La Ciudad del Zombi Comilón

Juan Díez García

Resumen— La ciudad del zombi comilón es un videojuego Arcade de dos dimensiones que crea una sensación de falso 3D. Para su creación se ha llevado a cabo una metodología Agile. Esto es debido a que para lograr la diversión en un juego se necesita bastante feed-back de jugadores. Todo el proyecto se ha realizado con el motor gráfico de UNITY, pero muchos de los recursos gráficos pertenecen al software RPG-MAKER. Las funcionalidades del juego se han realizado con C#, dividiéndolas en diferentes scripts dependiendo de su finalidad. Con estos algoritmos se incorpora inteligencia a los enemigos y al entorno, proporcionando una experiencia divertida. El producto consiste en una demostración jugable de un entorno interactivo que cumple con los objetivos presentados. Pero la finalidad de éste proyecto consiste en realizar una primera versión, que nos proporcione una buena experiencia en UNITY y en el desarrollo de un proyecto de tamaño considerable.

Palabras Clave— videojuego, falso 3d, Unity, C#, funcionalidades, jugable, inteligencia, experiencia divertida, feed-back, agile.

Abstract—La ciudad del zombie comilón is a project based on the development of a two-dimensional arcade video game creating a false 3D feel. In order to carry out this project, a hybrid methodology between Scrum and Agile has been used, this is because for a game to be fun, you need enough feedback from players. The entire project has been done with the graphic engine UNITY, but many of the graphic resources have been utilized through the software RPG-MAKER. The functionalities of the game have been made with C # and divided into different scripts depending on their purpose, thus giving intelligence to enemies and the environment, providing a fun experience. The final product of this project consists of a first playable version of an interactive environment that meets the expectations for this version. The purpose of this project is to make a first version of a product, which will provide us with our first experience in UNITY and in the development of a project of a substantial size

Index Terms— videogame, false 3d, Unity, C#, functionalities, playable, intelligence, a fun expericence, feed-back, agile.

1 Introducción

La idea inicial de este trabajo, consiste en la elaboración de un juego utilizando UNITY. Pero lo destacable no es la jugabilidad de este mismo, sino la I.A que incorpore el jugador, los enemigos y el entorno.
Éstos tendrán diferentes funcionalidades que variarán dependiendo del estilo de juego del usuario y de las acciones que tome en su partida.

El juego consistirá en un zombi que camina por la ciudad devorando todo lo que se encuentre en su camino, pero solo puntuarán aquellas que sean seres vivos. Mientras el zombi no ingiera personas, la vida bajará en base al tiempo. Cuando el tiempo avanza, la cantidad de objetos en su entorno podría variar y la "degradación" de su vida aumentar. De esta manera se aplica una variable de dificultad al usuario, que busca una supervivencia.

Inicialmente existirán diferentes enemigos base, pudiendo aumentar en versiones posteriores. Los principales son cuatro: policías, civiles, coches y un jefe que aparece en ciertas circunstancias. Dependiendo de una dificultad estimada por cada enemigo, éstos aportarán una puntuación.

La finalización de la partida se puede deber a dos causas; la primera viene definida por el arrollamiento del protagonista por un vehículo, y la segunda porque la vida llegue a 0. Cualquiera de estas dos razones acabará en un fin de juego, donde sí se ha conseguido una de las mejores puntuaciones, podrá ser insertado en una tabla de mejores puntuaciones.

1.1 Motivación

La idea de crear un juego ha sido una de las metas más importantes desde que decidí cursar esta carrera. De pequeño siempre me han gustado los videojuegos y he imaginado crearlos.

Con la cantidad de trabajo que he tenido estos años nunca he podido realizar un juego como deseaba. Pero al presentarse la oportunidad en el TFG no la podía dejar escapar.

El desarrollo de este juego representa una gran oportunidad en realizar más aplicativos. Los primeros pasos en una tecnología desconocida, son los más complicados, pero después puedes crear todo lo que imagines.

1.2 Objetivos

El objetivo principal de este proyecto es la realización de una Alpha jugable en Android así como aplicar los conocimientos obtenidos sobre algoritmos y resolución de problemas. Tras un previo estudio del estado arte se tomó la decisión de utilizar metodología completamente Agile al

necesitar un continuo feed-back para asegurar el balance del juego. Finalmente para asegurar un correcto desarrollo se estimó cumplir como mínimo los siguientes requisitos:

- Completar toda la instrucción de UNITY.
- Diseñar las funcionalidades del juego.
- Crear los escenarios.
- Conseguir los recursos básicos para poder desarrollar.
- Implementar la gestión de la movilidad del protagonista.
- Realizar una correcta gestión de la cámara entre los escenarios.
- Insertar enemigos y sus funcionalidades.
- Completar funciones de choque y SPAWN.
- Insertar interfaz del usuario.
- Creación de una pantalla de inicio.
- Implementar la gestión de la vida.
- Insertar derrota, puntuaciones.
- Insertar IAS a los enemigos.
- Insertar heurísticas de dificultad.
- Creación de Boses finales.
- Creación de IA de los enemigos y entorno basada en aprendizaje.

Como su prioridad, el proyecto no tiene como objetivo salir al mercado a competir ni conseguir enormes ingresos con los usuarios. Únicamente trata de cubrir los aspectos básicos de la creación dándonos nuestro primer contacto con las tecnologías y las limitaciones que representan.

