UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS DE GANDIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Diseño e implementación del prototipo de un nivel de videojuego "Real Time Strategy" (RTS)

Trabajo Fin de Grado

Grado en Tecnologías Interactivas

AUTOR/A: Miret Conejero, Pablo Jesús

Tutor/a: Lozano Quilis, José Antonio

CURSO ACADÉMICO: 2024/2025

Resumen

El propósito de este trabajo final de grado es llevar a cabo el diseño e implementación del prototipo de un nivel de videojuego *Real Time Strategy* (RTS). Este se desarrolla para el ordenador como plataforma objetivo, mediante el motor de desarrollo Unity con el cual se programa mediante el lenguaje C#.

El prototipo busca incorporar un estudio de las mecánicas básicas esperadas de un juego de este tipo, así como aportar propuestas para su implementación. Estas incluyen una escena donde el usuario puede interactuar, mediante unas estructuras que puede situar sobre el tablero y unas unidades que reciben instrucciones del usuario.

De la misma manera, se incorporarán sistemas adicionales que regulen las formas de interacción. Estos sistemas incluirán un mecanismo de recursos y otro que establezca un objetivo, obligando al usuario a utilizar de manera eficiente los medios a su disposición. Además, se contará con una versión simplificada de un enemigo simulado, que proporcionará al usuario un objetivo de victoria.

No obstante, el alcance del proyecto se limita al desarrollo de un prototipo de un nivel de juego. Los elementos utilizados representan solo una pequeña parte de lo que podría lograrse en un proyecto más ambicioso. Esto implica que aspectos como el artístico serán suplidos con recursos gratuitos o *assets* sencillos de creación propia. Adicionalmente, se pretende que el rival simulado interactúe únicamente con un conjunto limitado de mecánicas y realice acciones de baja complejidad.

Palabras Clave:

Videojuego, Unity, RTS, estrategia, programación.

Abstract

The purpose of this final degree project is to design and implement a prototype of a Real Time Strategy (RTS) video game level, using a computer as the target platform, through the Unity development engine with C# as the programming language.

The prototype aims to incorporate a study of the basic mechanics expected in this type of game, as well as to provide proposals for their implementation. These include a scene where the user can interact through structures that can be placed on the board and units that receive instructions from the user.

Additionally, supplementary systems will be included to regulate the forms of interaction. These systems will consist of a resource mechanism and another that establishes a goal, compelling the user to use the available means efficiently. Furthermore, a simplified version of a simulated enemy will be included, providing the user with a victory objective.

However, the scope of the project is limited to developing a prototype of a game level. The elements used represent only a small part of what could be achieved in a more ambitious project. This implies that artistic aspects will be supplemented with free resources or simple assets created by the user. Additionally, the simulated opponent is intended to interact only with a limited set of mechanics and perform low complexity actions.

Key words:

Videogame, Unity, RTS, Strategy, programming.

Índice

Contenido

Resumen ... 0

Abstract ... 1

Índice ... 2

Índice de imágenes ... 4

Índice de tablas ... 5

Índice de ecuaciones ... 5

1 Introducción ... 6

1.1 Presentación ... 6

1.2 Objetivos ... 6

1.3 Metodología ... 7

1.4 Etapas ... 7

1.5 Desafíos ... 8

1.6 Planificación previa ... 8

2 Implementaciones comerciales existentes ... 9

2.1 Contextualización ... 9

2.2 Casos de ejemplo ... 9

3 Idea para implementar ... 13

4 Diseño ... 14

4.1 Introducción ... 14

4.2 Metodología ... 14

4.3 Herramientas ... 14

4.4 Interacción con el medio ... 15

4.4.1 Cámara y controles ... 15

4.4.2 Interfaces de usuario ... 15

4.5 Rasgos y estadísticas de las entidades controlables ... 15

4.6 Usos de las utilidades NavMesh en el desplazamiento ... 18

4.7 Edificios ... 18

4.7.1 Base ... 19

4.8 Construcción ... 19

4.8.1 Zonas de control ... 19

4.8.2 Cuadrícula de construcción y terreno especial ... 20

4.8.3 Comprobaciones de construcción ... 20

4.8.4 Selección de edificios en interfaz de usuario ... 21

4.9 Sistema de recursos 21
4.10 Sistema de facciones 22
4.11 Representación alternativa del sistema de facciones 23
4.12 Lógica de funcionamiento de las unidades 24
4.13 Lógica del funcionamiento de los adversarios 25
4.13.1 Introducción 25
4.13.2 Opción estudiada 1. Modelo basado en Árbol de decisión 25
4.13.3 Opción estudiada 2. Modelo basado en máquinas de estados finitas 26
4.13.4 Opción elegida. Modelo basado en utilidad. 26
5 Implementación 30
5.1 Asignaciones iniciales 30
5.2 Sistema de Facciones 31
5.2.1 Script de gestión de facciones. Faction handler 31
5.3 Edificios no capturables 32
5.4 Bases y sistema de captura 34
5.5 Comportamientos generales de las unidades 36
5.5.1 Ejemplo de comportamientos específicos de unidad 38
5.6 Economía 38
5.7 Sistema de utilidad en la toma de decisiones de los jugadores no humanos 41
5.7.1 Aplicación a una decisión especifica 41
6 Debug 44
7 Conclusiones 46
8 Bibliografía 47
9 Anexo: Relación del trabajo propuesto con los ODS 49

Índice de imágenes

Figura 1 Tablero de distribución de tareas compatible con metodología Scrum 7
Figura 2 Producto referente en el mercado RTS con nombre comercial: Age of Empires II: Definitive Edition. 10
Figura 3 Información sobre la cantidad de usuarios activos del videojuego Age of Empires II en la plataforma de venta de videojuegos Steam. 10
Figura 4 Producto referente en el mercado RTS con nombre comercial: Starcraft II. . 11
Figura 5 Producto comentado como referente estético deseado, con nombre comercial: Supreme Commander 2. 12
Figura 6 Representación visual de la librería Navmesh y sus elementos básicos. 18
Figura 7 Representación visual de la división del mapa por zonas, divididas entre los jugadores, los cuales se identifican por diferencias de color. 19
Figura 8 Diagrama que representa las condiciones que permiten o deniegan realizar la construcción del edificio 20
Figura 9 Diseño que representa la interfaz de usuario para la selección de edificios, situada en la parte inferior de la pantalla 21
Figura 10 Diseño que muestra la interfaz de usuario para la representación de recursos del jugador, situada en la parte superior de la pantalla 22
Figura 11 Representación del flujo de sucesos esperado en la conquista de una base de un jugador por parte de otro jugador 22
Figura 12 Paleta de colores recomendada para mejorar la accesibilidad a perfiles de vision daltonicos 23
Figura 13 Diagrama de la máquina de estados básica que usan las unidades genéricas 24
Figura 14 Ejemplo de Intento de resolución del problema de toma de decisión, planteándolo en formato de diagrama de árbol. 25
Figura 15 Gráfica que representa la probabilidad de ejecución de los casos especificados 29
Figura 16 Asignación de las unidades iniciales en el script dedicado a configurar la partida 30
Figura 17 Asignación de color en una unidad, visible por el cambio en las texturas del modelo 32
Figura 18 Edificio de tipo mina con la plataforma de construcción con la visibilidad permitida 33
Figura 19 Edificio de tipo fábrica sobre el terreno. La plataforma ya no se encuentra visible 33
Figura 20 Elementos que componen el sistema de base incrustada en una zona de influencia 34
Figura 21 Indicador visual del progreso del estado de captura que presentan las bases 35
Figura 22 Demostración visual del resultado del usuario participando en el proceso de captura 36
Figura 23 Scriptable object presente en todas las unidades, con valores particularizados 36
Figura 24 Representación de los estados de la unidad mediante la herramienta animator de Unity 37
Figura 25 Ejemplo visual del usuario, seleccionando dónde situar el edificio de tipo mina 39

Figura 26 Interfaz de creación de unidades que se muestra al seleccionar una mina funcional 40
Figura 27 Unidad de tipo civil económica desplazándose para recolectar recursos ... 40
Figura 28 Ejemplo de edificio de tipo económico denominado “Panel solar” 41
Figura 29 Comprobación de la decisión tomada por la lógica de facción y su ejecución 43

Índice de tablas

Tabla 1 Planificación previa de la distribución de horas esperada por tareas 8
Tabla 2 Efectividad de unidades en el cálculo final del valor de ataque una vez aplicados los multiplicadores apropiados 17
Tabla 3 Comprobación de los valores de probabilidad esperados en cada una de las ecuaciones con los valores especificados 28
Tabla 4 Tabla referente a los objetivos de desarrollo sostenible y su relación con este proyecto 49

Índice de ecuaciones

Ecuación 1 Valor final en el cálculo del daño infligido en el ataque, una vez aplicadas las modificaciones por blindaje y tipo de unidad 16
Ecuación 2 Probabilidad de ataque total normalizada de 0 a 1 27
Ecuación 3 Probabilidad de defender normalizada de 0 a 1 27
Ecuación 4 Probabilidad de ataque limitado normalizada de 0 a 1 27

1 Introducción

1.1 Presentación

La motivación principal de este proyecto surge del interés por las experiencias interactivas, abordadas a lo largo de la carrera. Por ello, se ha decidido investigar posibles oportunidades de desarrollo de productos en este ámbito.

Entre las diversas opciones, se ha considerado el campo de los videojuegos como una opción de interés, tanto por su creciente impacto social y comercial como por su papel como motor de innovación en investigación, desarrollo e innovación (I+D+I). La categoría seleccionada para este pequeño prototipo corresponde a los videojuegos de estrategia en tiempo real, o por sus siglas en ingles *RTS*.

Los elementos esperados, en la naturaleza de un juego *RTS*, incluyen la presencia de unidades (personajes) que actúan como piezas a las que el jugador puede asignar órdenes y que pueden interactuar con elementos específicos de su entorno. Otro elemento esencial es un sistema de recursos que limita las acciones, obligando al usuario a utilizar de manera eficiente los medios disponibles.

Dentro de las posibilidades de los *RTS*, se elige un modelo con que permite la construcción. Esto significa que el usuario puede alterar el plano de juego mediante unas estructuras (edificios). De forma adicional, se espera implementar un sistema que condicione el flujo de la sesión (partida), obligando a tomar decisiones; esto se logrará mediante un pequeño sistema de puntos capturables.

Puesto que no se van a incluir funciones multijugador en el prototipo, se buscará proporcionar una condición de victoria al derrotar a un enemigo simulado. Este enemigo podrá tomar decisiones muy sencillas como atacar o defenderse, esto se implementa de forma orientativa y no se espera que sea capaz de interactuar con todos los sistemas.

Siendo únicamente un prototipo de un nivel, el alcance del proyecto se limita a diseñar e implementar una versión mínima viable que incluya algunas de las mecánicas básicas esperadas en un *RTS*. No se pretende que sea un juego completo ni un producto comercial.

