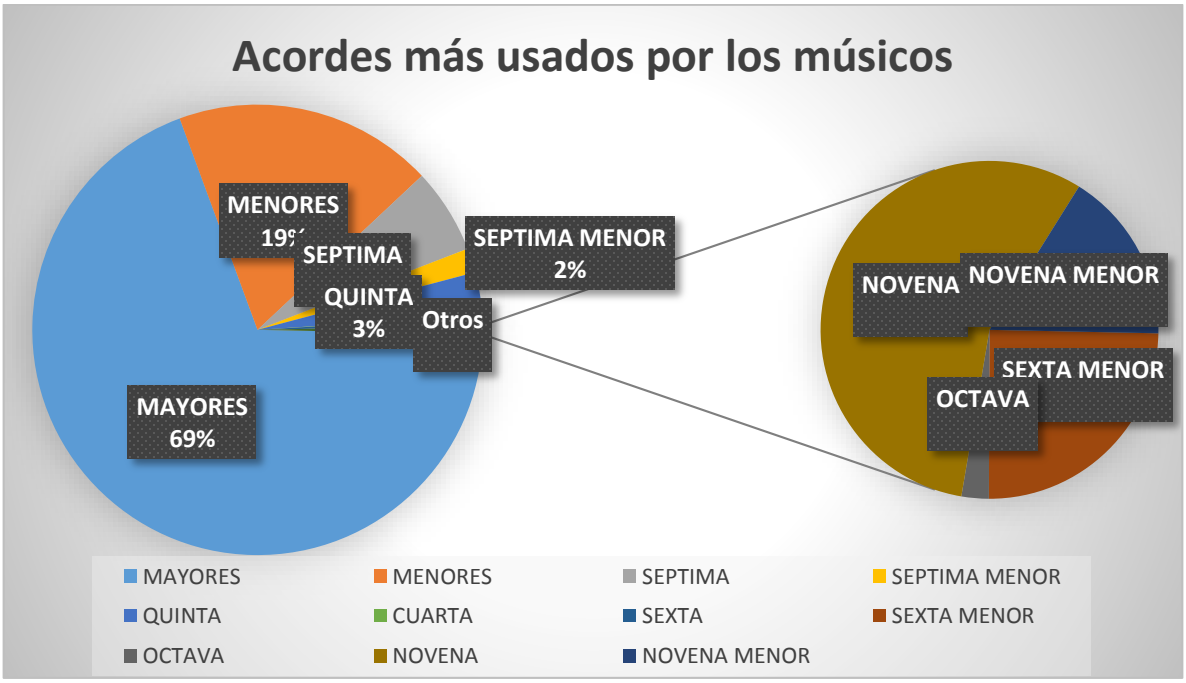


Figura 6.14 Grafica de los acordes mas usuales

De acuerdo a la figura 6.1 se concluye que los acordes mayores que prefieren los músicos son los acordes mayores y menores, esto era de esperarse ya que los acordes mayores son mayormente empleados para melodías alegres (amor, infantiles, etc.) y los menores se usan en su mayoría para melodías de despecho y los restantes se usan más para géneros no tan comunes entre las personas (jazz, blues, hardcore, metal, etc.), otra conclusión interesante es que el acorde más usual es "G mayor" (aunque uno pensaría que es A mayor) y casi ningún músico usa acordes mayores sostenidos a excepción de C# y F# (aunque si existen para las demás notas), entonces tomando de referencia este estudio se procedió a entrenar solo las neuronas que corresponden a los acordes más usuales.

Subsistema gestor de actividades

Cuando se transcribía cada actividad descrita en la tabla 4.7 al formato de autómata se llegó a la conclusión que no era lo suficientemente flexible para actualizaciones futuras, el problema no es que este mal representado ni mal diseñado, más bien es que aunque es posible actualizar las actividades se tiene que ser muy cuidadoso ya que al tener cinco autómatas para la representación de las actividades se tiene que fijar bien en que números pone porque si se equivoca en un valor se puede tardar más de lo previsto y lo ideal sería que cualquier persona pudiera insertar actividades sin afectar las anteriores y que no sea demasiado tardado, por eso se propone una solución a esta situación que aunque no se implementó ya que resultaría muy laborioso y se saldría del alcance del proyecto, se espera que resulte de utilidad para un proyecto de investigación que tenga como objetivo representar partituras de música a variables de un lenguaje de programación.

La propuesta para mejorar este punto consiste en programar un compilador que haga el análisis léxico, sintáctico y genere el código intermedio (arreglos de números) que se

ejecute antes de iniciar una partida y que sea capaz de transcribir la nomenclatura de las tablaturas y secuencias de acordes a arreglos de números enteros por ejemplo:

Entrada:

id=001

```
1|----------------------------------------5-8-----------|
2|-----------------------------------5-8----------------|
3|--------------------------5-7-------------------------|
4|------------------5-7--------------------------------|
5|-----------5-7---------------------------------------|
6|----5-8----------------------------------------------|
```

id=002

G,C#m,D#5.

Salida:

```
int[][] AutomataActividades=new int[][] {
                //Pentatonica Posicion1
                //1
                new int[]
{0605,0608,0505,0507,0405,0407,0305,0307,0205,0208,0105,0108},
                // Circulo armonico
                //2
                new int[] {0007,0101,0203},

}
```

En el ejemplo anterior se estableció una forma de representar el orden de las actividades mediante etiquetas, la etiqueta id=numero representa el número de actividad, también se establecieron dos formas de representación para las actividades por un lado dos tipos de representación una para las notas y otra para los acordes, en el caso de las notas los dos primeros dígitos representan la cuerda actual y los dos últimos representan el traste, en el caso de los acordes los dos primeros dígitos representa el tipo de acorde (para los mayores el 0, para menores el 1 y para los acordes en quinta el 2) y los últimos dos dígitos representan la posición de la tónica del acorde según la escala de "C". Esta forma de representación de las actividades daría la ventaja de otorgar la posibilidad de actualizar o modificar actividades externamente en un archivo de texto.

Bugs en algunos controladores

Durante la fase de desarrollo y prueba se presentaron varios errores de lógica en la programación de algunas clases que controlaban objetos de la escena, estos problemas se debieron a que previamente no se planteó bien todas las reglas que se debían seguir durante cada partida, aunque se plantearon algunos diagramas UML fue muy difícil de observar todas

las posibilidades que pudieran surgir durante la ejecución del programa, por esta razón se recomienda hacer lo siguiente:

a) Hacer un dibujo en papel con todos los elementos que va a poder hacer algún objeto en la escena.
b) Usar gráficos simplistas para probar el funcionamiento del objeto.
c) Aislar los objetos de los demás cuando se van a probar.
d) Programar el objeto de tal forma que no dependa de ningún otro en la escena.
e) Describir en una hoja de papel las reglas en general que va a tener cada reto en los diferentes niveles y hacer un dibujo que explique la rutina que va hacer.
f) Imprimir el valor de la variable que se crea que es la causante del problema.

Dependiendo de la naturaleza de los requerimientos funcionales, se recomienda probar programas parecidos al que se desea desarrollar, observar el comportamiento de cada objeto en el sistema desde un punto de vista técnico respondiéndose a uno mismo ¿Cómo puedo programar tal rutina?, si no se está seguro en un 80% de cómo se puede programar entonces se recomienda anotar las características más importantes y sus métodos del objeto y cómo se comporta si se aplica una entrada u otra, al final hacer un autómata que explique los estados del objeto.

6.3.- Evaluación

Para comprobar la exactitud de la red neuronal del sistema se diseñó la siguiente interfaz gráfica y programó la rutina que se muestra a continuación.