Igualmente no se descarta una futura comercialización al perfilar las funcionalidades como se necesitan.

1.3 Estado del Arte

El mercado de los videojuegos, es uno de los más rentables en la actualidad. En una época de oro de los smartphones y tablets, no solo juegan aquellos con dispositivos especializados sino todos los propietarios de un teléfono. Por lo tanto multitud de empresas invierten cada vez más recursos, para desarrollar mejores juegos, que aporten mayor diversión y altos ingresos.

Desde que comenzó este suceso, muchos juegos han tenido mucho éxito: el Candy crush, el clash of clans o el Angry birds. Todos ellos tenían un factor novedoso que llamó la atención de un gran número de jugadores de un público que nunca antes había participado en esta industria. Por lo tanto el impacto actual que tienen los videojuegos, es enorme, porque abarca desde usuarios de videoconsolas, pasando por los usuarios de ordenadores y acabando en todas las personas que poseen un dispositivo electrónico. Esta gran demanda de juegos ha concluido en un gran número de desarrolladores buscando satisfacer los gustos de los usuarios, creando un gran mercado con una inmensa variedad de conceptos.

Desgraciadamente, como el número de creadores aumenta por momentos, cada vez es más difícil llevar a cabo una idea triunfadora que destaque ante las demás

Existen una gran variedad de formas de obtener ingresos: incorporación de banners publicitarios, la venta información de los usuarios, gestionar un sistema micropagos, añadir videos promocionales o imponer un precio al descargar el juego. Actualmente la opción más lucrativa son los micropagos, donde los usuarios comiencen a jugar gratuitamente, pero si quieren avanzar más rápido u obtener algunas mejoras paguen dinero real. Definitivamente no todas las personas que juegan acaban pagando, pero aquellas que se afilian pueden llegar a invertir grandes cantidades. Un ejemplo es Clash of clans que llegó a ingresar 1345 millones de dólares en 2015.

1.3.1 Motores gráficos.

En la actualidad, existen un gran número de motores gráficos como por ejemplo: Unreal Engine, GameMaker, Unity o Blender. El principal objetivo de un motor gráfico es de proporcionar un motor para renderizar gráficos, detector de colisiones, escenario gráfico, animaciones o una correcta gestión de memoria.

La elección de UNITY ha sido por su variedad de assets y su calidad de documentación. Además dispone de una tienda de recursos tanto a nivel de software como gráficos. Por último el factor de decisión final ha sido su adaptación a casi cualquier sistema y la posibilidad de desarrollar software en multitud de plataformas.

1.4 Diseño.

1.4.1 Público objetivo.

El foco principal del juego, es un público casual mayor de 15 años que busque entretenerse en un rato. El objetivo de éste software no es obligar al jugador a invertir gran cantidad de horas, sino en entretener cortos periodos de tiempo. Estos perfiles habitualmente no suelen pagar por juegos a no ser que tengan una gran calidad gráfica. Por esta razón si queremos llegar al mayor público posible, la mejor opción es que sea gratuito.

El juego del zombi comilón no tiene un sistema de monetización integrado. La clave sería insertar un sistema de monedas y una colección de aspectos que por completarla debas pagar o ver videos promocionales.

1.4.2 Satisfacción del usuario.

Durante todo el desarrollo se han ido haciendo preguntas a los usuarios para asegurar el acierto en la dirección a seguir. Como los recursos disponibles no nos permiten realizar una gran obra artística, las preguntas siempre iban relacionadas a la experiencia de juego.

Ejemplos:

- Facilidad de control.
- Dificultad.
- Entretenimiento aportado.
- Si realmente se cree que representa un reto.

1.5 Metodología de trabajo.

Todo el software ha sido desarrollado utilizando C#, una de los posibles sistemas de programación UNITY, junto a JavaScript. La selección de C# ha sido por comodidad y estabilidad. A nivel práctico, la diferencia, no es relevante pero C# posee más documentación porque suele ser la elección de la mayoría de desarrolladores.

2 MECÁNICAS DE JUEGO.

2.1 Escena

La ambientación del entorno consiste en una estética cuotidiana de colores vivos que denoten tranquilidad. Todo el tiempo de juego transcurre durante el día y los enemigos actúan con normalidad hasta vernos. El tipo de música no está decidido, por la repercusión al rendimiento del juego. Actualmente solo se puede entrar en el interior de un hotel, pero en versiones revisadas se debería poder entrar en cualquier posible lugar como: cafeterías, fábricas, recintos comerciales…etc.

2.2 Habilidades

El protagonista como tal no tiene ninguna habilidad innata. Al principio se necesitaba atacar para poder comer a la gente, pero las opiniones obtenidas decían que era un control complicado y matar solo impactando era mejor.

Figura 1: Captura de pantalla donde se puede ver la posición de ataque del zombi.

Una de las cuestiones a comentar son las habilidades que otorga la Diablesa. Una vez por nivel si el zombi interactúa con ella y le entrega una droga aleatoria con efectos en la jugabilidad.

La primera consiste en un incremento de velocidad durante 20 segundos y la segunda en un estado de inmunidad al daño (temporal y enemigo) durante 15 segundos. La obtención de cualquiera de estos poderes se ve reflejado con un icono en el interfaz.

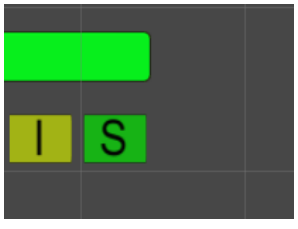

Figura 2: Captura de pantalla de la activación de los poderes en la interfaz S (velocidad) e I (inmunidad).

