Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo de Fin de Grado

Otomatïkï: Desarrollo de un Producto Software Interactivo Basado en Mecánicas de Automatización y Simulación

Autor: Miguel Gutiérrez de la Cruz
Tutor(a): Jose María Barambones Ramírez

Madrid, Junio 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Otomatïkï: Desarrollo de un Producto Software Interactivo Basado en Mecánicas de Automatización y Simulación

Junio 2024

Autor: Miguel Gutiérrez de la Cruz
Tutor: Jose María Barambones Ramírez
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

El presente Trabajo de Fin de Grado contiene el planteamiento, diseño, desarrollo, evaluación y conclusiones del proceso de creación de un producto software interactivo basado en mecánicas de automatización y simulación llamado "Otomatïkï".

Otomatîkî es un videojuego de simulación, gestión y automatización, con un componente de exploración. El jugador despierta en un espacio definido sin conocer nada del mundo, y usando los recursos que encuentre tiene que llegar a un punto de progresión que es alcanzable utilizando las mecánicas que hay implementadas en el juego, siendo estas: exploración, obtención de recursos, fabricación de objetos e investigación de tecnologías, teniendo la opción de utilizar la construcción de edificios y la automatización de la producción.

La estética de Otomatïkï está completamente basada en hexágonos, que llenan mejor un espacio bidimensional y son mucho más atractivos desde la perspectiva del usuario que una cuadrícula al uso. Tanto el mapa, como los edificios y las cintas para la automatización están basadas en esta forma. Sin embargo, el inventario y todo lo que tiene que ver con los objetos y la forma de almacenarlos utiliza cuadrados, ya que es más intuitivo y organizado ver una tabla cuadrada para gestionar la posición de los objetos del juego.

Otomatîkî se ha realizado utilizando Unity como motor de desarrollo, y por lo tanto el código de este proyecto está escrito en C#. Para los Sprites de los objetos se ha utilizado Blender, procesando fotogramas de los objetos previamente modelados, ya que visualmente era más atractivo que utilizar una aplicación de dibujo en 2D como Aseprite, cuya calidad es menor.

Uno de los objetivos principales de este Trabajo de Fin de Grado, era estudiar la viabilidad de los videojuegos como herramienta clínica de detección de rasgos motivacionales y perfiles de jugador. Esto se ha realizado de manera conjunta con otros dos alumnos, para cubrir más rasgos entre los tres, y así poder llegar a una conclusión con una investigación más completa detrás.

Aunque se ha intentado realizar estas pruebas con los videojuegos diseñados, se ha llegado a un resultado no concluyente, debido principalmente a la falta de tiempo de prueba, de acciones que registraran métricas y de viabilidad de los videojuegos que se diseñaron de cara a realizar esta tarea.

Abstract

This Bachelor's Thesis presents the planning, design, development, evaluation, and conclusions of the process of creating an interactive software product based on automation and simulation mechanics called "Otomatïkï".

"Otomatïkï" is a simulation, management, and automation video game with an exploration component. The player wakes up in a defined space without knowing anything about the world, and using the resources found, must reach a progression point that is achievable by using the mechanics implemented in the game, which include exploration, resource gathering, object crafting, and technology research, with the option to use building construction and production automation.

The aesthetics of Otomatïkï are entirely based on hexagons, which fill a two-dimensional space better and are much more attractive from the user's perspective than a traditional grid. The map, buildings, and automation belts are all based on this shape. However, the inventory and everything related to objects and their storage uses squares, as it is more intuitive and organized to see a square grid for managing the position of game objects.

Otomatïkï was developed using Unity as the development engine, and therefore the code for this project is written in C#. For the object sprites, Blender was used, processing frames of pre-modeled objects, as it was visually more appealing than using a 2D drawing application like Aseprite, whose quality is lower.

One of the main objectives of this Bachelor's Thesis was to study the feasibility of video games as a clinical tool for detecting motivational traits and player profiles. This was carried out jointly with two other students to cover more traits among the three of us, thus achieving a more comprehensive conclusion backed by more extensive research.

Although an attempt was made to conduct these tests with the designed video games, the result was inconclusive, mainly due to the lack of testing time, actions that recorded metrics, and the viability of the designed video games for carrying out this task.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación

De cara a buscar un tema sobre el que desarrollar mi TFG, quise buscar algo que al mismo tiempo aunase los conceptos y las habilidades adquiridas a lo largo de los cuatro años de la carrera con algo que me apasionara, para crear un producto atractivo y que me resultase interesante desarrollar. Además de la programación y lo estrictamente dependiente de la informática, este producto debía tener partes tanto de ingeniería de software como de gestión de proyectos y evaluación del producto.

A raíz de haber asistido en el primer cuatrimestre del curso 2023-2024 a la asignatura de Fundamentos de Videojuegos impartida por mi tutor de este Trabajo de Fin de Grado, decidí que mi TFG debía tener que ver con los videojuegos, ya que desde mi punto de vista contiene todos los puntos que he dicho anteriormente, y aparte es un tema que me genera interés.

Al ser los videojuegos un campo basado en el diseño centrado en el usuario, realizar una evaluación del producto, y teniendo la referencia de la asignatura no me fue complicado tomar la decisión de realizar el trabajo desarrollando uno.

No es raro ver juegos en el ámbito de la psicología, sirviendo tanto para educar como para detectar perfiles o comportamientos, de manera amena para el sujeto estudiado, y por lo tanto también se incluye en la motivación comprobar si "Otomatïki", junto con otros dos productos similares pero de géneros diferentes, podría servir como herramienta clínica para evaluar rasgos motivacionales mediante un MTQ (Motivational Trait Questionnaire), y detectar perfiles de jugador mediante un Test de Bartle.

1.2. Planificación

A nivel de planificación del proyecto, se realizó un diagrama de Gantt que definía las tareas y los requisitos que tenía que cumplir este TFG. Este diagrama actualizado a fecha de realización de esta Memoria de Seguimiento se puede

encontrar en el punto 2.3 de este mismo documento. Aunque la planificación no ha variado, ya que no se han modificado los requisitos ni se han añadido nuevos, a excepción de concretar el MTQ (que aún de manera conceptual sí que se especificaba en el Plan de Trabajo), ha variado el tiempo de algunas partes de la implementación, sobre todo de cara a las últimas semanas de esta misma. También ha cambiado la distribución de las tareas dentro del proceso de mejora, ya que al final solo se va a realizar una tanda de evaluaciones donde se tendrán en cuenta tanto el GEQ como el MTQ.

En la planificación se obtuvieron las siguientes fases de desarrollo del proyecto, en base a adaptar las fases de ciclo de vida del software a este producto [1]:

1. Definición de requisitos y tareas
2. Preproducción (diseño)
3. Implementación
4. Verificación
5. Pulido
6. Mantenimiento (mayoritariamente correctivo)

Estas fases son todas las que se planificaron para el proyecto, siendo la 3 y la 4 repetidas para todas las mecánicas que se implementaron en el producto. El resto se realizaron al principio (1 y 2) o al final tras implementar todas las mecánicas (5 a nivel visual y la 6 para corregir cualquier error que pudiera verse durante el pulido final o después).

1.3. Seguimiento

A nivel de seguimiento del proyecto se pueden diferenciar dos puntos principales: las reuniones cada dos semanas con el tutor, a fin de monitorizar el progreso realizado y recibir orientación sobre cómo continuar con el desarrollo del producto, y la realización y constante actualización de un Trello donde se llevó un registro de todas las tareas a realizar a partir de un backlog, que se iban actualizando según estaban en producción, por revisar o terminadas. un backlog, que se iban actualizando según estaban en producción, por revisar o terminadas.

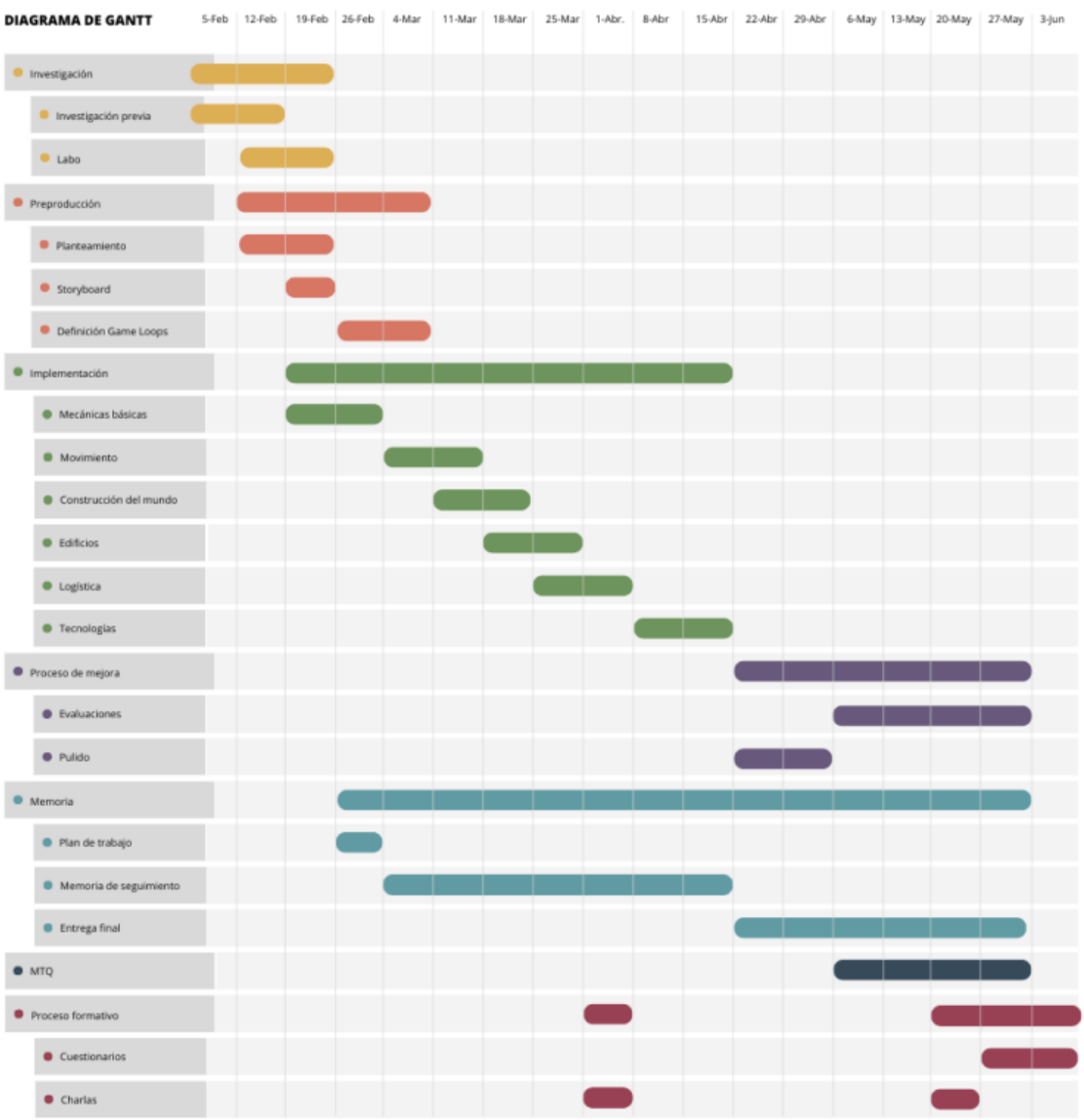


Figura 1.1: Diagrama de Gantt del trabajo completo

Capítulo 2

Estado del arte

2.1. Experiencia previa

Por haber cursado la asignatura de Fundamentos de Videojuegos el cuatrimestre anterior, ya tenía nociones sobre el desarrollo de videojuegos, las fases del desarrollo, o qué era necesario en cada una, por lo que no necesité una referencia bibliográfica al respecto. Sim embargo, sí que consulté las diferentes diapositivas de esa asignatura para resolver algunas dudas concretas que tenía.

2.2. ¿Qué es un juego?

Definir qué es un juego es una tarea complicada, pero es necesaria a la hora de hablar de diseño de videojuegos [2]. Para dar esta definición, primero hay que entender el juego como un subconjunto de jugar, que implica que se puede jugar a cosas sin que tenga que ser un juego propiamente dicho (como por ejemplo un balancín, en el que se juega sin que se constituya un juego como tal), y a la vez entender el jugar como un componente de cada juego, ya que los juegos están hechos para jugarse, siendo esto último una parte muy importante del concepto que se está intentando definir. Teniendo esto en cuenta, junto con la opinión de muchos autores que el libro previamente referenciado recoge, y con la opinión de los autores del propio libro, podemos concluir que un juego es un sistema donde los jugadores entran en un conflicto artificial, definido por reglas y cuyo resultado es cuantificable.

Para concretar esta definición, es importante destacar que el conflicto que define a todos los juegos puede tomar muchas formas, ya que dentro de este concepto se engloban la competición, la cooperación y el conflicto con el entorno sin necesidad de otros jugadores. También cabe destacar que la existencia de un resultado cuantificable no tiene por qué expresarse de manera explícita, sino que también se entiende como resultado cuantificable el haber realizado una progresión de cualquier tipo sin necesidad de que exista un marcador numérico. Los jugadores interactúan con el sistema (conjunto de elementos que se interrelacionan e interactúan entre si componiendo un conjunto complejo), y sus acciones

son limitadas por reglas.

2.3. Proyecto software enfocado a la creación y diseño de videojuegos

El Project Management Institute define un proyecto como una serie de tareas estructuradas que son ejecutadas cuidadosamente para obtener un resultado deseado. Se define también como una serie de esfuerzos temporales que crean valor a través de productos, servicios y procesos [3].

La gestión de un proyecto de software puede seguir varias metodologías, que podrían clasificarse de dos maneras [4]:

1. Las metodologías tradicionales, que siguen una planificación con una estructura muy estricta y marcada. En estas metodologías hay una fase de planificación en la que se enuncian unos requisitos en torno a los que se irá desarrollando el proyecto.Dos ejemplos de estas metodologías son el modelo de ciclo de vida en cascada, en el cual se cambia de fase cuando se han terminado todos los objetivos de la etapa anterior, por lo que sigue una secuencia ordenada, y el modelo de ciclo de vida incremental, el cual funciona a base de incrementos, que son funcionalidades que se van añadiendo al diseño, con la posibilidad de enunciar nuevos requisitos según va avanzando el desarrollo.

2. Las metodologías ágiles, que por el contrario, dividen cada fase en fases más pequeñas llamadas sprints. Al acabar uno de estos sprints abre la posibilidad de empezar un sprint perteneciente a una fase posterior sin necesidad de haber terminado todos los sprints de la fase actual. Esto hace que estas metodologías sean adaptativas y flexibles, lo que implica que el proyecto pueda evolucionar de una manera más ágil y menos encorsetada que si se usara una metodología tradicional.

A la hora de decidirse por una metodología u otra, hay que tener en cuenta que las tradicionales otorgan un seguimiento más preciso y previenen la aparición de errores, aunque la planificación suele ser más complicada y son menos adaptativos de cara a las dificultades que pueden surgir durante el desarrollo. Por el contrario, las ágiles son más funcionales al fragmentar el trabajo en sprints, con el objetivo de tener un producto funcional lo antes posible que va mejorando con cada sprint, aunque requieren de mucha implicación por parte del cliente, y un equipo muy cohesionado.

Con esta información, se ha decidido utilizar un modelo de vida en cascada, ya que se definen los requisitos al principio y no cambian de manera significativa ni se añaden nuevos, por lo que es el modelo más correcto a utilizar, ya que cuando se acaba de implementar y verificar cada mecánica, está completamente funcional y sin ningún tipo de error, lo que asegura que no haya conflicto entre las distintas mecánicas cuando se juntan.

2.4. ¿Qué es un documento GDD?

En el desarrollo de videojuegos, un Game Design Document (GDD) es un documento que sirve para organizar las ideas básicas sobre el juego que se está desarrollando [5]. Su importancia radica en que sirve para organizar las ideas y los contenidos que se incluyen, y sirve también como fuente de información y como diario de lo realizado a todos los implicados en el desarrollo del juego. Está estructurado en distintas secciones: resumen del juego, descripción del juego, elementos clave, mecánicas (la parte más importante del documento, ya que explica todo con lo que el jugador va a poder interactuar), diseño narrativo y diseño de personajes. En este TFG se va a integrar dentro del apartado diseño, al estar incluidos en ese apartado todos los puntos importantes de este documento.

2.5. ¿Qué es un test de Bartle?

El test de Bartle es una herramienta utilizada para identificar el tipo de jugador en los videojuegos, particularmente en los juegos multijugador en línea (MMORPGs) [6]. Fue desarrollado por Richard Bartle en 1996 y clasifica a los jugadores en cuatro categorías principales según sus preferencias y comportamientos en el juego:

- Asesinos (Killers): Estos jugadores disfrutan de la competencia y el desafío de enfrentarse a otros jugadores. Les gusta dominar y ganar contra otros en combates y competencias directas.
- Exploradores (Explorers): Los exploradores se sienten atraídos por la exploración y el descubrimiento dentro del juego. Les gusta encontrar secretos, aprender sobre el mundo del juego y experimentar con las mecánicas y la historia.
- Socializadores (Socializers): Para estos jugadores, la interacción social es lo más importante. Disfrutan de chatear, formar grupos y comunidades, y colaborar con otros jugadores.
- Triunfadores (Achievers): Los triunfadores se enfocan en alcanzar metas, completar misiones y acumular recompensas y logros. Les gusta progresar y medir su éxito dentro del juego.

En este TFG, se utilizará para asignar a cada jugador un perfil de jugador, y comprobar en qué grado "Otomatïki" puede servir para identificar perfiles de jugadores, en conjunto con otros dos juegos.

2.6. ¿Qué es un cuestionario MTQ?

Aparte del test de Bartle, que evalúa la experiencia general del usuario, también se va a realizar un Motivational Trait Questionnaire. Este test evaluará 7 rasgos de la personalidad, para averiguar cuáles se adecúan más al juego en base a ciertas acciones que los jugadores realicen o no a lo largo de la prueba. Este test taambién va a servir en qué grado

2.7. Tecnologías utilizadas

2.7.1. Unity

Unity es un motor de videojuegos que se utiliza para crear productos interactivos tanto en 2D como en 3D [7]. Este motor destaca por ser intuitivo, por ofrecer facilidades a la hora de desplegar juegos en diferentes plataformas y en la amplia variedad de herramientas y recursos que posee. Sus scripts se escriben comúnmente en C#, y el scripting se basa en MonoDevelop, que es un entorno de desarrollo diseñado para C# y otros lenguajes .NET [8]. La elección de esta herramienta sobre otras como Godot, tiene su origen en la asignatura de Fundamentos de Videojuegos, que se cursó utilizando esta herramienta, por lo que no hubo un proceso de aprendizaje de cero necesario.

2.7.2. Blender

Blender es una herramienta para modelado, animación, renderizado, composición y creación de gráficos en 3D [9]. Los modelos creados en esta herramienta pueden ser exportados a Unity en muchos formatos diferentes, para usarse como prefabs, elementos del entorno o incluso generar Sprites a partir de ellos. La alta compatibilidad de Blender con Unity, así como la amplia gama de materiales didácticos de la herramienta, es lo que ha inclinado la balanza por la elección de esta herramienta, si bien el trabajo previo que se ha tenido que realizar para aprender a utilizar esta herramienta de modelado ha sido considerable.

2.7.3. Trello

Trello es un software de administración de proyectos basada en tableros que permite organizar las tareas de manera visual [10]. Tanto la facilidad para manejarla como la experiencia previa obtenida en la asignatura de Fundamentos de Videojuegos, han tenido peso a la hora de elegir esta herramienta sobre otra como Notion.

Capítulo 3

Diseño

En este capítulo se exponen los contenidos que suelen ir en un Game Design Document, pero que se han integrado en este punto para que formaran parte del Trabajo de Fin de Grado, y se pudiera encontrar todo el contenido del desarrollo de "Otomatïkï" en el mismo documento.

3.1. Descripción del juego

"Otomatïkï" es un juego perteneciente a los géneros de automatización y simulación en el que se controla a un personaje en vista isométrica que deberá ir buscando recursos para investigar tecnologías, fabricar otros objetos y construir edificios, repitiendo este ciclo de manera incremental. El objetivo a conseguir en la Vertical Slice es fabricar un dron, aunque el árbol de tecnologías está pensado para ser ampliado en caso de seguir con la implementación más allá de la Vertical Slice, por lo que el objetivo del juego cambiaría según este fuera aumentando.

A la hora de darle una clasificación por edades, con toda probabilidad se le asignará una clasificación PEGI 3. El sistema de clasificación PEGI (Pan European Game Information) es un sistema europeo de clasificación de los videojuegos y otro software de entretenimiento [11].

Esta clasificación es la más baja de todo el sistema PEGI, y se asignaría a "Otomatïkï" ya que es un juego apropiado para todas las edades por diferentes motivos.

- Contenido no violento: "Otomatïkï" no contiene violencia gráfica, lenguaje inapropiado ni referencias a sustancias controladas.
- Temática adecuada para niños: "Otomatïkï" se centra en la obtención de recursos para construir edificios y fabricar materiales, lo cuál es adecuado para un público joven.
- Experiencia educativa: "Otomatïkï" puede tener valor educativo al introducir conceptos relacionados con la ingeniería y mejorar la capacidad de gestión de los jugadores.

Aunque "Otomatïkï" tenga esta clasificación, el sistema PEGI considera la idoneidad de edad de un juego, no el nivel de dificultad [11], por lo que la edad ideal dpara empezar a jugar a "Otomatïkï" sería superior a 3 años.

Figura 3.1: Clasificación PEGI 3

3.2. Gameplay y mecánicas del juego

Aunque el estado de Vertical Slice del juego desarrollado no muestra de manera completa todas las mecánicas que tendría un producto acabado, "Otomatïkï" se ha diseñado con la intención de ser un juego completo.

3.2.1. Gameplay

"Otomatïkï" utiliza a al hora de jugar unas físicas simples pero realistas, teniendo un área de exploración delimitado y colisiones con los edificios que se pueden construir. El mundo del juego es completamente plano, por lo que no hay ningún tipo de movimiento más allá del horizontal.

"Otomatïkï" cuenta solo con dos escenas, que son un menú inicial y el mundo del juego, aunque para acceder de una a la otra hay opción de hacerlo con tutorial o sin él, y el estado de la escena del juego cambiará dependiendo de la opción elegida. Esto deja un flujo entre escenas muy sencillo, que se puede observar a continuación:

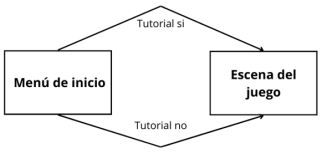


Figura 3.2: Flujo entre las dos escenas de "Otomatïkï"

3.2.1.1. Exploración

El jugador aparece en un territorio inexplorado pero delimitado, que deberá explorar a fin de encontrar todos los recursos disponibles. El control del movimiento se realizará con el click derecho del ratón.