1.2 Objetivos

En el presente apartado, se incluyen las metas propuestas en este proyecto, divididas en objetivos generales y específicos:

OBJETIVO GENERAL:

- Realizar una aplicación interactiva en la plataforma Unity, orientada a la realización de un prototipo del primer nivel en un hipotético producto.

OBJETIVOS ESPECÍFICOS:

- Aplicar los conocimientos adquiridos durante los estudios realizados, como podrían ser entre otros, programar o plantear un proyecto.
- Experimentar con plataformas que permiten la creación de experiencias interactivas, como es el caso de *Unity*.

- Explorar el potencial de estas herramientas para desarrollar el prototipo de un producto, que tiene las características propias de un juego de tipo *RTS*.

1.3 Metodología

Se prioriza el uso de buenas prácticas, en consecuencia, en lugar de proceder a implementar directamente el proyecto, se aplica el siguiente flujo de trabajo:

Diseño en diagrama

Antes de implementar nuevas secciones, se realiza un pequeño diagrama o representación visual, para asegurar su comprensión y cohesión con el resto de las ideas.

Testeo aislado

El proyecto usa control de versiones, gestionado con la aplicación *Git*. No se realizan cambios importantes sobre la rama principal, sin que hayan sido probados con anterioridad en una rama secundaria.

Implementación conjunta y revisión

Únicamente cuando se ha validado que el cambio funciona correctamente, se integra en la rama principal y se comprueba que no hay interacciones no deseadas.

Versión estable

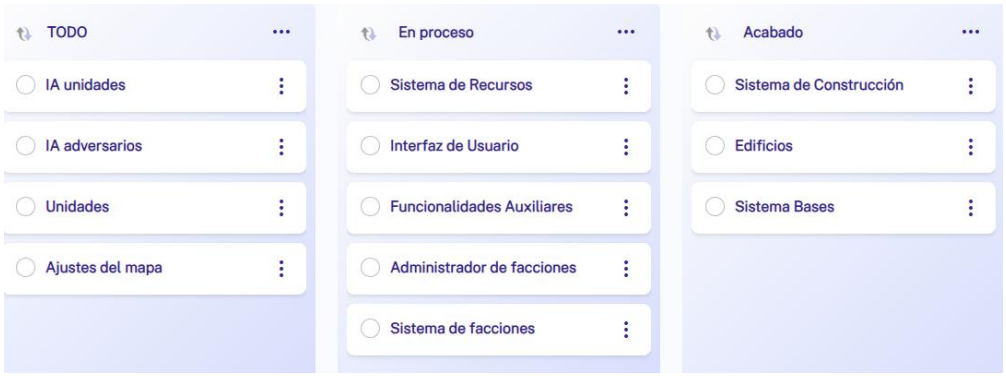
Una vez comprobado el correcto funcionamiento, se actualiza la rama principal y esta pasa a ser la nueva versión estable.

1.4 Etapas

El diseño e implementación no han sido realizados en cascada. Aun cuando hubo un proceso de diseño inicial, fue necesario realizar otras fases de diseño de menor tamaño intercalados, para adaptar el proyecto a cambios.

Figura 1

Tablero de distribución de tareas compatible con metodología Scrum



Se han establecido los objetivos en un tablero de organización de tareas, que incluye las mecánicas ya desarrolladas en la sección de diseño. En la Figura 1 se muestra un ejemplo de un momento durante la producción del proyecto, en el que se habían completado aspectos como el sistema de construcción, aunque el

funcionamiento interno de las unidades aún no había comenzado. El método de organización mediante el tablero permite un desarrollo ágil, facilitando el movimiento de las tareas entre las columnas.

1.5 Desafíos

El proyecto, tal y como ya se ha mencionado, se centra en crear un potencial prototipo de un producto. Por ende, genera la necesidad de dar preferencia a unas funcionalidades frente a otras, priorizando las más representativas.

En la perspectiva de que esta clase de producto requiere una gran variedad de perfiles, se opta por un enfoque mayoritariamente técnico. Los diseños que corresponderían a aspectos artísticos están muy simplificados y en el contexto de la implementación se utilizan recursos visuales de uso gratuito. Por consiguiente, el aspecto visual no es una representación precisa de la visión estética, propia del estándar de un potencial producto terminado.

1.6 Planificación previa

Plan de trabajo:

Se asume un ritmo de jornada de 8 horas, 5 días a la semana. Por ello 300 horas corresponden a 37,5 días en total. Realizándose la partición por semanas, corresponde a 7,5 semanas, aproximado al valor 8 para generar un margen de seguridad de imprevistos. Se puede ver en la Tabla 1 la estimación de la distribución de horas y tareas por semana, tratando de lograr una distribución regular de tiempo.

Tabla 1
Planificación previa de la distribución de horas esperada por tareas

Semana 1: (40 horas)	• Recopilación de información • Generación de diseños
Semana 2,3: (80 horas)	• Implementaciones mecánicas generales • Sistemas de unidades • Sistemas de edificios • Sistemas de recursos • Interacción entre estos sistemas
Semana 4: (40 horas)	• Mecánicas específicas del usuario • Interfaces e interactividad
Semana 5 y 6: (80 horas)	• Algoritmo básico capaz de utilizar las mecánicas generales
Semana 7 y 8: (80 horas)	• Comprobación de errores • Elaboración de Memoria y Presentación

2 Implementaciones comerciales existentes

2.1 Contextualización

La explicación de lo que se puede incluir dentro de un juego *Real Time Strategy* es compleja. El componente *real time* está presente para realizar una separación taxonómica, para remarcar que no se trata de una implementación por turnos; donde el usuario dispone de un tiempo virtualmente ilimitado para tomar decisiones.

El componente *strategy* es el que genera la necesidad de plantear una clara distinción con otros géneros de juegos. Siendo usada la acepción inglesa, se comprueba la definición que proporciona el diccionario Cambridge.

"*A detailed plan for achieving success in situations such as war, politics, business, industry, or sport, or the skill of planning for such situations*". (Definition of **strategy** from the Cambridge Advanced Learner's Dictionary & Thesaurus © Cambridge University Press)

De acuerdo con esta definición, se puede interpretar como estrategia el realizar acciones siguiendo la forma más eficiente posible en base a un plan. Esto genera fricción con la idea de que, al competir en cualquier ámbito, se presume que los participantes se propondrán alcanzar su mayor desempeño posible, maximizando su eficiencia y muy posiblemente, teniendo un plan.

El objetivo que se pretende alcanzar es que, incluso jugando a un partido de fútbol con intención táctica, la clasificación por géneros seguiría considerando este partido como la categoría de deportes. Aunque este ejemplo podría parecer evidente a simple vista, se presentan casos donde la línea es mucho más difusa, como podría ser un título de gestión de recursos en tiempo real. Mientras esto es una componente del género RTS, por las definiciones, se incluiría dentro del género de la simulación.

Para concluir la explicación esta sección, se presenta como punto adicional de distinción que los juegos de estrategia son herederos de los juegos de tablero. Por ello, su enfoque se centra en la representación y contextualización eficiente de información, aunque tenga que sea a coste de elementos cinemáticos, como por ejemplo el fotorrealismo. (Apperley, T., 2006)

2.2 Casos de ejemplo

Una vez determinados los límites de lo que se considera y lo que no se considera un RTS, el siguiente punto examina ejemplos de referentes relevantes para así analizar sus elementos.

Figura 2

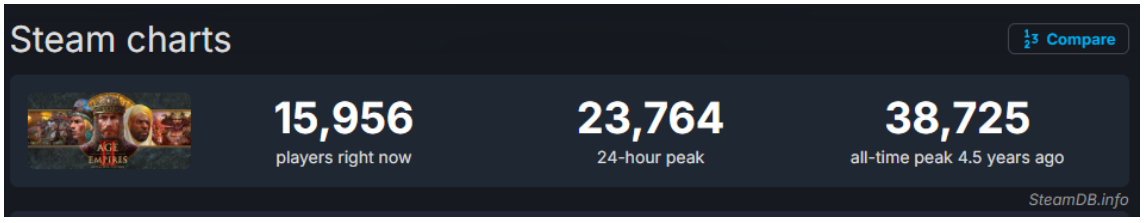
Producto referente en el mercado RTS con nombre comercial: Age of Empires II: Definitive Edition.

Nota: Imagen obtenida de la página oficial del producto Age of Empires II: Definitive Edition en la plataforma de venta de videojuegos Steam.

El RTS que se muestra en la Figura 2 corresponde al juego denominado *Age of Empires II*, en este caso no siendo la versión original de salida de 1999, sino una edición más reciente. El motivo de usar este título como primer referente, radica en tener una base de usuarios activa, pese a su longevidad y tras más de dos décadas en el momento que se redacta este documento, aún posee una comunidad activa.

Figura 3

Información sobre la cantidad de usuarios activos del videojuego Age of Empires II en la plataforma de venta de videojuegos Steam.



Nota: Información extraída del servicio web SteamDB, una plataforma de visualización de información third party de la plataforma de videojuegos Steam.

Las cifras presentes en la Figura 3, permiten generar una idea aproximativa sobre la cantidad de jugadores presentes en esta comunidad. Estos datos son extraídos de la plataforma online *steamDB*, que proporciona información y valores estadísticos de los juegos de la plataforma *Steam*. Estos datos serán utilizados como una aproximación u orientación, ya que no presenta la frecuencia de actualización de los datos que podría proporcionar la *Steam* o la empresa desarrolladora directamente.

Otro título digno de estudio, que ha logrado ser referente en el género, es el videojuego *Starcraft 2* presente en la Figura 4. Una de las principales razones de ser seleccionado es el contraste entre ambos títulos, tanto en términos de estética como de temáticas, así como en las diferencias en la implementación de sus mecánicas.

Figura 4

Producto referente en el mercado RTS con nombre comercial: Starcraft II.

Nota: Imagen obtenida de la página oficial del producto Starcraft II en la plataforma battle.net, cuyos propietarios corresponden a la empresa Blizzard Entertainment.

Los elementos comunes que se pueden encontrar en ambos incluyen:

- Se presentan dos o más facciones opuestas. Los adversarios simulados buscan activamente de lograr la victoria
- La acción se desarrolla en un mapa que actúa como un tablero de juego
- El usuario puede modificar el estado del tablero con unas entidades situadas sobre el mismo, denominadas edificios.
- En el tablero hay entidades con las que se puede interactuar para obtener recursos; en un caso, se trata de árboles, y en el otro, de minerales
- Existen unidades en el mapa que pueden desplazarse, permitiendo al usuario manifestar su voluntad sobre el plano de juego. Cada unidad posee características particulares.
- Dentro de las unidades, existen especializaciones, como las unidades civiles, que tienen habilidades de combate muy limitadas o casi inexistentes, pero pueden interactuar con sistemas que otras unidades no pueden utilizar.
- Existen elementos sobre el tablero que sirven para afectar la movilidad, generalmente en forma de obstáculos.

Mientras que estos dos títulos ofrecen una visión general, se requiere estudiar una referencia del enfoque especifico deseado.