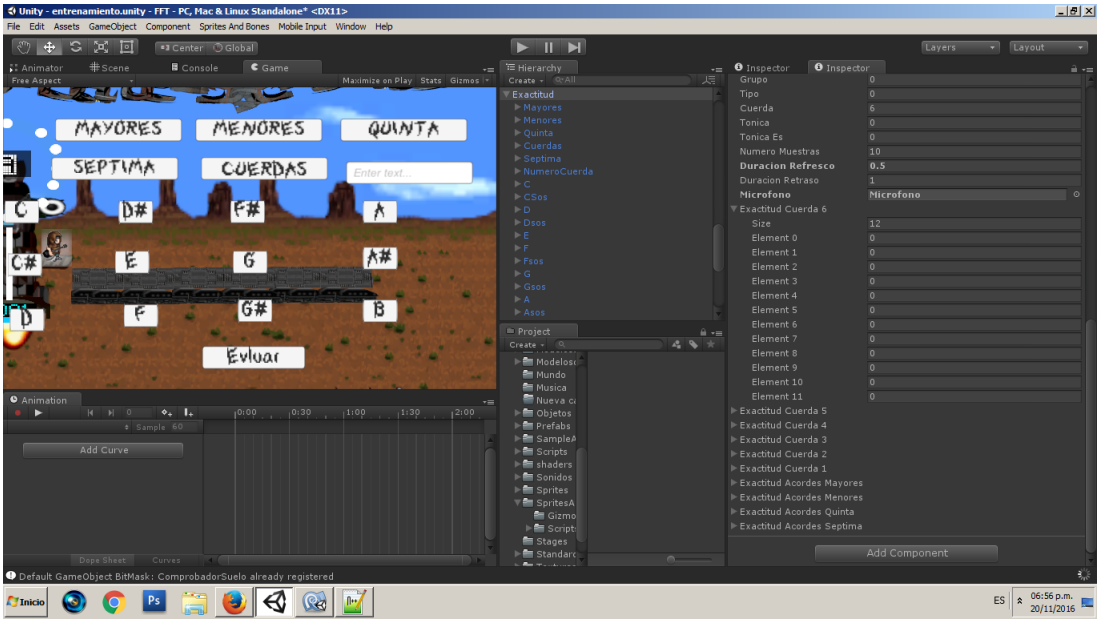


Figura 6.15 Interfaz para la evaluación

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class EvaluacionExactitud : MonoBehaviour {
		public Button[] botones;
		public Button[] botonesNotas;
		public Button botonEvaluar;
		public bool activado;
		public float muestrasMaximas;
		public Text numeroCuerda;
		public int grupo;
		public int tipo;
		public int cuerda;
		public int tonica;
		public int tonicaEs;
		public float numeroMuestras=10;
		float tiempoRefresco;
		public float duracionRefresco;
		float tiempoRetraso;
		public float duracionRetraso=1;
		public GameObject microfono;
		public float[] exactitudCuerda6;
		float[] muestrasCuerda6;
		public float[] exactitudCuerda5;
		float[] muestrasCuerda5;
		public float[] exactitudCuerda4;
		float[] muestrasCuerda4;
		public float[] exactitudCuerda3;
		float[] muestrasCuerda3;
		public float[] exactitudCuerda2;
		float[] muestrasCuerda2;
		public float[] exactitudCuerda1;
```

```
float[] muestrasCuerda1;
public float[] exactitudAcordesMayores;
 float[] muestrasAcordesMayores;
public float[] exactitudAcordesMenores;
float[] muestrasAcordesMenores;
public float[] exactitudAcordesQuinta;
float[] muestrasAcordesQuinta;
public float[] exactitudAcordesSeptima;
 float[] muestrasAcordesSeptima;
public void ActualizarTonicaEsperada(int x){
	tonicaEs = x;
}
public void Repetir(){
	if (grupo == 0) {
		if (cuerda == 6) {
			muestrasCuerda6 [tonicaEs] = 0;
			exactitudCuerda6[tonicaEs]=0;
		}
		if (cuerda == 5) {
			muestrasCuerda5 [tonicaEs] = 0;
			exactitudCuerda5[tonicaEs]=0;

		}
		if (cuerda == 4) {
			muestrasCuerda4 [tonicaEs] = 0;
			exactitudCuerda4[tonicaEs]=0;

		}
		if (cuerda == 3) {
			exactitudCuerda3[tonicaEs]=0;

			muestrasCuerda3 [tonicaEs] = 0;
		}
		if (cuerda == 2) {
			exactitudCuerda2[tonicaEs]=0;

			muestrasCuerda2 [tonicaEs] = 0;
		}
		if (cuerda == 1) {
			exactitudCuerda1[tonicaEs]=0;

			muestrasCuerda1 [tonicaEs] = 0;
		}
	} else {
		if(tipo==0){
			exactitudAcordesMayores[tonicaEs]=0;

			muestrasAcordesMayores[tonicaEs]=0;
		}
		if(tipo==1){
			exactitudAcordesMenores[tonicaEs]=0;

			muestrasAcordesMenores[tonicaEs]=0;
		}
		if(tipo==2){
			exactitudAcordesQuinta[tonicaEs]=0;

			muestrasAcordesQuinta[tonicaEs]=0;
		}
		if(tipo==3){
			exactitudAcordesQuinta[tonicaEs]=0;

			muestrasAcordesSeptima[tonicaEs]=0;
		}
	}
}
public void PresionoBotonNota(int x){
	for (int i=0; i<botonesNotas.Length; i++) {
		botonesNotas[i].GetComponent<Image>().color=Color.white;
	}
```

```
		botonesNotas[x].GetComponent<Image>().color=Color.green;
}
public void Activar(){
		activado = !activado;
		if(activado)
		tiempoRetraso = Time.time + duracionRetraso;
		if (activado) {
			botonEvaluar.GetComponent<Image> ().color = Color.green;

		} else {
			botonEvaluar.GetComponent<Image>().color=Color.white;

		}
}
public void PresionoBoton(int x){
		for (int i=0; i<botones.Length; i++) {
			botones[i].GetComponent<Image>().color=Color.white;
		}
		botones[x].GetComponent<Image>().color=Color.green;
}
public void CambiarGrupo(int x){
		grupo = x;
		microfono.SendMessage ("ActualizaGrupo",x);
}
public void ActualizarCuerda(){
		if (numeroCuerda.text.Length > 0)
			cuerda = int.Parse (numeroCuerda.text);
		else
			cuerda = 6;

}

public void CambiarTipo(int x){
		tipo = x;
		microfono.SendMessage ("ActualizaTipo",x);
}
// Use this for initialization
void Start () {
		muestrasCuerda6 = new float[13];
		 muestrasCuerda6 = new float[13];
		 muestrasCuerda5 = new float[13];
		 muestrasCuerda4 = new float[13];
		 muestrasCuerda3 = new float[13];
		 muestrasCuerda2 = new float[13];
		 muestrasCuerda1 = new float[13];
		 muestrasAcordesMayores = new float[13];

		 muestrasAcordesMenores= new float[13];
		 muestrasAcordesQuinta= new float[13];
		 muestrasAcordesSeptima= new float[13];

		for (int i=0; i<botonesNotas.Length; i++) {
			botonesNotas[i].GetComponent<Image>().color=Color.white;
		}
		botonesNotas[0].GetComponent<Image>().color=Color.green;
		botones[4].GetComponent<Image>().color=Color.green;

		numeroCuerda.text=cuerda.ToString();

}
void ActualizarTonica(int t){
		tonica=t;

}
// Update is called once per frame
void Update () {
		if (activado&&tiempoRetraso<Time.time) {
			if (tiempoRefresco < Time.time) {

				if(grupo==1){
```

```
                                        if(tipo==0){
                                                muestrasAcordesMayores [tonicaEs]++;
                                                if (muestrasAcordesMayores [tonicaEs] <=numeroMuestras)
{
                                                        if (tonica == tonicaEs) {
                                                                exactitudAcordesMayores   [tonicaEs]
= exactitudAcordesMayores [tonicaEs] + 1 / numeroMuestras;
                                                        } else {
                                                                if              (exactitudAcordesMayores
[tonicaEs] > 0)
                                                                        exactitudAcordesMayores
[tonicaEs] = exactitudAcordesMayores [tonicaEs] - 1 / numeroMuestras;

                                                        }
                                                }else{
                                                        Activar();

                                                }

                                        }

                                        if(tipo==1){
                                                muestrasAcordesMenores [tonicaEs]++;
                                                if (muestrasAcordesMenores [tonicaEs] <=numeroMuestras) {
                                                        if (tonica == tonicaEs) {
                                                                exactitudAcordesMenores         [tonicaEs]        =
exactitudAcordesMenores [tonicaEs] + 1 / numeroMuestras;
                                                        } else {
                                                                if (exactitudAcordesMenores [tonicaEs] > 0)
                                                                        exactitudAcordesMenores   [tonicaEs]
= exactitudAcordesMenores [tonicaEs] - 1 / numeroMuestras;

                                                        }
                                                }else{
                                                        Activar();

                                                }

                                        }
                                        if(tipo==2){
                                                muestrasAcordesQuinta [tonicaEs]++;
                                                if (muestrasAcordesQuinta [tonicaEs] <=numeroMuestras) {
                                                        if (tonica == tonicaEs) {
                                                                exactitudAcordesQuinta          [tonicaEs]        =
exactitudAcordesQuinta [tonicaEs] + 1 / numeroMuestras;
                                                        } else {
                                                                if (exactitudAcordesQuinta [tonicaEs] > 0)
                                                                        exactitudAcordesQuinta  [tonicaEs]  =
exactitudAcordesQuinta [tonicaEs] - 1 / numeroMuestras;