2.3 Controles

Unos de los principales cambios que se han realizado para el traslado a Android, ha sido la inserción de un sistema de controles adaptado para la plataforma. Después de varios intentos con diferentes tipos de controles, se decidió como mejor control un joystick de pantalla por su relativa comodidad.

La primera opción fue que el zombi se moviese siempre hacia una dirección pero presionando botones de dirección se pudiese rectificar. Después de algunas opiniones era bastante incómodo y difícil de controlar. Por tanto la experiencia de juego era bastante mala y fue descartado.

Como segunda versión se intentó que el zombi se dirigiese hacia el lugar tocado en la pantalla, de manera que localizaba el punto y tomaba la velocidad él. Las impresiones de jugadores, fueron mejores, pero no acabó de gustar porque se ocultaba parte del mapa. Además decían que era bastante complicado rectificar el movimiento y también fue descartada.

Después de tantos intentos fallidos se integró la opción más conservadora. Evidentemente los usuarios están más adaptados al uso de joysticks y botones por que la aceptación fue buena. Pese a todo esto las diferentes resoluciones de pantalla son un auténtico dolor de cabeza y el ajuste de un valor standard, sin realizar un escalado, fue bastante complejo.

Figura 3: Imagen de la interfaz de control en Android

2.4 Victoria/Derrota

La condición de victoria en este tipo de juego es relativa porque se considerará cuando consigamos mejor puntuación que nuestra partida anterior.
En cuanto la derrota, se define por la muerte del zombi y la posibilidad de iniciar otra partida.

Figura 4: Pantalla de títulos

3 DESARROLLO

3.1 Mapeado.

La estética seleccionada seria "retro" representando una ciudad común. Como el proyecto es en UNITY, si consiguiésemos otros SPRITES la modificación de la estética no sería muy costosa

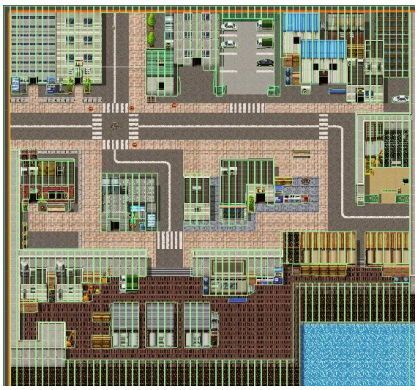

Figura 5: Mapa principal con el editor de colisiones.

3.1.1 Creación del mapa

El mapa ha sido creado utilizando un recurso comprado (RPG MAKER). Éste programa ha proporcionado un buen surtido recursos que facilitan enormemente el diseño del juego. Pero como punto más importante logramos una buena cohesión en los entornos.

RPG MAKER solo nos proporciona una imagen final de los mapas, pero para lograrla tenemos que ir posicionando los PNG en los lugares adecuados.

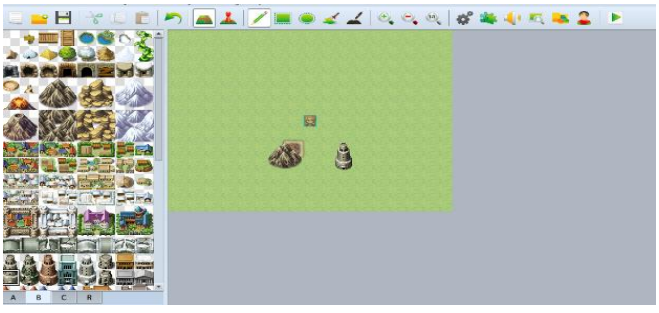

Figura 6: Captura de pantalla del editor RPGMAKER

El segundo paso es trasladar ésta imagen a TILED, programa que nos permite añadir las colisiones. Como se puede ver en la figura 3, los rectángulos verdes son los puntos definidos por TILED exportados posteriormente a UNITY.

3.2 Las animaciones.

Las animaciones en UNITY son máquinas de estados simples. Cada movimiento tiene asignado un SPRITE o una combinación de ellos llamada animación. Todos los personajes en el juego tienen la misma maquina con los estados: moviéndose, quieto y muerto.

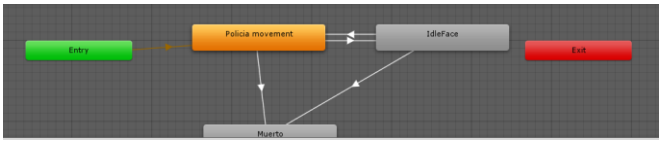

Figura 7: captura de pantalla de la máquina de estados de las animaciones.

Para determinar qué animación utilizar en cada momento, usamos las velocidades de los objetos en el espacio, creando un eje de coordenadas. Como se puede ver en la figura 7, mientras se está moviendo (que el boleando Moving está activo) se queda en el árbol de coordenadas, mientras que si pierde la velocidad pasa al estado "quieto". Para conocer qué imagen tomar al quedarse quieto, necesitamos dos variables donde salvamos los movimientos anteriores.

Por último está el estado "Muerto" que activa la animación del cadáver y desactiva la colisión con el objeto. Se ha decidido crear una animación para que el objeto no desaparezca sin más.

