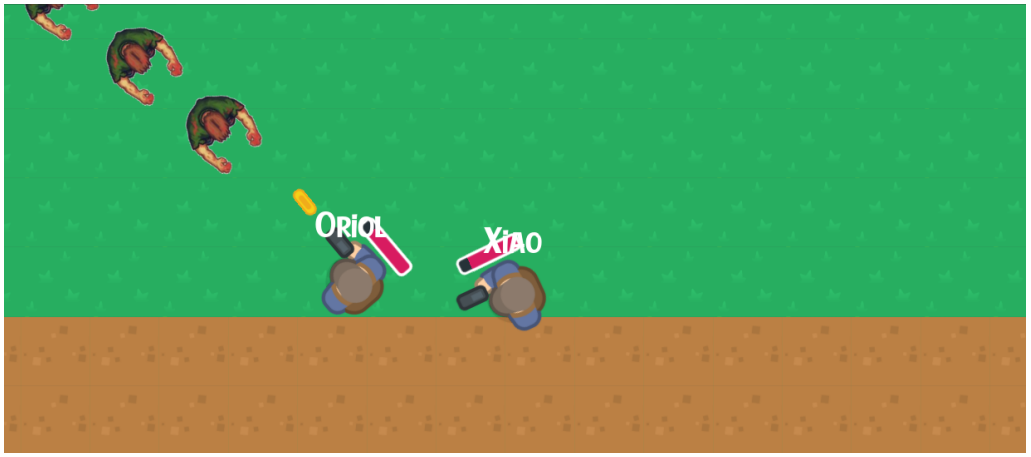
Institut Puig Castellar
Santa Coloma de Gramenet



Creación de juego multijugador (Godot)

(Proyecto de desarrollo)

2nDAM-A
CFGS Desenvolupament d'Aplicacions Multiplataforma

Oriol López, Xiaochao
Grupo A
2n DAM

Creacion de Juego Multijugador Online (Godot) | Oriol López, Xiaochao

Resumen del proyecto:

Este proyecto consiste en la creación del esqueleto de un videojuego con el que se pretenden poner a prueba todas las tareas propias de la creación de videojuegos con un framework de còdigo libre como es Godot, aunque se dará más importancia a la conexión entre clientes ya que se quiere crear un juego con comunicación a tiempo real en la que varios usuarios compartan una misma escena en sus respectivos dispositivos.

El objetivo del proyecto no es tener un juego terminado sino la estructura principal de este a nivel de backend ya que se pretende crear algo desde cero para entender la estructura que hay detrás. Una vez se tenga una estructura bien sólida se podrá terminar el juego en un futuro, o aportar a la comunidad como plantilla para este tipo de juegos en Godot.

Para llevar a cabo el proyecto se pretende seguir una metodología de investigación y desarrollo entremezclado, es decir, el proyecto requerirá de una investigación previa para determinar las metodologías que se usan en los videojuegos actualmente así como adentrarse en Godot. Una vez hecha la investigación se determinarán las metodologías a poner a prueba respecto a las diferentes partes de un juego y se continuará con la implementación en godot.

En resumen el proyecto pretende abordar las metodologías usadas actualmente en la creación de videojuegos e implementarlas haciendo el máximo uso de software libre y con la idea de crear un código aprovechable en un futuro y con posibilidad de aportar a la comunidad de desarrolladores de Godot.

Palabras clave:

Godot, juego, multijugador, conexión en línea, escritorio, software-libre.

Creacion de Juego Multijugador Online (Godot) | Oriol López, Xiaochao

Abstract:

This project consists of creating a videogame structure and learning all the jobs involving video game creation with an open source framework such as Godot,although we will focus on the connection between clients since we want a game with real-time communication in which several users share the same scene on their respective devices in real time.

The project's first goal is not to have a finished game but the main structure of it at the backend level since we intend to create something new and from scratch. Once we have a very solid structure, we will be able to finish the game in the future, or contribute our template for this type of game to the community in Godot and in general.
To carry out the project we intend to follow an intermingled research and development methodology. The project will require prior research to determine the methodologies currently used in video games (special emphasis on connections). Once the research is done, we will determine the methodologies that we will test regarding the different parts of a game and then we will implement them in godot.
This project aims to address the methodologies currently used in video games creation and implement them making maximum use of open source and with the idea of creating a code that can be reused in the future by us or by the developers community.

Keywords:

Godot , game, multiplayer, connection online, desktop, open-source.

Índex

Lista de figuras

1. Introducción

En este proyecto se van a abordar las diferentes partes en la creación de juegos de tipo shooter multijugador online en dos dimensiones. Se va a investigar cada una de las partes que conlleva el videojuego y se implementarán. Para hacerlo usaremos Godot Engine y el editor de Godot como principales herramientas y que están específicamente creadas para este propósito.

La prioridad principal en cuanto a implementaciones serán las conexiones multijugador en tiempo real que engloba la creación del servidor o conexión de pares, la interconexión entre clientes y toda la interfaz de usuario necesaria como menús de salas o creación de partidas.

En segundo lugar se le dará jugabilidad al juego y se implementarán las funcionalidades necesarias para cumplir los objetivos del tipo de juego que se está creando.

En tercer lugar se creará una interfaz de usuario que sea simple pero bien estructurada así como toda la parte más visual y gráfica como animaciones y sonidos. Implementarlo requerirá de cierto tiempo pero se simplificará al máximo para dar más recursos a otros objetivos.

En definitiva se pretende crear una estructura de juego o esqueleto con una jugabilidad concreta y con las conexiones implementadas, y teniendo en cuenta el tiempo del proyecto, dejar para el final la parte más estética.

1.1 Contexto

Para la elección del proyecto se ha decidido enfocarlo a los intereses personales del equipo como són los videojuegos a nivel usuario pero también a nivel de desarrollador ya que este sector está en auge y trabajar como desarrollador de videojuegos y en proyectos donde participen varios incluso muchos programadores es todo un reto.

La elección del tipo de juego está basada en un juego existente y que ya no se puede jugar aunque se le va a añadir una conexión online ya que supone un reto para el equipo y está muy involucrado en todo lo que implica tener varios dispositivos conectados en tiempo real y la forma de tratar los problemas en las conexiones sin involucrar la jugabilidad. Además un integrante del proyecto ha hecho ya una primera aproximación a las conexiones en tiempo real con TCP en javaFX que también se basa en el juego que sirve de inspiración para este proyecto y que ahora con la high level connection API de godot se espera conseguir un resultado más estable.

Por último Godot es una herramienta en auge y que no se usa tanto como otros motores y llama la atención del equipo poder aportar como programadores su granito de arena en forma de plantilla para un videojuego y así dar más valor a este motor “open source”.

1.2 Justificación

La elección de este proyecto nace ya que por un lado el mundo vive un auge de los videojuegos a todos los niveles, tanto juegos de altas prestaciones como juegos de desarrollo independiente como pequeños minijuegos que funcionan con un solo input. Por otro lado Godot es un framework para la creación de videojuegos que todavía no se ha explotado demasiado pero que pretende estar al nivel del resto de competidores siendo open source. En este contexto este proyecto es importante porque se entrará en contacto con las metodologías actuales que están compitiendo a nivel comercial en vista a un futuro laboral, pero más importante aún, se implementará todo ello en open source abriendo el camino a los desarrolladores que usen godot para crear este tipo de juegos.

1.3 Objetivos

1.3.1 Objetivos generales

El objetivo principal de este proyecto es la creación de un juego online que ponga a prueba los conocimientos adquiridos durante los dos años que ha durado el ciclo de DAM. Ya que un juego requiere de la implementación de varios sistemas interconectados, el proyecto intenta abarcar el màximo de estos sistemas sin profundizar excesivamente en ninguno de ellos pués para el proyecto se ha determinado que lo importante es ver las distintas partes y cómo funcionan y no se pretende invertir demasiado tiempo en resultados originales y excepcionales a no ser que se requiera para el funcionamiento del juego. A nivel general existen tres grandes objetivos:

1. Entender el funcionamiento de Godot, su sistema de ficheros y las herramientas de que dispone ya que será nuestra herramienta principal de trabajo y de puesta en funcionamiento.

2. Implementar todo tipo de sistemas o capas relacionadas con los videojuegos como imágenes, colisiones, spawns, mapas,etc.

3. Usar un lenguaje de programación y un framework desconocido para el equipo como pueden ser GDscript o C#, todavía por determinar, y Godot en la categoría de frameworks.

1.3.2 Objetivos específicos

Se organizan los objetivos de la siguiente forma:

Conexión:

Para este proyecto se va a crear una estructura de juego MOG (Multiplayer online games). pero existen variantes en cuanto al funcionamiento que se quiera implementar, por ello como primer objetivo hay que determinar qué tipo de conexión se va a implementar y qué requerimientos tiene. Para decidirlo se realizará también una investigación de la API de que dispone Godot para conexiones.

Interfaz de usuario:

Para este objetivo no se pretende reinventar nada, simplemente implementar una solución que se ajuste a las necesidades del juego y la conexión y aunque en la parte de diseño no se dedicaran muchos recursos, este objetivo es de vital importancia sobre todo en la parte que afecta a la jugabilidad donde sí que se aportarán recursos según se requiera.

Se divide entre:

- **Menús:** Se creará una estructura de menús simple que permita aplicar las funcionalidades de nuestro juego y sean intuitivos para el usuario.

- **HUD:** Interfaz de estado en el juego. Hay que crear un sistema de Hud que muestra al usuario en todo momento la información relevante del juego así como los controles o acciones disponibles, y para ello hay que tener en cuenta los dispositivos en que vamos a ejecutar el juego.

- **Tabla de puntuaciones, chat de sala, modalidades de juego especiales,** etc. Existen múltiples implementaciones que se podrían añadir si sobran recursos pero en tal caso no están planteadas en los objetivos iniciales del juego.

Controles y jugabilidad:

Los controles para el juego dependen de la plataforma en que esté el juego, los inputs de que disponga y por ello este objetivo depende de la decisión que se tome en cuanto a si el juego será multiplataforma o para qué plataforma se creará.

Es así que se requiere de una investigación más exhaustiva de las herramientas de que dispone Godot en lo que se refiere a la exportación del juego y por lo tanto más tiempo. Además esto también afecta a otras capas como la conexión o la interfaz, por ello se ha decidido centrar el objetivo en un

juego desktop. Aún así y por lo que respecta a la jugabilidad se tendrá en cuenta esta posible exportación a otras plataformas si sobra tiempo para ello.

Diseño y estética:

Este proyecto pretende crear una plantilla solamente y siempre prevalecerá aportar valor a la jugabilidad que a la estética o diseño en general. Aun así se requieren unos mínimos de diseño que se abordarán de la forma más simple, minimalista y austera.

1.4 Estrategia y planificación del proyecto

Teniendo en cuenta que la parte más gràfica, los acabados o la parte más creativa del juego a desarrollar no són relevantes para cumplir los objetivos planteados y para evitar gastar demasiado tiempo en todo el proceso de imaginar cómo debería de ser el juego, se ha decidido hacer una adaptación de un juego ya existente, sobretodo por lo que respecta a jugabilidad aunque el resultado final añade otros retos importantes como la conexión online o la implementación en Godot.

De esta forma se va a centrar el trabajo a lo que realmente interesa que es la implementación de todas las partes importantes para conseguir una estructura de juego lo más avanzada posible sin dedicar tiempo a diseñar, imaginar o pensar como debería de ser el juego.