                                                        }
                                                }else{
                                                        Activar();

                                                }

                                        }
                                        if(tipo==3){
                                                muestrasAcordesSeptima [tonicaEs]++;
                                                if (muestrasAcordesSeptima [tonicaEs] <=numeroMuestras) {
                                                        if (tonica == tonicaEs) {
                                                                exactitudAcordesSeptima         [tonicaEs]        =
exactitudAcordesSeptima [tonicaEs] + 1 / numeroMuestras;
                                                        } else {
                                                                if (exactitudAcordesSeptima [tonicaEs] > 0)
                                                                        exactitudAcordesSeptima [tonicaEs] =
exactitudAcordesSeptima [tonicaEs] - 1 / numeroMuestras;

                                                        }
                                                }else{
```

```
                                                Activar();

                                        }

                                }

                        }
                        if (grupo == 0) {
                                if (cuerda == 6) {
                                        if (muestrasCuerda6 [tonicaEs]<numeroMuestras) {
                                                if (tonica == tonicaEs) {
                                                        exactitudCuerda6 [tonicaEs] = exactitudCuerda6
[tonicaEs] + 1 / numeroMuestras;
                                                } else {
                                                        if (exactitudCuerda6 [tonicaEs] > 0)
                                                                exactitudCuerda6      [tonicaEs]      =
exactitudCuerda6 [tonicaEs] - 1 / numeroMuestras;

                                                }
                                                muestrasCuerda6 [tonicaEs]++;

                                        }else{
                                                Activar();

                                        }
                                }
                                if (cuerda == 5) {
                                        muestrasCuerda5 [tonicaEs]++;
                                        if (muestrasCuerda5 [tonicaEs] <=numeroMuestras) {
                                                if (tonica == tonicaEs) {
                                                        exactitudCuerda5 [tonicaEs] = exactitudCuerda5
[tonicaEs] + 1 / numeroMuestras;
                                                } else {
                                                        if (exactitudCuerda5 [tonicaEs] > 0)
                                                                exactitudCuerda5      [tonicaEs]      =
exactitudCuerda5 [tonicaEs] - 1 / numeroMuestras;

                                                }
                                        }else{
                                                Activar();

                                        }
                                }
                                if (cuerda == 4) {
                                        muestrasCuerda4 [tonicaEs]++;
                                        if (muestrasCuerda4 [tonicaEs] <=numeroMuestras) {
                                                if (tonica == tonicaEs) {
                                                        exactitudCuerda4 [tonicaEs] = exactitudCuerda4
[tonicaEs] + 1 / numeroMuestras;
                                                } else {
                                                        if (exactitudCuerda4 [tonicaEs] > 0)
                                                                exactitudCuerda4      [tonicaEs]      =
exactitudCuerda4 [tonicaEs] - 1 / numeroMuestras;

                                                }
                                        }else{
                                                Activar();

                                        }
                                }

                                if (cuerda == 3) {
                                        muestrasCuerda3 [tonicaEs]++;
                                        if (muestrasCuerda3 [tonicaEs] <=numeroMuestras) {
                                                if (tonica == tonicaEs) {
                                                        exactitudCuerda3 [tonicaEs] = exactitudCuerda3
[tonicaEs] + 1 / numeroMuestras;
                                                } else {
                                                        if (exactitudCuerda3 [tonicaEs] > 0)
```

```
                                                                                exactitudCuerda3      [tonicaEs]      =
exactitudCuerda3 [tonicaEs] - 1 / numeroMuestras;

                                                                                }
                                                                        }else{
                                                                                Activar();

                                                                        }
                                                                }

                                                                if (cuerda == 2) {
                                                                        muestrasCuerda2 [tonicaEs]++;
                                                                        if (muestrasCuerda2 [tonicaEs] <=numeroMuestras) {
                                                                                if (tonica == tonicaEs) {
                                                                                        exactitudCuerda2 [tonicaEs] = exactitudCuerda2
[tonicaEs] + 1 / numeroMuestras;
                                                                                } else {
                                                                                        if (exactitudCuerda2 [tonicaEs] > 0)
                                                                                                exactitudCuerda2      [tonicaEs]      =
exactitudCuerda2 [tonicaEs] - 1 / numeroMuestras;

                                                                                }
                                                                        }else{
                                                                                Activar();

                                                                        }
                                                                }

                                                                if (cuerda == 1) {
                                                                        muestrasCuerda1 [tonicaEs]++;
                                                                        if (muestrasCuerda1 [tonicaEs] <=numeroMuestras) {
                                                                                if (tonica == tonicaEs) {
                                                                                        exactitudCuerda1 [tonicaEs] = exactitudCuerda1
[tonicaEs] + 1 / numeroMuestras;
                                                                                } else {
                                                                                        if (exactitudCuerda1 [tonicaEs] > 0)
                                                                                                exactitudCuerda1      [tonicaEs]      =
exactitudCuerda1 [tonicaEs] - 1 / numeroMuestras;

                                                                                }
                                                                        }else{
                                                                                Activar();

                                                                        }
                                                                }



                                                        }

                                                        tiempoRefresco = Time.time + duracionRefresco;
                                                }
                                        }