En este prototipo, el objetivo es crear una experiencia intelectual en la que el usuario enfrente un reto utilizando ciertas piezas, sin que esto se convierta en una justificación para la violencia. Esto sugiere la necesidad de explorar un enfoque en el que las unidades tengan un trato de piezas, como ocurre en el ajedrez. Se busca una representación simple, pero que transmita de forma efectiva e inmediata la información necesaria sobre la pieza.

Figura 5

Producto comentado como referente estético deseado, con nombre comercial: Supreme Commander 2.

Nota: Imagen obtenida de la página oficial del producto Supreme Commander 2 en la plataforma de venta de videojuegos Steam.

En la Figura 5 se aprecia un ejemplo de la idea estética que se busca obtener. Los diseños presentan un enfoque geométrico con formas fácilmente reconocibles. La interacción entre las piezas se realiza con partículas básicas, viéndose de forma clara la acción, sin representar situaciones violentas o incómodas para los usuarios. Este mismo estilo se aplica en el diseño de los edificios.

En resumen, el primer punto de partida se estableció al intentar incluir en una versión mínima viable los elementos que se consideran esperables en estos títulos precedentes. A nivel visual, se tomó el último caso como referencia, aunque este aspecto se encontró condicionado por limitar los recursos a materiales gratuitos y contenido de generación propia de carácter muy simple.

3 Idea para implementar

El objetivo de crear el prototipo es una forma de estudiar el género de los RTS y explorar las mecánicas de cara a un potencial producto. A pesar de ello, resulta útil en el desarrollo visualizar cuál es el resultado final deseado.

Dejando de manifiesto desde el primer instante, que este punto no pretende ser un *game document* o ser tratado como si fuera el fragmento de uno. Esto se debe que para poder sostener tal afirmación sería necesario responder a cuestiones que no van a ser planteadas durante la sección.

Una vez que se ha establecido el alcance de esta sección, corresponde plantear las ideas que van a definir la idea central del producto; evitando así contradicciones durante este desarrollo.

Título: Próxima B, en referencia al planeta homónimo.

Temática: espacial, futurista.

Público objetivo: Existe preferencia por parte del público masculino. Según un estudio realizado: 18 % de las mujeres encuestadas respondió una preferencia alta por el género RTS frente al 36% de los hombres encuestados. (Try Evidence Research Team (2021)).

Contexto: Se plantea como un futuro que la humanidad, en su nulo deseo por la preservación de los recursos, se ha visto forzada a tener que extraerlos en otros planetas, con los peligros que ello conlleva, tanto por parte del entorno hostil como por la avaricia de corporaciones.

Flujo del nivel: El objetivo de este nivel es proporcionar un espacio para que los nuevos usurarios aprendan a jugar, actuando como un tutorial. Dado que solo se han integrado algunos elementos del producto final, esto puede ser beneficioso para el concepto del tutorial, ya que, al centrarse únicamente en lo esencial, no abruma a un jugador nuevo.

Este nivel se basa en la idea de que el adversario tiene objetivos más deseables que el de interactuar con el jugador. Ya que en este nivel el rival no va a construir una base, será suficiente con que el jugador use este tiempo para construir una, para poder ganar de forma sencilla.

4 Diseño

4.1 Introducción

Conforme a las referencias recopiladas en el estudio previo, se pueden inferir unas necesidades básicas, que deben estar presentes. En consecuencia, es necesario plantear como se van a enfrentar a estas situaciones.

De igual manera, se han identificado requerimientos de contenido de carácter visual. A pesar de que la intención sea poder suplir esta necesidad con recursos gratuitos, no se descarta la necesidad de que pueda llegar a ser necesario la elaboración de algún contenido muy básico o la modificación de alguno ya existente. Además, se suma la necesidad de desarrollar representaciones visuales sencillas para el diseño.

4.2 Metodología

Antes de comenzar la fase de implementación, se llevó un proceso en 3 fases:

- **Fase de planificación**: Se elaboró un listado de metas con el fin de definir cómo lograrlas.
- **Fase de diagramas**: Se realizaron esquemas y se hicieron preparativos para abordar estos objetivos.
- **Fase de búsqueda de tecnología**: Se revisó la documentación del motor de desarrollo y artículos académicos para verificar la validez del enfoque.

Tras concluir este este proceso inicial, se usó una metodología *scrum* para asegurar una distribución del tiempo que permita dar cabida a las tareas decididas.

4.3 Herramientas

Para llevar a cabo el desarrollo de una aplicación, como un videojuego, es esencial utilizar un programa que haga la función de motor. En este contexto, motor se refiere a un software que proporciona las herramientas necesarias para crear juegos. El software elegido para cumplir con este propósito es *Unity*.

Unity fue elegido por disponer tanto de una documentación extensa como de muchas herramientas útiles ya incluidas. Además de estas funciones ya incluidas de base, Unity también tiene otros paquetes de funcionalidades descargables para ayudar a los desarrolladores, como es el caso, entre otros, del paquete *NavMesh.*

Para crear el código necesario para generar el juego en *Unity* se requiere vincularle un procesador de código. Este proyecto utiliza *Visual Studio,* implementando sobre el lenguaje C#. Este genera los *scripts***,** que permiten al desarrollador interactuar con la aplicación.

Dentro de *Unity* existe una tienda digital denominada *UnityStore*. Esta contiene una cantidad de recursos digitales gratuitos, realizados por la comunidad. Esta es la principal fuente de recursos externos usados como elementos visuales en el proyecto, como texturas o modelos. Existen otros servicios desde los cuales se han descargado recursos gratuitos, pero en una proporción mucho menor, como puede ser el servicio *flaticon.com*. De este se han obtenido un par de iconos, para la representación visual de las unidades.

Las referencias a los recursos gratuitos usados son las siguientes:

- (*Poly Angel - Space Pack | 3D Space | Unity Asset Store*, 2025)
- (*Translucent Crystals | 3D Fantasy | Unity Asset Store*, 2022)
- (*Yughues Free Metal Materials | 2D Metals | Unity Asset Store*, 2013)

Para la generación de diagramas en formato digital se ha optado por la herramienta de diseño gráfico gratuita *Draw.io*.

El proyecto ha sido realizado en un ordenador con sistema operativo *Windows 11*, con un procesador del fabricante AMD y una tarjeta gráfica producida Nvidia. Al ser el único entorno donde ha sido puesto a prueba, se desconoce como interactúa en otras configuraciones.

4.4 Interacción con el medio

El juego está diseñado actualmente para ser usado jugado en un ordenador. Con ello está delimitado a los periféricos ratón y teclado. No se espera compatibilidad con mando u otros sistemas portables en ningún punto.

4.4.1 Cámara y controles

Representa la visión por parte del usuario del terreno de juego. La interacción radica en las acciones de desplazamiento:

- **Desplazamiento**: Teclas WASD en el sentido cardinal de las teclas.
- **Zoom**: Rueda del ratón.
- **Clic izquierdo seguido de arrastrar ratón**: Se crea un cuadro de selección. Si hay unidades dentro, se consideran seleccionadas.
- **Clic izquierdo sobre unidad**: Se selecciona la unidad específica.
- **Clic izquierdo sobre interfaz**: Se realiza la acción especificada, si es posible.
- **Clic izquierdo con unidad seleccionada:** Se asigna una orden a la unidad.

4.4.2 Interfaces de usuario

La interfaz de usuario se divide en dos partes, la interfaz superior y la inferior.

- **Interfaz superior**: De tipo informativo. Contiene la información de los recursos del usuario.
- **Interfaz inferior**: De tipo interactivo. Alterna los menús de construcción de edificios, menú de construcción de unidades y menús de unidades seleccionadas según cual sea la selección actual.

4.5 Rasgos y estadísticas de las entidades controlables

Las entidades del juego se dividirán en dos grandes categorías, unidades y edificios. Internamente comparten rasgos en común, pues ambos heredarán de un mismo origen. Por ello, las variables que todas las unidades disponen son las siguientes:

- **Tipo**: Hace referencia a la categoría de unidad. Cada unidad tiene sus propias fortalezas y debilidades contra otras unidades. Esto se ejerce con un multiplicador.

- **ID**: Identificador de la unidad. Funciona a su vez de nombre de unidad como identificador interno para el reconocimiento de la unidad. Se usará en la selección de unidades.
- **Iconos:** Representación gráfica de conceptos de la unidad, se usa en la creación y selección.
- **Tiempo de construcción:** Tiempo requerido para la producción de unidades. En caso de ser un edificio, se usa para su tiempo de construcción.
- **Costes**: Cantidad de recursos necesarios para producir o mantener la unidad en funcionamiento.
- **Estadísticas:** Muestran las capacidades de la unidad, representada cuantitativamente de forma numérica. En caso de no poder usar la estadística en cuestión, el valor será 0. Un ejemplo son las unidades sin capacidad de ataque. Entre los tipos de estadísticas se encuentran:
    - **Agresividad (rango de agresión):** Rango máximo de persecución de una unidad. Su uso permite ajustar rangos de detección.
    - **Rango de Detección:** Distancia máxima desde la cual la unidad declara a otra unidad como su objetivo.
    - **Rango de ataque**: Distancia máxima desde la cual la unidad puede declarar acciones ofensivas.
    - **Velocidad de ataque**: Marca el intervalo de tiempo entre cada acción ofensiva.
    - **Velocidad del proyectil:** Velocidad a la que el proyectil recorre la distancia.
    - **Vida**: Valor que representa la cantidad del valor numérico de unidades de daño que puede recibir una unidad antes de ser retirada del juego.
    - **Blindaje**: Modificador normalizado de 0 a 1 que se usa para representar la capacidad de resistencia a daños de la unidad.
    - **Daño de ataque**: Valor asignado al daño realizado, antes de aplicarse la fórmula para obtener el daño final.
    - **Daño final**: Valor total de la acción ofensiva, una vez aplicados los modificadores.

Ecuación 1
Valor final en el cálculo del daño infligido en el ataque, una vez aplicadas las modificaciones por blindaje y tipo de unidad

$$Daño_{Final} = Daño_{ataque} * (1 - Blindaje) * (Modificiador_{Tipo})$$

El motivo por la que, en la fórmula utilizada para calcular el daño infligido, se sigue la ecuación 1 en lugar de simplemente restar el valor del daño del valor de la vida, es para que las categorías de unidades tengan sentido. Se busca que las entidades más

resistentes, representadas por el modificador de blindaje, incentiven el uso de unidades con un multiplicador de efectividad alto para contrarrestar este valor de blindaje.