En vista del poco tiempo para tan amplio proyecto, también se va a crear un sistema de menús simple o minimalista y un sistema de juego arcade para que rápidamente se empiecen a integrar la conexión con el servidor, las salas, o la propia jugabilidad e interfaz de usuario.

1.5 Metodología de trabajo

Para la organización y gestión de este proyecto se han valorado diferentes metodologías de trabajo y las que más se adaptan al proyecto són las denominadas metodologías ágiles ya que no se tiene clara todavía la estructura global del proyecto y como se requiere de una previa investigación resulta muy difícil estimar los objetivos de trabajo que se requerirán a largo plazo. En cualquier caso se hará un modelo Gant simplificado para que se tengan claras las fechas de entrega y ver a medida que avanza el proyecto si se van cumpliendo los objetivos a tiempo.

Enlace al diagrama de Gantt: Diagrama de Gantt

Figura 1

De entre los distintos tipos de tecnologías que permiten trabajar de forma ágil se ha elegido Trello por ser la herramienta que más ha usado el equipo tanto a nivel personal como laboral y con la que se siente más cómodo. Además está disponible por la web o por aplicación móvil sin ningún tipo de requerimiento excepto crear una cuenta, lo que facilita mucho el empezar cuanto antes a trabajar.

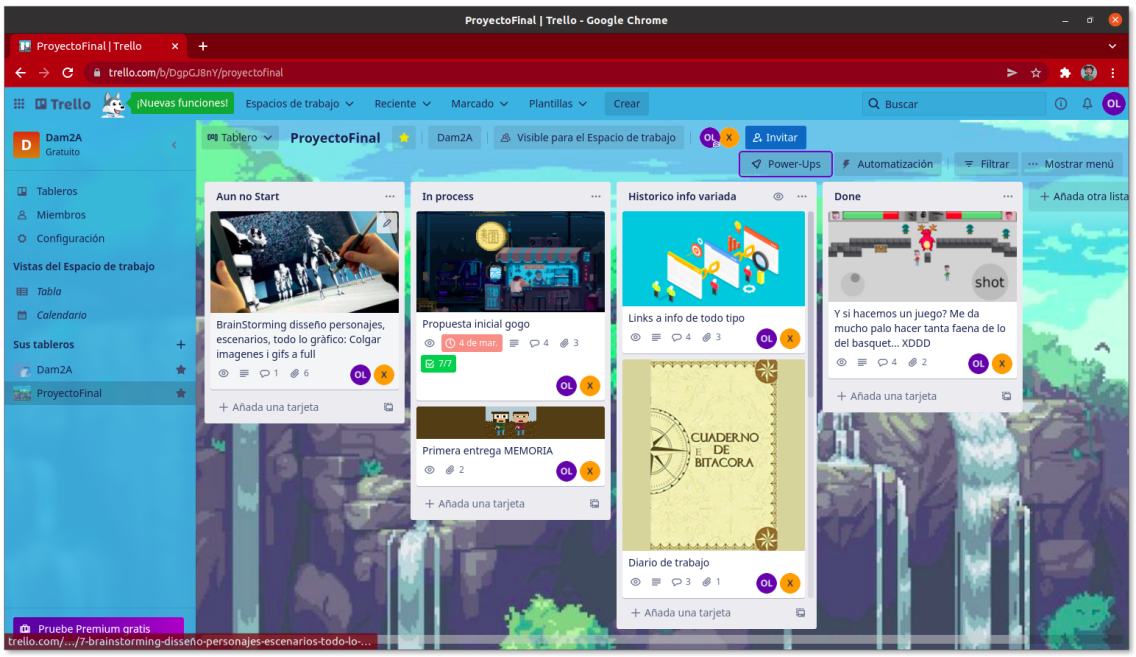


Figura 2

1.6 Estudio económico y presupuestario

Este proyecto está pensado como la etapa más inicial de creación de un juego. Aunque este tipo de proyectos suele contar con varios perfiles distintos de programadores en este caso se ha planteado para dos recursos únicamente que se distribuirán las distintas faenas y todo ello en un tiempo aproximado de 3 meses.

En este tiempo cada recurso tendrá las tareas de analista y programador y dedicaran gran parte de su tiempo a reuniones, búsqueda de información e implementar el backend como desarrolladores. Más tarde llegarán otras etapas como diseño de personajes, tests de usabilidad, lanzamiento, mantenimiento, que seguramente requieren muchos más recursos que la etapa que se trata en este proyecto. A continuación se desglosa el material y los costes de esta etapa primaria.

PC	
Prestaciones	Mínimo Windows 7 Mínimo Ram 8GB Mínimo GPU que tenga soporte de OpenGL 2.1
Programas necesarios	Godot Gimp
Periféricos	Raton Teclado Pantalla Antena wifi

Figura 3

Servidor	
Prestaciones	Mínimo Windows 7 Mínimo Ram 12GB Mínimo NVIDIA GTX 1050 Ti
Herramienta de desarrollo	Godot
Periféricos	Raton Teclado Pantalla Antena wifi
Descripcion*:* Para el servidor, en esta primera etapa del juego no requerimos ni muchas conexiones ni grandes prestaciones en velocidad y podemos intentar montar un servidor en el propio ordenador de trabajo con máquina virtual, o en otro pc si disponemos de él o en una raspberry o en un servidor	

contratado también se valora la idea que el cliente que crea una partida lanze el servidor (Peer2Peer).

Figura 4

Gasto inicial Material	
Raton	12 €
Teclado	40 €
Pantalla	100 €
Torre	500 €
Antena Wifi	10 €
Silla	60 €
Auriculares con micro	30 €
Escritorio	100 €
Total	**852 €**
Descripcion: (Se han hecho estimaciones del precio ya que realmente ya se dispone de este material por parte del cliente o empresa contratante y en cualquier caso no se tendrá en cuenta en el total)	

Figura 5

Programas	
Gimp	Gratis
Godot	Gratis
Planner	Gratis
Firefox	Gratis
Umbrello / Modelio	Gratis
Github Desktop	Gratis
Google Drive	Gratis
Comentario: La mayoría del software usado será libre, si no en su totalidad y el resto será gratuito.	

Figura 6

Figura 7

Gastos mensuales	
Servidor contratado	Min: 0 €(Máquina Virtual) Max: 575,99 € / Mes
Trabajador	996,8 €
Total	996,8 € — 1572,79 €
Link: https://www.ovhcloud.com/es-es/bare-metal/ https://es.indeed.com/career/analista-programador-junior/salaries **Comentario:** La nómina equivale a 800€ brutos aproximadamente	

Figura 8

Costes 3 meses sin gastos iniciales	
Material consumible	5 €
Trabajadores	996,8 €
Total	2995,4 €

Figura 9

Para las siguientes etapas de la creación del juego definitivo se requerirá de más de 3 meses pudiendo llegar a un año dependiendo de muchos factores como el diseño final del juego, si se le implementan más armas, o funcionalidades o costes de contratación de servidores, mantenimiento, etc. Si bien hasta que no se llegue a las últimas etapas de la creación del juego, no

llegarán la mayoría de estos gastos y se mantendrá con los gastos mensuales especificados en esta primera etapa.

Cabe destacar que la parte más crítica de la creación del videojuego y que se puede alargar más es la que menos costes fijos tiene gràcias al uso de software libre, y dependiendo del tiempo que se le emplee el resultado final será mucho más estable y robusto, que se traducirá en menos costes de mantenimiento en un futuro.

2 Descripción del proyecto:

2.1 Anàlisi de requisitos

El principal requisito de este proyecto y donde se busca mayor robustez a parte de la jugabilidad, es la implementación de la conexión, es decir interconectar clientes que creen partidas y jueguen entre ellos.

Entre otras cosas deberá tener un sistema de “lobby” o menú de partidas en línea que permita gestionar las comunicaciones entre clientes y lanzar las partidas.

Por otra parte y en lo que afecta a la jugabilidad, el juego se debe poder jugar como mínimo en modo supervivencia con lo que implica. Rondas de enemigos automáticamente generadas que van aumentando de dificultad ya sea por fuerza del enemigo, ataques de tipos diferentes o cantidad de enemigos diversos. Como mínimo se requerirá un tipo de enemigo para centrar el tiempo en el desarrollo de la jugabilidad del jugador.

Por último este juego está pensado como un arcade que se juega en un mapa cerrado donde hay que sobrevivir a oleadas continuas y como mínimo se requiere un mapa básico y simple justo para que se puedan ver colisiones de balas, jugadores y enemigos o las zonas de creación de oleadas.

2.1.1 Requisitos funcionales

Conexion Online:

Inicialmente se pretende crear un servidor donde se conectarán los clientes/usuarios y desde donde se lanzarán las partidas. Este servidor tiene que conectarse con los clientes para mandarles la información del lobby actualizada y lanzar y cerrar partidas además de controlar a los enemigos y otras funciones del juego. Cuando una partida no tiene jugadores conectados, la partida será destruida.
Una vez se ha investigado más el funcionamiento de la API de conexiones de que dispone Godot se ha visto que la diferencia entre crear una conexión peer to peer o servidor/cliente a nivel de funcionamiento y desarrollo es exactamente igual y se ha tomado la decisión de cambiar el tipo de conexión a peer to peer ya que la implementación es casi idéntica pero en peer to peer podemos ejecutar el juego de forma más rápida y simple sin tener que estar actualizando código en servidor y cliente cada vez que se hace un cambio.

Jugabilidad:

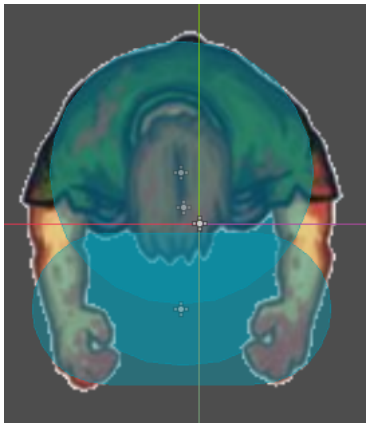
Para la jugabilidad del juego se tendrán que pensar muy bien los controles en vista a la posible implementación en distintas plataformas de este juego, como por ejemplo en tabletas o móviles. Existe una temática de juego y una serie de normas de juego que tendrán que cumplirse en lo que al jugador se refiere.
Los controles básicos són las típicas teclas de dirección correspondientes con las letras w,a,s,d que moverán el jugador en la correspondiente dirección además de en dirección diagonal en caso de pulsar dos teclas no contrarias a la vez. Además el jugador mirará continuamente hacia el puntero del ratón y con el clic izquierdo disparará.

Lobby sala de espera:

Para gestionar la creación y visualización de partidas habrá un lobby donde los usuarios puedan escoger la partida en la que conectarse. Este lobby va a mostrar las partidas que están activas en forma de lista y se va a mostrar información de cada una que permita diferenciarlas, como por ejemplo el nombre del usuario que ha creado la partida, o la IP. Además se va a crear una sala de espera para los usuarios que se unan a una partida y desde donde lanzar o iniciar el juego de forma sincronizada para todos los clientes.

Pseudo ia:

La modalidad principal del juego consta de un sistema de enemigos con diferentes niveles de dificultad que tendrán comportamientos programados además de un sistema de rondas que va aumentando en dificultad. Estas rondas aumentan en número y añaden nuevos tipos de enemigos más fuertes.
En cuanto al comportamiento programado este consistirá en seguir al personaje más cercano y al colisionar hará daño a los jugadores.