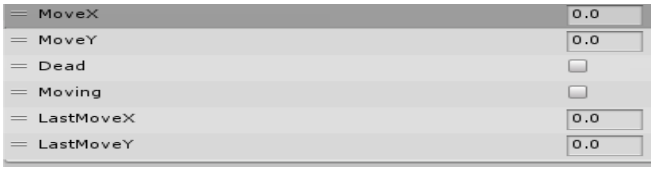

Figura 8: Variables utilizadas por las transiciones de animación.

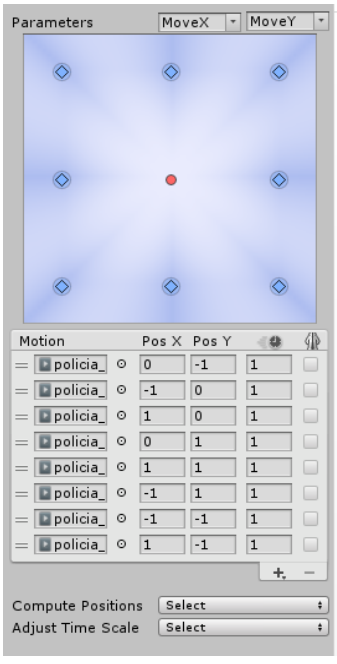

Figura 9: Eje de coordenadas que definen las animaciones con respecto a la velocidad del objeto.

3.3 Jugador.

La entidad del jugador está definida por el cuerpo principal y una espada asignada. Ésta es la denominación del área de ataque del jugador.
La construcción del jugador se ha definido de esta manera

porque necesitamos dos áreas de colisión. Una primera para controlar el daño recibido y otra para gestionar el daño a los enemigos.

Figura 10: Entidad jugador moviéndose negativamente en el eje de las Y.

Como se puede apreciar en la figura 10 el jugador posee dos colisiones marcadas en verde, la más pequeña es el punto de impacto que gestiona el controlador de vida y la mayor controla el daño que hará a sus enemigos. Por cada vez que un enemigo entre en el área de impacto recibirá un daño de 20, para realizar daño nuevamente el jugador deberá alejarse y volver a impactar. De la misma manera cuando el jugador recibe daño solo es la primera colisión.

3.4 Enemigos

3.4.1 Civiles.

La función de estos personajes es relativamente simple pero es la base de todo el juego. Su objetivo principal consiste en andar por la ciudad y ser comidos por el zombi para proporcionar vida.
El control de generación de estos personajes viene dado por el sistema de aparición que se describe en el punto 3.5.3. Cada vez que uno de estos personajes muere pasa al estado de muerto, desactiva su colisión y finalmente notifica al sistema de aparición su estado.

La IA consiste en una máquina de tres estados donde puede caminar, quedarse quieto o estar muerto. El movimiento viene determinado por un número aleatorio que otorga dirección y un factor de velocidad que viene predeterminado.
Anteriormente estos personajes podían ser asesinados por los vehículos que pasaban por la carretera, pero después de implementarlo se decidió que fuesen empujados para no afectar al sistema de aparición y al entorno porque la cantidad de atropellos era exagerada.

Finalmente comentar que se han probado diferentes IAS que escapaban del zombi o no se chocaban con cosas, pero esto hacía al jugo lento y tedioso. Cuando al jugador le resultaba complicado matar las victimas era bastante aburrido y poco gratificante. Seguramente podríamos calificar este tipo de personajes como "paja" la cual puede ser simple pero a la vez imprescindible.

Figura 11: Modelos de ciudadano.

3.4.2 Policías.

Los policías son los enemigos por excelencia de éste juego. Su objetivo consiste en liquidar al zombi que causa el terror en las calles de la ciudad. Comúnmente van patrullando en trayectorias horizontales o verticales, pero cuando ven al zombi se paran y comienzan a dispararle para liquidarlo.
Los comportamientos del policía se modifican dependiendo de los datos que le proporcionan algunos recursos de la entidad.

Figura 12: Modelos de policía.

El primer recurso es la visión que proporciona un área de colisión definida como triguer. Al entrar el personaje en esta área, el policía deja de caminar y calcula la posición a la que se encuentra el zombi. Una vez determinada la posición se activa la animación para que mire hacia la dirección de su enemigo. Esta funcionalidad se ha de incorporar en el bucle porque necesitamos que en cada momento revise la posición del punto de observación. Cuando el zombi sale de su área de visión vuelve a iniciar su patrulla original esperando volver a verlo nuevamente.

Figura 12: Área de visión del policía.

La gestión de diferentes colisiones dentro de un objeto no es posible, por lo tanto para poder definir un cuerpo donde el zombi pueda atacar, creamos otra entidad interior. Esta entidad de colisión será la que tendrá un "tag" de enemigo para que al entrar en contacto con la colisión, se aplique el daño al controlador.

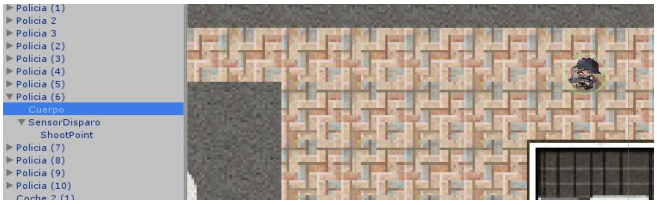

Figura 13: área de colisión del cuerpo del policía.