                                }
}
```

En la rutina se declaraba la salida esperada que se guardaba en la variable “tonicaEs” y se comparaba con el parámetro leído que es la variable “tonica”, todas las notas y acordes se leyeron a una tasa de 2 segundos de retraso, es decir, cada dos segundos se leía la entrada del micrófono, se tomaron 10 muestras para cada nota y cada acorde, los resultados se pueden observar en las tablas que se muestran a continuación.

Notas Cuerda 1	Exactitud
C	0.4
C#	0.4
D	0.4
D#	0.4
E	1
F	1
F#	1
G	1
G#	0.6
A	0.6
A#	0.6
B	0.6

Notas Cuerda 2	Exactitud
C	0.8
C#	0.8
D	0.8
D#	0.8
E	0.7
F	0.7
F#	0.7
G	1
G#	1
A	1
A#	1
B	1

Notas Cuerda 3	Exactitud
C	0.8
C#	0.8
D	0.8
D#	0.8
E	0.7
F	0.7
F#	0.7
G	1
G#	1
A	1
A#	1
B	1

Notas Cuerda 4	Exactitud
C	0.7
C#	0.7
D	1
D#	1
E	1
F	1
F#	1
G	0.8
G#	0.8
A	0.8
A#	0.8
B	0.8

Notas Cuerda 5	Exactitud
C	0.8
C#	0.8
D	0.8
D#	0.8
E	0.8
F	0.8
F#	0.8
G	0.8
G#	0.8
A	1
A#	1
B	1

Notas Cuerda 6	Exactitud
C	0.8
C#	0.8
D	0.7
D#	0.7
E	1
F	1
F#	0.8
G	0.8
G#	0.9
A	0.8
A#	0.8
B	0.7

Acordes Mayores	Exactitud
C	0.8
D	0.8
E	0.8
F	0.5
G	0.8
A	0.8
B	0.6

Notas Menores	Exactitud
D	0.9
E	1
F	0.8
F#	0.8
A	1
B	0.3

Acordes en quinta	Exactitud
C	0.8
C#	1
D	0.8
D#	0.5
E	0.8
F	0.8
F#	0.8
G	0.8
G#	0.8
A	1
A#	1
B	0.9

Acordes en séptima	Exactitud
C	0.8
D	0.4
E	0.8
F	0.5
A	0.8
B	0.5

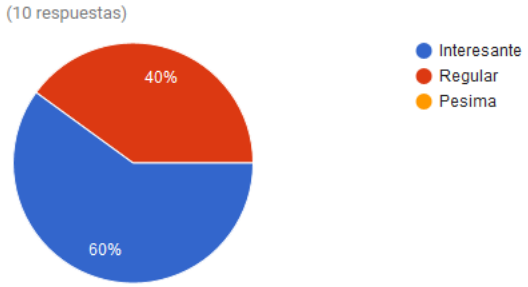
Tabla 6.2 Evaluación de la exactitud

Si se hace el promedio de los datos de todas las tablas se obtiene:

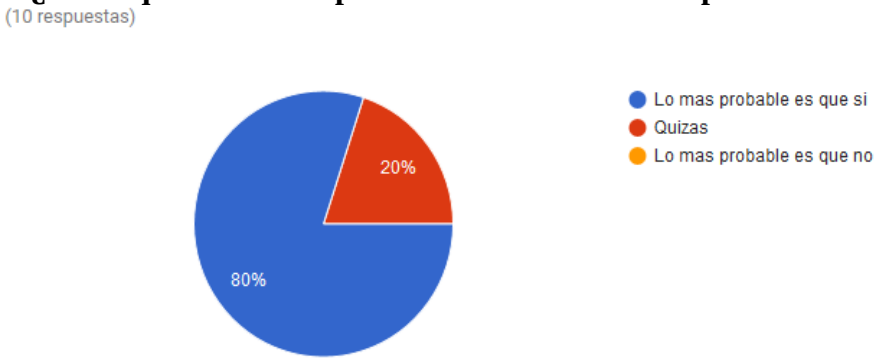
Exactitud Total=0.81165049=81%

Como se explicó en el tema 4.6.1 la edad recomendada para usar la aplicación es de 10 años en adelante ya que a una edad más temprana resulta difícil sostener y poner los dedos adecuadamente en el mástil de una guitarra[11], una vez implementadas las mejoras propuestas en el tema anterior se compilo un prototipo final que abarcaba una parte del contenido y se dio a probar la aplicación de 10 a 30 minutos a 10 personas, 3 personas tenían nociones de la guitarra acústica las 7 restantes les enseñe el manual y les explique lo mínimo que tenían que saber para usar la aplicación (todas las explicaciones para iniciar están en el manual de usuario para más información consultar el apéndice B), al finalizar se les dio una encuesta que constaba de 10 preguntas, los resultados de las respuestas de cada pregunta son los siguientes:

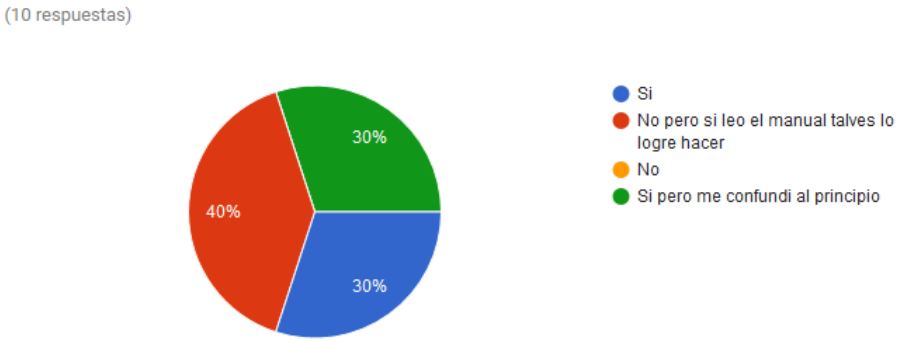
1.- ¿Qué te pareció la aplicación?



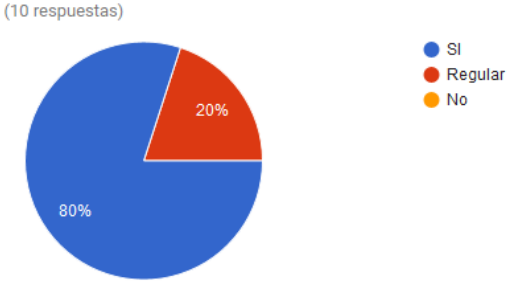
2.- ¿Crees que si usas la aplicación frecuentemente aprendas a tocar la guitarra?



3.- ¿Lograste interpretar los diagramas presentados que se proponía?

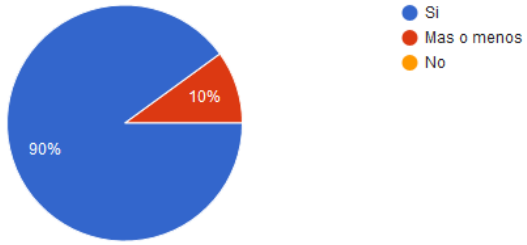


4.- ¿Te resulto fácil usar la aplicación?



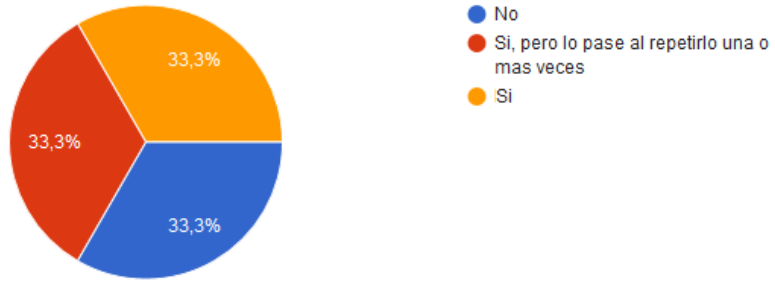
5.- ¿Te gusto el estilo visual?

(10 respuestas)



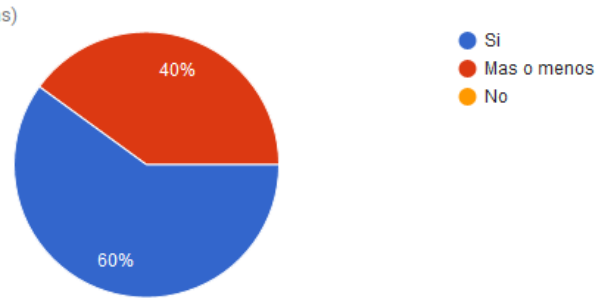
6.- ¿Tuviste problemas para pasar algún nivel?

(10 respuestas)



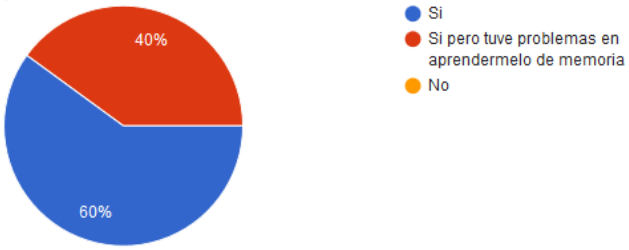
7.- ¿Te divertiste?

(10 respuestas)



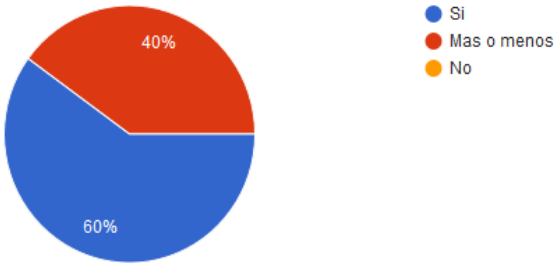
8.- ¿Aprendiste a tocar al menos un acorde en la guitarra?

(10 respuestas)



9.- ¿Aprendiste a reconocer algún sonido que emite una determinada nota?

(10 respuestas)



10.-Que te gustaría cambiar o agregar a la aplicación

(5 respuestas)

me gustaria que hubiera mas variedad de enemigos
nada
agregarle efectos de sonido
me gusto mucho pero le falto musica de fondo
que hubiera mas niveles

Los usuarios se pudieron dar cuenta los usuarios que si se usaba con frecuencia la aplicación podían aprender guitarra porque antes de probar la aplicación les indique a las personas que por experiencia propia (9 años) "aprender guitarra" no solo implica aprender a tocar varias canciones y memorizarlas para que en un año se olvide el 70% (que es justamente el error que cometí cuando empezaba a aprender), sino más bien implica aprender a leer notación americana preferentemente, aprender a ubicar las notas en el mástil, poner mal los dedos y equivocarse múltiples veces hasta ponerlos bien (recordar que el serious game está basado en el método de ensayo-error), identificar posiciones de las escalas para que cuando se olvide una canción se sepa improvisar adecuadamente, aprender las progresiones más importantes (círculos armónicos) e identificar los tonos con el oído cuando se quiera obtener una nota sin ver partituras (incluido en el modo "sin guitarra"), las personas que probaron la aplicación consideraron todo lo anterior antes de responder el cuestionario.