El círculo de la mano sirve para hacer daño al jugador.

El círculo de la espalda sirve para recibir daño del jugador.

El círculo del centro sirve para tener una colisión entre enemigos para que no se ponen una encima del otro.

Figura 10

Mapa:

Para este primer prototipo solo se implementará un mapa aunque se añadirán más en un futuro. Por esta razón se tendrá que implementar un sistema de elección de este así como la fácil integración de distintos mapas una vez acabado el proyecto. Como principales características todos los mapas tendrán las mismas dimensiones fijas. Si bien el juego que se usa de modelo no tiene un seguimiento de cámara, en este proyecto se va a usar ya que aporta más

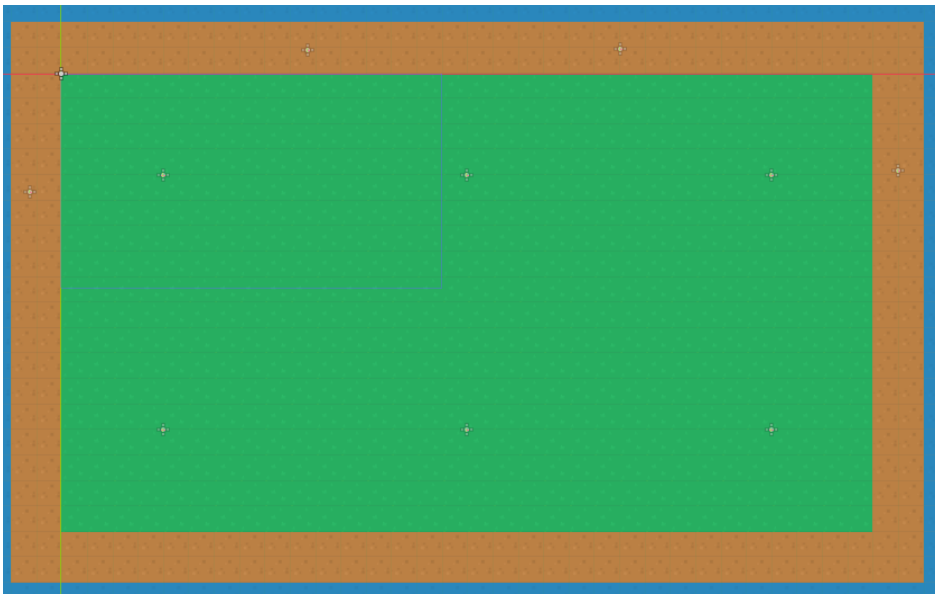


Figura 11

movimiento a la escena. En la imagen podemos ver el mapa creado junto a los puntos de spawn.

Hud:

El Hud es la parte que aunque no se le de una estética, ha de tener una serie de elementos que ayudarán a que el usuario entienda qué está pasando en cada momento y por ello es una funcionalidad necesaria para este proyecto. Los principales atributos o visualizaciones que tiene serán el nombre y vida de los jugadores, la vida de cada enemigo, la oleada actual, los enemigos restantes de la oleada actual o próxima y el tiempo de espera entre oleadas. Además cuando el usuario muera pero queden usuarios con vida

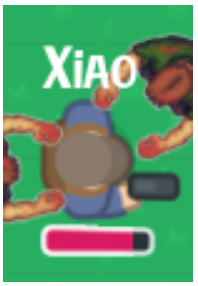


Figura 12

2.1.2 Requisitos no funcionales

Estable/Robusto:

Uno de los requisitos principales de este proyecto es la robustez del resultado final, ya que se trata de una plantilla de juego este debería de no tener ningún tipo de bug y teniendo en cuenta todas las problemàticas derivadas de las conexiones puede que este requisito sea de los más complicados teniendo en cuenta el tiempo. Por ello se va a crear una documentación que facilite el arreglo de cualquier posible bug así como mejorar la estructura de la plantilla que quede sin resolver.

Intuitivo/simple:

Por otro lado el juego resultante tiene que dar la sensación de control del usuario hacia el personaje así como ser intuitivo en los menús y creación de juegos o partidas. En definitiva empezar a jugar tiene que resultar simple así como el control del personaje.

Compatible:

Esta plantilla de juego está pensada para lanzarse en escritorio pero para múltiples sistemas operativos. Se dará el caso que el servidor y cliente no usen el mismo sistema operativo y esto no tiene que dar problemas de compatibilidad.

Actualizable:

Este juego está pensado como plantilla y se van a implementar algunas opciones donde solo habrá una posible selección y que más tarde se podrán ir ampliando con más opciones como nuevos mapas, nuevas armas, nuevos enemigos, etc, para que el juego vaya creciendo.

2.2 Tecnologías

2.2.1 Comparativa de les tecnologías valoradas

Para el proyecto se han valorado muchas tecnologías de diferentes tipos o enfocadas a distintas funciones. Ya que uno de los puntos clave del proyecto es usar tecnologías open source, se ha decidido agruparlas según si són de código abierto o si són simplemente gratuitas ya que en principio no se pretende gastar dinero en software.

Tecnologias Open source:

Para la creación del juego se ha valorado el uso de diferentes frameworks, motores y librerías siendo Godot el elegido. Entre las más valoradas se encuentran Unreal Engine quizás por tener altas prestaciones, Unity por tener una gran comunidad detrás y libGDX por ser un lenguaje de programación conocido al que se le añaden unas librerías específicas para creación de juegos.

Finalmente Godot tiene todo lo que se necesita para este proyecto y representa un reto a nivel personal y profesional.

Figura 13

Ha habido dudas en cuanto a herramientas como “gantt project” , “gantt planner” o la propia herramienta de trello y si bien todas permiten las funciones que se requieren para este proyecto, la exportación de los datos que se crea es más atractiva en gantt planner siendo el programa elegido finalmente.

Figura 14

Para crear los diagramas uml se va a usar modelio ya que es un programa open source y además el equipo ya lo conoce. También se ha valorado la herramienta Umbrello, la cual se ha investigado mínimamente siendo también open source. Finalmente se ha elegido modelio principalmente por disponer de la función de deshacer o Ctrl+Z.

Figura 15

Para la creación de gráficos se va a usar GIMP y aunque no se pretenden hacer muchos gráficos, se utilizará para editar imágenes simples que hagan de plantilla para poder probar efectos visuales y animaciones.

Figura 16

Tecnologias gratuitas:

Trello es la herramienta principal que se va a usar para la gestión y comunicación en el proyecto. Se ha elegido por lo fácil que es de usar aunque con la dirección que va tomando el proyecto habría sido muy útil montar taiga en un servidor por el tipo de metodología que usa.

Figura 17

Para compartir archivos y modificarlos se ha valorado la idea de crear un servidor en una raspberry con Next Cloud instalado pero se va a usar drive donde se ha creado una unidad compartida para todo el equipo y donde gracias a otras herramientas como google docs, se pueden modificar archivos al mismo

Figura 18

tiempo sin pérdida de información. Aun y no ser software libre nos da una serie de ventajas con las que no se ha encontrado competidor. Además es gratuito.

Para el diseño de menús y ventanas se ha elegido Figma de herramienta por estar web y por permitir trabajar a la vez sin problemas de pérdida de datos. Además esta herramienta se ha utilizado bastante durante este curso escolar y no se quiere perder tiempo en aprender a usar otra distinta.

Figura 19

2.3 Estructura del proyecto

En este proyecto hay una serie de elementos importantes relacionados entre sí que se estructuran de la siguiente forma.
En primer lugar vamos a tener diferentes dispositivos que van a tener el juego instalado y van a comunicarse entre ellos a través de un sistema “peer to peer” o de pares generado por la API de Godot. En el siguiente diagrama se puede ver cómo se comunican todos los clientes entre ellos y como no hay un servidor como tal que actúe de intermediario.

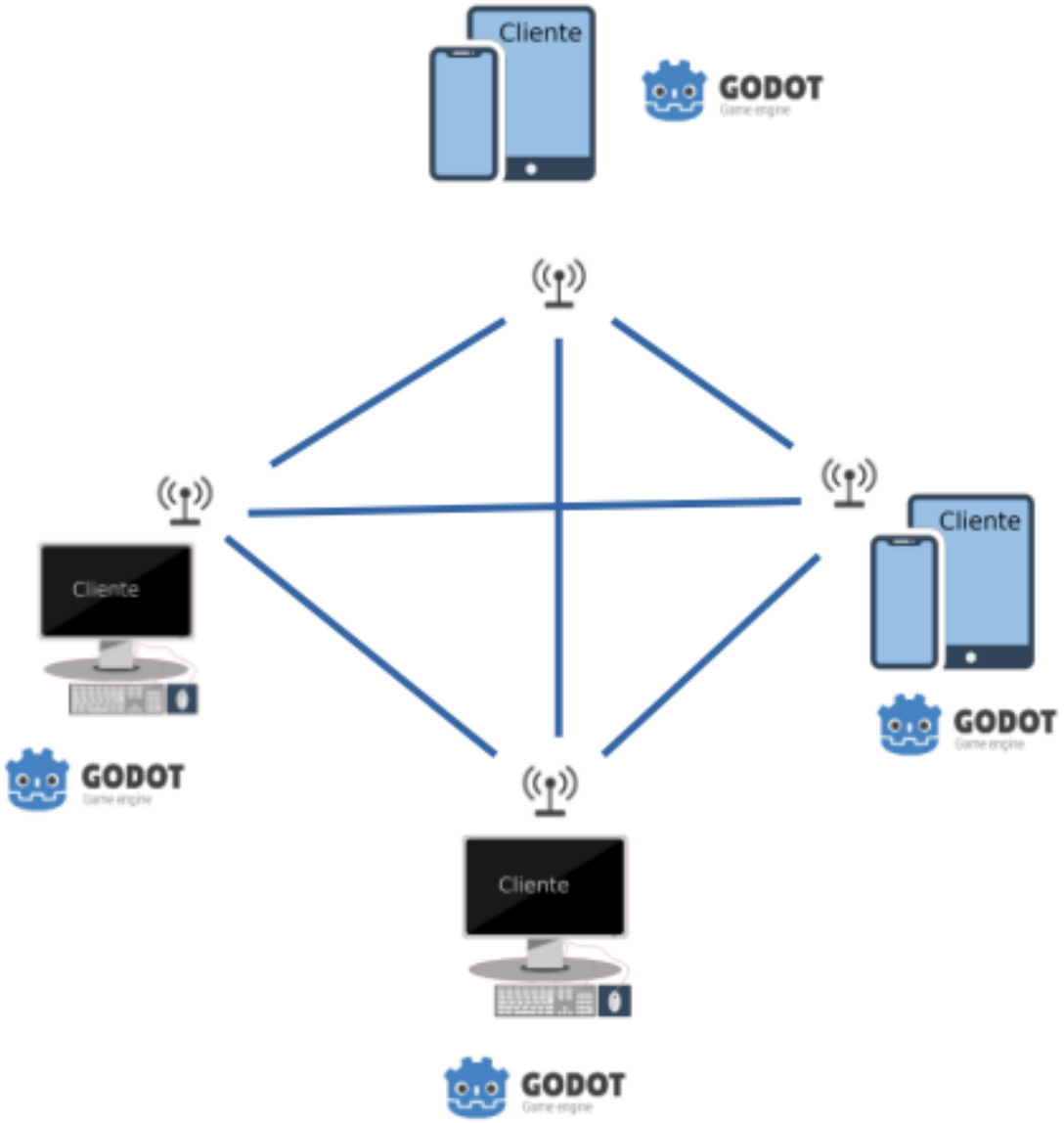
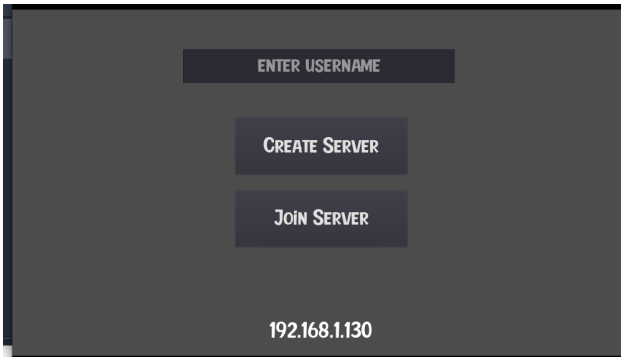


Figura 20

Cada uno de estos clientes ejecutan el juego en su dispositivo el cual se estructura de la siguiente forma.