Usando un ejemplo en que una unidad infringiera 10 unidades de daño, el cálculo total equivalente se muestra en la siguiente tabla:

Tabla 2

Efectividad de unidades en el cálculo final del valor de ataque una vez aplicados los multiplicadores apropiados

Daño origen	Blindaje	Efectividad	Estimación
Tipo 0			
10	0,9	0,5	0,5
10	0,7	0,5	1,5
10	0,5	0,5	2,5
10	0,2	0,5	4
Tipo 1			
10	0,9	1	1
10	0,7	1	3
10	0,5	1	5
10	0,2	1	8
Tipo 2			
10	0,9	2	2
10	0,7	2	6
10	0,5	2	10
10	0,2	2	16
Tipo 3			
10	0,9	5	5
10	0,7	5	15
10	0,5	5	25
10	0,2	5	40
Tipo 4			
10	0,9	10	10
10	0,7	10	30
10	0,5	10	50
10	0,2	10	80
Tipo 5			
10	0,9	50	50

La tabla 2 busca reflejar los casos extremos. El tipo 0 es un caso especial, ya que presenta una reducción de daño del 50%, por lo que incluso contra entidades con el blindaje más bajo, la perdida de daño es enorme. Se destaca también el tipo 5, siendo una medida para poder lidiar con una reducción de daño del 90% suplida con un multiplicador de 50 al daño reducido, pudiendo ignorar valor del blindaje.

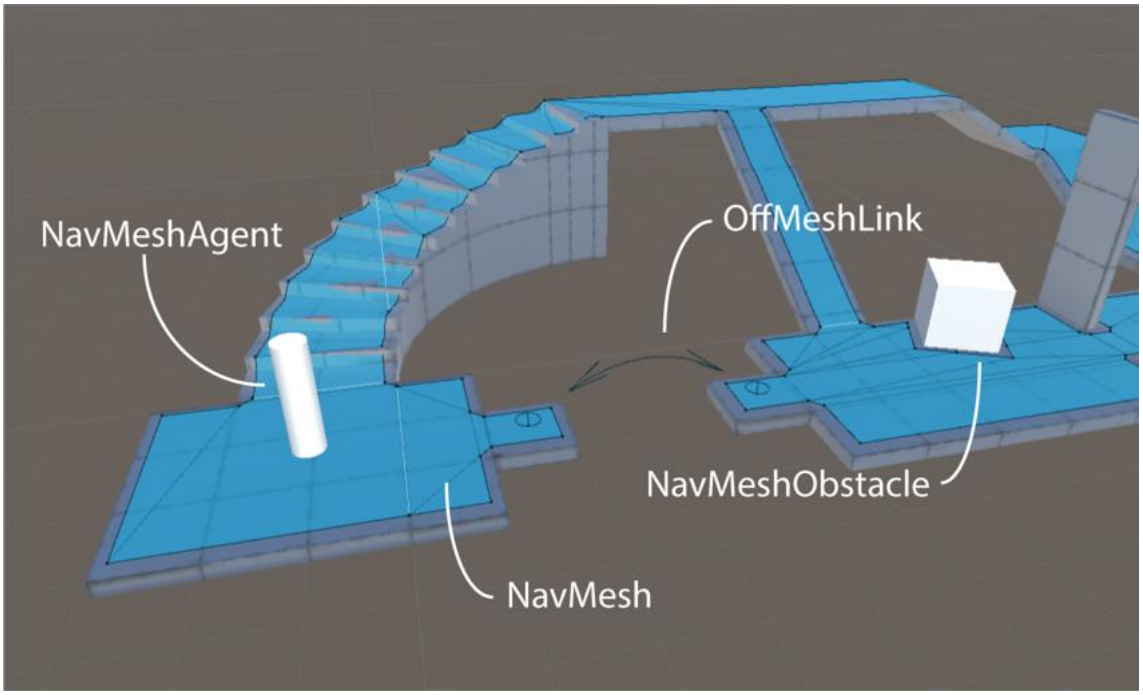
4.6 Usos de las utilidades NavMesh en el desplazamiento

Las unidades son entidades con capacidad de movimiento, las cuales usan esta capacidad para realizar acciones sobre el terreno. Esta capacidad de desplazarse la diferencia de los edificios, los cuales son estáticos.

Esta capacidad se debe al paquete de utilidades *NavMesh*. Este paquete permite definir qué zonas son accesibles y cuáles no. En las áreas a las que se les asigna esta clasificación, este paquete gestiona también las exigencias de *pathfinding* (búsqueda de caminos) requeridas para moverse por ellas, tratando de utilizar la mejor ruta posible en base a las condiciones establecidas.

Figura 6

Representación visual de la librería Navmesh y sus elementos básicos.



Nota: Ilustración extraída de la documentación oficial por el engine Unity (Unity Technologies, (2020)).

Como puede apreciarse en la Figura 6, en la documentación de esta utilidad se presenta un ejemplo donde se ilustran los elementos necesarios de este paquete de funcionalidades. En esta figura, algunas entidades están integradas dentro de la superficie representada en azul, mientras que otras no lo están. Esto indica cuáles participan en el movimiento. *NavMeshAgent* hace referencia a las entidades que pueden desplazarse y *NavMeshObstacle* a las entidades que ejercen el rol de obstáculos. Las unidades usarán esta primera utilidad y los edificios la segunda.

4.7 Edificios

Entidades que no cuentan con la capacidad para desplazarse. Esta situación es compensada con otras funcionalidades, las cuales dividen a los edificios en 3 categorías:

- **Económicos**: Generan ingresos para la economía del usuario. Algunos de estos edificios requieren ser operados por unidades especiales. En caso de necesitar dichas unidades, se pueden construir desde este edificio.
- **Producción**: Generan unidades para sus propietarios.
- **Apoyo**: Aportan utilidades de tipo ofensivas o defensivas.

4.7.1 Base

Existe una clase especial de edificio. Su característica principal es que puede ser tomado por otros jugadores, en lugar de ser destruido cuando su valor de vida desciende al valor 0.

La posesión de este tipo de edificio es imprescindible. Se requieren tanto para expandirse por el mapa como para no ser eliminado de la partida. Si no se tiene en posesión al menos una base por más de un periodo especifico, se traduce en que este jugador resulta eliminado.

4.8 Construcción

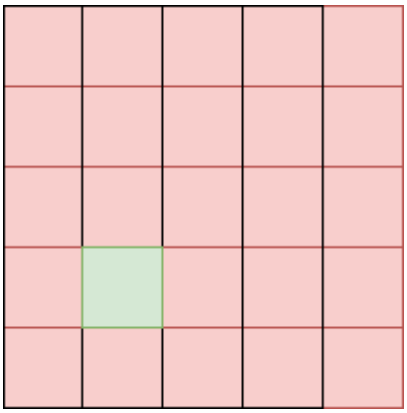
4.8.1 Zonas de control

Debido a que, por el modelo de implementación, no requiere que haya una unidad civil encargada de administrar la construcción; es necesario un factor limitante sobre qué zonas se puede edificar para evitar abusos del sistema.

Este sistema será proporcionado por los edificios de tipo capturable. Estos generarán una parcela o frontera que representa la zona bajo el control del usuario. A esta zona de color generada se le ha asignado de nombre zona de control.

Estas zonas de control constituyen una forma de objetivo, que actúa como una forma de forzar a los usuarios a interactuar con los sistemas y también entre ellos. Las zonas de control pueden ser capturadas y defendidas entre los jugadores.

Figura 7
Representación visual de la división del mapa por zonas, divididas entre los jugadores, los cuales se identifican por diferencias de color.



En la Figura 7, se muestra un caso extremo en el cual, el jugador que usa el color verde solo podría interactuar con la zona de este color, mientras que el jugador

representado por el color rojo dispone de todo el mapa. Esta diferencia trata de mostrar lo importante que es esta mecánica y las ventajas que aporta proporcionarle atención.

4.8.2 Cuadrícula de construcción y terreno especial

Es importante señalar que esta no es la única restricción en el proceso de construcción. Aparte de la cuadrícula de zona de control, existe internamente una cuadrícula de divisiones que representan los emplazamientos válidos para el usuario, llamada cuadrícula de construcción. Esta segunda cuadrícula es una utilidad para el sistema de construcción y no será visible.

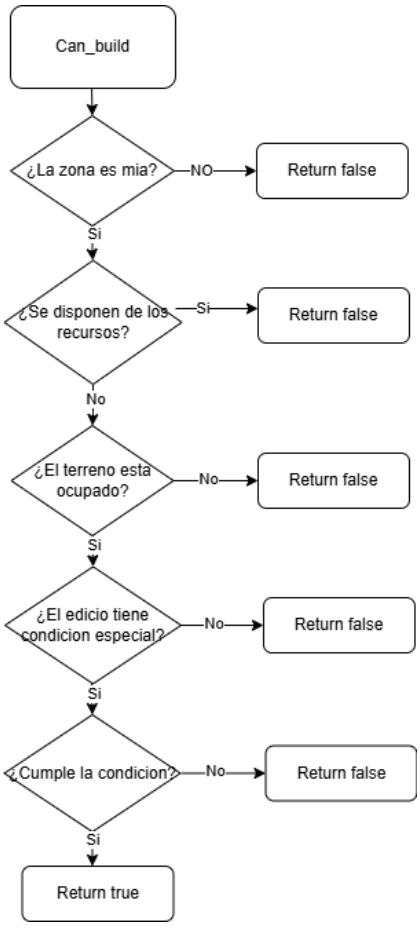
Una última condición puede aplicarse a algunos tipos de edificios, siendo que solo pueden construirse sobre un terreno especifico. A este tipo de terreno, le asignamos como nombre terreno especial. El caso más común de terreno especial es el terreno vinculado a un tipo de recurso, como serían los minerales.

4.8.3 Comprobaciones de construcción

Cada vez que se ejecute la construcción de un edificio, se realizan una serie de comprobaciones, antes de permitir esta acción. Esto se representan en la Figura 8:

Figura 8

Diagrama que representa las condiciones que permiten o deniegan realizar la construcción del edificio



Las primeras condiciones para comprobar posesión de la zona, la disponibilidad de la zona y la cantidad de recursos, son comprobaciones que se realizan independientemente del edificio seleccionado.

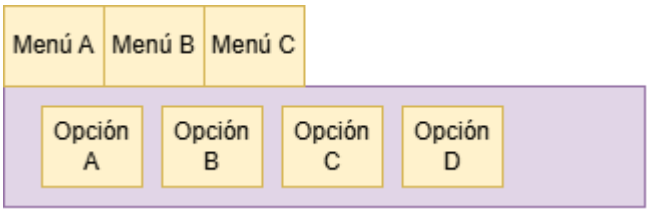
El aspecto al que se hace referencia en la Figura 8 respecto a una condición especial, ocurriría en el caso de un edificio de obtención de recursos. Si una mina se exige que se construya sobre un parche del mineral especifico, ocuparía este espacio en el flujo del diagrama representado.

4.8.4 Selección de edificios en interfaz de usuario

El paso previo a la construcción de estos edificios requiere que el usuario exprese su deseo de seleccionar específicamente cual desea posicionar en la cuadricula de construcción. Esto lo puede realizar tanto por los accesos rápidos en el teclado como por la interfaz de usuario

Figura 9

Diseño que representa la interfaz de usuario para la selección de edificios, situada en la parte inferior de la pantalla



A modo de diseño, se ha esbozado cómo se pretende que sea esta interfaz. En la Figura 9 se puede ver cómo se desea que el usuario interactúe con la mecánica. En la parte inferior de la pantalla se encontrarán 3 botones, referentes a las 3 categorías de edificaciones. Desde estos, una vez pulsados, se desplegarán las opciones permitidas de cada tipo.