Para finalizar yo mismo autoevaluare mi software como herramienta pues como estuve involucrado en el trabajo de investigación y desarrollo durante un año estuve probado y depurando múltiples errores, mientras depuraba tuve que emplear una guitarra y verificar cada actividad, a lo largo de ese tiempo me di cuenta que sin querer me aprendí de memoria las ubicaciones de las notas y a identificar los tonos de las notas de la escala pentatónica menor en tono de A, lo cual me parece algo muy positivo pues como experimentado en este ámbito sé que es lo más difícil de aprender, el único punto en contra que le encuentro es que sé que la mayoría de personas no lee los manuales que se les proporciona y no valoran la aplicación como se debería, entonces el trabajo que me corresponde es mejorar tutoriales ya que las pruebas que hice con los usuarios fue en un ambiente controlado y aunque el alcance de este proyecto no cubre la parte de programación de tutoriales dinámicos (solamente se cuenta con tutoriales fijos y manuales de usuario) se agregarán cuando se disponga del tiempo necesario pues al no generar ningún ingreso económico con este proyecto no me es posible dedicar más de 6 horas diarias al desarrollo.

6.4.- Conclusiones

A lo largo del desarrollo del sistema y la redacción de esta tesis se llegó a las siguientes conclusiones:

-Durante el diseño y desarrollo de la aplicación se descubrió que el desarrollo de un sistema basado en características de serious game no es una tarea fácil pues se requieren conocimientos bastos no solo de programación sino también del área que se va a tratar (guitarra y música para este proyecto), pedagogía, diseño gráfico, edición de audio, lógica, matemáticas, física, optimización de juegos y game design, es decir, el desarrollo de un serious game es una tarea multidisciplinaria y es muy difícil que una sola persona pueda desarrollar todo sin tener problemas.

- Resulta más fácil y conveniente diseñar el sistema como una combinación de subsistemas aislados uno de otros, ya que durante la fase de pruebas se necesita ejecutar algún método de forma repetitiva para verificar su buen funcionamiento y a veces la ejecución de algún método requiere que exista un objeto o que se ejecute otro(s) método(s) los cuales por cuestiones de diseño a veces dicha existencia o ejecución tiene un tiempo de retardo, es por ello que se debe declarar variables o métodos de prueba para hacer los testeos de forma rápida e independiente a otros elementos que conforman los demás subsistemas.

-Al requerir de un subsistema capaz de reconocer notas y acordes en tiempo real se analizaron diversos métodos para la implementación del subsistema y durante su estudio se percató que podría ser un tema de investigación completo debido a su complejidad, por esta razón es que en los temas referente a su estudio se profundizo más y se dieron propuestas más atractivas para la mejora del reconocimiento de tonos en tiempo real.

-Se verificó que el método de los seis pasos de Schottman para el diseño de un serious game resulta ser efectivo pues funciona como una guía para tener una visión más clara de ¿Hasta qué punto se quiere llegar? y ¿Cómo se va a llegar?

Bibliografía y Referencias

[1]How Much Does It Cost To Make A Big Video Game? (2014), http://kotaku.com/how-much-does-it-cost-to-make-a-big-video-game-1501413649.

[2]Desarrollo de videojuegos en Mexico, Obtenido de http://the-ciu.net/nwsltr/526_1Distro.html.

[3]Zyda, Michael (2005), *From Visual Simulation to Virtual Reality to Games*, USC Information Sciences Institute, (p. 26-29).

[4]Baratè Adriano, G. Bergomi Mattia, A. Ludovico Luca (2013), *Development of Serious Games for Music Education*, Laboratorio di Informatica Musicale (LIM) -Dipartimento di Informatica, 9 (2), (p. 92-95).

[5]Natalia Padilla Zea (2011), *Mitología para el diseño de videojuegos educativos sobre una arquitectura para el análisis del aprendizaje colaborativo*, tesis doctoral, Universidad de granada, departamento de lenguajes y sistemas informáticos, (p. 103-112).

