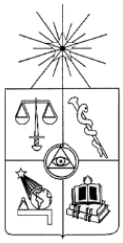

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

JUEGOS COLABORATIVOS MÓVILES DE APOYO A NIÑOS HOSPITALIZADOS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO *CIVIL EN COMPUTACIÓN*

RAMÓN IGNACIO CRUZAT HERMOSILLA

PROFESOR GUÍA:
SERGIO OCHOA DELORENZI

MIEMBROS DE LA COMISION
JOSE MIGUEL PIQUER GARDNER
JENS HARDINGS PERL

SANTIAGO DE CHILE
ENERO 2011

Resumen Ejecutivo

El objetivo general del presente trabajo de título fue desarrollar un tipo especial de juego, que funciona bajo dispositivos móviles de tipo PDA. Este proyecto nace como una manera de ayudar a atenuar los problemas de aburrimiento y aislamiento social que afectan a niños que se deben hospitalizar periódicamente; por ejemplo niños bajo tratamiento de quemaduras o cáncer. Para esto se requiere que el juego funcione sin utilizar los recursos del hospital, y que tenga además la capacidad de unirse a una red social, de manera de hacer del juego una experiencia colaborativa más completa y entretenida.

Para resolver el problema de la infraestructura de comunicaciones, se utilizó una red MANET (Mobile Ad hoc Network), la cual ayudó a hacer que la aplicación sea independiente del soporte de comunicaciones. Debido a la dificultad para construir protocolos de comunicación que ejecuten sobre una MANET, en este trabajo de memoria se utilizó una librería llamada HLMP (High Level MANET Protocol) que implementa y maneja toda la infraestructura de comunicaciones.

Como resultado final, se desarrolló un juego colaborativo móvil llamado "MagicRace", el cual es capaz de ejecutar sobre dispositivos móviles de bajo costo, como las PDA (Personal digital Assistants). El uso de este tipo de dispositivos es importante, debido a que se pretende que las instituciones de salud donde se utilice esta aplicación (particularmente hospitales públicos), no inviertan en la compra de los dispositivos sino que se espera que éstos sean donados por empresas o miembros de la comunidad.

El juego desarrollado consiste en una carrera de autos, donde se compite por equipos. Los autos durante la carrera obtienen objetos "mágicos", que provocan que el auto tenga comportamientos extraños durante la competencia. Esto resulta en un juego más dinámico, y por lo tanto se espera más entretenido, a pesar de que los dispositivos usados no permitan entregar un mejor desempeño, desde el punto de vista gráfico. Finalmente, el ganador es el equipo que obtuvo el mayor puntaje durante las etapas, haciendo que el juego motive la colaboración entre los jugadores.

Agradecimientos

Quiero dar las gracias a todas aquellas personas que me han brindado su apoyo durante mis estudios; en particular, mis padres y hermanos, mis tíos y primos, a mi polola Francisca que estuvo conmigo durante toda la carrera y a su familia.

También quiero agradecer a mi profesor guía Sergio Ochoa por su dedicación y apoyo durante todo el desarrollo de este trabajo.

Este trabajo de memoria ha sido apoyado por el proyecto LACCIR grant R1210LAC002, and Proyecto Enlace VID 2010 (Universidad de Chile), grant ENL 10/10.

Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

Hoy en día en casi todos los hospitales e instituciones de salud hay áreas dedicadas a albergar niños que requieren tratamientos médicos particulares. En general estas áreas requieren un equipamiento y un diseño especial para dar respuesta al alto nivel de energía física que usualmente acompaña a estos niños. Lamentablemente en la gran mayoría de las instituciones públicas de salud que albergan a estos menores, no cuentan con instalaciones especiales para este tipo de pacientes, sino que más bien tienen una ambientación para hacer a las instalaciones menos invasivas (por ej. dibujos en las paredes, sábanas con dibujos infantiles o un televisor para toda una sala).

En el caso de pacientes que requieren tratamientos prolongados, el problema de mantener controlado los movimientos de los niños, y a su vez hacer que los niños no se aburran, representa un problema mayor. Este es el caso, por ejemplo, de los niños que siguen tratamientos para el cáncer, que es exactamente donde se enmarca esta propuesta de memoria. Estos niños por lo general pasan gran parte del día sin muchas actividades que realizar, por lo que el aburrimiento puede llegar a ser crítico para ellos.

Junto con el problema del aburrimiento, dada la condición de su enfermedad, se encuentran mucho tiempo aislados, por lo que muchas veces los niños simplemente no comparten con nadie.

Además del aislamiento y aburrimiento, ocurre que los niños deben frecuentar el hospital más de una vez, y por períodos de más de un mes. Por la misma razón, los niños que van al mismo hospital a tratarse se ven en reiteradas ocasiones. Sin embargo, esto no implica que se generen lazos entre ellos durante su estadía, y mucho menos cuando ellos vuelven a sus hogares.

Dadas estas circunstancias y el tipo de tratamiento que llevan a cabo, es difícil mantener a los niños con un buen estado de ánimo, que contribuiría a asimilar mejor el tratamiento recibido. En el tratamiento de un paciente, sobre todo en niños, la calidad de vida de éste es un tema importante. Por esa razón el tratamiento no sólo se debe centrar en la administración de fármacos, sino también en ayudarle a mantener un buen estado emocional.

Es así como nace la necesidad de buscar distintos mecanismos que permitan mantener una buena situación anímica de los niños durante su tratamiento. Un ejemplo de esto, es el proyecto "Magia x una sonrisa" cuyo objetivo es llevar la entretención de la magia a los hospitales públicos de Santiago[1]. Esta iniciativa de los estudiantes de medicina de la Universidad de Chile,

[1] Fundación "Magia por una sonrisa"; www.magiaxunasonrisa.com

viene brindando un extraordinario aporte a los niños que están bajo tratamientos médicos prolongados. Ésta y otras iniciativas similares usualmente tienen lugar en días y horas pre-establecidos; y claramente la necesidad de este tipo de actividades supera la oferta de los voluntarios.

Este trabajo de memoria pretende abordar esta problemática, proveyendo una solución tecnológica que ayude a paliar el aburrimiento y el aislamiento social de los niños en el escenario antes mencionado. Particularmente se propone construir juegos colaborativos, capaces de funcionar en dispositivos móviles de bajo costo (por ejemplo, modelos antiguos de PDAs). Se espera que estos juegos contribuyan a generar lazos entre los niños, de modo que no se sientan tan aislados durante su estadía en el hospital. Estos juegos se integrarán a una red social para niños, que será desarrollada también en el marco de una memoria de ingeniería civil en computación, para tratar de mantener el vínculo entre los pacientes cuando éstos vuelven a sus casas. Tanto la red social, como los juegos que se desarrollen, estarán focalizados en una población destinataria de entre 4 a 12 años de edad.

1.1. Justificación del Trabajo

Se sabe que cuando los niños tienen un mejor estado de ánimo, los tiempos en que están en el hospital disminuyen considerablemente. Lo que se desea realizar en esta memoria, es ayudar a entretener a los niños utilizando tecnología de fácil acceso y bajo costo. Junto con esto se espera que a través del uso de una solución tecnológica, se puedan formar amistades entre los niños o en el peor de los escenarios, que esto sea un medio que les facilite las relaciones interpersonales durante la estadía en el hospital.

De esta forma, se lograría romper la barrera de comunicación que existe actualmente entre los niños, por lo que ir al hospital dejaría de ser un trauma para muchos niños. Como se requiere que la tecnología sea de bajo costo, se utilizarán dispositivos móviles de tipo PDA, incluyendo modelos antiguos. Por el hecho de que va a ser usado en un hospital público, se supone que no habrá acceso a redes del hospital o a Internet, por lo tanto los juegos deberán brindar una solución integral, tanto de software como de soporte para las comunicaciones entre los usuarios. El ancho de banda previsto para que funcione la solución completa son aproximadamente 200 kb/seg.

1.2. Objetivos de la Memoria

El objetivo general de este trabajo de memoria es ayudar a paliar los problemas de aburrimiento y aislamiento social que afectan a niños que están bajo tratamiento por cáncer. Para ello, esta memoria pretende desarrollar un juego colaborativo móvil, capaz de correr en dispositivos móviles de bajo costo, que puedan ser utilizados por los niños mientras están hospitalizados. Los objetivos específicos que se derivan del objetivo general, son los siguientes:

1. Diseñar y crear un juego colaborativo móvil.
2. Hacer que la solución sea independiente de la infraestructura de comunicaciones del recinto hospitalario.

3. Permitir que los juegos puedan ser integrados a una red social para niños, que compartirían los pacientes.

1.3. Requisitos de la Solución

Se realizó un análisis de las posibles características que el juego debería poseer, considerando de mayor relevancia, aquellas que logren aumentar la interacción entre los usuarios de los juegos. Es importante recordar que el juego fue diseñado para niños que actualmente reciben tratamientos de diversos tipos, por lo que pasan grandes períodos de tiempo en hospitales. Por lo tanto, también se espera que los usuarios puedan considerar el juego como un entretenimiento o distracción.

Además se espera que gracias a esta interacción, sea más fácil mejorar la comunicación entre los niños, de modo que el juego se convierta en un medio de sociabilización y los ayude a relacionarse. Es por esto que se vuelve de gran importancia el hecho de que los juegos consistan en varias etapas y que para poder ganar una etapa sea más conveniente jugar entre equipos más que como rivales, de modo que el juego sea motivador a largo plazo.

1.3.1. Requisitos Funcionales

Los requisitos funcionales se refieren a las características internas que debe poseer el juego, de manera que se cumplan los requisitos de la solución considerados anteriormente y que se facilite la interacción con los usuarios. Los requisitos finales estimados son los siguientes:

a) El juego deben tener una cantidad considerable de etapas.
b) El juego permita jugar equipo contra equipo.
c) La cantidad mínima de jugadores que debe haber es 2.
d) Deseable que el juego requiera de ingenio para ir avanzando en las etapas.
e) El juego podrá ser integrado a una red social para niños, de manera de facilitar la conectividad entre los usuarios.
f) El juego deben ser capaz de detectar automáticamente cuantos jugadores están conectados para aumentar la usabilidad de éstos.
g) El juego puede tener diferentes modalidades de escenarios para jugar:
    a. Jugador versus jugador
    b. Grupo de jugadores versus grupo de jugadores
    c. Grupo de jugadores versus jugador.

El caso jugador versus máquina no se considerará ya que uno de los requisitos del juego es que se pueda jugar por lo menos entre dos personas.

1.3.2. Requisitos de Restricción

a) El juego debe conectarse utilizando HLMP API [Rodriguez-Covilli, 2010].
b) El juego debe ser capaz de integrarse a una red social.

c) El juego se va a desarrollar para ser utilizado en pocket pc.
d) El juego no debe usar tecnología de los hospitales.

2. Trabajos Relacionados

En esta sección se presenta todo el trabajo previo que se realizó antes de comenzar con la implementación del juego. Lo que se trató de hacer fue principalmente enfocarse en aprender sobre temas que durante el desarrollo de la memoria permitieran el desarrollo exitoso de esta. Es por esto, que lo primero que se realizó fue detectar que juegos son los favoritos dentro de los niños dentro del rango de edad para el cual va dirigido.

Luego de esto, se procedió a estudiar la librería que posteriormente se usó para la comunicación entre los dispositivos más que nada para saber cómo funciona y ver si era posible usarla con los dispositivos viejos que se espera se use el juego. Además, sirvió para hacer pruebas y tomar una mejor decisión con respecto a la forma en que la información se transmitiría entre un dispositivo y otro, ya que según el desempeño que se apreciara entre los dispositivos se vería que información enviar y cual no.

Dado que para los dispositivos que se usan para el juego son antiguos y limitados en sus capacidades, no existen framework ni librerías dedicadas a la realización de juegos que permitiesen de forma exitosa el desarrollo. Es por esta razón que lo siguiente que se hizo fue investigar y aprender el uso de algún framework existente para el desarrollo de juegos. El requisito que debía cumplir este framework es que fuese sencillo de aprender y entregase resultados rápidamente. La idea también es aprender cómo se desarrolla un juego y tratar de seguir la misma lógica del framework para así disminuir el tiempo de desarrollo. El framework que se usó para este fin fue XNA 3.1. Con este framework, se hizo un prototipo que trata de seguir la misma idea que el juego “Lemmings”.

Como se mencionó anteriormente, el objetivo de realizar este prototipo fue dar un aprendizaje que a la larga fuese útil para el desarrollo final de un juego para los dispositivos. Es por esta razón que este juego quedó como prototipo y no se siguió en su desarrollo.

Finalmente, luego de haber desarrollado el prototipo, se definió cual debería ser el loop que sigue el juego. A continuación, una explicación en detalle de cada tarea que se realizó previamente:

2.1. Juegos Revisados como Ejemplo

Dado que los juegos estarán dirigidos a niños entre 4 y 12 años de edad, se procedió a revisar juegos existentes que estén pensados para estos usuarios. Adicionalmente, se espera que los juegos entretengan a los niños el mayor tiempo posible, por lo que la búsqueda se limitó a

juegos que, o bien tuvieran muchas etapas, o a juegos que tengan pocos escenarios, pero que incentiven a jugar entre varios usuarios.

Por otro lado, puesto que los juegos son para niños, lo que se hizo fue revisar un ranking de juegos online para detectar que características tienen de manera de analizar la posibilidad de incluirlas en el juego. Se revisaron 5 juegos que se encontraron en el sitio web para niños Yahoo[2], todos están implementados en flash[3] y no son colaborativos; sin embargo, el gameplay[4] es lo que se trata de rescatar de estos ejemplos.

A continuación una breve descripción de los juegos que destacan por sus características, ya que son juegos simples en su contenido y a la vez los favoritos de los niños:

1. "Taxi Gone Wild": Consiste en un auto que, para poder avanzar, debe saltar a aquellos que se encuentren en el camino. El principal desafío que presenta el juego, es que el usuario que maneja el taxi debe lograr recorrer una distancia en un tiempo determinado, si lo logra, pasa a la siguiente etapa que es más compleja por la combinación de estas variables (distancia versus tiempo). De este juego se destaca su simplicidad y posibilidad de jugar por mucho tiempo, no se ve compleja la opción de que sea colaborativo.

