UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Desarrollo de un videojuego educativo multijugador y su plan de marketing

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Davidescu Tanase, Stefanita Alin

Tutor/a: Golf Laville, Emilio Jesus

Cotutor/a: Linares Pellicer, Jordi Joan

Cotutor/a: Díaz-Madroñero Boluda, Francisco Manuel

CURSO ACADÉMICO: 2023/2024

Español

Título: Desarrollo de un videojuego educativo multijugador y su plan de marketing

Resumen:

La motivación tras este TFG es ofrecer un videojuego educativo que provea de una forma divertida de aprender conceptos de Estrategia de Operaciones, pudiendo ser usado tanto como introducción al material como para afianzar conceptos ya vistos. Mediante un estilo desenfadado, un diseño cuidadoso y la adición de minijuegos entre turnos, se pretende ofrecer una alternativa que sea al mismo tiempo amena y didáctica.

El juego se desarrollará en godot, un software gratuito y open-source, haciendo uso de una combinación de herencia y composición. El multijugador se realizará por conexión P2P, con uno de los jugadores actuando como servidor y arbitrando la partida (normalmente el profesor).

Para realizar el plan de marketing de dicho juego, realizaremos un estudio interno de la empresa, así como del entorno y los posibles competidores. Se realizarán encuestas y playtest, todo ello para culminar con los pasos a seguir para sacar adelante nuestro producto, y su viabilidad económica.

Palabras Clave: videojuego educativo, plan de marketing, estrategia de operaciones

Inglés

Title: Development of a multiplayer educational video game and its marketing plan

Summary:

The motivation behind this TFG is to offer an educational video game that provides a fun way to learn Operations Strategy concepts, and can be used both as an introduction to the material and to reinforce concepts already seen. Through a light-hearted style, careful design and the addition of mini-games between turns, the aim is to offer an alternative that is both enjoyable and didactic.

The game will be developed in godot, a free and open-source software, using a combination of inheritance and composition. The multiplayer will be done over a P2P connection, with one of the players acting as the server and refereeing the game (usually the teacher).

In order to make the marketing plan for the game, we will carry out an internal study of the company, as well as the environment and possible competitors. Surveys and playtesting will be carried out, all of which will culminate with the steps to follow in order to develop our product and its economic viability.

Keywords: educational video game, marketing plan, operations strategy.

Valencià

Títol: Desenvolupament d'un videojoc educatiu multijugador i el seu pla de màrqueting

Resum:

La motivació després d'aquest TFG és oferir un videojoc educatiu que proveïsca d'una forma divertida d'aprendre conceptes d'Estratègia d'Operacions, podent ser usat tant com introducció al material com per a afermar conceptes ja vistos. Mitjançant un estil desenfadat, un disseny acurat i l'addició de minijocs entre torns, es pretén oferir una alternativa que siga al mateix temps amena i didàctica.

El joc es desenvoluparà en godot, un programari gratuït i open-*source, fent ús d'una combinació d'herència i composició. El multijugador es realitzarà per connexió P2P, amb un dels jugadors actuant com a servidor i arbitrant la partida (normalment el professor).

Per a realitzar el pla de màrqueting d'aquest joc, realitzarem un estudi intern de l'empresa, així com de l'entorn i els possibles competidors. Es realitzaran enquestes i playtest, tot això per a culminar amb els passos a seguir per a tirar avant el nostre producte, i la seua viabilitat econòmica.

Paraules Clau: videojoc educatiu, pla de màrqueting, estratègia d'operacions

Índice

Introducción **5**
Descripción General 5
Motivación 5
Metodología 6
Análisis externo **7**
Macroentorno 7
Político 7
Económico 7
Social 8
Tecnológico 8
Ecológico 8
Legal 9
Microentorno 9
Poder de negociación de los clientes 10
Poder de negociación de los proveedores 10
Amenaza de productos sustitutivos 10
Amenaza de nuevos competidores 11
Rivalidad entre competidores existentes 11
Análisis interno **12**
Análisis Dafo **13**
Debilidades 13
Amenazas 13
Fortalezas 13
Oportunidades 13
Estrategia Comercial **14**
Estrategia de producto 14
Estrategia de precio 14
Estrategia de distribución 15
Estrategia de comunicación 15
Diseño del juego **15**
Diseño lógico 16
Objetivo del juego 18
Minijuego: AppleCatch 18
Establecimiento de prioridades **19**
Primera Fase: Demo Funcional 19
Segunda Fase: Demo Completa 20
Tercera Fase: Producto Mínimo Viable 20
Implementación del juego **21**
Resumen de Godot 21
Árboles de nodos 21
Autoload 23

Métodos base 23
Señales 24
Colisiones 24
User interface 25
Multijugador en Godot 26
Estructura de carpetas 28
Estructura de directorios en godot 28
Estructura del proyecto 28
Arquitectura del código 31
Main 31
MainUI 31
Global 33
Máquina de Estados 33
Configuración 33
Game 34
Turno 34
Demanda 35
Compañía 36
Pantallas 38
Áreas 41
Flujo general del juego 42
Estructura de minijuegos 47
Elementos comunes 47
Flujo 48
Apple Catch 49
Escena Base 50
Fruta 51
Personaje del jugador 53
Objetivos de desarrollo sostenible **56**
4 - Educación de calidad 57
11 - Ciudades y comunidades sostenibles 58
Conclusiones **58**
Anexo A: Tablas e imágenes **59**
Anexo B: Bibliografía **61**

Introducción

Descripción General

La intención de este Trabajo de Fin de Grado es la de desarrollar un juego didáctico multijugador, que pueda ser usado para enseñar materia de Estrategia de Operaciones pero con la posibilidad de adaptarse a otras áreas de administración y dirección de empresa, como podrían ser marketing, gestión de almacén, recursos humanos, etc.

Este juego se desarrollará con la intención de que tenga una gran personalización, pudiendo modificar según se necesite los distintos elementos de una partida, y permitiendo crear nuevas opciones en caso de que las que se proporcionan con el juego base no sean suficientes.

Se pretende también que además de cumplir con su propósito educativo, el juego sea ameno y ofrezca una divertida experiencia de competición y trabajo en equipo. A lo largo de una partida, los jugadores formarán grupos de una empresa cada una, y trabajarán juntos para tomar las mejores decisiones para su empresa y hacer frente a los cambios del mercado y a las demás empresas con las que compiten.

Se realizará también el plan de marketing de dicho videojuego, a fin de estudiar en detalle cuál sería la mejor forma de poner en práctica este juego como idea de negocio, ver nuestra situación frente a la competencia, estudiar nuestro público objetivo, y en base a todo esto tomar las decisiones pertinentes para dar a conocer nuestro producto al mercado.

Motivación

La principal motivación de realizar este juego como mi trabajo de fin de grado es proporcionar una herramienta que pueda ayudar a afianzar y ampliar conocimientos de Estrategia de Operaciones y otras materias similares.

Cuando uno piensa en juegos educativos, lo primero en lo que piensa es en juegos dirigidos a niños, con la finalidad de enseñar sobre temas simples como matemáticas o lengua, y con mecánicas aún más sencillas. Debido a su público objetivo, no pueden permitirse apenas complejidad en sus sistemas y mecánicas, lo cual hace que el juego pueda ser repetitivo y aburra con facilidad.

En juegos de este tipo dirigidos a adultos, se puede observar una tendencia hacia la simulación excesiva: Se pone demasiado énfasis en la materia que se pretende enseñar, lo cual lleva a que estos se parezcan más a un simulador que a un videojuego, sobrecargando con datos complejos al jugador y descuidando el gameplay.

Con este juego, tenemos la oportunidad de combinar lo mejor de ambos mundos: abstraer las partes más densas para agilizar la experiencia de juego, centrándonos en los puntos más importantes de la materia a impartir y en las decisiones que se deben tomar sobre ésta, e introduciendo varios minijuegos que añadan otro nivel de competitividad y contribuyan a hacer de éste un juego divertido, que no se juegue sólo por su valor educativo sino también por la experiencia que provee.

Metodología

El desarrollo del videojuego se realizará utilizando Godot, un motor de juegos gratuito y open source que por las herramientas y opciones que contiene se presta a estructurar su código haciendo uso de componentes.

Para este desarrollo, primero se establecerán los requisitos: las funcionalidades que ha de tener el juego, las pantallas por las que ha de pasar, y en general el funcionamiento deseado del juego. Después, se identificarán las funcionalidades más críticas a fin de establecer prioridades en las tareas a realizar y dividir el trabajo en varias fases.

Finalmente, se procederá a la implementación en Godot, detallando los distintos elementos que componen el videojuego, sus interacciones y su organización. Se hará uso de un repositorio de Git para compartir el código base del proyecto, así como para tener un back-up en la nube. Se tendrá también un back-up físico en el disco de otro ordenador.

Respecto al plan de marketing, se realizará primero un análisis externo, tanto del macroentorno como del microentorno, para tener una imagen global de todos los factores externos que puedan afectar a nuestra empresa.

Después se procederá con el análisis interno, y se realizará un análisis DAFO (Debilidades, Amenazas, Fortalezas y Debilidades) para ilustrar de forma clara y concisa lo que hayamos visto en los anteriores análisis. Una vez hecho esto, se elaborará lo referente al posicionamiento de la empresa, la estrategia comercial y el marketing mix.

Análisis externo

En este apartado se estudiará el estado del entorno en el que pretendemos lanzar nuestro producto, a fin de observar tanto la situación generalizada del mercado y de nuestros posibles clientes como la de la competencia a la que haremos frente.

Macroentorno

Para el análisis del macroentorno, realizaremos un análisis PESTEL, con el cual obtendremos una vista clara de las posibles oportunidades y amenazas a las que nos enfrentaremos.

Político

En política internacional, este año ha estado marcado por fuertes tensiones internacionales. La guerra en Ucrania, que comenzó el 24 de diciembre de 2022, sigue en su apogeo, y las tensiones en oriente próximo no hacen más que aumentar. Esto contribuye a que el clima internacional sea uno de inestabilidad y miedo.

Dentro de nuestro propio país, la situación también es de incertidumbre. En 2020 las elecciones acabaron con una precaria victoria del PSOE mediante una coalición con Unidas Podemos, y el gobierno de Pedro Sánchez terminará en diciembre de este mismo año. Mucha gente ha expresado su descontento con las políticas que ha realizado, y es imposible saber a ciencia cierta qué partido cogerá el relevo y se alzará con la presidencia en las nuevas elecciones.

Económico

En el ámbito económico, el principal tema a tratar es la inflación. Los cálculos del INE sobre la inflación anual estimada de este año apuntan al 3,6%, muy por encima del 2% que tiene como objetivo mantener el Banco Central Europeo.

Las causas de esta inflación son complejas, pero principalmente se corresponden con una falta de recursos como el gas natural o el petróleo causada por la situación con Ucrania.

A fin de hacer frente a la inflación, el Banco Central Europeo decidió subir los tipos de interés hace ya dos años, hasta alcanzar el 2.5%, y continuó subiéndolos progresivamente hasta alcanzar el valor que tiene actualmente, 4,25%. Esto tiene por objetivo reducir la inflación al incrementar el coste de los préstamos debido al más elevado tipo de interés, lo cual puede perjudicarnos al hacer más costoso financiarnos mediante préstamos.

Respecto al PIB, ha estado experimentando una subida lenta pero constante después de la crisis causada por el COVID-19, y actualmente se encuentra en una cifra récord de 1,4 billones de euros.

Social

Internet y las redes sociales han tenido un impacto significativo en la manera en que percibimos nuestra vida y nos comunicamos los unos con los otros. Hoy en día tenemos acceso a más información que nunca a través de internet, la información viaja a velocidades nunca antes vistas y tenemos disponibles enormes cantidades de conocimiento, desde tutoriales a libros y ensayos en forma de vídeo.

Pero esto también tiene su lado negativo. El enorme flujo de información al que nos vemos sometidos puede resultar abrumador a veces, y hace difícil comprobar que dicha información sea veraz, facilitando que se genere desinformación.

El flujo de datos que se maneja es tan grande que es necesario utilizar complejos algoritmos para manejarlos y tratar la información que se debería mostrar a un usuario, pudiendo estos algoritmos ser explotados una vez se conoce su funcionamiento, de forma similar a como se optimiza la posición de una página web en un motor de búsqueda mediante SEO.

Otro efecto negativo de esto es la reducción de nuestra capacidad de atención, sobre la cual se han realizado numerosos estudios. Un estudio de la Universitat Oberta de Catalunya, en colaboración con Accenture, ha observado que la capacidad de atención promedio se ha reducido en cerca de un 33% en menos de 15 años. Esto puede presentar un problema, ya que una reducida capacidad de atención puede afectar negativamente a la capacidad de aprender y retener información en un ámbito académico.

Tecnológico

En los últimos años ha habido un increíble boom tecnológico en el ámbito de la inteligencia artificial. Esta tiene infinidad de aplicaciones, desde tratamiento de texto y traducción a creación de imágenes o música.

Esto posibilita automatizar multitud de procesos de formas que antes no eran posibles, siendo el avance de la inteligencia artificial uno de los más destacables avances tecnológicos de esta década, y la principal tecnología disruptiva a día de hoy.

Ecológico

En el ámbito ecológico, España se encuentra en una posición favorable. Más del 60% de la generación de energía eléctrica en el país proviene de energías renovables, y es uno de los países con más superficie forestal de la Unión Europea, estando el 40% de dicha superficie en zonas protegidas.

En 2020 España ocupó el puesto 27 en la lista de países más contaminantes, representando un 0,58% de las emisiones totales con 213 millones de toneladas de CO2, lo cual equivale a aproximadamente un tercio de las emisiones de Alemania, y es inferior que las emisiones de Francia, Reino Unido e Italia en el mismo año.

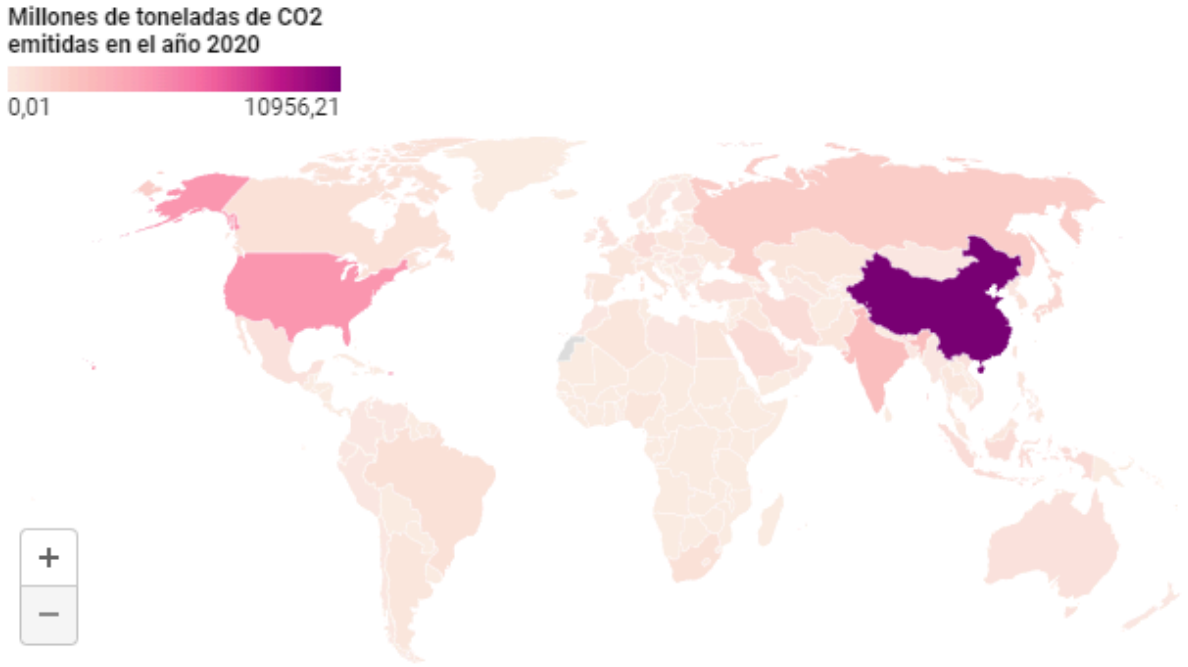

Figura 1: Mapa mundial con las emisiones totales de CO2 a la atmósfera, organizados por color.
Fuente: https://www.eldebate.com/sociedad/20221107/espana-solo-provoca-0-58-contaminacion-mundial-51-veces-menos-china_71049.html#:~:text=España%20ocupó%20el%20puesto%2027,de%20CO2%20emitidas%20por%20persona.

Legal

En lo que respecta al ámbito legal, lo más importante a destacar son las muchas ayudas y subvenciones que ofrece el gobierno español.

Debido a ser una empresa de nueva creación, es posible aspirar a varias ayudas y beneficios, tales como los establecidos por la ley de startups que se aprobó en 2023, entre

los cuales cabe destacar una reducción del tipo impositivo al 15% mientras se mantenga la condición de startup.

También existen multitud de iniciativas dedicadas a financiar nuevas startups, como Activa Startups, Líneas Enisa o el Programa Neotec.

Otro aspecto importante a mencionar es que el producto que pretendemos ofertar tiene un fin formativo y educativo, y gastos en dicho producto por parte de una empresa son desgravables, al tratarse de formación.

Microentorno

En este apartado realizaremos un análisis del microentorno, estudiando con atención a nuestros posibles competidores y sus productos para poder hacernos una idea general de la situación del mercado. Para ello, analizaremos las 5 fuerzas de Porter en relación con nuestro producto, simuladores empresariales con fines formativos.

Poder de negociación de los clientes

En este caso, tendremos principalmente dos tipos de clientes: centros formativos, como pueden ser universidades o centros de formación profesional, y empresas que busquen formar a sus trabajadores en ámbitos de gestión empresarial.

Puesto que existe una variada oferta de simuladores empresariales, y pasar de usar uno a otro no requiere grandes cambios, el cliente tiene un amplio poder de negociación. Por otro lado, es importante tener en cuenta que el hecho de que los gastos en formación sean desgravables por parte de la empresa resulta en que el cliente sea menos flexible al precio.

Poder de negociación de los proveedores

Al tratarse mayoritariamente de un producto de software que oferta una empresa en particular, a efectos prácticos no tenemos proveedores, ya que la propia empresa gestionaría todos los pasos previos a la entrega del producto, sin depender de terceros.

Una excepción notable de esto se da en el caso de que el servicio que se ofrezca se ejecute en la nube o en un servidor externo, en cuyo caso se habría de considerar al proveedor de dicho servicio.

Puesto que existen multitud de proveedores con gran variedad de equipos disponibles, así como distintas modalidades de pago, desde un pago fijo por equipo hasta un pago por hora que está el servidor encendido, podemos deducir que su poder de negociación es bajo debido a la abundancia de oferta. Es importante mencionar que una vez se esté usando un determinado servicio, cambiar a otro presenta una barrera de entrada importante, al necesitar de una migración durante la cual podría ser difícil mantener el servicio que se ofrece al cliente.

Amenaza de productos sustitutivos

Al tratarse de un producto de formación, el primer producto sustitutivo sería otro tipo de formación, como la que puede darse en un aula o mediante cursos online. Sin embargo, estos no son mutuamente excluyentes con nuestro producto, puesto que también podría ser un producto complementario, siendo uno de estos simuladores una parte más del currículum de una asignatura o un curso.

También se habría de considerar como productos sustitutivos otros medios de formación, como pueden ser medios sobre la materia o vídeos explicativos.

En la medida en que una simulación de empresa contribuye a cimentar conocimientos de la materia y a fomentar el trabajo en equipo, también se ha de tener en cuenta ciertas actividades y dinámicas de grupo, como puedan ser los juegos de rol.

Amenaza de nuevos competidores

En términos generales, se observa que existen multitud de competidores en el mercado, pero ninguno de ellos tiene una fama significativa hasta el punto de poder decir que está significativamente por delante del resto.

Los costes principales no son demasiado grandes, siendo en mayor medida el desarrollo del software para el simulador, su marketing y su distribución. Esto, sumado a las subvenciones y beneficios que se han mencionado anteriormente, hacen que las barreras de entrada en el mercado sean relativamente bajas, lo cual posibilita la fácil entrada de nuevos competidores, creando con ello un mercado saturado.

Rivalidad entre competidores existentes

Debido a que el mercado está repleto de simuladores empresariales de todo tipo, podemos deducir que se trata de un mercado saturado, con una competición fiera y en el que sin embargo pocas empresas logran diferenciarse debidamente.

Figura 2: Logo de Cesim Business Simulations.
Fuente: https://www.cesim.com

Por ejemplo, existe el caso de Cesim, una empresa que oferta simulaciones empresariales diversas. Con más de medio millón de usuarios, sus principales puntos fuertes son su amplio abanico de simuladores, desde marketing y pequeñas empresas hasta simuladores de comercio internacional, y sus muchas reseñas de clientes satisfechos.

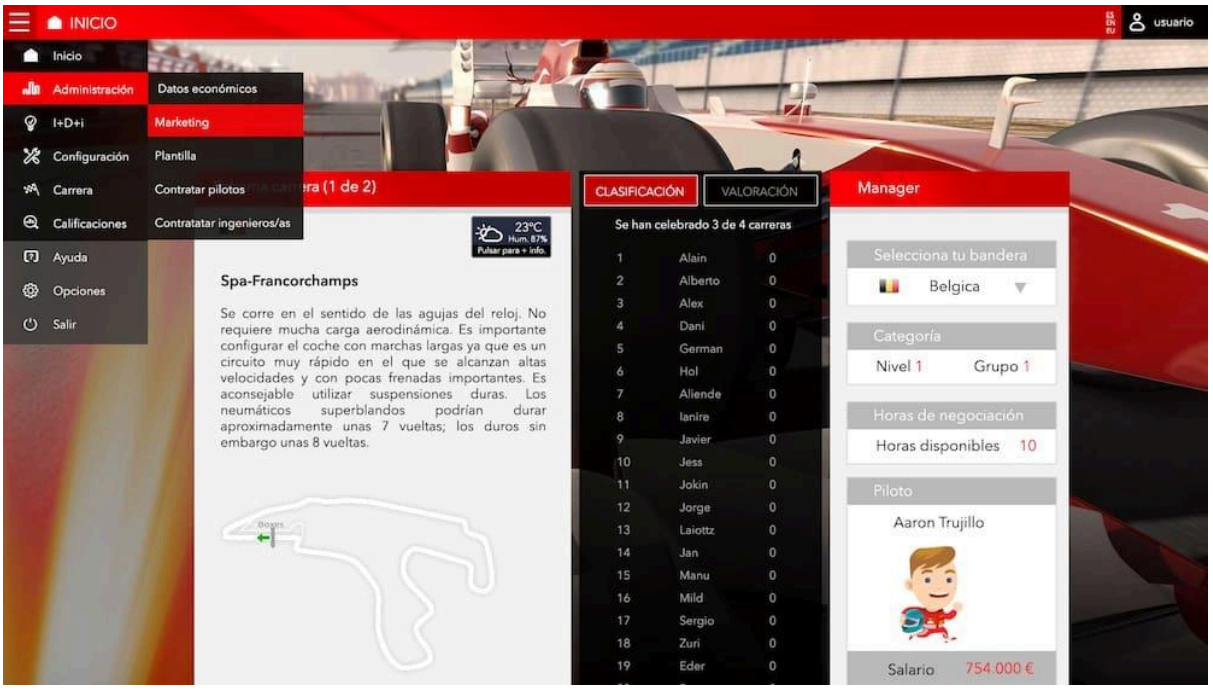

Figura 3: Captura de pantalla de la simulación de GestioNet “Innovación GP”.
Fuente: https://simuladores-empresariales.com/innovacion-gp/

GestioNet es otra empresa que opera en el mismo sector, intentando diferenciar su producto alterando el escenario base para presentarlo como algo fresco: en lugar de una empresa se trata de gestionar una isla, o una escudería de Fórmula Uno, pero el grueso de la experiencia de juego sigue siendo tomar una serie de decisiones a lo largo de varios menús.

Análisis interno

En este apartado realizaremos un análisis interno de nuestro producto, a fin de identificar fortalezas y debilidades.

Uno de nuestros puntos fuertes es el amplio grado de personalización de una partida, pero esto también implica al usuario final de manera que depende en parte de él configurar correctamente una partida, puesto que al poder acceder el usuario directamente a los ficheros de ciertos componentes puede provocar fallos si los modifica sin tener en cuenta el correcto funcionamiento del programa. Esto no debería ser un gran problema, puesto que junto con el juego se incluirá documentación explicando lo que se debe y no se debe hacer, pero sigue siendo un posible punto de fallo que pueda requerir soporte técnico.

Según se ha observado en los simuladores que ofrece la competencia, todos ellos claman ser divertidos y ofrecer una experiencia de aprendizaje amena, pero sin embargo prácticamente todos siguen una fórmula muy similar, bombardeando al usuario con menús y tablas sin ofrecer más elementos que puedan contribuir al disfrute del jugador.

Con este juego se busca ocupar ese nicho. Al simplificar y abstraer parte de la gestión de la empresa se prevee reducir la carga que tendrá el jugador, y cada periodo de decisiones y menús irá seguido de un minijuego competitivo en tiempo real, lo cual permitirá al jugador divertirse y competir, y así afrontar con más ganas las partes que de otra forma serían tediosas del juego.

Esto puede conllevar que se pierda algo de fidelidad en la simulación, pero lo que se busca es enseñar los conceptos base de una forma amena, más que hacer una simulación perfectamente realista. Otro posible problema al que podemos enfrentarnos es que nuestro producto sea percibido como menos serio por el simple hecho de incluir minijuegos y una estética más distendida.

Análisis Dafo

En este apartado realizaremos un análisis DAFO del entorno, para facilitar la toma de decisiones respecto a nuestro plan de marketing.

Debilidades

- Menos fidelidad en la simulación que la competencia
- El usuario puede provocar fallos con una configuración errónea
- Posibilidad de ser percibido como menos serio

Amenazas

- Inflación y tipos de interés altos
- Tensiones crecientes en Oriente Medio
- Mercado saturado de competidores

Fortalezas

- Producto más divertido que la competencia
- Amplia personalización de las partidas

Oportunidades

- Nicho de mercado de un simulador empresarial más cercano a un videojuego
- Subvenciones y ayudas para startups
- Gastos en formación desgravables, lo cual hace que el cliente pueda estar dispuesto a pagar más
- Hay tendencia hacia una menor capacidad de atención, para lo cual nuestro producto ofrece una mejor alternativa a otros simuladores más tediosos
- Innovación en inteligencia artificial permite nuevas vías de automatización

Estrategia Comercial

A lo largo de este apartado se definirá la estrategia comercial que seguirá nuestro plan de marketing, acorde a los datos observados en el análisis interno y externo, y enfocado en torno a los componentes del marketing mix.

Estrategia de producto

Nuestro producto ocupa un nicho sin explotar en el mercado, el de un simulador de empresa con más elementos de un videojuego, con un énfasis especial en la diversión sin dejar de lado el aprendizaje.

Se pretende ofrecer una experiencia más amena y accesible, logrando así una fuerte diferenciación de la competencia. Además, el amplio rango de personalización que permite cada partida posibilita que el juego se pueda adaptar sin problema a las necesidades de nuestro cliente.

Más adelante, se planea aprovechar la estructura base del juego para crear versiones más específicas, a fin de ofertar simuladores variados que puedan cubrir un amplio abanico de escenarios, manteniendo todos ellos nuestro objetivo de que el juego sea tanto educativo como divertido.

Para facilitar el uso del juego por parte del cliente, daremos también soporte técnico, siendo la mayor parte de este soportado por un chatbot entrenado para hacer frente a los problemas más comunes que pueda enfrentar un cliente, como pueden ser dudas generales sobre el juego o instrucciones para modificar la configuración o crear nuevos elementos.

Estrategia de precio

Teniendo en cuenta que nuestros costes serán bajos, gracias al hecho de no necesitar servidores debido al self-hosting del juego y a que el videojuego será desarrollado como parte de este trabajo de fin de grado, podremos ofrecer un precio competitivo, entre un 20 y un 30 por ciento por debajo de nuestra competencia.

Respecto al modelo de pagos, se trata de una suscripción anual, ofreciendo varios paquetes. El pack básico incluirá el juego para hasta 20 equipos, así como acceso al soporte técnico.

El pack real será ligeramente más caro que el básico, incluirá soporte para más equipos, y tendrá una ventaja especial: mientras se mantenga esta suscripción, se obtendrá acceso al contenido nuevo que vayamos sacando al mercado, sin coste adicional. Con ello se busca fidelizar a los clientes, ya que tienen nuevo contenido asegurado si mantienen su suscripción.

El pack imperial será el más caro, e incluirá la mayor cantidad de contenido, poder instalar el juego en equipos ilimitados, y la opción de recibir un soporte técnico más personalizado por parte de un miembro del equipo.

Con estos precios, se busca presentar el pack real como el más ventajoso, al ofrecer este grandes ventajas por apenas un poco más de lo que cuesta el pack básico.

Estrategia de distribución

La distribución del juego se realizará desde nuestra página web, en la que también se podrá encontrar el chatbot de soporte técnico.

Se planea ofrecer el juego de manera gratuita a varias universidades. Esto tiene un doble propósito. Por un lado, será posible obtener un muy valioso feedback de estos primeros clientes. Por otro, si el producto les gusta estaremos ya captando clientes y fomentando que nuestro producto se dé a conocer mediante el boca a boca, sin necesidad de una mayor inversión en anuncios.

También existen planes para hacer una competición anual del juego, abierta a todos los usuarios, en la que se repartirán diversos premios. Con este evento conseguiremos involucrar a la comunidad y generar publicidad para nuestro juego.

Estrategia de comunicación

En lo que respecta a la estrategia de comunicación, nuestros canales principales serán redes sociales y nuestra página web.

Tendremos una presencia en todas las redes sociales más importantes: X, Facebook, Instagram, y Linkedin. En estos canales buscamos establecer contacto con nuestra base de usuarios, generar imagen de marca y obtener feedback de los jugadores y clientes.

Nuestra página web tendrá tres propósitos principales. En primer lugar, es donde realizaremos la distribución del producto, por lo que tendrá toda la información necesaria para responder a las posibles dudas de nuestros clientes.

En segundo lugar, servirá para dar publicidad, publicando en ella las reseñas de nuestros clientes y los resultados de los eventos y competiciones que se realicen.

Por último, la página web será el lugar donde los usuarios podrán obtener soporte técnico, ya sea con nuestro chatbot personalizado, o por un miembro del equipo técnico en caso de tener el pack imperial.

Diseño del juego

En este apartado se planteará el funcionamiento deseado del juego, relacionándolo con la materia que se busca impartir, y se definirán los requisitos necesarios de cara a su implementación.

Diseño lógico

El juego ha de ser multijugador, distinguiendo entre dos tipos de jugadores: el Game Master, y los Jugadores.

El Game Master será el encargado de arbitrar la partida, pudiendo ver en todo momento los datos de los jugadores a fin de guiarles en las decisiones que van tomando. Los jugadores podrán pertenecer a uno de cuatro roles (Gerente, Fábrica, Marketing), cada rol tendrá unas responsabilidades diferentes en la gestión de la empresa.

Gerente: Se encargará de tomar las decisiones principales de la empresa, tales como el producto que van a vender, sus redes de suministro y proveedores, sus almacenes…

Fábrica: A cada turno decidirá cuántas unidades de producto se han de producir, y tendrá acceso a informes de costes.

Marketing: A cada turno decidirá a qué precio se han de vender las unidades, y tendrá acceso a informes de ingresos.

La intención con esto es que cada rol tenga datos limitados, que con esos datos por sí solos no puedan tomar decisiones, pero que si los ponen en conjunto y hacen algunos cálculos puedan llegar a obtener información útil que les ayude en la toma de decisiones.

Antes de comenzar la partida, los jugadores deberán tomar una serie de decisiones respecto a los componentes principales de su empresa: Producto, Proveedor, Almacenaje, Líneas de Suministro… El gerente será el encargado de tomar esta primera decisión, la del Producto, que tendrá el mismo nombre que el de la empresa a la que pertenezca. Para ver claramente el impacto de estas decisiones, habrá dos pantallas que lo muestren.

La primera es la del Canvas Business Model, en el que se mostrará cada área del Canvas, y a medida que se tomen esas decisiones, una barra de progreso en cada apartado se incrementará, hasta alcanzar el 100% en todos los apartados.

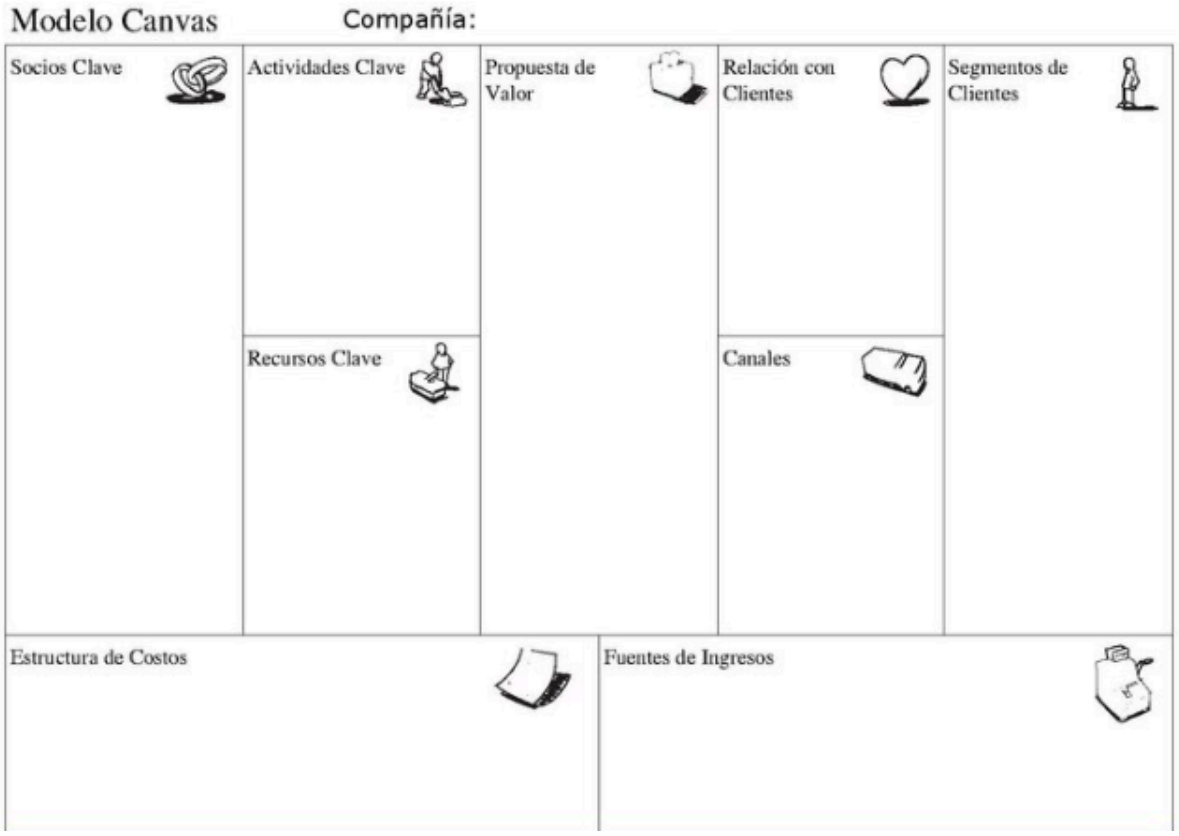

Figura 4: Modelo Business Canva.
Fuente: https://es.wikipedia.org/wiki/Archivo:Plantilla_CANVAS.pdf

La segunda es la matriz de Slack & Lewis, donde se iluminarán con un color distinto las intersecciones entre los elementos de la matriz, variando desde el naranja hasta el verde en función de qué valor tengan. Cada componente que tenga la empresa modificará esto en mayor o menor medida. Las áreas de decisión (Capacidad, Redes de suministro, Tecnología de procesos, y Desarrollo y organización) las usaremos para indexar los diferentes elementos que componen la empresa, mientras que las cualidades de dichas áreas (calidad, velocidad, fiabilidad, flexibilidad y coste) serán usadas en varios cálculos, siendo el más importante de entre estos el de las ventas que se realizan durante un determinado turno.

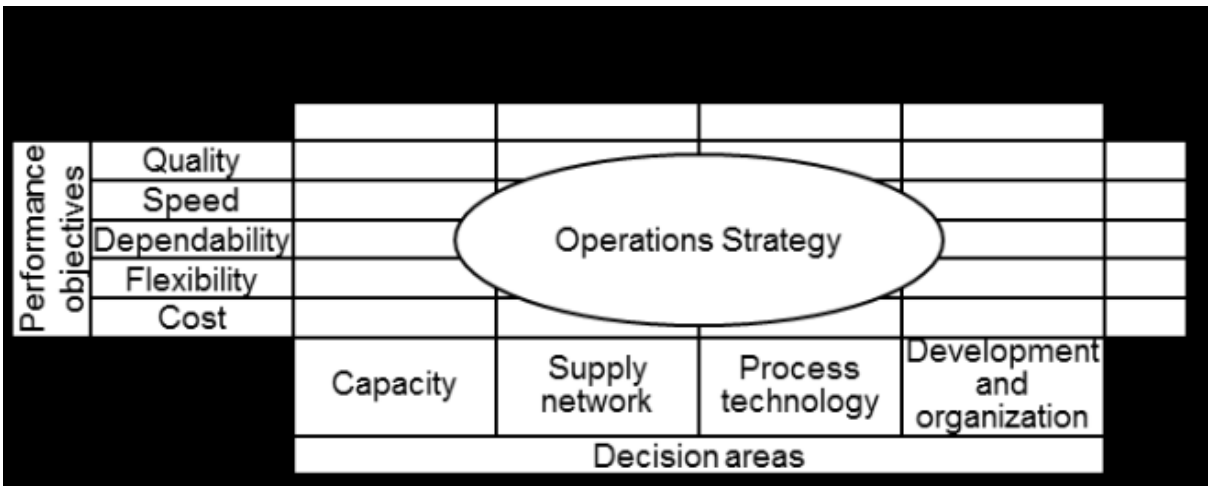

Figura 5: Matriz de Slack y Lewis.
Fuente:
https://www.researchgate.net/figure/Operations-Strategy-matrix-Slack-Lewis-2011_fig1_255820198

Objetivo del juego

El objetivo principal por el que competirán los equipos es por lograr la puntuación más alta. Hay diferentes factores que puedan afectar a la puntuación, pero el más importante es el beneficio. Se les mostrará a los jugadores información de ventas, gráficos de ventas y del calculo de punto muerto, pero en ningún momento podrán ver directamente su margen ni cantidad de beneficios. Con esto se busca que sean ellos quienes puedan calcular estas cifras, habiendo recibido la información necesaria, entremezclada con otros fragmentos de información que puedan ser menos relevantes.

Para generar sus ingresos, cada empresa venderá un producto determinado, y a cada turno deberá decidir cuántas unidades quiere producir (poniendo todas las unidades disponibles a la venta) y a qué precio las va a ofertar.

Su capacidad de producción, al igual que el rango de precios al que puede vender, vendrá dado por las elecciones que haya tomado a lo largo del juego, obteniendo un rango en el que podrán tomar dicha decisión.

Para realizar el cálculo de cuántas unidades vende cada empresa, primero hemos de tener en cuenta la demanda. Cada turno tendrá una demanda específica, que tendrá una cantidad máxima de ventas posibles, es decir el mercado total, y unos pesos que se corresponden a las cualidades de la matriz de Stack & Lewis: Calidad, Coste, Velocidad, Fiabilidad, y Flexibilidad.

Para cada producto de cada empresa, se mutiplicarán las cualidades de su producto por estos pesos para obtener su coeficiente, y al dividir dicho coeficiente por la suma de todos los demás coeficientes se obtendrá el porcentaje de ventas de dicho producto. Con esto podemos obtener las unidades vendidas, y realizar los cálculos pertinentes a costes, ingresos, e inventario final.

Minijuego: AppleCatch

Este será el primer minijuego a implementar, con la intención de que los jugadores puedan competir entre sí de forma más directa y simple, ofreciendo esto un respiro de las partes más complejas del juego.

En este minijuego cada jugador manejará un personaje, que se moverá por el mapa para recoger las frutas que caen del techo. Cada fruta dará dos puntos, pero también aparecerán de vez en cuando piedras, que restarán dos puntos si se recogen.

Además de tener movimiento horizontal y un salto, los jugadores tendrán una habilidad, que aumentará en gran medida su velocidad de movimiento y su altura de salto durante unos

pocos segundos. Una vez se haya usado, se tendrá que esperar 10 segundos hasta que se pueda usar de nuevo.

Este minijuego tendrá una duración de 60 segundos, mostrando en pantalla tanto el tiempo restante como las puntuaciones de cada equipo.

Al final de este minijuego se mostrará una pantalla con las puntuaciones obtenidas, indicando el equipo ganador.

Establecimiento de prioridades

Es vital para el éxito de todo proyecto el poder diferenciar entre aquellas tareas que son vitales para la realización del proyecto, ya sea porque tienen funcionalidades clave o porque otras dependen de ellas, y aquellas otras tareas que pueden realizarse más tarde, o que no son estrictamente necesarias. Esto permite un uso más eficiente del tiempo y una mejor organización del proyecto en general.

En este caso, tenemos principalmente dos prioridades. La primera de ellas es obtener una demo funcional, que incluya los requisitos más importantes y pueda ser jugada de principio a fin. La segunda prioridad es asegurar que el proyecto sea escalable.

Esta escalabilidad tiene dos aspectos. Por un lado, es necesario que el proyecto sea escalable a nivel de cuántos usuarios pueda soportar. Esto se conseguirá haciendo que el hosting del juego lo realice el propio cliente, con lo cual no será necesario establecer servidores y nuestra base de usuarios podrá crecer independientemente de la infraestructura que se tenga, incrementando solamente el tiempo que se pueda dedicar a dar soporte.

Por otro lado, es muy importante que esta primera demo funcional esté implementada de manera que sirva como una base sobre la que construir, facilitando añadir nuevo contenido al juego y continuar con su desarrollo. Es por esto que tiene mayor prioridad construir una estructura sólida que incluya los sistemas principales del juego que añadir todo el contenido requerido en las primeras versiones, puesto que será esta estructura la que condicione todo el contenido que se vaya a añadir a posteriori.

Una vez establecidas estas prioridades principales, se dividirán los requisitos y el contenido a implementar en varias fases, en función de su importancia y urgencia:

Primera Fase: Demo Funcional

El objetivo de esta fase es implementar las bases del videojuego, pudiendo probar sus sistemas y arreglar posibles bugs y errores pronto en el desarrollo, sin perder tiempo en aspectos menos prioritarios.

Durante esta fase se desarrollarán los sistemas básicos: mostrar las pantallas, cambiar de una a otra, gestionar los componentes de la empresa, pasar de turno, implementación de los minijuegos…

Se creará también el soporte para los tutoriales, poder mostrar las ventas totales de tu empresa, decidir cuánto producir y a qué precio venderlo, y los diferentes roles de cada usuario.

No entra dentro de esta fase el añadir contenido, más allá del necesario para probar los sistemas implementados. Por ello, se añadirán uno o dos elementos para cada funcionalidad: se añadirá el Producto, con dos opciones para elegir, y otro componente de la empresa que será el Almacén, con dos opciones también. Se implementará un minijuego, un tipo de demanda, una pantalla que permita crear un tipo de componente customizado, una pantalla que permita personalizar una partida, y una partida rápida de 3 turnos en total.

Tampoco entra dentro de esta fase el apartado gráfico y de sonido. Esta primera demo funcional no tendrá música, y los assets que tendrá serán placeholders: un botón y un fondo de fabricación propia, y assets gratuitos con licencia Creative Commons descargados de la página web de Kenney.

Segunda Fase: Demo Completa

Con esta segunda fase se pretende añadir el contenido requerido, teniendo una demo completa tanto a nivel de funcionalidad como de contenido. Se incluirán todas las opciones disponibles que puede escoger el jugador, más variedad de minijuegos, eventos y cuestionarios, permitiendo ofrecer una experiencia de juego completa.

El objetivo con esta segunda fase es probar el juego con varios jugadores en una beta cerrada, obteniendo su feedback sobre aspectos a mejorar, ajustando los números y las opciones para ofrecer una experiencia de juego lo más divertida posible.

Tercera Fase: Producto Mínimo Viable

En esta tercera y última fase del desarrollo el objetivo es tener un producto terminado, que si bien pueda ser actualizado con más contenido o nuevas opciones, sea por sí solo un producto terminado, un Producto Mínimo Viable.

Según su definición, un Producto Mínimo Viable, o MVP por sus siglas en inglés, es un producto capaz de satisfacer las necesidades de unos clientes iniciales a fin de proporcionar feedback de cara al futuro. Aunque puedan faltar ciertas funcionalidades y mejoras, este producto debe estar en un estado en el que se pueda vender y mostrar al público general.

En este MVP se ha de incluir todo el contenido de la fase anterior, habiendo realizado los ajustes pertinentes tras estudiar detenidamente el feedback recibido en esa fase. Con esto se pretende ofrecer al usuario una experiencia de juego pulida y equilibrada a nivel de gameplay.

Respecto al apartado gráfico y audiovisual, es en esta fase donde nos centraremos finalmente en ello, haciendo uso de música y assets con licencia Creative Commons que permita su uso comercial.

La banda sonora y efectos de sonido es una parte importante de la experiencia de juego, pero también es sencillo de integrar, puesto que es fácil ajustar los efectos de sonido para que vayan acorde con el estilo visual del juego, y a que hay multitud de canciones que podríamos poner como música de fondo en las distintas etapas del juego.

Desgraciadamente, no sucede lo mismo con el aspecto gráfico. A pesar de que hay multitud de paquetes de assets disponibles, tanto de pago como gratuitos con licencia Creative Commons, la posibilidad de encontrar un paquete con un estilo visual consistente que incluya tanto elementos de User Interface como personajes y animaciones para los minijuegos es muy baja.

Cuando hay una disparidad en el estilo de los diferentes assets del juego (imágenes, elementos del menú, animaciones…) el jugador lo nota enseguida, y percibe el juego como de baja calidad, debido al gran número de juegos, comúnmente llamados shovelware o asset flips, que se realizan con mínimo esfuerzo y reutilizando paquetes de assets populares o mezclando diferentes paquetes.

Es por esto que en el apartado gráfico se deberá invertir más esfuerzo, escogiendo un paquete de assets que encaje con el estilo del juego, y comisionando a un artista para que cree las imágenes y elementos que nos falten, de manera que se obtenga un estilo visual consistente y atractivo para el jugador.

Implementación del juego

En este apartado se procederá a implementar el juego, siguiendo todos los requisitos definidos en el apartado anterior.

Resumen de Godot

Godot es un motor de juegos 2D y 3D de uso general, gratuito y open source. Fue creado por un estudio de juegos argentino en 2001, y cobró popularidad en 2014, cuando se reescribió gran parte del código y se lanzó como proyecto open source.

Godot está programado en C y C++, y aunque permite usar esos lenguajes mediante GDExtension, tiene su propio lenguaje de programación, GDScript. Este lenguaje es de alto nivel, tipado dinámico, y orientado a objetos.

Cabe mencionar que el propio editor de Godot está construido usando su motor, lo cual prueba la confianza que tienen los desarrolladores en el software que han construido.

Árboles de nodos

Lo primero a destacar es el funcionamiento de los Nodos, la unidad básica sobre la que gran parte de la funcionalidad de Godot se construye. Un nodo es un objeto capaz de formar parte de un árbol de nodos. Todo en Godot se estructura mediante estos árboles, heredando las propiedades del nodo que tengan como padre. Por ejemplo, si un Nodo2D

tiene como hijo otro Nodo2D, al mover el nodo padre el hijo se moverá, pues hereda su propiedad “position”.

Todos los nodos que se pueden utilizar dentro del motor heredan de la clase Node. El color de los nodos indica su funcionalidad: Los azules son nodos 2D, los rojos son nodos 3D, y los verdes son nodos de Control, que componen la User Interface.

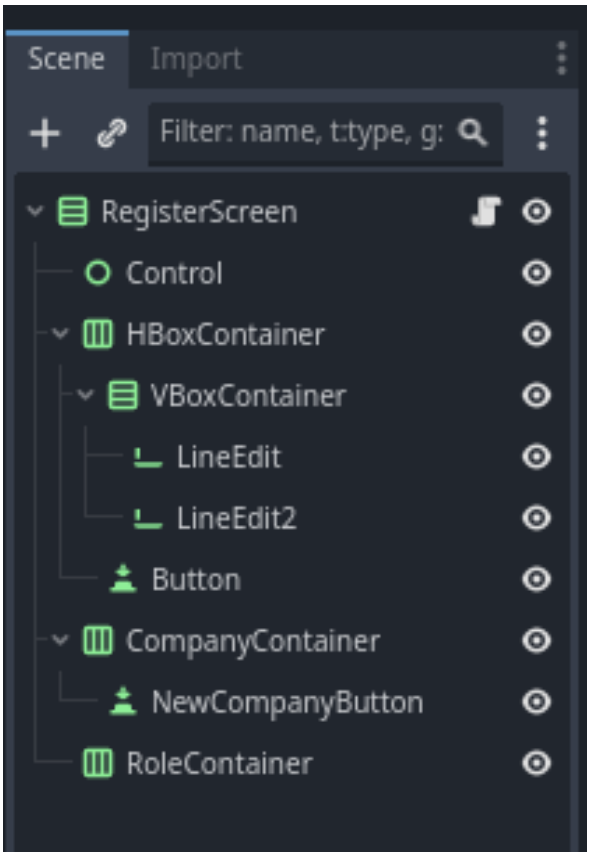

Figura 6: Ejemplo de un árbol de nodos perteneciente a una escena de Godot.
Fuente: elaboración propia

Este árbol de nodos se organiza en Escenas, ramas del árbol que pueden guardarse con una configuración determinada para instanciarse luego. Una vez creada, una escena se puede instanciar desde código, o directamente dentro de otra escena. Una de estas escenas es la que se configurará como main, y será la primera que se instancie cuando se inicie el juego.

Además de sus funcionalidades base, a cada uno de estos nodos se le puede asignar un script, permitiendo gestionar sus variables, así como acceder de forma directa a la funcionalidad de los nodos que tenga como hijos. Tanto los scripts como las propias escenas soportan herencia, pudiendo heredar métodos mediante la palabra clave “extends” en el caso de scripts, o heredar su árbol de nodos y su configuración en el caso de las escenas.

Autoload

En Godot es posible configurar uno o más scripts globales, llamado un autoload o singleton. Estos nodos se instancian de forma automática cuando se lance el main, al mismo nivel que la escena principal, y todos los nodos podrán tener acceso al script de este nodo, sin importar su posición en el árbol.

Esto permite, entre otras cosas, usar un autoload para coordinar nodos distantes que sería complicado coordinar de otro modo, o hacer accesible de forma sencilla una cierta funcionalidad para todos los nodos.

Métodos base

Además de los métodos que puedan tener los nodos que el motor nos ofrece, todos los nodos heredan unos métodos base que serán de gran importancia para manejar el comportamiento de los elementos que componen el juego. Los métodos más importantes de entre estos son _ready(), instantiate() y _process().

El método _ready() se llama la primera vez que el nodo se añade a la escena. Es en este método donde se realizará la configuración necesaria en el nodo, tal como pueda ser conectar señales o asignar valores a variables que dependan de otros nodos o componentes. Cabe mencionar que si no se ha añadido el nodo al árbol principal, no podrán accederse los nodos que tenga como hijo este nodo. Debido a esto, existe la keyword @onready, que permite inicializar una variable cuando se añada a la escena, en lugar de cuando se crea la instancia.

```
@onready var sprite = $RigidBody2D/Sprite2D
@onready var rigid_body = $RigidBody2D
```

Figura 7: Ejemplo de uso de la keyword @onready en el código.
Fuente: elaboración propia

El método instantiate() puede llamarse en una escena, y nos devolverá una instancia de esa escena, completa con su sub-árbol de nodos. Este es el método que utilizaremos para crear la mayoría de componentes, habiendo configurado todos sus nodos y elementos en la escena.

Hay ocasiones en las que se desea ejecutar un método o función repetidamente. Para ello se usa el método _process(), que es llamado por el motor de juego una vez por frame, es decir, cada vez que se actualice lo que se muestra en pantalla. Existe otro método, _physics_process(), que es llamado 60 veces por segundo, independientemente de los frames por segundo que tenga el juego. Este segundo método es el que se suele utilizar para gestionar el movimiento de los componentes del juego, puesto que si se hace en _process() la velocidad de movimiento dependería de los frames por segundo que tenga el juego.

Otro método importante es _init(), que se llama cada vez que se crea un nodo nuevo con el método new(), y puede ser sobreescrito para usarlo como método constructor. Debido a que los componentes que crearemos están compuestos por más de un nodo, será más conveniente usar instantiate(), ya que _init() sólo crea el primer nodo, sin tener en cuenta los nodos que pueda tener como hijos en su escena ni su configuración.

Señales

En caso de que se quiera acceder a alguna funcionalidad de un nodo hijo, se puede obtener el nodo mediante su ruta en el árbol y hacer lo que se necesite mediante esa referencia. En caso de querer afectar a un nodo padre o hermano, Godot recomienda el uso de señales.

Las señales son un mecanismo de delegación que permite a un objeto del juego reaccionar a un cambio en otro sin necesidad de que se referencien entre sí.

El funcionamiento de las señales es el siguiente: un objeto tiene la señal y puede emitirla, pudiendo incluir uno o más parámetros en la señal si hiciera falta. A esa señal se pueden conectar uno o más métodos, que serán llamados cuando la señal se emita. Los parámetros que emite la señal y los que recibe el método deben coincidir para que esto funcione correctamente.

Los métodos van asociados a su correspondiente nodo: si hay tres nodos con un método Prueba(), pero sólo uno de ellos lo ha conectado a una señal, únicamente se llamará al método de ese nodo al emitir la señal.

Colisiones

Godot utiliza varios tipos de hitbox, como pueden ser los nodos CollisionShape y CollisionPolygon, para gestionar cuando dos elementos chocan. Estas hitbox han de añadirse como nodo hijo de un Área o Body. El motor de juego distingue entre Áreas, que normalmente son usadas para comprobar si se está en contacto con algo de forma unidireccional, y Bodies, que tienen interacciones con el motor de físicas de Godot y suelen ser detectados, más que detectar ellos colisiones. Un componente de tipo Body es por ejemplo el que se usaría para el personaje del jugador (CharacterBody2D) o para un elemento del escenario (RigidBody2D o StaticBody2D).

Para comprobar si se está detectando una colisión, Godot hace uso de máscaras de colisión. Existen 2 tipo de máscara: Layer, y Mask.

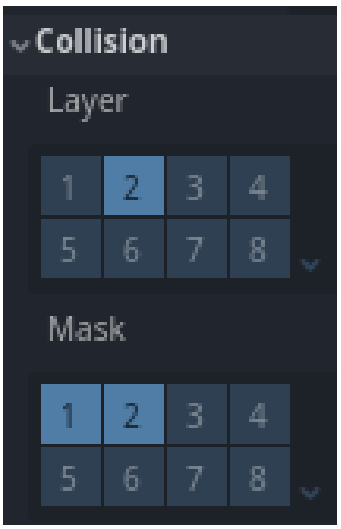

Figura 8: Menú de configuración de capa y máscara de colisiones.
Fuente: elaboración propia

Layer dicta en qué capa de colisión se encuentra el objeto, y Mask indica con qué capas puede colisionar. En el caso de la imagen, podemos ver que existe en la capa 2, y que detectará una colisión con nodos que estén en la capa 1 o 2.

User interface

Para hacer la UI, Godot hace uso de dos elementos principales: CanvasLayer, y nodos de tipo Control.

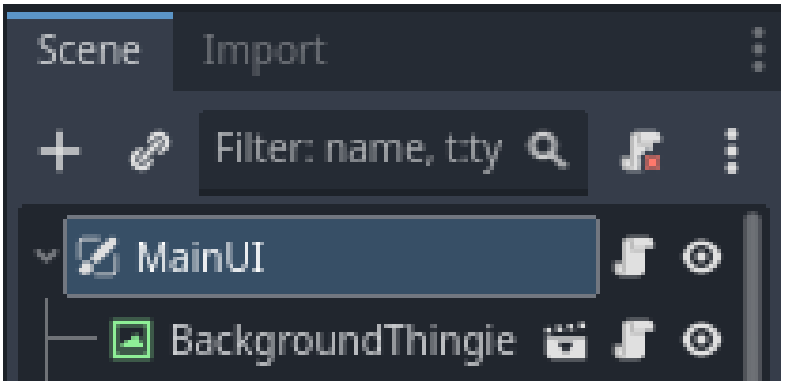

Figura 9: Ejemplo de una escena de Godot con nodos de tipo Control y CanvasLayer.
Fuente: elaboración propia

CanvasLayer (llamado MainUI en la imagen) asegura que sus hijos siempre se van a renderizar, sin importar la posición de la cámara.

Los nodos de tipo Control (todos aquellos con un icono verde) componen todos los elementos que crean la UI. En lugar de tener una posición fija, su posición es relativa, marcada por su nodo padre y varios de sus parámetros mediante el uso de anchors. Esto permite que los nodos puedan variar su tamaño en función de las dimensiones de la

pantalla, en relación a sus puntos de anclaje, permitiendo así adaptarse a diferentes ratios en la pantalla del usuario.

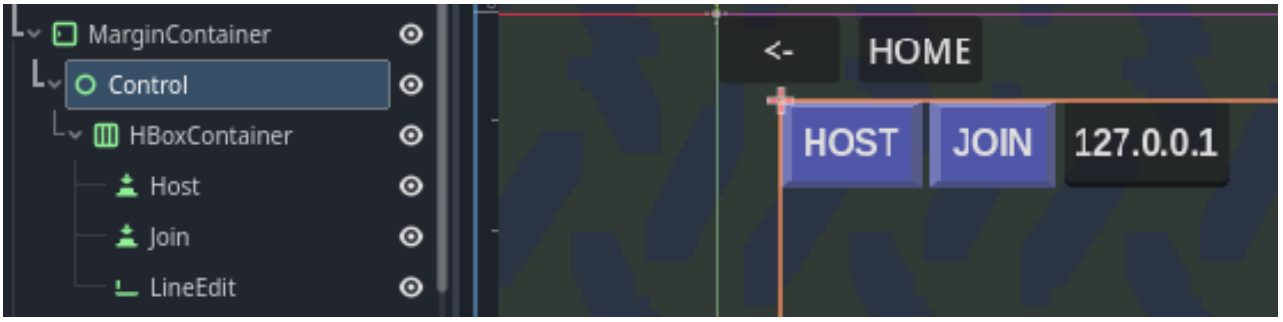

Figura 10: Ejemplo de una escena de UI construida con nodos de tipo control.
Fuente: elaboración propia

Otro aspecto importante a destacar de la UI es el uso de Temas, configuraciones generales de los varios elementos que componen la UI. Esta configuración también se hereda de nodos padre a nodos hijo, permitiendo crear un estilo visual consistente de forma fácil e intuitiva. Esta funcionalidad permite también visualizar de antemano cómo quedarían los distintos elementos de la interfaz aplicando la configuración del tema.

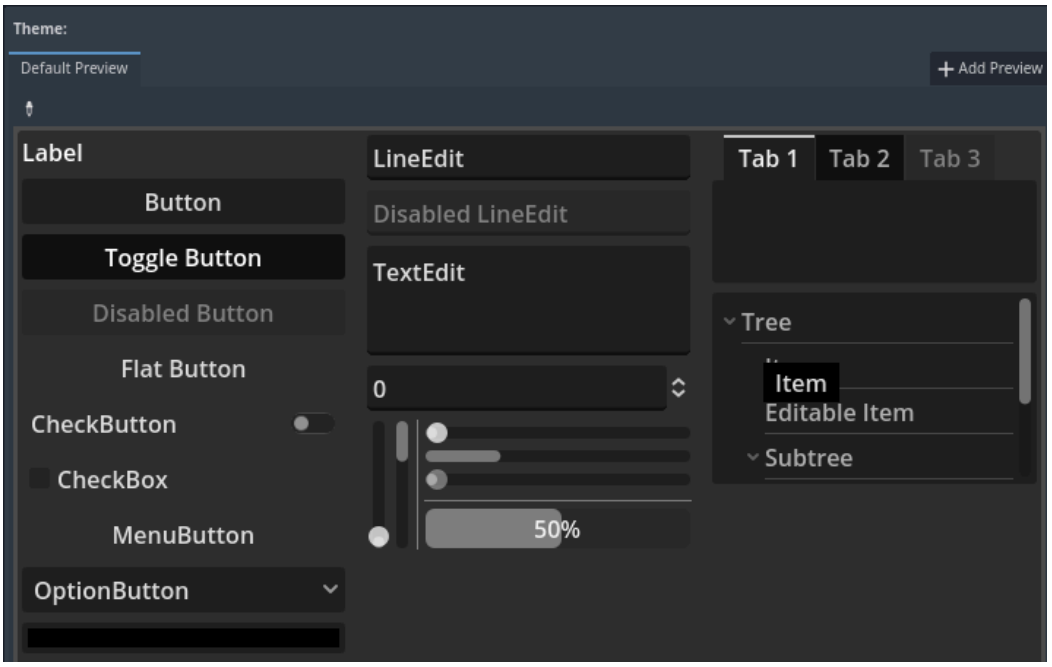

Figura 11: Pantalla de previsualización de un Tema de Godot.
Fuente: elaboración propia

Multijugador en Godot

Ya desde sus primeras versiones, Godot ha tenido soporte para networking de bajo nivel, mediante protocolo UDP o TCP, así como soporte para networking de alto nivel mediante

HTTP o SSL, más comúnmente usados para comunicarse por un server mediante llamadas a una API REST.

El protocolo TCP está optimizado para que los paquetes que se envían a través de la red lleguen en el orden correcto y sin errores, pero tiene una mayor latencia lo cual es algo indeseado cuando se trata de un videojuego, donde la lentitud al comunicarse servidor y cliente puede dar lugar a lag y causar problemas al usuario.

El protocolo UDP está optimizado al contrario, ofreciendo una menor latencia, pero al no corregir errores ni asegurar que los paquetes lleguen en el orden correcto, es más propenso a errores, sobre todo cuando se trata de paquetes grandes.

Por suerte, el motor también nos proporciona una API de alto nivel que se encarga de gestionar estos dos protocolos y las conexiones necesarias, pudiendo utilizar el que más se adecúe a nuestras necesidades.

Para ello, godot hace uso de un MultiplayerPeer, que representa a un equipo conectado a la red, ya sea el cliente o el servidor. Este MultiplayerPeer tiene un ID de conexión único, que es un número aleatorio para los clientes, y será siempre un 1 en el caso del servidor. MultiplayerPeer hereda de PacketPeer, por lo que tiene todos los métodos necesarios para para serializar, enviar y recibir datos sobre la red, y también incluye señales que permiten saber si un equipo se ha conectado o desconectado de la red. En nuestro caso, haremos uso de un ENetMultiplayerPeer, aunque también cabe mencionar que hay implementaciones de MultiplayerPeer basadas en WebRTC o WebSocket.

Para sincronizar los elementos del juego de manera que todos los equipos reciban los mismos datos sobre los demás jugadores y elementos del juego, Godot hace uso de dos elementos principales: MultiplayerSpawner y MultiplayerSynchronizer.

MultiplayerSynchronizer se encarga de sincronizar las variables que se le indiquen, manteniendo su valor entre todas las instancias de todos los equipos conectados. Uno de estos equipos (por defecto el server) tendrá autoridad sobre dichas variables, y será el único capaz de cambiar sus valores. Debido a esto, es común llamar a funciones del server mediante rpc_id para cambiar el valor de las variables sincronizadas. La funcionalidad rpc (Remote Procedure Call) se explicará más adelante.

MultiplayerSpawner se encarga de permitir el funcionamiento del nodo anterior, asegurando que los nodos a sincronizar (tales como podrían ser los diferentes personajes en un juego online) tengan una ruta común, siendo hijos de este nodo. Se les ha de asignar el mismo nombre en todos los peers para su correcto funcionamiento.

Además de estos nodos, se puede declarar una función como rpc (Remote Procedure Call), lo cual permite llamarla desde otros peers, ya sea desde un peer específico, añadiendo su ID de conexión a los parámetros del método y llamando mediante rpc_id, o llamando a todos los peers cuando se llame mediante rpc.

Una función declarada como rpc puede ser configurada con varios parámetros para alterar su funcionamiento, recibiendo como parámetros el modo, sincronización, modo de transmisión, y canal de transmisión.

Modo: Puede ser “authority”, en el que sólo puede ser llamado desde el servidor, o “any_peer”, donde cualquier equipo conectado puede llamarlo.

Sincronización: Modifica su comportamiento cuando se le llama por rpc. Puede ser “call_remote”, con lo que no se llamará en el propio equipo, o “call_local”, con lo que sí podrá llamarse en el equipo.

Modo de transmisión: Puede ser “reliable”, donde se asegura que los paquetes se obtengan todos y en el orden correcto, “unreliable” donde no comprueba si llegan correctamente ni en orden, y “unreliable_ordered” donde no se comprueba si los paquetes han llegado correctamente, pero sí se comprueba que lleguen en orden.

Canal de transmisión: Esta variable es el índice del canal de transmisión, por defecto 0.

Estructura de carpetas

Estructura de directorios en godot

Godot distingue principalmente entre dos tipos de directorios.

El primero se identifica con el prefijo “res”, y engloba todos los ficheros del propio juego: recursos, escenas, scripts, y assets. Durante el desarrollo se pueden modificar estos ficheros, pero una vez se exporta en un ejecutable, todo el contenido de esta carpeta se empaqueta en modo de sólo lectura, por lo que no nos será posible crear nuevos ficheros ni modificar los existentes dentro de esta carpeta.

El segundo se identifica con el prefijo “user”, y hace referencia a la carpeta donde se guardará todo el contenido del juego que necesite persistir: ficheros de configuración, partidas guardadas, etc. La localización de esta carpeta varía en función del sistema operativo en el que se ejecute el juego, en Windows por ejemplo se encuentra en C:\Usuarios\$USUARIO\AppData\Roaming\Godot\app_userdata\$NOMBRE_DEL_JUEGO.

Es posible acceder a otras rutas usando Godot, pero para las necesidades de este proyecto no será necesario.

Estructura del proyecto

En este apartado se mostrará la estructura de ficheros del proyecto en Godot. Se ha organizado en función de los componentes del juego y la funcionalidad que proveen.

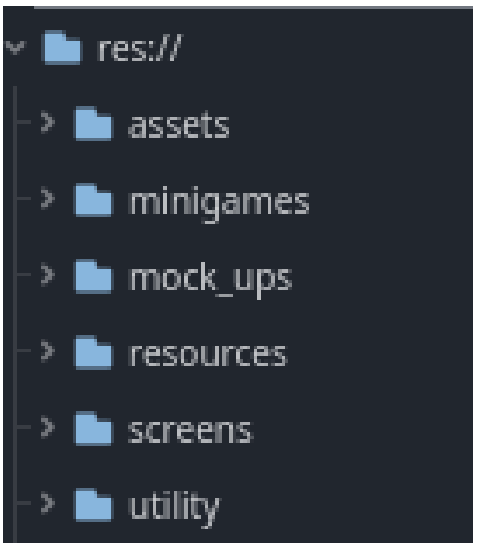

Figura 12: Carpetas principales del proyecto en Godot.
Fuente: elaboración propia

Carpeta Assets: En esta carpeta se guardan todos los assets del juego, ya sean imágenes, texturas, temas, archivos de sonido (de los cuales no hay ninguno actualmente). Dentro contiene más subcarpetas en función de para qué se van a usar dichos assets.

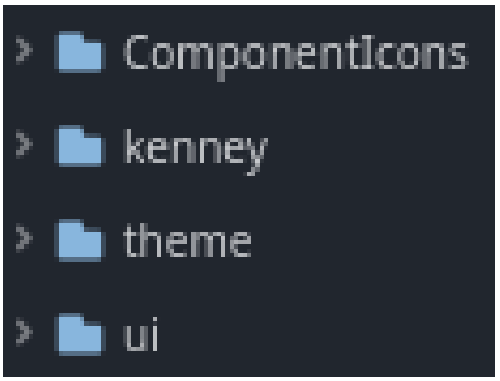

Figura 13: Subcarpetas dentro de la carpeta Assets.
Fuente: elaboración propia

Carpeta Minigames: Aquí se guardan los componentes necesarios para inicializar y hacer funcionar los minijuegos. Dentro de esta carpeta existe una subcarpeta para cada minijuego, y los elementos sueltos son componentes comunes a todos.

Figura 14: Subcarpetas dentro de la carpeta Minigames.
Fuente: elaboración propia

Carpeta Mock-Ups: En esta carpeta hay pantallas y elementos que existen para testear alguna funcionalidad aislada o para servir de placeholder mientras se desarrolla otro componente, pero que no tienen cabida en el producto final.

Carpeta Resources: Aquí se guardan los elementos más abstractos, tales como son los Productos, los Componentes que conforman la empresa (proveedores, almacenes, tecnologías, etc), la demanda, los turnos… Todos estos elementos reciben sus datos de ficheros json, guardados dentro de la carpeta user://

Carpeta Screens: Esta carpeta contiene las diferentes pantallas, organizadas en subcarpetas. Cada pantalla tiene tanto su escena, como otros componentes de UI que pueda necesitar, tales como botones, pop-ups, contenedores…

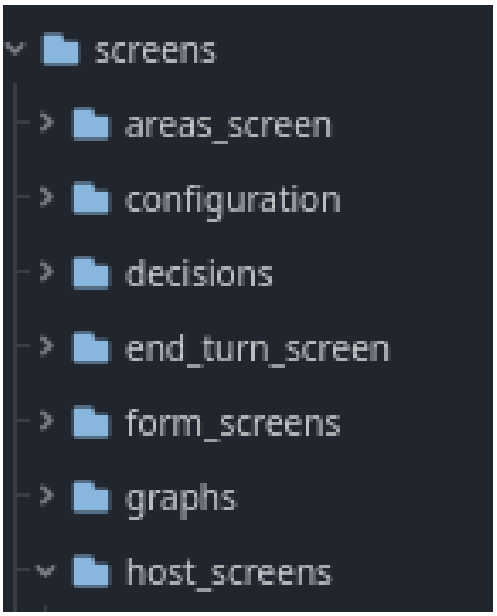

Figura 15: Subcarpetas dentro de la carpeta Screens.
Fuente: elaboración propia

Carpeta Utility: En esta carpeta se guardan componentes que implementan funcionalidad general, tales como los desplegables usados en la pantalla de configuración de la partida, ciertos MultiplayerSpawner, etiquetas genéricas y botones que muestran un pop-up al poner el ratón sobre ellos. También existe una subcarpeta con todos los componentes necesarios para la máquina de estados que gestiona el cambio de una pantalla a otra.

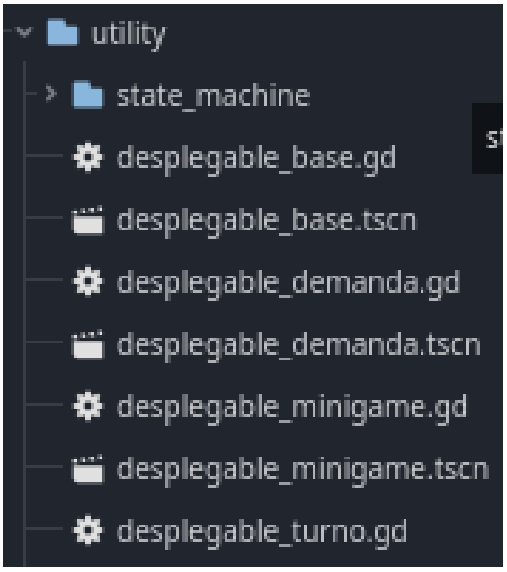

Figura 16: Subcarpetas y elementos sueltos dentro de la carpeta utility.
Fuente: elaboración propia

Arquitectura del código

En este apartado se detallan los principales elementos que componen el juego, así como su orquestación e interacción entre estas diferentes partes.

Main

La primera escena es nuestro Main. Esta escena tendrá como hijos a todos los demás elementos, a excepción del Global. Gestionará también la creación de los minijuegos, añadiendo su escena como hijo suyo, y se encargará también de coordinar la cámara.

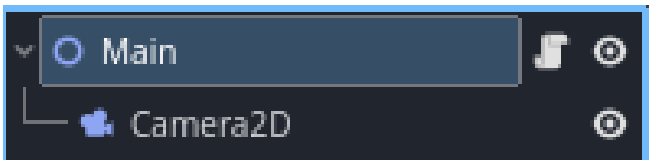

Figura 17: Árbol de nodos de la escena Main.
Fuente: elaboración propia

Dentro del script de esta escena se posiciona la cámara para que siga un nodo en particular. Al comienzo del juego y tras cada minijuego, este nodo es un nodo2D estático en Main, pero al comenzar un minijuego se le indica a la cámara a qué nodo ha de seguir, siendo éste normalmente el personaje del jugador.

MainUI

Este componente es hijo de Main, y es donde inicializamos el resto de componentes, siendo ésta la escena con la UI base. Aquí es donde se tiene una imagen que se desplaza lentamente en el fondo, así como los botones con los que se puede retroceder en las pantallas y regresar a la pantalla principal.

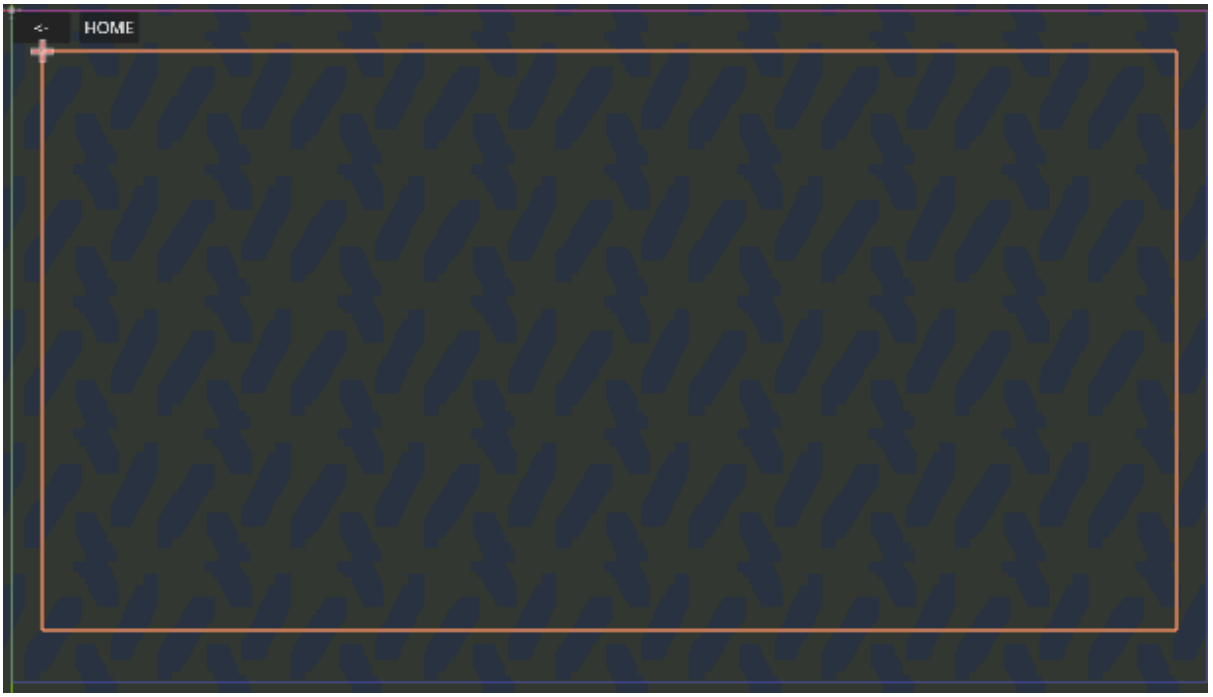

Figura 18: Captura de la pantalla mostrando la escena MainUI dentro del juego. Se resalta el MarginContainer donde se añaden el resto de pantallas.
Fuente: elaboración propia

También es en esta pantalla donde se inicializan los distintos componentes de la máquina de estados, así como algunos componentes que no tienen uso en la versión final pero que son útiles para realizar debug del videojuego.

En esta escena es donde se inicializarán las demás pantallas, dentro del contenedor MarginContainer, resaltado en la imagen que se muestra a continuación. Este componente

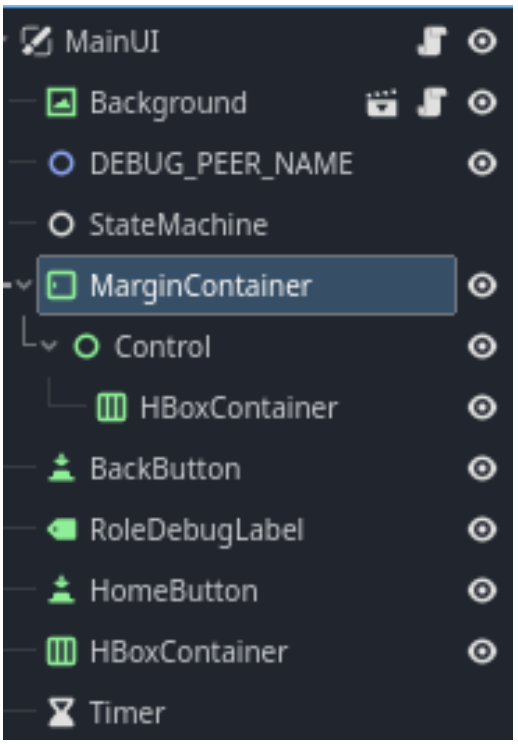

Figura 19: Árbol de nodos de la escena MainUI.
Fuente: elaboración propia

Global

Al mismo nivel en el árbol que la escena Main, tenemos un autoload llamado Global. Este nodo y su script cumple con diversas funcionalidades:

En primer lugar, se encarga de gestionar los cambios necesarios a los nodos que necesiten sincronización, llamando mediante RPC a estos métodos del Global desde los clientes para que el server cambie las variables necesarias.

También se encarga de gestionar el cambio de una pantalla a otra mediante una máquina de estados, pudiendo cambiar de una pantalla a otra desde cualquier otro nodo.

En el Global también se encuentra la funcionalidad necesaria para guardar y cargar los diferentes componentes en un fichero .json para que se persistan entre sesiones de juego.

Máquina de Estados

El juego dispondrá de varios estados, cada uno de ellos correspondiéndose con una pantalla del mismo. Todos estos estados heredarán de la clase base State, que contiene dos métodos:

enter(): Decide el comportamiento a ejecutar cuando este estado pasa a ser el estado actual.

exti(): Decide el comportamiento a ejecutar cuando este estado deja de ser el estado actual.

Ambos métodos reciben como parámetro un diccionario, donde se incluirán variables que permitan modificar su comportamiento en función de las necesidades del juego. Un ejemplo de esto es el stack de estados.

Cada estado se almacena en un stack cuando deja de ser el estado activo, permitiendo al jugador retroceder a una pantalla anterior al pulsar el botón de BACK, representado con un icono de una flecha apuntando hacia la izquierda. Dependiendo de lo que reciba en el diccionario de entrada, un estado puede no guardarse en ese stack, o puede limpiar todo el stack, por ejemplo, como sucede al finalizar un turno o tomar una decisión.

Todos los estados se instancian al comienzo del juego, formando parte de la escena Main y siendo gestionados por Global. Esto permite que se pueda cambiar de pantalla desde cualquier otro nodo, facilitando las transiciones.

Configuración

Se desea que los parámetros de una partida se puedan personalizar, pudiendo modificar la duración de los turnos, el comportamiento de la demanda, y el minijuego que se ha de jugar al final de cada turno.

Para ello, se hará uso de los nodos que se explicarán a continuación. Todos estos nodos obtienen sus datos de ficheros .json localizados en la carpeta del usuario, permitiendo que se añada nuevo contenido con tan sólo arrastrar los archivos en dicha carpeta, y que el jugador pueda crear nuevos elementos que se ajusten a sus necesidades.

El Usuario, Compañía, Turno actual y Game estarán sincronizados, siendo hijos de su correspondiente spawner.

Game

Este es un nodo que contiene un array de turnos, sincronizado entre todos los jugadores, y un número entero que apunta al turno actual.

Todos estos datos se configuran desde un fichero .json, pudiendo crear un juego el propio usuario al elegir cuántos turnos quiere y seleccionando la configuración de cada turno desde el desplegable. Al pulsar en el botón + que hay en cada desplegable, se avanza a la pantalla de creación de turno, que se explicará más adelante.

Figura 20: Pantalla de configuración del juego.
Fuente: elaboración propia

Game es el nodo a partir del que se obtienen los datos del turno actual, buscando el ID de dicho turno en el array para leer su información desde el fichero .json correspondiente.

Turno

Este es el elemento que contiene la información relevante a un determinado turno: qué comportamiento ha de tener la demanda, qué minijuego ha de jugarse al finalizar dicho turno, y qué eventos tiene.

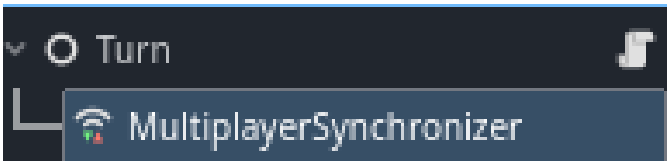

Figura 21: Árbol de nodos de la escena Turn.
Fuente: elaboración propia

El turno actual se instancia como un nodo sincronizado sus datos. Al cambiar de turno, se elimina el nodo del turno y se crea uno nuevo, cargando los datos desde su fichero .json. Esto se lleva a cabo desde Global.

Es posible crear un nuevo turno, accediendo a esta pantalla al pulsar el botón + en el desplegable del turno. Aquí se escoge desde un desplegable el comportamiento de la demanda y el minijuego del turno.

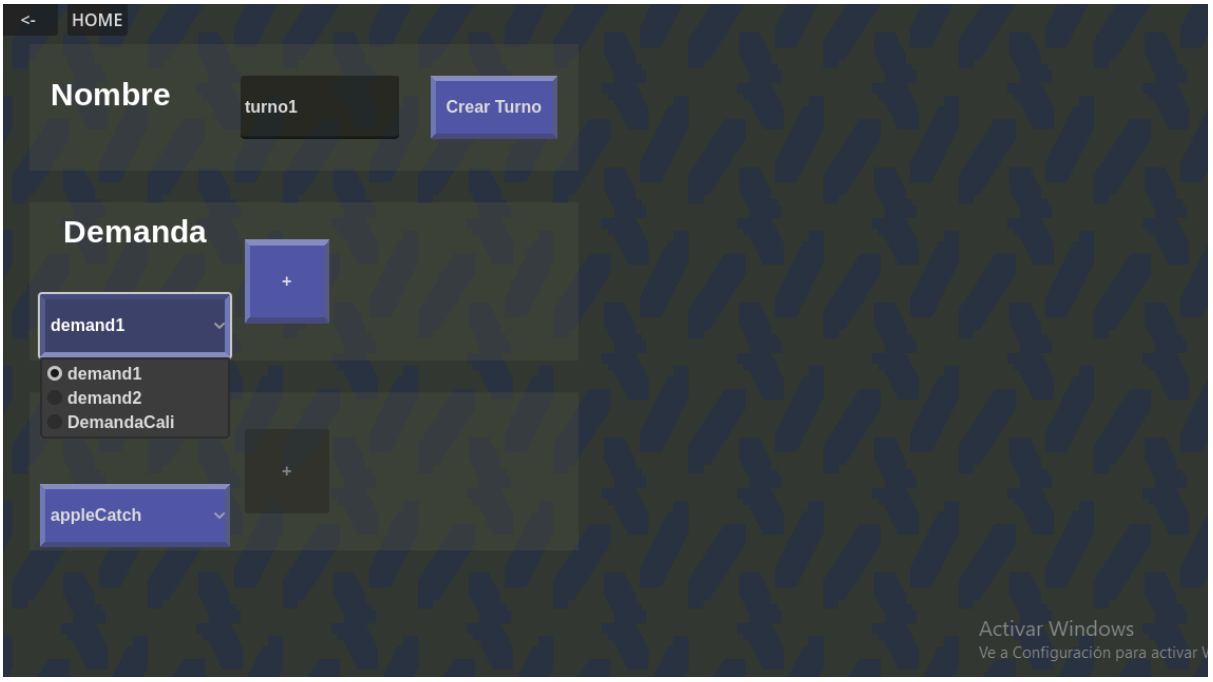

Figura 22: Pantalla de configuración del turno.
Fuente: elaboración propia

La opción de crear un minijuego está desactivada en este desplegable, debido a que no es un componente que pueda crear o configurar el usuario más allá de elegir qué minijuego va a usarse.

Demanda

Este componente decide cómo va a comportarse la demanda durante un turno determinado. Este nodo simplemente tiene un valor total de la demanda, representando todos los posibles compradores que existen en el mercado, y un número que representa el peso para cada una de las cualidades del producto.

Al igual que con los componentes anteriores, el usuario puede crear su propia configuración de la Demanda, al pulsar el botón + de su desplegable y rellenar los datos pertinentes a la demanda total y el peso de cada cualidad en la siguiente pantalla:

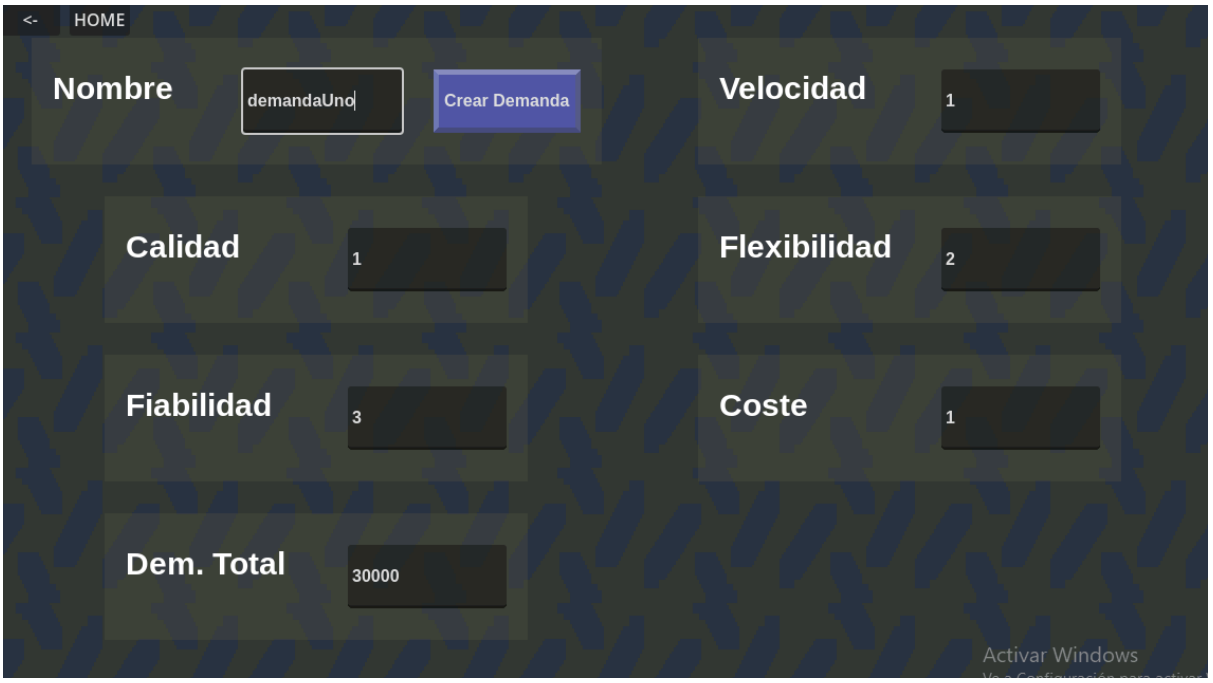

Figura 23: Pantalla de configuración de la demanda.
Fuente: elaboración propia

Compañía

Este se trata de un elemento clave para el juego, puesto que será el que los jugadores dirijan mediante sus decisiones a lo largo de la partida. El nodo Company representa la empresa, gestionada por un determinado equipo.

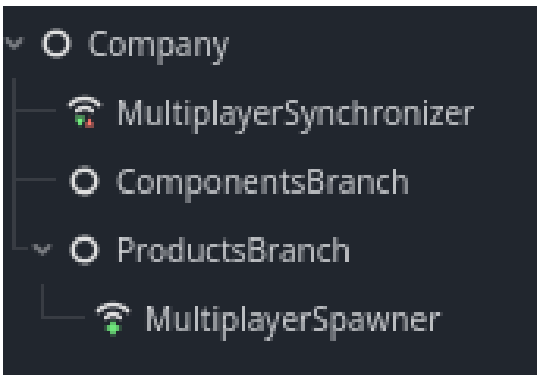

Figura 24: Árbol de nodos de la escena Company.
Fuente: elaboración propia

Cada compañía tiene su MultiplayerSynchronizer para sincronizar los datos con todos los equipos, y se añadirá como hijo de su correspondiente spawner.

La compañía tiene como variables sus estadísticas, almacenadas en dos diccionarios. Uno almacena estadísticas a lo largo de la partida: cuántas unidades se ha vendido en un

determinado turno, cuánto beneficio se ha obtenido, y más información relevante. El otro es el de las stats, que guarda todos los datos relevantes para las operaciones de la empresa: las cualidades del producto, modificadores de los costes de producción y almacenaje, capacidad de producción, y eficiencia. Este diccionario obtiene sus datos de otros componentes, mediante una señal.

Todos los componentes que puede tener la empresa: productos, proveedores, almacenes, tecnologías, y cualquier otro elemento que forme parte del proceso productivo o pueda dar una ayuda a la empresa, se abstrae en un Componente. Este nodo no necesita ser sincronizado, puesto que sus datos no van a cambiar a lo largo de la partida, por lo que es suficiente con inicializarlos en todos los clientes mediante llamada rpc.

Un Componente contiene su nombre, descripción, imagen, coste de adquisición, y puede tener un slot. Algunos Componentes son mutuamente excluyentes y necesarios para la empresa: Almacenaje, Proveedor, Producto, Tecnología y Organización. Al añadir uno de estos componentes a la empresa, reemplazará el que hubiese anteriormente en ese slot.

Todos los slots tienen un componente vacío que permite tomar la decisión de qué se quiere utilizar en dicho slot, mostrándose esta decisión en la pantalla de Áreas, que se explicará más adelante.

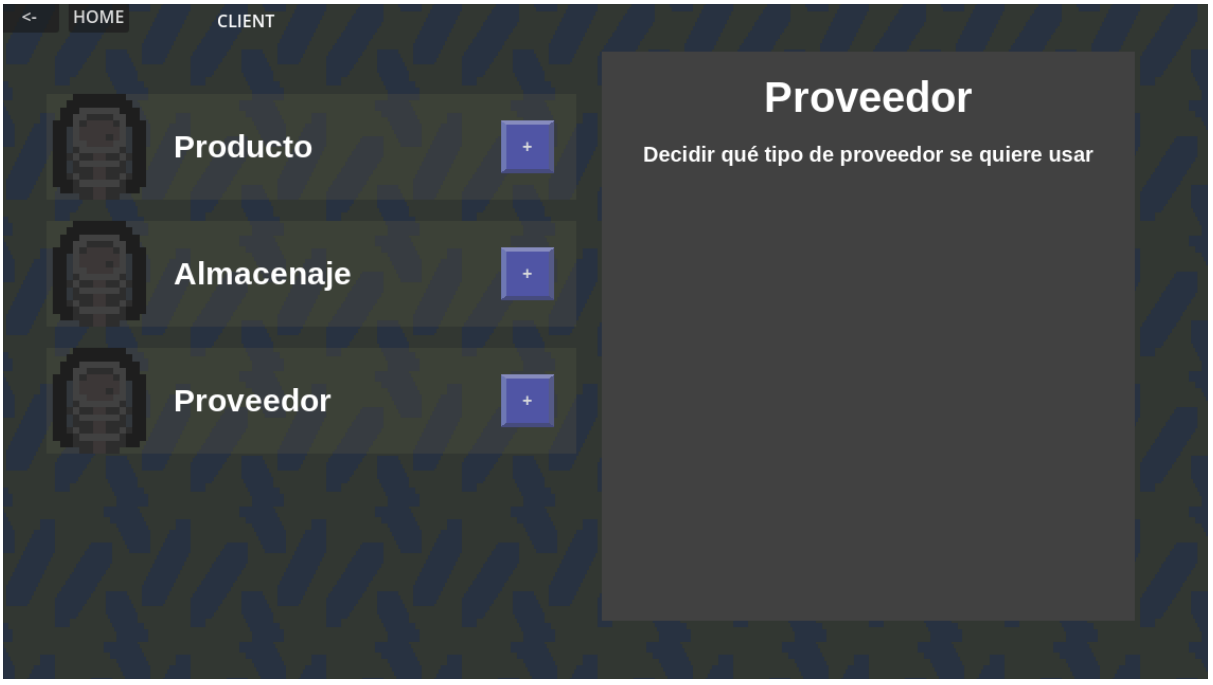

Figura 25: Captura de pantalla mostrando los componentes de cada slot, vacíos.
Fuente: elaboración propia

Cada componente tiene también un ID para la decisión que se usa en caso de querer cambiarlo. Este botón nos lleva a la pantalla de Decisión, donde se mostrará un botón para cada opción disponible, así como una caja de texto con la descripción.

Figura 26: Pantalla de decisión del Proveedor.
Fuente: elaboración propia

Para poder modificar las stats de la empresa, cada Componente tiene un StatChange por cada número que quiera afectar. Este tiene tres variables, dos se corresponden a las claves correspondientes del diccionario, y la tercera variable es el número que se sumará a la stat.

Al añadir el Componente, se conectan todos sus StatChange con un método de la compañía, pasándole a esa señal un diccionario con las mismas claves que el de las stats y todos los valores a 0. Al pasarse el diccionario por referencia, todos los StatChange podrán modificar su valor, resultando al final en un diccionario con las stats finales. Esto permite que un StatChange pueda sobreescribir la función conectada a esa señal para cambiar su valor en función de ciertas condiciones en caso de que sea necesario.

El Producto es un caso especial entre los Componentes. Cada Producto tiene su Componente, que se usa para tomar la decisión de qué producto ofertar, pero además este componente crea un Producto. Este nodo sí que estará sincronizado, ya que debe gestionar todos los datos relevantes a un producto: número de unidades en stock, capacidad de producción y almacenaje, costes, rango de precios… Gracias a esto podemos guardar los datos de cada producto en su propio nodo, siendo posible iterar por todos ellos cuando se necesite obtener información para realizar cálculos o mostrársela al jugador.

Pantallas

A continuación se va a explicar el funcionamiento general de las pantallas. Debido al gran parecido que hay entre ellas, se explicará su estructura general y se mostrarán algunos ejemplos, en lugar de mostrar en detalle todas y cada una de las pantallas del videojuego.

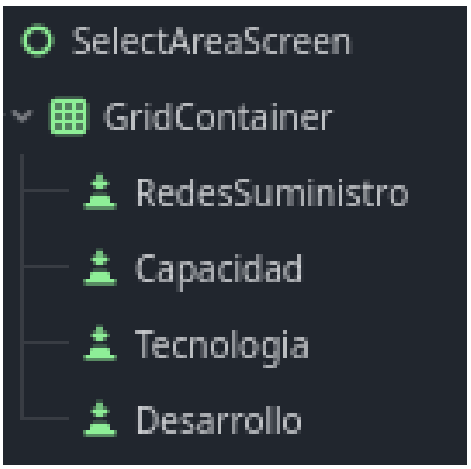

Figura 27: Árbol de nodos de la pantalla de selección de área.
Fuente: elaboración propia

Todas las pantallas son una colección de elementos de Control, principalmente contenedores que permiten modificar la posición de otros elementos, etiquetas y botones.

En ocasiones no es suficiente con utilizar un botón o una etiqueta, y se requiere una funcionalidad más compleja, como es el caso de los desplegables o de la opción de decidir cuánto se va a producir este turno. En este caso se crea dicho componente como una escena, para poder reutilizarlo tantas veces como sea necesario dentro de la pantalla.

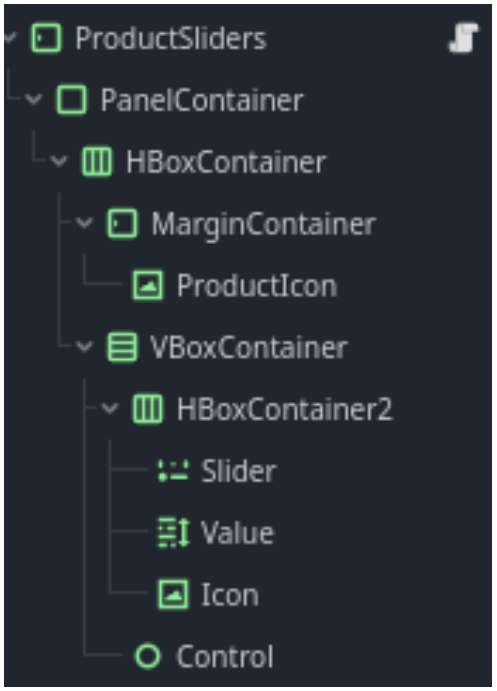

Figura 28: Árbol de nodos del componente ProductSlider.
Fuente: elaboración propia

En el caso del slider del producto, está compuesto por un PanelContainer, un contenedor que sobrepone un panel semitransparente en el espacio que ocupa, varios HBox y VBox Containers para manejar la posición de sus elementos, un MarginContainer para encapsular algunos elementos dentro de un margen y que así se vea mejor, y el propio slider.

Figura 29: Captura del componente ProductSlider dentro del juego.
Fuente: elaboración propia

Otro ejemplo de esto es el desplegable, utilizado en las pantallas donde se configura el juego y sus distintos elementos. Cada uno de los elementos utiliza un Desplegable distinto, pero todos heredan de un DesplegableBase. Este contiene una etiqueta para el nombre dentro de un MarginContainer, el desplegable con las opciones a mostrar, y un botón que lleva a la pantalla correspondiente para crear un nuevo elemento.

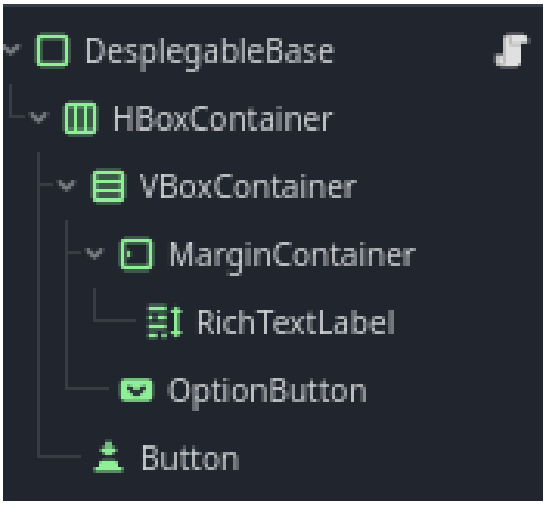

Figura 30: Árbol de nodos de la escena DesplegableBase.
Fuente: elaboración propia

Este componente obtiene los IDs a mostrar en su desplegable en función de los objetos de tipo .json que encuentre en una determinada carpeta, de manera que muestra aquellos elementos creados por el usuario tan pronto como se añaden. Estos IDs se corresponden con el nombre del fichero correspondiente.

```
func _ready():
    new_button.pressed.connect(go_to_make_demanda)

    var dirA = DirAccess.open(Global.demand_path)
    if dirA:
        dirA.list_dir_begin()
        var file_name = dirA.get_next()
        while file_name != "":
            if not dirA.current_is_dir():
                file_name = file_name.erase(file_name.find("."), 5)
                var path = dirA.get_current_dir()
                path = path + "/"
                var json_found = Global.get_string_from_file(path, file_name)
                desplegable.add_item(file_name)
            file_name = dirA.get_next()
    super()
```

Figura 31: Función _ready() del Desplegable.
Fuente: elaboración propia

Áreas

Las Áreas de la empresa son la pantalla en la que se muestran todos los componentes pertenecientes a una empresa, y desde donde se pueden obtener nuevos activos o modificar las decisiones que se han tomado.

Cada Componente tiene uno o más tags, que son usados por esta pantalla para filtrar, mostrando únicamente los Componentes relevantes. El tag “master” lo tienen todos los componentes, puesto que se le mostrará al Game Master una única Área con todos los componentes, mientras que los jugadores tendrán una pantalla por cada área de la empresa.

Figura 32: Pantalla de selección de área.
Fuente: elaboración propia

Cada pantalla de Área utiliza la misma escena, filtrando por el tag correspondiente, pero tendrá un estado diferente en la máquina de estados. Este estado es el que le pasará el tag por el que debe filtrar los componentes en el momento de instanciar la pantalla.

Flujo general del juego

En este apartado se mostrará de forma resumida el funcionamiento general del juego, así como las diferentes fases y pantallas por las que pasa a lo largo de una partida.

Al iniciar el juego, se muestra la siguiente pantalla, donde se permite introducir la IP con la que se quiere hostear el juego o conectarse, y un desplegable para la configuración del Game, explicada en un apartado anterior.

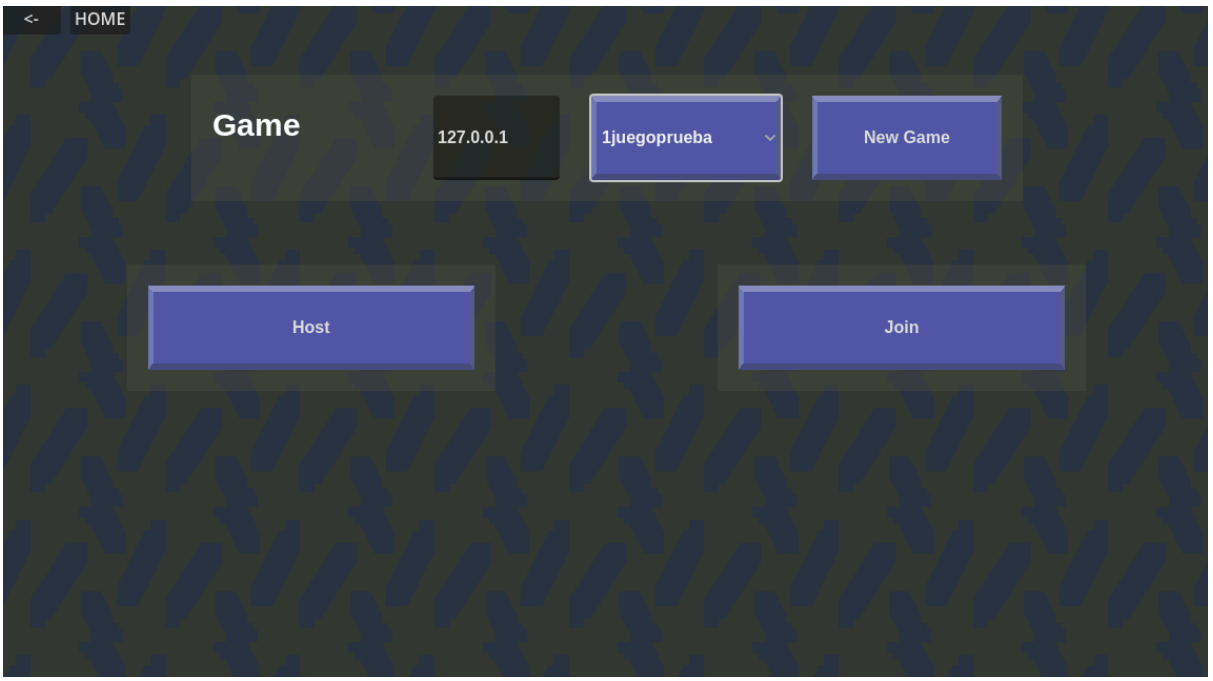

Figura 33: Pantalla inicial del juego.
Fuente: elaboración propia

Los dos botones que se muestran, Host y Join, permiten crear la partida, o unirse a una partida ya creada, respectivamente.

Al conectarse a una partida muestra una pantalla de carga, y al Game Master se le mostrará la siguiente pantalla.

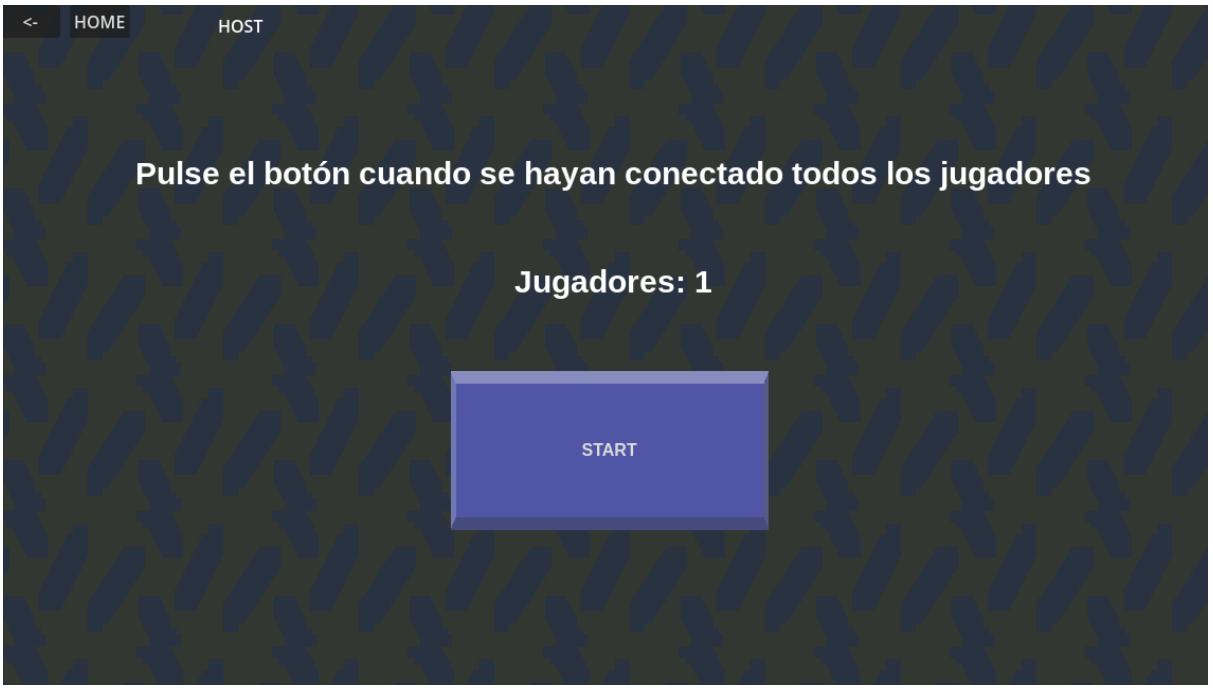

Figura 34: Pantalla de comenzar la partida, mostrada al Host.
Fuente: elaboración propia

Al pulsar en Start, el juego comenzará, mostrando al cliente un breve tutorial antes de llevarle a la pantalla de selección de empresa.

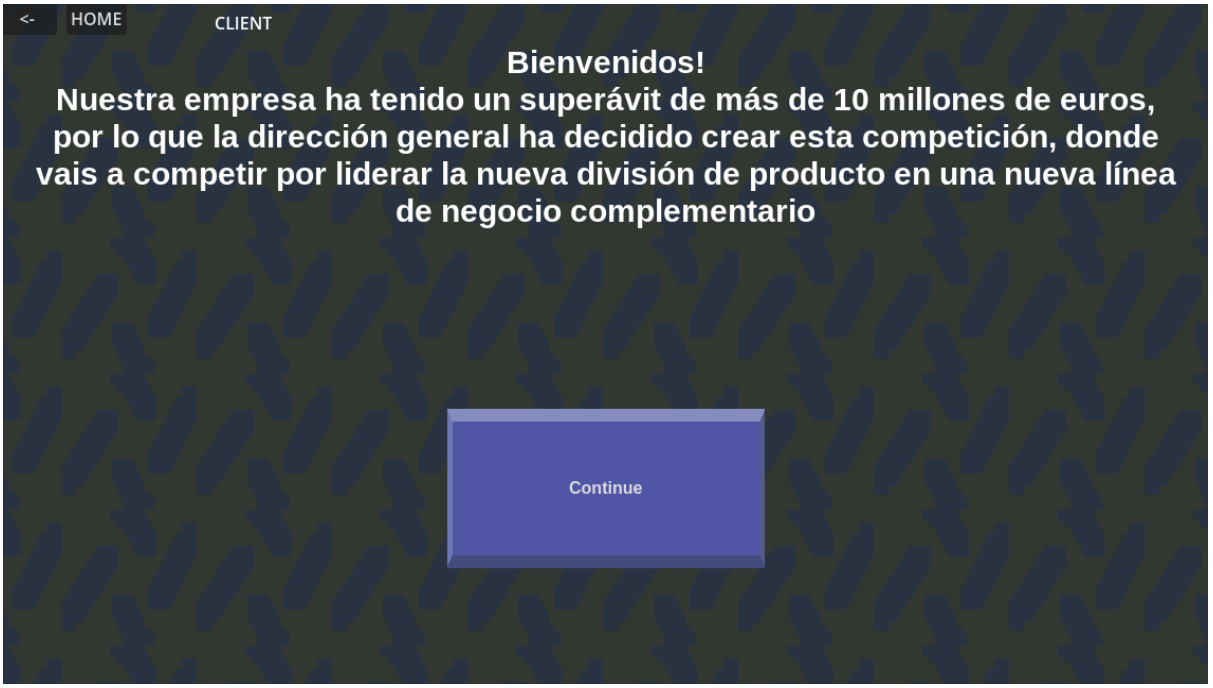

Figura 35: Pantalla de tutorial inicial.
Fuente: elaboración propia

En esta pantalla cada jugador introducirá un nombre de usuario y una contraseña, y se escogerá la Compañía y el Rol que desee el jugador.

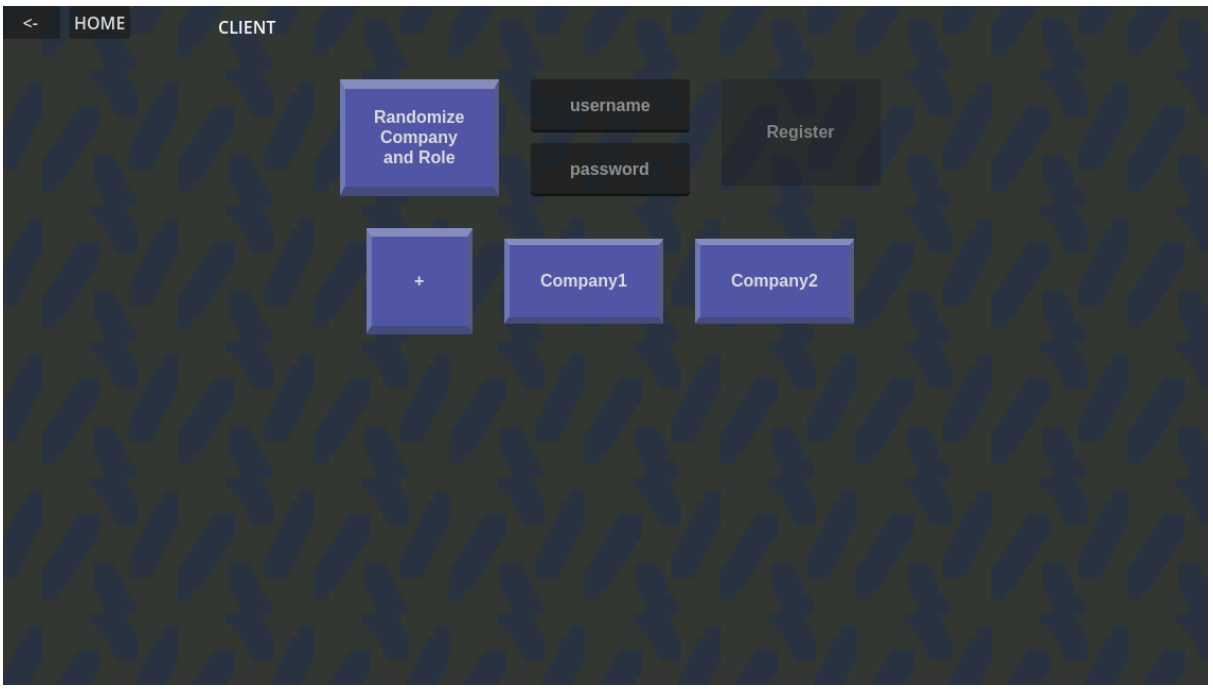

Figura 36: Pantalla de registro del jugador.
Fuente: elaboración propia

El botón de + permite crear una compañía con un nombre customizado, y una vez se seleccione una compañía se muestran los botones para seleccionar rol. En caso de que un rol ya esté escogido para esa empresa, esa opción aparecerá deshabilitada.

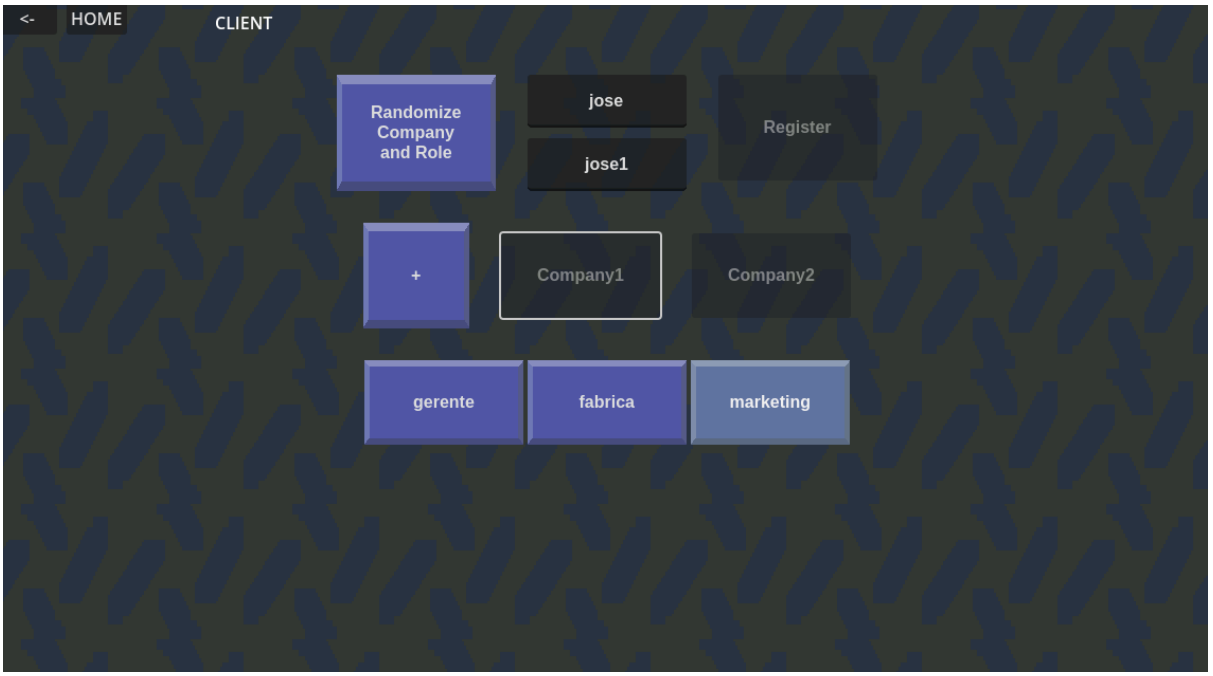

Figura 37: Pantalla de registro del jugador, habiendo seleccionado compañía.
Fuente: elaboración propia

También será posible randomizar la compañía y rol, pulsando en el botón de arriba a la izquierda.

Una vez todos los datos se hayan rellenado, se habilita el botón Register y puede comenzar realmente el juego.

El Gerente de una compañía es quien ha de seleccionar los componentes de la empresa, tras discutir con el resto del equipo sus objetivos y la estrategia a seguir. Una vez se hayan seleccionado todos los componentes obligatorios, es decir, aquellos que ocupan un slot, se puede pasar a la pantalla de turno.

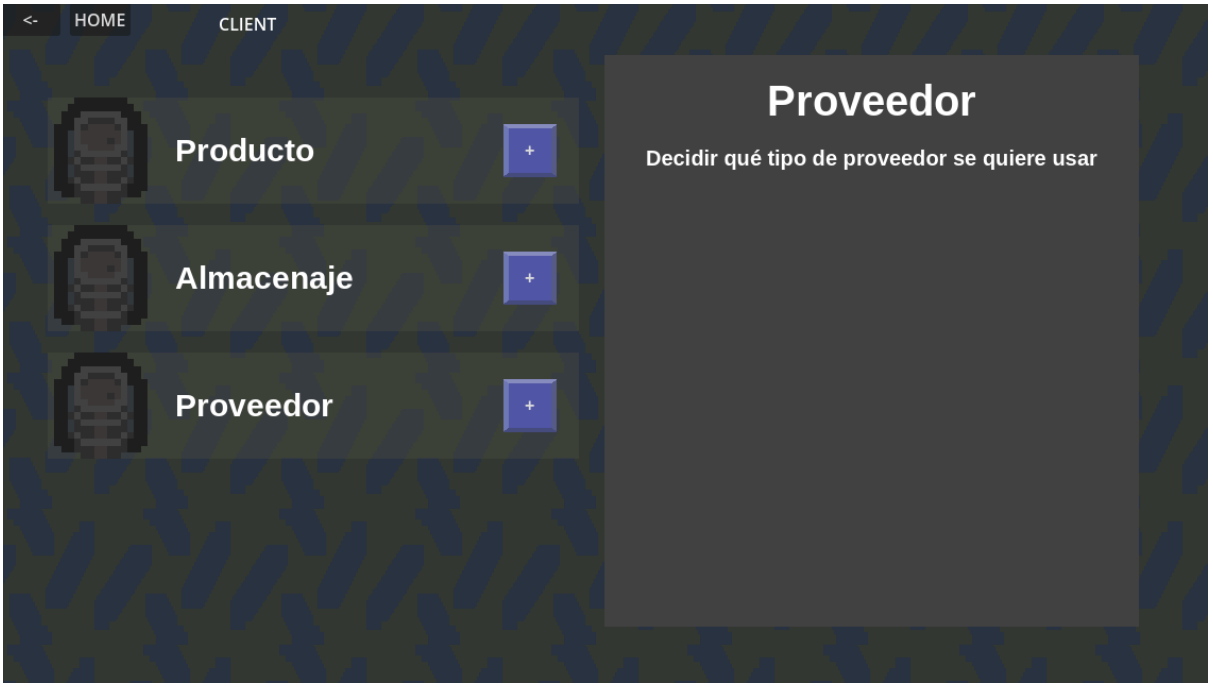

Figura 38: Pantalla del área de Capacidad, mostrando los componentes obligatorios.
Fuente: elaboración propia

En esta pantalla se muestra un botón que permite finalizar el turno, así como un slider que permitirá seleccionar el precio de venta este turno en caso de que el rol sea Marketing, como se muestra en la imagen, o la cantidad a producir este turno en caso de que el rol sea Fábrica.

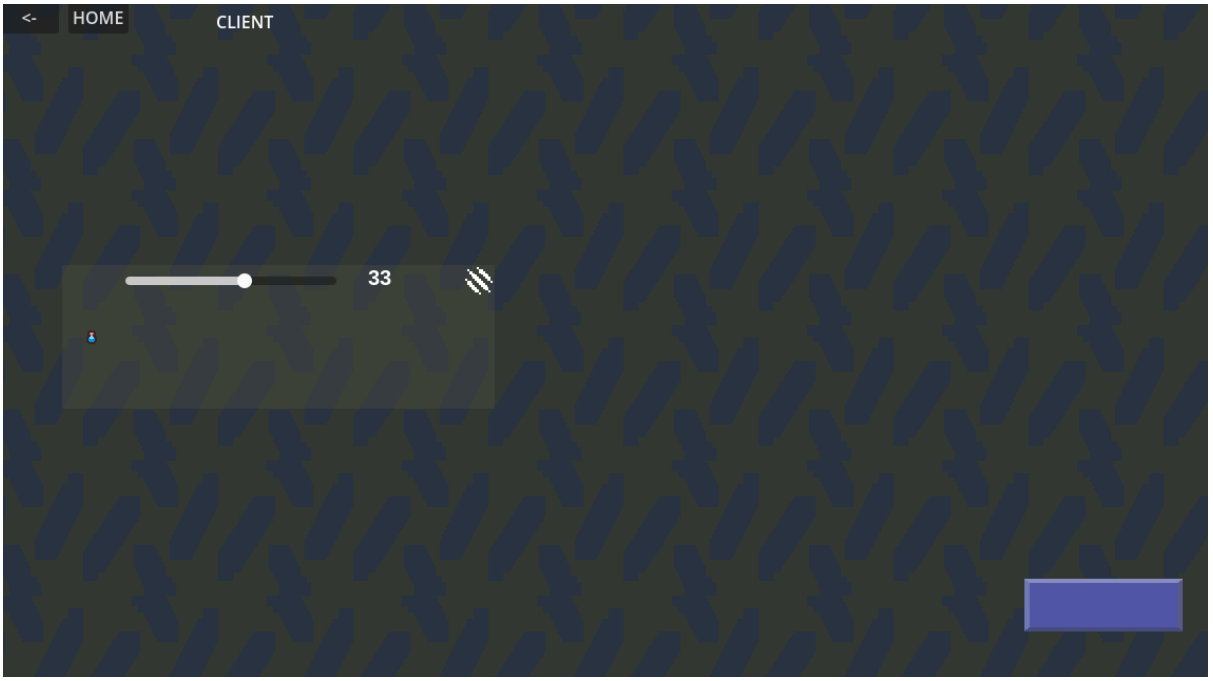

Figura 39: Pantalla de finalización de turno.
Fuente: elaboración propia

El botón de finalizar turno se queda presionado, y una vez todos los jugadores lo han pulsado, comienza la fase del minijuego, cuyo flujo se explica más adelante, y se realizan los cálculos del final del turno tal y como se definen en el apartado de diseño lógico.

Tras realizar los cálculos, las estadísticas de cuántas unidades se han vendido este turno, a qué precio, con qué coste, beneficio, etc se guardan en el diccionario correspondiente del producto. Esta información puede verla el jugador accediendo a la pantalla de información, donde se mostrarán estos datos, desglosados por turno.

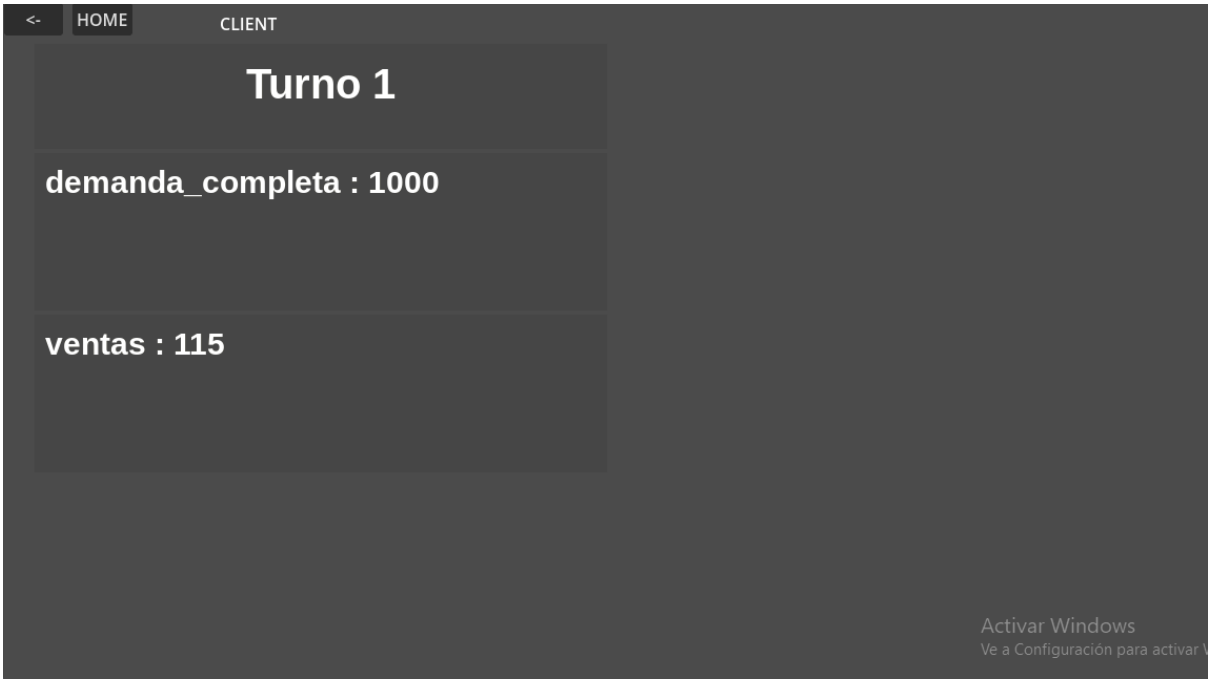

Figura 40: Pantalla de información, mostrando los datos de un turno.
Fuente: elaboración propia

Estructura de minijuegos

Elementos comunes

Todos los minijuegos han de tener ciertos elementos en común. Esto es necesario para que pueda coordinarse con los demás componentes del juego y mostrar la puntuación de cada equipo.

TeamScoreSpawner: El elemento que creará los TeamScore de cada equipo, un nodo sincronizado que se encargará de gestionar la puntuación. En Main se creará una RichTextLabel que muestre esta puntuación, conectándolo mediante una señal desde TrueMain.

TeamScore: El nodo encargado de almacenar la puntuación de cada equipo en el minijuego. Contiene un MultiplayerSynchronizer para sincronizar el valor de la puntuación, y se instanciará un TeamScore por cada equipo, cambiando este valor desde la escena principal y emitiendo una señal después para actualizar el valor que se muestra en pantalla.

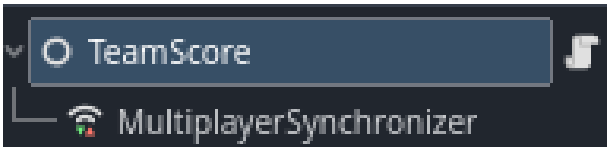

Figura 41: Árbol de nodos de la escena TeamScore.
Fuente: elaboración propia

GameTimer: Un temporizador con la duración del juego. Su señal de timeout se conecta a un método de TrueMain, que se encargará de cambiar de escena y realizar los cálculos una vez finalice el minijuego. En Main se creará una RichTextLabel que muestre el tiempo restante.

PlayerSpawner: El elemento que creará los personajes de cada jugador, que serán nodos sincronizados de manera que cada jugador pueda ver el movimiento y acciones de los demás. A diferencia de los demás nodos sincronizados, es el cliente quien tendrá autoridad sobre estos nodos, para que pueda moverlos y afectar a sus parámetros según se necesite. Además, cada jugador necesitará tener dentro un MultiplayerSynchronizer para poder sincronizar sus parámetros con el resto de jugadores.

Función new_player(): Una función que recibe como parámetro de entrada el id multijugador de un cliente. Se llama dos veces: la primera para instanciar al personaje del jugador en su mismo cliente, y la segunda mediante RPC configurado con el parámetro “call_remote”, para instanciarlo en todos los demás clientes.

Flujo

En este apartado se detallará el flujo que sigue un minijuego, desde que comienza y se instancian sus componentes hasta que finaliza. Esto se mostrará con el código del minijuego que se ha implementado, aunque será muy similar para todos los demás, debido a los elementos comunes que se han explicado en el apartado anterior.

En la pantalla donde se decide cuántas unidades producir cada turno y a qué precio, se encuentra el botón de finalizar turno. Este botón se queda presionado al pulsarlo, y una vez todos los jugadores lo han pulsado, se pasa a la pantalla de tutorial del minijuego, donde se muestra una breve descripción mediante texto e imágenes explicando el funcionamiento del juego en concreto.

Una vez todos los jugadores han leído el tutorial y pulsado el botón de comenzar, se emite una señal desde Global. Esta señal la emite el estado MinigameState en su función enter(), pasándole como parámetro el ID del minijuego que debe cargar, tal y como se ha configurado en el turno. Mientras el server realiza los cálculos pertinentes al número de ventas, gastos e ingresos de cada empresa, la señal enviada por el estado es recibida por el método start_minigame() de Main.

Este método comprueba el ID para instanciar la escena del minijuego, crea la etiqueta que muestra el tiempo restante y la conecta al timer de la escena que ha creado, y llama a la función new_player() pasándole como parámetro el ID multijugador necesario para que la autoridad sobre el jugador la tenga el cliente y no el servidor.

La función new_player() devuelve una referencia al jugador que se ha creado, lo cual usa Main para hacer que la cámara siga al jugador apuntando a su nodo camera_node, y para asignarle el ID de su equipo.

También conecta la señal timeout del contador de juego al método minigame_ended(), de manera que al terminar el minijuego se llame a esa función y el juego pase a mostrar la pantalla principal.

Al añadir al árbol la escena principal del minijuego, esta se encarga de instanciar los TeamScore, que guardarán la puntuación de cada equipo, y puesto que el nodo padre de estos componentes es un elemento común entre todos los minijuegos, Main puede iterar por él para crear una etiqueta que muestre la puntuación de cada equipo en tiempo real, conectando la etiqueta con el nodo mediante señales.

Una vez realizados todos estos pasos, el minijuego dará comienzo. Cuando termine su contador, se mostrará una pantalla con la puntuación obtenida, tras lo cual el juego volverá al menú principal, habiendo realizado los cálculos pertinentes al turno, como pueden ser el cálculo de costes, unidades vendidas y producidas, etc.

Apple Catch

Este es el primer minijuego a implementar, en el que cada jugador manejará un personaje, moviéndose por el mapa mientras compiten para conseguir la mayor cantidad de frutas posible.

Los tres componentes principales de este minijuego son la escena principal, el jugador y las frutas.

Escena Base

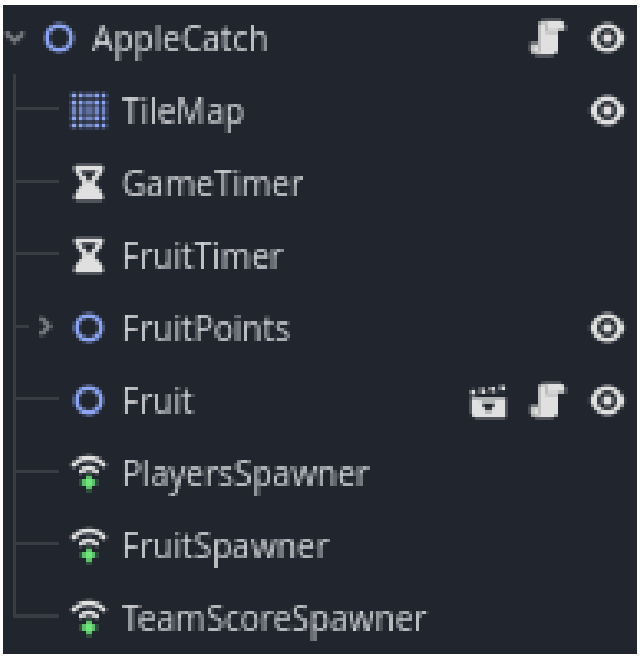

Figura 42: Árbol de nodos de la escena base del minijuego AppleCatch.
Fuente: elaboración propia

El nodo Tilemap es un mapa de casillas, cargando una textura para delimitar el suelo y las paredes del escenario. Las casillas usadas para ello tienen una CollisionShape2D, configuradas en la capa 1.

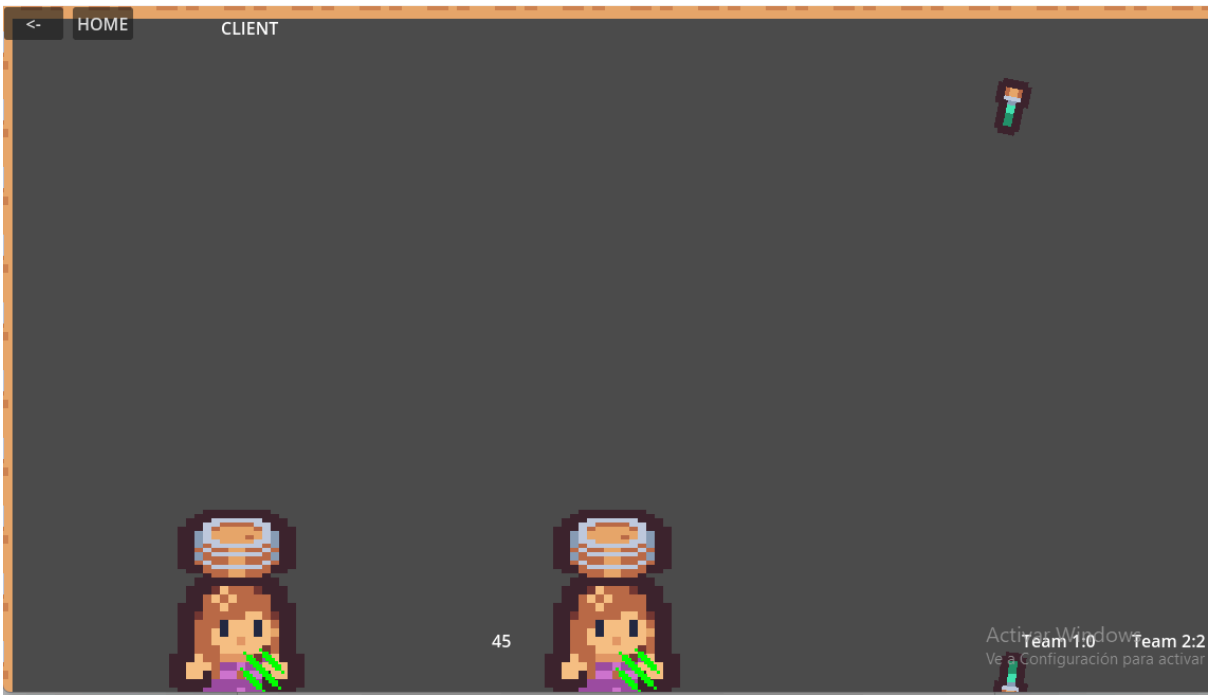

Figura 43: Captura de pantalla del minijuego AppleCatch, con dos jugadores.
Fuente: elaboración propia

Los tres spawners (PlayersSpawner, FruitSpawner, y TeamScoreSpawner) instancian los jugadores, los frutos y la puntuación, respectivamente, permitiendo su sincronización entre todos los peers.

FruitTimer es un contador que emite una señal para hacer aparecer una fruta en un punto aleatorio cada 2 segundos. FruitPoints contiene todos estos puntos, repartidos cerca del límite superior del mapa.

Fruta

La fruta está compuesta por una imagen (Sprite2D) y una hit box (CollisionShape2D). Está afectada por la gravedad, por lo que cae hacia abajo lentamente en cuanto entra en juego, y en _physics_process() se rota ligeramente hacia la derecha el sprite para que gire al tiempo que cae.

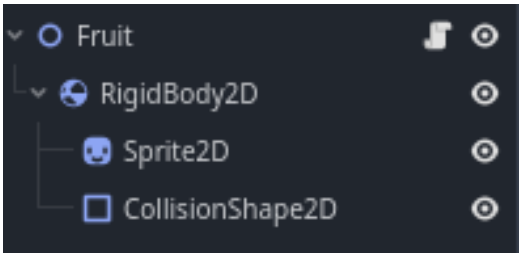

Figura 44: Árbol de nodos de la escena Fruit.
Fuente: elaboración propia

La fruta se instancia desde la escena principal, pasándole como parámetro su posición y un booleano que decide si se trata de una fruta (is_bad = false) o una piedra (is_bad = true).

```
func spawn_fruit():
    if Global.is_host:
        var is_bad = false
        var rng = RandomNumberGenerator.new()
        rng.randomize()
        var i = rng.randi_range(1, 6)
        if i == 1:
            is_bad = true
        var point = points_array.pick_random()
        spawn_fruit_on_everyone.rpc(point.x, point.y, is_bad)
```

Figura 45: Captura del método spawn_fruit(), dentro de la escena principal.
Fuente: elaboración propia

Su colisión está configurada para que exista en la capa 2, y detecte colisiones en la capa 1 y 2, siendo la capa 1 el suelo, y la capa 2 la colisión entre frutas y cesta.

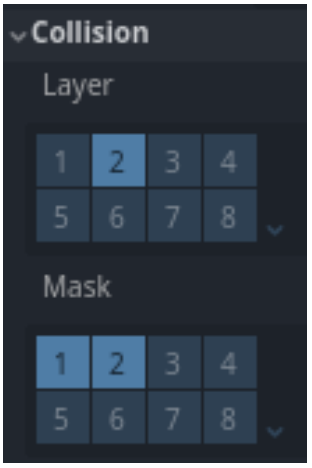

Figura 46: Configuración de colisiones de la fruta en el minijuego AppleCatch.
Fuente: elaboración propia

Dentro de la fruta existe una variable de tipo booleano que dicta si se trata de una fruta, que suma puntos, o una roca, que resta puntuación. Esta variable recibe su valor de la escena principal al crear la fruta, y después en su _ready() se comprueba qué valor tiene, y se le asigna una imagen de una poción (si se trata de fruta) o de una roca.

También es en _ready() donde, en función de su valor, se le asigna a un grupo u otro. Este grupo es el que será usado por la cesta para comprobar si debe añadir o restar puntuación.

```
var is_bad := false

func _ready():
    if is_bad:
        #print("c pone mala")
        rigid_body.add_to_group("bad_fruit")
        sprite.texture = bad_fruit_texture
    else:
        pass
        #print("c pone buena")
        rigid_body.add_to_group("good_fruit")
```

Figura 47: Captura del método _ready() dentro de la escena Fruit.
Fuente: elaboración propia

Por último, la fruta tiene un método conectado a su colisión, para que al tocar el suelo o la cesta se elimine.

Personaje del jugador

Cada uno de los jugadores manejará su propia instancia del personaje, estando todos ellos sincronizados en tiempo real para que cada jugador pueda ver la posición de los demás en la partida.

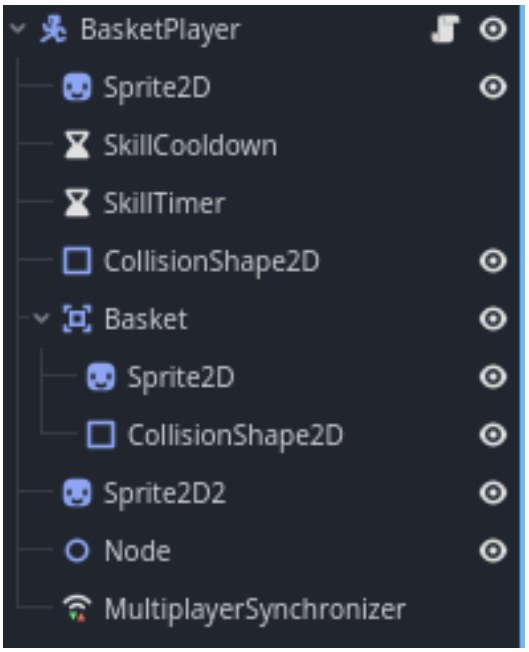

Figura 48: Árbol de nodos de la escena BasketPlayer.
Fuente: elaboración propia

El jugador se compone de un Sprite2D que muestra su imagen, dos timers, SkillCooldown que cuenta el tiempo que se tarda en poder utilizar la habilidad de nuevo, y SkillTimer, que cuenta el tiempo que la habilidad está activa.

Cuenta también con un MultiplayerSynchronizer, que sincroniza su posición y cuya autoridad la tiene el cliente, posibilitando que este pueda manejar su movimiento sin necesidad de hacer llamadas al servidor para ello.

Al pulsar la barra espaciadora, comprueba si el contador SkillCooldown está parado, y en caso de que lo esté inicia ambos contadores y pone la variable skill_active a true.

Esta variable se usará para decidir el valor de la variable skill, por la que multiplicaremos la velocidad de movimiento o de salto.

```
if Input.is_action_just_pressed("ui_up") and is_on_floor():
    var skill = 1
    if skill_active:
        skill = 2
    velocity.y = JUMP_VELOCITY * skill
```

Figura 49: Captura del código que gestiona el salto, dentro del método _physics_process().
Fuente: elaboración propia

El jugador podrá moverse mediante las flechas o WASD, utilizando la flecha de arriba o la tecla W para saltar. Todo el movimiento está dentro de la función _physics_process(), que comprueba constantemente si se está pulsando una de estas teclas. En caso de que las teclas de movimiento no se estén pulsando, el personaje decelera hasta detenerse completamente.

```
var direction = Input.get_axis("ui_left", "ui_right")
if direction:
    var skill = 1
    if skill_active:
        skill = 3
    velocity.x = direction * SPEED * skill
else:
    velocity.x = move_toward(velocity.x, 0, SPEED)
```

Figura 50: Captura del código que gestiona el movimiento horizontal, dentro del método _physics_process().
Fuente: elaboración propia

La habilidad está activa durante 3 segundos, y tarda 10 segundos en recuperarse, multiplicando la velocidad de movimiento por 3 y la de salto por 2. Con esto se busca ofrecer una ventaja fuerte durante poco tiempo, posibilitando que un jugador pueda reposicionarse o robarle una fruta a un rival gracias a su mayor movilidad y a saltar más que el resto.

Otro Sprite ha sido añadido para mostrar cuándo la habilidad está disponible, siendo esta imagen 3 líneas blancas superpuestas al personaje. Dentro del juego se modifica su color para que se muestre verde cuando la habilidad está disponible, y rojo cuando la habilidad todavía no se ha recargado tras usarse.

El Node2D llamado Node es el nodo al que ha de apuntar la cámara principal cuando el juego comienza, de manera que siga al jugador.

A diferencia de la fruta, el jugador tiene dos colisiones. Una es la del propio jugador, que está configurada para detectar colisiones con la capa 1. La otra es la de la cesta, que está configurada en la capa 2 y cuya máscara apunta también a la capa 2, de manera que pueda detectar colisiones con la fruta.

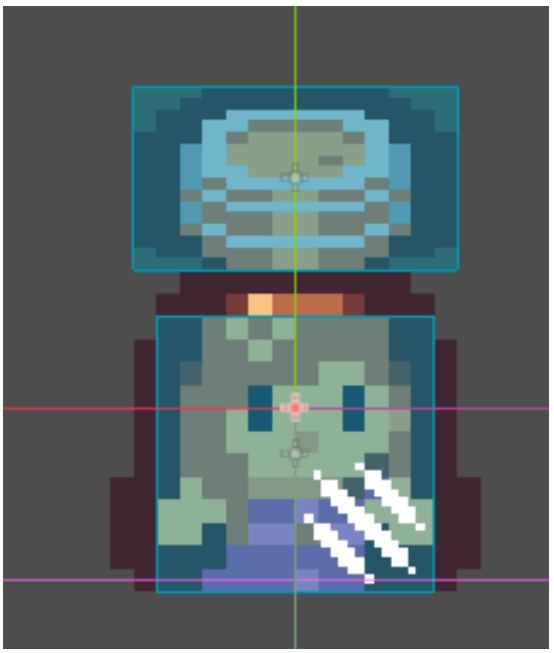

Figura 51: Jugador del minijuego AppleCatch, mostrando ambas hitbox, la del jugador y la de la cesta sobre su cabeza.
Fuente: elaboración propia

Al detectar la cesta una colisión, comprueba si el objeto con el que ha colisionado está dentro del grupo de fruta, en cuyo caso emite una señal para incrementar en 2 la puntuación, o si está dentro del grupo de piedras, en cuyo caso reduce en 2 la puntuación.

Al estar en la capa 2 y detectar objetos en esta misma capa, las cestas de los jugadores detectan colisiones entre sí, pero al comprobar en esa colisión si el objeto pertenece a un determinado grupo, se evita que suceda nada en ese caso.

```gdscript
func apple_fell(body):
	
	if body.is_in_group("good_fruit"):
		#print("SE EMITE")
		add_points.emit(team_id, +2)
	if body.is_in_group("bad_fruit"):
		add_points.emit(team_id, -2)
```

Figura 52: Captura del método apple_fell() dentro de BasketPlayer.
Fuente: elaboración propia

Las señales emitidas por el jugador conectan con la escena base, que comprueba el ID del equipo, pasado como parámetro en dicha señal, y modifica la puntuación del TeamScore pertinente mediante llamada rpc_id al servidor.

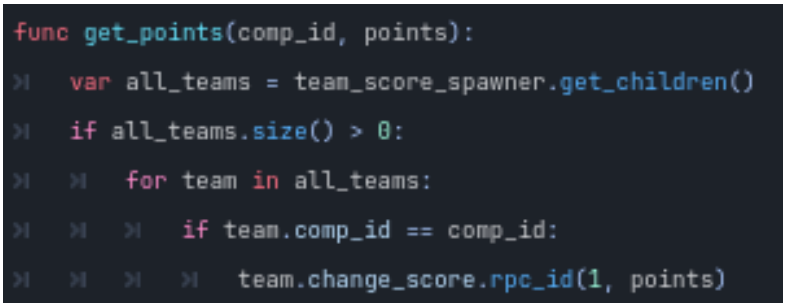

Figura 53: Captura del método get_points() dentro de la escena base.
Fuente: elaboración propia

Al ser la colisión de la cesta de tipo Area2D, no es afectada por las físicas ni causa ningún movimiento en otros objetos, permitiendo que un jugador salte por encima de otro sin colisionar con su cesta y que la cesta no bloquee al jugador si tocase una pared.

Para facilitar que se recojan las frutas, la hitbox de la cesta se ha hecho ligeramente más ancha que la del jugador.

Objetivos de desarrollo sostenible

En septiembre de 2015 la Asamblea General de las Naciones Unidas aprobó por unanimidad la Agenda 2030 para el Desarrollo Sostenible, comprendiendo esta un plan de acción con el que se pretende lograr un mundo mejor y hacer grandes avances para las personas, el planeta, la prosperidad y la paz mundial para el año 2030.

Figura 54: Objetivos de Desarrollo Sostenible.
Fuente: https://www.pactomundial.org/que-puedes-hacer-tu/ods/?gad_source=1&gclid=CjwKCAjw2Je1BhAgEiwAp3KY74A7cPVwR9MpsSzDi1umGp2T20gU9NIFb2uyB2Y1c3576-LeMufu5hoC1OAQAvD_BwE

Este plan contiene 17 objetivos de desarrollo sostenible, con los que se pretende alcanzar las metas propuestas para esa fecha. En este apartado, definiremos los objetivos a los que este trabajo de fin de grado pueda contribuir.

4 - Educación de calidad

Al tratarse nuestro producto de un videojuego enfocado a la formación, tanto para empleados de una empresa como para alumnos universitarios, contribuye principalmente a este objetivo de desarrollo sostenible.

Para que sea considerada una educación de calidad, considero que ha de ser capaz de alcanzar a todos los alumnos e impartir los conocimientos de forma suficiente. Cada persona aprende de manera distinta: hay quienes pueden absorber todo el conocimiento necesario de un libro, mientras que otros pueden aprender mejor con clases presenciales o ejercicios prácticos. Es por esto que cuantas más vías haya para el aprendizaje, mejor, y ahí

es donde entra nuestro producto, ofreciendo un complemento al aprendizaje que pueda servir para afianzar conocimientos de forma más amena que una clase o lección.

11 - Ciudades y comunidades sostenibles

Este objetivo de desarrollo sostenible tiene como fin lograr que las ciudades y los asentamientos humanos sean inclusivos, seguros, resilientes y sostenibles. Una parte importante de esto depende de los propios habitantes de dichas ciudades, y su concienciación con el medio ambiente, el reciclaje y otros hábitos de vida sostenibles.

Modificando el juego sería posible ofrecer escenarios enfocados a concienciar a nuestros jugadores con estos temas, aumentando así la conciencia ciudadana sobre este ámbito y contribuyendo de este modo a este objetivo.

Conclusiones

En el proceso de realizar este trabajo de fin de grado se ha realizado un análisis del mercado de simuladores empresariales, se ha desarrollado una estrategia de marketing en base a dicho análisis, se han recopilado requisitos y dividido la carga de trabajo en fases diferenciadas en función de su prioridad, y se han diseñado e implementado sistemas para programar el videojuego en cuestión acorde a los requisitos establecidos.

Considero que ha sido una gran oportunidad para aprender, y poner en práctica tanto los conocimientos aprendidos en la carrera como los obtenidos en mi trabajo y desarrollando videojuegos en mi tiempo libre.

Una de las mayores dificultades a las que he debido hacer frente en la realización de este TFG es a la falta de tiempo, ya que debido a circunstancias personales, entre las que se incluye el trabajar a tiempo completo desarrollando software para una empresa, no he podido desarrollar tanto del videojuego como me hubiese gustado, habiendo completado tan sólo la primera fase.

Sin embargo, considero que al haber definido las prioridades de esta manera, lo que ha dado tiempo a desarrollar es una base más que suficiente para proseguir el desarrollo del juego, puesto que todos los sistemas se han diseñado teniendo la versión final del juego en mente.

En mi opinión, basada en mi experiencia tanto en mis proyectos personales como en los desarrollados en mi trabajo, ser capaz de organizar las prioridades y dependencias de un proyecto, y medir con precisión el tiempo que pueden llevar es uno de los desafíos más importantes a los que se enfrenta un desarrollador. Una estimación no sólo da un tiempo total en que en teoría se puede terminar un proyecto, sino que, cuando está desglosado adecuadamente en fases y tareas, permite detectar con antelación si hay un retraso en alguna parte del proceso, ya sea por errores inesperados o por una estimación demasiado baja.

Es por esto que creo que detectar esto a tiempo y poder corregir el rumbo del proyecto es una habilidad importante, y que no se valora lo suficiente.

Anexo A: Tablas e imágenes

Figura 1: Mapa mundial con las emisiones totales de CO2 a la atmósfera, organizados por color.
Fuente: https://www.eldebate.com/sociedad/20221107/espana-solo-provoca-0-58-contaminacion-mundial-51-veces-menos-china_71049.html#:~:text=España%20ocupó%20el%20puesto%2027,de%20CO2%20emitidas%20por%20persona. 6
Figura 2: Logo de Cesim Business Simulations.
Fuente: https://www.cesim.com 8
Figura 3: Captura de pantalla de la simulación de GestioNet “Innovación GP”.
Fuente: https://simuladores-empresariales.com/innovacion-gp/ 9
Figura 4: Modelo Business Canva.
Fuente: https://es.wikipedia.org/wiki/Archivo:Plantilla_CANVAS.pdf 14
Figura 5: Matriz de Slack y Lewis.
Fuente: https://www.researchgate.net/figure/Operations-Strategy-matrix-Slack-Lewis-2011_fig1_255820198 15
Figura 6: Ejemplo de un árbol de nodos perteneciente a una escena de Godot.
Fuente: elaboración propia 19
Figura 7: Ejemplo de uso de la keyword @onready en el código.
Fuente: elaboración propia 20
Figura 8: Menú de configuración de capa y máscara de colisiones.
Fuente: elaboración propia 22
Figura 9: Ejemplo de una escena de Godot con nodos de tipo Control y CanvasLayer.
Fuente: elaboración propia 22
Figura 10: Ejemplo de una escena de UI construida con nodos de tipo control.
Fuente: elaboración propia 23
Figura 11: Pantalla de previsualización de un Tema de Godot.
Fuente: elaboración propia 23
Figura 12: Carpetas principales del proyecto en Godot.
Fuente: elaboración propia 26
Figura 13: Subcarpetas dentro de la carpeta Assets.
Fuente: elaboración propia 26
Figura 14: Subcarpetas dentro de la carpeta Minigames.
Fuente: elaboración propia 27
Figura 15: Subcarpetas dentro de la carpeta Screens.
Fuente: elaboración propia 27
Figura 16: Subcarpetas y elementos sueltos dentro de la carpeta utility.
Fuente: elaboración propia 28
Figura 17: Árbol de nodos de la escena Main.
Fuente: elaboración propia 28

Figura 18: Captura de la pantalla mostrando la escena MainUI dentro del juego. Se resalta el MarginContainer donde se añaden el resto de pantallas.
Fuente: elaboración propia 29
Figura 19: Árbol de nodos de la escena MainUI.
Fuente: elaboración propia 30
Figura 20: Pantalla de configuración del juego.
Fuente: elaboración propia 31
Figura 21: Árbol de nodos de la escena Turn.
Fuente: elaboración propia 32
Figura 22: Pantalla de configuración del turno.
Fuente: elaboración propia 32
Figura 23: Pantalla de configuración de la demanda.
Fuente: elaboración propia 33
Figura 24: Árbol de nodos de la escena Company.
Fuente: elaboración propia 33
Figura 25: Captura de pantalla mostrando los componentes de cada slot, vacíos.
Fuente: elaboración propia 34
Figura 26: Pantalla de decisión del Proveedor.
Fuente: elaboración propia 35
Figura 27: Árbol de nodos de la pantalla de selección de área.
Fuente: elaboración propia 36
Figura 28: Árbol de nodos del componente ProductSlider.
Fuente: elaboración propia 36
Figura 29: Captura del componente ProductSlider dentro del juego.
Fuente: elaboración propia 37
Figura 30: Árbol de nodos de la escena DesplegableBase.
Fuente: elaboración propia 37
Figura 31: Función _ready() del Desplegable.
Fuente: elaboración propia 38
Figura 32: Pantalla de selección de área.
Fuente: elaboración propia 39
Figura 33: Pantalla inicial del juego.
Fuente: elaboración propia 40
Figura 34: Pantalla de comenzar la partida, mostrada al Host.
Fuente: elaboración propia 41
Figura 35: Pantalla de tutorial inicial.
Fuente: elaboración propia 41
Figura 36: Pantalla de registro del jugador.
Fuente: elaboración propia 42
Figura 37: Pantalla de registro del jugador, habiendo seleccionado compañía.
Fuente: elaboración propia 42
Figura 38: Pantalla del área de Capacidad, mostrando los componentes obligatorios.
Fuente: elaboración propia 43
Figura 39: Pantalla de finalización de turno.
Fuente: elaboración propia 44
Figura 40: Pantalla de información, mostrando los datos de un turno.
Fuente: elaboración propia 44

Figura 41: Árbol de nodos de la escena TeamScore.
Fuente: elaboración propia 45
Figura 42: Árbol de nodos de la escena base del minijuego AppleCatch.
Fuente: elaboración propia 47
Figura 43: Captura de pantalla del minijuego AppleCatch, con dos jugadores.
Fuente: elaboración propia 48
Figura 44: Árbol de nodos de la escena Fruit.
Fuente: elaboración propia 48
Figura 45: Captura del método spawn_fruit(), dentro de la escena principal.
Fuente: elaboración propia 48
Figura 46: Configuración de colisiones de la fruta en el minijuego AppleCatch.
Fuente: elaboración propia 49
Figura 47: Captura del método _ready() dentro de la escena Fruit.
Fuente: elaboración propia 49
Figura 48: Árbol de nodos de la escena BasketPlayer.
Fuente: elaboración propia 50
Figura 49: Captura del código que gestiona el salto, dentro del método _physics_process().
Fuente: elaboración propia 51
Figura 50: Captura del código que gestiona el movimiento horizontal, dentro del método _physics_process().
Fuente: elaboración propia 51
Figura 51: Jugador del minijuego AppleCatch, mostrando ambas hitbox, la del jugador y la de la cesta sobre su cabeza.
Fuente: elaboración propia 52
Figura 52: Captura del método apple_fell() dentro de BasketPlayer.
Fuente: elaboración propia 52
Figura 53: Captura del método get_points() dentro de la escena base.
Fuente: elaboración propia 53
Figura 54: Objetivos de Desarrollo Sostenible.
Fuente: https://www.pactomundial.org/que-puedes-hacer-tu/ods/?gad_source=1&gclid=CjwKCAjw2Je1BhAgEiwAp3KY74A7cPVwR9MpsSzDi1umGp2T20gU9NIFb2uyB2Y1c3576-LeMufu5hoC1OAQAvD_BwE 54

Anexo B: Bibliografía

References

Alessandrelli, F. (2023, February 23). *Multiplayer in Godot 4.0: Scene Replication*. Godot Engine. Retrieved July 29, 2024, from https://godotengine.org/article/multiplayer-in-godot-4-0-scene-replication/

Arias, J., & Usero, N. (2023, March 13). *Un estudio revela el desplome de la capacidad de atención de los jóvenes por la digitalización*. The Objective. Retrieved July 29, 2024,

from https://theobjective.com/tecnologia/2023-03-13/capacidad-atencion-moviles-informacion/

Cock, F. (n.d.). *Archivo:Plantilla CANVAS.pdf*. Wikipedia. Retrieved July 29, 2024, from https://es.wikipedia.org/wiki/Archivo:Plantilla_CANVAS.pdf

Documentación de Godot 4.2. (n.d.). Godot Docs – 4.2 branch — Godot Engine (stable) documentation in English. Retrieved July 29, 2024, from https://docs.godotengine.org/en/stable/

El Crecimiento Exponencial de OpenAI: Ingresos Anualizados Triplican en Seis Meses. (2024, June 26). LinkedIn. Retrieved July 29, 2024, from https://es.linkedin.com/pulse/el-crecimiento-exponencial-de-openai-ingresos-en-seis-marcos-aqaqe

España solo provoca el 0,58 % de la contaminación mundial, 51 veces menos que China. (2022, November 7). El Debate. Retrieved July 29, 2024, from https://www.eldebate.com/sociedad/20221107/espana-solo-provoca-0-58-contaminacion-mundial-51-veces-menos-china_71049.html

Garbar, D. (2018, 11 11). *MVP Software Development*. belitsoft. https://belitsoft.com/custom-application-development-services/successful-saas-startups-mvp-lean-paying-customers-first

GestioNet. (n.d.). Simuladores Empresariales - Potencia habilidades de negocios. Retrieved July 29, 2024, from https://simuladores-empresariales.com

INEbase / Economía /Cuentas económicas /Contabilidad nacional anual de España: principales agregados / Últimos datos. (2023, September 18). Instituto Nacional de Estadística. Retrieved July 29, 2024, from https://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736177057&menu=ultiDatos&idp=1254735576581

Knight, S. (2021, October 27). *What Is Asset Flipping in Gaming and Is it Always Bad?* MakeUseOf. Retrieved July 29, 2024, from https://www.makeuseof.com/what-asset-flipping-gaming/

ODS. (n.d.). Objetivos Desarrollo Sostenible. https://www.pactomundial.org/que-puedes-hacer-tu/ods/?gad_source=1&gclid=CjwKCAjw2Je1BhAgEiwAp3KY74A7cPVwR9MpsSzDi1umGp2T20gU9NIFb2uyB2Y1c3576-LeMufu5hoC1OAQAvD_BwE

Operations Strategy matrix (Slack & Lewis, 2011) | *Download Scientific Diagram*. (n.d.). ResearchGate. Retrieved July 29, 2024, from https://www.researchgate.net/figure/Operations-Strategy-matrix-Slack-Lewis-2011_fig1_255820198