[6]Rob Nadolski*, Hans Hummel, Henk van den Brink, Ruud Hoefakker, Aad Slootmaker, Hub Kurvers, Jeroen Storm (2008), *EMERGO: methodology and toolkit for efficient development of serious games in higher education* , Open University of the Netherlands, (p 2-8).

[7]Chi Dung TRAN, Sébastien GEORGE, Iza MARFISI-SCHOTTMAN (2010) , *EDoS: An authoring environment for serious gamesdesign based on three models*, Université de LyonINSA-Lyon (p. 4-8).

[8]ECRM 2010 - 9th European Conference on Research Methods for Business and Management Studies - Madrid, Spain, Recuperado de http://www.academic-bookshop.com/ourshop/prod_1243619-ECRM-2010-9th-European-Conference-on-Research-Methods-for-Business-and-Management-Studies-Madrid-Spain-PRINT-version.html.

[9]Alfredo Prieto, David Diaz y Raul Santiago, *Metodologias indictivas "El desafio de enseñar mediante el cuestionamiento y los retos"*, (p. 3-12).

[10]Requisitos del sistema para la Unity versión, Recuperado de https://unity3d.com/es/unity/system-requirements

[11]At what age should my child start guitar?, Recuperado de: http://www.gregsguitarlessons.com/2010/07/at-what-age-should-my-child-start-guitar/

[12]Cook Andrew (2012), *Music Theory*. Recuperado de http://books.lardbucket.org/b%20ooks/music-theory.

[13]Danc (2006). *What are game mechanics?*, LostGarden blog. Recuperado de

http://www.lostgarden.com/2006/10/what-are-game-mechanics.html.

[14]Fujishima Takuya (1999), *Realtime Chord Recognition of Musical Sound: a System Using Common Lisp Music, OCRMA,* University Standford CA, USA.

[15]Hartquist Jonh (2012), *Real-Time musical Analysis of polyphonic guitar audio*, Master of Science in Computer Science, Faculty of California Polytechnic State University.

[16]Hagan, Martin T, Demuth, Howard B, Hudson Beale Mark y De jesus, Orlando (2014). *Neural Network Design 2nd Edition.* Recuperado de http://hagan.okstate.edu/.

[17]J. Osmalskyj, J-J. Embrechts, S. Piérard y M. Van Droogenbroeck (2012), *Neural, networks for musical chords recognition, INTELSIG Laboratory,* University of Liège, Departement EECS.

[18]Kato Romo, E. (2011). *Game design for a serious game to help learn programming*. Master in Multimedia. Faculdade de engenharia da universidade do Porto.

[19]Krenz, Steve (2010), *Gibson's Learn and Master*, China: Legacy Learning Systems.

[20]La tecla de escape (2015), *Frecuencia de las notas musicales*. Recuperado de http://latecladeescape.com/h/2015/08/frecuencia-de-las-notas-musicales

[21]Lee Kyogu (2006), *Automatic Chord Recognition from Audio Using Enhanced Pitch Class Profile*, Center for Computer Research in Music and Acoustics Department of Music, Stanford University.

[22]Unity 3d, *Manual de Unity*. Recuperado de https://docs.unity3d.com/es/current/Manual/index.html.

[23]Morales Moras Joan (2013), *El diseño de serious games: una experiencia pedagógica en el ámbito de los estudios de Grado en Diseño*, Facultad de Bellas Artes, Universidad de Barcelona, España.

[24]Rock & Roll para Muñones (2010), *Turbo-Manual de guitarra eléctrica v2.0*, Recuperado de www.rockandrollparamunones.com.

[25]Rouse III, Richard (2005), *Game Design theory & practice 2nd edition*, United States of America: Wordware Publishing, Inc.

[26]Sanchez Gomez Maria (2007), *Buenas Prácticas en la Creación de Serious Games (Objetos de Aprendizaje Reutilizables),* Universidad de Málaga. Facultad de Ciencias de la Comunicación. Campus de Teatinos, 318(32).

[27]Sheh Alexander y P.W Ellis Daniel (2003), *Chord Segmentation and Recognition using EM-Trained Hidden Markov Models*, LabROSA, Dept. of Electrical Engineering. Columbia University, New York NY 10027 USA.

[28]Schell Jesse (2008), *The Art of Game Design*, Estados Unidos: Morgan Kauffman Publishers.

[29]Tim Ryan (1999), *Beginning Level Design, Gamasutra the art & bussiness of making games*. Recuperado de http://www.gamasutra.com/view/feature/131736/beginning_level_design_part_1.php

[30]Wang Fei, Tan Chenhao, Konig Arnd Christian y Li Ping(2011), *Efficent Document Clustering via Online Nonnegative Matrix Factorizations*, Cornell University.

Apéndice A: Lista de tareas y diagrama de Gantt

Lista de tareas

Prioridad:

1.-Maxima
2.- Media
3.- Baja

A.- Especificación de objetivos pedagógicos

A1.- Analizar aplicaciones parecidas a la que se está desarrollando (2)
A2.- Hablar con experto en el área y obtener conclusiones al respecto (2)
A3.- Estudiar libros de enseñanza de guitarra y pedagogía (1)