Figura 1: Interfaz principal de Taxi Gone Wild

2. "Backyard Sports: Sandlot Sluggers Mini Game": Consiste en un partido de baseball donde el jugador debe acertar al balón que es lanzado por un contrincante. Para ir avanzando en niveles, el jugador debe cumplir ciertos requisitos que le permitan avanzar, como por ejemplo, sumar un cierto puntaje, lanzar la bola muy lejos, etc. Éste sugiere la

[2] http://kids.yahoo.com/

[3] Adobe Flash. Aplicación multimedia usada para aportar animación, video e interactividad a las páginas Web.

[4] Toda experiencia de un jugador durante la interacción con un sistema de juego.

idea de desarrollar un juego relacionado con el deporte, ya que de esta forma se pueden tener muchas escenas distintas cambiando un par de elementos de ésta.

Figura 2: Interfaz de Backyard Sports: Sandlot Sluggers Mini Game

2.2. Estudio de HLMP API y Framework

Uno de los temas más relevantes de investigación es el que se relaciona con el estudio de la api HLMP [Rodríguez-Covilli, 2010], la cual permitirá la comunicación ad hoc entre los pockets pc. El documento explica de forma sencilla el funcionamiento y rendimiento de la api ya mencionada, y representa un apoyo para la implementación del juego a desarrollar en la memoria, ya que se tiene la certeza de su buen funcionamiento y de lo bien que se comporta en la medida que se van agregando nuevos equipos, lo que permite al alumno obtener una ventaja temporal en lo relacionado a este tema, ya que fue investigado con éxito anteriormente.

Existen diversos Frameworks que resultan interesantes de estudiar, en particular, el Compact Framework de Microsoft .NET [ONet, 2010], el cual en su sitio oficial tiene un ejemplo para probar en pocket pc, que es el hardware que se utilizará para que los usuarios jueguen.

El ejemplo sirve para aprender cómo usar Sprite[5], detectar colisiones, generar movimientos, etc. En el fondo, todo lo necesario para producir juegos para estos dispositivos. A continuación se muestra una imagen del ejemplo probado para este Framework:

[5] Pequeño mapa de bits que se dibuje en la pantalla

Figura 3: Ejemplo de Compact Framework de Microsoft .NET

Luego de estudiar este Framework, se procedió a la investigación de otros con el fin de averiguar más sobre el funcionamiento de estos y tratar de lograr al menos un prototipo que permitiese a futuro disminuir el tiempo de implementación para los juegos a desarrollar. El siguiente Framework que se investigó fue el XNA 3.1[6], el cual fue utilizado para crear un prototipo.

Lo importante de este prototipo es que se programó en el mismo lenguaje en el que se implementarán los juegos, por lo que se espera que la complejidad disminuya notablemente, sobre todo si se considera que el alumno desconocía el lenguaje en cuestión (C#).

2.3. Prototipo: Lemmings Colaborativo

La investigación anterior ayudó a realizar un pequeño prototipo de juego, que busca cumplir la mayor cantidad de requisitos posibles a considerar en la memoria. El trabajo realizado consistió en basarse en un juego conocido llamado "Lemmings"[7] e intentar incorporarle los requisitos anteriormente indicados, como por ejemplo: que el juego sea colaborativo, que esté orientado a niños y que requiera cierto ingenio por parte de los usuarios para avanzar en las etapas. Es importante señalar, que se escogió este juego a implementar por el hecho que ofrece la opción de tener muchas etapas sin mayor esfuerzo. El prototipo consiste básicamente en 3 escenarios:

1. *Escenario 1:* Consiste en realizar al usuario una serie de preguntas relacionadas con temas escolares como matemáticas, historia, etc. Cada pregunta tiene un cierto valor, en función de la dificultad de la misma. La pregunta se presenta con 4 alternativas que representan 4 posibles respuestas, donde el usuario debe escoger la que él considere correcta. Una vez que ha escogido, se obtiene un puntaje por la respuesta, el cual puede ser máximo si la respuesta es correcta, o puede ser un valor aleatorio restringido (siempre

[6] El cual permite realizar juegos ya sea para Xbox 360 o para PC.

[7] Anexo 1: Detalle Juego Lemmings

menor al máximo). Esto ocurre para que los usuarios puedan seguir participando, pero que a la vez aprendan en el juego. Al finalizar la etapa, se suma todo el puntaje obtenido y se pasa a la siguiente. A continuación se observa una imagen con un ejemplo simple del escenario 1.

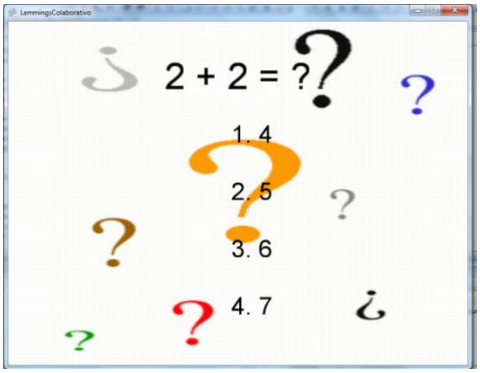

Figura 4: Imagen de preguntas

Lo rescatable del diseño de este escenario consiste en la forma en que se consigue el contenido de las preguntas, ya que este se obtiene leyendo la información desde un archivo XML[8]. Gracias a esto, es posible generar un listado enorme de preguntas que podrían permitir que el juego se utilizara en diversos escenarios educativos.

2. *Escenario 2*: Esta etapa consiste en la compra de los lemmings que se utilizarán en el juego. Existen 2 tipos de lemmings, lemmings 1 y lemmings 2, los cuales realizan distintas acciones en el juego. El lemmings 1, "bloqueador", tiene la habilidad de bloquear a otros lemmings y el lemming 2, "excavador", tiene la habilidad de excavar. La opción de compra está ligada al puntaje que se obtuvo de la etapa anterior, ya que éste se convierte en el dinero disponible para obtener los lemmings.

El valor varía según el lemming que se quiera usar, para el caso de bloqueadores es 10 y para los excavadores 15. Con esto se espera restringir el número de lemming e incentivar el aprendizaje en los niños. A continuación se observa una imagen con un ejemplo simple del escenario 2:

[8] Anexo 2: Diseño de archivo XML para obtener el contenido de las preguntas.

Figura 5: Imagen de Compra de Lemmings

Acá se observan los tipos de lemmings existentes para la compra, y el total de dinero con el que se dispone (en el extremo superior de la pantalla).

3. Escenario 3: Consiste en jugar las distintas etapas y resolver los desafíos típicos del conocido juegos "Lemmings". Al igual que en el escenario 1, la creación de la escena se hace a partir de un archivo XML el cual describe la posición en la que debe ir cada elemento de esta, como por ejemplo, los bloques, la entrada de los lemmings, la salida, etc. A continuación se observa una imagen con un ejemplo simple del escenario 3.

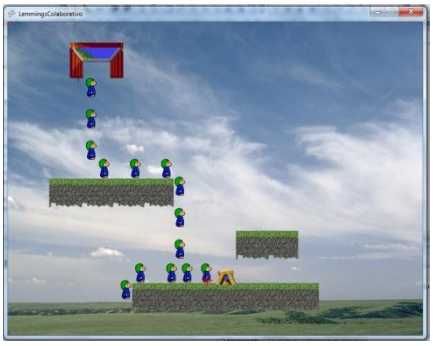

Figura 6: Etapa 1 del juego Lemmings

La importancia de este prototipo está en que se logró entender de mejor manera el funcionamiento de un Framework dedicado a juegos. A partir de eso se reconoce la gran utilidad que significará esta herramienta en el desarrollo de los juegos, debido al poco tiempo que con el que se dispone (4 meses aproximadamente).

Junto con lo anterior, el prototipo ofrece la posibilidad de jugar en forma colaborativa en todo momento. En el primer escenario un grupo de jugadores debe responder cada pregunta en conjunto y decidir cuál es la respuesta correcta, luego en la etapa siguiente entre todos los jugadores deben decidir que herramientas comprar, y en la etapa final, cada jugador representa una cierta cantidad de lemmings.

Por último, de los escenarios 1 y 3, se puede desprender la idea de obtener los datos de una escena a partir de archivos, ya sea XML o un archivo de texto plano, lo que se espera en el desarrollo del juego es obtener una escena a partir de un archivo y así disminuir el tiempo que toma desarrollar etapas.

2.4. Game loop

Luego de realizar este prototipo, se procedió a seguir investigando cómo funciona la lógica de un juego, la cual sigue los siguientes pasos:

a. *Leer eventos de entrada*: Consiste en detectar las indicaciones entregadas por los instrumentos que utilizan los usuarios, como joysticks, mouse, teclado, etc.
b. *Procesar entrada*: Una vez que el evento es detectado, se debe llevar a cabo la orden que el usuario desea que el juego realice. Por ejemplo, mover el jugador, hacer que salte, etc.
c. *Lógica de juego*: Este es el momento en que el juego debe realizar todas las acciones coherentes con lo que está sucediendo en el juego, incluyendo las consecuencias de las entradas recién procesadas. Por ejemplo, que un enemigo responda al ataque del jugador, detectar colisiones entre objetos de la escena, etc.
d. *Realizar otras tareas*: Añadiduras que logran complementar las acciones del juego, mejorando la experiencia del usuario. Por ejemplo, el sonido, animaciones de fondo, etc.
e. *Mostrar frame*: Consiste en mostrar la escena (render).

La siguiente figura muestra una idea de lo que se señala en la explicación anterior.

Figura 7: Game loop

3. MagicRace

Como ya fue estudiado, el desarrollo de un juego en un período corto de tiempo representa una tarea bastante compleja. Esta tarea se dificulta aún más al diseñar el juego para ser utilizado en dispositivos antiguos, los cuales no cuentan con muchos de los recursos necesarios para crear un juego que esté a la altura de los juegos actuales, en particular si se considera que el objetivo del juego es entretener y generar lazos entre los usuarios de este.

Cabe destacar, que se desea que el juego sea colaborativo y al mismo tiempo competitivo, por lo tanto es importante que se pueda jugar en equipos contra equipos. Además, es importante recalcar, que se hace necesario lograr que el juego sea dinámico y atractivo para los niños, ya que en caso contrario, ellos se aburren rápidamente; por lo que es deseable que el juego tenga una cantidad importante de etapas para evitar ese escenario.

Otro detalle a considerar para la decisión del desarrollo de este juego, es que los niños van a tener un espacio físico compartido, por lo tanto, la comunicación que se generará entre estos será, de manera casi inevitable, de conversación entre ellos.

3.1. Características del juego

El juego desarrollado consiste en un juego de carreras de autos llamado *MagicRace*. El juego tiene como objetivo para los usuarios, el ir avanzando en las etapas, ya que el equipo que más puntos junte será el ganador.

3.1.1. Logo

Figura 8: Logo del juego

3.1.2. Jugador individual versus equipo

Ésta es una de las diferencias más importante a considerar entre *MagicRace* y los juegos de carreras clásicos, ya que los competidores son equipos completos, no personas individuales; lo que significa que lo más relevante no es quien gane primero cada carrera, sino más bien lo importante es la suma del equipo.

En cada etapa se considera el número “n” total de competidores, y se considera ese valor como el número máximo de puntos que se pueden ganar. Claramente esos puntos son ganados por el competidor que llego en primer lugar (n), el competidor que llega en segundo lugar ganará “n-1” puntos, el tercer lugar “n-2” puntos, y así sucesivamente hasta que llegue el último competidor. Luego cada equipo sumará los puntos de los competidores que lo conforman.

Tabla 1: Puntajes según el lugar

Lugar	Puntos
1°	n puntos
2°	n-1 puntos
...	...
n°	1 punto

Por ejemplo, se tienen 3 equipos con 2 jugadores cada uno y se da la siguiente situación:

Tabla 2: Ejemplo de puntajes

Por esta razón, pese a que ganar la carrera es importante, en la medida que el equipo completo no presente un buen rendimiento, no servirá de nada ganar la carrera. En el ejemplo entregado anteriormente se observa, que pese a que el competidor 1 verde llegó primero (6 competidores, gana 6 puntos), fue superado por el equipo azul ya que ellos, como equipo, trabajaron de mejor manera.

La razón para que esto sea así tiene principalmente dos motivos:

1. Fomentar la colaboración entre los niños (usuarios) que están jugando, ya sea ayudándose o preguntando cómo van en la carrera y así lograr, en cierta medida, instancias para que estos niños puedan comunicarse entre ellos y generar un lazo.

2. Considerar la posibilidad de que haya niños que deseen dejar de jugar y se retiren de la etapa mientras estén jugando, ésta situación no detendrá el juego ya que todos los niños deberán

hacerlo también; por lo tanto si un niño deja el juego en la mitad de este, simplemente no se contará al momento de evaluar el orden en que llegaron todos.

3.1.3. Equipos

Existen 4 tipos diferentes de equipos, los cuales se diferencian por el color de auto que utilizan. El nombre de los equipos tiene relación con el color que tienen los vehículos. Los nombres de los equipos son los siguientes:

Tabla 3: Imágenes autos de cada equipo

Equipo	Sprite
Equipo azul	
Equipo verde	
Equipo rojo	
Equipo naranjo	

3.1.4. Energía

Cada jugador tendrá una energía determinada para competir, y a medida que el auto avance y cometa errores dentro de la etapa, el jugador perderá esta energía. Por error se considerará que el vehículo se salga del camino. De este modo, a medida que esto ocurra, el rendimiento del auto irá disminuyendo. Por el contrario, si pasa un tiempo determinado sin que el auto cometa errores, la energía del auto se recupera progresivamente y así vuelve a su normalidad.

Los jugadores tendrán energía para lograr dos objetivos:

1. Realizar una diferencia entre un jugador más cuidadoso en la carrera y otro que no lo es.
2. Incentivar la concentración del usuario en el juego, debido a que si pierde el control, entonces el jugador perdería energía.

Esta pérdida de energía se ve directamente reflejada con la velocidad máxima que puede alcanzar el competidor en el momento de juego, la pérdida afectará su velocidad y por lo tanto sus probabilidades de ganar y de juntar puntos para su equipo.