Una vez seleccionado el edificio, desde este menú, el usuario puede pulsar sobre este, lo que inicia el modo de construcción. Si cumple las limitaciones nombradas en la sección previa, se le permite construir.

4.9 Sistema de recursos

Para el correcto funcionamiento del juego, cada jugador debe tener su propia economía, donde cada participante obtiene unos inputs desde los edificios económicos. Estos son almacenados en su propia reserva de recursos, para cubrir los costes de producir unidades y edificios.

Estos recursos representan ideas de manera abstracta, por lo tanto, lo relevante no es el término en sí, sino cómo se obtienen y se utilizan. Estos recursos son los siguientes:

- **Minerales:** Requieren un terreno especial vinculado a este recurso y unidades dedicadas a su extracción.
- **Petróleo:** Similar en funcionamiento a los minerales. Pero tiene una presencia menor en el mapa.

- **Energía:** No requiere terreno especial ni unidades de extracción. Para compensar esta ventaja, este edificio es menos productivo, para condicionar que sea necesario construir más edificios de este tipo.
- **Piezas:** Similar en extracción a la energía. Pero presenta una ratio bastante peor de producción de recurso en relación con el coste.
- **Población:** Restricción en el número de unidades que cada jugador puede poseer simultáneamente. Este tipo de edificaciones aumenta el límite en lugar de producir un recurso. El valor máximo de esto está limitado por el mapa.

Figura 10

Diseño que muestra la interfaz de usuario para la representación de recursos del jugador, situada en la parte superior de la pantalla

Los recursos del jugador deben poder visualizarse en todo momento, en este caso, es la parte superior de la interfaz. En la Figura 10 se muestra de qué forma se desea que se representen estas cantidades. Los cuadrados de colores serán remplazados por una imagen significativa que permita su inmediata interpretación.

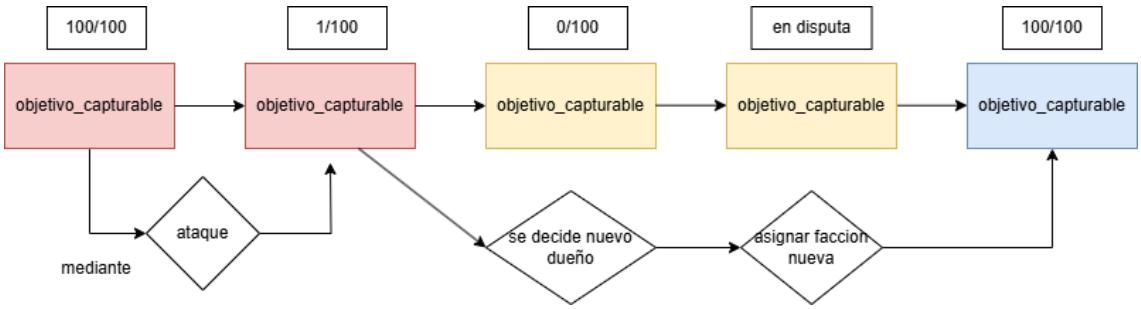
4.10 Sistema de facciones

Cada jugador cuenta con una facción asignada. Esta simboliza al jugador y abarca todos los elementos relacionados con él. Cada facción posee su propia economía, unidades, edificaciones y, al menos, una zona de control.

Para poder integrarse con estos sistemas, cada unidad de la facción contendrá un indicador de a qué facción pertenece, dentro de su *script* de gestión de facciones. Este actúa en las circunstancias que exigen verificar qué unidad o edificio se ve afectado por una acción. Esto puede incluir tareas como asignar la facción a una unidad en el momento de su creación o sistemas más complejos como el sistema de captura.

Figura 11

Representación del flujo de sucesos esperado en la conquista de una base de un jugador por parte de otro jugador



En la sección de zonas de influencia se comentó que estas deben poder ser capturadas. Estas se centralizan desde una entidad llamada base. El sistema de captura funciona de acuerdo con el esquema mostrado en la Figura 11. La premisa en la que se sustenta es que las bases, mediante acciones ofensivas, pasan de las manos de un jugador o de estado neutro a un estado de conflicto o disputa. Durante el estado de conflicto se requiere ser el único jugador con mayor presencia en la zona, durante un periodo de tiempo. En caso de lograrlo, se adquiere la base y con ello la zona.

Esta mecánica se comenta en este apartado, puesto que la comprobación de qué unidades están participando en la captura y por ende quién está ganando, pasa por contabilizar las unidades por facción de cada bando implicado, en la fase de captura.

4.11 Representación alternativa del sistema de facciones

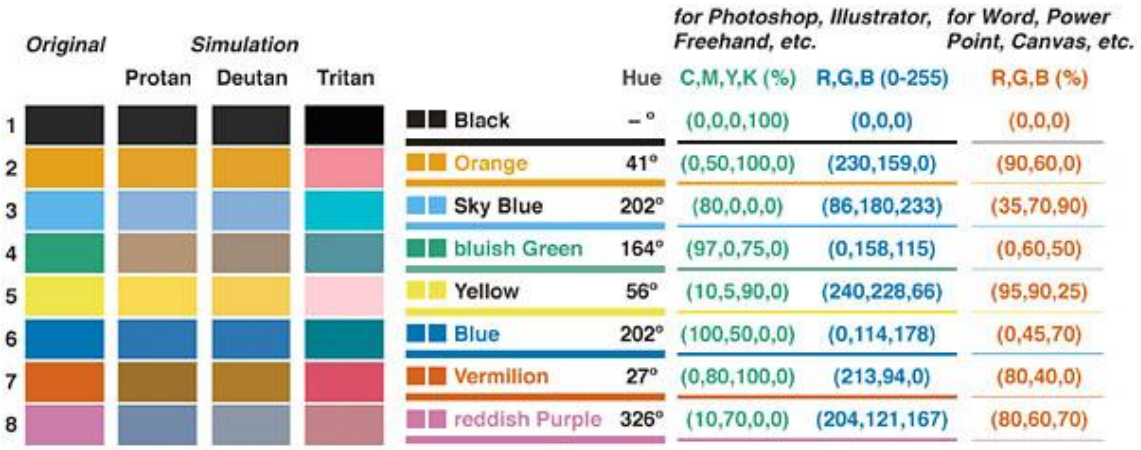
Tras realizar un segundo análisis en retrospectiva, se estudia la posibilidad de que un sistema, cuya representación principal se realiza a través de colores, podría generar dificultades a una parte de los jugadores, en caso de presentar un impedimento al reconocimiento de colores.

Mientras que esta es una implementación habitual, encontrarse en la situación de no disponer de la capacidad de poder diferenciar rápidamente a qué unidad pertenece a cada jugador, supone una desventaja.

La mejor solución sería contar con un sistema complementario que no dependa únicamente de esta característica. Tras determinar que la implementación actual carece de un equivalente que ofrezca la misma rapidez en el reconocimiento, se estudia la posibilidad de que hasta que se implemente una solución más efectiva, se ofrezca una paleta de colores alternativa, que sea lo más accesible posible para acomodar a los usuarios con diferentes percepciones del color.

Figura 12

Paleta de colores recomendada para mejorar la accesibilidad a perfiles de vision daltonicos



Nota: Estudio realizado en el campo del diseño universal de colores (Kei Ito and Masataka Okabe, August, 2002).

De acuerdo con el estudio representado de la Figura 12, usando estas tonalidades para representar los colores, podría lograrse que los principales tipos de daltonismo:

protan, deutan y tritan, los correspondientes respectivamente a las tonalidades rojas, verdes y azules/amarillas; generen un menor grado de probabilidad de que presenten problemas por los colores utilizados.

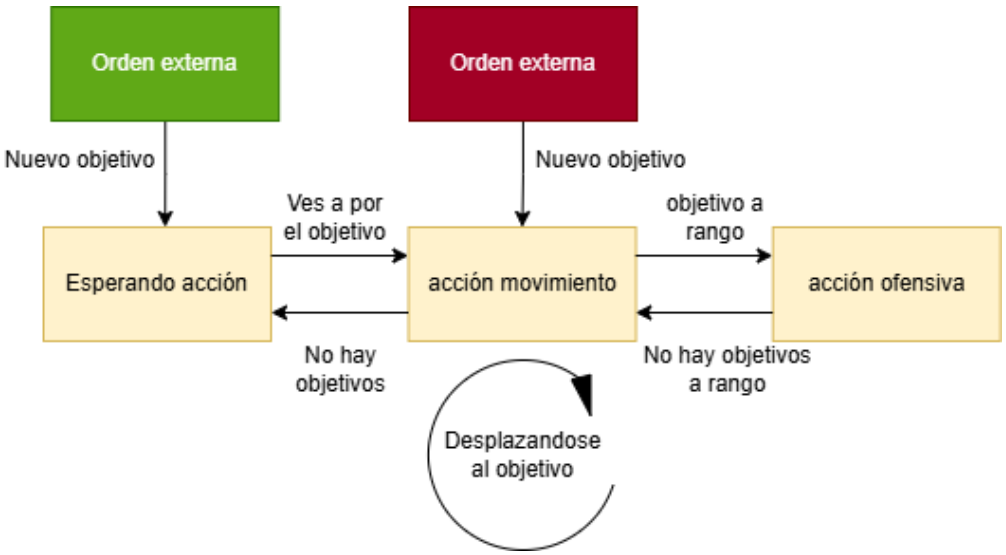
4.12 Lógica de funcionamiento de las unidades

Las unidades son entidades que deben realizar múltiples tareas, pues actúan como manifestación de las intenciones del jugador. Esto requiere la capacidad de poder desplazarse entre tareas de forma lógica, lo cual sugiere el uso de una máquina finita de estados como solución.

"Una máquina de estados consiste en un set de estados S_I Y un set de transiciones entre estados S_I Y S_J. Una transición contiene una condición y una acción. La condición es la causa que causa la transición y la acción se lleva a cabo cuando se realiza la transición". (Ben-Ari, Mordechai & Mondada, Francesco. 2018)

Figura 13

Diagrama de la máquina de estados básica que usan las unidades genéricas



Se ha representado un diagrama sencillo en la Figura 13 de la máquina de estados básica. En esta se muestra el flujo en el cual la unidad busca dentro de su área de detección objetivos o le son asignados. Una vez los identifica, se desplaza mediante la acción de movimiento hasta llegar al objetivo a rango de ataque e inicia las hostilidades contra la unidad, con la acción ofensiva.

Una vez realizada la acción ofensiva, se comprueba si tiene un objetivo a rango. En caso de que no cumpla con la distancia, realiza una acción correctiva con el movimiento, hasta que procede a estar en el rango de acción ofensiva otra vez. Si en cambio, la situación fuera que ya no tiene objetivo, vuelve al estado de esperar, donde busca objetivo y si no tiene ninguno a rango, espera que se le asignen.