El sistema de disparo viene definido aislado del sensor principal porque en el futuro puede ser que necesitemos implementar el disparo en otros enemigos.
El disparo se basa en un sensor que detecta la posición exacta del zombi. Una vez conocida la posición exacta se crea una entidad a la que se aplica una velocidad y dirección.

Formula: myRigidBody.velocity = (jugador.transform.position - transform.position).normalized * velocidad;

Concluyendo, una vez hemos creado el objeto que impactará al objetivo, solo necesitamos decidir una cadencia de disparo (veces que se llamará a la función) y una distancia de lanzamiento (tiempo que tardará en destruirse la entidad bala). Si no se definen estos dos factores el rendimiento del juego se ve afectado enormemente causando una gran ineficiencia con una caída de FPS bastante considerable.

Figura 13: Sensor de disparo del policía.

3.4.3 Vehículos

EL funcionamiento de los vehículos consiste en una trayectoria definida por carreteras que pueden ser transitadas. Éstas están definidas por entidades de colisión que al detectarlas invierte su movimiento.

Esta parte ha costado bastantes recursos porque no hay nada parecido diseñado. Anteriormente se había realizado un sistema de transito bastante complejo en el cual se generaban vehículos que seguían la carretera mediante puntos de colisión. Mediante un sensor buscaban continuamente pasos de cebra y si encontraban un ciudadano en uno se detenían.

Las IAS de los vehículos hacía que la fluidez del tráfico sea muy escasa y el juego perdía bastante. A su misma vez se podían aumentar el tránsito de la misma trayectoria pero una circulación continua de vehículos era injugable.

Finalmente aunque se puede llegar a pensar que la IA implementada es pobre, es la que necesita el juego, porque al ver las trayectorias se pueden predecir los movimientos pero también representan una amenaza.

Figura 14: área de desplazamiento del vehículo.

3.4.4 Enemigo Jefe.

La idea de realizar un jefe final fue para solventar la problemática quedarse permanentemente en los puntos de aparición. Esta herramienta posee una IA muy similar a la del policía pero con algunas modificaciones.

Las diferencias del Boss con el policía son:

- Las animaciones
- La cadencia de disparo
- La rectificación del ángulo de tiro dependiendo de los movimientos del jugador.
- Aumento de vida y aguante.
- No es destruido de un golpe, sino que se implementa una gestión de daño.
- Es temporal, por lo tanto al pasar el tiempo de abuso se elimina.

La rectificación del ángulo de tiro se comienza a notar cuando el personaje realiza numerosos movimientos. Como son entidades temporales y su función es expulsar al jugador rápidamente esta habilidad no resulta muy útil a no ser que el usuario decida atacarlo.

Figura 15: Modelos del jefe final.

3.5 Elementos Funcionales.

3.5.1 Gestión de vida.

La gestión de la vida en el juego puede ser separada en diferentes situaciones. La primera situación viene cuando el zombi recibe daño, la segunda cuando por tiempo se reduce y la tercera sucede cuando se come a alguna persona. Todas estas acciones se visualizarán en la interfaz, donde gráficamente podremos observar las variaciones en la vida del jugador.

Al recibir un impacto de un enemigo, la vida será reducida en función del daño que se asigne a cada individuo. Tener una gestión de estas características proporciona un abanico de posibilidades para actualizaciones futuras.
El jugador posee una vida máxima y una vida actual, pero al impactar o una acción enemiga la vida actual disminuye según el daño que queramos asignar. Los daños se asignan por cada bala a 100 pero si el coche nos atropella se pierde toda la vida en el instante. Se ha diseñado de esta manera para que los impactos de balas hagan un daño considerable pero asumible y por la contra los vehículos estés obligado a esquivarlos.

El transcurso del tiempo es uno de los factores a tener en cuenta, porque si no devoras individuos, la vida, se reducirá en función del nivel actual y una tasa standard de reducción. Normalmente se pierde 20 de vida cada 10 segundos, pero si el nivel aumenta el tiempo se divide entre la mitad del nivel actual y la vida se reduce restando el valor asignado por el nivel actual. Estas dos heurísticas se han asignado a base de diferentes pruebas de balance y consejos de diferentes usuarios.

El último factor a tener en cuenta son los ciudadanos que va ingiriendo el zombi durante la partida. Cada ciudadano otorga 40 por el nivel actual del jugador, esta heurística balancea la perdida de vida que hace crece con la dificultad. Además si se consigue matar a un jefe o a un policía el factor standard de vida es diferente, porque si comparamos la perdida de vida potencial del riesgo, se ha de compensar.

3.5.2 Interfaz.

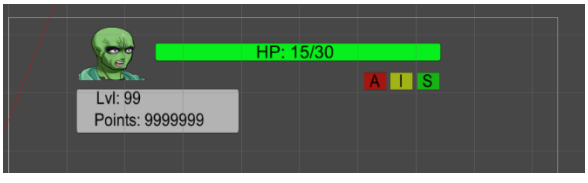

Figura 16: pantallazo de la interfaz.

Como se puede observar en la figura 16 la interfaz de usuario está definida por diferentes elementos:

- La puntuación.
- El nivel del jugador
- Imagen del protagonista
- Indicadores de penalización o de poder.
- La vida del jugador.

Éste tipo de interfaces se integran en un canvas, un elemento que se superpone al mapa y se puede ver en cualquier momento y situación. Se hace así para ser acoplado a la cámara y con el movimiento siempre sea visible.
El mismo tipo de elementos se han utilizado para el menú de inicio de partida donde puedes salir del juego o jugar una nueva partida.