En sí el juego consta de una pantalla inicial para elegir si se quiere hostear la partida o unirse a una. En el primer caso el player genera la partida y actúa como servidor entrando directamente dentro de la partida a la sala de espera donde se irán conectando el resto de usuarios y desde donde el usuario que hostea la

Figura 21

partida decidirá iniciarla con un botón específico que solo aparece en su dispositivo.

Por otro lado el usuario que no hostea la partida entrará a la sala de partidas o “lobby”. En este menú aparecen las partidas creadas por otros usuarios en la misma red permitiendo al usuario ver el nombre de quien hostea la partida y su IP y este puede unirse a la partida a través de un botón “join”. Al unirse a la

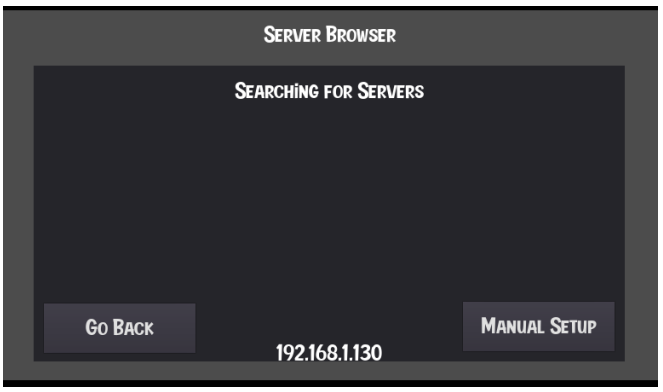


Figura 22

partida entra en la sala de espera donde se instancian ya los personajes para que todo esté preparado para comenzar la partida.

Una vez iniciada la partida comienza el juego que consiste en un arcade de supervivencia donde todos los jugadores deben sobrevivir a las oleadas de enemigos que se van generando. Todos los nodos que existen en los diferentes clientes se instancian en una clase llamada persistent_nodes que se usa para referenciar todos estos objetos y para saber en cada momento qué nodo de un cliente corresponde con el de otro cliente para efectuar las operaciones correspondientes.

En el siguiente diagrama de clases se puede ver cómo se relacionan los diferentes nodos que no están directamente en el nodo Game sino que se instancian de forma global y se añaden al nodo Persistent_nodes que actuará como árbol de nodos para los objetos del juego que tienen que estar sincronizados como los players, enemigos y balas.

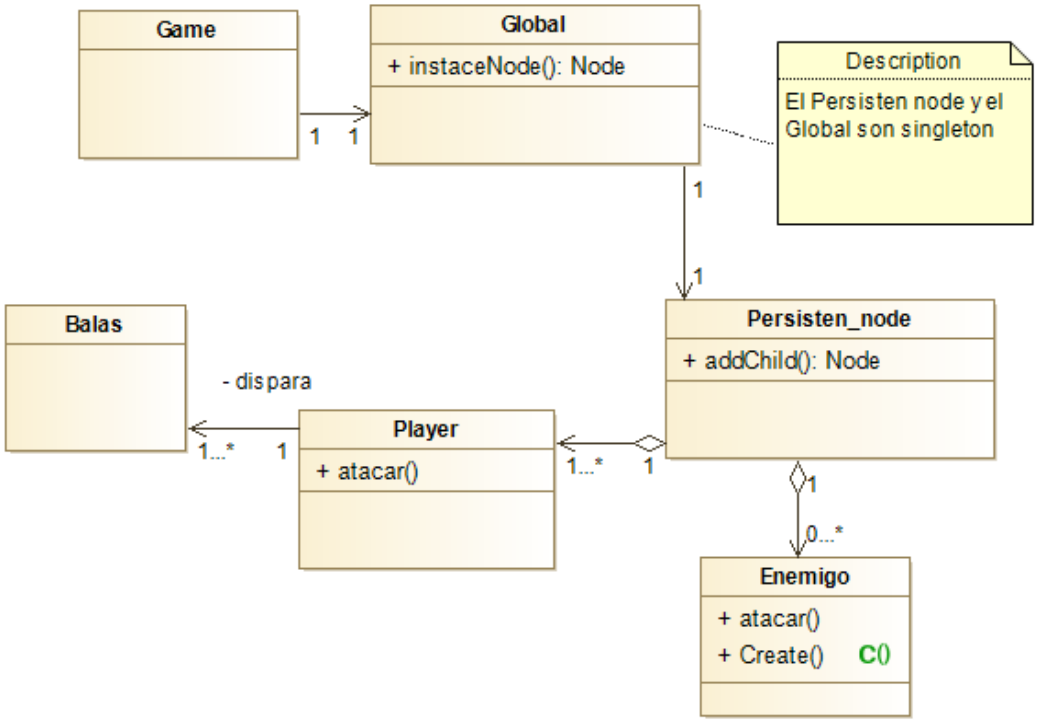


Figura 23

2.4 Descripció dels components

Este proyecto consta de varios elementos clave para su puesta en funcionamiento descritos a continuación.

Conexión:

El componente principal para llevar a cabo este proyecto es la conexión que a su vez consta de varios elementos.
Se va a diferenciar entre la conexión de la sala de partidas donde los

Para gestionar la conexión se va a usar la High Level Connection API de GODOT, que es una herramienta preparada y documentada para Godot. Esto va a facilitar la implementación de este componente aunque habrá que diseñar una interfaz de usuario y una serie de reglas y funciones para gestionar desconexiones, conexiones nuevas y posibles conflictos que vayan surgiendo.

Interfaz:

Otro de los componentes que más tiempo requiere es la interfaz con el usuario. Por un lado a nivel de conexiones hay que crear un menú de partidas o “lobby”, y una sala de espera para conectar los clientes a la misma partida y también las distintas ventanas de mensajes relacionados

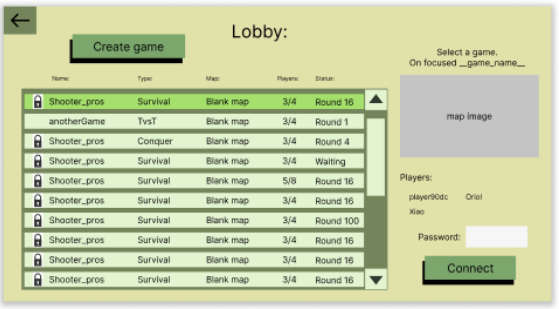


Figura 24

con la conexión. Avisos, errores, advertencias, etc. Por otro lado se crearán los menús necesarios para la creación de partidas y la puntuación final de cada partida y un menú de configuración básico de volumen de música y de efectos. Por último hay que implementar un hud con la información de la partida.

Lógica del juego:

Otro componente básico para la realización del proyecto es la lógica del juego. Este consiste en un shooter en dos dimensiones y consta de colisiones, puntuaciones, objetos, cambios de armas, munición, vida, etc. Todo esto tendrá que relacionarse cumpliendo los requisitos de jugabilidad del proyecto y a su vez tendrá que conectarse con el hud y con la conexión entre servidor y clientes. Algunas de las partes más importantes són los movimientos y colisiones que implican directamente a la conexión entre clientes, los controles que permitirán al usuario tomar el control de su personaje y aportará jugabilidad al juego, la implementación de múltiples armas y objetos así como tipos de enemigos distintos con habilidades distintas.

Pseudo ia:

La modalidad principal de este juego es survival, y consta de un sistema de generación de rondas u oleadas de enemigos que aumentarán de dificultad a medida que avance el juego ya sea aumentando el número de enemigos como añadiendo nuevos enemigos más fuertes o con habilidades diferentes.
Los enemigos tendrán distintos comportamientos programados y distintos atributos según el tipo siguiendo el estilo de Minecraft donde existen distintas habilidades y atributos como la velocidad, que cambian según el enemigo. Esta lógica de juego se genera desde el servidor y por ello se podrán hacer cambios que afecten a todos los clientes sin necesidad de instalar o actualizar a una nueva versión.

2.5 Definició de les funcionalitats

2.5.1 Conexión Online

La conexión online es una de las funcionalidades más importantes para este proyecto y se puede dividir en dos grandes partes.
La primera parte es la gestión de las conexiones de clientes para comunicarnos y añadirlos en una misma partida.
La segunda parte consiste en las conexiones que sincronizan los datos entre nodos en la ejecución del juego en cada cliente de tal forma que todos los clientes vean lo mismo y no se vea perjudicada la jugabilidad.

2.5.2 Lobby - Sala de espera

El lobby va muy ligado a la conexión y administración de partidas. Tendrá varias opciones para visualizar las partidas activas así como acceder a la visualización de los datos de cada una de ellas y además dará acceso a la creación de nuevas partidas.

2.5.3 Jugabilidad

El movimiento en el juego tiene bastantes partes pero se van a tratar de forma simplificada en cada caso.

2.5.4 La IA

La IA básicamente es el script encargado de gestionar el movimiento, spawn y vida de los enemigos. Para este proyecto se ha planteado un sistema de oleadas que van a ir aumentando en cantidad de enemigos o fuerza constantemente hasta llegar a una dificultad tan alta que los jugadores no puedan superar.

2.5.5 Mapa

El mapa va a ser todo plano con unas colisiones en la esquina para que el jugador no salga del mapa y algún bloque intermedio para que se puedan ver las diferentes colisiones del juego, luego al final si hay suficiente tiempo se irán haciendo diferentes tipos de mapas.

2.5.6 Hud

En principio se va a hacer el HUD de personajes usando GIMP, también se pretende hacer un par de armas diferentes que necesitaran un indicador de munición.

3 Desarrollo

3.1 Introduccion

A continuación se detallan las partes más relevantes en cuanto a implementaciones realizadas a nuestro juego así como la información más importante para nuestros objetivos de investigación sobretodo en lo que se refiere al uso de conexiones distintas y metodologías en los juegos multijugador actuales así como el uso de las herramientas del framework utilizado.
En todo caso si alguna parte de las implementaciones no queda clara el código de nuestro juego contiene toda la información referente al uso de cada método o variable así como notas allí donde hay datos que conectamos entre clientes.

3.2 Conexión:

Como ya se comentó en puntos anteriores, para este proyecto se ha dedicado gran parte del tiempo a la búsqueda de información y toma de decisiones a la vez que se han hecho pruebas para ir comprobando el funcionamiento, haciendo forks de otros proyectos que implementan diferentes tipos de conexiones, viendo tutoriales o siguiendolos al pie de la letra y leyendo una y otra vez la documentación de Godot. Todos los repositorios usados así como tutoriales y videotutoriales están especificados en la bibliografía.

Elección del tipo de conexión:

La primera decisión importante ha sido la elección de esta conexión que finalmente ha sido peer to peer por distintos motivos. En primer lugar la implementación de la conexión depende de la API de que dispone Godot para conexiones y una vez investigadas las implementaciones de servidor dedicado y peer to peer se ha visto que se parecen mucho en cuanto a código se refiere. Siendo así se ha valorado el uso de peer to peer ya que facilita mucho el trabajo de desarrollo y no hay que estar modificando servidor y cliente.
Una vez tomada esta decisión se han comenzado a probar implementaciones de todo ello y básicamente se ha investigado a fondo la high level API de Godot.

La High Level Network API:

Inicialmente se pensó que había que implementar el sistema de envío de paquetes tcp o udp en el juego pero Godot ya tiene una API que gestiona estas conexiones así que el primer gran punto del proyecto ha cambiado desde un inicio y lo que ha llevado más tiempo ha sido entender el funcionamiento y

todos los métodos de que dispone esta API pero sobretodo ha habido un proceso de adaptación en cuanto a la forma de concebir la comunicación tal y como lo hace Godot y su sistema de organizar el código.
A continuación se desglosan los puntos más importantes que permiten gestionar las conexiones y que se han valorado como más relevantes..

Roles(master y puppet):

La high level network API dispone de un protocolo de roles para indicar a una clase quién controla ese nodo o instancia y así facilitar el desarrollo de las comunicaciones entre clientes. Básicamente tenemos dos grandes roles. El Máster (Dueño) y el Puppet (Marioneta). Este sistema facilita determinar en un mismo script si se tienen que enviar datos, o recibirlos.
En cualquier caso todo depende del punto de vista y quién se comunica con quién. En el siguiente diagrama se puede ver mucho más claro quien tiene el rol de master o de puppet visto como la comunicación de el servidor y el cliente uno y el servidor con el cliente dos. Los nodos con círculo doble representan los masters y los otros són puppets.

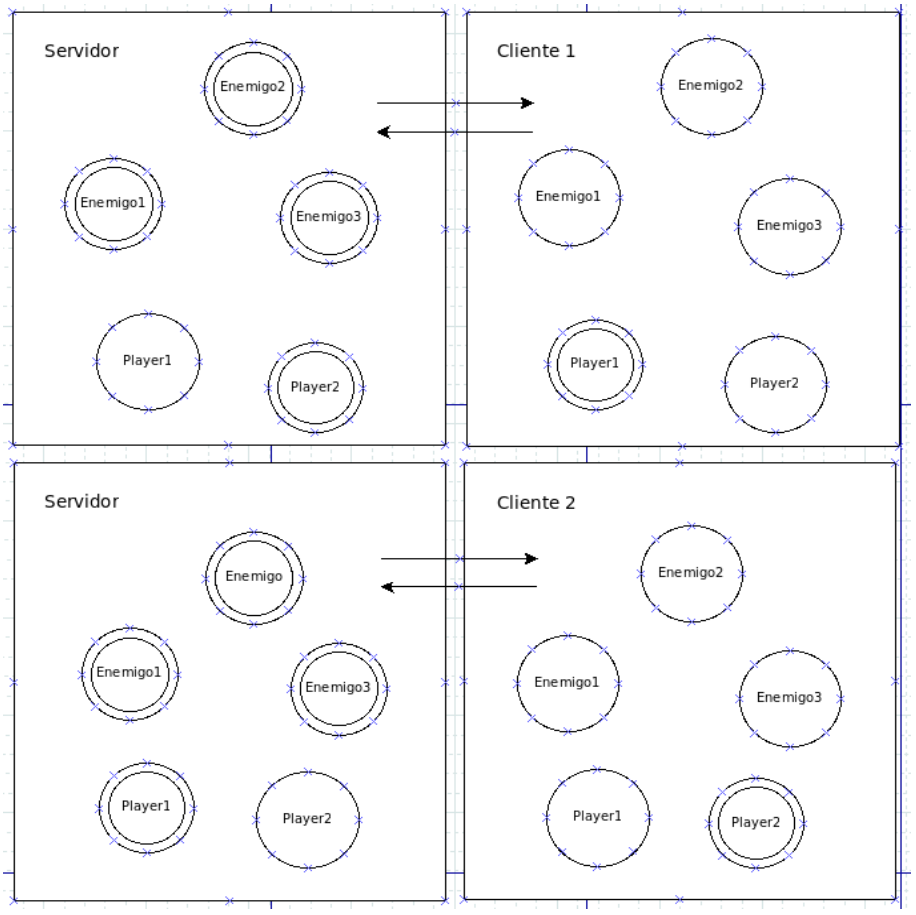


Figura 25

Esto significa que cuando el script del player se ejecuta en el cliente que controla ese player trataremos los inputs del teclado, pero si se está ejecutando en el servidor se actualizan los datos generados por los inputs del master de la instancia de ese nodo. Para ello disponemos de varios métodos y etiquetas que nos facilitan determinar qué rol tienen las instancias en cada tipo, y sobre qué rol queremos actuar.

Para entenderlo mejor a nivel práctico, si miramos el siguiente ejemplo de código vemos cómo a través del método **is_network_master()** determinamos si el player en cuestión pertenece al cliente o no. De esta forma podemos indicarle el movimiento que en este caso tendrá una serie de inputs para moverse. Por otro lado si el usuario/cliente no es master del player le tenemos que pasar la información del player master al player en cuestión que ahora cumple el rol de puppet o marioneta.

Para ello se hace uso de los métodos rset que són setters y que están actualizando el atributo de forma remota. Básicamente dice que las instancias de ese nodo en concreto que tengan la misma ”**networking_unique_id**”, actualizarán el valor (segundo parámetro) del atributo (primer parámetro). La diferencia entre los dos métodos radica en el tipo de conexión que usa, siendo el **rset** de tipo TCP y **rset_unreliable()** UDP en los cuales se profundizará más adelante. Si no es **network_master** se actualiza la posición con el valor del atributo **slave_position** qué habrá sido actualizado por el master de ese nodo.

Además podemos dar un rol a las variables y métodos de un script de forma que solo los usara el rol especificado en cada instancia de cada cliente. Este funcionamiento requiere de un análisis detallado de los roles que cumplen las diferentes instancias de un nodo en los diferentes clientes y servidores.

TCP vs UDP:

En las herramientas que nos brinda godot para sincronizar datos entre clientes encontramos siempre dos opciones como en el caso de **rpc()** y **rpc_unreliable()** o **rset()** y **rset_unreliable()** que son de tipo TCP y UDP respectivamente.
Ante esto se ha investigado el uso que se le da en el desarrollo de aplicaciones y se han determinado dos aspectos claves en la decisión del tipo de conexión.

El primero y más importante es la velocidad ya que el proyecto consiste en un juego online a tiempo real el envío de paquetes tiene que ser lo más rápido

posible y por ello se intentará enviar el máximo de paquetes en UDP aunque la pérdida de algunos de estos paquetes puede llevar a errores en el juego, .

```
if is_network_master():    # network_master significa que la
    if Input.is_action_pressed('left'):
        direction = MoveDirection.LEFT
    elif Input.is_action_pressed('right'):
        direction = MoveDirection.RIGHT
    elif Input.is_action_pressed('up'):
        direction = MoveDirection.UP
    elif Input.is_action_pressed('down'):
        direction = MoveDirection.DOWN

    # Con los metodos rset actualizamos (setter) el atributo
    rset_unreliable('slave_position', position)
    rset('slave_movement', direction)
    _move(direction)
else:
    _move(slave_movement)
    position = slave_position
```

Figura 26

El segundo gran aspecto consiste en solucionar el problema de la pérdida de paquetes en UDP. Para ello hay que tener muy claro qué tipo de información estamos enviando y que puede suponer el hecho de que no llegue esa información en uno de esos paquetes concretos entre muchos otros que sí que llegan, y esto es importante ya que realmente la perdida es minima pero si justo lleva una información decisiva para el funcionamiento del juego puede llevar a problemas graves en la sincronización entre clientes.

En el caso de los **rset** de la imagen anterior en el primer caso se está haciendo un setter remoto a la variable **slave_position** en UDP (esta variable la usan los clientes para actualizar la posición de los otros jugadores). Este código está dentro de **_process()** , un método que se ejecuta cada frame por ello estamos enviando 60 paquetes por segundo y si uno no llega, inmediatamente llega otro que actualiza a una nueva posición que corresponde con la más actual. Si enviamos un paquete TCP y no llega, se envía de nuevo el mismo paquete otra vez y este paquete contendrá la posición del paquete original y no la más actual. En este caso podríamos llegar a ver una cierta ralentización del movimiento del resto de jugadores mientras que en el caso anterior simplemente se produce un pequeño salto que realmente es imperceptible al ojo humano.

```
var hp=100;

func herido():
    rpc('resta_vida',10)

func resta_vida(aRestar):
    hp-=aRestar
```

Figura 27

En el caso contrario si imaginamos el caso de una bala colisionando con un enemigo hay un solo paquete que envía la información de eliminar al enemigo ya que en ese mismo frame la bala se destruye y ya no se siguen enviando paquetes de estado por tanto el enemigo no morirá en uno de los clientes.

En definitiva y como protocolo a seguir se ha determinado que se pondrá UDP por norma evitando así la ralentización en muchos casos, luego se pensará si se pueden dar errores por pérdida de datos, en tal caso se determinará si el

uso de TCP puede llevar a una ralentización del juego. Si fuera el caso se buscará una forma de solucionar el problema haciendo cambios al código.
Un ejemplo de cambio en el código sería crear una variable en el enemigo que contenga un booleano para saber si esta vivo o muerto y activar un timer para que la destrucción de la instancia ocurra después y se manden los suficientes paquetes UDP con el nuevo estado **ha_muerto** igual a true y así asegurar que la variable se ha actualizado en todos los clientes antes de destruir el nodo enemigo.

Actualización y reconciliación de datos:

En cualquier juego online tenemos que enviar constantemente información entre clientes para conseguir dos cosas. Actualizar la información de las instancias de nodos en los distintos dispositivos y reconciliar los casos en que se creen incongruencias en los datos entre las diferentes instancias de un mismo nodo. Para ello Godot dispone de varias herramientas y en concreto de unas etiquetas que indican el rol (master o puppet) de cada nodo en cada caso y en qué caso hay que ejecutar cada método.

Un ejemplo básico para entenderlo es el caso en que posicionados en el nodo player, en su script principal vamos a ejecutar un método llamado herido cuando le colisione una bala. Cuando se lance este método se enviará una llamada para ejecutar el método **resta_vida()** de forma remota. De esta forma cuando un player recibe daño ejecuta el método en todas las instancias de ese nodo de tipo player y les resta vida a todos.
Al hacerlo de esta forma ocurre que todas las instancias de player reciben una colisión y todas ejecutan el método remoto resta vida de tal forma que cuantas más instancias existan del mismo player (una por cada cliente), se va a restar tantas veces la vida. Por ello hay que hacer uso de los métodos de la classe Networking, como són “**is_network_server**“ o “**is_network_master**” para determinar si el nodo en concreto corresponde con el cliente que hace de servidor en el primer caso o si la instancia de nodo en el cliente que se ejecuta tiene el rol master o no.
Si determinamos que la instancia que ejecuta el script es master, entonces si se mandará ejecutar el método resta_vida de forma remota ejecutando una sola vez en todas las instancias de ese nodo.
Por otro lado, con el método **is_network_server** se usará ampliamente para la reconciliación de datos.

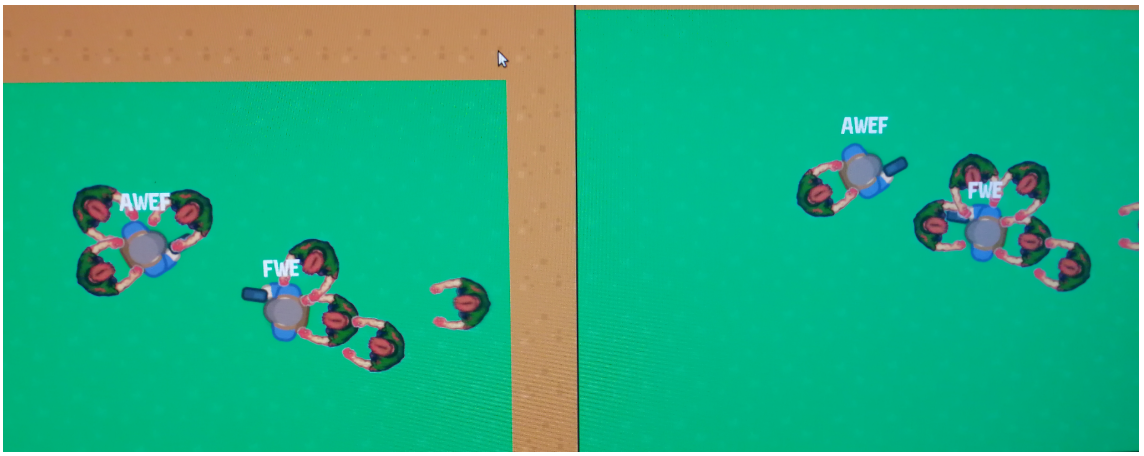


Figura 28

La reconciliación de datos existe ya que cada cliente que ejecuta el juego tiene una gran cantidad de datos que van ejecutándose en cada máquina llegando a crearse incongruencias en el juego. Un ejemplo es el caso de los enemigos que tienen una IA que los mueve. Generar el movimiento de estos enemigos desde el servidor ralentiza el juego así que en cada cliente los enemigos se mueven siguiendo lo que marca su script evitando de esta forma la ralentización y dando mucha más fluidez al movimiento que al generarse enteramente desde el servidor crearía pequeños saltos en el movimiento perceptibles por el usuario. Al hacerlo de esta forma puede ocurrir que haya pequeñas variaciones en los datos que no ocurren exactamente en el mismo instante en cada cliente y por ello una misma instancia de un enemigo puede ser que tengan posiciones algo distintas y con el paso del tiempo se vaya sumando el error creando grandes diferencias entre lo que pasa en un cliente y otro. Por ello hay que reconciliar los datos o lo que es lo mismo actualizar la posición del enemigo con la instancia máster, que en este caso corresponde con el servidor y por tanto cada cierto tiempo se enviaran datos como la posición exacta y el player al que está siguiendo para que se actualicen todas las instancias en los diversos clientes y solucionar las incongruencias que se creen.

3.3 La IA:

Una de las funcionalidades más importantes junto con la implementación de múltiples jugadores, es la creación de enemigos y el comportamiento de estos por una parte, y la conexión sincronizada en todos los clientes. Si bien la conexión entre jugadores ha resultado un reto, ha sido la implementación de los enemigos la que ha llevado más tiempo y ha supuesto el verdadero reto de este proyecto. Al crear enemigos estos los genera el servidor que en este caso es uno de los clientes y es el servidor quien controla su spawn y su lógica en general que luego se replica en todos los clientes.

A continuación se detalla el funcionamiento de estos enemigos y cómo traspasan la información al resto de clientes así como las soluciones a las incongruencias que se puedan generar.

Como ejemplo, en el siguiente diagrama de secuencia podemos ver cómo se crea uno de estos nodos, exactamente el nodo de un enemigo.

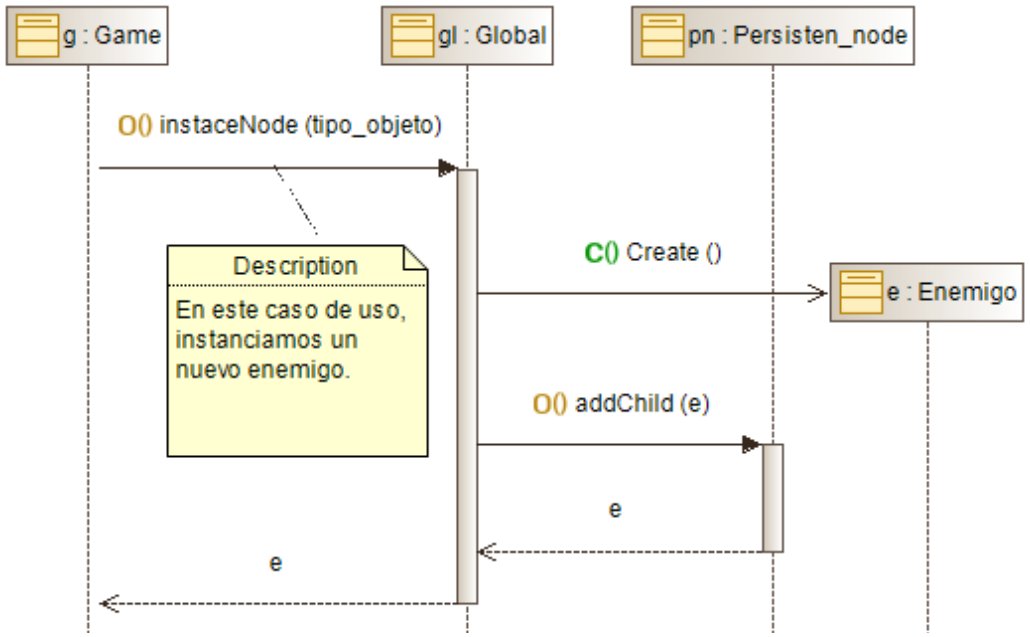


Figura 29

Para que el juego tenga una jugabilidad y una fluidez, es cada cliente el que trata el movimiento de cada enemigo. A su vez el movimiento de los enemigos se genera en base a la posición de un jugador y a su posición global en el mapa. De este modo el enemigo puede llegar a tener comportamientos diferentes en los diferentes clientes, como podría ser que en uno de los clientes el enemigo siga a un jugador y en otro siga a otro distinto. De esta forma se consigue un movimiento mucho más fluido pero luego hay que contrarrestar estos errores.

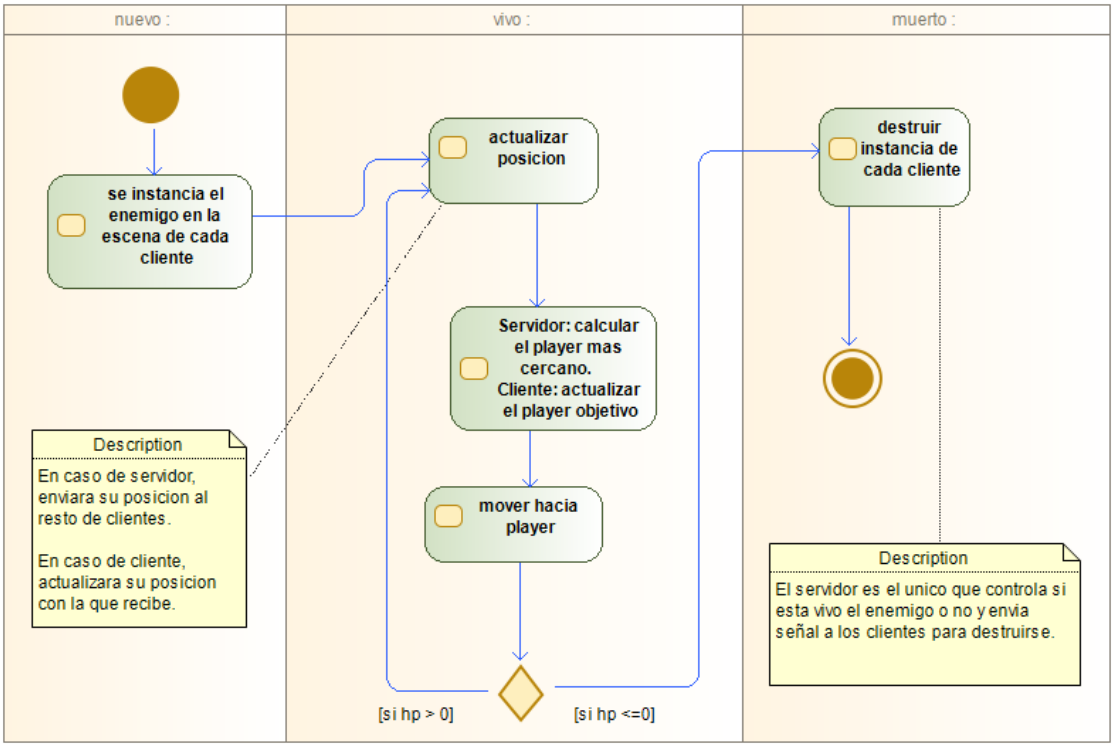


Figura 30

Para contrarrestar estos errores hay que implementar un sistema de reconciliación de datos como el explicado anteriormente y para ello lo primero es especificar qué cliente o instancia del enemigo prevalecerá por encima del resto. Una vez elegido, este cliente , que en este caso será el que hace de servidor, es el encargado de enviar a cada frame su información simplificando su posición y el player al que sigue. El resto de instancias de ese enemigo en los diferentes clientes deberán actualizar su posición y player al que siguen y así solucionar cualquier incongruencia entre clientes pero dando el control del movimiento a cada cliente.

Respecto a la implementación se ha creado un sistema simple pero efectivo que cumple muy bien con la idea del juego que se tiene. Los enemigos se dirigen constantemente al player más cercano y miran hacia él y por ello tiene tres funciones principales que se ejecutan cada loop del juego. En primer lugar se busca el jugador que esté más cercano. En segundo lugar se genera un vector en dirección a la posición del player que persigue junto con su velocidad de movimiento por defecto de los enemigos. Por último a través de la función **face_to()** junto con la posición del player a perseguir le indicamos la rotación del nodo para que mire hacia el player.
Este sistema es muy simple y si hay por ejemplo un obstáculo de por medio los enemigos se chocarán continuamente contra este.

Si se quisiera solucionar este caso se tendría que implementar generando el movimiento desde el servidor únicamente pués aumentaría significativamente los datos incongruentes.

3.4 El lobby:

El lobby o sala de partidas es un punto importante para este proyecto ya que gestiona las conexiones entre diferentes clientes. Para realizarlo se han probado soluciones ya implementadas sobre las cuales hemos realizado todo tipo de pruebas para entender cómo las gestiona godot. Concretamente godot pone a disposición varios repositorios por categorías y una concretamente sobre conexiones que ha servido para comprender las bases de la implementación de conexiones con godot. Este punto en concreto ha condicionado totalmente el tipo de solución implementada, y se ha simplificado el modelo inicial de lobby que se hizo en un principio y se ha concretado definitivamente la conexión que se ha usado en el prototipo. Este muestra el jugador que crea la partida y la IP por si se quiere conectar manualmente y el botón para seleccionar la partida de una lista de los clientes que han iniciado una partida en la misma red.

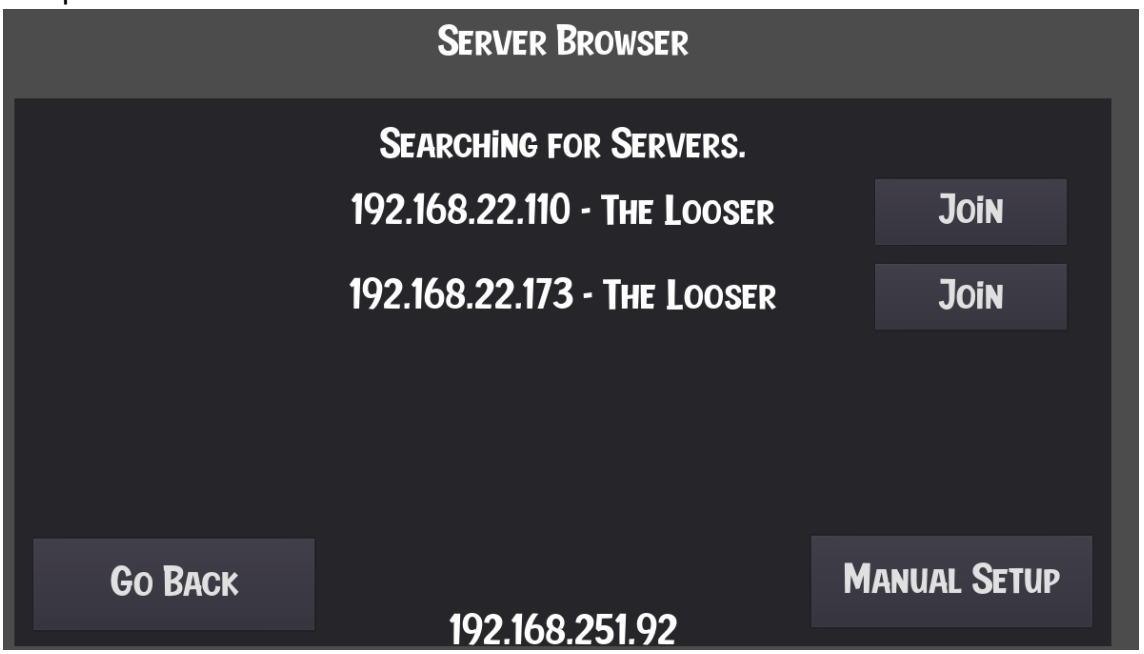


Figura 31

A nivel de implementación godot ofrece un sistema de señales (triggers) que nos permiten gestionar una serie de sucesos que pueden ocurrir en una conexión como són la desconexión de un cliente o un servidor o la conexión de uno nuevo o la conexión a una partida por parte de un jugador. Cuando estas señales se activan o disparan, podemos determinar una serie de métodos a ejecutar para cada una que se ejecutarán como un demonio (daemon).

Este sistema es muy óptimo e intentar implementarlo desde cero no tiene sentido así que se ha usado una solución que cumple exactamente con las necesidades del proyecto.

3.5 La sala de espera:

Otro elemento importante es la sala de espera. Este paso previo al juego se planteó desde un inicio viendo diferentes formas de abordarlo ya que todos los usuarios tienen que empezar el juego al mismo momento y en el caso de este proyecto no se admiten conexiones nuevas una vez se ha iniciado la partida. La implementación en este caso es muy simple y consta de una serie de pasos.

Cuando un cliente le da a join en el lobby, envía una conexión con el servidor que en este caso es un cliente y ya está en la sala de espera. Esta conexión establece un nodo player para el nuevo usuario así como establece la relación entre los diferentes players y sus propietarios y envía esta información a los clientes para que repliquen estos nodos en sus máquinas. De hecho se ha creado un nodo específico para tal propósito (Persistent_Nodes) donde guardamos las instancias de los nodos que se tienen que sincronizar como pueden ser balas, enemigos, objetos o los propios players junto con sus identificadores únicos de tal forma que podamos relacionar estos nodos entre los diferentes clientes para ejecutar acciones sobre instancias diferentes pero que en realidad són réplicas una de la otra o representan el mismo objeto o nodo. Una vez hecho esto el cliente se pone a la espera hasta que el cliente que hace de servidor decida iniciar la partida. Este servidor es el cliente que creó la partida y es el único que puede iniciarla en este caso, siendo una solución simple pero efectiva de abordar el inicio sincronizado del juego en todos los clientes.
Para ello envía una llamada remota a todos los clientes invocando el método iniciar partida.

Figura 32

3.6 Seguridad:

Este apartado se ha realizado a modo de investigación únicamente y sin llegar a entrar en muchos detalles se ha llegado a una conclusión por parte del equipo y sobre todo teniendo en cuenta la evolución del proyecto.
Hacer juegos multijugador conlleva enviar información a través de internet y por ello requiere de una serie de implementaciones que solucionen las posibles vulnerabilidades del sistema. Si bien Godot implementa la high level Networking API con una serie de facilidades para gestionar el envío de paquetes y en definitiva la sincronización y reconciliación de datos entre clientes y servidores, va a depender de cómo se estructure el código y el tipo de datos que transfieras donde se producen esta serie de vulnerabilidades. Más allá de intentar implementarlo y teniendo en cuenta que se trata de un nivel avanzado en cuanto a conexiones, se ha decidido hacer hincapié en la importancia de este punto en un futuro y por el momento se dará por hecho que el juego no es seguro y requerirá de más tiempo de investigación y pruebas.

3.7 Organización y gestión del trabajo:

A lo largo del proyecto se han realizado modificaciones en la organización del trabajo que han supuesto grandes cambios en cuanto a efectividad y eficiencia del equipo siendo el propio equipo el que ha visto la necesidad de estos cambios. por ello se ha decidido explicar en este punto en que ha consistido.

Después de algunos días de proyecto el equipo ha decidido reunirse para valorar cómo está funcionando el sistema de gestión del trabajo y se ha decidido hacer algunos cambios. A partir del 30 de marzo se aplica sobre la pizarra Kanban un modelo parecido a scrum con el que se pretende abordar cada semana como un sprint de la siguiente forma:

De lunes a jueves los desarrolladores tendrán trabajos programados que ir completando y a su vez durante estos días irán llenando una columna de la pizarra con tareas que vayan surgiendo para el próximo sprint a modo de “backlog”. Además se ha creado un diario donde se irán apuntando las tareas hechas por cada programador como una bolsa de horas además de ser un diario de tareas realizadas.

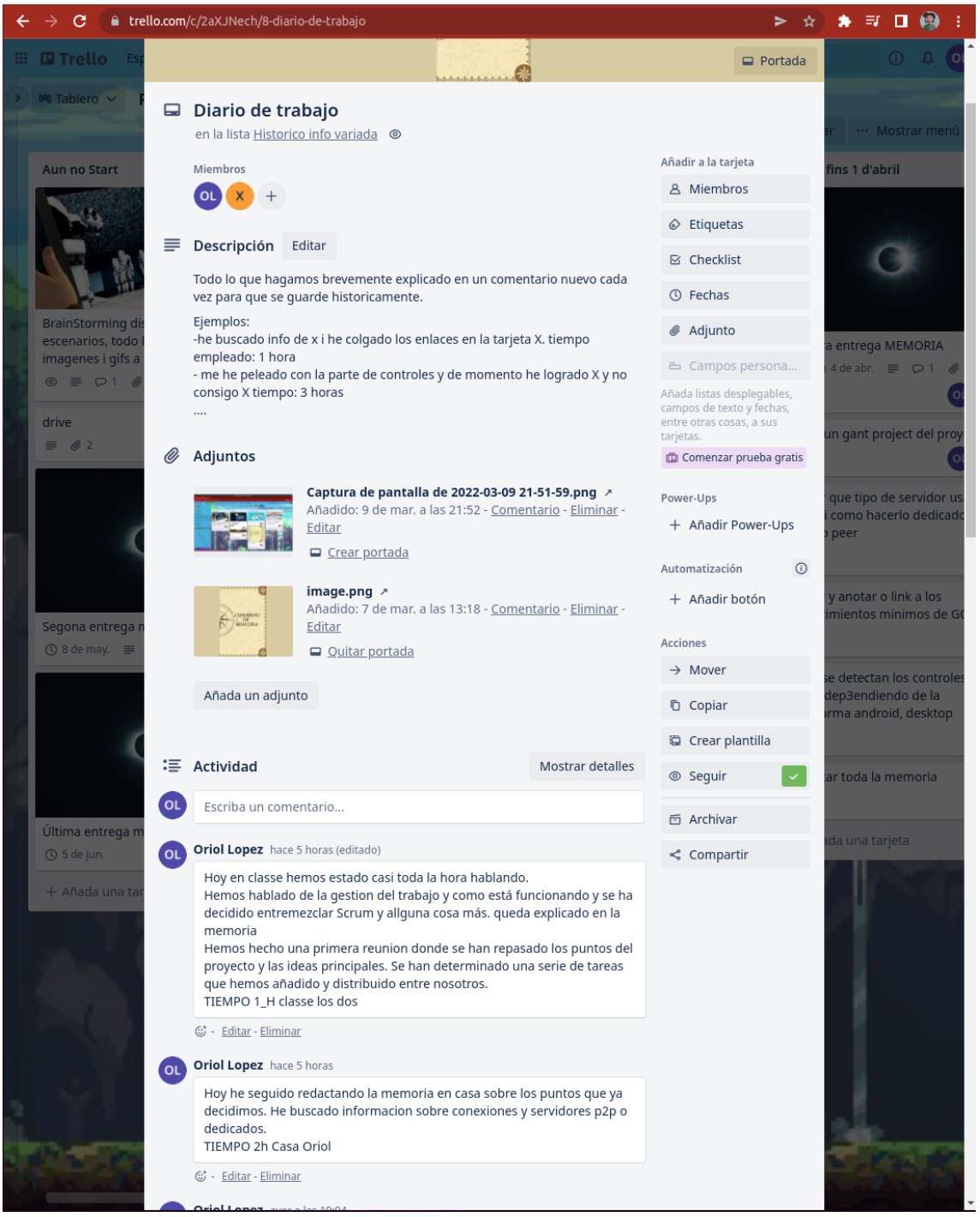


Figura 33

El viernes a primera hora se hará una reunión, “sprint review” y “sprint retrospective”, para valorar el sprint actual y se pensarán y valorarán las tareas

del próximo sprint teniendo en cuenta el modelo gantt inicial y todos los conocimientos nuevos adquiridos.
La última hora del viernes se harà otra reunión “sprint planning” y “sprint goal”, para desglosar estas tareas y distribuirlas entre los desarrolladores de tal forma que el lunes se tenga ya preparado el sprint incluso si se quiere avanzar trabajo el fin de semana este ya esté programado. Además cada día de trabajo se realizará un “daily scrum” o pequeña reunión para estar al día de la evolución del “sprint” y en consecuencia poder alterar los recursos dedicados a cada objetivo.

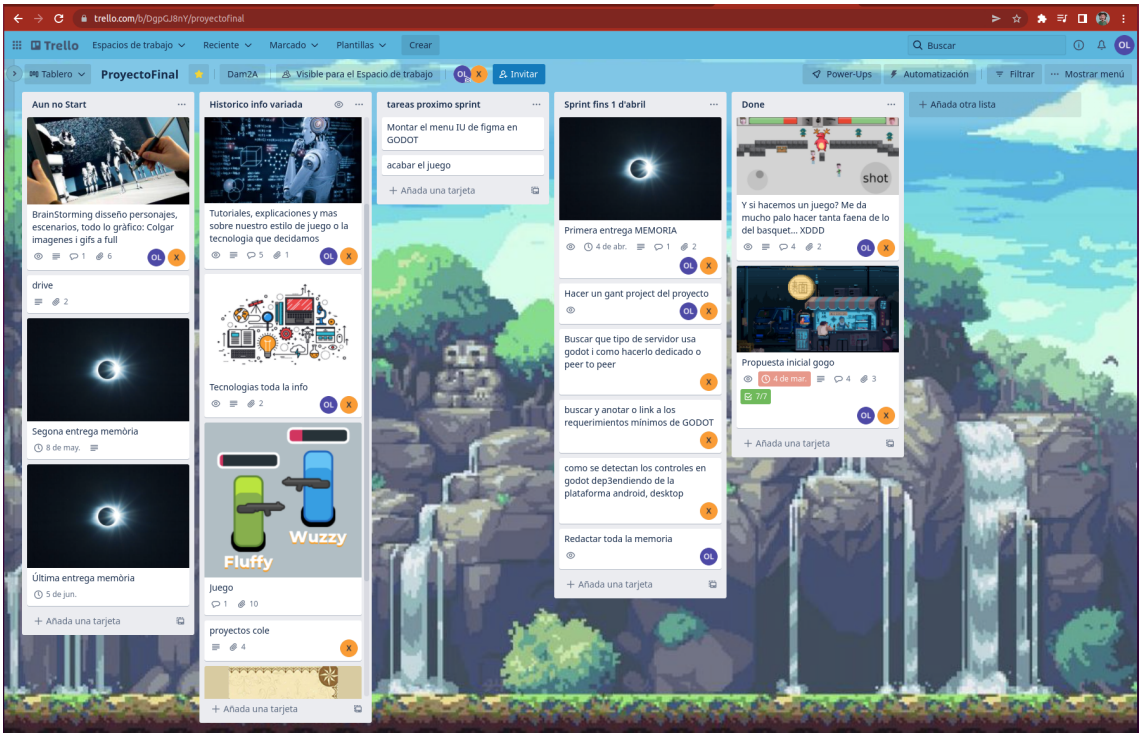


Figura 34

Por último se han definido unas horas de trabajo semanales que se corresponden con las horas de proyecto que tengamos en clase, y 4 horas más fuera de ese horario que se distribuirá cada desarrollador como decida. Con esto se pretende dar sentido a los sprints y poder hacer estimaciones de tiempos de trabajo más ajustados. También se pretende equilibrar las horas entre los desarrolladores ya que no todos pueden dedicarle las mismas horas al proyecto pero todos han aceptado estas horas mínimas. Por último este horario puede que varíe si los desarrolladores lo consideran oportuno, previa reunión.

4 Conclusions

4.1 Conclusiones generales del proyecto

Este proyecto ha puesto a prueba al equipo a todos los niveles. Ha supuesto un cambio de paradigma en cuanto al desarrollo e implementación de una estructura, muy distinto del tipo de programación al que estamos acostumbrados. Se ha usado un framework nuevo o por primera vez, un lenguaje nuevo, un tipo de juego con conexiones, triggers, interfaz, un sistema de programación por nodos, etc.
En definitiva el equipo se ha puesto al día en varios ámbitos nuevos a la vez y aunque también ha logrado superar cada obstáculo en forma de conocimiento adquirido, la sensación general que ha producido este proyecto al equipo es de haber arañado la superficie solamente pero a la vez ha aportado una visión global de todas las partes involucradas en la creación de videojuegos.

4.2 Consecución de los objetivos

Uno de los puntos clave planteados en este proyecto era comprender mejor cómo funcionan las conexiones en los juegos actuales y profundizar en ello y así se ha hecho aunque ha requerido mucho más tiempo entender la API de godot y se ha empezado a trabajar las conexiones a partir de la documentación de la API pero sin tener suficientemente claro como se trabaja en el mundo real y cómo se gestiona este traspaso de información es decir, que información es importante enviar o recibir en cada caso. Si bien ha costado bastante entender este funcionamiento, una vez comprendido se ha podido investigar mejor el funcionamiento en los juegos actuales lo cual ha sido muy enriquecedor.

El lobby es otro de los puntos importantes desarrollados ya que pone a prueba las conexiones pero en vista a la evolución que ha ido tomando el proyecto y si bien se ha podido profundizar en él incluso implementando diferentes formas, en el proyecto que se ha desarrollado se ha implementado una solución ya existente y que cumple los requisitos del proyecto para así poder trabajar en la conexión referente a la jugabilidad.

La IA es otro de los objetivos importantes para el juego y su jugabilidad. La implementación de los enemigos es donde se han puesto a prueba los conocimientos adquiridos sobre la API de conexiones y el funcionamiento de Godot como editor y muchas de las interfaces de que dispone. Esta implementación ha sido clave para comprender mejor el funcionamiento de este tipo de juegos online.

Por otro lado se han implementado y experimentado con muchos de los elementos relacionados con la creación de videojuegos como interfaz, menús, players, enemigos, colisiones, imágenes, etc. Por parte del equipo se ha visto un gran progreso y aprendizaje de todo ello y ahora ya se conoce mejor cómo funcionan los frameworks de creación de videojuegos.

4.3 Valoració de la metodologia i planificació

Si se compara la previsión inicial del trabajo en el diagrama de gantt con el diario de trabajo, no se ha tenido en cuenta todo el tiempo que se ha acabado invirtiendo en investigación. Sobretodo se ha dedicado bastante tiempo a la investigación y búsqueda de información de la documentación de godot, y también a pruebas usando forks a otros proyectos y probando diferentes sistemas de conexiones así como de implementaciones del movimiento sincronizado o de la IA de los enemigos y no tanto a implementar un juego o plantilla de juego como se pretendió en un inicio. En cualquier caso los tiempos se han respetado bastante y una vez determinada la estructura, la implementación ha sido rápida si bien se ha aprovechado código de diferentes proyectos que se han usado durante el transcurso de las pruebas y se ha simplificado alguna funcionalidad sobre todo a nivel gráfico. En resumen la valoración del equipo respecto al uso del tiempo es positiva.
Además como ya se explica en el punto sobre el desarrollo del proyecto se hicieron cambios en la metodología aplicada en cuanto a la gestión del proyecto y esta ha permitido una mayor implicación en las tareas del trabajo así como mayor rendimiento a la hora de tener claras las tareas de cada uno. De todas formas hay que decir que en algunos momentos del proyecto no ha sido casi necesario el uso de la pizarra, sobre todo por el hecho de que el equipo está en constante comunicación y en cambio se han hecho muy útiles incluso indispensables las reuniones propias de la metodología SCRUM.

4.4 Visión de futuro

El proyecto tal y como está ahora tiene mucho potencial y mucho trabajo todavía por implementar. Ahora que el equipo está ya introducido tanto en Godot como en las conexiones multiplayer se pretende seguir implementando el juego ya que se ha hecho la parte más dura del trabajo y tocaría comenzar a implementar más mapas, enemigos distintos, más armas para los players, en definitiva crear un juego al gusto. Seguramente no se va a seguir implementando sobre este esqueleto ya que ha servido más como modelo de pruebas que como un esqueleto en vista a un producto final.
Si bien se han investigado muchas partes de la implementación de videojuegos, todavía falta mucho por aprender tanto de Godot como por lo que respecta a las conexiones como a la implementación a nivel de diseño que

apenas se ha trabajado. Todo esto se quiere seguir investigando ya sea como hobby o a nivel profesional y si bien se ha priorizado el uso de software libre en este proyecto, este ha despertado un interés por conocer otros frameworks de edición y creación de juegos.

5. Glosario

Máster: Significa dueño, que es quien controla ese nodo.

Puppet: Significa marioneta, que es el nodo que está controlado por el master.

API: Application programming interface es un interfaz que se utiliza para ampliar más funcionalidades del network.

Lobby: Es una sala que podemos seleccionar a qué partida unirse

TCP: Transmission Control Protocol es un protocolo que se utiliza para asegurar que ha llegado el paquete al destino.

UDP: User Datagram Protocol es un protocolo que envía paquetes sin comprobar la integridad.

Framework: Es un marco de trabajo utilizado para facilitar el desarrollo de la aplicación.

IP: Protocolo de Internet es un conjunto de reglas que controla el formato de datos enviados a través de Internet o la red local y la dirección IP es un identificador que permite el envío entre redes.

6. Bibliografia

Documentación oficial de Godot Engine:
https://docs.godotengine.org/en/stable

Repositorios de ejemplo sobre conexiones de Godot:
https://github.com/godotengine/godot-demo-projects/tree/master/networking

Video de exportación de un servidor dedicado:
Exporting for dedicated servers

Documentación de MOG:
Videojuego multijugador en línea

Video de videojuego 2D top-down shooter:
https://www.youtube.com/watch?v=HycyFNQfqI0

Documentacion de The tale of the extended warranty:
https://vidvadgames.itch.io/the-tale-of-the-extended-warranty

Documentacion de videojuego multiplayer con Godot y Nakama:
https://heroiclabs.com/blog/tutorials/godot-fishgame/

Video de Godot Multiplayer Server dedicado:
https://www.youtube.com/watch?v=lnFN6YabFKg

Video de 2D Game con Godot:
https://www.youtube.com/watch?v=Mc13Z2gboEk

Video de 3D Top Down Shooter con Godot:
https://www.youtube.com/watch?v=WjThx-Bdn5gqI0

Video de Godot Multiplayer con High Level Networking:
https://www.youtube.com/watch?v=TGIWD24QIvY

Video de hacer una lista en Godot:
https://www.youtube.com/watch?v=_g9MnludJnw

Tutorial de Godot Networked Multiplayer Shooter:
https://www.youtube.com/watch?v=lpkaMKE081M

Documentación de autenticación con High-tech y Low-tech:
https://www.jesperjuul.net/text/independentstyle/

Documentación de conocimientos que tienes que saber sobre red de un juego:
https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/

GitHub de un juego con Servidor:
https://github.com/jlcmes/libGDX-Net

GitHub de un juego shooter:
https://github.com/hey-almas/godot-shotgun-party

Lenguaje de programar un videojuego:
https://www.tokioschool.com/noticias/lenguajes-programar-videojuegos/

Software para crear tu propio videojuego:
https://www.toulouselautrec.edu.pe/blogs/herramientas-crear-videojuego

Framework y librerías para desarrollo de juegos:
https://docs.hektorprofe.net/escueladevideojuegos/articulos/frameworks-librerias-bibliotecas-recopilacion/