A4.- Definir objetivos según el alcance del proyecto (1)

B.- Elegir el modelo del juego

B1.- Bosquejar que tipo de juego según los objetivos pedagógicos (1)
B2.- Programar un primer prototipo pequeño del modelo elegido (1)
B2.1.- Hacer gráficos simples (1)
B2.2.- Programar un generador de instancias de tipo NPC (2)
B2.3.- Programar generador de ondas (objetos con etiquetas diferentes según la tecla) (1)
B2.4.- Probar primer prototipo (2)
B2.5.- Evaluar cómo se desenvuelven los usuarios en el tipo de juego (2)
B3.- Programar un segundo prototipo pequeño (2)
B3.1.- Rediseñar el tipo de juego (1)
B3.2.- Programar algunas actividades diseñadas (1)
B3.3.- Probar segundo prototipo (1)
B3.4.- Evaluar cómo se desenvuelven los usuarios en el tipo de juego (1)

C.- Diseñar el escenario del serious game

C1.- Dibujar gráficos y crear sonidos específicos que no se encuentren de forma gratuita (3)
C2.- Buscar gráficos y sonidos (royalty free) que mejor acompañen al contexto del juego (3)
C3.- Bosquejar cada nivel detallando que elementos se van a poner en pantalla (2)
C4.- Diseñar y programar el subsistema “Entrada de datos” (2)
C4.1.- Investigar trabajos relacionados con el reconocimiento de tonos polifónicos (1)
C4.2.- Investigar y estudiar el diseño de redes neuronales (1)
C4.3.- Diseñar red neuronal (1)
C4.4.- Programar todo lo relacionado con el subsistema “Entrada de datos” (1)
C4.4.1.- Programar red neuronal (1)
C4.4.2.- Programar el guardado de los pesos sinápticos (2)

D.- Búsqueda de componentes de software

D1.- Abrir los proyectos realizados anteriormente y buscar scripts que se acoplen al proyecto (3)
D1.1.- Copiar partes programadas que se pueden reusar y borrar las partes que no se requieran (3)

D2.- Buscar gráficos que se pueden reutilizar y convertirlos en hojas de sprites o secuencias de imágenes para el caso de las animaciones (3)

E.- Controlar calidad pedagógica

E1.- Probar múltiples veces la aplicación durante el desarrollo (1)

E2.- Dar a probar a diferentes personas ajenas al desarrollo del proyecto (2)

E2.1.- Tomar nota de las dificultades que tuvo cada persona (2)

E3.- Rediseñar y volver a elegir componentes (2)

F.- Detallar pantallas y especificaciones HCI (Human Computer Interactions) (1)

F1.- Diseñar y programar todo lo referente al subsistema "Gestor de actividades" (1)

F1.1.- Documentación de material didáctico para aprender a tocar escalas y progresiones (1)

F1.2.- Diseñar las actividades referentes a las escalas y progresiones(1)

F1.3.- Diseñar la forma en que se va a programar las actividades y concluir como se va a llevar a cabo la interacción (1)

F1.3.1.- Programar la representación de las actividades que se eligió (1)

F1.3.2.- Elegir que elementos visuales se van a relacionar con las actividades (1)

F2.- Diseñar y programar lo referente a la "Lógica del juego" (2)

F2.1.- Bosquejar las reglas y los sucesos que se ejecutaran dentro de la partida (2)

F2.1.1.- Diseñar y programar el comportamiento de los NPC según el modo de juego (con guitarra o sin guitarra) (2)

F2.1.2.- Diseñar y programar el comportamiento del objeto que controla el jugador según el modo de juego (2)

F2.1.3.- Diseñar y programar mecánicas del juego (1)

F2.1.4.- Diseñar y programar el orden de aparición de las actividades y como se va a representar con objetos gráficos cada actividad (2)

F2.1.5.- Programar el controlador del objeto "jugador" (2)

F3.- Diseñar y programar lo referente al subsistema de "Presentación" (3)

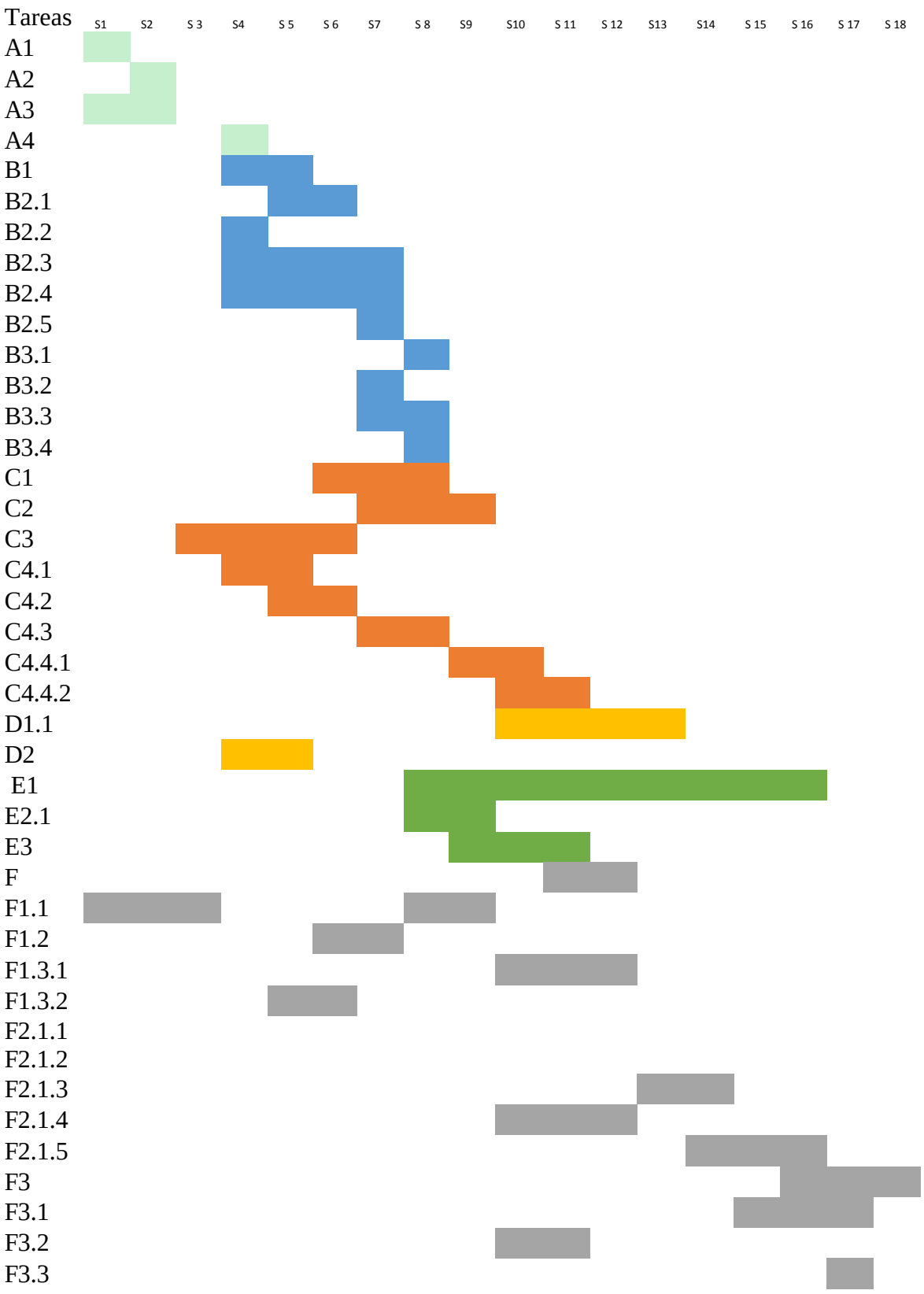
F3.1.- Analizar aplicaciones que tienen la función de guardado y se asemejan a la interfaz de usuario que se desea realizar (2)

F3.2.-Diseñar navegación de pantallas (2)

F3.3.- Programación de la interfaz de usuario (botones, información relevante al perfil, aparición de iconos, etc.) (3)

Nota: en el diagrama de Gantt no se incluyeron las tareas con mayor jerarquía porque se consideraron como la unión de las tareas más pequeñas.

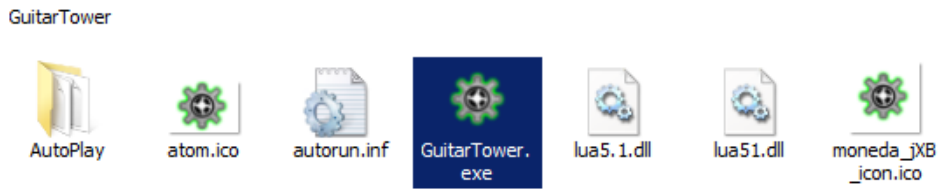
Diagrama de Gantt



Apéndice B: pantallas del juego a nivel usuario

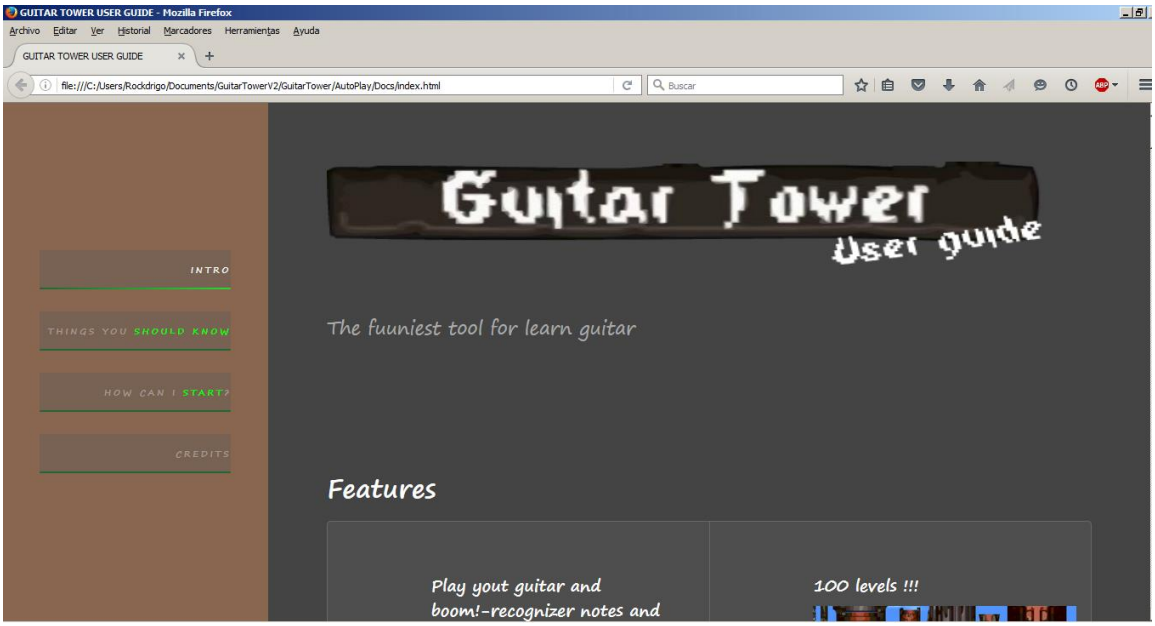
Si se desea tener una idea más clara de algunas de las mecánicas que cuenta el juego ver el siguiente video: https://www.youtube.com/watch?v=N-jUuAA9tY8

Los archivos que se incluyen en la versión para PC son los que se muestran en la siguiente figura