3.5.3 Sistema de Aparición.

La aparición de enemigos es la IA más compleja de nuestro proyecto, se debe al aprendizaje del estilo de juego del jugador y lo penaliza si cree que es incorrecto. Además el sistema de aparición varía dependiendo del mapa en que esté, adaptándose a las diferentes situaciones que queramos desarrollar en la experiencia juego. Un ejemplo es que en las ciudades los sistemas de aparición funcionan por áreas mientras que en los interiores, como el hotel, simplemente existen respawns.

Figura 17: imagen de la entidad sistema de aparición.

Las áreas son unos de los puntos más importantes en el sistema, cada área tiene un mecanismo que controla si el jugador se encuentra en la misma y poder asignar las muertes al sector. También se necesita hacer una cuenta de los NPCS que pertenecen a esa zona para organizar la heurística posteriormente. Por cada sector existen dos puntos de aparición que se intercalan equitativamente, de ésta manera se equiparan los NPC.

Figura 18: imagen de las áreas de respawn posicionadas.

Cada dos segundos un algoritmo selecciona el área de menos personajes y selecciona aleatoriamente un modelo de NPC. Como los NPCS están prediseñados con un prefav, simplemente invocamos el modelo en la posición decidida.

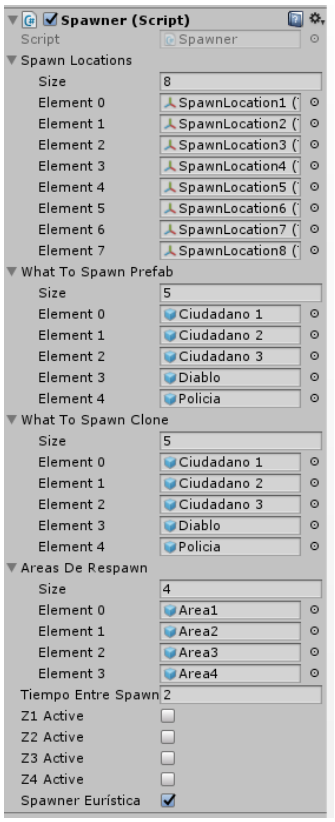

Figura 19: variables de la entidad del sistema de aparición

Un sistema de aparición que se asigna donde hay menos personajes tiene una imperfección, en la que si transitas todo el rato la misma posición puedes comer sin moverte fácilmente. Éste problema se soluciona con el sistema de penalización de heurística donde detectamos si el jugador está realizando un abuso. Si se está realizando un abuso ((NPCS asignados al área – NPCS asesinados) < NPCS asignados al área*0.3) aparece un recuadro rojo con una y se invoca un jefe temporal (20sec).Como es difícil de matar obliga a el jugador a moverse del área, y además bloquea la aparición de NPCS durante 10 segundos. No siendo suficiente el jugador pierde el doble de vida durante el tiempo de penalización. Estas medidas obligan al usuario a prestar atención del abuso de cada zona y a moverse por los diferentes mapas y lugares.

3.5.4 Gestión de escenarios.

Los diferentes mapas en UNITY se definen como escenas y hemos de gestionar su transición utilizando scripts. Por lo tanto debemos definir qué elementos no deben destruirse en la transición como el jugador, el canvas o las estadísticas.

La mayor complicación se centra en una correcta gestión de escenarios en que el personaje aparezca en el punto por el que entre anteriormente. Si no se controlaran los puntos de aparición y cruzamos una puerta al volver apareceríamos en un lugar diferente cosa bastante incoherente. La solución fue crear dos puntos de colisión en cada transición, uno donde volver y el gatillo que activa la transición.

3.5.5 Diablesa y sistema de bonificaciones.

El sistema de niveles es bastante útil porque los usuarios son conscientes del aumento de dificultad en el transcurso de la partida. Normalmente al subir un nivel que requiere un esfuerzo y se inserta una mejora u obsequio para premiar al usuario. Después de pensar bastante decidí bajo encuesta agregar diferentes poderes temporales, que se entregan al jugador, hablando con la diablesa.

Durante el desarrollo se han probado diferentes métodos como una historia repartida en cartas por la ciudad. Desgraciadamente en las fases de test se observó que no era una mejora en la experiencia de juego. La mayoría de usuarios no buscaban las cartas, el tiempo es muy valioso y preferían realizar otras acciones.

Los poderes temporales no han sido una opción fácil de decidir porque afecta directamente al balance del juego. Primero se intentó entregar las bonificaciones automáticamente al subir solamente de nivel, pero no era bastante gratificante. Unos días más tarde se me ocurrió que la diablesa podría entregar algún tipo de droga que al subir de nivel te la regale como recompensa. En las test fue bastante bien aceptado, porque al conseguir un nuevo nivel todos los usuarios se dirigían a hablar con la diablesa sin perder tiempo.

Figura 22: captura de pantalla en el momento de la entrega de poderes.

3.6 Testing y modificaciones.

El correcto desarrollo de un proyecto requiere de opiniones para modificar los requisitos funcionales dependiendo de la experiencia del usuario. Como diseñadores podemos pensar multitud de eventos, pero hasta que no se prueban en conjunto, no podemos decidir si son adecuados para el conjunto general. Si queremos lograr que el juego sea divertido necesitamos opiniones de cada funcionalidad.

El equipo de test que ha intervenido consiste en 7 usuarios de edades diferentes entre 23 a 27. Como casualmente es nuestro público objetivo nos viene perfecta para captar sus opiniones. Cuando un prototipo de funcionalidad estaba implementado se entregaba a testear y posteriormente recolectaba las opiniones al respecto. Éste hecho ha sido el más determinante para el diseño porque desde la idea final

hasta la final han cambiado muchas cosas.
El mayor número de cambios han sido ajustes de balance del juego como: vida del personaje, daño de los enemigos, vida que pierdes, velocidad de los elementos…etc. Pero también ha habido cambios significativos como el las IAS de los vehículos.
Como he comentado anteriormente las primeras generaban vehículos y si pasabas por el paso de cebra el vehículo se detenía. Como el flujo de vehículos era muy escaso o muy frecuente este modelo fue inservible y se insertaron tramos de recorrido entidad.

Otra de las funciones que se han adaptado a los usuarios son las cantidades de poderes que recibe el personaje. El diseño inicial consistía en 2 poderes adicionales, uno que paraba el tiempo y otro que te dejaba comer vehículos. Como los poderes son seleccionados aleatoriamente por la diablesa, un número excesivo confundía a los jugadores y bajaba las posibilidades que les tocara el más útil.

Finalmente hablar del Jefe enemigo que posee una corrección de la trayectoria dependiendo del estilo de juego del usuario. Anteriormente también disparaba misiles que seguían al jugador hasta que impactara contra otro objeto, pero la dificultad era demasiado grande. Casi todos los usuarios notificaron que cuando salía el jefe era todo muy engorroso. Por estas mismas razones el ajuste de la trayectoria está desactivado de manera que proporciona mayor diversión.

4 ANÁLISIS DE RESULTADOS.

4.1 Dificultades encontradas.

La aparición de errores, como en cualquier proyecto que dependa de programación, ha sido bastante abundante. Los errores más preocupantes han sido aquellos que afectaban al rendimiento del dispositivo. Como es evidente al ser un juego simple era imprescindible que fuese fluido, sino las valoraciones de los usuarios serian negativas. Al ser bastante numerosos, solo nombraré aquellos errores que han sido complicados de solventar y han planteado un reto al desarrollo.

El primer error importante apareció con la integración del sistema de aparición de NPCS. El incremento de elementos afectó muy negativamente al rendimiento del juego, donde las imágenes por segundo bajaron de 60-80 hasta 10. Después de investigar descubrí que el problema estaba en la llamada de animaciones, la cual realizaba dentro de los bucles. Al momento comenzó una de las refactorizaciones más grandes del proyecto, afectando a casi todos los scripts funcionales.

La solución se basó en realizar llamadas recursivas cada cierto tiempo a las acciones de movimiento o IAS. Las únicas funciones que se quedaron dentro de los bucles fueron aquellas que necesitaban de una respuesta rápida por la interacción con el jugador.

Figura 23: imagen de la caída de frames y su rectificación.

El segundo gran problema, que incluye diferentes errores, apareció en la creación de los policías y su sistema de disparo. La cadencia de disparo necesitaba ser asegurada a ciertos intervalos porque si no creaba entidades en cada imagen posible. El resultado era un efecto manguera que colapsaba todo el juego. Además se tuvo que mover el cálculo de posición fuera de bucles y asignar temporalidad de vida a las entidades, porque si no las balas seguían al jugador hasta impactarle.

Como último dato curioso, comentar que se necesitó regular la IA para que cuando el policía muriese dejase de disparar. Si esto no se realizaba el muerto continuaba disparando e igual con los ciudadanos, que todavía muertos continuaban su paseo por la ciudad.

Figura 24: disparo sin cadencia asignada.

4.2 Estado del Proyecto.

En la actualidad el proyecto se encuentra en un estado bastante avanzado pero no lo suficiente para comercializarlo. Para lograr que un juego sea divertido se necesita muchísimo feed-back que por cuestiones de tiempo ha sido imposible llevar a cabo totalmente. Solamente queda insertar la música y las tablas de puntuaciones de google play. Estas últimas no se han implementado porque necesitaba pagar 25 euros de la licencia de desarrollador y me ha parecido caro solo por el uso de un Asset.

Podríamos definir el proyecto, como un juego funcional en estado de preventa como un alpha. Lo único que necesitaríamos subirlo a un servidor de test y esperar feed-back de jugadores para posibles mejoras.

4.3 Conclusiones

Durante el proyecto se han realizado numerosos cambios en la planificación. Muchas veces el desarrollo había partes más ágiles en fases tempranas y otras cosas que se podían dejar de lado. El ejemplo más claro son las pantallas de inicio y los enemigos jefe, al ser secundarios se podían dejar para el final. Por otra parte las funcionalidades básicas del juego era necesario priorizarlas, y comenzar a trabajar en las IAS lo antes posible.

Como punto a destacar, decir que las fases de aprendizaje de la tecnología y diseño consumen gran parte del tiempo, cosa que no se había planificado. Supuestamente la escenificación de los recursos iba a ser rápida, pero lamentablemente crear una cohesión apropiada ha sido bastante costosa.

El factor detonante para la reducción del tiempo en el diseño fue la adquisición de los recursos proporcionados por el RPG MAKER. Nos proporcionó casi todo lo gráficamente necesario para realizar el juego en su totalidad.

Por lo tanto podríamos afirmar que nuestra planificación era bastante idílica, surrealista y fuera de los datos reales. Todo esto viene porque era imposible cuantificar certeramente los tiempos para tareas que desconocíamos.