3.1.4.1. Máxima energía permitida

Por otro lado, el número de ocasiones que el jugador pierde el control del vehículo también será castigada, ya que la energía máxima permitida para cada vehículo irá disminuyendo en

función de éste número, es decir, luego de la i-ésima vez que el usuario pierde el control del vehículo este número disminuirá.

De todas formas, es posible que los jugadores se equivoquen bastante, por lo que si bien la barra de energía no se recuperará a un cien por ciento, el desgaste por error no será mayor a un dos por ciento de la energía original. Por ejemplo, si un jugador pierde el control del vehículo un número de veces menor a "i", su energía será representada de la siguiente manera:

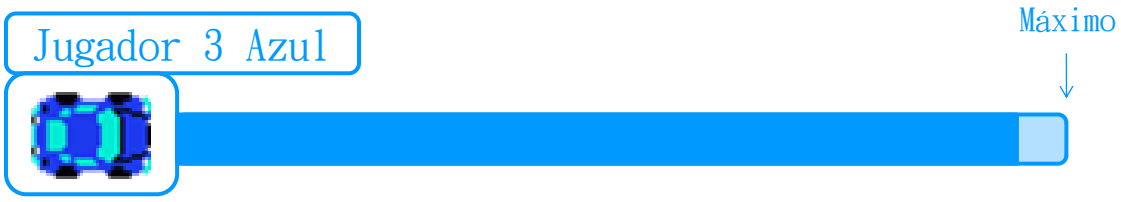

Figura 9: Ejemplo de energía máxima

Si el usuario sigue jugando la etapa y vuelve a perder el control del vehículo (esta vez un número mayor a i de veces), la barra de energía del jugador se verá así:

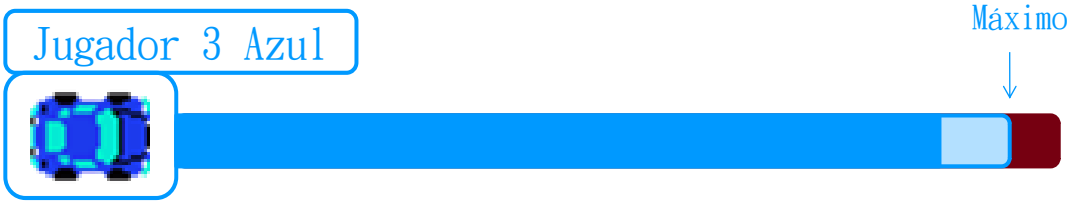

Figura 10: Ejemplo de energía máxima

Si el jugador sigue jugando la etapa y pierde el control del vehículo j veces más, con j > i, entonces la barra de energía del jugador se verá así:

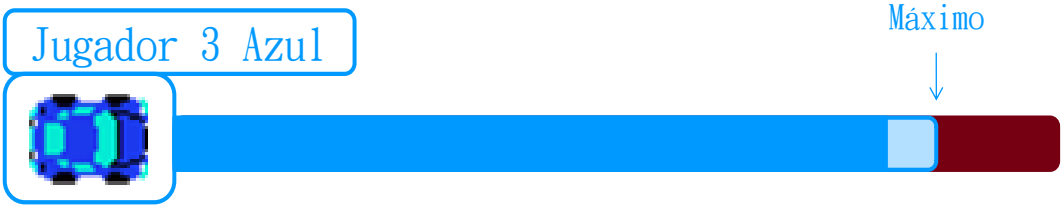

Figura 11: Ejemplo de energía máxima

El número i de veces que implican la disminución de la energía máxima permitida irá aumentando de manera que el jugador pueda seguir compitiendo. La energía máxima permitida es personal de cada jugador y se renueva en cada nueva etapa. Además dentro de la etapa existirán formas para recuperar la energía máxima permitida al 100%. Es importante señalar que el auto si va bien todo el camino su energía no se verá afectada en lo absoluto.

3.2. Objetos Mágicos

Los objetos mágicos se refieren a imágenes que los jugadores encontrarán a medida que avancen en el camino, y que cuando los jugadores pasen sobre uno de estos objetos, los autos cambiarán su comportamiento. Los objetos los pueden ayudar o perjudicar, de manera personal o grupal.

Esta idea trata de seguir un modelo similar al que tiene el juego "Super Mario Kart[9]" para la consola SNES. El jugador pasa por sobre un objeto que a priori, no conoce su utilidad, y luego de un breve período de tiempo, dentro del panel del usuario se puede ver que objeto recibió.

A diferencia del juego "Super Mario Kart", el usuario no tiene la opción de decidir el momento de ejercer este objeto; en MagicRace en el minuto que se acepta el objeto mágico se ejecutará. Esto es debido a que, cada vez que el jugador tome un objeto que le haga daño a él o a todo su equipo, decidirá no ejecutarlo para no recibir este perjuicio. De esta manera se intenta evitar la aversión al riesgo que tienen los jugadores, haciendo el juego más difícil, desafiante y entretenido.

3.2.1. Clasificación de objetos mágicos

Los objetos creados para el juego se pueden clasificar en diferentes tipos, los cuales dependen, por un lado, de si perjudica o ayuda y por otro, si afecta de manera personal o grupal a los usuarios del juego. Estos se pueden clasificar en cuatro diferentes tipos:

a) **Personales positivos**: Cuando un jugador pase por un objeto de este tipo, el rendimiento del mismo mejorará, sin importar su equipo.

b) **Personales negativos:** Cuando un jugador pase por un objeto de este tipo, el rendimiento del mismo empeorará, sin importar su equipo.

c) **Grupales positivos:** Cuando un jugador pase por un objeto de este tipo, todo el equipo al cual pertenece el usuario se verá beneficiado.

d) **Grupales negativos:** Cuando un jugador pase por un objeto de este tipo, todo el equipo al cual pertenece el usuario se verá perjudicado.

[9] http://en.wikipedia.org/wiki/Super_Mario_Kart

3.2.1.1. Personales positivos

Tabla 4: Lista de objetos mágicos personales positivos

N°	Imagen	Descripción
1		Estrella básica: Objeto que beneficia a aquel jugador que la obtenga aumentado su velocidad por 1 segundo. En caso de estar fuera del camino, no se descontará la energía.
2		Estrella normal: Objeto que beneficia a aquel jugador que la obtenga aumentando su velocidad por 3 segundos. En caso de estar fuera del camino, no se descontará la energía.
3		Estrella plus: Objeto que beneficia a aquel jugador que la obtenga aumentando su velocidad por 5 segundos. En caso de estar fuera del camino, no se descontará la energía.
4		Fantasma: Objeto que produce que el jugador no sea visto por los otros jugadores por un período de tiempo de 5 segundos.
5		Fruta: Objeto que recupera la cantidad de energía perdida que tiene en ese momento el auto.
6		
7		
8		Botiquín: Objeto que permite recuperar el 100% de la barra de energía permitida del auto.

3.2.1.2. Personales negativos

Tabla 5: Lista de objetos mágicos personales negativos

N°	Imagen	Descripción
1		Agua: Objeto que provoca que el jugador pierda el control del auto moviéndose de manera impredecible para este, por 3 segundos.
2		Tornado básico: Objeto que provoca que se cambien los controles del jugador por 1 segundo.
3		Tornado normal: Objeto que provoca que se cambien los controles del jugador por 3 segundos.
4		Tornado plus: Objeto que provoca que se cambien los controles del jugador por 5 segundos.
5		Fruta podrida: Objeto que provoca que la barra de energía baje su capacidad máxima de energía permitida en un 5%.
6		Botella de alcohol: Objeto que provoca que la energía del jugador en ese momento, disminuya en un 10%.

3.2.1.3. Grupales positivos

Tabla 6: Lista de objetos mágicos grupales positivos

N°	Imagen	Descripción
1		Rayo básico: Objeto que permite aumentar la velocidad de todo el equipo por un período de 1 segundo. En caso de estar fuera del camino, no se descontará la energía.
2		Rayo normal: Objeto que permite aumentar la velocidad de todo el equipo por un período de 3 segundos. En caso de estar fuera del camino, no se descontará la energía.
3		Rayo plus: Objeto que permite aumentar la velocidad de todo el equipo por un período de 5 segundos. En caso de estar fuera del camino, no se descontará la energía.
4		Doctor: Objeto que permite recuperar la capacidad de energía máxima permitida a un 100%, a todo el equipo.
5		Inmunidad: Objeto que permite no recibir objetos negativos por un período de 5 segundos.

3.2.1.4. Grupales negativos

Tabla 7: Lista de objetos mágicos grupales negativos

N°	Imagen	Descripción
1		Tortuga básica: Objeto que provoca que todo el equipo disminuya su velocidad en un 10% durante un período de 1 segundo.
2		Tortuga normal: Objeto que provoca que todo el equipo disminuya su velocidad en un 10% durante un período de 3 segundos.
3		Tortuga plus: Objeto que provoca que todo el equipo disminuya su velocidad en un 10% durante un período de 5 segundos.
4		Diablo básico: Objeto que prohíbe que todo el equipo consuma objetos grupales positivos durante un período de 1 segundo.
5		Diablo normal: Objeto que prohíbe que todo el equipo consuma objetos grupales positivos durante un período de 3 segundos.
6		Diablo plus: Objeto que prohíbe que todo el equipo consuma objetos grupales positivos durante un período de 5 segundos.
7		Cruz: Objeto que prohíbe que todo el equipo consuma objetos personales positivos por un período de 3 segundos.

3.2.2. Descripción de los objetos mágicos

Como es posible que varios jugadores de un mismo equipo obtengan distintos objetos mágicos de manera simultánea, en particular cuando el poder ofrecido por el objeto que un jugador del equipo en cuestión obtiene sea grupal, se tendrá que para cada jugador se ejecutarán primero sus objetos mágicos personales, y luego el efecto del objeto grupal, ya sea negativo o positivo.

Lo que ocurrirá en este escenario es que el jugador tendrá una cola de objetos mágicos, los cuales se irán añadiendo a medida que los vaya recibiendo, y los irá desencolando en la medida que el objeto mágico que tiene termine su acción.

En el caso de los objetos que provocan interacciones con otros objetos mágicos, como por ejemplo el grupal positivo de inmunidad, estos se incluirán en la cola de manera normal, pero cuando se ejecuten, lo harán de manera paralela con los objetos mágicos que siguen en la cola después de él.

Dada esta condición en el juego, a medida que existan más usuarios jugando, este provoca que sea mucho más entretenido, ya que eventualmente podrían estar recibiendo durante toda la competencia objetos mágicos. De esta forma, se lograría que los niños quieran jugar entre varios y además se tiene que nunca se tendrán los mismos objetos mágicos en las etapas, ya que estos aparecen en el juego de manera aleatoria; lo anterior implica cierta incertidumbre para los jugadores, lo cual iría en beneficio de la entretención y no de la monotonía. Además, con este hecho, se puede concluir que una etapa nunca va a ser igual, pese a que se juegue muchas veces, ya que los eventos que ocurren durante ésta hacen que sean diferentes. Por otro lado, el juego también se encargará de que no ocurran muchos eventos negativos, ya que puede que a un mismo equipo le toquen muchos eventos negativos por lo que podría provocar que los niños se aburran o se sientan frustrados.

Otro detalle a mencionar es que los objetos grupales, ya sean positivos o negativos, tendrán consecuencias en todos los integrantes del equipo, pero eso no significa que les ocurra a todos al mismo tiempo. Esto se debe a que los objetos mágicos ingresarán a la cola de cada jugador, y lo único que asegura esta situación, es que eventualmente el objeto se ejecutará.

El usuario, a medida que va jugando, podrá ver el objeto mágico que está ejecutando y una lista de los próximos que se ejecutarán.

3.3. Escenarios

El juego cuenta con un total de 30 escenarios, los cuales se diferencian principalmente por el recorrido que tienen que seguir, es decir, representan diferentes circuitos. Los jugadores partirán desde la línea inicial y comenzarán a avanzar en el recorrido adecuado, en una dirección determinada; y como es una carrera, el orden final de los jugadores se verá según el orden de llegada a la meta. Sin embargo, como se mencionó previamente, los ganadores de un escenario serán aquel equipo que en suma tuvo un mejor rendimiento durante la carrera.

Figura 12: MagicRace

La posición de un jugador durante la carrera se irá actualizando a medida que éstos pasan a través de unos *checkpoints* durante la competencia. Los checkpoints serán representados por unos conos de color naranjo como el de la siguiente figura:

Tabla 8: Tiles con checkpoints

Imagen	Descripción
	Cono inferior
	Cono derecho
	Cono superior
	Cono izquierdo

De esta forma, el jugador no sabrá en tiempo real su posición actual, pero dado que los *checkpoints* se encontrarán separados de distancias cortas, no será tan importante que esto ocurra.

El número de vueltas asociados a cada escenario dependerá del tamaño de la etapa. Si la etapa tiene un circuito muy largo, la carrera tendrá muy pocas vueltas para evitar que sea muy monótono el juego, en caso contrario, la cantidad de vueltas será mayor.

Las etapas a su vez tendrán una característica diferenciadora entre ellas que hará que la probabilidad de que un tipo de objeto mágico ocurra, sea diferente en las distintas etapas. Junto con esto, las etapas indicarán el tipo de escenario al que los jugadores se enfrentarán con su nombre. Las etapas se clasifican en los diferentes tipos:

1) Etapas positivas: En estas etapas la probabilidad de que un objeto mágico sea positivo es mayor.
2) Etapas negativas: En estas etapas la probabilidad de que un objeto mágico sea negativo es mayor.
3) Etapas grupales: En estas etapas la probabilidad de que un objeto mágico sea grupal sin importar si es positivo, negativo o neutral es mayor.
4) Etapas neutras: En estas etapas la probabilidad de los objetos mágicos son iguales.

3.4. Niveles del juego

La dificultad del juego tiene una directa relación con la velocidad a la cual los autos pueden correr, ya que en la medida que la velocidad del auto aumente el control del auto disminuirá, de lo contrario si la velocidad del auto disminuye, el control del auto también aumentará.

Por otro lado, los escenarios siempre serán los mismos, por lo que la dificultad no tendría relación con que las etapas serían diferentes, ya que como se acaba de mencionar, la dificultad se asocia a la velocidad que andan los autos.

Los niveles del juego, al igual que el juego “Super Mario Kart”, son 3 y el nombre de estos tiene relación con eventos mágicos, para darle mayor importancia al hecho de que el juego se llama “MagicRace”.

El nombre de los niveles son los siguientes:

1) Magia verde: Nivel más simple del juego, los autos andan a una velocidad máxima que es lo suficientemente lenta para que los niños con menor motricidad fina lo puedan jugar.
2) Magia blanca: Nivel normal del juego, los autos andan a una velocidad máxima que se considera normal para un niño que tiene 8 años.
3) Magia negra: Nivel más difícil del juego, los autos andan a una velocidad máxima tal que el control del auto es muy complejo.

Los nombres escogidos tienen que ver directamente con el significado que tienen en la vida real. El caso de la magia verde, se asocia a toda aquella magia cuyo componente central es el uso de plantas y hierbas. Al ser considerada un tipo de magia blanca, se considera que esta es un subconjunto por lo que se entiende que es un nivel inferior dentro del juego. Por otro lado, se denomina magia blanca a aquellos actos mágicos cuya naturaleza, métodos u objetivos son comúnmente aceptados por la sociedad donde se producen. Finalmente, se denomina magia negra a aquellos actos mágicos cuya naturaleza, métodos u objetivos no son comúnmente aceptados por la sociedad donde se producen.

Claramente no se espera que un niño menor a 12 años tenga estos conceptos adquiridos sobre magia, sin embargo, se espera que dado estos nombres, se pueda intuir fácilmente la dificultad asociada a ella.

Por último, la explicación que se da sobre los nombres es una forma de justificar el por qué cada nombre se asocia a cada nivel de dificultad del juego.

3.5. Características de los Usuarios

En MagicRace se consideran dos tipos de usuarios, los que poseen las siguientes características:

1. *Host del juego*: Usuario que tendrá la posibilidad de escoger todo lo relacionado con las opciones que da MagicRace para jugar. Se espera que este usuario sea capaz de definir características de las etapas que escogerá, tales como: la dificultad de ésta, la cantidad de etapas a realizar y las características propias del usuario, como por ejemplo el equipo al cual pertenece.

2. *Jugador:* Usuario que realiza las etapas a medida que aparecen en el juego, pero que no las escoge. Además interactúa con otros jugadores ya sea en forma colaborativa, o como una competencia.

Por último, cabe señalar, que el juego se llama MagicRace principalmente por dos razones:

1. Crear un juego de palabras entre el nombre del juego y la "Fundación Magia por una sonrisa".
2. Se relaciona con los objetos mágicos que adquieren los jugadores durante la carrera y el hecho que los autos cumplen fenómenos que en la realidad no ocurren, como por ejemplo, el hecho que los autos no choquen entre ellos o que pasen uno por sobre otro.

4. Diseño del Sistema

Dado que la aplicación es un juego distribuido, se consideran 3 componentes como las más importantes para su buen funcionamiento: el juego, la comunicación y los formularios.

Estos componentes son importantes porque permiten separar la aplicación y así facilitar la escalabilidad del programa. Además facilitan conectar el juego a otras aplicaciones sin previo conocimiento de las mismas, ya que de esta manera la conexión sucede de forma natural. Esto es relevante, ya que es parte de los objetivos iniciales el permitir que una red social pueda utilizar estos juegos.

Por otro lado, la modularidad de la aplicación facilita el rescatar parte de ésta para futuras aplicaciones. A continuación se explicará en detalle cada una de las componentes para entender su funcionamiento:

4.1. Juego

Esta componente es la que contiene todo lo relacionado con el juego, es decir, las imágenes, reglas, escenarios y todas aquellas características que hacen posible que el juego funcione correctamente y tenga lógica. El juego se divide en distintas funcionalidades requeridas para su uso, las cuales son:

4.1.1. Entrada de comandos

Existen muchas variables que podrían resultar en el éxito o fracaso del juego, y una de estas es tan simple como la elección de las teclas que se requieren para realizar los movimientos que permitan que el juego funcione de la manera esperada; esto es ya que, si los usuarios finales del juego no se sienten cómodos con las teclas escogidas, podría resultar en la no utilización del juego, no cumpliendo con el objetivo principal del mismo que es que los niños se entretengan.

Por lo mismo, un requerimiento deseable dentro del juego es la opción de configurar la entrada del dispositivo a gusto del usuario o dar la opción de que éste en cualquier momento pueda decidir que hace cada entrada en el juego.

Pese a que el juego no tiene esta alternativa, es importante destacarla, ya que si eventualmente el público objetivo al cual está dirigido no puede usar el teclado como se espera, el costo de cambiar el teclado, desde el punto de vista del desarrollo; es prácticamente nulo.

Para lograr esta solución, se crearon 2 clases que se preocupan de esta tarea, una es la que representa la acción que realiza la entrada del teclado, la cual se llama GameAction. El objetivo de esta clase es detectar si la tecla que representa cierto movimiento fue presionada o no, si fue liberada y la cantidad de veces que ha sido presionada.

Es importante señalar en este caso, que la entrada no es necesariamente una tecla, ya que también existe, para las pocket pc, el "stylus" que funciona igual que un mouse en un computador. En el caso de los dispositivos más nuevos, se podría agregar el acelerómetro, pero esa tarea quedaría como un trabajo a futuro y sólo en el caso que fuese posible tener ese tipo de dispositivos.

La segunda clase que se creó para procesar la entrada se refiere a un administrador de acciones. La idea de esta clase es que las acciones se "suscriban" a ésta para luego asociar el evento de entrada con respecto a la acción que representa.

De esta forma, todos los eventos relacionados con el input están unidos en una misma clase, mejorando la eficiencia de la aplicación y simplificando eventuales cambios de la misma.

4.1.2. Despliegue de imágenes (Renderer)

El despliegue de imágenes consiste en mostrar todas las figuras que componen el escenario y todas aquellas que, unidas, generan la sensación que un objeto esté en movimiento (avanzar, retroceder, rotar, etc.).

Cada vez que el usuario hace que su avatar se mueva de un punto a otro en el espacio del juego, puede cambiar en algún nivel el escenario en el que se encuentra, o la dirección en la que se mueve el avatar. Para que esta tarea se realice de manera lógica para los usuarios se crea un "sistema de dibujo por ciclo", esto significa que en el gameLoop [10]; se cargan todas aquellas imágenes que se modificaron después de un período de tiempo determinado, llamado *ciclo*, ya

[10] Revisar 2.4

sea por la lógica del juego (cambios en el escenario porque se trasladó), por el usuario (porque el avatar rotó), etc.

El encargado de realizar el "sistema de dibujo por ciclo" es la clase Renderer la cual se preocupa de dibujar todos los objetos que deben estar en la escena al momento que el usuario está jugando.

En una primera instancia, la solución encontrada fue la de actualizar la imagen cada vez que se cumpliera un *ciclo*. Esto puede provocar, en dispositivos con muy pocos recursos, parpadeos al momento de mostrar la imagen, en particular cuando el número de imágenes es muy grande. Es importante recordar que el juego estará dirigido a niños, que se encuentran hospitalizados por variadas razones; por lo que es posible, que muchos usuarios utilicen el juego a la vez, disminuyendo su rendimiento. Al ser esta una posibilidad, aunque tal vez pequeña, que disminuye la calidad del juego, es importante considerar este escenario ya que también puede llevar al fracaso del juego si los niños consideran estos parpadeos como una molestia.

Por esta razón fue necesario acudir a una metodología diferente, para así considerar esta posibilidad, para lo cual se utilizó una técnica llamada "Off Screen Bitmap Technique for painting". Esta técnica consiste básicamente en los siguientes pasos:

a) La creación de un bitmap fuera de la pantalla
b) Obtener un Graphics para este.
c) Realizar todas las operaciones necesarias para mostrar una imagen en este Graphics (en memoria)
d) Copiar el resultado del bitmap creado fuera de la pantalla en el que se muestra en el juego.

4.1.2.1. Sprites

En computación gráfica, un sprite[11] es una imagen en dos dimensiones o una animación que va agregada dentro de la escena, con el fin de representar algún objeto.

En el caso de *.NET compact framework*, los sprites son representados a través de rectángulos en los cuales se agrega la imagen que represente el objeto. El desafío que se presenta en este caso, es que la mayoría de las figuras que se usan no son rectangulares, por lo que se hizo necesario buscar una alternativa para mostrar las imágenes dando la impresión que el sprite es curvo.

[11] También es conocido por otros nombres, como por ejemplo: Gráficos jugador-proyectil, Movable Object Block, OBJ, etc.

Lo que se hizo en este caso fue usar una técnica llamada “Color Key Transparency” la cual consiste en sacar un color de la imagen al momento en que se hace el rendering de esta.

El problema que tiene .NET es que sólo puede sacar un solo color de la imagen, por lo tanto, lo que se debe hacer para dar la sensación de una figura es pintar todo el borde de un mismo color. Para pintar las imágenes de un mismo color se usó el programa Paint.net, el cual es un editor de fotos open source gratuito para Windows. Para cargar las imágenes, se procedió a cargarlas como un recurso embebido.

El siguiente desafío que se abordó para los sprites fue que .NET tampoco es capaz de realizar rotaciones, por lo que llegó a ser un problema crítico para el juego, ya que si no era posible solucionar esto el juego quedaría muy limitado.

Lo primero que se hizo para solucionar esto fue tener una imagen cada una cierta cantidad de ángulos de modo que diera la impresión que el objeto efectivamente rota. El ángulo que se consideró para esto fue de 3 grados, por lo tanto, por cada sprite se requerirían 120 imágenes. Esto sin duda es una solución al problema, sin embargo, es una alternativa poco eficiente, ya que el costo de colocar cada sprite se vuelve muy elevado. La solución final fue la utilización de openNETCF, ya que provee la opción de hacer 4 rotaciones; 0, 90, 180, 270 grados.

Esto significa que se generaron imágenes cada 3 grados entre 0 y 90 teniendo un total de 31 imágenes rotadas. Luego, al momento de cargar las imágenes, se procede a usar openNETCF con el objetivo de tomar cada una de estas imágenes y luego rotarlas para que haya una imagen por cada cuadrante, esto es, rotar el ángulo respectivo en 90,180 y 270 grados respectivamente.

Figura 13: Sprites de rotación, auto azul

4.1.2.1.1. Animación

Para dar la sensación de que los sprites son animados, lo que se hace en este caso es separarla en dos partes. La primera tiene relación con lo explicado previamente, esto es, que el objeto rote y se traslade. La segunda tiene que ver con el hecho que uno podría desear que el sprite al momento en que realiza alguna acción, ya sea trasladarse o rotar, en ese mismo instante este genere otro movimiento sin perjuicio de la acción realizada. Por ejemplo, si un sprite se mueve alrededor de otro podría ser que este cambiara de color, tirara humo o escribiera algo. Para

realizar esta tarea, se requiere una acción distinta a la que permite que el sprite realice movimientos.

Lo que se hace en este caso es darle al sprite imágenes que representen la acción extra que quiere realizar con los respectivos tiempos asociadas a esta animación. La explicación de la implementación de esto, se verá en detalle en el próximo capítulo.

4.1.3. Administrador de recursos (Resource Manager)

Para cargar todas las imágenes que luego serán utilizadas ya sea para representar un sprite o un tile, lo que se tiene es un administrador de recursos, el cual se preocupa de cargar todas las imágenes necesarias y las deja listas para ser pedidas por cualquier otro objeto que la requiera; de esta forma no es necesario pedir cada vez la imagen que se requiera al disco, mejorando la eficiencia de la aplicación.

4.1.4. Mapas

Dado que el juego es 2D, el mapa de una etapa en particular contendrá varias pantallas del dispositivo de ancho y largo. El tamaño por lo general puede depender de la etapa, pero perfectamente podría tener 20 o 30 pantallas de espacio. En estos casos la imagen se va moviendo a medida que el jugador principal también lo hace.

La primera y peor solución posible sería tener una sola imagen que representase al mapa. El problema de esta opción es que claramente requiere mucha memoria y es muy posible que haya algunos dispositivos que no puedan siquiera cargar el juego.

Es por esto que la solución por la que se optó fue usar mapas basados en tiles. Esta alternativa lo que hace es dividir el mapa en una matriz donde cada coordenada tiene una referencia a un objeto. En caso de no existir nada en alguna coordenada en particular, entonces el objeto es nulo y al momento de representar la imagen se dibuja el fondo, que termina siendo un rectángulo con un color determinado.

Un tile es una imagen cuadrada que sirve para formar la imagen total del mapa y el tamaño que se escogió fue de 32x32 pixeles: El tamaño se decidió principalmente a que era deseable que fuese una potencia de 2 para así simplificar algunos cálculos y además mejorar la eficiencia de estos, tema muy importante para el juego ya que los cálculos que se deben hacer para mostrar el mapa se usan siempre y por lo mismo es crítico que no use muchos recursos, para que el juego funcione correctamente.

Otra ventaja que se obtiene con esta técnica es que se necesitan muy pocas imágenes para crear escenarios mucho más complejos, por lo que desde ese punto de vista también hay una mejora, por otro lado, también se tiene el beneficio que se puede usar el mismo tile para mostrar diferentes partes del mapa.

Por último, cabe mencionar, que esta técnica entrega como beneficio el hacer muy simple determinar sobre que está parado un jugador durante el juego, por lo que establecer alguna interacción sobre este no tiene mayor complejidad.

4.1.4.1. Cargar un mapa

Dado que la idea es que el juego cumpla la función de entretener por largos períodos de tiempo a los niños, se espera que tenga muchas etapas distintas en las cuales puedan jugar, sin que esto vaya en perjuicio del rendimiento de este. Además se espera que el modo en que se cambie un mapa a otro no tenga mayor conflicto, por lo tanto, el ideal que se debe cumplir es que al momento de terminar de jugar una etapa se pase a la siguiente sin que los usuarios tengan mayores problemas.

En muchos casos, los juegos tienen un editor de escenarios para la creación de mapas. Para el alcance de esta memoria, esa opción fue descartada, ya que el tiempo necesario para hacer todas las tareas necesarias para el desarrollo del juego provocó que no fuese una necesidad crítica dentro de éste. Sin embargo, se trató de buscar una forma que se acercase lo más posible a una forma de crear mapas que fuese simple y rápida.

Para el juego, lo que se hizo fue generar archivos de texto que representaran un mapa. De esta forma, simplemente cada carácter dentro del archivo representa un tile o algún sprite que debe ser mostrado en el mapa. En caso de haber alguna letra que no represente ningún tile o sprite simplemente el cargador de mapas lo ignora y sigue leyendo el archivo.

Para determinar el ancho del mapa simplemente toma la línea con la mayor cantidad de caracteres. El largo se considera como el total de líneas válidas dentro del archivo. Parsear el archivo con el mapa es relativamente simple y consiste en los siguientes pasos:

- Leer cada línea, ignorando las líneas comentadas y agregar cada una de estas en una lista.
- Crear un objeto que represente el mapa. El ancho y largo se definen como se explicó anteriormente.
- Tomar cada carácter en cada línea y agregar el tile o sprite apropiado al mapa según el carácter que sea. Si el carácter no está considerado como los apropiados el tile se considera como vacío y no se dibuja nada en esa coordenada.

4.1.4.2. Dibujando mapas

Como se mencionó anteriormente, los mapas basados en tiles son mucho más grandes que la pantalla del dispositivo, por lo tanto, sólo una parte del mapa es mostrado en la pantalla en un tiempo dado. Cuando el jugador se mueva, el mapa se mueve manteniendo, la mayor parte del tiempo, al jugador en el centro de la pantalla.

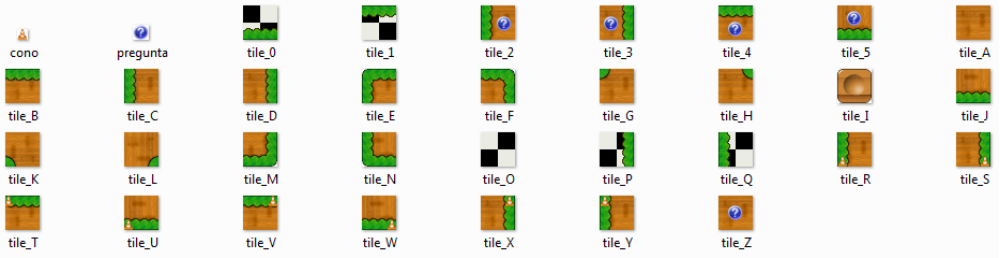

Figura 14: Tiles para hacer el mapa

El detalle de cómo se hace esto, se explicará en la implementación.

4.2. Comunicación

La comunicación en el juego se definirá como toda aquella información que es necesaria para que un dispositivo se comunique con otro de modo que permita el desarrollo del juego exitosamente.

Dado que uno de los requisitos es no usar tecnología perteneciente a los hospitales, se hizo necesario que los dispositivos se pudieran comunicar, independientemente del lugar donde estén. Para esto, se puede utilizar el concepto de una MANET[12], el cual resuelve de manera exitosa este conflicto. Sin embargo, la dificultad para construir protocolos de comunicación que permitieran desarrollar este concepto es demasiado alto y por lo mismo podría provocar que el avance de la aplicación fuese muy poco en relación a la creación de un juego. Es por esto que otro de los requisitos es utilizar una librería que ya realiza esta tarea. Esta librería llamada HLMP API, proporciona todas las herramientas necesarias para poder cumplir con este objetivo de manera exitosa.

Para realizar la comunicación entre el juego y la librería, primero se procedió a definir como se enviaría la información entre los dispositivos. El requisito que debía cumplir para el envío de la información, es que un dispositivo le debe despachar la menor cantidad de datos posible y al mismo tiempo que entregase la mayor cantidad posible de información a todos los dispositivos para que puedan procesarla correctamente.

Por un lado, se necesita enviar toda la información relacionada con las opciones que el usuario escoge para jugar, como por ejemplo, quien inicia la primera etapa, a que equipo pertenece, etc.

Para enviar esta información, lo que se hace es utilizar la opción que ofrece la librería de agregar información al usuario a través del arreglo UplayerData del NetUser que entrega la librearía HLMP API.

[12] Mobile Ad hoc Network

La ventaja que se obtiene con este arreglo, es que se asegura que la información llegará cada cierto período de tiempo, con los datos que la librería envía desde un usuario a otro. La desventaja que tiene este arreglo, es que el período en el que envía la información es muy lento para efectos del juego (cercano a un segundo aproximadamente) por lo tanto, es necesario buscar otro mecanismo que permita el envío de información más rápido, para el tipo de información que se requiera con mayor urgencia.

En este caso, lo que se hace es generar un protocolo de mensaje nuevo, el cual utiliza una arquitectura UDP usando Multicasting. La razón por la cual se usó una arquitectura UDP, se debió a que permite mejorar la rapidez y eficiencia del mensaje notablemente entre los jugadores, principalmente de los movimientos y ciertos eventos en los cuales no es tan grave si la información llega o no.

La arquitectura UDP no asegura que los mensajes lleguen a los destinatarios, sin embargo, aumenta la velocidad del mensaje. Por otro lado, se usó Multicasting para aprovechar la opción que entrega la librería de hacerlo. De este modo se evita que la aplicación se preocupe de revisar a quien enviar los mensajes, minimizando notablemente la complejidad para realizar esta tarea.

La explicación de los protocolos de comunicación y los mensajes se explicará con mayor detalle en el próximo capítulo.

4.3. Formularios

Los formularios se crearon para ser la cara visible de la aplicación, aunque también sirve para conectar el juego con cualquier aplicación que lo requiera. La razón por lo cual se realizó esto fue para cumplir con uno de los objetivos del trabajo, el cual consiste en que el juego pueda ser utilizado para una red social. En este caso, se asume que la aplicación a la cual el juego está "conectado" cumple el mismo fin que si fuese una red social, por lo tanto, para el caso en que la red quiera utilizar el juego, simplemente tiene que seguir las reglas necesarias para que esto ocurra. Como en todas las explicaciones anteriores, el detalle de la implementación se comentará en el próximo capítulo.

Esta componente se divide en 4 tab los cuales despliegan información necesaria para el funcionamiento correcto de la aplicación. A continuación se muestran imágenes de tabs del juego, a excepción del tab donde muestra los usuarios conectados:

Figura 15: Tabs del juego

El primero sirve para que los usuarios se puedan conectar a la MANET. Una vez que el usuario se ha conectado, los otros tab comienzan a ser útiles y por lo tanto, vale la pena observarlos.

El segundo tab muestra los usuarios que están conectados a la MANET, incluyendo al mismo usuario que visualiza el tab. Una de las gracias que tiene este tab es que junto con ver a los usuarios conectados, también ve la calidad de su conexión dentro de la MANET.

El tercer tab consiste en definir las opciones para poder jugar en el juego. Una vez que las opciones se escogieron, se acepta y se guarda la información en el uplayerData del NetUser para que posteriormente los otros usuarios de la red puedan pedir esta información y así agregarla en su aplicación. Por último, el cuarto tab es donde los usuarios pueden jugar el juego.

5. Implementación del Sistema

En este capítulo se explicará el detalle de la implementación de las componentes explicadas en el capítulo anterior, debido a que era necesario explicar la implementación para describir el comportamiento de estas de una manera más precisa y correcta.

5.1. Juego

Las clases que están relacionadas con el juego se pueden dividir según el comportamiento que tengan dentro de esta componente. Por un lado están las clases que tienen que ver netamente con el juego, por otro lado están aquellas que tienen relación con la comunicación del juego a través de la MANET y por último, están las clases que se relacionan con los formularios para que este pueda ser visto.

El juego tiene 2 clases como las más importantes. La primera es una clase abstracta llamada GameCore donde lo crítico dentro de esta clase es que acá se encuentra el gameLoop del juego. Luego, la segunda clase importante se llama Game, esta clase hereda de GameCore, la razón de hacerlo de esta forma se debió a que en un principio pareció importante separar estas clases debido a que si posteriormente se desea hacer otro juego, el costo de este sería mucho menor.

5.1.1. Entrada de comandos

Como se mencionó en el capítulo anterior, para la entrada de comandos se usan 2 clases que se preocupan de esta tarea. La primera se llama GameAction y lo que hace esta clase es representar alguna tecla presionada durante el desarrollo del juego. Esta clase devuelve un valor en caso que la tecla fue presionada o no, si fue liberada o no y la cantidad de veces que ha sido presionada.

Los atributos de esta clase se pueden separar en los que son de uso interno y los que sirven para obtener información que determina si el evento de presionar la tecla se realizó o no. Los atributos son los siguientes:

```
/// <summary>
/// estados de la accion
/// </summary>
public static int NORMAL = 0;
public static int DETECT_INITAL_PRESS_ONLY = 1;
private static int STATE_RELEASED = 0;
private static int STATE_PRESSED = 1;
private static int STATE_WAITING_FOR_RELEASE = 2;


/// <summary>
/// nombre que recibe este GameAction
/// </summary>
private String name;
```

```
/// <summary>
/// cantidad de veces que fue presionado este GameAction
/// </summary>
private int amount;
/// <summary>
/// Estado en que se encuentra este GameAction
/// </summary>
private int state;
/// <summary>
/// nombre que recibe este GameAction (está creado como
    propiedad)
/// </summary>
public string Name;
```

Es importante que los métodos sean llamados una sola vez y asegurar que la cantidad de veces en que fue presionada la tecla sea la correcta. Para esto se utilizó la opción de escribir en el encabezado de cada método lo siguiente:

```
[MethodImpl(MethodImplOptions.Synchronized)]
```

Esto permite que si la tecla fue presionada varias veces, se ingresará al método sólo una vez que este haya salido. Los métodos más importantes dentro de esta clase son: isPressed() e isReleased(), los cuales consisten en saber si la tecla fue presionada o liberada respectivamente. El resumen de la clase se muestra a continuación:

```
/// <summary>
/// Crea un nuevo GameAction con el comportamiento normal
/// </summary>
/// <param name="name">nombre que recibe el GameAction</param>
public GameAction(String name);
/// <summary>
/// Resetea este GameAction
/// </summary>
public void reset();
/// <summary>
/// Tap , es lo mismo que press() seguido por release()
/// </summary>
public void tap();
/// <summary>
/// señal que la tecla fue presionada
/// </summary>
public void press();
/// <summary>
///  Señal que la tecla fue liberada
/// </summary>
public void release();
/// <summary>
/// Retorna si la tecla fue presionada o no
/// </summary>
public bool isPressed();
/// <summary>
/// Retorna si la tecla fue liberada o no
/// </summary>
```

```
public bool isReleased();
/// <summary>
/// Para la tecla, este es el numero de veces que esta fue
    presionada
/// </summary>
/// <returns></returns>
public int getAmount();
```

La segunda clase que se creó para procesar la entrada se refiere a un administrador de estas acciones. La clase se llama InputManager y debe recibir como parámetro en su constructor a que Control se asocia los eventos de entrada. Luego, el constructor suscribe a los eventos de entrada con el componente recibido dentro de éste. Hay que recordar que el objetivo que se persigue con esto es que exista solo un objeto que se encargue de recibir los eventos de entrada, ya sea para simplificar esta tarea, como para también mejorar la eficiencia. El código que hace esto es el siguiente:

```
/// <summary>
/// Crea un nuevo InputManager que escucha la entrada desde un
    componente especifico
/// </summary>
public InputManager(System.Windows.Forms.Control comp)
{
      interfaceActions = new Hashtable();
      this.comp = comp;
      comp.KeyDown += new
            System.Windows.Forms.KeyEventHandler(this.keyDown);
      comp.KeyUp += new
            System.Windows.Forms.KeyEventHandler(this.keyUp);
      comp.MouseDown += new MouseEventHandler(MouseDown);
      comp.MouseUp += new MouseEventHandler(MouseUp);

  }
```

Para que los GameAction puedan suscribirse a esta clase lo que se hizo fue tener dentro de la clase Game 4 GameAction's que representen los movimientos básicos que se espera que haga el juego, estos son:

- moveLeft: GameAction que representa la acción que permite moverse hacia la izquierda.
- moveRight: GameAction que representa la acción que permite moverse hacia la derecha.
- moveUp: GameAction que representa la acción que permite moverse hacia arriba.
- moveDown: GameAction que representa la acción que permite moverse hacia abajo.

Es importante señalar, que las acciones representan una acción y no una tecla que haga esto, ya que eventualmente más de una tecla puede realizar la misma acción durante el juego.

Dentro de la clase Game existe un método que se llama initInput() el cual consiste en asociar las acciones a una respectiva tecla y luego dar esta información a la clase InputManager. El detalle de este método se muestra a continuación:

```
/// <summary>
/// inicializa el administrador de entrada
/// </summary>
private void initInput()
  {

      moveLeft = new input.GameAction("moveLeft");
      moveRight = new input.GameAction("moveRight");
      moveUp = new input.GameAction("moveUp");
      moveDown = new input.GameAction("moveDown");

      inputManager = new input.InputManager(control);

      inputManager.maptoInterface(moveLeft, (int)Keys.Left);
      inputManager.maptoInterface(moveRight, (int)Keys.Right);
      inputManager.maptoInterface(moveDown, (int)Keys.Down);
      inputManager.maptoInterface(moveUp, (int)Keys.Up);
      inputManager.maptoInterface(moveUp,
                                  (int)MouseButtons.Left);

   }
```

Como se puede apreciar la clase InputManager tiene un método, el cual consiste en realizar un "mapeo" entre la acción y una tecla del dispositivo. A su vez, la clase InputManager tiene una tabla de Hash para realizar estos "mapeos", la cual asocia la acción con la respectiva tecla. Es importante señalar, que como se mencionó en el capítulo anterior, el costo de cambiar las teclas del juego es muy poco costoso desde el punto de vista del desarrollo, ya que simplemente basta con cambiar la tecla a la que se está asociando el GameAction en el método initInput().

Otro detalle a observar, es el hecho que dentro de este método hay dos diferentes entradas asociadas a la misma acción (moveUp), esto se hizo a modo de ejemplo para mostrar que es posible tener diferentes teclas asociadas a la misma acción sin mayores problemas. El detalle de la función maptoInterface(GameAction gameAction, int code) de la clase InputManager se muestra a continuación:

```
/// <summary>
/// metodo que sirve para hacer el mapeo entre la tecla y el
    GameAction
/// </summary>
public void maptoInterface(GameAction gameAction, int code)
  {
      interfaceActions.Add(code, gameAction);

  }
```

Luego, para determinar si una tecla fue presionada o no, dentro de la clase Game, existe un método el cual se llama checkInput(long elapsedTime) y lo que hace es verificar si alguna acción se cometió o no. Debido a lo extenso de la función solo se mostrará lo relevante dentro de este método y que tiene relación con lo que se está explicando:

```
/// <summary>
/// Chequea algun input realizado desde el dispositivo
/// </summary>
/// <param name="elapsedTime"></param>
private void checkInput(long elapsedTime)
  {
     if (moveLeft.isPressed()){//instrucciones… }
     if (moveRight.isPressed()){//instrucciones… }
     if (moveUp.isPressed()){//instrucciones… }
     if (moveDown.isPressed()){//instrucciones… }
     //sector de soltar botones
      if (moveUp.isReleased()){//instrucciones… }
      if (moveDown.isReleased()){//instrucciones… }
   }
```

Como se puede apreciar, para el caso de moveUp no se necesita más de una pregunta para determinar si la acción fue realizada o no. Esto ocurre debido a que el chequeo de las entradas se considera como una acción y no como el simple hecho de apretar una tecla durante el juego. Por último, el detalle restante de la clase InputManager se muestra a continuación:

```
/// <summary>
/// Control necesario para suscribir los eventos de entrada a
    la Clase
/// </summary>
private System.Windows.Forms.Control comp;
/// <summary>
/// Hashtable que hace el mapeo entre el GameAction y alguna
    entrada
/// </summary>
private Hashtable interfaceActions;
/// <summary>
/// Resetea todos los GameActions para que de la impresion que
    nunca han sido presionados
/// </summary>
public void resetAllGameActions();
/// <summary>
/// Eventos relacionados con detectar las acciones de entrada ///
</summary>
private GameAction getKeyAction(KeyEventArgs e);
private GameAction getMouseAction(MouseEventArgs e);
public void keyDown(object sender, KeyEventArgs e);
public void keyUp(object sender, KeyEventArgs e);
public void MouseDown(object sender, MouseEventArgs e);
public void MouseUp(object sender, MouseEventArgs e);
```

5.1.2. Despliegue de imágenes

Como se explicó en el capítulo anterior, el despliegue de imágenes consiste en mostrar las todas las figuras que componen el escenario y aquellas que, unidas dan la sensación que un objeto se está moviendo.

La clase que se encarga de hacer esta tarea se llama Renderer y es la que se preocupa de dibujar todos los objetos en la escena. Esta clase se compone principalmente de métodos que se encargan de hacer la conversión entre el mapa de tiles y los pixeles y los métodos que se preocupan de hacer el renderer de la escena.

Sobre los atributos que tiene esta clase, sus características están principalmente relacionadas con la información asociada con el tamaño de un tile y la conversión que se necesita para pasar de un tile a pixel.

Con respecto a los métodos encargados de hacer la conversión entre tiles y pixeles se usan debido a que las coordenadas del mapa se almacenan como tiles y no pixeles, por lo tanto es importante tener métodos que permitan cambiar rápidamente entre un sistema de coordenadas a otro sin mayores inconvenientes. Los métodos mencionados son los siguientes:

```
/// <summary>
/// Convierte desde pixel posicion a tile posición.
/// </summary>
public static int pixelsToTiles(float pixels);
public static int pixelsToTiles(int pixels);
/// <summary>
/// Convierte desde tile posicion a pixel posición.
/// </summary>
public static int tilesToPixels(int numTiles);
```

El método principal de la clase Renderer se llama draw(Graphics g, tilegame.TileMap map, int screenWidth, int screenHeight, List<game.Opponent> contrincantes, Sprites.Player player) y es llamado desde el gameLoop que se encuentra en la clase GameCore. Este método consiste básicamente en dibujar todo el escenario según donde se encuentre ubicado el jugador en ese momento. La idea es tener ubicado al jugador siempre en el centro de la pantalla del dispositivo y luego dibujar el resto de la escena en función de esta posición. Junto con esto, como se mencionó en el capítulo anterior, todo lo que se dibuja en el dispositivo primero debe pasar por un bitmap que se dibuja en un Graphics distinto del que se muestra en el dispositivo, y una vez que se dibujó todo aquello que se quería dibujar, se traspasa al bitmap que el usuario ve durante el juego. Resumiendo, los elementos que debe dibujar en la escena el juego son:

- Mapa
- Jugador
- Objetos mágicos
- Posición del jugador en la competencia
- Oponentes

Dicho esto, la función draw(Graphics g, tilegame.TileMap map, int screenWidth, int screenHeight, List<game.Opponent> contrincantes, Sprites.Player player) se muestra a continuación:

```
        /// <summary>
        /// Draws the specified TileMap.
        /// </summary>
        public void draw(Graphics g, tilegame.TileMap map, int screenWidth,
int screenHeight, List<game.Opponent> contrincantes, Sprites.Player player)
        {
            // dibujar background
            drawMenu(g, new Rectangle(0, 0, screenWidth, screenHeight));
            // instrucciones para detectar la posición del jugador
            int mapWidth = tilesToPixels(map.getWidth());
            int mapHeight = tilesToPixels(map.getHeight());
            int player_X = (int)Math.Round(player.X);
            int player_Y = (int)Math.Round(player.Y);

            int offsetX = screenWidth / 2 - player_X - TILE_SIZE;
            offsetX = Math.Min(offsetX, 0);
            offsetX = Math.Max(offsetX, screenWidth - mapWidth);

            int offsetY = screenHeight / 2 - player_Y - TILE_SIZE;
            offsetY = Math.Min(offsetY, 0);
            offsetY = Math.Max(offsetY, screenHeight - mapHeight);
            int firstTileX = pixelsToTiles(-offsetX);
            int lastTileX = firstTileX + pixelsToTiles(screenWidth) +
                            1;

            int firstTileY = pixelsToTiles(-offsetY);
            int lastTileY = firstTileY + pixelsToTiles(screenHeight) +
                            1;

            // dibujar el mapa
            for (int y = firstTileY; y < lastTileY; y++)
            {
                for (int x = firstTileX; x <= lastTileX; x++)
                {
                    Image image = map.getTile(x, y);
                    if (image != null)
                    {
                        int pos_x = tilesToPixels(x);
                        int pos_y = tilesToPixels(y);
                        g.DrawImage(image, pos_x + offsetX, pos_y +
                                    offsetY);
                    }
                }
            }

            //dibujar Player
            _drawPlayer(g, player, offsetX, offsetY);

            //dibujar objetos mágicos y la posición
            _drawPosition(g, player);
            _drawMagicObject(g, player);

            //dibujar oponentes
            _drawOtherPlayers(g, contrincantes, offsetX, offsetY);
        }
```

Por último, cabe señalar que este método sigue el algoritmo del pintor, por lo tanto, en caso que un oponente esté en la misma posición que el jugador, entonces se verá el oponente por sobre este.

5.1.2.1. Sprites

Para representar un sprite en el juego, se procedió a crear una clase abstracta desde la cual heredarán todos los sprites requeridos en el juego, ya sea un auto o algún objeto mágico. Esta clase se llama Sprite y tiene todo lo necesario para que cualquier sprite del juego pueda ser representado correctamente.

Básicamente lo principal que requiere un sprite dentro del juego es una posición, la velocidad y una animación. Los atributos que tiene esta clase son los siguientes:

```
// animación
protected graphics.Animation anim;
// posición (pixeles)
protected double x;
protected double y;
// velocidad del sprite
private float dx;
private float dy;
```

Los métodos que contiene la clase Sprite son 2 y son abstractos. El primero es para hacer un update, que cualquier sprite deberá realizar durante el gameLoop y el otro es para clonar, ya que también es una propiedad importante dentro de estos objetos.

Para tener la imagen que represente el sprite, se tuvo que realizar una clase aparte llamada Movement, esto se hizo de esta forma, debido a que como se mencionó en el capítulo anterior .NET compact framework no es capaz de realizar rotaciones, por lo que fue necesario realizar una clase que tuviera todas las imágenes almacenadas dentro de un arreglo y además realizara tareas con esta imagen. De esta forma se puede modularizar más la aplicación y simplifica el entendimiento del código.

La clase Movement tiene como atributo 3 variables; la primera es un arreglo con todas las imágenes necesarias para dar la sensación de movimiento para el sprite. La segunda indica cual es la imagen actual que se está utilizando, y la tercera es un índice que se utiliza para el caso en que se quiera clonar este objeto asignar la imagen actual que está mostrando el sprite en ese momento.

Los métodos que tiene esta clase cumplen diferentes funcionalidades; una es la de verificar la imagen que se debe mostrar durante el juego, la cual funciona con los métodos:

```
public void update(int index)
```

```
public Bitmap getImage()
public bool IsImage(int index)
```

El primer método lo que hace es realizar un update según el índice que reciba, el segundo retorna la imagen actual que tiene que se está realizando y el último método verifica si existe alguna imagen según el índice que fue entregado.

Los otros métodos creados tienen finalidades diferentes, el primero, sirve para clonar la clase[13], el segundo método, sirve para agregar las imágenes que posteriormente serán utilizadas en la clase. El encabezado de los métodos se muestra a continuación:

```
public Object clone();
public void addImage(Bitmap image, int index);
```

Para la técnica "Color Key Transparency", al momento de dibujar los sprites, se saca el color que tiene el primer pixel de la figura, el código que realiza esto es el siguiente:

```
m_mattr.SetColorKey(image.GetPixel(0, 0), image.GetPixel(0, 0));
```

m_mattr es una variable de tipo ImageAttributes, clase que contiene información acerca de un bitmap, como el color que es manipulado durante el rendering de la imagen.

5.1.2.1.1. Animación

Para la animación, lo que se hizo fue crear 2 clases, Animation y AnimFrame respectivamente. La primera, es la encargada de manejar los tiempos en los cuales las imágenes se van a mostrar, la cantidad de imágenes que debe tener, hacer los cambios de imágenes, etc.

La clase AnimFrame contiene simplemente una imagen con el tiempo asociado a esta. De esta forma, la clase Animation lo que hace es tener una lista con la clase AnimFrame con el fin de facilitar el manejo de las imágenes en la animación. El método encargado de realizar los cambios de las imágenes para dar la sensación de animación se llama update(long elapsedTime). Este método recibe el tiempo transcurrido y según este verifica que imagen mostrar o no. El detalle de este se muestra a continuación:

```
public void update(long elapsedTime)
  {
      if (frames.Count > 1)
      {
          animTime += elapsedTime;
```

[13] Esto es importante, ya que hay una clase que se encarga de administrar los recursos del juego llamada ResourceManager , esta clase se explica en la próxima sección de este capítulo

```
            if (animTime >= totalDuration)
            {
                animTime = animTime % totalDuration;
                currFrameIndex = 0;
            }

            while (animTime > getFrame(currFrameIndex).endTime)
            {
                currFrameIndex++;
            }
        }

    }
```

5.1.3. Administrador de Recursos

Para el administrador de recursos se creó una clase llamada ResourceManager la cual se preocupa de cargar todas las imágenes que posteriormente serán utilizadas en el juego. Para esto, tiene métodos encargados de cargar las imágenes y luego asignar estás imágenes a un objeto que posteriormente será pedido. Por su parte, estos objetos, tienen un método llamado clone(), el cual consiste en retornar un nuevo objeto con las imágenes asociados a este. Si tomamos como ejemplo los objetos que representan los autos, es creado el método loadCar(string car, string extension, int car_case), el cual se detalla a continuación:

```
public void loadCar(string car, string extension, int car_case)
  {
      graphics.Movement movement = new graphics.Movement();
      for (int i = 0; i < 90; i += 3)
      {
          string name = car + i + "." + extension;
          Image image = loadImage(name);
          //guardo la imagen
          if (image != null)
          {
              Bitmap images = new Bitmap(image);
              movement.addImage(images, i);
              //por cada foto la roto a los 3 ejes restantes
              Bitmap image_90 = _rotateImage(images, 90);
              movement.addImage(image_90, (270 + i));
              Bitmap image_180 = _rotateImage(images, 180);
              movement.addImage(image_180, (180 + i));
              Bitmap image_270 = _rotateImage(images, 270);
              movement.addImage(image_270, (90 + i));
          }
      }
      movement.update(270);
      switch (car_case)
      {
          case 0:
              grayPlayerSprite = new Sprites.Player();
              grayPlayerSprite.Movement = movement;
              break;
          case 1:
              bluePlayerSprite = new Sprites.Player();
              bluePlayerSprite.Movement = movement;
```

```
                break;
            case 2:
                greenPlayerSprite = new Sprites.Player();
                greenPlayerSprite.Movement = movement;
                break;
            case 3:
                redPlayerSprite = new Sprites.Player();
                redPlayerSprite.Movement = movement;
                break;
            case 4:
                orangePlayerSprite = new Sprites.Player();
                orangePlayerSprite.Movement = movement;
                break;
        }
    }
```

Como se puede ver, el método realiza lo que se explicó en el capítulo anterior sobre tener imágenes en el primer cuadrante y luego rotarlas en los 3 cuadrantes restantes. Luego de esto, se agrega la clase Movement al objeto del auto, según el tipo de auto que requiera ser creado.

Una vez que se crearon estos objetos, existe otro método que es el encargado de retornar el objeto pedido, pero clonado. El método que hace esto se llama getPlayer(util.Team car_case) y retorna un auto según el equipo al que pertenezca. A continuación se puede revisar el detalle del método:

```
public Sprites.Player getPlayer(util.Team car_case)
{
      object player= null;
      switch (car_case)
      {
          case util.Team.GRAY:
              player = grayPlayerSprite.clone();
              break;
          case util.Team.BLUE:
              player = bluePlayerSprite.clone();
              break;
          case util.Team.GREEN:
              player = greenPlayerSprite.clone();
              break;
          case util.Team.RED:
              player = redPlayerSprite.clone();
              break;
          case util.Team.ORANGE:
              player = orangePlayerSprite.clone();
              break;
      }
      return player as Sprites.Player;
  }
```

5.1.4. Mapas

Para crear los mapas en el juego, se realizó una clase llamada TileMap, la cual representa un mapa en el juego. Como se mencionó en el capítulo anterior, los mapas se representan a través de

tiles. Para almacenar los tiles, se creó una matriz de Bitmap llamado tiles. Junto con los tiles, también se crearon atributos que ayudan en la representación de un mapa. Ejemplo de esto, es el ancho y largo del mapa. Otros atributos que se crearon tienen relación con la dirección del mapa. Al igual que los tiles, existe una matriz donde se almacena la dirección permitida que puede ir un jugador durante la competencia. Por último, el mapa también tiene como atributo el número de vueltas y la cantidad de checkpoints. Los atributos se muestran a continuación:

```
private Bitmap[,] tiles;
private int[,] directions;
private int width, height;
private int _inicioX, _inicioY;
private int _laps;
```

Los métodos de la clase se pueden dividir en aquellos que sirven para agregar información relacionada con el mapa, desplegar la información y un método que sirve para posicionar un auto al comienzo de la carrera. Los encabezados de estos métodos se muestran a continuación:

```
public int getDirection(int x, int y)
public void setDirections(int x, int y, int value)
public Bitmap getTile(int x, int y)
public void setTile(int x, int y, Bitmap tile)
public void initPlayer(Sprites.Player player)
public bool isRoad(int x, int y)
```

5.1.4.1. Cargar un mapa

Como se mencionó en el capítulo anterior, los mapas se cargan leyendo un archivo de texto plano. La clase que lee estos archivos y carga el mapa es la misma que se encarga de cargar los sprites, esta es ResourceManager y el método que carga un mapa se llama loadMap(String filename). Este método se divide en dos partes, la primera parte se encarga de leer toda la información directamente desde el archivo de texto: A medida que va leyendo las líneas, las va agregando dentro de una lista, con esta información se obtiene inmediatamente el ancho, largo, número de vueltas y la cantidad de checkpoint en la etapa. El archivo también puede tener comentarios, las líneas que tienen comentarios son aquellas que comienzan con el carácter “#”.El carácter “@” indica información relacionada con el número de vueltas, checkpoint, etc. La primera parte del método se puede observar a continuación:

```
private TileMap loadMap(String filename)
{
      List<String> lines = new List<String>();
      List<String> directions = new List<String>();
      int width = 0;
      int height = 0;
      int checkpoints = 0;
      int laps = 0;

      // read every line in the text file into the list
      String prefijo = "HLMPFileSharing";
```

```
            String nameMap = prefijo + ".maps.tiles." + filename;
            Assembly asm = Assembly.GetExecutingAssembly();
            StreamReader sr = new
StreamReader(asm.GetManifestResourceStream(nameMap));
            while (true)
            {
                String line = sr.ReadLine();
                // no more lines to read
                if (line == null)
                {
                    sr.Close();
                    break;
                }
                if (line.Length > 0)
                {
                    String firstLetter = line.Substring(0, 1);
                    // add every line except for comments
                    if (firstLetter != "#" && firstLetter !="@")
                    {
                        lines.Add(line);
                        width = Math.Max(width, line.Length);
                    }
                    if (firstLetter == "@") {
                        char[] split = { '@', '|' };
                        string[] info = line.Split(split);
                        laps = Convert.ToInt32(info[1]);
                        checkpoints = Convert.ToInt32(info[2]);
                    }
                }

            }
```

La segunda parte del método se encarga de leer la lista donde se almacenó la información relevante para crear el mapa (esto es, evitar las líneas con comentarios y las que comienzan con "@") y luego guardar en la matriz del mapa los tiles respectivos. El detalle se muestra a continuación:

```
            // parsea las lineas para crear el TileEngine
            height = lines.Count;
            tilegame.TileMap newMap = new tilegame.TileMap(width, height);
            newMap.Laps = laps;
            newMap.CheckPoints = checkpoints;
            for (int y = 0; y < height; y++)
            {
                String line = (String)lines.ElementAt(y);
                for (int x = 0; x < line.Length; x++)
                {
                    char ch = line[x];

                    int tile = ch - 'A';
                    if (tile >= 0 && tile < tiles.Count)
                    {
                        newMap.setTile(x, y,
(Bitmap)tiles.ElementAt<Bitmap>(tile));
                        string dir = directions.ElementAt<string>(y);
                        string test = "" +dir[x];
                        int value = Convert.ToInt32(test);
```

```
                    newMap.setDirections(x, y, value);
                    if (ch == 'O') {
                        newMap.InicioY = y;
                        newMap.InicioX = x;
                    }
                }
            }
        }
        return newMap;
    }
```

5.2. Comunicación

Para la comunicación, se tuvo que crear varias clases para poder realizar las tareas necesarias para que los dispositivos se pudieran comunicar correctamente. Las primeras clases que se realizaron tienen que ver con la generación de mensajes y un protocolo que permita el envío de estos.

Estas clases son necesarias para el correcto uso de la librería HLMP, ya que si bien, esta librería asegura la correcta comunicación entre los dispositivos, es necesario especificar ciertos atributos, como por ejemplo, que tipo de información se está enviando, los mensajes son multicast o unicast, los envíos de los mensajes son TCP o UDP, etc. Los objetos que fueron creados para esto se pueden dividir según la tarea que deban realizar, estas son:

- Clase para el protocolo de comunicación
- Interfaz para eventos de mensajes para el juego
- Clases para representar los mensajes

La clase que implementa el protocolo se llama RaceGameProtocol y hereda de la clase SubProtocolI, esta clase tiene 4 métodos, los cuales sirven principalmente para definir los tipos de mensajes que recibirá este protocolo, el procesamiento de los mensajes, el procesamiento de los mensajes cuando no es posible enviarlos y el envío de un mensaje. Los encabezados de los métodos se muestran a continuación:

```
public MessageTypeList getMessageTypes()
public void proccessMessage(Message message)
public void errorMessage(Message message)
public void sendTextMessage(NetUser netUser, String text, int type)
```

La interfaz que es creada para manejar los eventos de mensajes en el juego se llama RaceGameHandlerI. Esta clase es implementada por la clase Brain, la cual será explicada luego en este capítulo. Volviendo a la interfaz, los encabezados que tiene son los necesarios para permitir el manejo de mensajes. Estos son:

```
void raceGameMessageReceived(NetUser netUser, String message)
void groupRaceGameMessageReceived(NetUser netUser, String message)
void raceGameWarninglInformation(String text)
void serverRaceGameMessageReceived(NetUser netUser, String message)
```

Para el envío de mensajes se crearon las clases GroupRaceGameMessage, RaceGameMessage y ServerRaceGameMessage respectivamente. La primera se realizó para que un jugador les envíe mensajes a todos los usuarios que están jugando en ese momento. Es por esto, que esta clase hereda de MulticastMessage, así, cuando un jugador quiera enviar un mensaje al resto, simplemente lo envía a través de esta clase.

La clase RaceGameMessage, se creó en caso que un jugador le quiera enviar información a otro respectivamente. En este caso, la clase hereda de SafeUnicastMessage, se asume en este caso que es importante el envío de esta información, razón por la cual se escogió este tipo de mensaje y no de tipo UDP.

Finalmente, la clase ServerRaceGameMessage, se creó para el envío de mensajes entre un jugador y el encargado de configurar el juego. Si bien, se pudo haber utilizado la misma clase anterior para realizar esta tarea, se optó por tener una clase distinta para darle mayor énfasis al hecho que el mensaje se envía al servidor y no a cualquier usuario. Esta clase hereda de UnicastMessage y se hizo así para no sobrecargar al dispositivo que está a cargo.

Para representar a los jugadores que están conectados, se creó la clase Opponent. Esta clase sirve tanto para todo lo que tenga relación con algún jugador conectado, ya sea, la información relacionada con el auto de carrera, los mensajes que recibe desde este jugador, etc. Para representar el auto de carrera, simplemente se agrega una clase de tipo Player llamada contrincante. De esta forma, al recibir los mensajes que contienen la información relacionada con el auto, simplemente se le agrega la información al atributo contrincante. Junto con estos atributos, esta clase también tiene un identificador que sirve para luego consultar si el usuario desde donde se recibe el mensaje pertenece al juego al que se está jugando o no. Este identificador es de tipo Guid, y lo trae el NetUser dentro de sus propiedades. De esta forma, siempre que reciba un mensaje desde algún usuario conectado a la MANET, se podrá descartar o no el mensaje recibido en la medida que el usuario esté jugando con el usuario que recibe el mensaje o no.

La clase que se preocupa de recibir y descartar los mensajes se llama Brain e implementa la interfaz RaceGameHandlerI, Para aceptar o descartar, lo que tiene esta clase es una lista de tipo Opponent, y a medida que un usuario entra al juego, se suscribe a esta lista. El criterio para decidir que jugadores pertenecen a un juego consiste en verificar a quien escogieron como el encargado de decidir la configuración del juego, esto es, decidir la cantidad de etapas a realizar, la dificultad, etc. Si el jugador que decide es el mismo, entonces pertenecen al mismo juego, caso contrario, no se incluye en la lista.

5.3. Formularios

Como se mencionó en el capítulo previo, los formularios se crearon para ser la cara visible de la aplicación, aunque también sirven para conectar el juego con la aplicación que lo requiera. La clase que se preocupa de la conexión entre el juego y cualquier objeto de tipo Control, se llama ManageGame. Esta clase no sólo se encarga de la conexión entre el juego y un XX, también se encarga de realizar las referencias respectivas entre la clase Brain y el juego. Esto se hace así debido a que el juego debe enviar información a los otros jugadores para que eventualmente estos la procesen.

El método que se encarga de la unión entre el juego y un formulario se llama connectControltoGame() y lo que hace es simplemente decirle al juego quien es el formulario. El juego necesita recibir esta información ya que los dibujos se deben hacer dentro de un objeto de tipo Graphics, y esta clase se encuentra dentro de un objeto de tipo Control.

Otra tarea que cumple la clase ManageGame es la de inicializar el juego. Para esto, se crea un Thread, el cual es inicializado dentro del método start() de la clase. La implementación se muestra a continuación:

```
      public void start() {

            game.GameBounds =new System.Drawing.Rectangle(0, 0,
       control.Size.Width, (int)(control.Size.Height * 1.05)) ;
            ThreadStart starter = new ThreadStart(makeThread);
            thread = new Thread(starter);
            thread.Start();
}
```

Por otro lado, el método makeThread es el llamado a inicializar el juego, cuando el Thread comienza a funcionar. El detalle de este método a continuación:

```
      private void makeThread() { game.run(); }
```

Junto con la tarea de iniciar el juego, esta clase también es la dedicada a terminarlo. El método que lo realiza se llama stop() y se puede apreciar a continuación:

```
      public void stop() {
            game.stop();
            Brain.cleanOpponents();
            if (thread != null) {
                thread.Join();
            }

         }
```

El formulario principal que se utiliza en el juego se llama MainForm. Aquí es donde se crean todos los tab necesarios para mostrar los diferentes formularios en la aplicación del juego. Junto con esto, esta clase también se encarga de inicializar los atributos relacionados con la librería

HLMP. Además, se suscribe a los eventos relacionados con esta librería, como por ejemplo, que clase recibe los mensajes, que evento se debe hacer al momento que el usuario se conecta a la MANET, que hacer en caso que un usuario se desconecte, etc.

Para la configuración del juego se creó otro formulario el cual se llama SetupHostGame y sirve para definir todos los datos para poder jugar, ya sea el usuario que está a cargo del juego, el equipo al que se unirán y los datos relacionados con las etapas, que sólo el usuario que esté a cargo del juego lo puede modificar.

Por último, el formulario en el cual el juego se muestra se llama RacingGameCompact, y sirve como se mencionó previamente, para obtener un Graphics y con eso poder desplegar los dibujos.

6. Evaluación Inicial de la Solución

Para la evaluación del juego desarrollado, se probó con niños que no están en las circunstancias en las cuales se espera que sea usado el juego. De hecho, el grupo socioeconómico al cual pertenecen tampoco es el mismo. Sin embargo, estas situaciones ponen al juego en una situación de exigencia mayor, ya que los niños en cuestión están acostumbrados a jugar con consolas de última generación, se aburren rápido cuando algo no les gusta y por último, no están enfermos en cama sin nada más que realizar, por lo que la posibilidad de aburrirse si el juego no les guste es mayor.

La cantidad de niños que se entrevistó fueron 5, donde las edades se muestran a continuación:

Sexo	Edad
Masculino	9
Masculino	14
Masculino	10
Femenino	13
Femenino	6

Tabla 9: Edad de los entrevistados

La evaluación estuvo orientada principalmente a 2 puntos, evaluación de la iconografía y una evaluación inicial del juego.

6.1. Evaluación de la iconografía

Para la evaluación de la iconografía, se consideró relevante determinar si la imagen que se observa se asocia a algo bueno o a algo malo, esto es así, debido a que las imágenes serán utilizadas cuando el auto obtenga un objeto mágico. Es por esta razón, que en cierto modo, es necesario saber la opinión de los niños con las imágenes que se usan en el juego.

Los resultados indican que al 80% de las figuras le dan un sentido correcto, esto es, asocian que la figura hace un daño o beneficia al auto durante la carrera. La imagen donde los niños más se equivocaron fue la imagen donde se ve la opción “prohibido”, ya que por lo general, no entendían que significaba la figura.

Figura 16: Imagen "Prohibido"

Esto tiene sentido, ya que por lo general es una imagen que no están acostumbrados a ver normalmente, por lo que les es difícil asociarlo con algo positivo o negativo. Para el resto de las preguntas, por lo menos más de la mitad respondió correctamente.

Para el caso en que figuras similares cambian en pequeños detalles, como por ejemplo, las estrellas o los diablos, se les pidió a los niños que los ordenaran según fuese el caso, ya sea del más bueno al menos bueno, o del más malo al menos malo.

Para este escenario no se indica si los resultados son correctos o no, ya que si bien, en una etapa inicial, las imágenes tenían una tarea asignada en el juego por ser un objeto mágico, los valores obtenidos no concuerdan con lo esperado, por lo tanto, se puede desprender con esto, que no estaban bien asignados los órdenes. Debido a esto, lo que se hizo fue cambiar las imágenes a lo que la mayoría de los niños quiso.

6.2. Evaluación del juego

Para evaluar el juego, lo que se hizo fue mostrar el juego instalado en un dispositivo móvil para ver como los niños reaccionaban ante él.

Figura 17: Evaluación del juego

Lamentablemente los niños fueron entrevistados en situaciones distintas, por lo que no se pudo probar el juego entre varios niños jugando al mismo tiempo. Sin embargo, al menos se probó con 2 personas jugando al mismo tiempo. La entrevista en esta parte, fue más orientada a obtener resultados cualitativos más que cuantitativos.

Las opiniones de los niños por lo general fueron satisfactorias, no les importó mayormente que el dispositivo fuese antiguo, ni menos que la calidad gráfica de este no estuviese a lo que ellos están acostumbrados a ver. Lo que rescataron de la experiencia fue que el hecho de jugar el juego entre amigos al mismo tiempo les parecía una idea bastante entretenida, sobre todo si se les preguntaba en el caso que estuviesen enfermos en una cama acostados. Uno de los niños entrevistados comentó que para él "jugar entre hartos es más entretenido que jugar sólo". Sobre el formato de competencia, también lo vieron como algo positivo, ya que encontraban que el hecho que la competencia fuera en grupos lo hacía más entretenido.

Un tema a tener en consideración es las teclas que utilizan para poder jugar, ya que si el dispositivo era muy viejo y tenía como botones una "palanca", les costaba mucho doblar, por lo que el juego se les hacía muy difícil.

Figura 18: Ejemplo de PDA con palanca

Otro aspecto a considerar, es que el juego fue probado por períodos cortos de tiempo, por lo que no se podría concluir que el juego es entretenido para un niño si juega por horas con él.

Se les preguntó si era importante que el juego tuviese sonido o no, y si bien ninguno se había dado cuenta de esta situación, la respuesta de casi todos fue que no era algo que determinara que el juego fuese entretenido, pese a que les gustaría que lo tuviese.

Sobre la velocidad de los autos para avanzar les pareció adecuada, excepto en el caso en que el dispositivo tenía la "palanca", ya que les dificultaba mucho el doblar. Otro comentario que llamó la atención fue que algunos sugirieron que el camino tuviese flechas para saber cuál es la dirección correcta que debe seguir el auto.

Por último, hicieron algunas sugerencias que podría tener el juego, como por ejemplo, que a medida que van jugando etapas, vayan apareciendo más autos y mejores que los originales. Esto se ve un poco complejo de hacer en la práctica, debido a que los dispositivos que se utilizan para poder jugar son demasiado limitados.

7. Conclusiones y Trabajo a Futuro

En este trabajo de memoria, se realizó un juego colaborativo móvil para niños hospitalizados en hospitales públicos para niños entre 4 y 12 años de edad. Este consiste en un juego de carreras en el cual los usuarios juegan dentro de un equipo, donde el ganador se decide según el equipo que obtuvo un mayor puntaje durante la competencia. De esta forma, en el juego no existe un solo ganador, sino un grupo perteneciente a un mismo equipo. Así, se espera lograr una competencia y un trabajo colaborativo al mismo tiempo. La decisión que el juego fuera de carrera, se debe a que la entretención en este tipo de juegos pasa por la competencia y no por el escenario, por lo que eventualmente un niño puede pasar mucho más tiempo jugando sin aburrirse.

El juego fue probado con 5 niños de distintas edades con la finalidad de comprender si el mismo cumplía con el objetivo mínimo requerido de un juego: que sea entretenido. Lamentablemente, el juego no pudo ser probado con los niños al cual va dirigido el juego debido a la clara dificultad de ir a un hospital y hacer preguntas a niños desconocidos. Sin embargo, los niños entrevistados respondieron todas las preguntas de con gran disposición.

Es importante destacar que los niños entrevistados pertenecen a un grupo socioeconómico distinto al cual va orientado el juego. Pese a esto, la opinión general del juego fue positiva, los niños quedaron contentos y animados para jugar más tiempo con este. Por otro lado, es importante señalar, que el tiempo en que los niños probaron el juego fue corto, por lo que no se puede concluir que el juego sea entretenido para tiempos largos. Otro aspecto importante que se puede concluir, es que a los niños no les importa mucho la gráfica de un juego, en la medida que este sea entretenido para ellos y represente un reto.

Sobre los objetivos de la memoria, se puede decir que se cumplieron a cabalidad. El juego que se creó funciona bajo dispositivos móviles, la infraestructura del hospital jamás se utiliza, exceptuando el caso en que sea necesario enchufar estos dispositivos debido a que su batería esté agotada producto del tiempo que se jugó. Para la conexión del juego con una red social, está hecho y es relativamente sencillo de implementar para la persona que realice esta red.

Sobre la dificultad de realizar un juego, se puede decir que es una tarea bastante compleja, sobre todo si se considera que el trabajo fue realizado por una sola persona. Los conocimientos que se requieren para desarrollar un juego de manera exitosa es alta, si a eso se considera el tiempo que se requiere para desarrollarlo la dificultad es mayor. A esto, hay que agregar que para los dispositivos en que el juego va a ser utilizado, no existen herramientas que faciliten esta tarea, ya que al ser muy antiguos, las posibilidades para desarrollar un juego es bastante limitada.

Por otro lado, se puede decir, que las herramientas entregadas por la Universidad a lo largo de la carrera, permitieron enfrentar este desafío de la manera adecuada, ya sea desde el punto de vista técnico como de la gestión para poder implementar el juego.

Sobre los trabajos a futuro sobre esta memoria, se espera que la red social sea implementada para que el juego pueda ser agregado y así dar mayores distracciones a los niños, ya que las condiciones en que están día a día no son las mejores.

Sobre el juego, se espera que alguien pueda tomar parte de este trabajo y poder seguir desarrollando otro tipo de juegos para estos niños, ya que en este caso, las posibilidades de realizar un juego más elaborado aumenta, puesto que ya existe una base bastante importante realizada, lo que a la larga disminuiría los tiempos de desarrollo. También se espera que el juego pueda tener otras características, tales como un editor de escenarios, o que tenga sonido.

8. Bibliografía y Referencias

[Blow, 2004] Blow, J. Game development harder than you think. Queue, Vol. 1, Issue 10, pp. 28 – 37. 2004.

[Carro, 2002] Carro, R.M., Breda, A.M., Castillo, G., Bajuelos, A.L.: Generación de Juegos Educativos Adaptativos. En: Actas del III Congreso Internacional de Interacción Persona-Ordenador, Eds. Aedo, I., Cuevas, P., Fernández, C., pp. 164-171. 2002.

[Herskovic, 2011] Herskovic, V., Ochoa, S. F., Pino, J., Neyem, A. "Mobile Collaborative Systems: Behind the User Interface". Accepted in the Journal of Universal Computer Science. To appear in 2011.

[Magia, 2010] Magia por una Sonrisa. URL: http://www.magiaxunasonrisa.com/. Última visita: Abril 2010.

[Martínez, 2007] Martínez, M., Martín, E. Un Modelo para el Diseño de Juegos Adaptativos y Colaborativos. En: Velázquez, A., Paredes, M. (eds.) Tecnologías del Software. Seminario de Investigación e Innovación en Tecnologías del Software (ISBN 978-84-9849-050-3), pp.167-180. Editorial Dikinson, S.L. 2007.

[ONet, 2010] Online.NET. C# Language Specification. URL: http://en.csharp-online.net/CSharp_Language_Specification , Última visita: Abril 2010.

[Rodriguez-Covili, 2010a] Rodriguez-Covili, J., Ochoa, S.F., Pino, J.A. Enhancing Mobile Collaboration with HLMP. 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD'2010), IEEE Press, Los Alamitos, CA. Shanghai, China, 7-9 April 2010, pp. 467-472.

[Rodriguez-Covili, 2010b] Rodriguez-Covili, J., Ochoa, S.F., Pino, J.A., Messeguer, R., Medina, E., Royo, D. HLMP API: A Software Library to Support the Development of Mobile Collaborative Applications, 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD'2010), IEEE Press, Los Alamitos, CA. Shanghai, China, 7-9 April 2010,pp. 479-484.

[Sanneblad, 2003] Sanneblad, J., Holmquist, L. R. Designing Collaborative Games on Handheld Computers. En: International Conference on Computer Graphics and Interactive Techniques. SIGGRAPH 2003.

[Sharp, 2007] Sharp, J. Microsoft Visual C# 2005: Step by Step. Microsoft Press, 2007.

Anexo 1: Detalle juego Lemmings

Lemmings es un videojuego publicado en el año 1991 y que tuvo versiones para diferentes computadoras y videoconsolas. En el momento de su publicación fue uno de los juegos de más éxito y logró puntuaciones máximas en las revistas especializadas.

Los personajes del juego están basados en la creencia popular de que los lemmings se suicidan en masa en situaciones de peligro. El objetivo del juego es el de salvar a un número determinado de lemmings en cada nivel, para lo que se cuenta con ocho habilidades distintas que se pueden repartir en número limitado a cada lemming para lograr alcanzar el final de cada fase.

El juego está dividido en un número de niveles. Cada nivel puede tener una o más salidas y una o más entradas, y consiste en controlar a unas unidades (los lemmings) conduciéndolas a través de diversos obstáculos (barrancos, paredes, montañas, etc.) y con el objetivo de llegar a una posición final. Estas unidades están capacitadas para realizar una serie de acciones, entre las que se cuentan la de construir escaleras, bloquear el paso a otros lemmings, lanzarse en paracaídas, etc. También tienen la opción de suicidarse (se supone que es para el caso de que ya sea imposible terminar la partida, por haber perdido demasiados lemmings). En caso de que a un lemming no se le asigne alguna habilidad, caminará ignorando a cualquier otro lemming (salvo a los bloqueadores) y cayendo fuera del mapa o cambiando de dirección cuando algún obstáculo le impida el paso. Los lemming pueden morir al caer de una gran altura, caer en agua, lava o fuera del mapa o ser capturados por una trampa. También morirán cuando se les asigne la habilidad de convertirse en una bomba.

Figura 19: Lemmings

Para completar con éxito cada nivel es necesario salvar a un número determinado de lemmings. Para ello el jugador dispone un número limitado de habilidades a utilizar. Hay ocho habilidades que pueden ser utilizadas. Los "escaladores" pueden subir por paredes verticales, los "paracaidistas" sobreviven a caídas desde gran altura. Existen tres habilidades que permiten al lemming crear un túnel de forma lateral, vertical o diagonal, pero siempre en terrenos que lo permitan. Los “constructores” crean escaleras de 12 escalones por lo que cuando finalizan las mismas, es necesario volver a darles la habilidad si todavía no han llegado a su objetivo. Los “bloqueadores” impiden el paso del resto de los lemmings. Por último, existe la posibilidad de asignar la habilidad de convertirse en bomba, de forma que pueda destruirse alguna zona.

Anexo 2: Diseño de archivo XML para obtener el contenido de las preguntas

El XML que se realizó para obtener el contenido de las preguntas tiene la siguiente estructura:

```
<preguntas>
        <pregunta>
                <detalle>[enunciado de la pregunta] </detalle>
                <alternativa1>[enunciado alternativa 1] </alternativa1>
                <alternativa2>[enunciado alternativa 1] </alternativa2>
                <alternativa3>[enunciado alternativa 1] </alternativa3>
                <alternativa4>[enunciado alternativa 1] </alternativa4>
                <respuesta>[alternativa correcta]</respuesta>
                <valor>[puntaje por respuesta correcta]</valor>
        </pregunta>
</preguntas>
```

Anexo 3: Archivo de texto para cargar el mapa

A continuación se presenta el archivo de texto para cargar el mapa.

```
# @[vueltas]|[checkpoints]|[tipo de pantalla]
@1|6|N
  CAAD                                                                                CAAD
  CAAD                                                                                CAAD
  CAAHBBBBBBBBBBBBBTBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBTBBBBBBBBBBBBBBBBBBBBBBBBGAAD
  CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD
  CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD
  CAALJJJJJJJJJJJJJUJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJUJJJJJJJJJJJJJJJJJJJJJJJJKAAD
  CAAD                                                                                CAAD
  RAAS                                                                                YAAX
  CAAD                                                                                CAAD
  CAAD                                                                                CAAD
  CAAD                                                                                CAAD
  CAAD                                                                                CAAD
  QOOP                                                                                YAAX
  CAAD                                                                                CAAD
  CAAD                                                                                CAAD
  CAAHBBBBBBBBBBBBBVBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBVBBBBBBBBBBBBBBBBBBBBBBBBGAAD
  CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD
  CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD
  CAALJJJJJJJJJJJJJWJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJWJJJJJJJJJJJJJJJJJJJJJJJJKAAD
  CAAD                                                                                CAAD
  CAAD                                                                                CAAD
```

Figura 20: Ejemplo de archivo de texto para el mapa

Anexo 4: Resultados Entrevistas

Las tablas que se muestran a continuación indican las respuestas entregadas por los niños al preguntarles si la imagen representaba algo bueno o malo para el auto durante la carrera. La imagen que tiene un color diferente, indica la que tuvo peor resultado durante las entrevistas.

Tabla 10: Resumen respuestas por imagen, parte 1

N°	Imagen	Correcta
1		4
2		4
3		4
4		2
5		5
6		5
7		5

Tabla 11: Resumen respuestas por imagen, parte 2

N°	Imagen	Correcta
8		3
9		5
10		5
11		5
12		3
13		5
14		5
15		3

Tabla 12: Resumen respuestas por imagen, parte 3

N°	Imagen	Correcta
16		3
17		3
18		3
19		5
20		5
21		5
22		3
23		3
24		3

Las siguientes tablas muestran los resultados obtenidos al preguntarles el orden que deberían tener las imágenes para los objetos mágicos, esto es, ordenar desde el más bueno al menos bueno para los objetos positivos y ordenar desde el más malo al menos malo para los negativos. El cuadro de color indica la mayoría escogida por los niños.

Tabla 13: Jerarquía de imágenes, parte 1

N°	Imagen	Básica	Normal	Plus
5		0	4	1
6		0	1	4
7		5	0	0
9		2	0	3
10		1	3	1
11		2	2	1
16		3	2	0

Tabla 14: Jerarquía de imágenes, parte

N°	Imagen	Básica	Normal	Plus
17		2	2	1
18		0	1	4
19		4	1	0
20		1	4	0
21		0	0	5
22		1	1	3
23		2	2	1
24		2	2	1