Para poder realizar órdenes complejas, en lugar de implementar más estados, se asignan permisos para condicionar cómo realiza las acciones. Por ejemplo, para que la captura sea posible, se limita la acción de movimiento a no poder alejarse a una distancia determinada del punto asignado.

4.13 Lógica del funcionamiento de los adversarios

4.13.1 Introducción

A las facciones no controladas por jugadores humanos, a excepción de la facción neutral, se les puede asignar una lógica. En este contexto, la lógica se entiende como la capacidad de tomar decisiones, en base a la situación actual de la partida. Esto requiere un sistema de observación, evaluación y respuesta.

En esta implementación se ha optado por no recurrir a un módulo de IA o una funcionalidad descargable desde el store de Unity para esta función. Por este motivo, dentro del código se usa un pequeño algoritmo para ello. Para evitar confusiones, las referencias a esta capacidad de toma de decisiones se referirán a ellas como lógica.

4.13.2 Opción estudiada 1. Modelo basado en Árbol de decisión

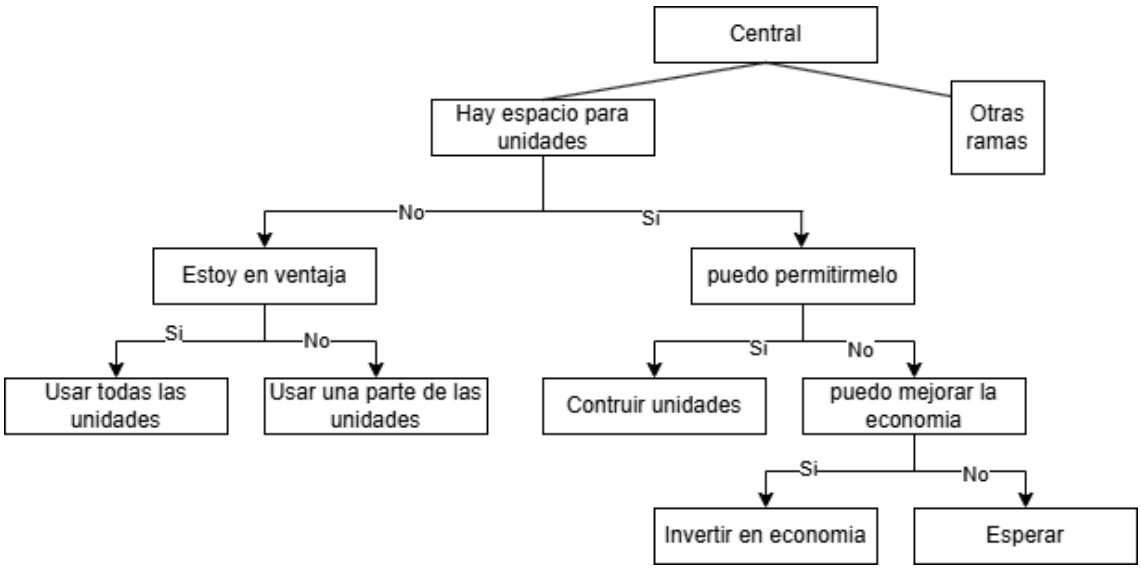
Un modelo de diagrama cuya toma de decisión se basa en unos puntos de decisión, llamados nodos y las rutas que conectan estos nodos, se denominan ramas.

La conexión de estos nodos representa las fases de un problema secuencial. Dentro de un nodo de decisión, se elige una acción dentro de las conexiones. Dentro de un nodo de probabilidad, se elige de forma aleatoria. Los nodos terminales representan el fin de una secuencia de toma de acciones. (Kamiński, B., Jakubczyk, M., & Szufel, P. (2018).)

En este caso específico, un único árbol no resultaría suficiente. Pese a ello, se estudió la posibilidad de dividirlo en varios árboles, con un sistema centralizado para gestionarlo.

Figura 14

Ejemplo de Intento de resolución del problema de toma de decisión, planteándolo en formato de diagrama de árbol.



Para comprobar la viabilidad del sistema, se plantea en la Figura 14, una resolución por diagrama de árbol, en el caso específico de la toma de decisión respecto

a si se debe realizar un ataque sobre el jugador o esperar a tener una fuerza mayor y la ruta por las ramas y nodos, para llevarlo a cabo.

Una vez se ha intentado crear una estructura para este modelo, surge la realización y para que se pudiesen obtener decisiones con sentido, no solamente sería necesario convertir cada nodo de la Figura en otro árbol, sino que sería necesario crear árboles paralelos para todas las decisiones necesarias.

Existen situaciones en las que este modelo sería deseable, especialmente si el mapa posee una narrativa en la cual se desee que el adversario realice un guion especifico de acciones y no baste con solamente condicionar a que ocurran unas situaciones específicas. Otra consideración que no se descarta es que hubiera una forma mejor de aplicar el algoritmo y simplemente no se ha logrado.

4.13.3 Opción estudiada 2. Modelo basado en máquinas de estados finitas

El siguiente punto de estudio recae en la posibilidad de darle otro uso al sistema de máquinas de estados, ya planteados para las unidades, en esta ocasión para la lógica.

El planteamiento trata de recoger las lecciones aprendidas del intento previo y dividir las ramas en máquinas de estados, los cuales no interactúan entre ellos directamente. Cada máquina realizaría la función de un sensor que activa una respuesta.

Mientras la alternativa en el contexto teórico es implementable, la razón por la que no se considera la elección, se debe a que se han encontrado dificultades en la gestión de las interacciones no deseadas. Al igual que en el punto anterior, seguramente existan mejores formas de implementarlo que las intentadas.

4.13.4 Opción elegida. Modelo basado en utilidad.

“La particularidad de un modelo de utilidad recae en el aspecto de que cada acción se le asocia a una utilidad, que define la relación entre los factores que han llevado a la decisión y la medida de la decisión. Estos factores son llamados consideraciones. El resultado de procesar curvas de la misma acción, pero con diferentes consideraciones genera el valor de la utilidad percibida”. (Świechowski, M., Lewiński, D., & Tyl, R. (2021, December))

Tras observar las oportunidades y problemas que presentan las anteriores opciones estudiadas, se determinó que se buscaría un sistema que logre adaptarse a un entorno que pueda presentar cambios con frecuencia, en gran medida por la necesidad de reaccionar a las acciones del usuario y cómo estas modifican el estado de la partida.

La implementación de este enfoque se lleva a cabo mediante el seguimiento de unas variables en la ejecución, que asignen un valor a las decisiones. Cada una de estas decisiones contiene una curva de decisión que le permite evaluar unos valores. Además, para proporcionar más opciones de cara a modificar el valor final, se combinan con un multiplicador que permite hacer función de peso, para poder aumentar o reducir el valor final.

El valor obtenido de esta consideración se denominará probabilidad de ejecución en lugar de utilidad, ya que se analiza la posibilidad de asignar al valor del peso una cantidad aleatoria dentro de un rango de valores específicos, para ocultar el resultado final al usuario. Pese a esto, para llevar a cabo las comprobaciones, se utilizan valores predefinidos en lugar de aleatorios para el peso, lo que permite predecir el resultado y corregir errores. Puesto que se requiere ajustar los valores para poder ser comparados, por el momento, la probabilidad se normaliza entre 0 y 1.

La primera prueba para diseñar el sistema se realizó con un conjunto de tres posibles decisiones. Estas son las siguientes:

- **Atacar**: usada idealmente si se aprecia una ventaja aparente.
- **Defender**: usada idealmente solo si se presenta una gran desventaja.
- **Ataque limitado**: usada idealmente si se presenta una pequeña desventaja para conseguir ventaja. El objetivo son las posiciones menos defendidas.

En base a la petición, se determina que, para satisfacer las necesidades, se requieren comparar la fuerza de cada parte implicada y una posición sobre la que considerar el valor del ataque limitado.

La fuerza propia se obtiene de la cantidad de unidades propias. Se denomina a esta como U_P. De la misma manera, la ventaja enemiga también se obtiene de la cantidad total de las unidades enemigas, U_E.

El valor de la posición enemiga con menor cantidad de defensa se obtiene consultando en la lista de bases. De esta lista, se comprueba la zona no propia con la menor cantidad de unidades hostiles. Este valor se nombra como “base menos defendida”, acortado a BDM.

De forma adicional, el valor por el que se multiplica cada expresión se denomina peso, usando la sigla inglesa W. Cada expresión tiene su propio valor de W. Este puede ser arbitrario o intencional, para reforzar determinadas conductas. Un ejemplo sería el asignar un valor mayor al peso en ataque para simular una actitud más agresiva. A continuación, se plantean unas posibles formas de curva en ecuación con estos valores.

Ecuación 2
Probabilidad de ataque total normalizada de 0 a 1

$$P_{Normalizada} = W_1 \cdot Log_{100}(100 \cdot \frac{U_P}{U_E})$$

Ecuación 3
Probabilidad de defender normalizada de 0 a 1

$$P_{Normalizada} = W_2 \cdot Log_{100}(100 \cdot \frac{U_E}{U_P})$$

Ecuación 4
Probabilidad de ataque limitado normalizada de 0 a 1

$$P_{Normalizada} = W_3 \cdot \frac{1}{(BMD \cdot (U_P - U_E))}$$

Con respecto a la Ecuación 2 y la Ecuación 3, se logra la forma de curva con el logaritmo. Se puede observar que son la misma expresión, pero inversa. Esto se debe a que, se espera generar las curvas opuestas. Se desea una reacción muy fuerte en caso de tener una ventaja o desventaja significativa mediante las respectivas fracciones.

En cuanto a la Ecuación 4, se logra la forma de curva mediante la fracción. El motivo de usar esta forma es que se desea tanto que los valores desciendan rápidamente. Tanto fuera del empate como si el valor BDM no es deseable. Este caso tiene un caso problemático en que el resultado de la sustracción puede ser 0. Por el momento, se ajusta sumando un decimal pequeño al valor de U_P si esto ocurre.

A continuación, se representan los valores esperados en un rango especifico de diferencia de unidades, tanto a favor como en contra. En la Tabla 3 se han incluido los valores numéricos de la probabilidad normalizada que abarca un rango entre 25 unidades a favor y en contra. En la Figura 15, se muestran la representación en gráfica, de forma ampliada, del rango de valores entre 50 unidades a favor y en contra. Los pesos utilizados, asignados para mantener los valores normalizados, corresponden a:

- W_1 **ataque** = 0,5.
- W_2 **defensa** = 0,4.
- W_3 **ataque limitado** = -5.

Adicionalmente, se han comprobado dos valores BDM para el ataque limitado. Uno siendo favorable, teniendo presente una base enemiga defendida con solo 2 unidades y otro desfavorable, en el cual la posición menos defendida contaba con 15 unidades.

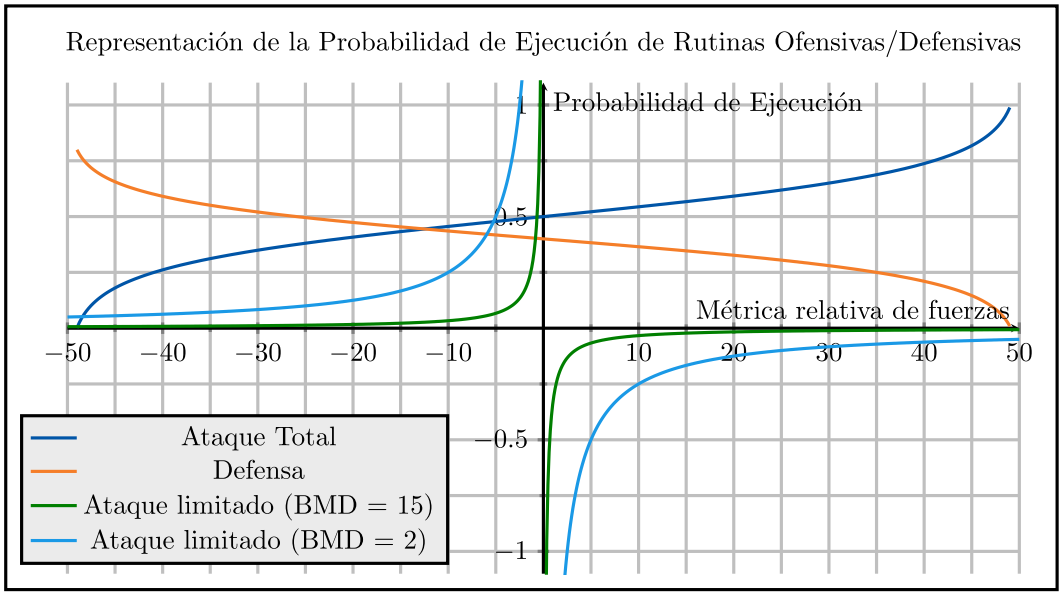
Tabla 3
Comprobación de los valores de probabilidad esperados en cada una de las ecuaciones con los valores especificados

Diferencia	Atacar	Defender	Llimitado contra 15	Limitado Contra 2
25	0,38353994	0,49316805	0,01333333	0,1
23	0,39448158	0,48441474	0,01449275	0,10869565
21	0,40494719	0,47604225	0,01587302	0,11904762
19	0,41501298	0,46798961	0,01754386	0,13157895
17	0,4247425	0,460206	0,01960784	0,14705882
15	0,43418964	0,45264829	0,02222222	0,16666667
13	0,44340091	0,44527928	0,02564103	0,19230769
11	0,45241708	0,43806634	0,03030303	0,22727273
9	0,46127451	0,43098039	0,03703704	0,27777778
7	0,47000617	0,42399506	0,04761905	0,35714286
5	0,47864245	0,41708604	0,06666667	0,5
3	0,48721187	0,4102305	0,11111111	0,83333333
1	0,49574167	0,40340667	0,33333333	0,5
-1	0,50425833	0,39659333	-0,3333333	-0,5
-3	0,51278813	0,3897695	-0,1111111	-0,8333333
-5	0,52135755	0,38291396	-0,0666667	-0,5
-7	0,52999383	0,37600494	-0,047619	-0,3571429

-9	0,53872549	0,36901961	-0,037037	-0,2777778
-11	0,54758292	0,36193366	-0,030303	-0,2272727
-13	0,55659909	0,35472072	-0,025641	-0,1923077
-15	0,56581036	0,34735171	-0,0222222	-0,1666667
-17	0,5752575	0,339794	-0,0196078	-0,1470588
-19	0,58498702	0,33201039	-0,0175439	-0,1315789
-21	0,59505281	0,32395775	-0,015873	-0,1190476
-23	0,60551842	0,31558526	-0,0144928	-0,1086957
-25	0,61646006	0,30683195	-0,0133333	-0,1

Figura 15

Gráfica que representa la probabilidad de ejecución de los casos especificados



De los valores obtenidos se han representado en la Figura 15. En el eje de ordenadas se tiene el valor de la probabilidad de ejecución, del cual los valores más cercanos a 1 tienen prioridad de ejecución y los valores 0 o negativos no se consideran. En contraste, en el valor de abscisas, utiliza como unidad el valor de “Métrica relativa de fuerzas”, este es valor resultante entre la diferencia de unidades propias y enemigas. Se requiere usar este valor porque en este contexto, una cantidad negativa de “unidades controlables” no tiene sentido, pero si una desventaja relativa negativa de unidades.

La línea naranja correspondiente a la defensa siempre ocupa el valor más grande de probabilidad, cuando se presenta una desventaja de -15 unidades o más. De forma opuesta, la línea azul oscuro correspondiente al ataque, siempre es el mayor valor cuando la ventaja es positiva. Ambos resultados son deseables.

Los valores de ataque limitado requerirían de un ajuste mayor. Mientras que la línea azul cian cumple con el resultado deseado de cubrir una diferencia de -5 unidades, la saturación del caso desfavorable, representado en verde, requiere un ajuste más significativo que el pequeño decimal.

En conclusión, la solución, aunque requiere un ajuste mayor al esperado para solucionar la división entre 0 del ataque limitado, proporcionaría una respuesta válida a las necesidades planteadas.

5 Implementación

5.1 Asignaciones iniciales

La decisión inicial al iniciar la implementación se centra en establecer el orden de ejecución de los *scripts* y funciones del programa. Con el fin de asegurar que el programa no tenga fallos, se han concentrado todos los elementos necesarios para el inicio del mapa en un script llamado Player*_Start_Config.*

A partir de este *script*, se pondrán en marcha todos los elementos del mapa, incluido definir el número de jugadores, iniciar scripts auxiliares, asignar materiales o declarar los objetos que administran las economías de los jugadores. Como referencia se incluye un segmento.

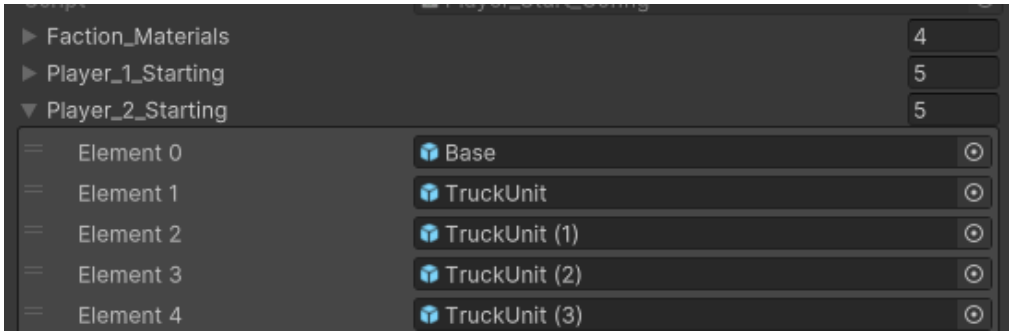
```
public class Player_Start_Config : MonoBehaviour
{
   private void Awake()
   {
      Map_Start_Config.Init_Start_Config(Player_Amount, Faction_Materials);
      Economy_Player.Instance = new Economy_Player[Player_Amount];
      for (int i = 0; i < Player_Amount; i++)
      {
         GameObject gameobject = new GameObject($"Economy_Player_{i}");
         Economy_Player.Instance[i] = gameobject.AddComponent
         <Economy_Player>();
}
```

Otra funcionalidad clave de este *script* es la asignación de las unidades iniciales de cada facción. Es importante que se lleve a cabo desde este punto para evitar situaciones en las que se inician componentes en un orden no deseado.

```
   public void Set_Player_Unit(GameObject[] Player, int number)
   {  foreach (var player in Player)
      {
         player.GetComponent<Faction_Handler>().Change_Faction_OnStart(number);
      }
   }
```

Figura 16

Asignación de las unidades iniciales en el script dedicado a configurar la partida



Con esta implementación, tal y como se puede apreciar en la Figura 16, es suficiente con arrastrar la referencia a la lista, o iniciar el prefab desde las funciones dedicadas a la creación de unidades, para que el bucle for each indicado en el código se encargue de realizar las asignaciones pertinentes.

Mientras que esto solo va a ser usado una única vez en toda la sesión, ya que la mayoría de las unidades van a ser creadas durante la partida, esta utilidad permite realizar cambios en el balance de las posesiones iniciales de cada jugador en un solo instante. Esta característica es relevante mencionarla puesto que, en las versiones iniciales, al no estar presente esta jerarquización podían ocurrir problemas con el orden de las llamadas.

La última funcionalidad que se va a comentar sobre este *script* es que incluye las llamadas que asignan valores a una clase estática llamada Map_Start_Config, de modo que todos los scripts que lo requieran puedan acceder a valores decididos en el inicio, puesto que se pretende que este script inicial no lleve a cabo más acciones, una vez que haya completado esta fase de inicio.

5.2 Sistema de Facciones

Para poder reservar funciones de Unity, como el sistema de capas para usarlo en el sistema de construcción, y poder reservar el sistema de etiquetas para otras funciones, ha sido necesario desarrollar otro sistema que facilite la identificación. Esta situación ha condicionado la creación del sistema de gestión de facciones denominado sistema *faction handler*, desde el que se centralizan estas las peticiones de identidad.

Un caso de aplicación se da en el momento de construir una unidad, en la cual, mediante la ejecución de este *script* ubicado en el edificio, se otorga a la unidad la facción del edificio y se descuenta el costo al jugador apropiado.

5.2.1 Script de gestión de facciones. Faction handler

En este *script* en cuestión, las funciones se dividen en su mayoría en dos tipos. Las funciones que intervienen en el sistema de identificación y las funciones de gestión de visualización de facción.

```
public class Faction_Handler : MonoBehaviour
{
   public Faction Current_faction;
   public Transform Emblem;
   void Start() {
      Render = Emblem.GetComponent<Renderer>();
      Control_Check();
}
```

Current faction es una de las múltiples versiones en las que se compartiría este valor de facción, existiendo otras maneras como *faction number* para compartir únicamente el identificador sin necesidad de proporcionar todo el objeto.

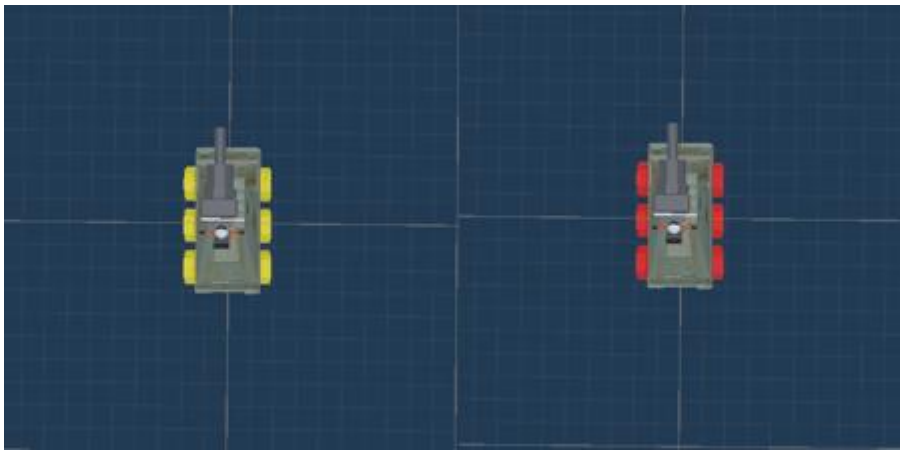
Las funciones que realiza se pueden clasificar según su objetivo, en 3 categorías:

- **Solicitar la facción:** Interviene en interacciones, por ejemplo, determinar si otra unidad que ha entrado en el rango es hostil o no.