El mapa está conformado por casillas delimitadas de forma hexagonal, formando un cuadrado alrededor de la ubicación en la que aparece el jugador al comienzo del juego. Las casillas exteriores de ese cuadrado simulan de manera eficaz el agua, creando así una isla que delimitará el movimiento del jugador a un espacio reducido.

3.2.1.2. Objetivo

En esta Vertical Slice, el objetivo del jugador será fabricar y tener en el inventario un dron. Para lograr este objetivo, será necesario investigar y fabricar la gran mayoría de los objetos disponibles en el juego, dejando de manera opcional la construcción de edificios.

3.2.1.3. Progresión

La progresión en "Otomatïkï" se da según se van investigando tecnologías en el árbol, lo cual desbloquea materiales para fabricar, los cuáles son en su mayoría indispensables para completar el objetivo de la Vertical Slice. Esta progresión es incremental, y de manera habitual es necesario fabricar el objeto que se acaba de investigar para poder investigar el siguiente.

Esto hace que "Otomatïkï" genere una necesidad constante de materias primas, ya que según avanza el juego para fabricar un objeto hacen falta más recursos básicos. Esta necesidad hace a su vez que sea útil construir minas para obtener los recursos, e incluso hornos y fábricas para automatizar la producción de materiales más allá de los recursos básicos, y poder realizar varias fabricaciones de manera simultánea, acelerando la velocidad a la que se avanza por el árbol de tecnologías, y sobre todo para aprender a utilizar las diferentes mecánicas disponibles a fin de completar, de manera opcional, el último de los logros que se exponen en el siguiente punto.

3.2.1.4. Logros opcionales

En este apartado se encuentran 4 logros opcionales, que servirán como incentivo para los jugadores más complecionistas. Estos 4 logros son:

- Conectar dos minas con un horno usando cintas de transporte.
- Fabricar un dron, que es además el objetivo principal de la Vertical Slice.
- Investigar todas las tecnologías, que comprenden las necesarias para fabricar un dron y la rama de construcción de edificios.
- Tener simultáneamente 10 unidades de cada objeto disponible en el inventario. Este último logro está orientado a las personas que quieren comple-

tar el 100 % del juego, investigando y masterizando todas las mecánicas del juego.

3.2.2. Mecánicas del juego

3.2.2.1. Movimiento

El personaje en "Otomatïkï" se mueve a lo largo y ancho del territorio disponible utlizando el click derecho del ratón. Para ello, se utiliza un componente de Unity llamado NavMeshAgent, del cual se excluyen tanto el agua que rodea la isla donde aparece el jugador (delimitando así el espacio de juego), como los edificios colocados, resultando en una malla que se va actualizando de manera continua teniendo en cuenta los cambios realizados en el entorno por el jugador al colocar edificios o quitarlos.

3.2.2.2. Obtención de recursos

Para obtener los 5 tipos diferentes de materias primas que hay en el mundo de "Otomatïkï", será necesaria encontrar los diferentes depósitos que hay a lo largo del mapa. Una vez se encuentre el jugador situado sobre uno de estos depósitos, mantener presionada la tecla F durante un par de segundos añadirá una unidad del recurso asociado a ese depósito al inventario.

3.2.2.3. Gestión del inventario

El jugador dispone de un inventario de 20 casillas dividido en dos partes. La primera de ellas es una barra en la parte inferior del HUD que puede intercambiar objetos con los menús de los edificios, a fin de recolectar los recursos obtenidos en las minas o los objetos creados en los hornos y fábricas, e introducir recursos manualmente en esos hornos y fábricas a fin de obtener un producto. La segunda de ellas es una parte que está oculta, que se puede ampliar con la tecla E, y que a priori no es accesible desde los menús de los edificios.

Ambas partes del inventario tienen 10 casillas, y pulsando la tecla Tab se pueden intercambiar los objetos entre las dos, lo cual permite acceder a los objetos con el menú de un edificio abierto, sin tener que mover los objetos manualmente entre las casillas.

Todas estas casillas son interactuables, y pulsando click izquierdo se pueden mover los objetos de una casilla a otra. Estas casillas no tienen un límite de objetos que pueden contener, pero no puede haber dos objetos iguales en la misma casilla.

Adicionalmente, se puede seleccionar una de las casillas de la barra del inventario utilizando uno de los números que hay encima del teclado (del 1 al 0), a fin de poder colocar los edificios o cintas de transporte cuando el correspondiente modo esté activado.

3.2.2.4. Investigación de tecnologías

Pulsando la tecla T, se abre el árbol de tecnologías. En este menú, al sostener el cursor sobre uno de los objetos disponibles (que se mostrarán rodeados de amarillo), se abre un desplegable que muestra la cantidad necesaria de recursos para investigar la tecnología en caso de tenerlos disponibles, o un cero en color rojo en caso de que no haya ningún recurso de ese tipo en el inventario. Si en el inventario se encuentra una unidad de cada recurso necesario, al hacer click izquierdo sobre el objeto este se investigará, marcándose en verde y desbloqueando la posibilidad de investigar los inmediatamente posteriores. Cuando una tecnología se investiga, el objeto que lleva asociado se desbloquea para fabricarse tanto en la fábrica como en el menú del inventario.

Este árbol está pensado de manera incremental y tiene tres ramas principales:

- La primera de ellas es la de construcción. Esta es una rama opcional de cara al objetivo pirncipal de la Vertical Slice (ya que no es necesario investigar ninguno de sus componentes para construir el dron), que contiene las tecnologías necesarias para fabricar y construir edificios como las minas, los hornos o las fábricas.
- La segunda rama tiene que ver con aumentar la cantidad de objetos que se pueden fabricar a partir del mineral de hierro y sus derivados.
- La tercera de las ramas funciona igual que la segunda, pero utilizando en este caso mineral de cobre y sus derivados.

Estas dos últimas ramas acaban en el mismo objeto (el dron, objetivo de la Vertical Slice), el cual sólo se podrá investigar si ambas ramas están investigadas completamente.

3.2.2.5. Fabricación de objetos

Al ampliar en inventario con la tecla E (tal y como se ha explicado en el punto anterior), se puede encontrar también un menú de fabricación, donde se pueden fabricar los objetos previamente investigados en el árbol de tecnologías.

En este menú, al sostener el cursor sobre uno de los objetos, se abre un desplegable que muestra la cantidad necesaria de recursos para fabricar ese objeto en caso de tenerlos disponibles, o un cero en color rojo en caso de que no haya ningún recurso de ese tipo en el inventario. Si en el inventario se encuentra una unidad de cada recurso necesario, al hacer click izquierdo sobre el objeto este se añadirá a una cola de fabricación que se mostrará en una esquina de la pantalla siempre y cuando haya al menos un objeto fabricándose. Adicionalmente, se eliminará una unidad de cada uno de los recursos necesarios para ese proceso de fabricación. Al cabo de un tiempo determinado, se añadirá el objeto que se ha fabricado en el inventario.

Por otra parte, también se pueden fabricar objetos en los hornos y en las fábricas, que tienen otro tipo de interfaz.

Los hornos, por un lado, solo admiten para fabricar carbón en la casilla inferior y mineral de hierro, mineral de cobre, o arcilla en la casilla superior. El producto de estas fabricaciones viene determinado por el tipo de recurso básico que se introdujo en la casilla superior. De la misma manera que en el menú de fabricación adjunto al inventario ampliado, se consume una unidad de cada recurso y se genera una unidad del producto.

Las fábricas en cambio son más similares a la fabricación en el inventario. En estos edificios, primero se ha de seleccionar el objeto que se quiere fabricar utilizando el click derecho. Después, la fábrica solo admitirá los materiales que ese objeto previamente seleccionado requiere para su fabricación. De la misma manera que en el menú de fabricación adjunto al inventario ampliado, se consume una unidad de cada recurso y se genera una unidad del producto. Sin embargo, el menú que aparece al seleccionar un objeto para crear en la fábrica también dispone de un botón que sirve para volver hacia atrás si se quiere utilizar la fábrica para crear otro objeto diferente.

3.2.2.6. Edificios

En "Otomatïkï" hay 3 tipos de edificios: la mina, el horno y la fábrica. Al pulsar la tecla C, se activa el modo de construcción. En este modo, mientras haya uno de los tres edificios en la casilla seleccionada de la barra del inventario, se podrá apreciar una sombra del edificio sobre el mapa en el punto donde se halle el cursor. Al hacer click izquierdo, se colocará el edificio en la casilla donde aparece ese layout. Sin embargo, si el edificio en concreto es una mina, solo se podrá colocar sobre un depósito de recursos.

Si se quiere eliminar un edificio, al pulsa la tecla X se abrirá el menú de destrucción, el cual marcara en rojo el edificio sobre el que se posa el cursor, y al hacer click derecho ese edificio se eliminará y se añadirá al inventario.

Para acceder al menú de cualquiera de estos edificios, bastará con hacer click derecho sobre él para que el personaje del jugador se acerque mientras no esté activo el modo de construcción ni el de destrucción.

Si bien en el punto anterior se ha explicado el funcionamiento tanto del horno como de la fábrica, falta explicar el funcionamiento de la mina. Este edificio se colocará, como se ha explicado antes, sobre un depósito, lo que hará que se vaya extrayendo el recurso asociado a dicho depósito de manera continua, hasta que la capacidad de la mina alcance su máximo (10 unidades del recurso). El máximo de producción en el caso de la mina y el horno es de 20 unidades del recurso en cuestión.

3.2.2.7. Cintas de transporte

Estas cintas son el componente principal de la logística en "Otomatïkï". Si la casilla seleccionada de la barra del inventario contiene este objeto, y está activado el modo de construcción (se ha pulsado la tecla C), al pulsar primero un edificio y después otro, se generará una cinta que los conecte utilizando un algoritmo

A*. La cinta tendrá como origen del transporte el primer edificio clickado, y el segundo como destino.

Sin embargo, la generación de las cintas puede no suceder debido a distintos factores:

- No hay suficientes cintas: este error viene dado por la falta de cintas entre los dos edificios. Será necesario una unidad de cinta para cada edificio y para cada casilla que hay entre ellos.
- Una cinta no puede acabar en una mina: si el edificio de destino de una cinta al colocarla es una mina, esa cinta no se colocará.
- Un edificio no puede tener dos salidas: cada edificio solo tiene la capacidad de tener una cinta de salida, por lo que si se quiere colocar una segunda, esta última no aparecerá.

Todos estos mensajes de error se muestran de manera visual en el HUD.

3.3. Interfaz de usuario

”Otomatïkï” cuenta con una interfaz que se compone de un HUD permanente, menús desplegables y avisos o paneles que aparecen dependiendo de las acciones realizadas. Tanto la pantalla del juego como la interfaz se han preparado para una resolución de 16:9 sin posibilidad de escalarlo, al menos en la versión de Vertical Slice en la que se encuentra ”Otomatïkï”.

A continuación se muestran esquemas de la interfaz en todas las formas en que esta se puede mostrar:

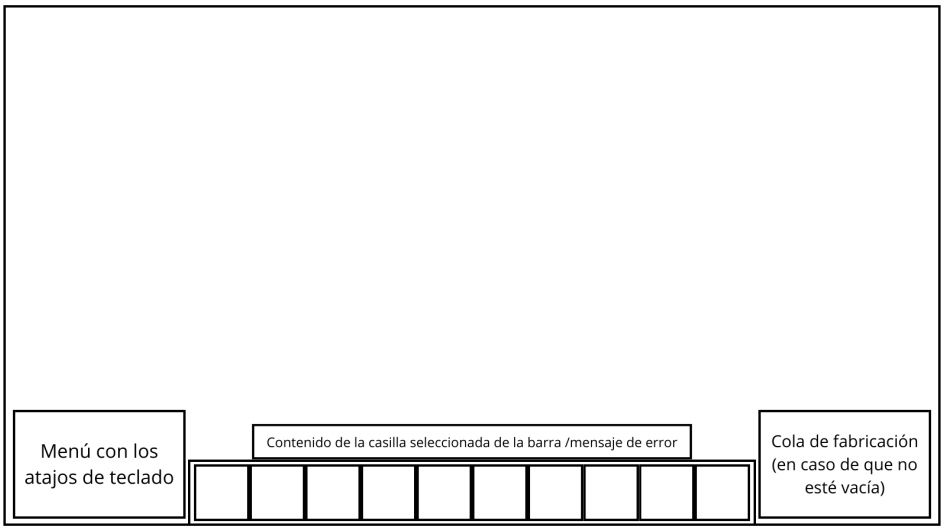


Figura 3.3: Interfaz de usuario sin ningún menú abierto

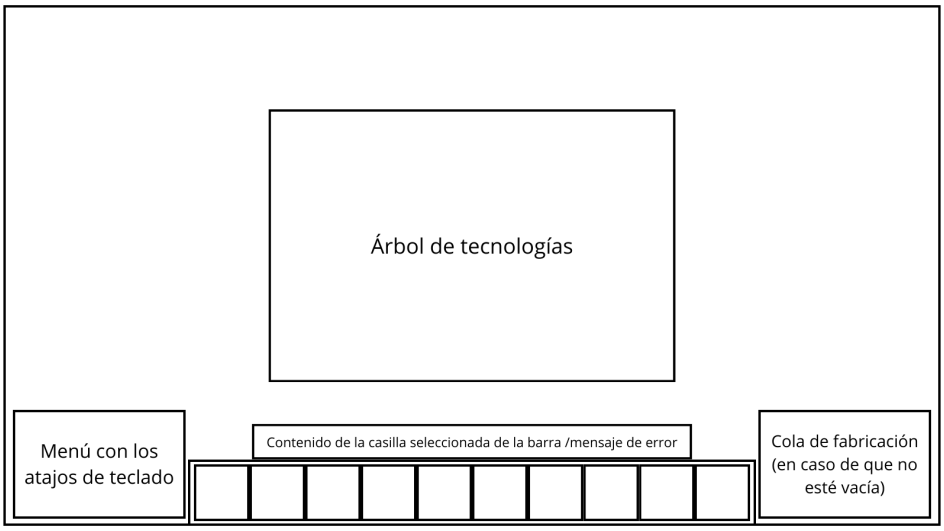


Figura 3.4: Interfaz de usuario con el árbol de tecnologías

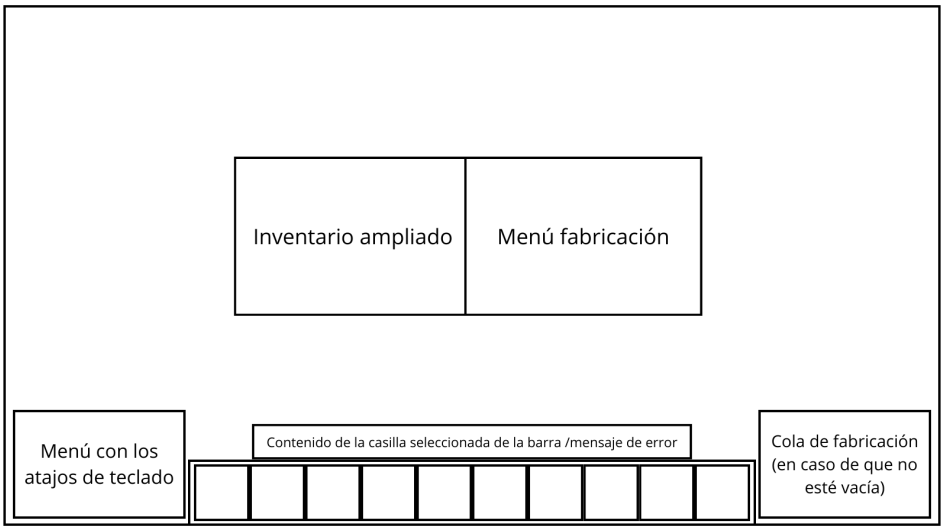


Figura 3.5: Interfaz de usuario con el inventario ampliado y el menú de fabricación

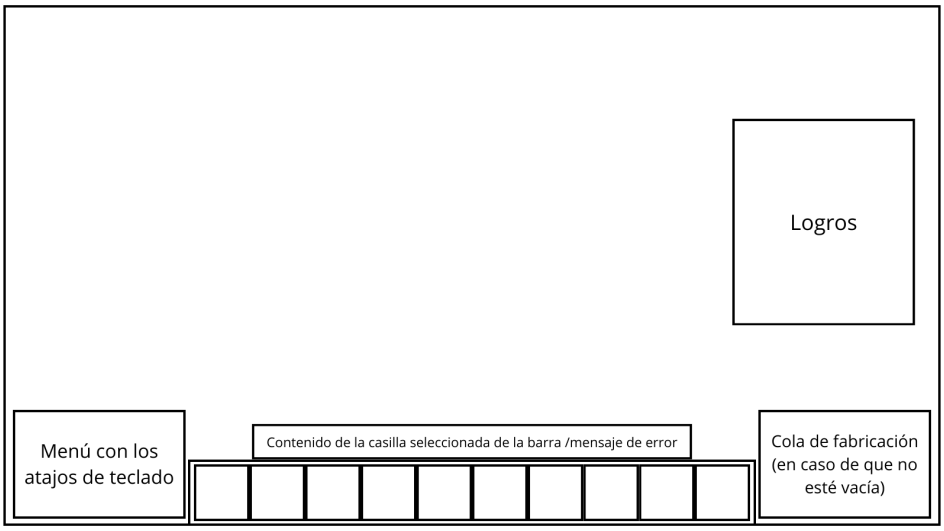


Figura 3.6: Interfaz de usuario con el panel de logros

En las figuras 3.3, 3.4, 3.5 y 3.6, se puede apreciar que el HUD (Heads Up Display), que es la parte de la interfaz que permanece siempre en la pantalla durante el juego, está compuesta por la barra del inventario, el menú de ayuda con los atajos del teclado y el panel que muestra el nombre del objeto contenido en la casilla seleccionada de la barra del inventario, en caso de que haya alguno, y que también muestra los errores que se pueden producir al colocar las cintas entre edificios.

Por otro lado, tanto el menú de los logros, como el inventario ampliado junto con el menú de fabricación, el panel que muestra la cola de fabricación y el árbol de tecnologías, son elementos que van a ir apareciendo y desapareciendo de la interfaz según se cumplan o no ciertas condiciones.

De la misma manera, y como se podrá ver a continuación, los menús de los edificios se abrirán solo si se cumplen las condiciones que generan su apertura. Además, abajo se muestran la apariencia que tienen todos estos paneles y menús en la Vertical Slice de ”Otomatïkï”.

3.4. Arte y vídeo

3.4.1. Estética

”Otomatïkï” tiene una estética low poly para el personaje, que encaja muy bien con el mundo formado por hexágonos. Esta estética es fácil de modelar debido a la baja cantidad de vétices que maneja, y en un entorno que acompañe, como el que se encuentra en ”Otomatïkï”, resulta atractivo a la vista.

3.4.2. Arte 2D

El arte 2D de "Otomatïkï" no sigue la estética general low poly del juego, pero al ser imágenes que tendrán un tamaño pequeño en el juego y que pertenecieran completamente a los elementos de la interfaz, no tiene tanto impacto en la escena que haya un conteo de vértices superior.

Para mostrar los iconos de cada objeto del juego, primero se modeló en Blender en 3D cada uno de ellos, y tras haber colocado unos puntos de luz y una cámara, se exporta una animación de un único fotograma, creando así un sprite en 2D. Como ejemplo, se va a usar el lingote de hierro que se puede ver debajo, primero como modelo en Blender y después tras exportarlo:

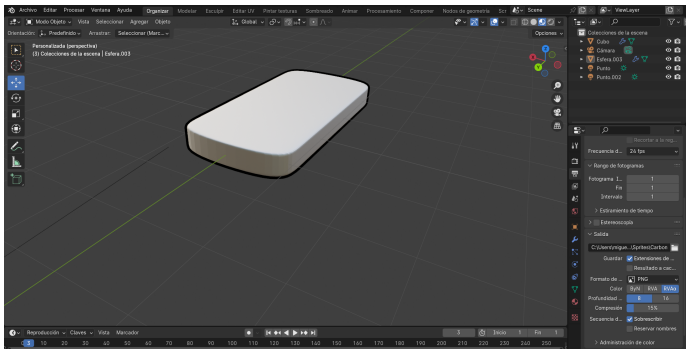
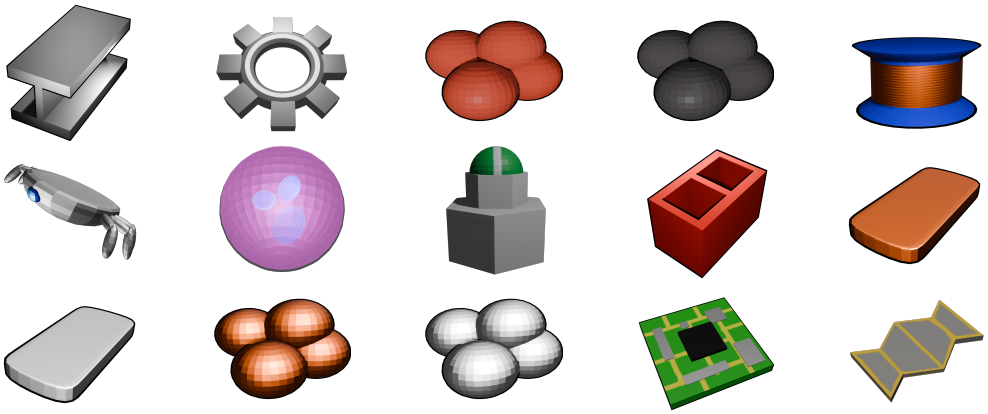


Figura 3.7: Modelado del lingote de hierro en Blender

Figura 3.8: Imagen capturada que aparecerá en el juego

Este proceso se ha llevado a cabo con todos los sprites de los objetos que aparecen en el juego, modelándolos desde cero a partir de polígonos básicos en 3D, y se han utilizado en la interfaz en un tamaño reducido. Estos son todos los sprites que se han creado de esta manera:



3.4.3. Arte y animación 3D

El arte 3D de "Otomatïkï" se divide entre el jugador y el entorno del mapa y los edificios.

3.4.3.1. Modelo del jugador

El personaje controlado por el jugador está hecho enteramente en Blender, modelando desde cero utilizado y extruyendo polígonos sencillos para construir el cuerpo, mientras que para la cabeza se utilizó el modelo de la arandela que se había creado para el objeto previamente. Para la inspiración del personaje, según avanzaba el modelado surgió la similitud con el pokemon Meltan, por lo que se coloreó de manera similar.

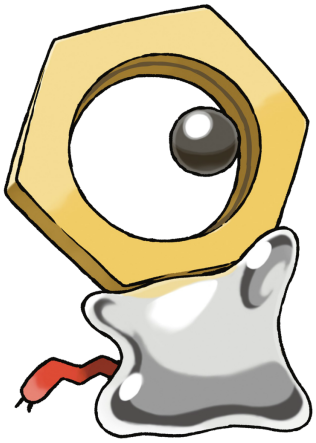


Figura 3.9: Pokemon que sirvió de inspiración para el diseño del personaje

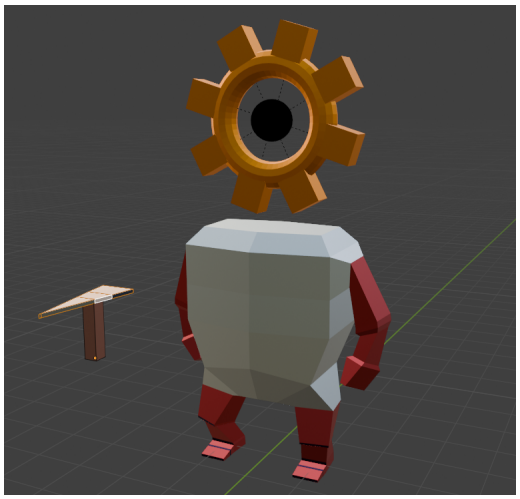


Figura 3.10: Modelado final del personaje jugable con la herramienta de minado

3.4.3.2. Modelado del entorno

El entorno de "Otomatïkï" no se ha modelado en Blender, si no que se genera un mundo conformado por una matriz de hexágonos mediante un script de Unity y se le asigna un material dependiendo de la función que tenga ese prisma hexagonal en concreto. Se asignará un material básico a todas las casillas del suelo, menos a aquellas que formen parte de un depósito de recursos, a las que se asignará el material correspondiente.

Así mismo, los edificios como las minas, los hornos y las fábricas se modelan de la misma manera que cualquier casilla del suelo, siendo asignado a ellos el material que corresponde a cada edificio, a fin de distinguirlos.

Por último, las cintas de transporte se modelan utilizando fragmentos de los prismas previamente desarrollados para el suelo, concatenándolos entre si para seguir el camino más corto.

3.4.3.3. Animaciones del personaje

Las animaciones del personaje se realizaron añadiendo un esqueleto al modelo que se realizó en Blender. Este esqueleto tiene tres animaciones diferentes:

La primera de ellas es la animación Idle, que simplemente baja y sube el tronco del cuerpo del personaje para representar una pose relajada mientras el personaje no se está moviendo. Aunque en las siguiente fotografías no sea muy apreciable, es distinguible en la Vertical Slice.

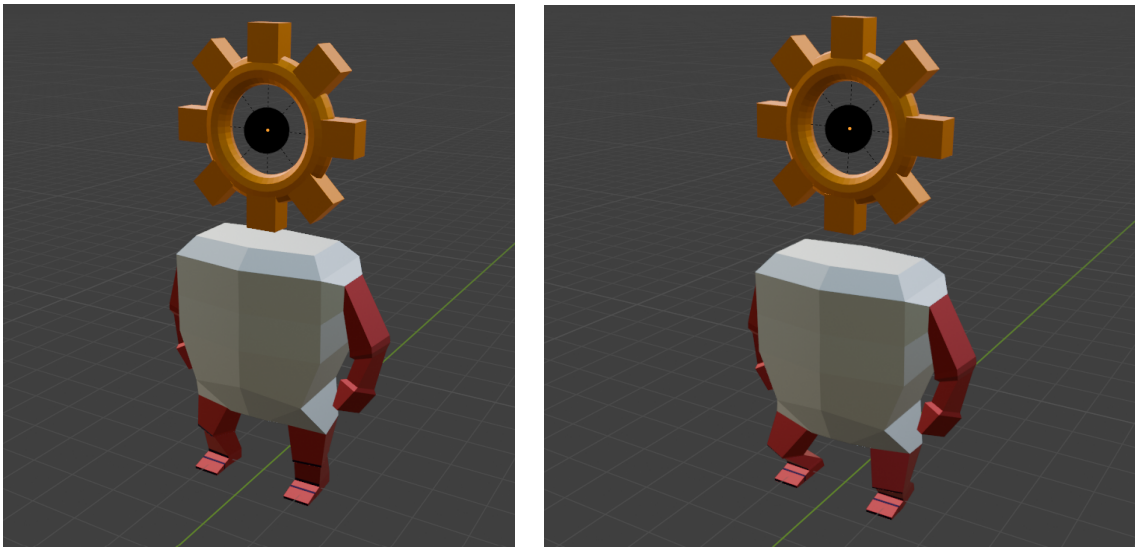


Figura 3.11: Animación Idle, que se reproduce cuando el personaje está parado

La segunda de estas animaciones es la animación Run, que se reproduce cuando el personaje se está moviendo. Esta animación consta, al igual que la anterior, de dos keyframes entre los que el personaje va moviendo los brazos y las piernas para dar una sensación de movimiento exagerada pero efectiva.

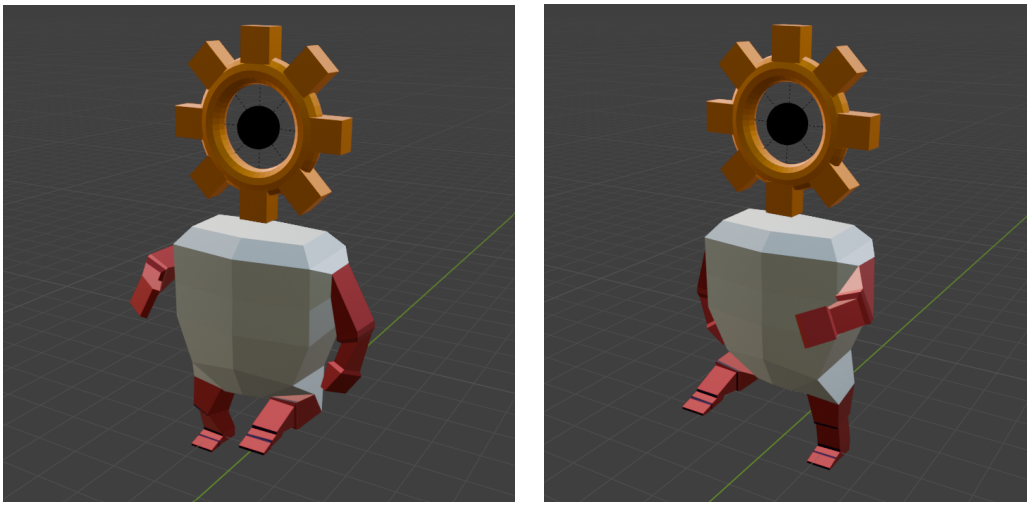


Figura 3.12: Animación Run, que se reproduce cuando el personaje está en movimiento

La última animación que utiliza este esqueleto es la animación Minar. Esta se ejecuta solo mientras el personaje está extrayendo un recurso de un depósito, por lo que es la menos utilizada. Sin embargo, tiene asociado un pico, que se agrega como un GameObject dependiente del brazo en unity para dar una sensación de movimiento realista. Sin embargo, por la parte de Blender, el esqueleto no tiene en cuenta la existencia de este pico ,resultando la animación así:

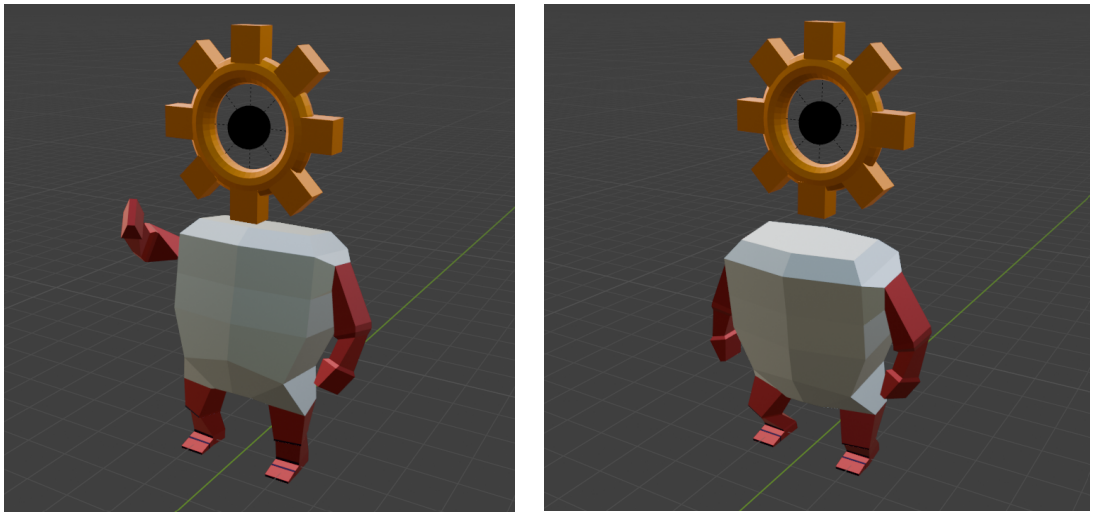


Figura 3.13: Animación Minar, que se reproduce cuando el personaje está extrayendo recursos

De manera adicional a estas tres animaciones que se han expuesto, hay una cuarta que se está ejecutando continuamente y de manera independiente. Est animación hace que la cabeza del personaje rote en torno a su centro, aportando dinamismo a la escena independientemente de la acción que se esté realizando.

3.5. Diseño de niveles

"Otomatïkï" solo cuenta con un nivel jugable, y aunque se ampliara el juego más allá de la Vertical Slice no se crearían más niveles, ya que los juegos de este género solo tienen un nivel por norma general, y simplemente se ampliaría el nivel existente para adecuarlo al crecimiento del juego.

3.5.1. Diseño molecular del nivel

Para el diseño molecular del nivel, si bien no hay ningún elemento por encima del suelo, si que hay que colocar los depósitos en determiandas casillas. Se tomó la decisión de hacer que cada depósito estuviera compuesto por 7 casillas, formando un hexágono más grande que escaja con la estética elegida del juego, basada en polígonos.

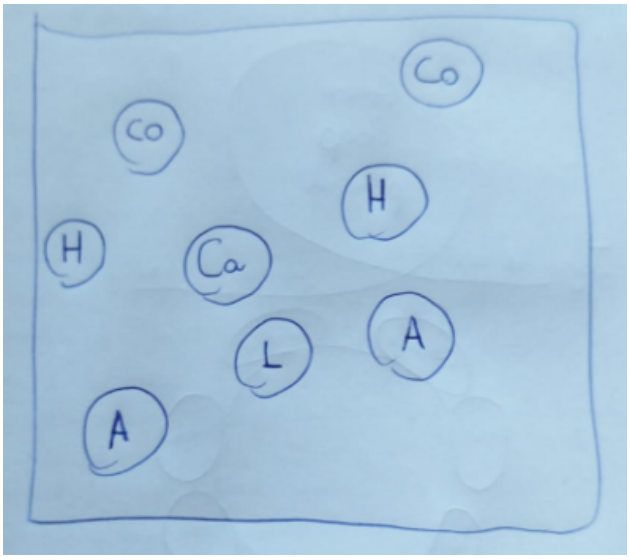


Figura 3.14: Diseño molecular planficado

La figura 3.14 muestra el esbozo de lo que acabaría siendo el mapa del mundo de "Otomatïki". Los círculos señalan la ubicación de los depósitos de materias primas en el mapa, y la leyenda de dicha figura es la siguiente:

- Co: Depósito de mineral de cobre
- Ca: Depósito de carbón
- L: Depósito de luz
- H: Depósito de mineral de hierro
- A: Depósito de arcilla

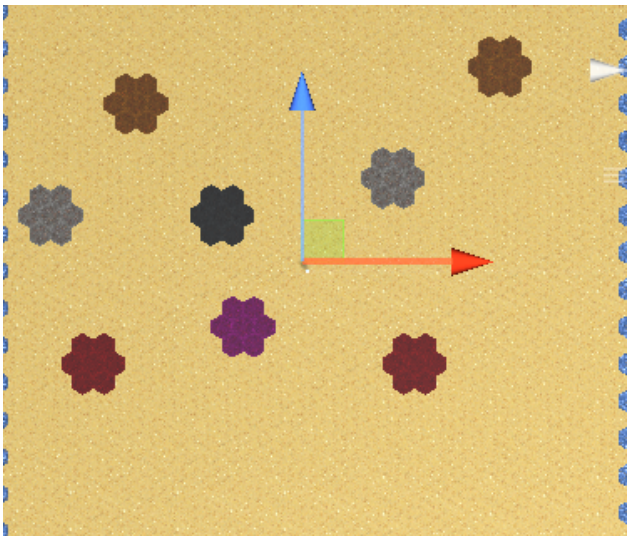


Figura 3.15: Mapa final de la Vertical Slice

En la figura 3.15 se puede ver cómo quedó el mapa tras implementarlo. Se puede notar que se dejó un espacio en la esquina inferior derecha, que serviría para ampliar en caso de que hubiera más recursos, ya que en la Vertical Slice el objetivo fue que tanto la luz como el carbón no fueran tan abundantes como los otros recursos.

3.5.2. Game Loop

El game loop de "Otomatïkï" empieza cuando se selecciona en el menú de inicio si se quiere empezar con tutorial, y acaba cuando se construye el dron (en esta fase de Vertical Slice). El game loop tiene 5 actividades principales que

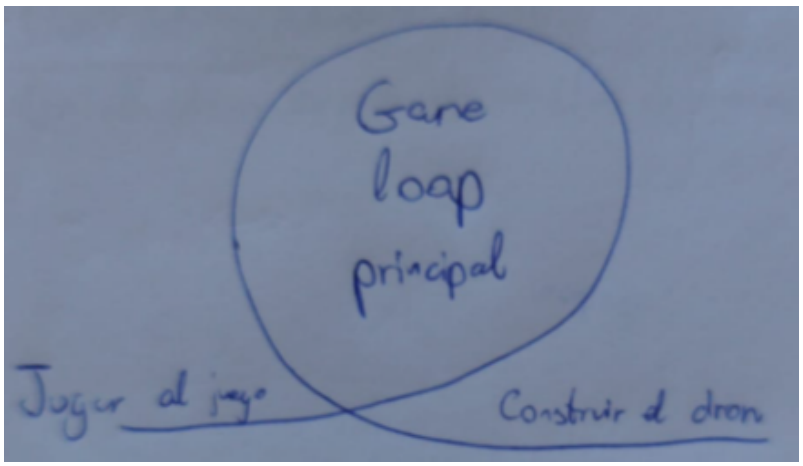


Figura 3.16: Entrada y salida del game loop principal

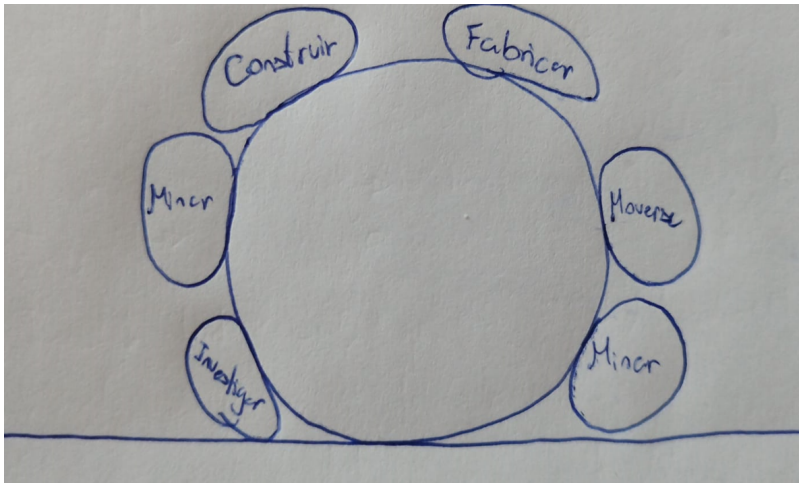


Figura 3.17: Mecánicas recurrentes en el game loop

En la figura 3.16, se puede apreciar que el game loop se ejecuta hasta que se cumple la condición de salida. Por otro lado, en la figura 3.17 se aprecia que ese bucle está compuesto de bucles más pequeños, cada uno de ellos correspondiente a una mecánica. Podría fijarse el paso del bucle a la siguiente iteración

del mismo cada vez que se investiga una tecnología.

En la figura que viene a continuación, se explica cómo pueden interactuar las diferentes mecánicas en el cada iteración del game loop, ya que si bien el investigar una tecnología marca el final de cada una de las iteraciones, las mecánicas no tienen por qué interactuar en el mismo orden cada vez que se quiere investigar una tecnología.

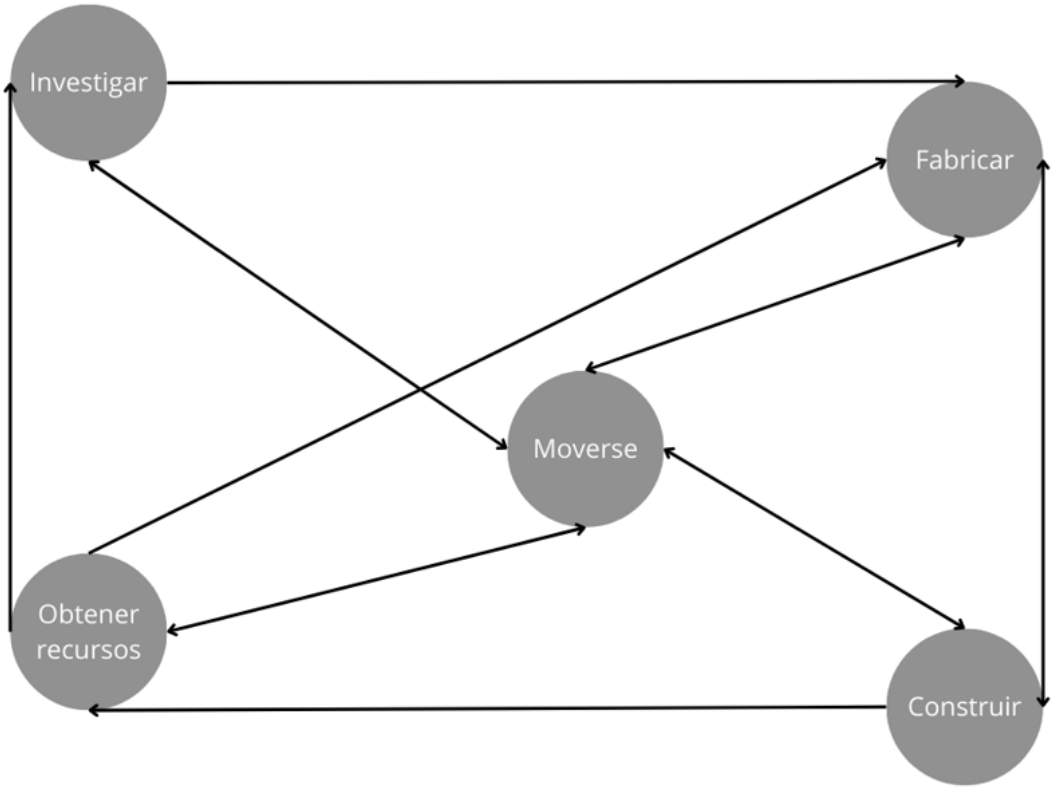


Figura 3.18: Interacción de las diferentes mecánicas en cada bucle del game loop

Capítulo 4

Desarrollo

A la hora de desarrollar todo lo diseñado previamente, hacía falta preguntarse cómo hacer que se comunicaran todos los diferentes objetos que llevaban scripts asociados que debían intercambiar información, ya fuera para items, la generación del terreno, colocar edificios, o un largo etcetera de situaciones. Para ello, y como se iba a tener que acceder a esa clase muchas veces, la mayoría, por no decir todos, los atributos de acceso global se guardaron en la clase del inventario. Además, el propio GameObject del inventario también tenía asignado el script ListaObjetos, que contiene una lista disponible con la información de todos los objetos disponibles del juego, lo cual hace que pasando como atributo ese GameObject a cualquier clase, se tenga la información necesaria para trabajar con el estado del mapa y del inventario del jugador en ese momento.

En adelante se expondrán todas las clases creadas, explicando sus atributos y funciones, el proceso de desarrollo de cada una, y los comentarios específicos que cada una de ellas requiera. Para hacer más comprensible la organización de las clases, se dividirán las clases por grupos según los objetos de la escena a los que están asociados. Cabe destacar que todas las clases que aparecen en las subsecciones que empiezan por GameObject derivan de MonoBehaviour (clase de Unity que permite a los scripts asociarse con GameObjects) a menos que se especifique lo contrario.

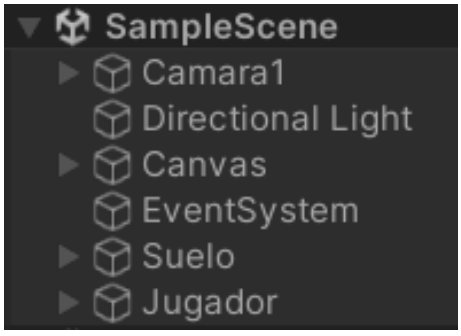


Figura 4.1: Objetos de la escena principal

4.1. Clases auxiliares

En esta primera sección, se explicarán las clases auxiliares, que no están asociados a un objeto en concreto de manera específica, si no que se utilizan de manera complementaria a las demás clases

4.1.1. Clase Objeto

Esta clase se utiliza para definir los parámetros de todos los objetos cuando se van a meter en el inventario o cuando se asocian a un depósito. Deriva de la clase MonoBehaviour, lo cual permite que se pueda asociar a un GameObject de Unity.

Nombre	Modificador	Tipo	Descripcion
ID	public	int	Identificador único del Objeto
nombre	public	string	Nombre único del Objeto
descripcion	public	string	Grupo de objetos al que pertenece el Objeto
icono	public	Sprite	Imagen del Objeto

Figura 4.2: Atributos de la clase Objeto

4.1.2. Clase Object

Esta clase se utiliza para definir los parámetros de todos los objetos cuando se añaden a la lista de todos los objetos de "Otomatïkï". La diferencia con la clase Objeto, es que al heredar esta última de MonoBehaviour, no se puede instanciar para ingresarlo en la lista, por lo que esta clase solo se utiliza para crear la lista y obtener la información necesaria a la hora de instanciar un Objeto.

Nombre	Modificador	Tipo	Descripcion
ID	public	int	Identificador único del Object
nombre	public	string	Nombre único del Object
descripcion	public	string	Grupo de objetos al que pertenece el Object
icono	public	Sprite	Imagen del Object

Figura 4.3: Atributos de la clase Object

4.2. UI

En esta sección se hablará sobre todas las clases que estaban ubicadas dentro del objeto Canvas, el cual contiene todos los elementos pertenecientes a la interfaz de usuario, tanto el HUD, como los menús desplegables y los contenidos que no se muestran siempre en pantalla.

Figura 4.4: Objetos del canvas de la escena

4.2.1. Clase Slot

Esta clase deriva de la clase MonoBehaviour, lo cual permite que se pueda asociar a un GameObject de Unity. El script Slot va asociado a cada casilla del inventario, así como a cada una de las casillas que tengan los edificios. Aunque sí que se asigna a otros objetos, se ha añadido al principio de esta sección ya que aparece en muchas zonas de la interfaz de usuario. Como se puede ver debajo en la figura 4.5, el GameObject Slot promedio (casilla) tiene un componente Image que ejerce como borde, un primer componente Panel que ejerce como fondo de la casilla, un segundo componente Panel que actúa imagen del objeto contenido, y un texto que indica la cantidad de unidades que contiene la casilla.

Figura 4.5: Estructura del slot

Nombre	Modificador	Tipo	Descripcion
movil	public	Slotm	GameObject de la casilla movil

ID	public	int	Identificador único del objeto contenido en la casilla
nombre	public	string	Nombre único del objeto contenido en la casilla
descripcion	public	string	Grupo de objetos al que pertenece el objeto contenido en la casilla
icono	public	Sprite	Imagen del objeto contenido en la casilla
vacio	public	bool	Variable que comprueba si la casilla está vacía
n	public	int	Cantidad de unidades del objeto contenido en la casilla
investigado	public	int	Variable que se utiliza en las casillas del árbol de tecnologías para alternar entre no disponible, disponible e investigada
doble	public	int	Variable que se aplica a las casillas del árbol de tecnologías que requieren dos tecnologías para ser investigadas
boton	public	GameObject	GameObject del botón interactuable asociado a la casilla
completado	public	bool	Variable que se aplica a las casillas del árbol de tecnologías que requieren dos tecnologías para ser investigadas

Figura 4.6: Atributos de la clase Slot

Nombre	Modificador	Parámetros	Tipo	Descripcion
FixedUpdate			void	Actualiza la imagen de la casilla. En caso de que pertenezca a una fábrica, en base al objeto elegido, y en caso de que pertenezca al árbol de tecnologías, actualiza el color del recuadro en base al valor de investigado
Clickado	public		void	Intercambia el contenido del movil con el de la casilla

Figura 4.7: Funciones de la clase Slot

4.2.2. Clase Desplegable

Esta clase deriva de la clase MonoBehaviour, lo cual permite que se pueda asociar a un GameObject de Unity. El script Desplegable va asociado a todos los botones que hay sobre las casillas del menú de fabricación y el árbol de tecnologías, por lo que se ha incluido al principio de este apartado al estar presente en muchos puntos diferentes de la UI.

Nombre	Modificador	Tipo	Descripcion
ID	public	int	Identificador único del objeto que se fabrica
ID1	public	int	Identificador único del primer objeto necesario para la fabricación
ID2	public	int	Identificador único del segundo objeto necesario para la fabricación
inventario	public	GameObject	GameObject del inventario
objeto	private	Objeto	Objeto que se fabrica
o1	private	Slot	Casilla del inventario que contiene el objeto con ID1

o2	private	Slot	Casilla del inventario que contiene el objeto con ID2

Figura 4.8: Atributos de la clase Desplegable

Nombre	**Modificador**	**Parámetros**	**Tipo**	**Descripcion**
Start	private		void	Inicializa objeto, o1 y o2
FixedUpdate	private		void	Asigna objeto si no lo está ya, y asigna o1 y o2 a las casillas del inventario que contienen objetos con identificador igual a ID1 e ID2 respectivamente. También colorea las casillas del panel hijo desplegable en caso de que exista o no ese objeto en el inventario
enviar	public		void	Envía objeto a Fabricación, y quita una unidad al n de o1 y o2 en el inventario
desplegar	public		void	Activa el panel visual hijo del botón al que está asociado Desplegable
replegar	public		void	Desactiva el panel visual hijo del botón al que está asociado Desplegable
OnDisable	private		void	Desactiva el panel visual hijo del botón al que está asociado Desplegable cuando este botón se desactiva

Figura 4.9: Funciones de la clase Desplegable

4.2.3. GameObject Inventario

Este objeto, como se ha explicado anteriormente, se pasa como atributo a muchas de las clases de "Otomatïkï", a fin de acceder a los valores que se guardan en él. Contiene todos los objetos de menús desplegables a los que se accede desde otras clases, cada uno con su script asociado.

Figura 4.10: Objetos del inventario

4.2.3.1. Clase Inventario

Esta es probablemente la clase más importante entre todas las que se pueden encontrar en "Otomatïkï". Está asociada al GameObject Inventario y, tal y como se ha explicado en el párrafo de introducción de este capítulo, esta clase contiene los atributos globales necesarias para que todo el juego funcione.

La barra del inventario, que se puede ver debajo, comprende los GameObjects RawImage, Slots y los dos Text(Legacy) vistos en la figura 4.10.

Figura 4.11: Barra del inventario

Nombre	Modificador	Tipo	Descripcion
inventario	public	GameObject [ ]	Array de los objetos de casillas del inventario

slots	public	GameObject	Objeto padre de las casillas de la barra del inventario
slots1	public	GameObject	Objeto padre de las casillas del inventario ampliado
creado	private	bool	Variable que comprueba que el array inventario se haya creado
construccion	public	bool	Variable que activa el modo de construccion
destruccion	public	bool	Variable que activa el modo de destruccion
posI	public	GameObject	Variable que almacena la posición del edificio inicial de una cinta que se quiere colocar en la matriz de casillas
posF	public	GameObject	Variable que almacena la posición del edificio final de una cinta que se quiere colocar en la matriz de casillas
items	public	List<int>	Lista de los ID de los objetos investigados
codigo	public	int	Variable a la que se asigna un código de error en caso de ser necesario, o 0 en caso contrario
seleccionado	public	Slot	Casilla seleccionada de la barra del inventario

Figura 4.12: Atributos de la clase Inventario

Nombre	Modificador	Parámetros	Tipo	Descripcion
Update			void	Asigna el valor a uno de los logros y procesa el código de error para mostrar por la interfaz el mensaje que sea necesario
crear_inv	public		void	Si el inventario no está creado, lo crea utilizando los objetos hijos de slots y slots1
investigado	public	int id	void	Añade el int id a la lista items
anadir	public	Objeto objeto, int n	void	Añade n unidades del objeto a la primera casilla que contenga ese mismo objeto, o a una que esté vacía
quitar	public	int ID, int n	void	Elimina n unidades de la primera casilla que contenga un objeto con identificador igual a ID
getSlot	public	int ID	Slot	Devuelve la primera casilla que contiene un objeto con identificador igual a ID o null en caso de que no haya ninguno
seleccionar	public	int i	void	Asigna seleccionado al contenido del array inventario en la posición i-1, o si i es igual a 0, se asigna al contenido del array inventario en la posición 9
actualizar	public		void	Actualiza el contenido de los iconos de las casillas de todo el inventario. Se llama cada vez que se modifica algo del inventario

invertir	public		void	Cambia el contenido de los 10 primeros elementos de inventario por el de los 10 últimos y viceversa

Figura 4.13: Funciones de la clase Inventario

4.2.3.2. Clase ListaObjetos

Esta clase, que también va a asociada al GameObject inventario, tiene la finalidad de que todas las clases puedan acceder al conjunto total de objetos que tiene "Otomatïkï".

Nombre	Modificador	Tipo	Descripcion
lista	public	List<Object>	Lista de todos los objetos de "Otomatïkï"
mHierro	public	Sprite	Imagen del mineral de hierro
mCobre	public	Sprite	Imagen del mineral de cobre
piedra	public	Sprite	Imagen de la arcilla
hierro	public	Sprite	Imagen del lingote de hierro
cobre	public	Sprite	Imagen del lingote de cobre
ladrillo	public	Sprite	Imagen del ladrillo
mina	public	Sprite	Imagen de la mina
fabrica	public	Sprite	Imagen de la fábrica
horno	public	Sprite	Imagen del horno
arandela	public	Sprite	Imagen del engranaje
bobina	public	Sprite	Imagen del cable
luz	public	Sprite	Imagen de la luz
cinta	public	Sprite	Imagen de la cinta de transporte
chip	public	Sprite	Imagen de la placa base
carbon	public	Sprite	Imagen del carbón
acero	public	Sprite	Imagen del acero
parte	public	Sprite	Imagen del dron

Figura 4.14: Atributos de la clase Inventario

Nombre	Modificador	Parámetros	Tipo	Descripcion
Lista	public		List<Object>	Crea la lista y la devuelve como salida

Figura 4.15: Función de la clase ListaObjetos

4.2.3.3. Clase Fabricacion

Esta clase sirve para fabricar objetos en el menú de fabricación asociado al inventario ampliado (tercer hijo del Game). Esta clase almacena los objetos que se seleccionan para fabricar, y los añade al inventario tras esperar un determinado tiempo en el orden en que se seleccionaron

Nombre	Modificador	Tipo	Descripcion
cola	public	List<Tuple> Tuple<Objeto,int>	Lista de todos los objetos que hay en la cola de fabricación y el número de ellos a fabricar
panel	public	GameObject	GameObject que actúa de interfaz visual para la cola de fabricación
barraP	public	GameObject	GameObject de la barra de progresión de fabricación
slider	private	Slider	Componente Slider de barraP
progreso	private	float	Progreso de la barra del slider

Figura 4.16: Atributos de la clase Fabricacion

Nombre	Modificador	Parámetros	Tipo	Descripcion
Start			void	Asigna slider e inicializa la cola
Update			void	Invoca o cancela los métodos de fabricación y barra según la cola o no tiene elementos, y actualiza las casillas hijas del panel
recibir	public	Objeto objeto	void	Recibe un objeto para fabricar y lo añade a la cola

fabricacion	private		void	Añade el primer objeto de la cola al inventario, y luego disminuye el contador de la tupla, o elimina el objeto si el contador es 1.
barra	private		void	Iguala el porcentaje del slider con un número proporcional a progreso, y aumenta este último

Figura 4.17: Funciones de la clase Fabricacion

4.2.4. GameObject Menu ampliado

Este GameObject es el que contiene tanto el inventario ampliado como el menú de fabricación. No tiene ningún script asociado, ya que todo el control necesario del inventario ampliado se realiza desde el componente Inventario de su objeto padre (GameObject Inventario), y el control necesario del menú de fabricación se realiza en el componente Fabricacion del GameObject Inventario. Sin embargo, esta es su estructura interna, ya que ambos tienen contenedores de casillas (clase Slot), y el menú de fabricación tiene también botones desplegables (clase Desplegable).

Figura 4.18: Objetos del menú ampliado

Figura 4.19: Inventario ampliado y menú de fabricación

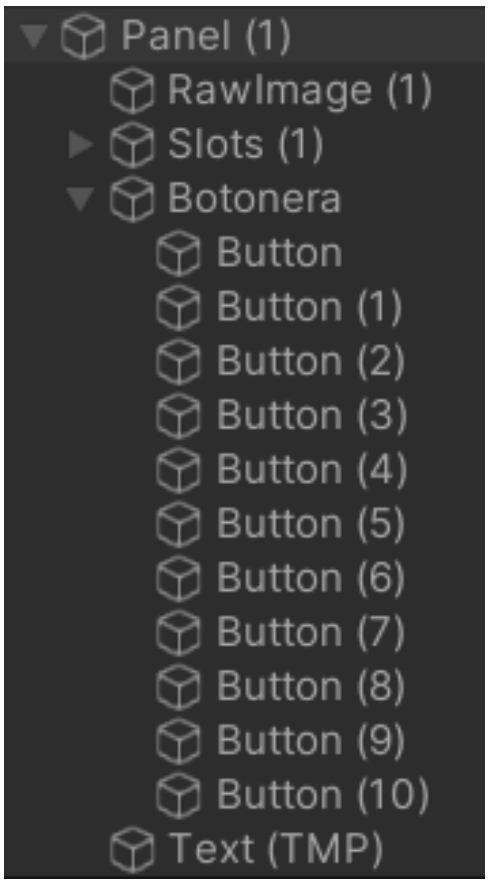


Figura 4.20: GameObjects del inventario ampliado

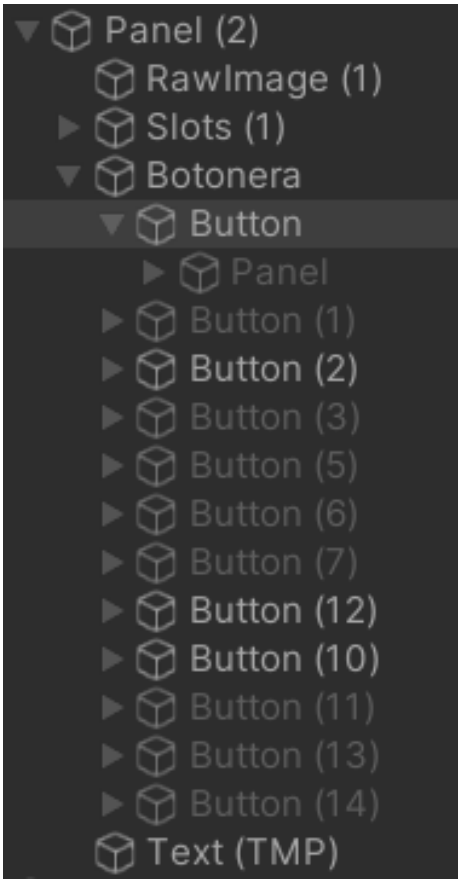


Figura 4.21: GameObjects del menú de fabricación

La figura 4.18 muestra como se dividen los GameObjects hijos de Menu ampliado, correspondiendo Panel con el fondo, Panel (1) siendo el inventario ampliado, y Panel(2) siendo el menú de fabricación. En la figura 4.19 se puede observar cómo quedan estos elementos colocados entre sí.

En las figuras 4.20 y 4.21 se muestra la distribucion de GameObjects de ambos paneles, siendo muy similares las dos. Sin embargo, y como se ha especificado anteriormente, la principal diferencia es que los botones del menú de fabricación tienen asociado el Script Desplegable, y un objeto hijo que hace las veces de menú desplegable. Este desplegable se puede apreciar también en la figura 4.19, a la derecha del icono del lingote de hierro.

La otra diferencia entre estos dos Panel, es que los botones que activan los desplegables al mantener el cursor por encima de ellos, solo se activan cuando la tecnología que desbloquea los objetos de la casilla que tienen asociada está investigada. Esto hace que sólo se puedan ver los desplegables de los objetos disponibles para fabricar.

4.2.5. GameObject Tecnologias

Este GameObject contiene el árbol de tecnologías y todo lo relativo a él. Este objeto no tiene un script asociado, pero sin embargo contiene una serie de casillas, que a su vez tienen asociado el script Investigar, que desbloquea tecnologías si se cumplen ciertas condiciones.

La función principal del árbol de tecnologías es la de desbloquear la capacidad de fabricar ciertos objetos, y para ello tiene que ser progresivo e incremental, lo que facilita la curva de aprendizaje, a la vez que alarga de manera amena la duración del Game Loop general.

Este GameObject se ha estructurado separando, al igual que los expuestos en la anterior subsección, las casillas de los botones, lo que facilita interactuar con estos últimos. A su vez, se han dividido en sub-árboles para facilitar su ordenación, tal y como se puede ver en la figura 4.22.

Sin embargo, este menú en concreto tiene unos palos que conectan entre sí las casillas del árbol con las inmediatas, estableciendo una relación entre ellas. Como se puede ver en la figura 4.23, cada GameObject Slot (casilla), tiene a su vez un GameObject hijo que guarda los palos que salen de esa casilla.

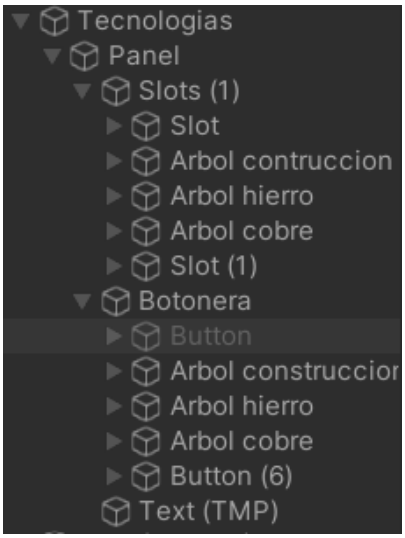


Figura 4.22: GameObjects del árbol de tecnologías

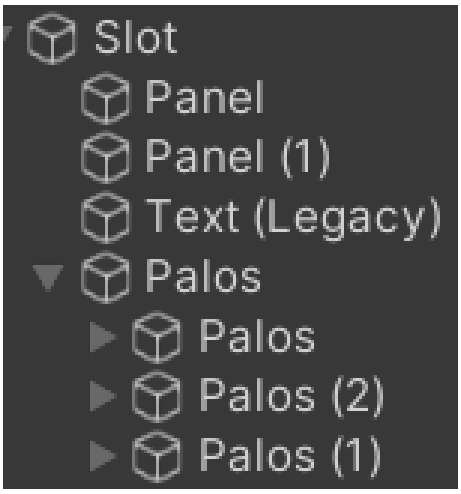


Figura 4.23: GameObjects de cada casilla del árbol de tecnologías

Como se puede apreciar en la figura 4.24, hay tres estados diferentes para las casillas del árbol: no disponible, con el recuadro blanco; disponible, con el recuadro amarillo; e investigada, con el recuadro verde. La condición para que esté disponible se gestiona con la clase que se verá a continuación.

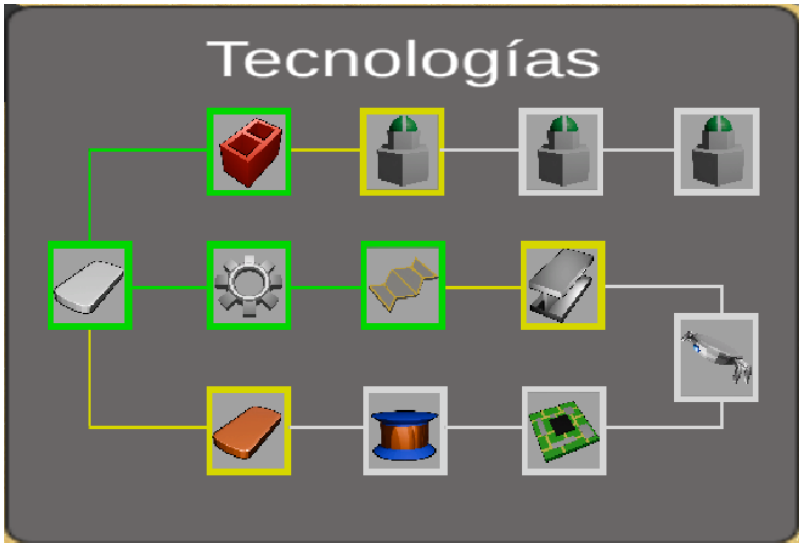


Figura 4.24: Árbol de tecnologías

4.2.5.1. Clase Investigar

Si bien esta clase no se asocia con el GameObject Tecnologías, si que lo hace con todos los botones que hay en él, por lo que se ha incluido en este punto. Además, es este script el que controla todo lo que tiene que ver con la investigación de nuevas tecnologías. Es relativamente similar a la clase Desplegable, pero con algunos cambios estructurales para adecuarse a la tarea que desempeña.

Nombre	Modificador	Tipo	Descripcion
ID	public	int	Identificador único del objeto que se investiga
ID1	public	int	Identificador único del primer objeto necesario para la investigación
ID2	public	int	Identificador único del segundo objeto necesario para la investigación
inventario	public	GameObject	GameObject del inventario
o1	private	Slot	Casilla del inventario que contiene el objeto con ID1

o2	private	Slot	Casilla del inventario que contiene el objeto con ID2
previo	public	GameObject	GameObject del conjunto de palos anterior, en caso de que exista
previo2	public	GameObject	GameObject del conjunto de palos anterior, en caso de que exista
actual	public	Slot	Slot en contacto con el botón que tiene este script asociado
siguientes	public	List<Slot>	Lista de los Slots que quedan disponibles tras investigar esta tecnología

Figura 4.25: Atributos de la clase Investigar

Nombre	Modificador	Parámetros	Tipo	Descripcion
Start	private		void	Inicializa objeto, o1 y o2
FixedUpdate	private		void	Asigna o1 y o2 a las casillas del inventario que contienen objetos con identificador igual a ID1 e ID2 respectivamente. También colorea las casillas del panel hijo desplegable en caso de que exista o no ese objeto en el inventario
investigando	public		void	Si la tecnología está disponible, añade ID a Inventario.investigado y quita del inventario una unidad al n de o1 y o2

desplegar	public		void	Activa el panel visual hijo del botón al que está asociado Desplegable solo si está disponible
replegar	public		void	Desactiva el panel visual hijo del botón al que está asociado Desplegable
OnDisable	private		void	Desactiva el panel visual hijo del botón al que está asociado Desplegable cuando este botón se desactiva

Figura 4.26: Funciones de la clase Investigar

4.2.6. GameObject Logros

Como se puede observar en las figuras 4.27 y 4.28, el GameObject logros está compuesto por dos paneles que hacen las veces de fondo, un conjunto de 4 casillas y un texto que explica qué hay que conseguir en cada logro. Cuando un logro se completa, se marca la casilla correspondiente con el único Sprite en "Otomatïki" que se obtuvo de una librería abierta.

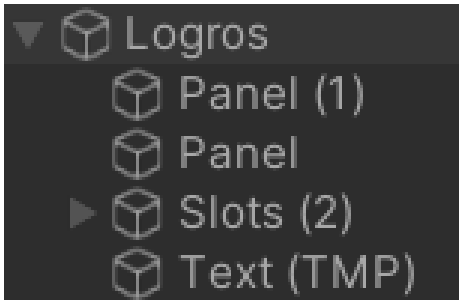


Figura 4.27: GameObjects del panel de logros

Figura 4.28: Panel de logros con uno de ellos completado

4.2.6.1. Clase Logros

Nombre	Modificador	Tipo	Descripcion
inventario	public	GameObject	GameObject del inventario
uno	public	bool	Booleano que comprueba si el primer logro se cumple
dos	public	bool	Booleano que comprueba si el segundo logro se cumple
tres	public	bool	Booleano que comprueba si el tercer logro se cumple
cuatro	public	bool	Booleano que comprueba si el cuarto logro se cumple
n	private	int	Numero de logros completados. Variable exclusiva para el MTQ
fase	private	int	Fase de los logros (entre 1 y 2). Variable exclusiva para el MTQ

Figura 4.29: Atributos de la clase Logros

Nombre	Modificador	Parámetros	Tipo	Descripcion
Start	private		void	Inicializa uno, dos, tres, cuatro y n
FixedUpdate	private		void	Comprueba si los tres primeros logros se han completado y, en caso de que así sea, activa la marca de compleción. Luego evalúa el cuarto logro y repite lo mismo

Figura 4.30: Funciones de la clase Logros

4.2.7. GameObject Tutorial

El GameObject Tutorial está compuesto por un GameObject que contiene dos paneles y dos cuadros de texto, tal y como se puede observar en las figuras 4.31 y 4.32. El primero de estos textos, que se encuentra sobre el primer panel ,es el que cambiará según avance el tutorial, mostrando los diferentes consejos. El segundo, sin embargo, contiene la instrucción para avanzar los consejos del tutorial, y no se modifica nunca.

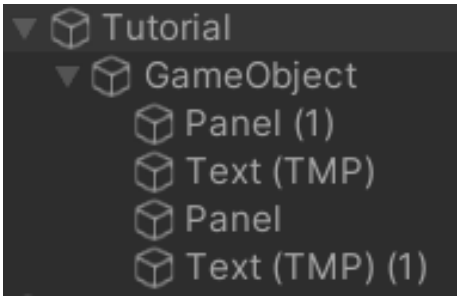


Figura 4.31: GameObjects del panel de tutorial

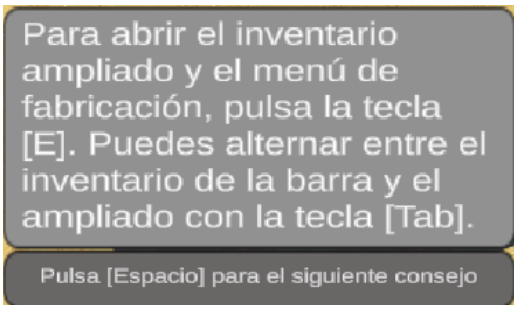


Figura 4.32: Panel de tutorial con un consejo activado

4.2.7.1. Clase Tutorial

Nombre	Modificador	Tipo	Descripcion
texto	private	TextMeshPro	Texto asociado al componente Text que cambia.
tutorial	public	int	Identificador único del consejo

Figura 4.33: Atributos de la clase Tutorial

Nombre	Modificador	Parámetros	Tipo	Descripcion
Start	private		void	Asigna texto al componente correspondiente, y recibe de la escena del menú inicial si se empieza con tutorial
Update	private		void	Cambia el texto en función al número asignado a tutorial, y si se pulsa espacio avanza al siguiente consejo

Figura 4.34: Funciones de la clase Tutorial

4.2.8. GameObject Movil

El GameObject Movil es el que sirve para mover los diferentes objetos entre las casillas del inventario con otras casillas del inventario, o con las casillas de los menús de la interfaz de los edificios.

Lleva asociado un componente de tipo Slotm, que es muy similar al componente Slot, pero con varias diferencias. La principal de ellas a nivel visual es que es una casilla sin fondo, que sigue al ratón continuamente, pero solo es visible cuando un lleva un objeto asociado. Por lo tanto, a nivel visual y como se puede apreciar en la figura 4.35, esta casilla solo tiene activos un panel, el cual lleva la imagen del objeto contenido, y un texto indicando la cantidad de unidades contenidas de ese objeto. El panel que no está activado es el que haría las veces de fondo de la casilla, que se mantuvo en el GameObject para dejar la posibilidad de añadirlo de manera sencilla.

Figura 4.35: GameObjects del Movil

4.2.7.1. Clase Slotm

Como ya se ha explicado antes, esta clase es muy similar a la clase Slot, pero sin tener las especificaciones para los menús como tecnologías o edificios, y teniendo una función para cambiar su contenido por el de otro Slot.

Nombre	Modificador	Tipo	Descripcion
ID	public	int	Identificador único del objeto contenido en la casilla
nombre	public	string	Nombre único del objeto contenido en la casilla
descripcion	public	string	Grupo de objetos al que pertenece el objeto contenido en la casilla
icono	public	Sprite	Imagen del objeto contenido en la casilla

vacio	public	bool	Variable que comprueba si la casilla está vacía
n	public	int	Cantidad de unidades del objeto contenido en la casilla

Figura 4.36: Atributos de la clase Slotm

Nombre	**Modificador**	**Parámetros**	**Tipo**	**Descripcion**
Start	private		void	Inicializa ID, nombre, descipcion, icono, vacio y n. Desactiva el GameObject Movil
FixedUpdate	private		void	Coloca Movil en la posición del ratón
actualizar	public		void	Activa el objeto si no está vacío, y le asigna la imagen del objeto contenido
cambiar	public	Slot objeto	Slot	Mete el contenido de objeto a las variables locales de Movil, y devuelve el valor que tenían estas para asignarlo a la casilla que llamó a este método

Figura 4.37: Funciones de la clase Slotm

4.3. Jugador

En esta sección se hablará del GameObject jugador, y de todo lo que este objeto lleva asociado.

Para empezar, el GameObject jugador es un objeto modelado completamente en Blender, por lo que se explicará el modelo.

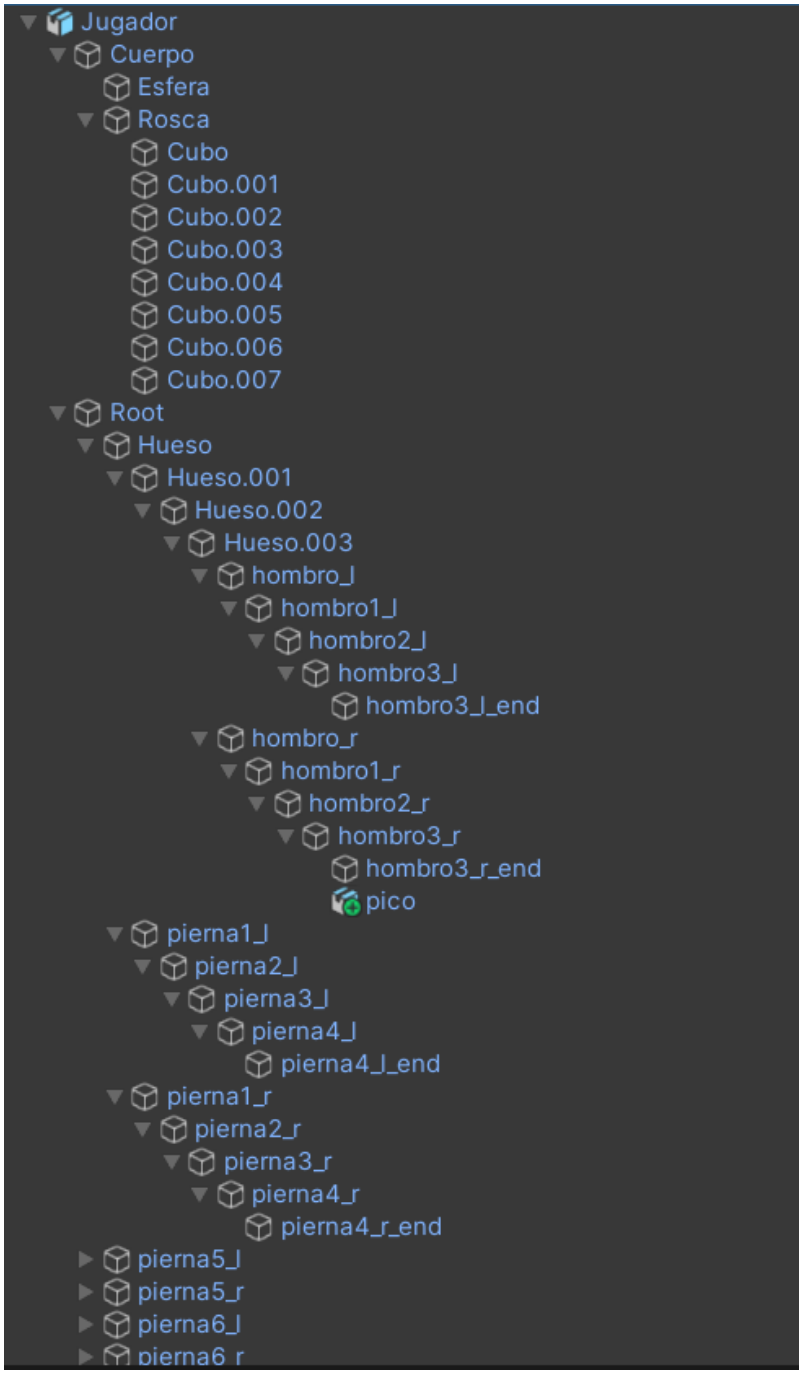


Figura 4.38: Modelo del jugador importado desde Blender

4.3.1. Modelo del jugador

Tal y como se puede ver en la figura 4.38, el modelo del jugador se divide en dos partes.

La primera de ellas, el GameObject Cuerpo, que es el modelo principal del cuerpo del personaje del jugador. Este modelo se generó a partir de un rectángulo, generando cortes a este para dar al torso una forma más natural, y extruyendo caras concretas para dar forma a los brazos y piernas. Además, contiene la cabeza del modelo, compuesta por la rosca y la esfera central de esta.

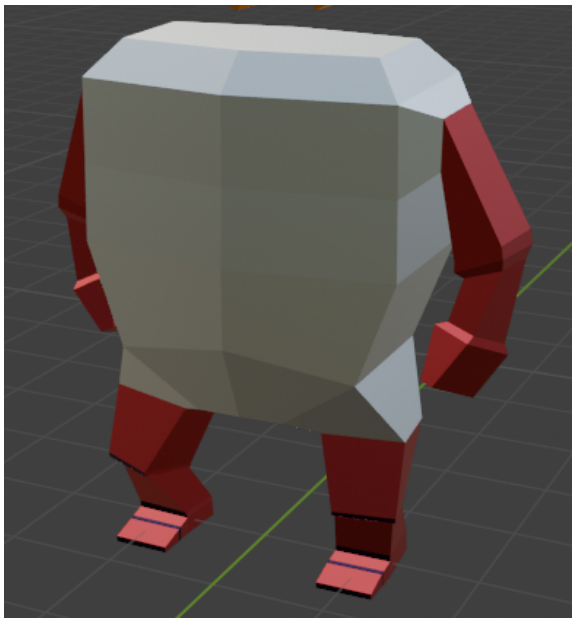


Figura 4.39: Modelo del cuerpo

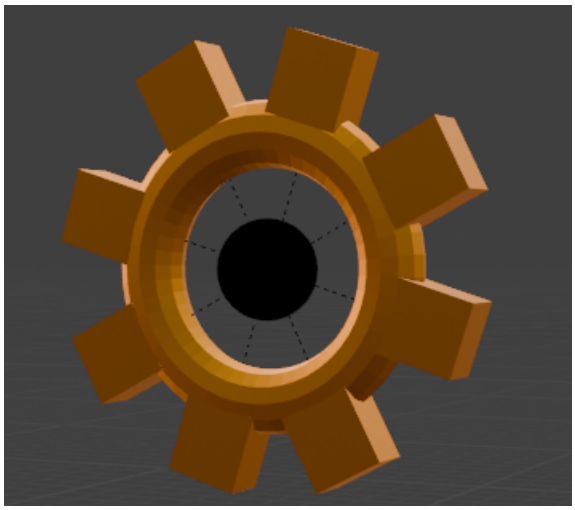


Figura 4.40: Modelo de la cabeza, conformada por los hijos del GameObject Cuerpo

El GameObject que le da el movimiento a todo el cuerpo como conjunto es el GameObject Root, ya que está situado en el centro de la cadera del personaje, y todos los demás huesos (que están unidos a trozos del modelo) dependen de él.

Como se puede comprobar en la figura 4.38, Root tiene cinco objetos hijos: Hueso, y los 4 pivotes de movimiento de las piernas (pierna5_l, pierna5_r, pierna6_l, pierna6_r). Para obtener un movimiento realista en Blender a la hora de animar unas piernas, se deben agregar dos pivotes a cada una de ellas, uno delante de la rodilla y otro tras el tobillo, que tras asociarse con esas partes del modelo, hacen que al mover los pivotes se mueva el hueso correspondiente del modelo, restringiendo movimientos antinaturales.

Por otra parte, el objeto Hueso contiene el esqueleto que permite el movimiento del modelo Cuerpo. Estos huesos se dividen de la siguiente manera:

- El torso, que tiene los huesos Hueso.001, Hueso.002 y Hueso.003.
- Los brazos, que tienen los huesos que dependen de hombro_l y hombro_r, y que ambos tienen un hueso para el hombro, otro para el brazo superior, otro para el antebrazo, uno para la mano y otro justo al final de esta. Además, la mano derecha lleva también asociado el GameObject Pico, que se hará visible cuando el jugador esté minando sobre un recurso. Se añade en ese punto para que siga el movimiento de la mano con naturalidad.

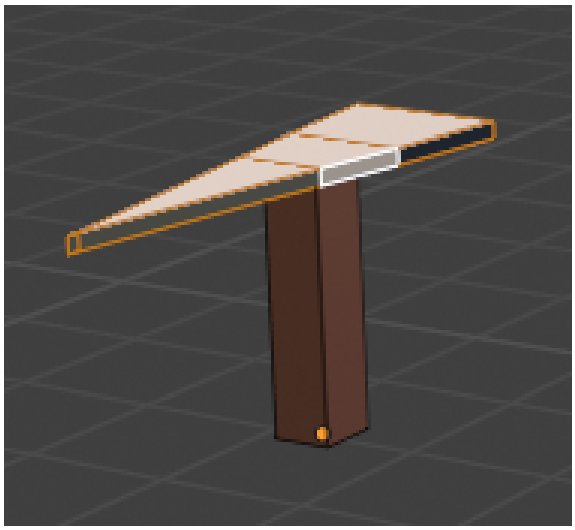


Figura 4.41: Modelo del pico, oculto en la mayoría de casos

- Las piernas, con una estructura similar a los brazos, teniendo huesos que dependen de pierna1_l y pierna1_r, siendo estos la cadera, el fémur, la tibia, el pie y el final del mismo. Como ya se ha comentado anteriormente, también existen los pivotes que ayudan al movimiento de estas extremidades, que se sitúan fuera de esta jerarquía para moverse con libertad sin desplazar otros huesos que no sean los puntos articulares del modelo.

4.3.2. Clase jugador

A continuación, se explica la clase Jugador, que se asocia al personaje añadiendo el script del mismo nombre al GameObject Jugador. Esta clase controla la gran mayoría de inputs que se reciben por teclado o ratón en el juego, a excepción del botón para avanzar el tutorial.

Nombre	Modificador	Tipo	Descripcion
inventario	public	GameObject	GameObject del inventario
minando	public	bool	Booleano que comprueba si se está minando
mena	private	GameObject	GameObject de la baldosa del suelo con depósito que se está pisando
panel	public	GameObject	Panel de ayuda con información sobre los menús
rb	private	Rigidbody	Booleano que comprueba si se está minando
ray	private	Ray	Ray que sale del ratón y colisiona con un rigidbody
hit	private	RaycastHit	Objeto golpeado por el ray
pico	public	GameObject	GameObject del pico del modelo
barraP	public	GameObject	GameObject de la barra de progresión de minado
slider	private	Slider	Componente Slider de barraP
progreso	private	float	Progreso de la barra del slider
ayuda	private	bool	Variable para comprobar si se accede varias veces al inventario. Exclusivo para el MTQ
lista	private	List>Object>	Lista de objetos del juego

Figura 4.42: Atributos de la clase Jugador

Para entender las siguiente funciones que se encuentran en la clase Jugador, es necesario señalar los GameObjects a los que hacen referencia los atributos de esta clase:

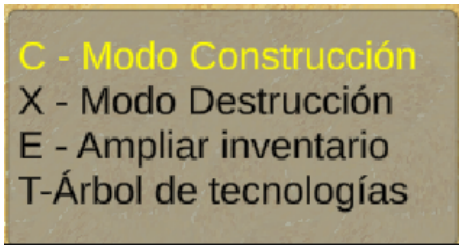


Figura 4.43: Panel de ayuda con la clave de las teclas, estando el modo de construcción activo

Figura 4.44: Barra de progresión de minado

Nombre	**Modificador**	**Parámetros**	**Tipo**	**Descripcion**
Start	private		void	Inicializa todos los atributos, crea el inventario llamando a crear_inv y llama al método reset
Update	private		void	Ver subsección 4.3.2.1.
OnCollision-Stay	private	Collider collider	void	Si está en contacto con un depósito, y minando es true, invoca minar y barra, y cambia la animación a minar. Si no está minando, para la invocación de minar y barra, para resetear slider, y cambiar la animación a Idle
OnCollision-Exit	private	Collider collider	void	Para la invocación de minar y barra, para resetear slider, y cambiar la animación a Idle

minar	public		void	Si está sobre un depósito, añade una unidad del objeto asociado con mena a inventario mediante el método anadir.
barra	private		void	Iguala el porcentaje del slider con un número proporcional a progreso, y aumenta este último
reset	public		void	Marca como deseleccionados todos los elementos de panel

Figura 4.45: Funciones de la clase Jugador

4.3.2.1. Upgrade de la clase Jugador

Como el método Update de esta clase es demasiado largo para describirlo en una tabla, se va a describir a continuación. Para ello, cabe dividirlo en dos partes principales: la parte de inputs, y la parte que gestiona el Raycast.

La primera de las partes gestiona los inputs de todo el juego, siendo estos la manera de activar los distintos modos, abrir los menús y gestionar si el personaje está minando. Cabe destacar que, a excepción de la tecla F y la tecla H, todas las demás desactivan los menús y modos con los que no están relacionadas:

- La tecla F: cuando se pulsa, coloca minando a true, y cuando se levanta coloca minando a false. Esto hace que minado se mantenga a true siempre y cuando se mantenga presionada la tecla.
- La tecla H: cuando se pulsa, activa el tutorial si no lo está ya.
- La tecla C: cuando se pulsa,activa el modo construcción del inventario, o lo desactiva si ya estaba activo. Luego llama al método reset y, si se ha activado el modo correspondiente, lo marca como activo en panel.
- La tecla X: cuando se pulsa,activa el modo destrucción del inventario, o lo desactiva si ya estaba activo. Luego llama al método reset y, si se ha activado el modo correspondiente, lo marca como activo en panel.
- La tecla E: cuando se pulsa,abre el inventario ampliado, o lo cierra si ya estaba abierto. Luego llama al método reset y, si se ha abierto el menú correspondiente, lo marca como activo en panel.
- La tecla T: cuando se pulsa,abre el árbol de tecnologías, o lo cierra si ya estaba abierto. Luego llama al método reset y, si se ha abierto el menú correspondiente, lo marca como activo en panel.

- La tecla L: cuando se pulsa, abre el menú de logros, o lo cierra si ya estaba abierto. Luego llama al método reset.
- La tecla Tab: cuando se pulsa, llama al método invertir del inventario.
- La tecla Esc: cuando se pulsa, cierra todos los menús, desactiva todos los modos y llama a reset.
- Número del 1 al 0: cuando se pulsa llama a inventario.seleccionar pasando el código numérico de esa tecla como parámetro.

La segunda de las partes del método Update, lo primero que hace es asignar ray a la posición del ratón en la pantalla, y si no hay menús abiertos crea el hit haciendo un Raycast a ray. Después, se abre un if-else if:

- Si se pulsa el click derecho del ratón, el jugador se mueve utilizando su NavMeshAgent, y si se toca un edificio, se abre su menú. Si el jugador se está moviendo, activa su animación Run, y si está parado activa su animación Idle.
- Por otro lado, si el modo de destrucción está activado, se entra en este else if: Si el objeto en contacto es un edificio, asigna su booleano destruyendo de la clase Edificio a true. Si por el contrario, el objeto en contacto es una conta de transporte, asigna su booleano destruyendo de la clase Via a true.

  Después, cuando se pulsa el click izquierdo, añade una unidad de ese edificio al inventario en caso de que lo sea, o n unidades en caso de que sea una cinta de transporte, siendo n el numero de hijos del objeto padre del golpeado por el raycast.

4.3.3. Componente animator

Para animar el movimiento de las diferentes partes del modelo del jugador, se asocia al GameObject de ese modelo un componente Animator. Para crear ese movimiento, se realizaron en Blender distintas capturas de animaciones. Tres de estas cuatro capturas se realizaron moviendo partes del cuerpo del personaje, por lo que se usó el esqueleto previamente creado. En el apartado 3.4.3.3. se pueden observar estas animaciones, teniendo cada una de ellas dos keyframes entre los que se alterna el movimiento.

Para crear las transiciones entre estas animaciones, se utlizó un componente animator, que se divide en dos layers, que son grupos de animaciones independientes que pueden ser ejecutados simultáneamente:

- El primero y más sencillo de ellos, es la animación de la cabeza. Esta consta solo de una animación, que no utilizó el esqueleto del modelo, ya que tiene la única función de girar el engranaje que hace las veces de cabeza del personaje sobre si mismo.

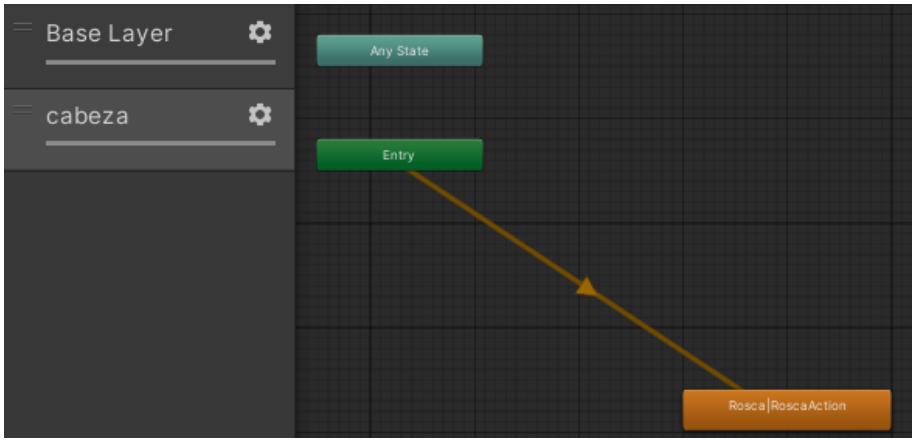


Figura 4.46: Layer de animación de la cabeza

- El segundo es el que gestiona la transición entre las animaciones del cuerpo. Para ello utiliza dos parámetros llamados Run y Minar. Como se puede apreciar en la imagen 4.47, solo hay transiciones entre el estado Idle con los estados Run y Minar. Esto se debe a un que, según como se han estructurado los parámetros (figura 4.48), podría haber conflicto en caso de tener ambos parámetros en verdadero. Para evitarlo, si se quiere transicionar de la animación Run a la animación Minar o viceverse, se deberá pasar obligatoriamente por la animación Idle. Los parámetros se modifican, como se ha explicado en el punto 4.3.2., con la clase Jugador, de manera acorde con la transición que se acaba de explicar.

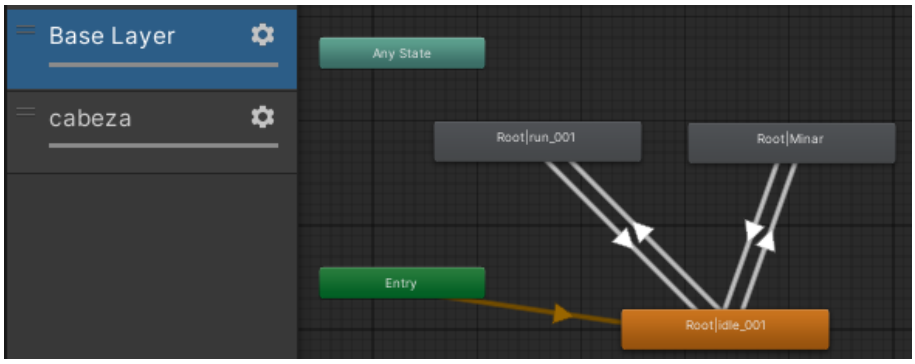


Figura 4.47: Layer de animación del cuerpo

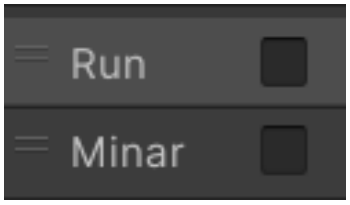


Figura 4.48: Parámetros del animator

4.3.4. Componente NavMeshAgent

Para hacer que el movimiento fuera fluido y atractivo para el jugador, teniendo en cuenta los elementos que iban a aparecer y desaparecer en el entorno, y sobre todo, para poder mover el personaje con el ratón, se utilizó un componente NavMeshAgent, que facilita la implementación de todos los requisitos que se tenían que cumplir en cuanto al movimiento.

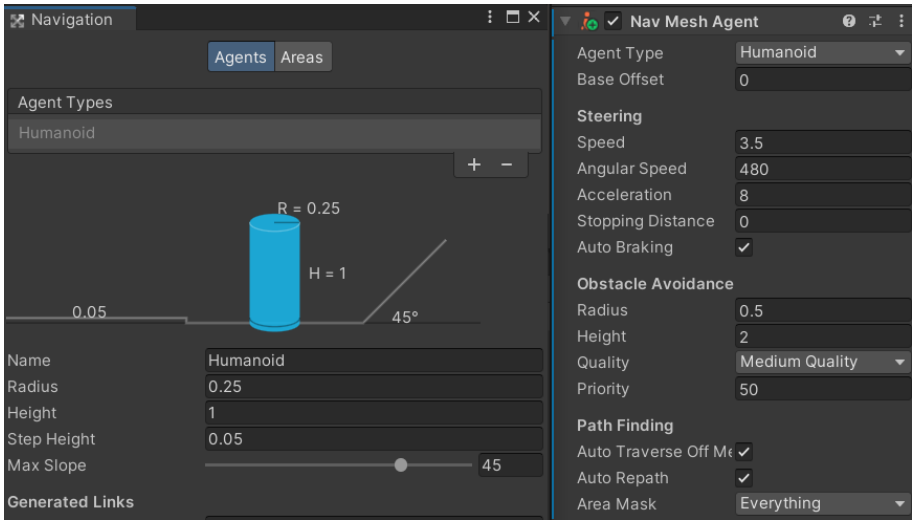


Figura 4.49: Componente NavMeshAgent y agente humanoide

Como se puede comprobar en la figura 4.49, para utilizar el componente se creó un agente humanoide con diferentes propiedades. El radio del agente se tuvo que disminuir para que cupiera entre dos edificios relativamente cercanos sin problema, y la altura del escalón se tuvo que disminuir a 0.05, a fin de hacer que los límites del mapa, delimitado por agua, no tuvieran que estar muy por debajo o por encima del nivel, aportando una sensación de continuidad en el terreno que no se puede pisar. Por último, la velocidad angular de giro del NavMeshAgent se cambió para dar un movimiento más realista al personaje cuando cambiaba de dirección.

4.4. Cámara

Los GameObjects Camara1 y Directional Light cumplen con la función de mantener al jugador inmerso en el mundo.

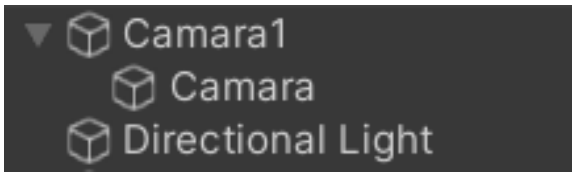


Figura 4.50: GameObjects Camara1 y Directional Light

El primero de ellos es un contenedor para la cámara propiamente dicha, que es el GameObject Camara. Este contenedor utiliza un script para seguir al jugador, el cual se expondrá a continuación. Cabe destacar que se colocó la cámara con un offset 0,14,-7, lo que crea un ángulo de 26 grados con el personaje, que al estar en el centro de la imagen, genera una visión del entorno completa e inmersiva.

Por otro lado, el segundo de ellos es un foco que alumbra en todas las direcciones, y se colocó de tal manera que la sombra sobre el mundo fuera diagonal, para ser apreciable y aumentar la sensación de profundidad que de por si ya da Camara.

4.4.1. Clase Camara

Nombre	Modificador	Tipo	Descripcion
target	public	Transform	Objetivo a seguir por la cámara, en este caso el jugador
offset	public	Vector3	Ditancia relativa a la que tiene que quedar la cámara del objetivo

Figura 4.51: Atributos de la clase Camara

Nombre	Modificador	Parámetros	Tipo	Descripcion
LateUpdate	private		void	Coloca la posición del objeto asociado a este script en la posicion de target más offset

Figura 4.52: Función de la clase Cámara

4.5. Suelo

Suelo es el último de los GameObjects de la escena principal. Dentro de este GameObject se guardan las baldosas (casillas que componen el mundo, para futuras referencias) dentro del GameObject Mundo, y las cintas de transporte dentro del GameObject Logistica.

Figura 4.53: GameObjects Suelo, Mundo y Logística

El GameObject Suelo lleva asociado un componente NavMeshSurface, que se genera teniendo en cuenta el agente utilizado en el Jugador. Este componente define la superficie por la que puede caminar el agente teniendo en cuenta las restricciones impuestas por este último.

Sin embargo, se ha configurado más allá, excluyendo la Layer Logística, que se asigna a todas las cintas de transporte. Esto hace que no tengan impacto en el NavMeshAgent, y por tanto que no impidan su movimiento, al contrario que los edificios, que sí que establecen una barrera que el Jugador no puede atravesar.

Para entender mejor todos los componentes que llevan asociados este GameObject, y por extensión todo lo que acaba dependiendo de él, se van a dividir las clases por funcionalidades.

4.5.1. Generación del mundo

Para generar el mundo se utiliza el script HexGrid, que va asociado al GameObject Suelo. Este script contiene toda la información necesaria para crear el mundo, y utiliza un algoritmo que crea el entorno utilizando un tamaño que se asigna en el inspector de Unity. Se crea una matriz nxn de baldosas que va de -n/2 a (n/2)-1. Esto acaba creando un cuadrado, que para la Vertical Slice se ha decidido que sea de 52x52, lo cual, como se explicará a la hora de exponer el método que genera el mundo, deja una superficie caminable de 29x29 baldosas.

4.5.1.1. HexGrid

Nombre	Modificador	Tipo	Descripcion
tamano	public	Vector2Int	Tamaño total del mapa
rad1	public	float	Radio del hueco central de la baldosa
rad2	public	float	Radio total de la baldosa

alt	public	float	Altura del prisma hexagonal que compone la baldosa
material	public	Material	Material de las baldosas básicas del mapa
agua	public	Material	Material de las baldosas de agua del mapa
hierro	public	Material	Material de las baldosas de hierro del mapa
cobre	public	Material	Material de las baldosas de cobre del mapa
piedra	public	Material	Material de las baldosas de arcilla del mapa
luz	public	Material	Material de las baldosas de luz del mapa
carbon	public	Material	Material de las baldosas de carbon del mapa
construyendo	public	Material	Material de los edificios mientras se construyen
destruyendo	public	Material	Material de los edificios mientras se destruyen
hornom	public	Material	Material de los hornos construidos
minam	public	Material	Material de las minas construidas
fabricam	public	Material	Material de las fábricas construidas
lista	private	List<Object>	Lista de los objetos del inventario
inventario	public	GameObject	GameObject del inventario
minaPrefab	public	GameObject	Prefab de la mina
fabricaPrefab	public	GameObject	Prefab de la fábrica
hornoPrefab	public	GameObject	Prefab del horno
viaPrefab	public	GameObject	Prefab de la vía
movil	public	GameObject	GameObject de la casilla movil

hierros	public	List<Vector2Int>	Lista de baldosas con depósitos de hierro
cobres	public	List<Vector2Int>	Lista de baldosas con depósitos de cobre
piedras	public	List<Vector2Int>	Lista de baldosas con depósitos de arcilla
luces	public	List<Vector2Int>	Lista de baldosas con depósitos de luz
carbones	public	List<Vector2Int>	Lista de baldosas con depósitos de carbón

Figura 4.54: Atributos de la clase HexGrid

Nombre	**Modificador**	**Parámetros**	**Tipo**	**Descripcion**
Start	private		void	Asigna lista, inicializa las listas de baldosas y añade a ellas las casillas que contendrán depósitos, para luego llamar a Grid
Grid	private		void	Ver subsección 4.5.1.2.
Coordenadas	private	Vector2Int coord	Vector3	Obtiene las coordenadas tridimensionales de la baldosa a partir de su posición en la matriz
actualizar	public		void	Actualiza el componente NavMeshSurface del GameObject Suelo
AddHex	private	List <Vector2Int> list, Vector2Int pos	void	Añade a la lista 7 baldosas que forman un hexágono entre sí, siendo pos la esquina superior izquierda

Figura 4.55: Funciones de la clase HexGrid

4.5.1.2. Función Grid

La función Grid tiene como objetivo el crear el mapa desde cero. Este mapa siempre es el mismo, ya que tiene definido por parámetros el tamaño del mapa y la posición de los depósitos y el agua. Esta función consta de un doble bucle, que acaba creando una matriz en este caso de 52x52, y que crea la matriz por filas. Este es el proceso que se sigue en la creación de cada una de las baldosas.

- El primer paso es crear un objeto HexRender y asignarle la posición tridimensional que resulta de llamar a Coordenadas con su posición en la matriz para luego asignar todos los atributos de ese HexRender con los que HexGrid recibe como parámetro, lo cual hace que no sea necesario añadirlos en el Inspector de Unity de manera manual, simplificando mucho la creación del entorno.
- Después, se comprueba si esa baldosa recién creada está en algún punto entre (-n/2)+12 y (n/2)-12 en alguno de los dos ejes. En caso de ser cierto, a esa casilla se le asignarán el material agua, el tag ”agua”, y se reducirá su altura en 0.1f, lo que causará que el componente NavMeshSurface no lo procese como superficie caminable, por lo que no se podrá caminar por ella. Esto crea una superficie caminable de 29x29 en el caso de la Vertical Slice.
- Luego, se comprueba si la baldosa está en alguna de las listas de los depósitos (hierros, cobres, piedras, luces o carbones), y en caso de ser afirmativo, se le asigna el material correspondiente a ese tipo de baldosa, y se le asocia un componente Objeto que contendrá el recurso básico correspondiente a esa lista.
- Si la baldosa no es agua y no está en ninguna de las lista de los depósitos, se le asigna el tag ”suelo”, y el material material, creando así una baldosa de suelo básica.
- Tras este proceso, se llama al método DrawMesh del HexRender, y se asigna ese HexRender como hijo del GameObject Mundo.

Fuera del doble bucle y para acabar, se llama al método actualizar de HexGrid para calcular la NavMeshSurface.

4.5.1.3. Clase HexRender

Se ha mencionado antes que HexGrid crea un HexRender para cada baldosa. Esta clase gestiona todas las interacciones posibles con cada una de las baldosas del suelo, y los otros objetos que toman la forma utilizando esta clase, que se expondrán en los puntos posteriores.

Nombre	Modificador	Tipo	Descripcion
selec	public	int	N de la casilla del inventario seleccionada

rad1	public	float	Radio del hueco central de la baldosa
rad2	public	float	Radio total de la baldosa
alt	public	float	Altura del prisma hexagonal que compone la baldosa
material	public	Material	Material de la baldosa
construyendo	public	Material	Material de los edificios mientras se construyen
destruyendo	public	Material	Material de los edificios mientras se destruyen
hornom	public	Material	Material de los hornos construidos
minam	public	Material	Material de las minas construidas
fabricam	public	Material	Material de las fábricas construidas
inventario	public	GameObject	GameObject del inventario
edificado	public	GameObject	GameObject del objeto que se edifica sobre la baldosa
minaPrefab	public	GameObject	Prefab de la mina
fabricaPrefab	public	GameObject	Prefab de la fábrica
hornoPrefab	public	GameObject	Prefab del horno
viaPrefab	public	GameObject	Prefab de la vía
movil	public	GameObject	GameObject de la casilla movil
mesh	private	Mesh	Malla que conforma el prisma hexagonal
mfilter	private	MeshFilter	Componente que accede al mesh
mrenderer	private	MeshRenderer	Componente que renderiza el mesh
mcaras	private	List<Cara>	Lista de caras que conforman el mesh
ocupado	public	bool	Booleano verdadero si hay un edificio sobre la baldosa
pos	public	Vector2	Posición de la baldosa en la matriz

Figura 4.56: Atributos de la clase HexRender

Nombre	Modificador	Parámetros	Tipo	Descripcion
Awake	private		void	Asigna mfilter, mrenderer e inicializa mesh. Asigna el mesh de mfilter al nuevo mesh
DrawMesh	public		void	Asigna material a mrenderer, y luego llama a DrawFaces y a CombineFaces
DrawFaces	private		void	Crea el prisma hexagonal llamando a NuevaCara seis veces para la parte superior, seis veces para la parte inferior y doces veces en total para los lados, y añade todas a mcaras
Combine-Faces	private		void	Combina las caras creadas con DrawFaces utilizando mcaras para obtener sus vértices, ángulos y triángulos, para añadir estas tres categorias al mesh. Luego congela la rotación y la traslación del objeto usando su componente Rigidbody, crea un MeshCollider con la forma del mesh creado, y se lo añade
NuevaCara	private	float rad1, float rad2, float alt, float alt1, int punto, bool inv	Cara	Crea cuatro vectores usando Punto, con los que crea los vértices de una Cara, que se devuelve tras componerla

Punto	protected	float tam, float alt, int ind	Vector3	Crea un punto usando la posición relativa al centro de la cara mediante ind, multiplicando tam con el coseno y el seno del ángulo obtenido con ind, y colocandolo en altura alt
OnMouse-Over	private		void	Ver subsección 4.5.1.5.
OnMouse-Exit	private		void	Destruye edificado en caso de exista
Coordenadas	private		Vector3	Obtiene las coordenadas tridimensionales de la baldosa a partir de pos

Figura 4.57: Funciones de la clase HexRender

4.5.1.4. Struct Cara

Se ha mencionado que el método NuevaCara crea una Cara. Este es un struct auxiliar que tiene como atributos List<Vector3>vértices, List<int>triangulos y List<Vector2>coord. Este struct posee un constructor Cara, que asigna todos los atributos con los datos que recibe como parámetros.

4.5.1.5. Función OnMouseOver

Por último, cabe explicar la función OnMouseOver. Esta es una función nativa de Unity, que se ejecuta en un fram si el ratón está encima de la baldosa en cuestión, en este caso. Este método se ha utilizado para llevar a cabo la construcción de edificios y elementos logísticos.

Primero, se realizan varias comprobaciones para entrar al método en sí: la casilla Movil tiene que estar vacía, el modo de construcción tiene que estar activado, y el inventario ampliado tiene que estar oculto. Después se crea un Slot que guarda la información de la casilla seleccionada de la barra del inventario.

Luego, se comprueba que la baldosa en cuestión no esté ocupada, que el Slot que se acaba de crear contenga un edificio o una cinta de transporte, que el selec sea igual al Slot que se ha creado.

En caso de que ninguno de los ifs de los dos párrafos anteriores se cumpla, se destruye edificado en caso de que exista. Por el contrario, si se han cumplido ambas condiciones, se entra en la parte principal del método, que se divide en dos partes principales:

- La primera de ellas instancia el prefab en el atributo edificado si el objeto contenido en el Slot es un edificio, y le asigna un mesh igual que el de la baldosa, pero con una altura superior. Se coloca su variable construyendo en true, y se le asigna el material de construcción. Si por otro lado el Slot contiene cintas de transporte, comprueba que posI y posF del inventario sean distintas de null y entre sí, y llama al método Construir de la clase Logística, que está asociada al GameObject Suelo, y se explica en profundidad en el punto 4.5.3.
- La segunda parte solo se aplica si el objeto del Slot es un edificio. Para ello, comprueba el tipo de edificio que se contiene, se le asignan los atributos de Edificio y, si se pulsa el click izquierdo del ratón, se coloca el edificio, activando el MeshCollider que se ha creado previamente, se asigna edificado como hijo de la baldosa donde se crea, y se le añade la clase concreta de su tipo de edificio (Mina, Fabrica u Horno, que se explican en el 4.5.2.). Tras hacer todo esto, se marca la baldosa como ocupada, se quita una unidad del edificio colocado del inventario, iguala edificado a null y actualiza el NavMeshSurface de HexGrid.

Figura 4.58: Edificio asignado como hijo de una baldosa

4.5.2. Edificios

Se ha explicado antes que hay tres tipos de edificios: las minas, los hornos y las fábricas. Estos cumplen la función de automatizar las funciones de minado y fabricación, que ayudan al jugador acelerando el ritmo al que avanza en el juego. Todos estos edificios tienen un script propio de su clase, y un script Edificio que comparten los tres, por lo que se explicará este último el primero.

4.5.2.1. Clase Edificio

Esta clase es la que controla todo lo relacionado con el edificio de manera general, sus materiales, las cintas que tiene asociadas, y atributos comunes a todos los edificios.

Nombre	Modificador	Tipo	Descripcion
inventario	public	GameObject	GameObject del inventario
pos	public	Vector2	Posición en el mundo tridimensional, si ncontar la altura

gridp	public	Vector2	Posición de la baldosa sobre la que está el edificio en la matriz de baldosas
menus	public	bool	Verdadero si el menú del edificio está abierto
cerca	private	bool	El jugador está cerca del edificio
vias	public	List <GameObject>	Lista de cintas que tienen como destino el edificio
salida	public	GameObject	Cinta que sale del edificio
material	public	Material	Material básico del edificio, que cambia dependiendo del tipo
destr	public	Material	Material del edificio en destrucción
constr	public	Material	Material del edificio en construcción
destruyendo	public	bool	Edificio en destrucción si es verdadero
construyendo	private	bool	Edificio en construcción si es verdadero

Figura 4.59: Atributos de la clase Edificio

Nombre	**Modificador**	**Parámetros**	**Tipo**	**Descripcion**
Start	private		void	Inicializa la salida a null
GetPos	private		void	Si el objeto seleccionado de la barra del inventario es una cinta de transporte, asigna las variables del inventario posI si es nula, o posF si es nula pero posI no lo es

FixedUpdate	private		void	Coloca el material del MeshRenderer a constr si se esta construyendo, a destr si se está destruyendo, o a material en caso de que no sea ninguno de los dos. Activa el menú del edificio se se cumple menus, cerca, y están cerrados el inventario ampliado y el árbol de tecnologías, y lo desactiva en caso contrario
OnTrigger-Enter	private	Collider coll	void	Si un objeto con el tag "Player" (Jugador) entra en contacto con el collider del edificio, asigna cerca en true
OnTrigger-Exit	private	Collider coll	void	Si un objeto con el tag "Player" (Jugador) para de estar en contacto con el collider del edificio, asigna cerca y menus en false
OnDestroy	private		void	Cuando se destruye el edificio, destruye la cinta asociada a salida y todas las cintas que contiene la lista vias, en caso de que haya alguna. Esto se realiza con el objetivo de que no haya un cintas sin edificios asociados. También se añaden al inventario los objetos que contuvieran las casillas tanto del edificio como de las vías asociadas

Figura 4.60: Funciones de la clase Edificio

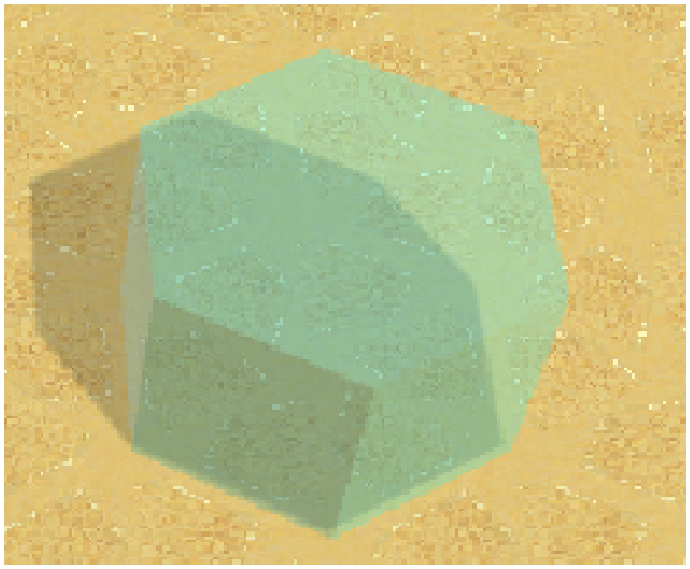


Figura 4.61: Edificio en construcción

Figura 4.62: Edificio en destrucción

4.5.2.2. Clase Mina

Esta clase es la que se asocia a los prefabs Mina tras instanciarlos, y contiene todo lo necesario para que el edificio recoja objetos del depósito sobre el que se coloca. De manera adicional al resto de edificios, al colocar uno de esta clase, solo se puede hacer sobre un depósito de recursos.

Nombre	Modificador	Tipo	Descripcion
objeto	public	Objeto	Objeto del recurso que hay en el depósito
slot	public	Slot	Casilla de la mina
limite	public	int	Limite asignable de objetos máximos en la casilla de la mina
slots	public	List<Slot>	Lista de las casillas de la mina, que aunque sea una se utiliza para que el método OnDestroy de edificio sea universal
lleno	private	bool	El n de slot ha alcanzado limite
barraP	public	GameObject	GameObject de la barra de progresión de minado
slider	private	Slider	Componente Slider de barraP
progreso	private	float	Progreso de la barra del slider

Figura 4.63: Atributos de la clase Mina

Nombre	Modificador	Parámetros	Tipo	Descripcion
Empezar	public		void	Asigna slider, lleno, asigna todos los elementos de slot con el contenido de objeto, añade slot a slots e invoca los métodos minar y barra
FixedUpdate	private		void	Si el n del slot alcanza el límite y no está lleno, se cancela la invocación, se resetea progreso y se asigna lleno a true. Si no, vuelve a invocar minar y barra

minar	public		void	Añade una unidad al n de slot
barra	private		void	Iguala el porcentaje del slider con un número proporcional a progreso, y aumenta este último
actualizar	public		void	Actualiza la imagen y el texto de la casilla

Figura 4.64: Función de la clase Mina

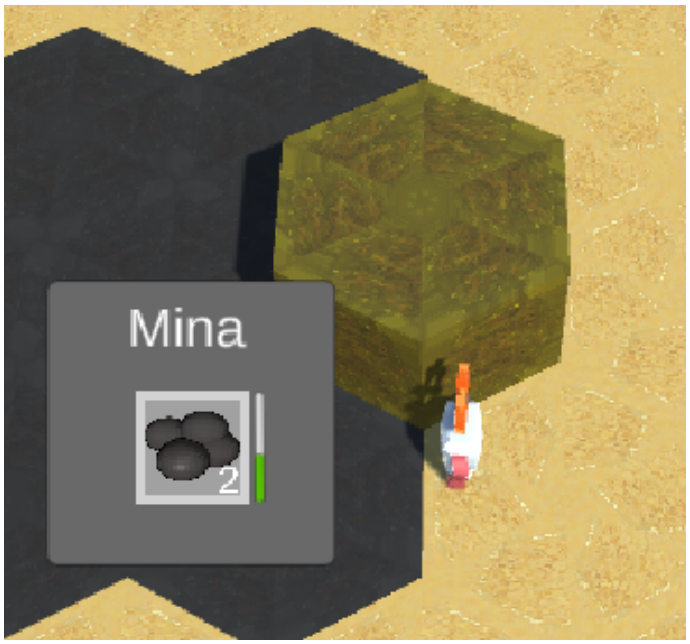


Figura 4.65: Mina colocada sobre un depósito de carbón con el menú abierto

4.5.2.3. Clase Horno

Esta clase es la que se asocia a los prefabs Horno tras instanciarlos, y contiene todo lo necesario para que el edificio fabrique los materiales básicos utilizando recursos y carbón.

Nombre	Modificador	Tipo	Descripcion
inventario	public	GameObject	GameObject del inventario
lista	public	List<Object>	Lista de los objetos del juego

limite	public	int	Limite asignable de objetos máximos en la casilla de salida del horno
slots	public	List<Slot>	Lista de las casillas de la mina, que aunque sea una se utiliza para que el método OnDestroy de edificio sea universal
barraP	public	GameObject	GameObject de la barra de progresión de minado
slider	private	Slider	Componente Slider de barraP
progreso	private	float	Progreso de la barra del slider

Figura 4.66: Atributos de la clase Horno

Nombre	Modificador	Parámetros	Tipo	Descripcion
Start	public		void	Asigna slider, lista, y crea tres casillas vacias que añade a slots
fabricar	public		void	Elimina una unidad al n de las dos primeras casillas, y añade una al n de la tercera casilla
barra	private		void	Iguala el porcentaje del slider con un número proporcional a progreso, y aumenta este último
actualizar	public		void	Actualiza la imagen y el texto de la casilla

FixedUpdate	private		void	Asigna a la segunda casilla el objeto carbón, que será el único que acepte. Luego, asigna a la tercera casilla un objeto en base al que se encuentra en la primera. Invoca fabricar y barra si hay materiales suficientes y no está lleno, comprobando los objetos de las casillas por seguridad, y los cancela en caso contrario, reseteando progreso. Vacía la primera casilla en caso de que su n sea 0 para evitar olapamientos de los materiales, y por último llama a actualizar

Figura 4.67: Funciones de la clase Horno

Figura 4.68: Horno fabricando un ladrillo a partir de arcilla y carbón

4.5.2.4. Clase Fabrica

Esta clase es la que se asocia a los prefabs Fábrica tras instanciarlos, y contiene todo lo necesario para que el edificio fabrique todos los objetos que no puede fabricar el horno. Este prefab cuenta con dos menús, el primero de ellos para elegir el objeto a fabricar, y el segundo de ellos siendo muy similar al de los prefabs Horno. Para cambiar entre ellos, se asignará un valor a IdItem pulsando los botones sobre las casillas del primer menú.

Nombre	Modificador	Tipo	Descripcion
inventario	public	GameObject	GameObject del inventario
lista	public	List<Object>	Lista de los objetos del juego
limite	public	int	Limite asignable de objetos máximos en la casilla de salida del horno
slots	public	List<Slot>	Lista de las casillas de la mina, que aunque sea una se utiliza para que el método OnDestroy de edificio sea universal
barraP	public	GameObject	GameObject de la barra de progresión de minado
slider	private	Slider	Componente Slider de barraP
progreso	private	float	Progreso de la barra del slider
IdItem	private	int	Identificador único del objeto que se va a fabricar
menu	private	bool	Booleano para el cambio de pantalla

Figura 4.69: Atributos de la clase Fabrica

Figura 4.70: Menú de la fábrica con todos los objetos que se pueden crear

Nombre	Modificador	Parámetros	Tipo	Descripcion
Start	public		void	Asigna slider, lista, y crea tres casillas vacias que añade a slots
fabricar	public		void	Elimina una unidad al n de las dos primeras casillas, y añade una al n de la tercera casilla
barra	private		void	Iguala el porcentaje del slider con un número proporcional a progreso, y aumenta este último
actualizar	public		void	Actualiza la imagen y el texto de la casilla
seleccionar	public	int id	void	Asigna id a IdItem
resetear	public		void	Método que se llama si se pulsa el botón de volver hacia atrás en el segundo menú, que añade al inventario cualquier objeto que hubiese en cada uno de las casillas de slots

FixedUpdate	private		void	Realiza un switch de IdItem. Si el valor es 0, se activa el primero de los menús, que permite seleccionar el objeto que se quiere poner a fabricar. Si no lo és, se activa el segundo de los menús, que coloca el objeto de lista con IdItem en la tercera casilla, y la primera y la segunda solo aceptarán los objetos previamente definidos que son necesarios para fabricar el objeto con identificador IdItem. Invoca fabricar y barra si hay materiales suficientes y no está lleno, comprobando los objetos de las casillas por seguridad, y los cancela en caso contrario, reseteando progreso. Vacía la primera casilla en caso de que su n sea 0 para evitar solapamientos de los materiales, y por último llama a actualizar

Figura 4.71: Funciones de la clase Fábrica

Figura 4.72: Fábrica creando un cable a partir de un lingote de cobre y una luz

4.5.3. Logística

El GameObject Suelo también tiene asociado el script Logística. Este script se ocupa de crear una vía cada vez que se llama desde HexRender, siendo una vía un objeto que recoge los prefabs Vía que se crean, conformando así un grupo por cada vez que se unen dos edificios. Esto facilita tanto eliminar la vía con el modo de destrucción, como acceder a ella si se quiere eliminar cuando se borra un edificio que la lleva asignada. Para colocar la vía, el script Logísitica realiza un algoritmo A* a fin de encontrar el camino más corto entre el edificio de origen y el de final.

Antes de ir con la propia clase Logistica, se tienen que explicar las dos clases que se crean y que son necesarias para llevar a cabo el algoritmo:

- La primera de ellas es la clase Nodo, que tiene los atributos públicos Vector2 posicion, int coste, int heurística y Nodo padre. El único método que contiene esta clase es el constructor, que asigna el valor pasado como parámetro a cada uno de sus atributos.
- La segunda de ellas es PriorityQueue<T>, que crea una lista de tuplas de un objeto de tipo T y un int, y que tiene varios métodos: int Count, que devuelve la cantidad de elementos de la lista; void Enqueue, que añade un elemento a la lista, recibiendo como parámetros el objeto en cuestión y la prioridad; y T Dequeue, que devuelve el elemento de la lista con el menor valor en su int y lo elimina de la lista.

Con esta información, ya se puede entrar de lleno en la clase Logística propiamente dicha.

4.5.3.1. Clase Logística

Nombre	Modificador	Tipo	Descripcion
inventario	public	GameObject	GameObject del inventario
edificado	public	GameObject	GameObject que se genera alinstanciar el prefab Via
viaPrefab	public	GameObject	Prefab de la via
cinta	public	Material	Material básico de la cinta
destruccion	public	Material	GameObject del inventario
posI	private	Vector2	Posicion en la matriz del edificio inicial de la cinta
posF	private	Vector2	Posición en la matriz del edificio final de la cinta
via	private	GameObject	GameObject que se asignará hijo del GameObject Logística (a su vez hijo del GameObject Suelo), y que contendrá cada una de los prefabs Via que lo conforman
nHijo	private	int	Posición del prefab Via en el array de hijos del contenedor via (cinta completa)

Figura 4.73: Atributos de la clase Logística

Nombre	Modificador	Parámetros	Tipo	Descripcion
Error	private	int n	IEnumerator	Asigna codigo en el inventario a n, espera dos segundos, y lo vuelve a asignar a 0
Coordenadas	private	Vector2Int pos1	Vector3	Obtiene las coordenadas tridimensionales de pos1, que es una posición en la matriz

Heuristica	private	Vector2Int posI, Vector2Int posF	int	Devuelve la distancia máxima que se tiene que recorrer, sea la diferencia absoluta de posI a posF en el eje de las x, o en el eje de las y
Construir	public		int	Llama a Pathfinding, que devuelve la lista camino. Manda los códigos de error (consultar punto 3.2.2.7.) haciendo una llamada al método Error, crea el objeto via, el cual se añade a la lista vías del edificio posF y se asigna como salida al edificio posI del inventario. Luego, llama al método direccionar, y devuelve la longitud del camino + 1
Pathfinding	private	Vector2 posI, Vector2 posF	List <Vector2>	Ver punto 4.5.3.2.
Direccionar	public	List <Vector2> camino	int	Crea un componente vía para cada una de las casillas que se obtienen en camino, conectando la primera con el edificio de posI, y la última con el de posF. Para calcular la dirección en la que se tiene que crear el prefab

Figura 4.74: Funciones de la clase Logística

4.5.3.2. Método Pathfinding

El método pathfinding es el encargado de implementar el algoritmo A*. Este es un algoritmo que encuentra el camino más corto entre dos puntos. Normalmente, estos algoritmos se realizan para matrices (es decir filas y columnas), por lo que cada casilla está en contacto con otras 4.

Sin embargo, aunque para este juego se ha utilizado una matriz de dos dimensiones para expresar de manera simple cómo se organizan las baldosas, a la hora de colocarlas en el espacio tridimensional, su forma hexagonal acabó suponiendo que cada una de las baldosas tuviera otras 6 en contacto, lo cual se solucionó gracias a la estructura matricial de la grid del GameObject Suelo.

El método utiliza una priorityQueue<Nodo>y un HashSet>Vector2>, la primera de ellos siendo utilizada para agregar la dirección posterior a la inmediantamente procesada con una nueva heurística y un coste que se suma al existente, y el segundo para agregar todas las direcciones posibles que puede tomar el algoritmo. Este método acaba devolviendo el camino que sigue la cinta de transporte al crearse.

La solución principal, e innovación sobre los algoritmos A* clásicos, es la diferenciación de la posición de la casilla en su fila respectiva. Tal y como se desarrolla la creación del suelo, las filas de la matriz van realizando una suerte de zig-zag. Esto causa que, para la misma posición en el eje x de la matriz, las baldosas se encuente a izquierda o derecha, dependiendo de si su posición en el eje y de la matriz es par o impar, siendo estas últimas aquellas que se colocan a la izquierda según la vista de la cámara. Por lo tanto, es un factor a tener en cuenta a la hora de calcular el vecino. Si se toma una dirección dada, se puede dividir en los tres siguientes grupos:

- El primero, compuesto por las direcciones 0 y 3 (derecha e izquierda, según la vista de la cámara), siempre añaden el mismo vector, (1,0) y (-1,0), respectivamente.
- El segundo de estos grupos es el compuesto por las direcciones 1 y 5 (arriba-derecha y abajo-derecha), que añaden los vectores (0,-1) y (0,1) en caso de encontrarse a la izquierda, y añaden (1,-1) y (1,1) en caso contrario.
- El último de estos grupos es el compuesto por las direcciones 2 y 4 (arriba-izquierda y abajo-izquierda), que añaden los vectores (-1,-1) y (-1,1) en caso de encontrarse a la izquierda, y añaden (0,-1) y (0,1) en caso contrario.

Este método devuelve el orden en el que se construirán las vías, sin incluir aquella que existe en la posI (edificio de comienzo de la via), que se agregará de manera adicional en el método Construir.

4.5.3.3. Clase Via

Esta clase va asociada al prefab Via, que conforma cada fragmento de la cinta de transporte que se crea con la clase Logística.

Nombre	Modificador	Tipo	Descripcion
descargando	private	bool	Se están moviendo objetos a un edificio
rad1	public	float	Radio de la parte central de la cinta
rad2	public	float	Radio total de la cinta
alt	public	float	Altura donde ser coloca la cinta
material	public	Material	Material de la cinta
destruyendo	public	boool	El edificio se está destruyendo
inventario	public	GameObject	GameObject del inventario
movil	public	GameObject	GameObject de la casilla movil
mesh	private	Mesh	Malla que conforma la cinta
mfilter	private	MeshFilter	Componente que accede al mesh
mrenderer	private	MeshRenderer	Componente que renderiza el mesh
mcaras	private	List<Cara>	Lista de caras que conforman el mesh
pos	public	Vector2	Posición de la cinta en la matriz
nHijo	public	int	Posición de la via dentro de su objeto padre
pos	public	Vector2	Posición de la cinta en la matriz
transportando	private	bool	Se mueven objetos entre las cintas adyacentes
destruccion	public	Material	Material de la cinta cuando se está destruyendo
slotE	public	List<Slot>	Lista de las casillas del edificio al que está asociada esta vía, en caso de estarlo
limite	private	int	Limite de cada via, fijado en 1

Figura 4.75: Atributos de la clase Via

Nombre	Modificador	Parámetros	Tipo	Descripcion
Awake	private		void	Asigna mfilter, mrenderer e inicializa mesh. Asigna el mesh de mfilter al nuevo mesh
DrawMesh	public	int dir1, int dir2	void	Asigna material a mrenderer, y luego llama a DrawFaces con dir1 y dir2, y luego a CombineFaces
DrawFaces	private	int dir1, int dir2	void	Crea una cara en dir1, y si dir1 es diferente de dir2, crea otra Cara en dir2. Después, crea una cara completa con la mitad de radio que las anteriores, llamando a NuevaCara 6 veces.
Combine-Faces	private		void	Combina las caras creadas con DrawFaces utilizando mcaras para obtener sus vértices, ángulos y triángulos, para añadir estas tres categorias al mesh. Luego congela la rotación y la traslación del objeto usando su componente Rigidbody, crea un MeshCollider con la forma del mesh creado, y se lo añade
NuevaCara	private	float rad1, float rad2, float alt, float alt1, int punto, bool inv	Cara	Crea cuatro vectores usando Punto, con los que crea los vértices de una Cara, que se devuelve tras componerla

Punto	protected	float tam, float alt, int ind	Vector3	Crea un punto usando la posición relativa al centro de la cara mediante ind, multiplicando tam con el coseno y el seno del ángulo obtenido con ind, y colocandolo en altura alt
FixedUpdate	private		void	Si se está destruyendo, asigna el material de MeshRenderer a destrucción, y a material en caso contrario. Después, si es la primera vía de la cinta, invoca a transportar si la casilla de salida del edificio está llena, y lo cancela en caso contrario; si es una vía intermedia, invoca transportar si la vía anterior está llena, y lo cancela en caso contrario; si es la última vía de la cinta de transporte, invoca transportar si la via anterior está llena, y lo cancela en caso contrario, y también invoca descargar hasta que se vacíe su casilla
transportar	private		void	Elimina una unidad de la via o el edificio anterior, y la añade a la actual

descargar	private		void	Deposita el contenido de la via en una de las casillas de entrada del edificio de destino, priorizando la que ya contenga ese objeto

Figura 4.76: Funciones de la clase Via

Figura 4.77: Cinta de transporte compuesta de 4 vias

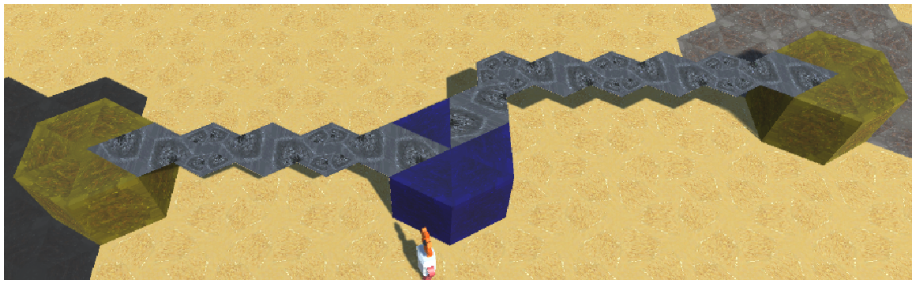


Figura 4.78: Cintas de transporte que van desde dos minas hasta un horno

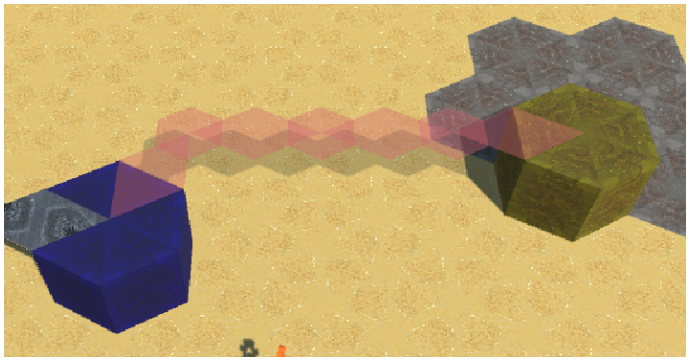


Figura 4.79: Cinta de transporte marcada para destruirse

4.6. Menú de inicio

Cuando se empieza a jugar a "Otomatïkï", se encuentra un menú muy sencillo, en el que tras ingresar el nombre y pulsar el botón de jugar, se abren dos opciones: jugar con tutorial o jugar sin él. Independientemente de la decisión que se tome, se va a cargar la escena principal del juego, pero con una particularidad: si se eligió jugar con tutorial, se podrá ver en la esquina superior izquierda de la pantalla, y si no la persona que esté jugando en ese momento empezará a jugar sin ningún tipo de ayuda, aunque siempre podrá volver a consultar el tutorial.

Figura 4.80: Menú inicial del juego

Figura 4.81: Elección de jugar con tutorial o no hacerlo

Capítulo 5

Evaluación y resultados

En este capítulo se hablará sobre el proceso de evaluación que se ha seguido en las pruebas de usabilidad, MTQ y test de Bartle que se han realizado.

5.1. Características de la evaluación

La evaluación del producto desarrollado se llevó a cabo junto con dos alumnos más del Trabajo de Fin de Grado. Ellos también desarrollaron un videojuego utilizando la plataforma Unity, y los tres seguimos el mismo proceso de evaluación. El objetivo de este proceso, era comprobar si nuestros juegos en conjunto podían servir para obtener los rasgos motivacionales (dados por el MTQ, o Motivational Trait Questionnaire, para el cual se definieron una serie de acciones que daban una puntuación a cada uno de los grupos motivacionales), y los perfiles de jugador, otorgados por el Test de Bartle (para el cual se utilizaron ciertas acciones diseñadas para el MTQ, pero a las cuales se les asignaron unos pesos diferentes a la hora de agregarlos para obtener la puntuación del test de Bartle).

Para la recolección de los resultados, se utilizó Google Forms para recoger los datos demográficos de cada uno de los nueve participantes, y sus respuestas al Test de Bartle y el MTQ propiamente dichos. También se realizó un cuestionario de impresiones generales específico de "Otomatïkï", que recogió la experiencia general que los usuarios obtuvieron tras probar el juego. Este último cuestionario también tenía preguntas relativas al conocimiento previo sobre los productos de este tipo.

Cabe mencionar que en total este proceso de evaluación se realizó teniendo en cuenta a nueve usuarios de diferentes perfiles, a fin de obtener un resultado amplio y diverso. A cada uno de estos usuarios se le asignó un identificador único que utilizaron a lo largo de todo el test, siendo así más sencillo recopilar la información de los diferentes formularios y analizarla para cada uno de los usuarios. Esta prueba duró 15 minutos para cada usuario, por lo que "Otomatïkï" no se pudo mostrar en su totalidad a ninguno de los jugadores, al ser la extensión del juego mucho más larga que el tiempo que duró la prueba.

5.2. Datos demográficos de los usuarios

A la hora de pedir los datos demográficos, se barajó pedir más datos como el género, pero se acabó descartando por no tener ningún tipo de relación con el desempeño a la hora de jugar, y se acabaron preguntando factores que sí que influencian el desempeño que puede llegar a tener un jugador. Concretamente, este test estaba compuesto de las siguientes preguntas:

- Edad
- ¿Con qué frecuencia juegas videojuegos?
- ¿Cuál es tu plataforma preferida para jugar?
- ¿Qué juegos sueles jugar más?
- ¿Qué genero de videojuegos sueles jugar?

El primero de los items consultados es la edad, que se recogió entre 19 y 26 años. Como el intervalo es pequeño, no se va a tener en cuenta a la hora de evaluar el desempeño de los usuarios, ya que todos son personas familiarizadas con la tecnología, que juegan ocasionalmente o frecuentemente, tal y como se recoge en la segunda pregunta de este test. Cuando se evalúe el desempeño y la experiencia de cada usuario por separado, si que se tendrá en cuenta tanto la frecuencia a la que el usuario en concreto juega, como los siguientes items, que se presentan en los siguientes gráficos:

Figura 5.1: Plataforma en la que juegan los usuarios

Figura 5.2: Tipo de juego al que suelen jugar los usuarios

Figura 5.3: Género de juego al que suelen jugar los usuarios

Como se puede observar en las imágenes anteriores, la respuesta de los diferentes usuarios es muy variada. Para el caso concreto de "Otomatïkï", se tendrá en cuenta para analizar el rendimiento de los jugadores, y del juego interactuando con ellos, si acostumbran a jugar en individual y en la plataforma PC. A la hora de analizar el género al que suelen jugar, al tener "Otomatïkï" componentes de juegos de estrategia y aventuras, se tendrán un poco en cuenta, pero en menor medida que los puntos anteriores, ya que no hay ningún usuario que acostumbre a jugar el género de simulación.

5.3. Impresiones generales de cada usuario

A la hora de evaluar las impresiones de cada usuario, se preguntó lo siguiente:

1. ¿Has jugado a algún juego similar previamente?
2. ¿A qué juegos te recuerda que hayas probado?
3. ¿Qué te ha gustado del juego?
4. ¿Qué mejorarías del juego?
5. Rendimiento general del usuario (métrica registrada por el facilitador).

A continuación se exponen las respuestas de cada usuario a estas preguntas, referenciandose con el número de la pregunta que se ve en la lista anterior:

5.3.1. Usuario 1

1. No ha jugado a nada similar.
2. Le recuerda al Minecraft.
3. Atractivo visualmente.
4. Añadir los costes de fabricación a las tecnologías investigadas en el árbol, mejorar la comprensión del tutorial, y que aparezca cuando se prueba una nueva mecánica.
5. Completa 0 logros, y no investiga muchas tecnologías.

5.3.2. Usuario 2

1. No ha jugado a nada similar.
2. Le recuerda al Minecraft, y al LoL por el movimiento.
3. Atractivo visualmente, se siente bien el movimiento, mecánicas complejas y que interactúan bien entre ellas.
4. Mina le recuerda a bomba, añadir un mapa, mover entre casillas con la rueda del ratón, y poder minar mientras se camina.
5. Completa 1 logro, e investiga parte del árbol de construcción. Coloca una mina. Acaba el juego en 12 minutos.

5.3.3. Usuario 3

1. Conoce el concepto de los juegos como el factorio.
2. Le recuerda al Minecraft.
3. Entretiene, da ganas de seguir jugando.
4. Costes de la fabricación en el árbol de las tecnologías desbloqueadas, poder volver hacia atrás en el tutorial.

5. Completa 2 logros, tarda 12 minutos, cuesta un poco al principio pero acelera cuando aprende.

5.3.4. Usuario 4

1. No ha jugado a nada similar, ni juega en general.
2. Le recuerda al Minecraft.
3. Le ha gustado mucho aunque le haya costado, visualmente atractivo el personaje. Lo considera intuitivo aunque le haya costado un poco.
4. Curva de aprendizaje alta para alguien que no juegue, los colores pueden confundir.
5. Completa 0 logros, y no investiga muchas tecnologías. Le cuesta aprender pero lo acaba pillando.

5.3.5. Usuario 5

1. No ha jugado a nada similar.
2. Le recuerda al Minecraft.
3. Muy bien en general. Intuitivo y facil de usar, le gusta probar todas las mecánicas, que le parecen cmpletas y que se complementan bien.
4. Minar con click del ratón le parece más intuitivo, aunque en general lo considera intuitivo y facil de usar. Mina le recuerda a una bomba, y poner el precio de lo investigado en el árbol de tecnologías.
5. Tarda 13 minutos en completar el juego, completa 1 logro.

5.3.6. Usuario 6

1. No ha jugado a nada similar, pero ha jugado aventura y exploración.
2. Le recuerda al Minecraft.
3. Atractivo visualmente, y le ha gustado el género pero es complicado empezar sin ayuda. Le sorprende que no haya encontrado ningún bug.
4. Le confunden un poco los colores de los objetos, porque los sprites son parecidos. La curva de aprendizaje es alta si no tiene ayuda para empezar, al no conocer el género.
5. Completa 0 logros, pero casi completa el objetivo de la Vertical Slice.

5.3.7. Usuario 7

1. No ha jugado a nada similar, pero conocía el género.
2. Le recuerda al Minecraft.

3. Atractivo visualmente, le gusta que parezca un juego completo, y le gusta la estética low poly, le parece que está bien aprovechada.
4. Mantener el ciclk para caminar, mejorar el tutorial.
5. Acaba en 11 minutos, completa 3 logros.

5.3.8. Usuario 8

1. No ha jugado a nada similar.
2. Le recuerda al Minecraft.
3. Atractivo visualmente, tutorial suficiente para saber jugar, le ha gustado en general.
4. Le gustaría que los logros dieran recompensas, y que existieran decoraciones o elementos estéticos.
5. Completa 3 logros, cumple el objetivo en 8 minutos.

5.3.9. Usuario 9

1. No ha jugado a nada similar, conocía el género.
2. Le recuerda al Hay Day, por la obtención de recursos.
3. Le ha gustado en general, le parece que está muy completo y que las interfaces son intuitivas. También le parece bien planteada la curva de aprendizaje.
4. Cambiaría la tecla para abrir el inventario de la E a la I.
5. Completa 0 logros, y no investiga muchas tecnologías.

Estas son todas las respuestas que se han recogido, y de las obtenido las siguientes conclusiones:

- El juego es atractivo, sobre todo el personaje, y la estética ha sido un acierto en general, al ser la low poly fácil de desarrollar y agradable en un entorno que encaje con ella, como el mundo formado por hexágonos que se encuentra en "Otomatïkï".
- La curva de aprendizaje del juego ha sido relativamente buena, ya que al final de los 15 minutos todos los usuarios han entendido las mécanicas de obtención de recursos, fabricación e investigación, y algunos incluso empezaron con la construcción.
- Como punto a mejorar que se ha repetido varias veces, sería el incluir los costes de fabricación de un objeto en el árbol de tecnologías tras desbloquearlo.

Por lo demás, no se ha podido encontrar relación entre las personas que suelen jugar en PC y el rendimiento en la prueba, ni con el tipo de juego que acostumbran a jugar (individual, multijugador cooperativo y multijugador competitivo).

Como ninguno de los usuarios que participaron en la prueba había probado un juego de este género, no se puede determinar con seguridad si una persona con experiencia habría tardado menos tiempo en entender las mecánicas del juego.

5.4. MTQ y Test de Bartle

Esta parte de la evaluación se dividió en dos partes: la primera de ellas, la captura de unas métricas mientras el jugador estaba probando la Vertical Slice, asignadas a unas acciones concretas, y la segunda de ellas la realización de los tests de Bartle y el MTQ por parte de los usuarios.

Para la primera parte, se definieron las siguientes acciones, con los siguientes pesos dependiendo del test:

Acción que registra una métrica	**MTQ**	**Bartle**
Sigue el tutorial del juego	Determination 30 %	
El jugador persevera por terminar el juego	Determination 20 %	Achiever 18 %
Consigue logros	Mastery Goal 40 %	Achiever 18 %
Investiga todas las tecnologías existentes	Mastery Goal 30 %	Achiever 18 %
Consulta el tutorial de nuevo	Desire to learn 20 %	
Conecta dos minas con un horno	Desire to learn 20 %	Achiever 18 %
Vuelve a consultar el tutorial del juego	Failure Avoidance 30 %	
El jugador desiste del juego	Failure Avoidance 20 %	

Figura 5.4: Acciones del MTQ y el test de Bartle

Para capturar estas acciones, se han utilizado 5 scripts que fueron facilitados por el tutor, los cuales se modificaron para poder capturar las diferentes métricas que se tenían. A continuación se describe el papel de cada script:

- MainManager: Script que contiene el DontDestroyOnLoad, lo cual permite que no se elimine en el cambio de escena del menú al juego.
- Persistence: Crea la carpeta donde se guardaran los perfiles en caso de que no exista.
- MotivationalProfile: crea el .txt donde se almacenarán las métricas y las escribe en él.

- MTQ82: crea un diccionario con todos las métricas que se van a capturar, y aumenta el peso de estas con la funcion capture.
- MTQController: se asocia a un objeto, que hace una llamada al método del tipo de métrica correspondiente, mandando un int del valor correspondiente. Originalmente solo registraba las del MTQ, y se añadió un método por cada perfil de jugador del test de Bartle, y se eliminaron los de OtherReferencedGoals y CompetitiveAction, que son partes del MTQ que no reúne ninguno de nuestros juegos.

Utilizando estos scripts se recogieron métricas mientras los usuarios probaban la Vertical Slice. Después se compararon con los resultados de cada uno de ellos en los tests de Bartle y el MTQ propiamente dichos.

5.4.1. Motivational Trait Questionnaire

Para los resultados del MTQ, de las 82 preguntas que los usuarios contestaron en el test, se eligieron 31 de ellas, que eran las que más podrían llegar a medirse con nuestros juegos:

- Determination: Cuando comienzo un proyecto estoy decidido a llevarlo a cabo hasta el final.
- Determination: A menudo continúo trabajando en una tarea después de que todos los demás hayan abandonado.
- Determination: Dejo de trabajar en tareas complicadas cuando me quedo atascado.
- Determination: A menudo me doy por vencido en la mitad de un proyecto.
- Determination: Sigo trabajando en tareas difíciles a pesar de las frustraciones con las que me encuentro.
- Determination: Soy persistente en mis intentos de superar cualquier dificultad mientras se trabaja en un proyecto.
- Determination: Me pongo niveles altos de calidad y trabajo para alcanzarlos.
- Mastery Goals: Cuando estoy trabajando en algo tiendo a establecer objetivos para mejorar.
- Mastery Goals: Intento encontrar caminos para hacer mi trabajo mejor.
- Mastery Goals: Si ya hago algo bien no veo la necesidad de desafiarme para hacerlo mejor.
- Mastery Goals: Compito conmigo mismo retándome para hacer las cosas mejor que las he hecho antes.
- Mastery Goals: Me establezco objetivos como una forma para mejorar mi rendimiento.

- Mastery Goals: El nivel de calidad de mi trabajo establecido por mí a menudo excede los requeridos para la finalización con éxito de un proyecto.
- Mastery Goals: Cuando aprendo algo nuevo me centro en mejorar mi rendimiento.
- Desire to learn: A veces me preparo una clase solo por poder aprender más.
- Desire to learn: Me gusta tomar clases que me desafíen.
- Desire to learn: Hago preguntas para ayudarme a aprender mejor.
- Desire to learn: Incluso cuando he estudiado lo suficientemente duro para conseguir una buena calificación, estudio más porque quiero entender completamente el material.
- Desire to learn: Estoy motivado a aprender de forma natural.
- Desire to learn: Debido a que quiero conocer más materia, hago más trabajo para la clase del que me piden.
- Desire to learn: Cuando estoy aprendiendo algo nuevo trato de entenderlo por completo.
- Desire to learn: Es importante para mí aprender tanto como sea posible.
- Failure Avoidance: La probabilidad de fallo no me impide hacer las cosas.
- Failure Avoidance: Estoy preocupado acerca de la posibilidad de fallar.
- Failure Avoidance: Intento evitar situaciones en las que puedo experimentar el fracaso.
- Failure Avoidance: Me cuesta dormir cuando estoy preocupado pensando en un posible fracaso.
- Failure Avoidance: Evito situaciones en las que podría terminar pareciendo poco inteligente.
- Failure Avoidance: Prefiero dejar pasar una buena oportunidad que arriesgarme a fracasar o a ser rechazado.
- Failure Avoidance: La posibilidad de que no lo vaya a hacer bien me impide asistir a clases difíciles.
- Failure Avoidance: Dirijo mis esfuerzos a evitar situaciones en las que puedo fallar.
- Failure Avoidance: Preferiría dejar pasar una buena oportunidad que arriesgarme a fracasar o a ser rechazado.

Con estas preguntas en mente, vemos a continuación la siguiente tabla, donde la primera columna es el ID del jugador, y luego los resultados del MTQ tanto para el test, como para las métricas capturadas por los scripts y sumadas a las obtenidas por los otros dos alumnos.

ID usuario	Determination	Mastery Goals	Desire to learn	Failure avoidance
1	Test 97 % Scripts 55 %	Test 83 % Scripts 0 %	Test 72 % Scripts 15 %	Test 85 % Scripts 5 %
2	Test 78 % Scripts 65 %	Test 85 % Scripts 0 %	Test 66 % Scripts 45 %	Test 50 % Scripts 35 %
3	Test 64 % Scripts 60 %	Test 66 % Scripts 50 %	Test 68 % Scripts 40 %	Test 59 % Scripts 65 %
4	Test 50 % Scripts 65 %	Test 42 % Scripts 20 %	Test 56 % Scripts 40 %	Test 40 % Scripts 5 %
5	Test 52 % Scripts 75 %	Test 64 % Scripts 0 %	Test 52 % Scripts 45 %	Test 53 % Scripts 15 %
6	Test 76 % Scripts 55 %	Test 66 % Scripts 0 %	Test 45 % Scripts 55 %	Test 50 % Scripts 0 %
7	Test 78 % Scripts 45 %	Test 66 % Scripts 50 %	Test 50 % Scripts 25 %	Test 18 % Scripts 0 %
8	Test 90 % Scripts 70 %	Test 83 % Scripts 70 %	Test 39 % Scripts 15 %	Test 18 % Scripts 0 %
9	Test 80 % Scripts 85 %	Test 71 % Scripts 30 %	Test 72 % Scripts 40 %	Test 57 % Scripts 5 %

Figura 5.5: Resultados MTQ

Al analizar esta tabla, podemos comprobar que no existe una relación apreciable entre los resultados de los tests y lo obtenido mediante el registro de las métricas durante la prueba. Sin embargo, si que se puede apreciar una tendencia en algunos casos, donde el valor más alto que se obtiene entre todos los de test, coincide con el rasgo motivacional más alto obtenido mediante los scripts. Esto lleva a la conclusión de que con más métricas y más tiempo de prueba, podrían obtenerse valores similares.

5.4.2. Test de Bartle

Al igual que con el MTQ, para el Test de Bartle se han registrado las métricas utilizando los scripts.Sin embargo, al resultar el test propiamente dicho en un porcentaje numérico de cada perfil de jugador, no se han despreciado preguntas como en el MTQ. Los resultados obtenidos al cruzar el test con los scripts (tras agregarle lo obtenido por los otros dos alumno) son:

ID usuario	Killer	Achiever	Socializer	Explorer
1	Test 0 % Scripts 0 %	Test 73 % Scripts 0 %	Test 60 % Scripts 100 %	Test 67 % Scripts 17,5 %
2	Test 87 % Scripts 0 %	Test 33 % Scripts 18 %	Test 33 % Scripts 100 %	Test 47 % Scripts 65 %

3	Test 33 % Scripts 0 %	Test 20 % Scripts 18 %	Test 60 % Scripts 100 %	Test 87 % Scripts 35 %
4	Test 20 % Scripts 67 %	Test 47 % Scripts 24 %	Test 67 % Scripts 100 %	Test 67 % Scripts 65 %
5	Test 0 % Scripts 9 %	Test 53 % Scripts 18 %	Test 73 % Scripts 100 %	Test 73 % Scripts 52,5 %
6	Test 20 % Scripts 0 %	Test 40 % Scripts 0 %	Test 67 % Scripts 100 %	Test 73 % Scripts 100 %
7	Test 47 % Scripts 0 %	Test 27 % Scripts 18 %	Test 67 % Scripts 100 %	Test 60 % Scripts 100 %
8	Test 73 % Scripts 78 %	Test 20 % Scripts 46 %	Test 40 % Scripts 100 %	Test 67 % Scripts 17,5 %
9	Test 40 % Scripts 100 %	Test 53 % Scripts 54 %	Test 40 % Scripts 0 %	Test 67 % Scripts 17,5 %

Figura 5.6: Resultados test de Bartle

De la misma manera que sucedió con el MTQ, al analizar esta tabla, podemos comprobar que no existe una relación apreciable entre los resultados de los tests y lo obtenido mediante el registro de las métricas durante la prueba. La principal diferencia con el caso del MTQ, es que en este caso, es todavía menos apreciable la relación, probablemente debido a que hay menos items que se evalúen, por lo que es mucho menos preciso que el MTQ en ese sentido.

Capítulo 6

Conclusiones y líneas futuras

Para empezar la conclusiones de este trabajo, primero se van a tratar con las del desarrollo.

En la fase de planificación, se decidió utilizar un método de desarrollo en cascada, que probó ser muy efectivo, ya que la integración de una mecánica con la posterior fue mucho más fácil al estar la primera acabada completamente, que si hubiera sido al contrario.

Cabe destacar también que el utilizar una herramienta conocida como Unity facilitó mucho todo el proceso inicial, ya que se pudo empezar a desarrollar "Otomatïkï" justo cuando se completó el diseño. Aunque ciertas partes de este diseño fueron cambiando para adaptarse a las necesidades que iban surgiendo durante el desarrollo, el núcleo del juego se mantuvo bastante estable, lo cual es señal de un diseño consistente.

Por otro lado, hay que extraer las conclusiones externas al desarrollo, que se obtienen analizando los datos recogidos durante la evaluación con los usuarios, que se puede ver en el Capítulo 5.

La primera de ellas, es que "Otomatïkï" es producto que, aun en una fase de Vertical Slice, incorpora varias mecánicas bien integradas entre ellas, lo cual genera una experiencia de juego inmersiva, pulida, clara, atractiva y amena.

El diseño que se ha elegido, con una estética low poly en un mundo completamente conformado por hexágonos, hace que sea atractivo visualmente, y ha gustado la libertad de moverse y poder hacer diferentes cosas por el mapa, aunque hubiera un objetivo a cumplir. A muchos de los usuarios les ha recordado a Minecraft, tanto por la barra del inventario, la capacidad de encontrar recursos, como la calma que transmite jugar a algo basado en la recolección y la fabricación.

También se han recibido aspectos negativos, no como queja, pero si como vías de mejora, como por ejemplo la existencia de un mapa, o el poner los recursos necesarios para fabricar un objeto en el árbol de tecnologías cuando ese objeto ya ha sido desbloqueado.

Aunque hubo alguna queja con la curva de dificultad, esto es algo típico de los juegos de gestión, y como todos los usuarios se acabaron habituando al juego en menos de 10 minutos, considero que no es un aspecto a mejorar ni a tener en cuenta.

Una parte muy importante de la evaluación con los usuarios fue la de comprobar, junto con otros dos compañeros, si un conjunto de juegos de géneros diferentes podría servir para hacer las veces de un test de Bartle o un MTQ. Desde mi punto de vista, creo que no se ha conseguido analizar de manera correcta ni los rasgos motivacionales, ni los perfiles de jugador, aunque no porque no sea posible, ya que si que se ha podido apreciar cierta tendencia en el caso del MTQ de algunos usuarios.

Si hubiese habido más tiempo de demo para probar "Otomatïkï", se habrían podido extraer datos de todas las acciones colocadas, ya que por ejemplo las mecánicas de construcción y logística fueron lo que menos probaron los usuarios por culpa del tiempo. También creo que la cantidad de acciones que planteamos para el MTQ y sobre todo para el test de Bartle, fue insuficiente, porque había muchos de los aspectos que se tratan en los cuestionarios siendo imposibles de medir.

Sí que creo, sin embargo, que este no es solo un problema de tiempo o de cantidad de acciones de las que se extraen métricas, si no que también si se quiere hacer una prueba como esta que de verdad funcione, habría que construir los juegos alrededor de los tests, y no al revés. Me parece positivo que haya más de un juego, y que cada uno de ellos se centre en unos aspectos específicos, pero los productos que habíamos desarrollado los otros dos alumnos y yo distaban mucho de ser útiles a este respecto.

Es necesario tener en cuenta todos esos factores para explicar por qué el experimento no ha sido satisfactorio en el sentido de poder evaluar lo mismo que el MTQ o el test de Bartle con nuestros juegos, aunque sí que ha sido satisfactorio probar, tanto que "Otomatïkï" funcionaba y era atractivo para las personas que lo probaron, como que con unos juegos orientados y construidos alrededor de los tests, el experimento habría sido un éxito.

Como líneas futuras, lo primero sería añadir los materiales necesarios para fabricar cada recurso al árbol de tecnologías, que ha sido lo más repetido por los usuarios, así como implementar funciones de accesibilidad y sonido al juego, ya que no fue posible añadirlos debido a los plazos. Otra mejora que considero que sería positiva sería un mapa que ubicara los depósitos de recursos y la posición del jugador, haciendo mucho más fácil la orientación en el entorno.

Como he mencionado varias veces a lo largo de este Trabajo, "Otomatïkï" está pensado ya para que sea muy fácil de incrementar, por lo que otro punto de las líneas futuras sería agregar más objetos, tecnologías, edificios, y mejorar el tutorial y los materiales de los que se componen las baldosas del mundo, para darle un pulido visual de mayor calidad.

Bibliografía

[1] Wikipedia. «Proceso para el desarrollo del software». (En línea), dirección: https://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software (visitado 14-04-2024).

[2] E. Z. y K. Salen, *Rules of Play: Game Design Fundamentals*. MIT Press, 2013.

[3] P. M. Institute. «What is Project Management?» (En línea), dirección: https://www.pmi.org/about/what-is-project-management (visitado 16-04-2024).

[4] P. Pro. «Metodologías Ágiles vs Tradicionales». (En línea), dirección: https://www.positivo.pro/blog/metodologias-agiles-vs-tradicionales/ (visitado 18-04-2024).

[5] R.Tokio. «¿Qué es necesario poner en un GDD?» (En línea), dirección: https://www.tokioschool.com/noticias/que-es-necesario-poner-en-un-gdd/ (visitado 19-04-2024).

[6] Wikipedia. «Bartle taxonomy of player types». (En línea), dirección: https://en.wikipedia.org/wiki/Bartle_taxonomy_of_player_types (visitado 27-05-2024).

[7] Wikipedia. «Unity (motor de videojuego)». (En línea), dirección: https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego) (visitado 20-04-2024).

[8] Wikipedia. «MonoDevelop». (En línea), dirección: https://es.wikipedia.org/wiki/MonoDevelop (visitado 20-04-2024).

[9] Wikipedia. «Blender». (En línea), dirección: https://es.wikipedia.org/wiki/Blender (visitado 20-04-2024).

[10] Wikipedia. «Trello». (En línea), dirección: https://es.wikipedia.org/wiki/Trello (visitado 20-04-2024).

[11] PEGI. «¿Qué son las clasificaciones?» (En línea), dirección: https://pegi.info/es/node/19 (visitado 21-05-2024).

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Mon Jun 03 23:24:42 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)