4.3.1 Conocimiento adquirido

Todos estos meses de desarrollo me han proporcionado numerosos beneficios. El primero y más destacable es un conocimiento del motor gráfico UNITY bastante considerable. Como segundo punto he de nombrar el poder planificar con relativa corrección un proyecto de un tamaño considerable, sin ceder ante la presión.

Pero el verdadero conocimiento ha sido el aprender donde buscar las cosas que necesitaba. Muchas veces tenía problemas, que si no llego a encontrar referencias por algunos foros, no habría conseguido solventar.

4.3.2 Futuros desarrollos

Además de los sonidos y las tablas de puntuaciones como he nombrado anteriormente, se debería implementar un sistema de SKINS que el jugador pueda coleccionar. Es más, si se pudiese disponer de un diseñador gráfico la experiencia de juego mejoraría enormemente porque los recursos visuales han sido el gran reto de todo el desarrollo. Finalmente se deberían añadir todos los mapas a los interiores de los edificios y acabar el diseño de la ciudad completa. Con todo esto, un buen feed-back y un testeo podría nacer un juego muy divertido.

4.3.3 Planificación.

El trabajo está basado en una metodología AGILE planificada, en la cual se asignan prioridades de proyecto a cada dos semanas. Cuando una tarea no se puede realizar u ocupa demasiados recursos, se retrasa para el sprint siguiente.

En un principio se había pensado realizar SPRINTS semanales, pero a la práctica ha sido imposible. Al compaginarse con la vida cotidiana, ciertas funcionalidades se retrasaban y finalmente se desarrollaban en un tiempo más extenso.

Para concluir comentar que las metodologías agiles suelen tener ciclos cortos para diseñar funcionalidad a funcionalidad. Durante el proyecto se han extendido lo ciclos porque no se ha podido trabajar en los tiempos planeados. Pero el desarrollo en general cumple con los valores de AGILE, donde realizamos tareas cortas y el usuario las evalúa. Prueba de esto son los grandes cambios que han sucedido en los requisitos iniciales donde el juego final es bastante diferente al pensado.

5 Agradecimientos

En primer lugar me gustaría agradecer a mi tutor Alejandro Párraga por su gran comprensión y calma a la hora de guiar el proyecto. Cada vez que iba a una reunión en un estado de nervios bastante desmesurado, salía de la reunión pensando que iba bien y que simplemente siguiese adelante.

En segundo lugar me gustaría agradecer a mis amigos y familiares que me han ayudado a testear las funcionalidades de este juego. Sin sus comentarios, todo el desarrollo habría sido mucho menos fructífero.

6 Bibliografía

[1] UNITY, lugar de consulta de la api y de scripts funcionales: https://goo.gl/NdfXI1 Última consulta: 25/06/2017

[2] Metodología, punto de información sobre agile: https://goo.gl/rSu8U6 Última consulta: 10/05/2017

[3] Gamification, curso informativo: https://goo.gl/Z0FVsO Última consulta: 9/06/2017

[4] Unreal Engine, web informativa sobre el motor para realizar una comparativa: https://goo.gl/cHCCUa Última Consulta: 5/03/2017

[5] RPG MAKER, web informativa del funcionamiento del aplicativo: https://goo.gl/lWepOr Última Consulta: 14/06/2017

[6] RPG GAME, punto de información sobre juegos RPG: https://goo.gl/ZFvRVj Última Consulta: 20/02/2017

[7] GAME MAKER, , web informativa sobre el motor para realizar una comparativa: https://goo.gl/29zDOw Última Consulta: 20/02/2017

[8] SPRITES and RESOURCES, tienda de Unity: https://goo.gl/6gLftS Última Consulta: 14/06/2017

[9] GAME OBJECTS, ejemplos y funcionalidades de objetos: https://goo.gl/9EAdcF Última Consulta: 14/06/2017

APÉNDICE A

GLOSARIO:

Agile: Método de gestión ágil de proyectos o Agile Project Manegement es un conjunto de metodologías agiles para el desarrollo de proyectos que precisan una especial rapidez y flexibilidad en su proceso.
SPRITES: Imagen utilizada para dar aspecto a un objeto en el juego.
NPCS: Entidades del juego que no son el jugador pero interactúan con él.
FPS: Frames por segundo o imágenes por segundo.
Frame: Imagen de un conjunto que se muestran para visualizar movimiento.
Scripts: Programas creados para funcionalidades determinadas.
Colisiones / colliders: Entidades que sirven de referencia en el juego para saber si un objeto choca con otro.
Spawn: Sistema de aparición de objetos.
TILED: Programa de creación de mapas.
RPG MAKER: Software de creación de juegos RPG.
SKINS: Aspectos intercambiables en un juego.
Diablesa: NPC que otorga los poderes al zombi.
Zombi: Personaje utilizado por el jugador.
Prefav: Entidad creada con unas características standard.
Standard: Se dice de algo común.
Entidad: Cualquier objeto evaluado por el motor gráfico.
Test Fase de pruebas.
Bug: Error en el juego.
IA: Inteligencia artificial.
Interfaz: Parte de la pantalla que otorga al jugador información sobre las estadísticas, vida o información ajena al escenario.
Jugador: Persona que utiliza el juego.

APÉNDICE B

Gráfico del tiempo utilizado:

Total aproximado: 350h