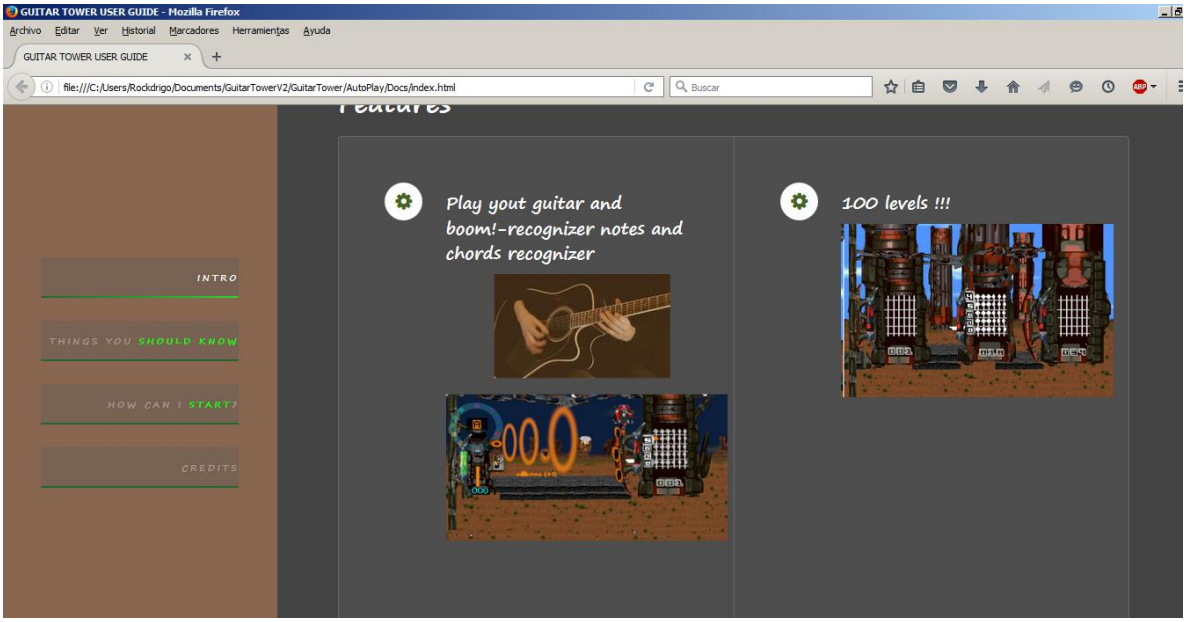


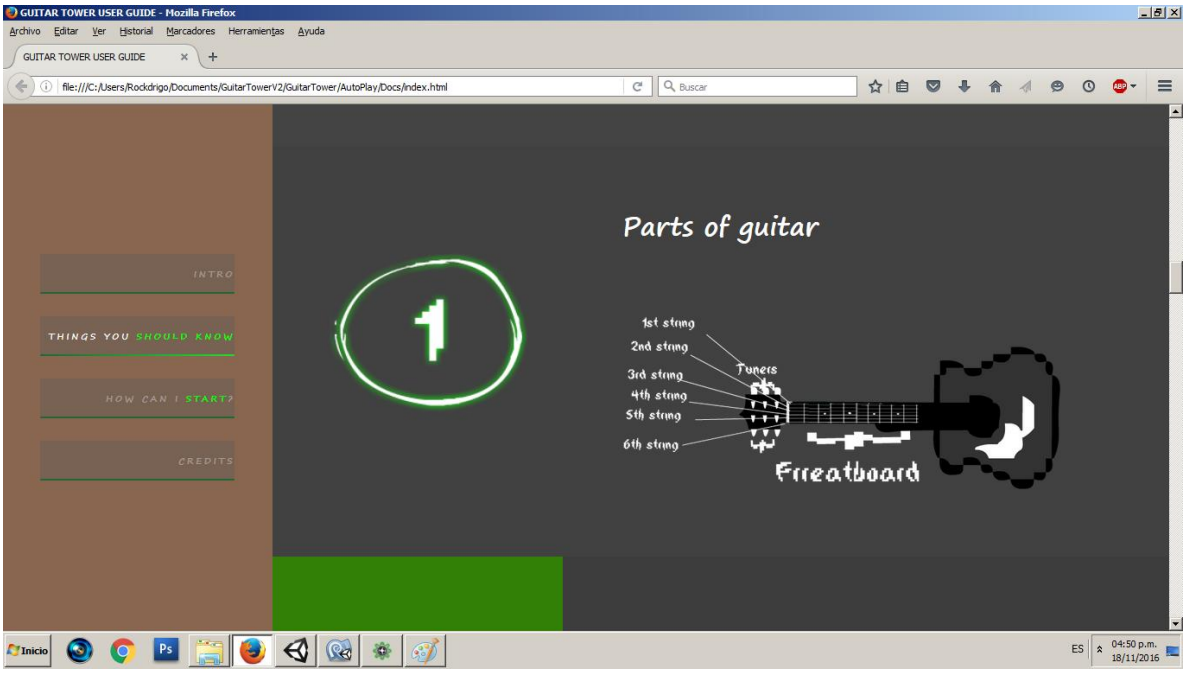
Al ejecutar GuitarTower.exe se muestra la siguiente ventana

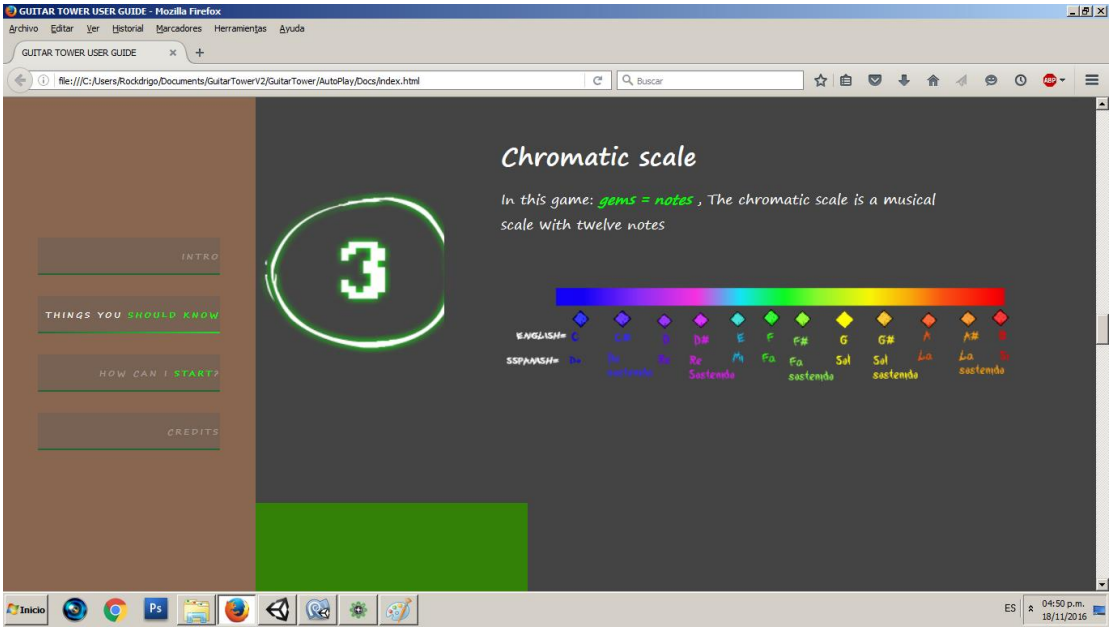
Se ofrecen dos opciones, la primera ejecuta el juego y la segunda sirve para aquellos usuarios que no saben nada acerca de una guitarra, si se elige dicha opción se mostrara la siguiente página web local.



En la página web se ofrecen diversas opciones, la primera opción muestra algunas características especiales del sistema, la segunda opción informa al usuario de que es lo mínimo que debe saber para usar el software, la tercera muestra un mini tutorial de como empezar a usar el software y la última muestra los créditos.




GUITAR TOWER USER GUIDE - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
GUITAR TOWER USER GUIDE
file:///C:/Users/Rockdrigo/Documents/GuitarTowerV2/GuitarTower/AutoPlay/Docs/index.html
Buscar
INTRO
THINGS YOU SHOULD KNOW
HOW CAN I START?
CREDITS
Parts of guitar
1
1st string
2nd string
3rd string
4th string
5th string
6th string
Tuners
Freatboard
Inicio
ES
04:50 p.m.
18/11/2016

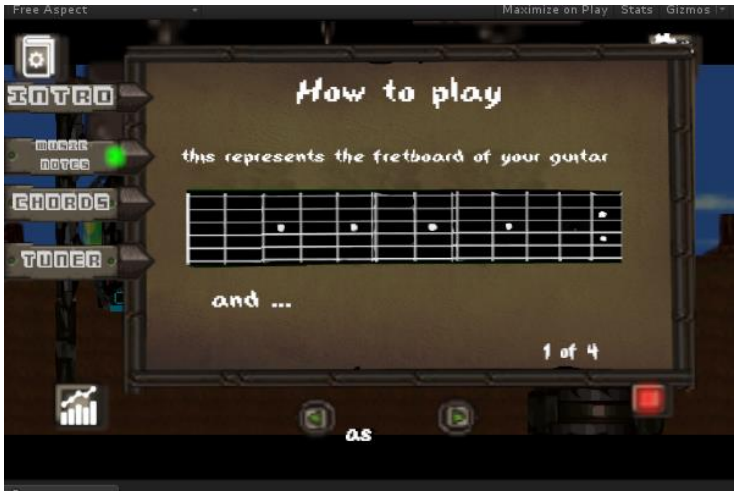

GUITAR TOWER USER GUIDE - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
GUITAR TOWER USER GUIDE
file:///C:/Users/Rockdrigo/Documents/GuitarTowerV2/GuitarTower/AutoPlay/Docs/index.html
Buscar
INTRO
THINGS YOU SHOULD KNOW
HOW CAN I START?
CREDITS
Chromatic scale
In this game: gems = notes , The chromatic scale is a musical scale with twelve notes
3
ENGLISH=
SSPANISH=
Inicio
ES
04:50 p.m.
18/11/2016

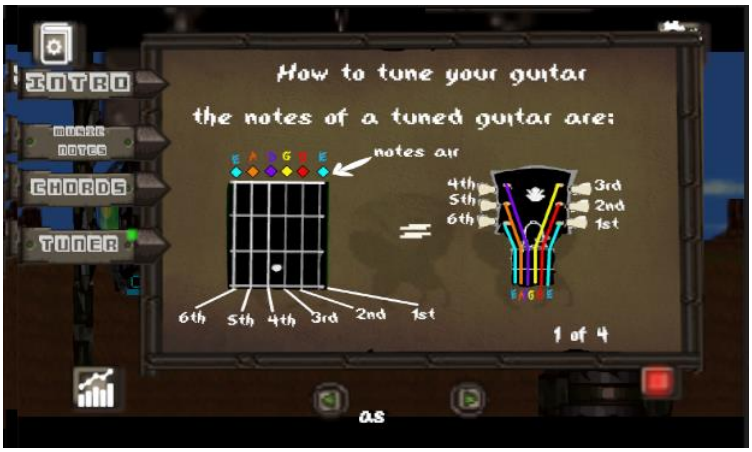
Cuando el usuario ya ha leído debidamente la guía y ha comprendido y entendido las recomendaciones que se menciona en la página o con anterioridad ya sabía todos los puntos sugeridos, entonces puede proceder a ejecutar el programa, lo primero que se encontrara será la siguiente pantalla (algunas opciones no son válidas para la versión de ordenador):

Si ya se tenían partidas guardadas entonces se muestran dos botones diferentes: "continuar" y "seleccionar otro perfil".

Alrededor del menú se ofrecen distintas opciones, si se presiona el icono de la parte superior derecha se mostrara la pantalla de configuración con las siguientes opciones:

Si se escoge la opción superior izquierda se mostrara un manual parecido al de la página con las opciones que se muestran a continuación:





Si se elige la opción de la parte inferior izquierda se muestran las estadísticas del perfil actual como se muestra en la siguiente captura:

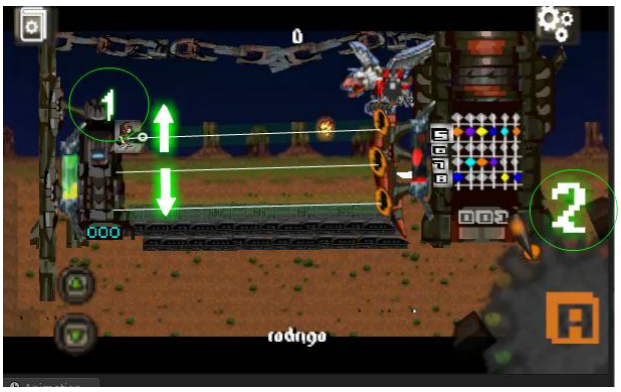
Si se escoge la opción “juego nuevo” o “seleccionar perfil” entonces se la pantalla cambia como se muestra a continuación:

Se disponen de 3 ranuras, si se da clic en cualquiera de ellas pide al usuario que elija un modo de juego:

Una vez elegido el modo de juego pide al usuario que introduzca un sobrenombre:

Si se eligió el modo de juego “sin guitarra”, lo que estará aprendiendo el usuario será a reconocer notas y acordes y a identificar la notación americana de notas y acordes por medio de los sonidos que se generan cada que cambia el color de las notas, la pantalla para este modo será la siguiente:

El usuario (1) puede mover su PC (playable carácter) en tres posiciones definidas por las líneas blancas, el objetivo será defender la torre cambiado la nota de la escala cromática (2) dependiendo el audio que sonó cuando se generó el NPC o dependiendo del color del mismo.



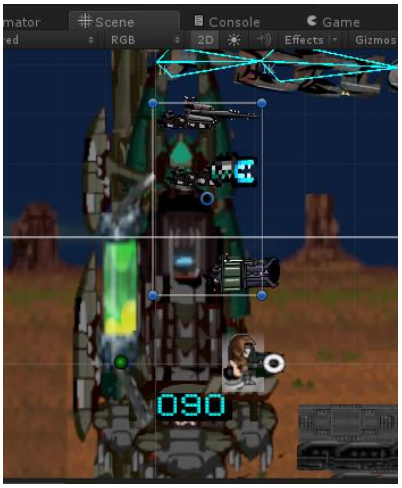
Si se elige el modo de juego “con guitarra” entonces los objetos que se mostrarán en escena serán los siguientes:

En la parte derecha se puede observar una especie de círculo cromático que indica que nota o acorde (notación en la parte central) está reconociendo la red neuronal y esto se transforma a un gráfico de la onda del color que representa la nota o acorde.

Cada vez que se pase de nivel “la torre” del jugador prendera su propulsor y lo situara en el siguiente nivel además se guardara el progreso de la partida.

Para evitar que la diversión no se pierda se agregaron puntos de experiencia representados por engranes:

Si se juntan los suficientes la torre del jugador puede desbloquear nuevos “Power-Ups” y nuevos skins:



Para finalizar, también se cuentan con 10 diferentes skins para las torres enemigas, por ejemplo: