



FACULTAD DE INGENIERÍA

**Desarrollo de un videojuego con micro-transacciones
para iOS y Android
Trutruka Game Studio**

**Proyecto Aplicado de Titulación para Optar al Título de
Ingeniero en Computación en Informática**

Alumno: Vicente Andrés Conejeros de la Cruz

**Profesor Guía: Cristian Barría Huidobro
Ingeniero Informático**

**Santiago de Chile
Junio 2012**

RESUMEN

El siguiente proyecto cuyo título es "Desarrollo de un videojuego con micro-transacciones para iOS y Android" nace a partir de la necesidad de la empresa Trutruka Game Studio de crear un nuevo juego para plataformas iOS y Android. El proyecto busca que la empresa compita en el mercado de las micro-transacciones de los videojuegos de dispositivos móviles. Para lograr este cometido se definieron tres fases. Fase 1 de "Diseño del videojuego", donde mediante una toma de requerimientos y reuniones de diseño se elaboro un documento con el diseño del videojuego. Fase 2 de "Desarrollo del proyecto" en la cual se creó tanto el contenido grafico de la aplicación como su código fuente. Fase 3 de "Cierre del proyecto" donde se implementaron los instaladores del videojuego, para luego ser publicados en Google Play y App Store. Se concluye con lo importante que fue para el desarrollo del videojuego haber realizado las distintas etapas del diseño y de como la flexibilidad del desarrollo de los diversos módulos ayudo al rápido desarrollo del proyecto. Si bien el cierre del proyecto no es una etapa que se haya alcanzado, los pasos a seguir en esa etapa se encuentran claramente definidos. Del proyecto se obtuvo como resultado un videojuego para plataformas iOS y Android pronto a su completitud, se espera que a fines del mes de Julio del 2012 el proyecto se encuentre finalmente terminado.

INDICE GENERAL

	Pagina
INTRODUCCION	1
CAPITULO I PLANTEAMIENTO DEL PROBLEMA	5
1.1 Definición del problema.....	5
1.2 Preguntas de investigación.....	5
1.3 Objetivo general.....	6
1.4 Objetivos específicos.....	6
1.5 Justificación.....	6
1.6 Alcances.....	6
CAPITULO II MARCO TEORICO	7
2.1 Definiciones Conceptuales.....	7
2.2 Fundamentos teóricos.....	8
CAPITULO III ELABORACION DEL DISEÑO DEL VIDEOJUEGO	14
3.1 Recopilación de requerimientos de la gerencia.....	14
3.2 Recopilación de requerimientos de los clientes.....	14
3.3 Reunión de definición la jugabilidad	14
3.4 Reunión de definición del estilo grafico.....	16
3.5 Reunión de definición de las características del héroe.....	18
3.6 Reunión de definición de los enemigos.....	19
3.7 Reunión de definición de la historia.....	20
3.8 Reunión de definición de los ítems.....	20
3.9 Creación del documento de diseño del videojuego.....	22
3.10 Definición del material artístico del videojuego.....	22
3.11 Definición de los módulos y clases del videojuego.....	23
CAPITULO IV DESARROLLO DEL PROYECTO	24
4.1 Creación de los escenarios.....	24
4.2 Creación de los fotogramas del héroe.....	26
4.3 Creación de los fotogramas de los enemigos.....	27
4.4 Creación de las interfaces del inventario.....	29

4.5 Creación de la clase Game.....	30
4.6 Creación de la clase DataGame.....	30
4.7 Creación de la clase Character.....	30
4.8 Creación de la clase Hero.....	30
4.9 Creación de la clase BasicEnemy.....	31
4.10 Creación de la clase Item e Inventory.....	31
4.11 Creación de la clase Attack.....	32
4.12 Creación del guion.....	32
4.13 Creación de los niveles.....	32
4.14 Creación de los enemigos.....	33
4.15 Coordinación con el proveedor de sonidos.....	34
4.16 Integración de los sonidos al videojuego.....	34
4.17 Integración de las micro-transacciones.....	34
4.18 Integración de TapJoy.....	35
CAPITULO V CIERRE DEL PROYECTO.....	36
5.1 Asignación de recompensas con TapJoy.....	36
5.2 Registro del producto en Google Play.....	36
5.3 Registro de los paquetes de gemas en Google Play.....	37
5.4 Registro de producto en App Store.....	38
5.5 Registro de los paquetes de gemas en App Store.....	38
CAPITULO VI RESULTADOS OBTENIDOS.....	40
CONCLUSIONES.....	41
BIBLIOGRAFIA.....	42
ANEXOS.....	44
Anexo 1: Estados del videojuego.....	44
Anexo 2: Funcion Update de la clase Game.....	45
Anexo 3: Funcion InGameUpdate de la clase Game.....	46
Anexo 4: Funcionalidad de orientación de la clase Character.....	47
Anexo 5: Funcion tryToMove de la clase Character.....	48
Anexo 6: Funcion InGameUpdate de la clase Hero.....	49
Anexo 7: Funcion InGameUpdate de la clase BasicEnemy.....	51

Anexo 8: Tipos de ítems del videojuego.....	52
Anexo 9: Contenedor de ítems del videojuego.....	53
Anexo 10: Funciones de habilitación de ataques.....	54

INDICE DE IMAGENES

	Pagina
Imagen 1: Ilustraciones de las ciudades del videojuego.....	24
Imagen 2: Ilustraciones de los campos de batalla del videojuego.....	25
Imagen 3: Unión de los fotogramas del héroe.....	26
Imagen 4: Unión de los fotogramas de los enemigos.....	27
Imagen 5: Las cuatro lengüetas del inventario.....	29
Imagen 6: Herramienta de demarcación del área de juego de los niveles...	32
Imagen 7: Algunos objetos animados de los niveles.....	33
Imagen 8: Enemigos creados en el videojuego.....	33
Imagen 9: Menú principal del videojuego.....	40

INDICE DE TABLAS

	Pagina
Tabla 1: Jugabilidad del videojuego.....	14
Tabla 2: Estilo grafico del videojuego.....	16
Tabla 3: Línea de tiempo del videojuego.....	20
Tabla 4: Ítems del videojuego.....	20

INTRODUCCION

Desde su fundación Trutruka Game Studio (o simplemente Trutruka) tiene como misión el desarrollo de videojuegos para plataformas web y móviles. Durante el 2011 su principal objetivo fue el darse a conocer a la industria a través del desarrollo de videojuegos para páginas web. Para el 2012 se decidió comenzar a desarrollar videojuegos para dispositivos móviles. Este proceso se inicia con el desarrollo de un videojuego para celulares BlackBerry, con lo cual Mouse Hunter 3000 fue lanzado en Enero del 2012.

Si bien los resultados de Mouse Hunter 3000 no fueron buenos, el tener un videojuego móvil desarrollado constituye un hito importante para la empresa. Puesto que, al crear videojuegos con micro-transacciones, está intentando cambiar la forma en que se perciben ingresos. En vez de orientar el desarrollo a la venta de productos a otras empresas. Trutruka intentara vender contenido de los videojuego de manera directa a los usuarios de AppStore y GooglePlay. De esta manera se busca cambiar la forma en que funciona el negocio en la actualidad pasando a percibir ingresos proporcionales al desempeño de los videojuegos en las tiendas virtuales, las cuales han podido alcanzar cifras de millones de dólares en algunos casos emblemáticos.

Por eso, la empresa con este videojuego busca una nueva fuente de ingresos, y a su vez una forma de consolidar relaciones comerciales con TapJoy, empresa norteamericana que financiara este proyecto.

Planteamiento del problema

Si bien el desarrollo de videojuegos para dispositivos móviles es parte de la misión de la Empresa, hasta principios del año 2012 tan solo había sido desarrollado uno solo para dispositivos BlackBerry (Mouse Hunter 3000). Los resultados de este videojuego fueron decepcionantes debido a que RIM, compañía tras la marca BlackBerry, nunca ha demostrado interés en distribuir ni promocionar el software desarrollado por terceros, situación que en enero del 2012 sumió a la empresa en una crisis económica y de confianza con sus consumidores.

Por lo tanto se decidió continuar con la creación de videojuegos para dispositivos móviles pero principalmente a plataformas iOS y Android, el primero por los más de doscientos millones de usuarios que posee (Lara, 2011) y el segundo por su tasa de crecimiento de cerca de diez mil millones de descargas hasta diciembre del 2011 (Chu, 2011). La posibilidad de llegar a esta cantidad de usuarios hace que participar de este mercado sea algo atractivo para una empresa que desarrolla videojuegos para dispositivos móviles.

Con esos números en mente en el año 2012 la empresa comenzó a ofrecer el servicio de desarrollo de videojuegos para plataformas móviles. Pero aún el nulo portafolio en este tipo de juegos causa reparos en las empresas del sector más conservadoras. De todas formas se logro firmar el desarrollo de tres videojuegos con TapJoy, la cual busca promocionar su sistema de anuncios en los dispositivos Android. Los proyectos deben ser desarrollados en un periodo máximo de 6 meses, dando especial énfasis a la integración de micro-transacciones.

La decisión de desarrollar videojuegos con este tipo de características se tomo en base a que Trutruka no es una empresa que posea un gran capital, por lo tanto se busca con estos productos nuevas alternativas de ingreso. Esta alternativa, la de ingresos mediante micro-transacciones debe ser desarrollada principalmente mediante la fórmula de vender contenidos de los videojuegos que no pueda ser adquirido por otras vías. A cambio a los usuarios se les bonificara y se les dará la posibilidad de diferenciarse dentro del videojuegos.

Propósito

A principios del 2012, Trutruka llego a un acuerdo con TapJoy de desarrollar tres videojuegos para Android e iOS durante el primer semestre del 2012. Este acuerdo implica utilizar la plataforma de anuncios de TapJoy, la cual busca premiar a los jugadores de acuerdo a la cantidad productos publicitarios que sean capaces de consumir.

El desarrollo del proyecto busca crear un producto que pueda incluir el sistema de anuncios de TapJoy y estar dentro del género de los juegos RPG (Role Playing Game). Este género fue elegido debido a que permite ofrecer ítems que puedan ser comprados usando las tiendas virtuales AppStore y Google Play; o ser obtenidos como recompensas por ver anuncios de TapJoy.

Al integrar este tipo de características a los videojuegos de la compañía Trutruka busca generar una nueva fuente de ingresos a la vez que consolida su relación comercial con TapJoy.

Metodología Empleada

La naturaleza de los productos que se desarrollan dentro de la industria de los videojuegos hacen que en su desarrollo participe un equipo multidisciplinario los cuales si bien deben utilizar metodologías de trabajo a veces las mismas metodologías no se adecuan a las necesidades de sus usuarios. Algunas empresas planifican el desarrollo de sus proyectos de acuerdo a las necesidades de los mismos, Trutruka no es la excepción ya que este proyecto fue dividido en tres fases.

Primera fase de elaboración de requerimientos y diseño, en la que se tomaran los requerimientos del videojuego que provengan tanto de TapJoy como de la gerencia de Trutruka. Diseñándose los distintos componentes que satisfagan las necesidades del proyecto. También habrán reuniones entre los equipos de arte, diseño e ingeniería en las cuales se buscara acordar el estilo grafico, las proporciones visuales, sus personajes y enemigos, la historia, se propondrán los escenarios en los cuales transcurre el videojuego, como también se definirá su jugabilidad.

Segunda fase de desarrollo del proyecto, en esta etapa el equipo de Arte trabajara en crear los diseños de los personajes, enemigos, escenarios e interfaces graficas del videojuego. Por otra parte el equipo de Ingeniería se dedicara a darle vida a este material grafico creando sistemas de puntuación, ataques, animaciones, navegación, compra y venta de ítems. Se integrara el sistema de recompensas de TapJoy como el de las Tiendas virtuales GooglePlay y AppStore.

Tercera fase de cierre del proyecto en la que el equipo de Ingeniería se dedicara de manera exclusiva a la integración del videojuego con Google Play y App Store. Se firmaran los certificados que acreditan al videojuego como parte del material disponible en las dos tiendas virtuales y se subirán versiones de prueba las cuales podrán ser probadas en distintos modelos de celulares y tablets para su posterior publicación.

Cada una de las fases del proyecto compondrán un capítulo de esta memoria.

En el primer capítulo Planteamiento del problema, se definirá el problema, se plantean los objetivos generales, los objetivos específicos, la justificación y el alcance del proyecto.

En el segundo capítulo Marco teórico, es desarrollado un glosario de términos comunes dentro del desarrollo de videojuegos y se expone el marco teórico del cual se infieren las decisiones de: desarrollar un videojuego del tipo RPG, utilizar Unity y C# como herramientas de desarrollo.

En el tercer capítulo Elaboración del diseño del videojuego en el cual se definirán los métodos con los cuales se levantarán los requerimientos del proyecto tanto de parte de la gerencia como de TapJoy. Se plantea la manera en que se diseño el videojuego en términos de funcionalidad como artísticos, diseño que con posterioridad debe ser usado como una guía por los distintos miembros del equipo.

En el cuarto capítulo Desarrollo del proyecto son expuestos los distintos elementos de los que se compone la aplicación. También son detallado los componentes

programados por el equipo de Ingeniería y de los distintos elementos creados por el equipo de Arte, de cómo estos elementos son finalmente integrados entre sí. Se profundiza sobre el funcionamiento del videojuego en iOS y Android. Adicionalmente es aclarada la integración del sistema de anuncios de TapJoy dentro de la aplicación y cómo funciona el sistema de ventas en las tiendas virtuales de App Store y Google Play.

En el quinto capítulo Cierre del proyecto están explicadas las distintas etapas por las que debe pasar el videojuego para ser publicado tanto en Google Play como en App Store.

En el sexto capítulo Análisis de resultados son presentados los resultados del desarrollo del videojuego, entregándose cifras de los tiempos que tomo su desarrollo.

Para finalizar se pueden encontrar las conclusiones obtenidas del proyecto, las fuentes bibliográficas que fueron utilizadas y un apartado de anexo en el cual se incluyen trozos de códigos utilizados en él.

CAPITULO I PLANTEAMIENTO DEL PROBLEMA

1.1 Definición del problema

Debido a la poca rentabilidad del desarrollo de videojuegos web, se ha tomado la decisión de cambiar el modelo de negocios. Pasando de vender un videojuego totalmente acabado cuyo desarrollo es auspiciado por una empresa externa a cambio de poner un logo, hacia el desarrollo de videojuegos que utilicen micro-transacciones.

Por lo tanto Trutruka mediante un acuerdo con la empresa norteamericana TapJoy tiene la posibilidad de desarrollar tres videojuegos que incluyan micro-transacciones con los cuales generar ingresos, concretamente este proyecto es sobre el desarrollo de uno de esos videojuegos.

Antes de que se comenzara el desarrollo de este proyecto ya se había tomado la decisión de que su género fuese RPG, pero temas tan importantes como el diseño de la mecánica del videojuego, su estilo grafico, la historia y la manera de integrar tanto la plataforma de TapJoy como el sistema de micro-transacciones quedo para una definición posterior.

Por lo tanto mediante un proceso que une el levantamiento de requerimientos con el diseño se busca delinear los distintos elementos que componen un videojuego y que deberán solucionar distintos temas tanto del área artística como de diseño e ingeniería.

1.2 Preguntas de investigación

Como pregunta de investigación general se presenta

- ¿Cómo desarrollar un videojuego para iOS y Android que integre micro-transacciones?

De esta pregunta general se derivan las siguientes preguntas especificas.

- ¿De qué manera se diseña un videojuego?
- ¿De qué forma desarrollar los distintos componentes que surgen a partir del diseño del videojuego?
- ¿Qué hacer para que el videojuego esté disponible dentro del catalogo de las tiendas App Store y Google Play?

1.3 Objetivo general

Crear un videojuego del genero RPG (Role Playing Game) para plataformas iOS y Android que integre micro-transacciones.

1.4 Objetivos específicos

- Elaborar el diseño para un videojuego RPG.
- Desarrollar un videojuego RPG con micro-transacciones.
- Integrar el videojuego en el catalogo de las tiendas digitales de App Store y Google Play

1.5 Justificación

Se busca cambiar el actual modelo de negocios pasando del desarrollo de videojuegos como productos que pueden ser utilizados por otras empresas para recibir visitas o mover usuarios a recibir ingresos por el rendimiento comercial de los videojuegos en plataformas móviles de manera directa utilizando las tiendas virtuales.

De esta manera se busco el patrocinio de TapJoy el cual promociona su sistema de recompensas dentro de los videojuegos. Con esta empresa se firmo un acuerdo comercial por el desarrollo de tres videojuegos que utilicen su sistema de recompensas en dispositivos móviles. Por lo tanto, se busca la posibilidad de fortalecer las relaciones comerciales con esta empresa, ofreciendo a los jugadores la oportunidad de ver anuncios publicitarios y recibir los premios ofrecidos por su plataforma.

También se busca, establecer presencia en el mercado de las aplicaciones para iOS y Android. Por lo tanto, se intentara presentar este proyecto como uno de los productos estrellas de la empresa desarrollado durante el año 2012.

1.6 Alcances

- Se desarrollo el videojuego según el diseño y los acuerdos de los distintos departamentos de la empresa.
- Se implemento la venta de la moneda virtual mediante las tiendas virtuales App Store y Google Play.
- Se incluye el sistema de recompensas de TapJoy.
- Se integro el videojuego en las tiendas virtuales App Store y Google Play.

CAPITULO II MARCO TEORICO

Para abordar un tema como el desarrollo de videojuegos se hace necesario elaborar un listado de definiciones conceptuales, de manera que un lenguaje común pueda ser utilizado durante su desarrollo. En el marco del proyecto se ha desarrollado el siguiente listado.

2.1 Definiciones Conceptuales

Videojuego: Software creado para el entretenimiento, el cual se basa en la interacción entre una o más personas por medio de un controlador y aparato que ejecute el videojuego. (Baer, Rusch, & Harrison, 1969)

Videojuego de rol: Videojuegos cuya forma de jugar han sido adaptadas de los juegos de Rol como calabozos y dragones. Implican tomar el rol de un personaje el que puede estar especializado en combate o tener habilidades mágicas las cuales pueden ser modificadas a través de la historia del videojuego. (Adams & Rollings, 2003)

Micro-transacción: Son transacciones financieras que envuelven pequeñas cantidades de dinero las cuales por lo general ocurren a través de internet. En el contexto de los videojuegos son usadas como otra fuente de ingreso por las empresas desarrolladores las cuales pueden vender Monedas virtuales o Bienes virtuales con el fin de que los jugadores modifiquen aspectos del videojuego. (PayPal)

Moneda virtual: Es el tipo de dinero utilizado en los videojuegos con los cuales se pueden comprar bienes virtuales.

Bien virtual: Son objetos no reales que son comprados o intercambiados, estos no poseen un valor real. En el contexto de los videojuegos, son todos los ítems que son comprados o encontrados y que pueden o no afectar las características del videojuego. (Wu, 2007)

Unity: Es un motor de desarrollo para la creación de contenido interactivo 3D el cual provee un set de herramientas que aceleran el desarrollo de los videojuegos. Viene en versiones Free y Pro, y puede ser utilizado para desarrollar aplicaciones para Windows, Mac OSX, Navegadores Web, iPhone, iPad, dispositivos Android, Wii, PlayStation 3 y Xbox360. (Unity3D)

C#: Lenguaje de programación orientado a objetos creado y estandarizado por Microsoft para ser parte de su plataforma .NET. Posee una sintaxis derivada del C++ y el modelo de objetos de .NET.

App Store: Es un servicio disponibles en los dispositivos Apple con el cual los usuarios pueden buscar y descargar aplicaciones informáticas desarrolladas mediante la SDK de iPhone y publicadas por Apple. Las aplicaciones pueden ser bien descargadas gratuitamente o compradas. Apple otorga el 70% de las ganancias directamente al vendedor de la aplicación el 30% restante corresponde a Apple. (Apple Developer)

Google Play: Anteriormente llamada Android Market, es una tienda de software en línea desarrollada por Google para los dispositivos Android. Generalmente se encuentra preinstalada en la mayoría de estos dispositivos y permite buscar y descargar aplicaciones publicadas por terceros. (Rosenberg, 2012)

Metodología de cascada: Es un enfoque metodológico que ordena las etapas del proceso de desarrollo de software de manera que el inicio de cada etapa requiera el termino de la etapa anterior. Las etapas de la metodología de cascada son las siguientes: levantamiento de requerimientos, diseño, implementación, verificación y mantenimiento. (Select Business Solutions)

HUD: El HUD del inglés "Head-Up Display" es el método con el cual se le entrega información a los jugadores mediante una interfaz de usuario dentro del videojuego. Por lo general incluye diversa información como la vida de los personajes, diversos ítems o el puntaje del juego. (Wilson, 2006)

Documento de diseño de videojuegos: Documentación altamente descriptiva sobre el diseño del videojuego la cual es creada, mantenida y utilizada por los miembros del equipo que desarrollara el videojuego. Su formato es contextual al proyecto y su elaboración es generalmente demandado por las empresas que posean los derechos de distribución de los videojuegos. (Bates, 2004)

Atlas de textura: Es una gran imagen que contiene porciones de sub-imágenes, son utilizadas principalmente para optimizar el rendimiento de las tarjetas de video debido a la carga que supone cambiar la textura utilizada para dibujar. (Nvidia, 2004)

Estas definiciones son utilizados para la realización de los fundamentos teóricos del proyecto al ofrecer un lenguaje común sobre los videojuegos.

2.2 Fundamentos teóricos

El tener una base común de información puede ser utilizada para recopilar diferentes declaraciones y escritos que den un sustento teórico al desarrollo del proyecto, partiendo del como el desarrollo actual del mercado vuelve interesante el desarrollo de este tipo de proyectos.

Se puede extraer la siguiente cita de Briley Kenney al querer explicar lo atractivo que se ha vuelto el desarrollo de juegos para plataformas móviles.

"La industria de los videojuegos móviles está en auge; tan solo el año pasado (2010) hizo un estimado de 800 millones de dólares. Las personas están comprando juegos y aplicaciones en cualquier dispositivo que tenga disponible incluyendo dispositivos con iOS, Smartphone, Tablets y etc. Se ha llegado al punto en que la industria de videojuegos móviles esta actualmente quitando jugadores y ganancias de los mercados tradicionales de videojuegos". (Kenney, 2011)

Este crecimiento en las audiencias se ve reflejada por la distribución en las edades como bien se señala en el mismo artículo.

"Los videojuegos sociales están distribuidas en las audiencias de todas las edades, y son generalmente más usados por adultos entre 50 y 59 años en los Estados Unidos, con adultos entre 30 y 49 años justo atrás.

Es obvio que los videojuegos sociales se correlacionan directamente con la industria de videojuegos móviles y no es solo algo en lo que están muchos niños y adolescentes; todos lo están!". (Kenney, 2011)

Debido a esta nueva distribución de edades, se hace atractivo entrar en el mercado de las micro-transacciones, como señala Nathan Brown en un artículo referente a la empresa Kongregate.

"El portal de juegos flash Kongregate ha triplicado sus ganancias de las ventas de bienes virtuales el año pasado. Debido a un cambio de foco con relación a las micro-transacciones de Kongregate - la cual fue adquirida por GameStop en el año 2012- duplicando el numero de videojuegos con bienes virtuales en su sitio web en los últimos 12 meses. Las ventas de ítems virtuales ahora son el 80 por ciento de las ganancias de la compañía; el sitio tiene 15,5 millones de usuarios y provee de 50.000 juegos, siendo la mayoría gratis". (Brown, 2012)

Sobre cómo utilizar micro-transacciones dentro de los videojuegos Mike O'Brien habla de su experiencia en Guild Wars.

"Esta es nuestra filosofía sobre las micro-transacciones: Pensamos que los jugadores deben tener la oportunidad de gastar dinero en ítems que les provean una diferenciación visual y que les ofrezcan mayor maneras para expresarse. Ellos también deberían poder gastar su dinero en servicios o en ítems que les ayuden a ahorrar tiempo. Pero no está bien que los jugadores que compran un juego no sean capaces de disfrutar por lo que pagaron sin compras adicionales, y tampoco

está bien que los jugadores que gastan su dinero el tener ventajas injustas sobre los jugadores que gastan su tiempo.

Yo se que nada de esto es nuevo; el Guild Wars original también tenía micro-transacciones. Pero las micro-transacciones fueron algo pensado al final en Guild Wars, en cambio con Guild Wars 2, tuvimos la oportunidad de integrar las micro-transacciones desde el comienzo, dándole a los jugadores mayores opciones sin sacrificar nuestros principios de diseño. Así que, aquí esta lo que estamos haciendo distinto en esta oportunidad.

En Guild Wars 2 tenemos tres tipos de monedas: oro, karma y gemas. El oro es la moneda común del juego. El karma, el cual los jugadores ganan durante el juego no puede ser intercambiado, pero es usado para recompensas únicas. Y las gemas son la moneda que compran y usada para las micro-transacciones". (O'Brien, 2012)

Inclusive se han creado nuevas formas de generar ingresos donde no se requiera de micro-transacciones.

"En lo referente al marketing en plataformas iOS, no creo que sea difícil. Lo que si creo difícil es obtener una audiencia numerosa que se quede en tu juego, y que además gaste dinero en tu título. nosotros en Tap.Me estamos ayudando a los desarrolladores a resolver estos dos problemas. Le damos a los desarrolladores un set de herramientas con la cual monetizar a la gran mayoría de los jugadores que no están realizando micro-transacciones con una tarjeta de crédito mejorando su experiencia mediante mejoras dentro del juego o ítems a cambio asociarse con una marca. En otras palabras la marca se vuelve la forma de 'pago'. Como resultado los jugadores se quedan mayor tiempo en el juego y obtienen mejores resultados. Estamos haciendo algo que es simple de implementar, propio del juego y es una manera escalable para conectar a las marcas con los consumidores de una manera que beneficia a todos". (Kerr, 2012)

Por lo tanto elegir un proyecto en que se desarrolle un videojuego de rol que integre estas dos formas de conseguir ítems en el videojuego debe ser sustentado por una metodología de desarrollo. Por ello Bob Bates propone una modificación de la metodología de cascada

"La cascada modificada permite la sobre posición de pasos. Por ejemplo, tu puedes comenzar a programar alguna sección antes de que el diseño detallado haya sido completado, especialmente si esas subsecciones están bien entendidas y son módulos relativamente independientes. Si estás trabajando en un juego de trivia, por ejemplo, puedes decidir tempranamente como se verán las interfaces de

usuario, y las preguntas pueden ser desarrolladas en cualquier momento durante el desarrollo.

Otro género que se adecua al método de la cascada modificada es el de los juegos de aventura. Las interacciones con el escenario son algo muy importante para la forma de jugar, pero los diseñadores usualmente no sabrán exactamente como se verán estos ambientes hasta que los artistas los creen. Incluso cuando las bases de la historia y los puzzles son conocidos cuando el desarrollo comience, algunas interacciones específicas deben esperar hasta que el diseñador haya visto el mundo que el artista ha creado". (Bates, 2004)

Aparte de una metodología se hace necesario elegir las herramientas adecuadas para crear videojuegos. Daniel Will expone algunas características de por qué Unity es una buena herramienta de desarrollo de videojuegos

"Unity es simple. las herramientas de creación de mundos ayudan a los desarrolladores de aplicaciones iPhone a montar y modificar niveles de las aplicaciones a la velocidad del pensamiento. Editar, testear y probar las aplicaciones en desarrollo utilizando Unity. Tus ideas están solo a un clic de distancia. Unity te permite trabajar para encontrar la diversión, después controlar y pulir hasta que todo sea perfecto. Las plataformas pueden ser cambiadas solo con un clic. Desarrollar desde un solo código fuente para iPhone, iPad, móviles, webs, Mac/PC y consolas. Los desarrolladores de aplicaciones necesitan Unity para diseñar un ambiente sonoro y audio. Estos están incluidos. Unity incluye mapeo de luces y el sistema de oclusiones Umbra para hacer que los videojuegos se vean bien y corran rápido. El sistema de dibujado de Unity ayuda a los juegos a verse genial para el mercado de juegos casuales". (Will, 2011)

La siguiente es la metodología que la empresa ha desarrollado para el diseño de los videojuegos

- Recopilación de requerimientos de la gerencia
En estas reuniones se busca obtener la opinión de los mandos superiores de la empresa sobre los distintos elementos que componen el proyecto. También se busca llegar a consensos con respecto a los tiempos de desarrollo y a los recursos, monetarios y humanos, que se asignaran.
- Recopilación de requerimientos de los clientes
Las acotaciones del cliente resultan especialmente relevantes a la hora de definir los videojuegos. Por lo tanto mediante correos y conferencias electrónicas se debe tomar nota de los distintos requerimientos y peticiones que pueda tener la empresa con el proyecto.

- **Reunión de definición la jugabilidad**
Se deben realizar reuniones para definir la manera como se jugara el videojuego. Se trataran temas como: la manera en que el jugador interactuara, los tipos de estas interacciones, el sistema de combate y otros temas que dependen de la temática del proyecto. Estas reuniones son de carácter amplio y no buscan profundizar sobre los temas que se discuten, sino mas bien establecer una guía para evitar tocar generalidades en otras reuniones
- **Reunión de definición del estilo grafico**
En esta reunión se definirá el estilo grafico que guiara el proceso de desarrollo del arte del videojuego. Los resultados de esta reunión serán visibles en el documento de diseño del videojuego bajo el apartado de referencias, el cual es un capítulo destinado a que los artistas agreguen imágenes de otros videojuegos o películas que puedan servir como material de inspiración para el desarrollo de su trabajo.
- **Reunión de definición de las características del héroe**
Al tener un héroe sobre el cual se centrara la historia del videojuego es necesario realizar reuniones en las cuales se defina el progreso del personaje a lo largo de la esta. También se definirán sus movimientos y animaciones necesarios para que realice los distintos ataques que hayan sido previamente definidos.
- **Reunión de definición de los enemigos**
Debido a la naturaleza del videojuego se hace necesario el combate contra distintos enemigos. Durante esta instancia se deben definir la categoria de los enemigos, su aspecto visual, sus dimensiones, sus ataques, etc. Esta información debe ser traspasada al documento de diseño del videojuego en el apartado de enemigos en el cual se incluye un dibujo, el nombre y la descripción de sus ataques.
- **Reunión de definición de la historia**
Se deben realizar reuniones con el objetivo de definir los acontecimientos del videojuego. Se creara una línea de tiempo de los eventos que sucederán a lo largo del desarrollo de la historia. Línea de tiempo que finalmente será utilizada como pauta para la escritura del guion del videojuego.
- **Reunión de definición de los ítems**
Al ser RPG el género del juego se necesitara de diversos ítems que ayuden a modificar la experiencia del jugador a medida que se desarrolla la historia. Por

lo tanto se hace necesario generar una instancia en la que, a partir de los tipos de ítems definidos en las reuniones de jugabilidad, se genere un listado dividido por categorías donde se incluya la siguiente información tipo de moneda con la cual se venderá un ítem, descripción de los ítems, como afecta el utilizar el ítem. Esta información será traspasada al documento de diseño del videojuego en el apartado de ítems

- **Creación del documento de diseño del videojuego**
En un esfuerzo conjunto los equipos de arte, ingeniería y diseño plasmaran las ideas y temas relacionados con el proyecto. en un documento que sirve como guía para el desarrollo del proyecto. Los puntos a tocar en el videojuego si bien se desglosan de los puntos previamente nombrados, para este proyecto es como requisito como mínimo documentar: la historia, la línea de tiempo, la mecánica, los escenarios o niveles, el héroe, los enemigos, los personajes no jugadores, los ítems, los menús, el HUD y las referencias artísticas del videojuego.
- **Definición del material artístico del videojuego**
En esta etapa el líder del proyecto se reunirá con el director de arte para definir todo el material artístico que deberá ser realizado durante el desarrollo del proyecto. Cada ítem de esta lista corresponderá a una tarea que deberá ser asignada, de acuerdo a las habilidades requeridas, a uno de los miembros del equipo de arte.
- **Definición de los módulos y clases del videojuego**
Si bien es común que en la industria de los videojuegos los programadores escriben código a medida que lo necesitan, esta práctica resulta muy discutible cuando por su abuso una clase o modulo no considera a otros elementos del proyecto y crea incompatibilidades entre los distintos sistemas del programa, lo que finalmente resulta en una rescritura de una parte importante del código. Para evitar ese tipo de situaciones el equipo de ingeniería realizara una reunión en la cual se enumeren los distintos módulos que componen al videojuego, de esto se creara un listado de tareas que deberán ser resueltas por el equipo durante el desarrollo del proyecto.

CAPITULO III ELABORACION DEL DISEÑO DEL VIDEOJUEGO

Al ser este proyecto desarrollado con el auspicio de una empresa externa, se da mayor libertad a Trutruka sobre el contenido del producto. por lo tanto posee un mayor control sobre el producto y un gran número de elementos del videojuego deben ser definidos por la compañía. Por lo tanto utilizando el proceso ya definido por la empresa, y expuesto en el marco teórico, se especifican las distintas características del videojuego.

3.1 Recopilación de requerimientos de la gerencia

El siguiente es un listado de requerimientos que fueron obtenidos de parte de la gerencia.

- El juego debe funcionar en Ipad 2 sin problemas gráficos debido a su ratio gráfico (4:3)

3.2 Recopilación de requerimientos de los clientes

A continuación se enumeran los requerimientos expuestos por el cliente sobre el desarrollo del videojuego

- El videojuego debe tener integrado el sistema de recompensas solo en Android
- Se debe utilizar una librería proporcionada por ellos para la integración del sistema

3.3 Reunión de definición la jugabilidad

La siguiente es una tabla con las características de jugabilidad que se pudieron recopilar en la reunión entre los distintos equipos que participan del desarrollo del videojuego

Tabla 1: Jugabilidad del videojuego

Requerimiento	Información
Interacción del jugador	<p>El jugador interactuara con el videojuego presionando la pantalla táctil de los dispositivos, este podrá:</p> <ul style="list-style-type: none">• Utilizar botones• Seleccionar enemigos para atacar• Seleccionar un lugar del mapa y dirigirse a el

Forma como el jugador sabe el objetivo del juego	Al jugador se le hará saber el objetivo mediante diálogos en los cuales se le asigna misiones que debe cumplir a lo largo del juego
Número de jugadores	El juego será programado y diseñado para soportar un solo jugador
Flujo entre los menús y niveles	El juego tendrá un menú de inicio y durante el videojuego otro de pausa. Desde el menú de inicio se puede iniciar un nuevo juego o cargar uno anterior. Ya en el videojuego se cargará una ciudad en la que al jugador se le dará una misión, la cual al ser aceptada cargará una escena especialmente diseñada para cumplirla
Elementos presentes en los niveles del juego	Cada nivel del juego debe tener por lo menos una caja que define el área sobre la cual el héroe puede moverse
Manera en que se contara la historia del juego	La historia será contada mediante diálogos e ilustraciones animadas durante el juego
Manera en que serán utilizadas las micro-transacciones	Las micro-transacciones serán utilizadas para comprar gemas las cuales son un tipo de moneda especial en el juego, útiles para comprar ítems especiales
Tipo de ítems del videojuego	Se definieron las siguientes categorías de ítems: <ul style="list-style-type: none"> • Objetos de misiones • Espadas • Escudos • Armaduras • Botas • Anillos • Pociones

Sistema de combate	<p>Se creara un sistema de combate que soporte los siguientes tipos de ataques:</p> <ul style="list-style-type: none"> • Cuerpo a cuerpo <ul style="list-style-type: none"> • Con armas • Cargando el cuerpo • A distancia <ul style="list-style-type: none"> • Con proyectiles lineales (ej: flechas) • Con proyectiles tele dirigidos (ej: bolas de poder) • Mágicos (ej: robo vida)
Forma en que las tiendas venderán los ítems del juego	<p>Existirán tiendas en las ciudades del juego que venderán distintos ítems, sus vendedores son los siguientes personajes no jugables</p> <ul style="list-style-type: none"> • Un guerrero: el cual vende espadas y armaduras • Un herrero: el cual vende escudos y botas • Una mercader: la cual vende pociones y anillos

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

3.4 Reunión de definición del estilo grafico

La siguiente tabla resume las distintas definiciones que tomo el equipo de arte durante la reunión que buscaba definir el estilo grafico del videojuego

Tabla 2: Estilo grafico del videojuego

Requerimiento	Información
La paleta de colores	El juego no tiene una paleta previamente definida, se utilizaran distintas paletas que dependerán principalmente de los escenarios
Estilo y grosor de la línea de los gráficos	Los gráficos no deben tener líneas negras y los grosores deben ser de un solo pixel. Sin considerar los contornos que los artistas quieran utilizar para acentuar características de los personajes

Estilo de los menús	El menú principal del videojuego cargara uno de los escenarios de las misiones como fondo animado y tendrá una ilustración, también animada, realizada a lápiz del héroe
Estilo del HUD del videojuego	Los elementos del HUD deben situarse en la parte superior e inferior de la pantalla, con el fin de no entorpecer las interacciones del jugador
Estilo del inventario	El inventario deberá ser dividido en cuatro lengüetas <ul style="list-style-type: none"> • Estadísticas del personaje • Habilidades y pociones • Inventario • Opciones del juego
Estilo visual de las ilustraciones animadas	Las ilustraciones animadas del videojuego seran dibujadas de tal modo que se aprecie el uso de lápices en ellas
Forma de composición de los niveles	La parte superior de los escenarios corresponde al fondo de ellos y la inferior a espacio no utilizado sobre el cual se dibujara el HUD. Se estima que el área por la cual el héroe podrá pasar es solo la mitad del escenario
Forma en que se deben diseñar los personajes	En la mayoría de los casos se deberán crear animaciones para las siguientes acciones: <ul style="list-style-type: none"> • Espera • Ataque • Caminata • Muerte

Proporciones de los personajes	Se definió que la altura del héroe fuese de aproximadamente un cuarto de la altura del escenario y los distintos enemigos pueden ir desde pequeños enemigos del porte de los pies del héroe hasta gigantes que utilicen la mayor parte de la pantalla
Tipografías	Las siguientes tipografías fueron elegidas para ser utilizadas en el videojuego: <ul style="list-style-type: none"> • Fontin SmallCaps • Folk Solid
Escenarios	La siguiente lista es el tipo de escenarios sobre el cual se basaran las ilustraciones de ciudades y campos de batalla <ul style="list-style-type: none"> • Ciudad medieval • Bosque elfico • Isla polinésica • Región volcánica • Inframundo

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

3.5 Reunión de definición de las características del héroe

Se realizó una reunión con el solo fin de definir las características principales del héroe, de manera que se pudiesen esclarecer algunos atributos y motivaciones del personaje

El resultado fue definirlo como:

- Un caballero oscuro
- De pelo castaño y muy pálido
- El cual es inmortal
- Utiliza espadas para atacar
- Tiene 10 habilidades especiales
- Le motiva el buscar venganza

3.6 Reunión de definición de los enemigos

El siguiente es el listado de enemigos del videojuego

- Murciélago
- Parca
- Fantasma de un ojo
- Teniente parca
- Lord demoniaco nivel 1
- Oruga
- Planta
- Hombre lobo
- Gran hombre lobo
- Arquera elfo oscura
- Verdor oscuro
- Guerrero elfo oscuro
- Cuervo
- Nube eléctrica oscura
- Fantasma pirata
- Blood steel
- Sombra de blood steel
- Ladrón
- Kraken
- Abeja demoniaca
- Diablillo
- Cráneo de fuego
- Caballero rojo
- Golem de lava
- Magma Fenix
- Beholder
- Alma maldita
- Brujo
- Ejecutor
- Empalador
- Aurora
- Lord demoniaco nivel 2

3.7 Reunión de definición de la historia

La siguiente es la línea de tiempo que se definió para los distintos hechos del videojuego. De esta línea de tiempo se confeccionara el guion que finalmente será utilizado

Tabla 3: Línea de tiempo del videojuego

Misiones	Combate
1	Primer combate
3	Teniente parca
4	Lord demoniaco nivel 1
7	Hombre lobo
9	Verdor oscuro
14	Blood Steel
17	Kraken
21	Golem de lava
24	Magma Fenix
27	Ejecutor
29	Empalador
30	Aurora
31	Lord demoniaco nivel 2

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

3.8 Reunión de definición de los ítems

La siguiente es una lista con los diferentes ítems ordenados por categoría

Tabla 4: Ítems del videojuego

Categoría	Ítems
Pociones	<ul style="list-style-type: none">• Poción de vida• Poción de magia• Poción de furia• Poción de piel de hierro• Poción de suerte• Poción de Hermes• Poción de lagrimas de fénix• Poción de elixir
Espadas	<ul style="list-style-type: none">• Espada del adepto• Espada del acolito• Espada bendita• Espada de los templarios

	<ul style="list-style-type: none"> • Espada del centinela • Espada del vigilante • Espada de los sabios del bosque • Espada del árbol de la vida • Espada del marino • Espada del contramaestre • Espada del bucanero • Espada del capitán • Espada forjada • Espada de mythril • Espada volcánica • Espada del dragón • Espada etérea • Espada de los espíritus • Espada maldita • Espada demoniaca • Espada sagrada
Escudos	<ul style="list-style-type: none"> • Escudo del adepto • Escudo del centinela • Escudo del marino • Escudo forjado • Escudo etéreo • Escudo de los templarios • Escudo de los sabios del bosque • Escudo de los bucaneros • Escudo volcánico • Escudo maldito • Escudo sagrado
Armaduras	<ul style="list-style-type: none"> • Armadura del adepto • Armadura del acolito • Armadura bendita • Armadura del centinela • Armadura del vigilante • Armadura de los sabios del bosque • Armadura del contramaestre • Armadura del bucanero • Armadura forjada

	<ul style="list-style-type: none"> • Armadura de mythril • Armadura volcánica • Armadura etérea • Armadura de los espíritus • Armadura maldita • Armadura sagrada
Botas	<ul style="list-style-type: none"> • Botas del adepto • Botas del acolito • Botas benditas • Botas del centinela • Botas del vigilante • Botas de los sabios del bosque • Botas del marino • Botas del contramaestre • Botas del bucanero • Botas forjadas • Botas de mythril • Botas volcánicas • Botas etéreas • Botas de los espíritus • Botas malditas • Botas sagradas

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

3.9 Creación del documento de diseño del videojuego

Se creó un documento titulado "Documento de diseño Soul Avenger" el cual es un compendio con la información definida en las reuniones. En este documento se agregan imágenes de los resultados finales ya que este documento se va actualizando de manera constante a lo largo del proyecto. Aquí se describen las distintas características de los enemigos, los ítems y las misiones.

3.10 Definición del material artístico del videojuego

Se creó una hoja de cálculo titulada "Tareas Soul Avenger" en la cual se enumeran las distintas tareas del equipo de arte en las siguientes categorías

- Integración de enemigos
- Misiones
- Inventario

- Tiendas de compra
- Sistema de estadísticas
- Sistema de diálogos
- Sistema de guardado
- Menús y HUD
- Héroe

3.11 Definición de los módulos y clases del videojuego

Si bien es común que en la industria de los videojuegos los programadores escriben código a medida que lo necesitan, esta práctica resulta muy discutible cuando por su abuso una clase o modulo no considera a otros elementos del proyecto y crea incompatibilidades entre los distintos sistemas del programa, lo que finalmente resulta en una rescritura de una parte importante del código. Para evitar ese tipo de situaciones el equipo de ingeniería realizó una reunión en la cual se enumeraron los distintos módulos que componen al videojuego, el siguiente es el listado de los distintos módulos y clases que se listaron en esa reunión

- Un modulo que maneje el progreso
- Un modulo que maneje la escritura y carga de datos
- Una clase que maneje al personaje principal del videojuego
- Una o varias clases que manejen a los enemigos del videojuego
- Módulos para manejar los distintos ataques del juego
- Un modulo para manejar las interacciones del jugador (input)
- Un modulo que se encargue de manejar el inventario
- Un modulo de compra y venta de ítems
- Un modulo para manejar las micro-transacciones del videojuego
- Un modulo que se encargue de dibujar los menús del juego
- Un modulo encargado de dibujar el HUD
- Un modulo responsable de presentar los diálogos

CAPITULO IV DESARROLLO DEL PROYECTO

Al ser la naturaleza del proyectos de un carácter multidisciplinario distintas disciplinas convergen en su desarrollo. A continuación se describen las actividades desarrolladas por los equipos de arte, diseño e ingeniería para construir el videojuego.

4.1 Creación de los escenarios

Fueron creadas en total 5 ciudades y 17 escenarios de batalla. A continuación se pueden observar algunas de las ilustraciones que fueron creadas.

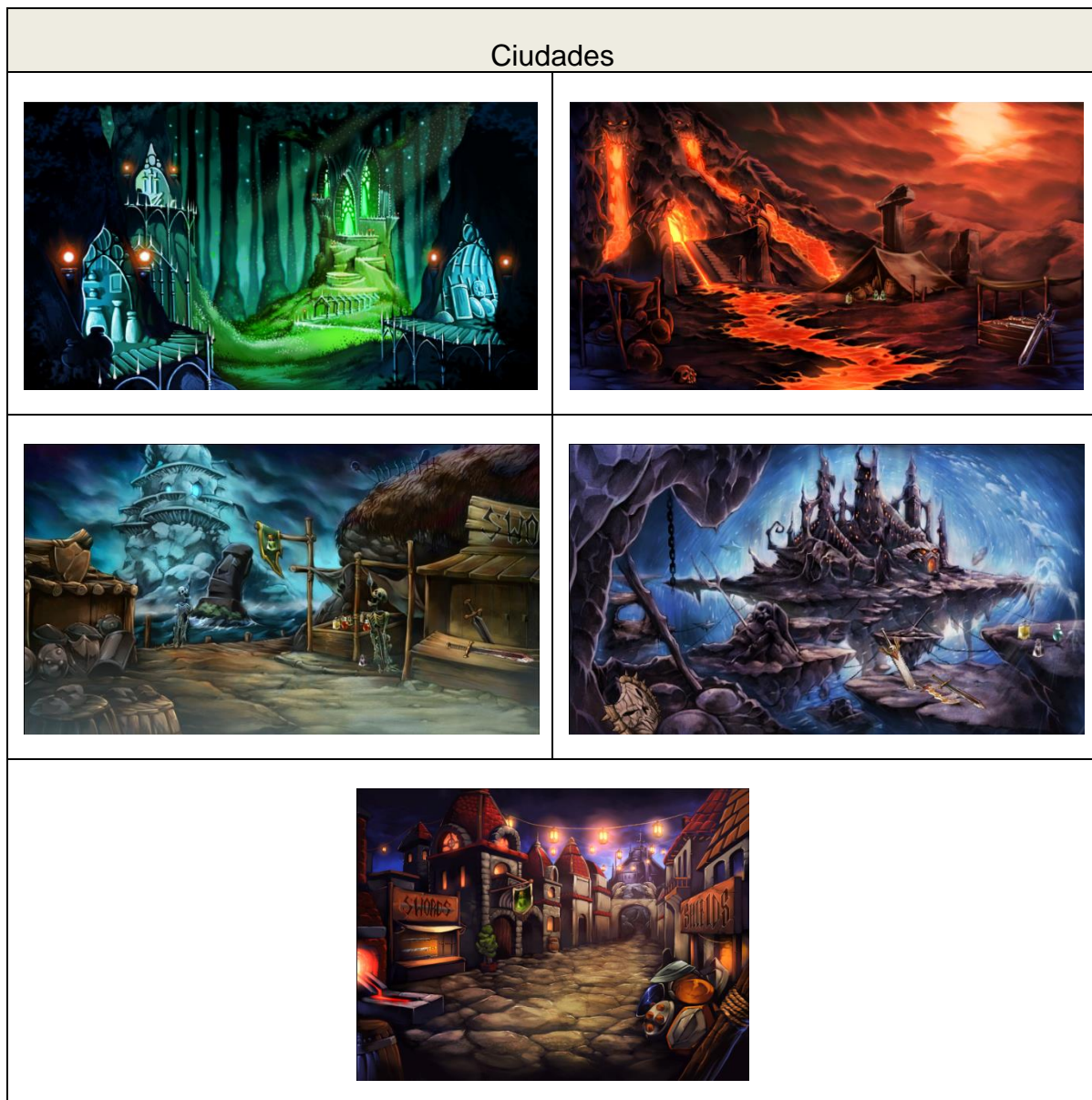


Imagen 1: Ilustraciones de las ciudades del videojuego.

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

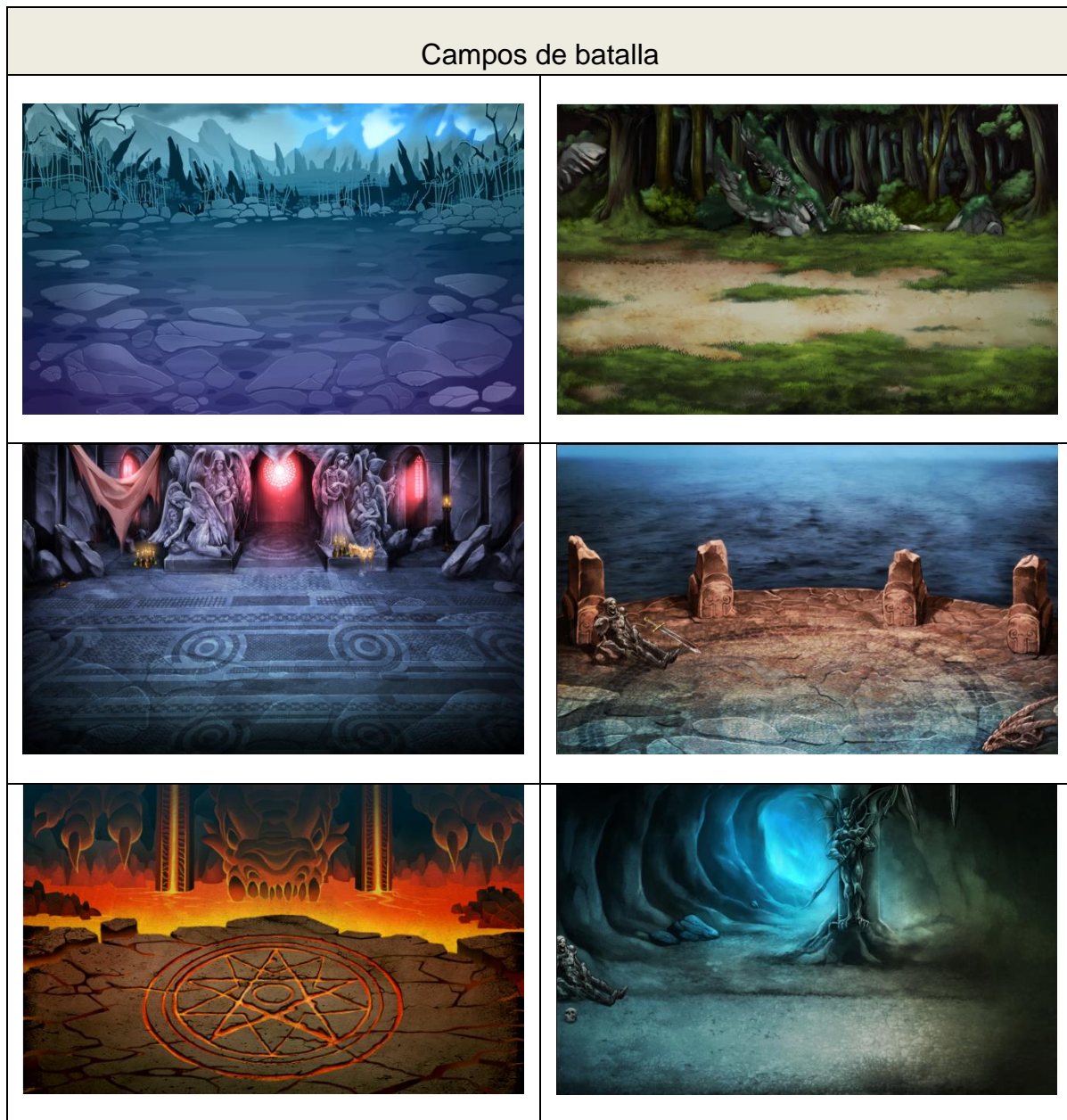


Imagen 2: Ilustraciones de los campos de batalla del videojuego.
Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

La paleta de colores de los escenarios es única pero temática. Tonos púrpuras para un escenario de cementerio, verdes para un bosque y rojos para un volcán. Es importante entender la configuración de los campos de batalla, la parte superior corresponde al fondo del escenario con el cual se intenta dar un contexto a la misión en curso y la parte inferior corresponde al área en la cual ocurre la acción, por esta área el héroe y los enemigos se pueden mover y atacar ya sea con ataques cuerpo a cuerpo o a distancia.

4.2 Creación de los fotogramas del héroe

Se crearon fotogramas para el héroe los cuales buscan animar las distintas acciones que puede realizar, estos fotogramas son aglomerados en un grupo de varias imágenes llamadas atlas, la siguiente imagen corresponde a una de ellas.

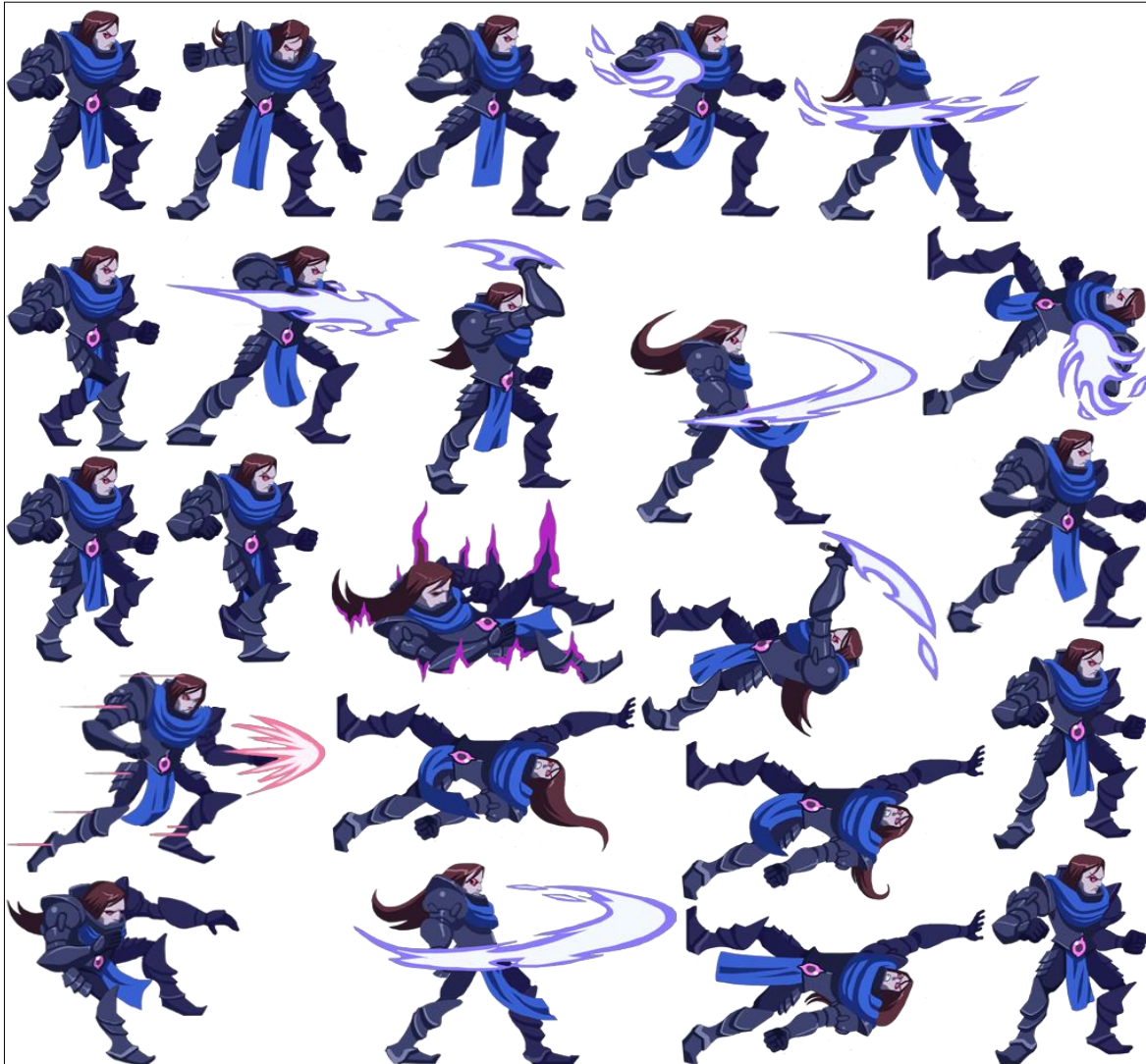


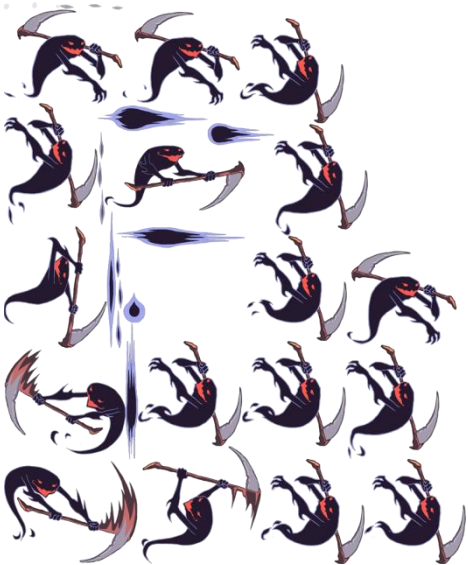


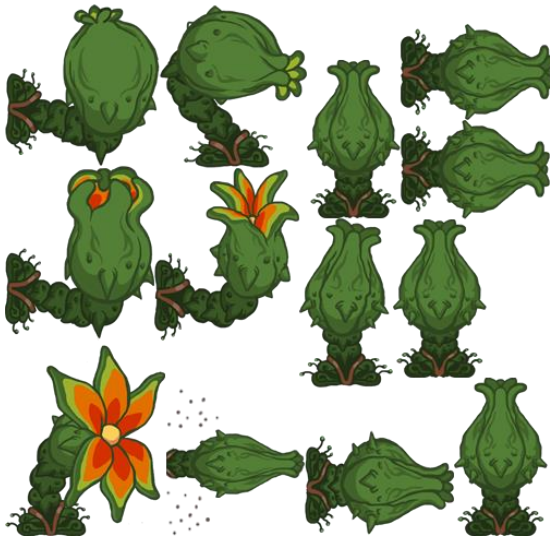
Imagen 3: Unión de los fotogramas del héroe.

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

Esta imagen es resultado de juntar los distintos fotogramas en un elemento llamado Sprite Collection el cual se encarga de optimizar el uso de memoria de los fotogramas utilizando el mínimo espacio necesario para su construcción. Se puede notar que estos objetos al aglomerar los recursos en una sola gran imagen optimizan espacio orientando algunos de sus fotogramas de tal manera que queden de manera horizontal, esto no requiera de consideraciones especiales ya que es manejado de manera interna por las tarjetas de video de los dispositivos.

4.3 Creación de los fotogramas de los enemigos

Los distintos fotogramas de los enemigos dependen principalmente de las acciones de ellos. Como base todos los enemigos tienen una animación de esperar, moverse, atacar y morir. A continuación se presentan algunos atlas de distintos enemigos en los cuales se pueden observar los distintos fotogramas.

Parca	Hombre lobo
	
Ejecutor	Planta
	

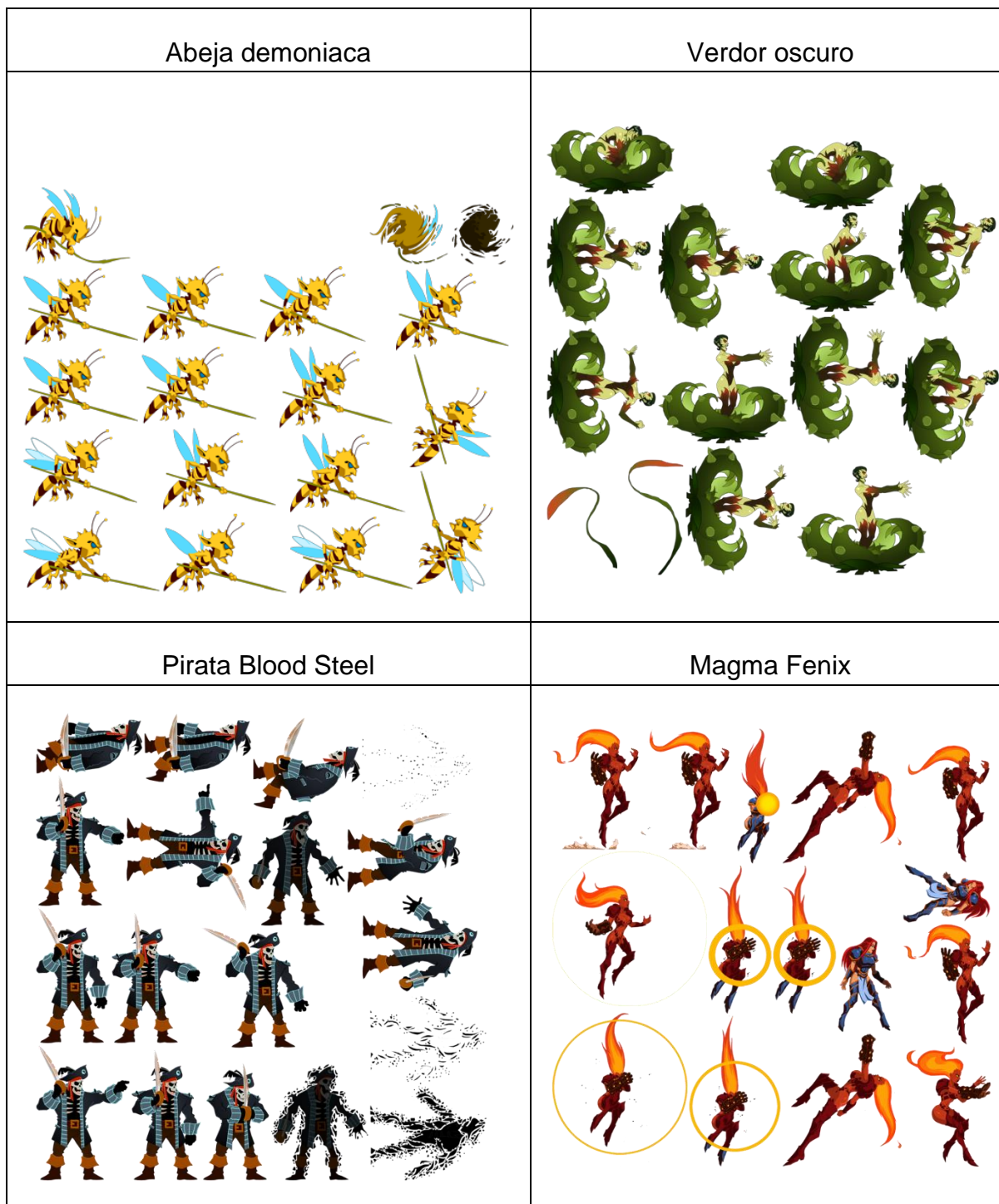


Imagen 4: Unión de los fotogramas de los enemigos.

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

Estas imágenes corresponden a algunos de los enemigos del videojuego. En total el proyecto se compone de 153 atlas que contienen los distintos fotogramas de los enemigos del proyecto.

4.4 Creación de las interfaces del inventario

Las siguientes son las 4 lengüetas base del inventario



Imagen 5: Las cuatro lengüetas del inventario.

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

Cada una de estas pantallas se encarga de cambiar distintos elementos del juego.

- La pantalla de estadísticas del héroe permite al jugador aumentar sus puntos de fuerza, agilidad, vida y magia.
- La pantalla de habilidades y pociones permite asignar cualquiera de las 10 habilidades del héroe a uno de los slots de acceso rápido de uso, también permite conocer y consumir, en caso de existir, cualquiera de las 8 pociones del videojuego.
- La pantalla de equipo permite seleccionar los distintos ítems que pueden ser equipados por el personaje, estos pueden ser de las siguientes categorías: espadas, escudos, botas, armaduras y anillos.
- La pantalla del juego permite volver al menú principal, cargar la última partida guardada y cargar una partida distinta a la actual

4.5 Creación de la clase Game

el juego se compone de una clase Game la cual se instancia mediante un singleton y se encarga de manejar la inicialización del videojuego como también del flujo entre los distintos estados del juego, para ver el código fuente que enumera esos estados, ver Anexo N°1.

Este singleton posee una función que es llamada en cada cuadro la cual se encarga de ordenar la profundidad de los objetos, evaluar los tutoriales del juego y actualizar los estados del juego, el código fuente de esta función puede ser encontrado en el Anexo N°2.

De esa manera solo cuando el estado del juego sea InGame se llamara a la función UpdateInGame, la cual se encarga de crear nuevas oleadas de enemigos y evaluar si las condiciones para terminar la misión han sido dadas. Su código fuente puede ser revisado en el Anexo N°3.

4.6 Creación de la clase DataGame

La clase DataGame se encarga de cargar y guardar datos desde y hacia archivos XML, de manera de administrar la partida del jugador. Posee 2 funciones básicas

- public static void writeSaveGame(int slot), la cual recolecta los datos de la partida y escribe un archivo XML indexado por la variable slot.
- public static bool loadSaveGame(int slot), la que carga un archivo XML cuyo nombre depende de la variable slot y escribe los valores en la sesión de juego (clase Game).

4.7 Creación de la clase Character

La clase Character posee las características básicas de un personaje capaz de interactuar en el videojuego. Posee funcionalidades para orientarse a la izquierda o derecha, el código asociado a esa funcionalidad puede ser visualizado en el Anexo N°4.

También posee una función para mover al personaje, la cual recibe la nueva posición estimada mas una variable que permite escapar de los límites del escenarios conforme al Anexo N°5.

4.8 Creación de la clase Hero

De la clase Character se deriva la clase Hero, la cual posee funcionalidades extendidas ya que puede interpretar los clics del usuario en la pantalla y tomar la

decisión si debe dirigirse al lugar o atacar a un enemigo en esa posición, el código fuente que agrega estas características puede ser revisado en el Anexo N°6.

4.9 Creación de la clase BasicEnemy

La clase BasicEnemy posee una función InGameUpdate la cual es llamada solo cuando el estado del videojuego es InGame. La clase BasicEnemy posee características similares a la clase Hero, salvo que la manera de realizar acciones es mucho más simple, el enemigo intenta obtener un objetivo en la escena (el héroe) y un ataque con el cual poder atacarlo, el código fuente correspondiente al Anexo N°7.

4.10 Creación de la clase Item e Inventory

La clase Item posee un enumerador para los distintos tipos de ítems del videojuego, estos pueden ser enumerados en el Anexo N°8.

La clase Inventory es un singleton capaz de manejar los distintos ítems que posee el jugador. maneja un diccionario donde liga los ítems con la cantidad de veces que estos sean poseídos por el jugador. El código con el contenedor del inventario del jugador está reflejado en el Anexo N°9.

Además posee funciones de administración como:

- `public int getItemAmount(Item item)`, la cual permite conocer la cantidad de ítems de un cierto tipo.
- `public void addItem(Item item)`, la cual permite agregar un ítem nuevo al diccionario.
- `public void removeItem(Item item,int amount)`, la cual permite eliminar un ítem del listado.
- `public Dictionary<Item,int> getItemsOfType(int begin,int amount,int mask)`, la cual permite crear un subconjunto de ítems a partir del diccionario base y filtrar mediante el uso de la variable mask los distintos tipos de ítems que se quieran obtener.

Finalmente posee una función "consume" la cual es declarada de la siguiente manera `public void consume(Item item,int amount)`. Esta función es utilizada solo en el caso de las pociones, ya que dependiendo de que poción sea utilizada un evento especial podría ejecutarse, como por ejemplo crear un efecto visual sobre el héroe o cambiar algunas características del juego.

4.11 Creación de la clase Attack

Se implementó una clase Attack de la cual se derivan los ataques del videojuego, esta función base posee como responsabilidad el definir si el ataque puede o no puede ser utilizado en un determinado momento, utiliza un sistema de daño por segundo, el cual es un ratio entre el daño medio del ataque y la duración de su animación. Las funciones utilizadas para calcular el daño de un ataque y de si este puede ser utilizado esta reproducido en el Anexo N°10.

4.12 Creación del guion

Se coordinaron esfuerzos para crear un guion para el videojuego, guion que en total se compone de 31 misiones con diversos objetivos las cuales se ajustaron a las características de la línea de tiempo ya definida. Este documento tiene el nombre de "Guion Soul Avenger" y es usado como referencia para esclarecer los distintos eventos del juego.

4.13 Creación de los niveles

los niveles de batalla del videojuego requieren que un área de colisión sea creada. Mediante una herramienta de edición, los Sprite Collections pueden definir geometrías de colisiones asociadas a una de sus imágenes, este proyecto requiere que se defina un área cuadrada por la cual el héroe del juego se pueda mover.

A continuación se muestra una imagen de la herramienta utilizada para este proceso.



Imagen 6: Herramienta de demarcación del área de juego de los niveles.

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

Después de haber definido el área sobre la cual pueden caminar el héroe y los enemigos, se vuelve necesario agregar la ambientación del escenario, para eso se crearon distintos objetos animados con el fin de agregarle dinamismo y evitar la monotonía de ver un escenario estático.







Objetos animados de los escenarios		
		
		

Imagen 7: Algunos objetos animados de los niveles.

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

4.14 Creación de los enemigos

Se integraron los distintos enemigos en el juego, de manera que pudiesen ser utilizados para contar la historia del videojuego. En la siguiente imagen se puede observar a distintos enemigos instanciados en una escena.



Imagen 8: Enemigos creados en el videojuego.

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

Cada enemigo fue configurado de manera que sus diálogos, sus ataques y sus movimientos permitiesen integrarlos dentro de los eventos del juego.

4.15 Coordinación con el proveedor de sonidos

Se realizaron reuniones con el proveedor de sonidos en la cual se llegó al siguiente acuerdo.

- Cada ciudad tiene un tema musical.
- Crear 6 temas musicales de batalla.
- Crear 3 temas musicales para los jefes.
- Crear 1 tema musical para el menú principal.
- Crear 1 tema para cuando el jugador pierda una misión.
- Crear 1 tema para cuando el jugador complete una misión.
- Crear 1 tema para ser utilizado en el menú de navegación entre ciudades.
- Crear 4 temas para las cinemáticas de la historia del juego.
- Crear más de 150 efectos de sonido.
- Crear una introducción que permita conocer la historia del juego.

4.16 Integración de los sonidos al videojuego

Los distintos sonidos del videojuego fueron integrados de dos maneras, la música fue asignada a cada misión dependiendo de sus características y los efectos sonoros fueron integrados en los enemigos como parte de sus ataques, en total se estima que se integraron mas de 150 efectos de sonido incluyendo golpes, gritos, magias y sonidos ambientales de los niveles del juego. La introducción fue utilizada con una secuencia animada para contar la historia del videojuego.

4.17 Integración de las micro-transacciones

Se adquirió un modulo cuyo nombre es TPaymentManager capaz de hacer la conexión entre Unity y la interfaz nativa de compra de los dispositivos móviles. Esta librería permite inicializar el servicio de ventas, agregar un producto al carro de compras y esperar por el resultado de la transacción. En caso de que la compra haya sido realizada entregar el ítem virtual en el videojuego.

En el videojuego se ofrecen 6 productos, los cuales son paquetes de gemas, las cantidades a ofrecer serán definidas en la etapa de cierre del proyecto, pero sus valores deben ser escalonados, como también la cantidad de gemas ofrecidas.

4.18 Integración de TapJoy

TapJoy provee un modulo Java cuya ruta es `com.tapjoy.TapjoyConnect`, la cual se encarga de proveer el acceso a TapJoy para dispositivos Android. para operar con este modulo se debe crear un puente de conexión entre C# y Java lo cual se realiza utilizando una clase llamada `AndroidJavaClass` que permite crear instancias de una clase solo con solo utilizar su nombre, también permite llamar a los distintos métodos de sus instancias.

Mediante una función que sea llamada cuadro a cuadro se deben consultar las funciones `didReceiveGetTapPointsData` y `getTapPointsTotal`, la primera función permite saber si las operaciones de TapJoy que entregan puntos a los jugadores han finalizado y la segunda permite saber la cantidad de gemas que el jugador recibió.

CAPITULO V CIERRE DEL PROYECTO

La etapa de cierre considera hacer los arreglos finales para la eventual publicación del producto, al ser este proyecto multiplataforma se deben hacer arreglos para integrarlo en Google Play y App Store, pero antes, la cantidad de gemas que TapJoy entrega como recompensa debe ser calibrado.

A continuación se explican los pasos de cada una de estas tareas.

5.1 Asignación de recompensas con TapJoy

TapJoy posee un sistema online de registro de recompensas, en el la empresa puede agregar distintas aplicaciones. El proyecto debe ser registrado llenando un formulario en la que se indica el nombre de la aplicación y su plataforma (Android).

Se registrara producto creando un App ID y un App Secret Key datos que son utilizados por la librería de comunicaciones provista por esta empresa para conectarse al sistema de recompensas y ofrecer distintas alternativas al jugador, como ver publicidad o llenar formularios a cambio de gemas.

Las gemas son creadas en el formulario "Virtual Currency" en donde se crearan las gemas en el sistema indicando cuantas son equivalentes a 1 dólar, este valor ya fue definido, haciendo equivalentes 50 gemas a 1 dólar norteamericano. Se debe indicar positivamente si TapJoy debe manejar la entrega de recompensas e indicar el nombre de la moneda virtual, el cual es "Gems" para este caso.

Luego se debe enviar un correo support+enable@tapjoy.com pidiendo la habilitación de la moneda virtual y agregar el ID de un dispositivo de prueba para testear el sistema de recompensas, para lograr esto último se debe instalar un programa llamado TapJoy Device ID Tool el cual puede ser encontrado en Google Play.

Una vez que el producto sea publicado el "Application State" de la aplicación debe ser cambiada de "Not Live" a "Live"

5.2 Registro del producto en Google Play

Para registrar el videojuego en la plataforma Google Play se deben realizar los siguientes pasos.

- Crear un instalador del videojuego mediante el botón "Build" en los "Build Settings" de Unity, esto creara un archivo de extensión .apk en el directorio que se le indique al proceso.

- Acceder con la cuenta de desarrollo registrada en Google Play a la "Android Developer Console".
- Hacer clic en Subir Aplicación.
- Seleccionar el instalador del archivo (el archivo con extensión .apk) y seleccionar "Publicar".
- Luego se deben subir al menos 2 capturas de pantalla del videojuego y un icono en alta resolución.
- Se debe agregar un título para el proyecto y una descripción, ambas en inglés.
- El tipo de aplicación debe ser clasificado como "Juegos" y en el apartado de "Categoría", seleccionar "Arcade y acción".
- Luego se debe dar el consentimiento de que la aplicación cumple con las directrices de contenidos para Android y que está sujeta a las leyes de exportación de Estados Unidos independiente de la ubicación gráfica o nacionalidad.

5.3 Registro de los paquetes de gemas en Google Play

Para agregar la opción de que los jugadores compren gemas en la versión Android de la aplicación se deben realizar los siguientes pasos.

- Agregar el permiso "com.android.vending.BILLING" al manifiesto de la aplicación, este permiso añade la opción de vender contenido dentro del videojuego.
- Volver a generar el instalador de Android utilizando el botón "Build" de Unity.
- Ingresar a la pagina del proyecto en "Android Developer Console".
- Ingresar a la opción de "Archivos APK".
- Hacer clic en "Subir APK".
- Seleccionar el archivo previamente generado y esperar a que su envío sea completado.
- Volver al portal de "Android Developer Console".
- Hacer clic en la opción "Productos integrados en la aplicación" de la aplicación.
- Ingresar a "Añadir producto integrado en la aplicación", esto redirige a un formulario titulado "Creación de nuevo producto integrado en la aplicación".
- Agregar un ID de producto, un título y una descripción al producto.
- Agregar un precio en dólares y hacer clic en el botón de Autocompletar, para calcular los precios en las distintas monedas compatibles.
- Hacer clic en Publicar

5.4 Registro de producto en App Store

El registro del videojuego para iOS debe realizarse utilizando un ordenador Mac, para este cometido fue especialmente comprado un Mac Mini con el fin de crear los instaladores para esta plataforma. Para ello Unity debe ser instalado en el ordenador en conjunto con el proyecto, a continuación se describen los pasos para registrar el producto en App Store

- En el "Apple Developer Portal" se debe crear un Application ID para el proyecto.
- Crear el proyecto XCode del videojuego mediante el botón "Build" en los "Build Settings" de Unity.
- Abrir el proyecto generado mediante XCode y generar el instalador del proyecto, al finalizar el proceso generara un archivo con extensión .ipa.
- Conectarse a la web de "Itunes Connect" y hacer clic en "Manage Your App".
- Hacer clic en "Add New Application".
- Ingresar en el formulario de producto el nombre y la descripción de la aplicación.
- Asignarle una categoría al producto
- Completar su Copyright, su versión y su número SKU
- Añadir palabras claves del producto para el buscador de aplicaciones (RPG, Soul, Avenger, Battle, etc)
- Subir el archivo .ipa, un icono de 512x512 pixeles y 2 screenshots de 480x320 pixeles.
- Finalmente se debe indicar la fecha desde la cual el producto estará disponible y su valor expresado en distintas monedas (dólares, euros, libras, etc.).y tu

5.5 Registro de los paquetes de gemas en App Store

- Ingresar al sitio de "Itunes Connect" y hacer clic en "Manage Your App".
- En las opciones de la aplicación, hacer clic en "Manage In-App Purchases".
- Hacer clic en "Create New".
- Elegir el tipo de compra, para este caso es "Consumable".
- A continuación aparecerá un formulario en el cual se deben llenar los siguientes datos.
- Nombre de despliegue.
- ID del producto.
- Lenguajes que soporta el producto.
- Notas de revisión.

- Capturas de pantalla.
- A continuación se debe llenar un formulario llamado "Pricing and Availability Information", en el cual se indica si el producto está listo para ser ofrecido (Cleared for sale) y un valor para el producto (Price Tier).
- Finalmente se debe aceptar la creación del producto dentro de App Store.

CAPITULO VI RESULTADOS OBTENIDOS

Se logro crear un videojuego que esta pronto a entrar a la etapa de cierre, se implementaron los eventos del juego, los diversos enemigos y sus distintas características, dando como resultado final un producto para plataformas iOS y Android. El videojuego es capaz de realizar micro-transacciones utilizando los servicios de Google Play y App Store. Mediante TapJoy es posible premiar a los usuarios con gemas, las cuales pueden ser canjeadas por ítems que mejoran la experiencia del juego.

La siguiente es una imagen del producto



Imagen 9: Menú principal del videojuego.

Fuente: Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.

CONCLUSIONES

1. Mediante las distintas instancias de diseño, se logro crear un documento capaz de reflejar partes tan importantes del videojuego como su jugabilidad, el estilo grafico, sus enemigos o su línea de tiempo. Al haber realizado el diseño de esta manera fue clave al planificar la siguiente etapa del desarrollo del proyecto, ya que se logro disminuir la incertidumbre que había con el proyecto por haber definido su listado de tareas.
2. Con gran parte del diseño del videojuego ya realizado, los equipos de arte e ingeniería comenzaron a trabajar en sus distintas actividades, las cuales al haber sido planificadas dieron como resultado que el equipo de arte no tuviese ningún atraso importante en sus entregas. El equipo de ingeniería, logró utilizar las librerías de micro-transacciones de manera adecuada en sistemas Google Play y App Store. También, el hecho de haber definido la herencia de las clases y la manera en que se crearían los niveles, dio como resultado un proceso flexible y simple para la creación de los niveles y los enemigos del juego.
3. Si bien la etapa de cierre del proyecto no ha sido alcanzada aún, sus pasos ya han sido definidos. Por lo tanto se espera que durante un periodo de dos semanas, el proceso de certificación del juego se encuentre finalizado, puesto que Apple debe revisarlo y dar el visto bueno para la publicación.

BIBLIOGRAFIA

- Adams, E., & Rollings, A. (2003). Andrew Rollings and Ernest Adams on Game Design. New Riders Publishing.
- Apple Developer. (s.f.). Recuperado el 3 de 6 de 2012, de Apple Developer: <https://developer.apple.com/programs/ios/distribute.html>
- Baer, R. H., Rusch, W. T., & Harrison, W. L. (1969). Patente nº 3659285. Estados Unidos.
- Bates, B. (2004). Game Design. En B. Bates, Game Design (págs. 225-226). Boston Massachusett: Thomson Course Technology.
- Brown, N. (20 de 3 de 2012). EDGE. Recuperado el 13 de 5 de 2012, de EDGE: <http://www.edge-online.com/news/kongregate-triples-virtual-goods-revenue>
- Chu, E. (6 de 12 de 2011). Google Official Blog. Recuperado el 21 de 5 de 2012, de Google Official Blog: <http://googleblog.blogspot.com/2011/12/10-billion-android-market-downloads-and.html>
- Conejeros, V. (2012). Soul Avenger "Game Design Document". Santiago.
- Kenney, B. (2011). thetechlabs. Recuperado el 13 de 5 de 2012, de <http://www.thetechlabs.com/tech-news/its-time-to-take-mobile-gaming-seriously/>
- Kerr, C. (14 de 4 de 2012). GamrReview. Recuperado el 13 de 5 de 2012, de <http://www.gamrreview.com/article/88926/is-it-time-to-take-mobile-gaming-seriously>
- Lara, P. (7 de 7 de 2011). Apple. Recuperado el 21 de 5 de 2012, de Apple: <http://www.apple.com/es/pr/library/2011/07/07Apples-App-Store-Downloads-Top-15-Billion.html>
- Nvidia. (2004). Improve Batching Using Texture Atlases.
- O'Brien, M. (20 de 3 de 2012). ArenaNet. Recuperado el 13 de 5 de 2012, de ArenaNet: <http://www.arena.net/blog/mike-obrien-on-microtransactions-in-guild-wars-2>
- PayPal. (s.f.). Obtenido de https://www.paypalobjects.com/IntegrationCenter/ic_micropayments.html

- Rosenberg, J. (6 de 3 de 2012). Google Official Blog. Recuperado el 3 de 6 de 2012, de Google Official Blog: <http://googleblog.blogspot.com/2012/03/introducing-google-play-all-your.html>
- Select Business Solutions. (s.f.). Recuperado el 3 de 6 de 2012, de Select Business Solutions: <http://www.selectbs.com/analysis-and-design/what-is-the-waterfall-model>
- Unity3D. (s.f.). Recuperado el 3 de 6 de 2012, de Unity3D: <http://unity3d.com/create-games/>
- Will, D. (30 de 11 de 2011). Ezine Articles. Recuperado el 13 de 5 de 2012, de Ezine Articles: <http://ezinearticles.com/?iPhone-App-Developers-Harness-The-Potential-Of-Unity-Game-Engine&id=6725692>
- Wilson, G. (3 de 2 de 2006). Gamasutra. Recuperado el 3 de 6 de 2012, de Gamasutra: http://www.gamasutra.com/view/feature/2538/off_with_their_huds_rethinking_.php
- Wu, S. (20 de 6 de 2007). TechCrunch. Recuperado el 3 de 6 de 2012, de TechCrunch: <http://techcrunch.com/2007/06/20/virtual-goods-the-next-big-business-model/>

ANEXOS

- Anexo 1: Estados del videojuego

```
public enum GameStates{  
    Pause = 0  
    ,MainMenu  
    ,Cinematic  
    ,Town  
    ,InTownKeeper  
    ,InGame  
    ,WorldMap  
    ,CompleteLevelQuest  
    ,InTutorial  
    ,InTutorialTown  
    ,GameOver  
    ,Intro  
}
```

- Anexo 2: Función Update de la clase Game

// Update is called once per frame

```
void Update (){
```

```
    sortRenderQueueDepth(); //sort objects in the render depth queue
```

```
    Tutorial.evaluateTutorials(); //evaluate game tutorials
```

```
    switch(currentState){
```

```
        case GameStates.InGame:
```

```
            UpdateInGame();
```

```
        break;
```

```
        case GameStates.Intro:
```

```
            UpdateIntro();
```

```
        break;
```

```
        default:
```

```
            break;
```

```
    }
```

```
}
```

- Anexo 3: Función InGameUpdate de la clase Game

```
void UpdateInGame(){  
  
if(battlefieldIsLoaded && currentQuestInstance && !currentQuestIsCompleted)  
{  
    if(allowEnemySpawn)  
    {  
        tryToSpawnEnemies();  
    }  
    evaluateQuestComplete();  
}  
}
```

- Anexo 4: Funcionalidad de orientación de la clase Character

```

public enum FACING
{
    RIGHT = 0
    ,LEFT
};
private FACING _currentFacing = FACING.RIGHT;
public FACING currentFacing
{
    get {return _currentFacing;}

    set
    {
        //the facing direction is different than the current one
        if(_currentFacing != value)
        {
            //get the current actor's(?) scale
            Vector3 scale = transform.localScale;

            //scale the x axis if needed
            if( (value == FACING.LEFT && scale.x>0.0f) ||
                (value == FACING.RIGHT && scale.x<0.0f))
            {
                scale.x*=-1.0f;
                transform.localScale=scale;
            }
            _currentFacing = value;
        }
    }
}

```

- Anexo 5: Función tryToMove de la clase Character

```

public void tryToMove(Vector3 newFeetPos,bool keepInArena)
{
    Vector3 feetPos  = this.getFeet().position;
    Vector3 diff = newFeetPos - feetPos;
    Game game = Game.game;
    if(keepInArena)
    {
        bool currentCoordsPlayable = game.worldCoordInPlayableArea(feetPos);
        bool newCoordsPlayable = game.worldCoordInPlayableArea(newFeetPos);

        //if was in a valid position and the new position is not valid
        if(currentCoordsPlayable && !newCoordsPlayable)
        {
            float mag = diff.magnitude;
            float angle = Vector3.Angle(Vector3.right,diff);

            if(angle<45.0f){diff = Vector3.right*mag;}
            else if(angle<135.0f){return;}
            else if(angle<225.0f){diff = Vector3.left*mag;}
            else{return;}
        }
    }
    transform.position = this.transform.position + diff;
    FACING currentF = (diff.x>0)? FACING.RIGHT:FACING.LEFT;
    currentFacing = currentF;
}

```

- Anexo 6: Función InGameUpdate de la clase Hero

// Update is called once per frame

```
public override void InGameUpdate ()
```

```
{
```

```
    base.InGameUpdate();
```

```
    stats.health = Mathf.Min(stats.health, getMaxHealth());
```

```
    stats.magic = Mathf.Min(stats.magic, getMaxMagic());
```

```
    Game game = Game.game;
```

```
    Vector2 mousePos = new Vector2(Input.mousePosition.x);
```

```
    if(isAlive())
```

```
    {
```

```
        if(game.currentDialog==null && Time.timeScale>0 &&
```

```
            game.currentState != Game.GameStates.Pause &&
```

```
            game.currentState != Game.GameStates.InTutorial)
```

```
        {
```

```
            if(!usingSkill)
```

```
            {
```

```
                if(Input.GetMouseButtonDown(0))
```

```
                {
```

```
                    SoulAvenger.Character newTarget = pickTargetUsingMouse();
```

```
                    if(newTarget!=null && newTarget!=currentTarget)
```

```
                    { currentTarget = newTarget; }
```

```
                    else if(newTarget == null)
```

```
                    {
```

```
                        Vector2 mousePos = new Vector2(Input.mousePosition);
```

```
                        if(Game.game.screenPosInPlayableArea(mousePos))
```

```
                        {
```

```
                            if(currentAttack!=null) { currentAttack.onEndAttack(); }
```

```
                            currentTarget = null;
```

```
                        }
```

```
                            game.showEnemyTarget(false,null);
```

```
                            mouseFollowTarget = Game.game.screenPosToWorldPos(mousePos);
```

```
                        }
```

```
                    }
```

```
                }
```

```
            }
```

```
        if(currentTarget!=null && currentTarget.isAlive())
```



```

{
    followingMouse = false;
    game.showEnemyTarget(true,currentTarget);

    if(currentAttack == null)
    { currentAttack = pickAttackForTarget(currentTarget); }

    if(currentAttack!=null)
    { currentAttack.attackUpdate(); }
}
else if(followingMouse)
{
    if(moveToTarget(mouseFollowTarget,getSpeed()))
    { changeAnimation("run"); }
    else
    { changeAnimation("idle"); }
}
}
}
else if(!isDying && !isDead)
{
    changeAnimation("death",onDie);
    isDying = true;
}
}

```

- Anexo 7: Función InGameUpdate de la clase BasicEnemy

// Update is called once per frame

```
public override void InGameUpdate (){

    if(isAlive()){
        if(spawningComplete){
            if(Game.game.currentDialog==null)
            {
                Hero hero = Game.game.playableCharacter;
                if(hero.isAlive()){
                    if(currentTarget == null)
                    { currentTarget = hero; }}
                    else
                    {
                        currentTarget = null;
                        changeAnimation("idle");
                    }

                    if(currentTarget!=null) {
                        if(currentAttack == null)
                        { currentAttack = pickAttackForTarget(currentTarget); }

                        if(currentAttack!=null)
                        { currentAttack.attackUpdate(); }
                    }
                }
            }
        }
        else if(!isDying && !isDead)
        {
            getSprite().localTimeScale = 1.0f;
            changeAnimation("death",onDie);
            isDying = true;
        }
    }
}
```

- Anexo 8: Tipos de ítems del videojuego

```
public enum Type
{
    Consumable = 0
    ,Weapon
    ,Shield
    ,Armor
    ,Boot
    ,Ring
    ,QuestItem
};
```

- Anexo 9: Contenedor de ítems del videojuego

```
public Dictionary<Item,int>      itemList = new Dictionary<Item, int>();
```

- Anexo 10: Funciones de habilitación de ataques

```

public int getDPS(){
    tk2dAnimatedSprite sprite = character.getSprite();
    int animationIndex = sprite.anim.GetClipIdByName(attackAnimation);
    tk2dSpriteAnimationClip clip = sprite.anim.clips[animationIndex];
    int midDamage = (minDamage + maxDamage)>>1;
    float activationTime = (float)clip.frames.GetLength(0)/clip.fps;
    int DPS = (int)((float)midDamage/(activationTime + rechargeTime));
    return DPS;
}

public int getDPA(){
    tk2dAnimatedSprite sprite = character.getSprite();
    int animationIndex = sprite.anim.GetClipIdByName(attackAnimation);
    tk2dSpriteAnimationClip clip = sprite.anim.clips[animationIndex];
    int midDamage = (minDamage + maxDamage)>>1;
    float activationTime = (float)clip.frames.GetLength(0)/clip.fps;
    int DPA = (int)((float)midDamage/(activationTime));
    return DPA;
}

public virtual bool canUseAttack(){
    return rechargeTimer<=0 && attackEnabled;
}

```