- **Asignar una facción:** Se usa en la creación de unidades.
- **Cambiar la facción:** Mayoritariamente usadas para el sistema de captura.

En lo relativo al sistema de la asignación de los materiales visuales, su gestión se usa principalmente sobre un componente denominado emblema. Este contiene los elementos del modelo que van a ser modificados para permitir la identificación.

Figura 17
Asignación de color en una unidad, visible por el cambio en las texturas del modelo



Tal y como se puede apreciar en la Figura 17, el componente emblema se encuentra situado en las ruedas. Se han pintado las texturas sobre estas, preservando el resto del modelo sin alterar. Específicamente se habría pintado del color amarillo de la facción neutral a el color de la facción roja.

Un aspecto importante de esta implementacion recae en que toda entidad controlable, dentro del tablero, necesita tener una facción asignada en todo momento. Porque si no es asi, muchas funcionalidades simplemente no podrían ejecutarse porque se estaria esperando este valor.

Para asegurar que este es el caso y evitar errores, dentro del *script* existe una función denominada *control check*. Esto garantiza que si no se le otorga una facción en creación, por defecto se le asigne la faccion neutral.

Hay una razón por la cual la opción por defecto no fue establecer la unidad como neutral y que se modifique al momento de construcción. Esto es debido a que en esta implementación, el unico motivo válido para pertenecer a la facción neutral, es que la entidad es capturable, como seria el caso de la base o que comience siendo neutral al inicio del mapa. Fuera del caso mencionado, es una medida de seguridad para contener errores más graves, pues si se ha requerido, significa que algún otro sistema ha fallado.

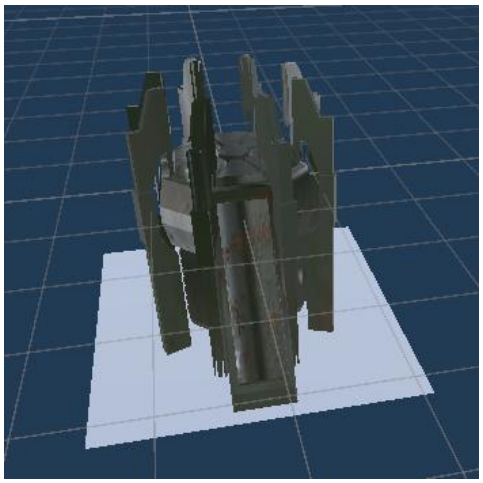
5.3 Edificios no capturables

Una vez el sistema de facciones permite asignar la pertenencia de las unidades al jugador, el siguiente paso radica en proporcionarle una forma de crear entidades sobre el espacio de juego. Tratando este espacio como un tablero, dividiremos este tablero en cuadrados, los cuales se comprobará si están ocupados o no.

Mientras Unity ya cuenta con funcionalidades para implementar un tablero con una funcionalidad que denomina *grid*, se ha optado por simplemente dividir el tablero en partes iguales durante la creación, generando una cuadrícula invisible. Esto se debe a que solo se necesita para comprobar si el espacio se encuentra disponible. Como no se van a utilizar el resto de las funciones, se apreciaba como una solución sobredimensionada. Para implementar esta decisión, todos los edificios cuentan con una plataforma que comprueba los aspectos decididos durante la fase de diseño respecto a la validez de la ubicación en la cuadrícula. Será visible mientras esté seleccionado el edificio dentro de la opción de construcción.

Figura 18

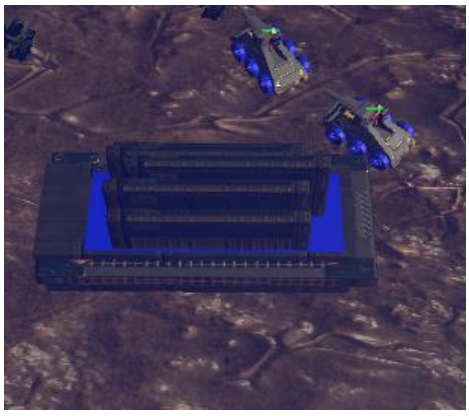
Edificio de tipo mina con la plataforma de construcción con la visibilidad permitida



Esta plataforma es la que impide la construcción sobre las cuadrículas ya ocupadas. Como se puede ver en la Figura 18, esta plataforma cubre todo el edificio. Esto hace que cualquier intento de construir, sobre él, sería considerado no valido, ya que la plataforma de este segundo edificio detectaría la colisión con la plataforma del ya presente. Una vez seleccionado desde el menú de construcción, se iniciará el proceso de construcción. Trascurrido este tiempo, el edificio pasa a estado operativo.

Figura 19

Edificio de tipo fábrica sobre el terreno. La plataforma ya no se encuentra visible



Se puede observar en la Figura 19 otro ejemplo de edificio. Este ya construido y funcional. La parte que actúa de emblema, que se localiza en la parte superior para

facilitar la vista aérea, ha adquirido los colores de la facción que lo ha construido, los mismos presentes en el tinte de color azulado del suelo. Al finalizar este procedimiento, si el usuario posee los suficientes recursos, se le permite comenzar a generar unidades desde este edificio. Para lograr esto, debe elegir el edificio y desde la interfaz de usuario, pulsar el icono correspondiente a la unidad.

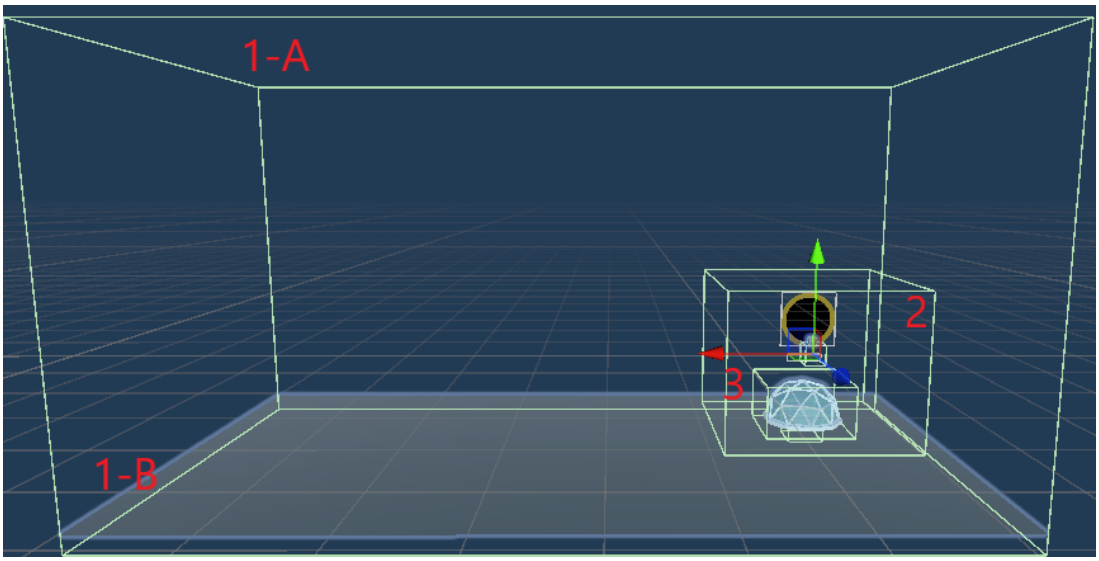
5.4 Bases y sistema de captura

Tal y como se explicó en la fase de diseño, el mapa está dividido en unas zonas de control, las cuales están presididas por una base que gestiona el proceso de su captura.

Dado que estos dos elementos deben ir juntos, se ha optado por el desarrollo de diferentes tipos de formas de áreas, incluyendo su base, para poder adaptarse al mapa según las necesidades, en un enfoque de crear unos diseños preestablecidos y arrastrarlo sobre el terreno. La razón para esta implementación se basa en el hecho de que, aunque este mapa no presenta complicaciones al distribuir las áreas, según el tipo de terreno, en otros mapas el diseño resultaría demasiado repetitivo si se limita únicamente a cuadrados.

Para ilustrar mejor el concepto, se mostrará cómo se representaría un caso de una base junto a su zona de influencia. En este ejemplo, se diseña una forma rectangular. Si consideramos cuadrados, esto se traduciría en una forma 2x1. Dentro de este enfoque, sería viable tener zonas con configuraciones en forma de letra "T" o letra "L", sin quedar restringido a algo simétrico. Dicho esto, si el mapa lo requiriera se podría tener una forma menos regular aún, pero requeriría ser revisado acordemente.

Figura 20
Elementos que componen el sistema de base incrustada en una zona de influencia



En la Figura 20 podemos apreciar los siguientes elementos mínimos que conforman el conjunto base con una zona de influencia y serán explicados a continuación.

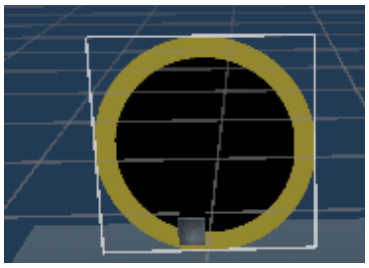
- **Influencia**: La combinación de la plataforma más el área que rodea toda la entidad, que visualizamos como el rectángulo verde de gran tamaño que

envuelve el conjunto, en la figura 20 está representado con los indicadores "**1-A**" y "**1-B**". Su función principal es delimitar el espacio donde el usuario puede edificar. Además, este *colider* es el que permite conocer dónde están presentes las unidades en el tablero.

- ***Check(comprobación)***: Área que envuelve la base, que visualizamos como el cubo de bordes verdes de menor tamaño que el del punto anteriormente mencionado, en la figura 20 está representado con el indicador "**2**". Esta zona comprueba qué unidades interactúan con la base, y por ello en el proceso de captura.
- **Base**: Estructura visible que supone el edificio que permite interactuar directamente en este bloque, en la figura 20 representado con el indicador "**3**". En esta implementación se usa un edificio en forma de cúpula, pero puede perfectamente usarse otro modelo para adaptarse a la temática del nivel.
- **Indicador de captura**: Representación visual del progreso de captura. Este elemento se muestra por separado en la Figura 21, para poder visualizarse correctamente. En su funcionamiento, este cambia de color, en sentido horario, comenzando desde la parte que corresponderia a las 6, en formato analogico. Esto se realiza hasta que se completa la circunferencia con el color del jugador que lidera la captura.

Figura 21

Indicador visual del progreso del estado de captura que presentan las bases



Una vez introducidos los componentes, se explica brevemente el sistema de captura. Las bases poseen una característica única conocida como el estado de captura. Esto significa que cuando su cantidad de vida llega a 0, en lugar de ser destruidas, entran en un estado denominado modo de captura.

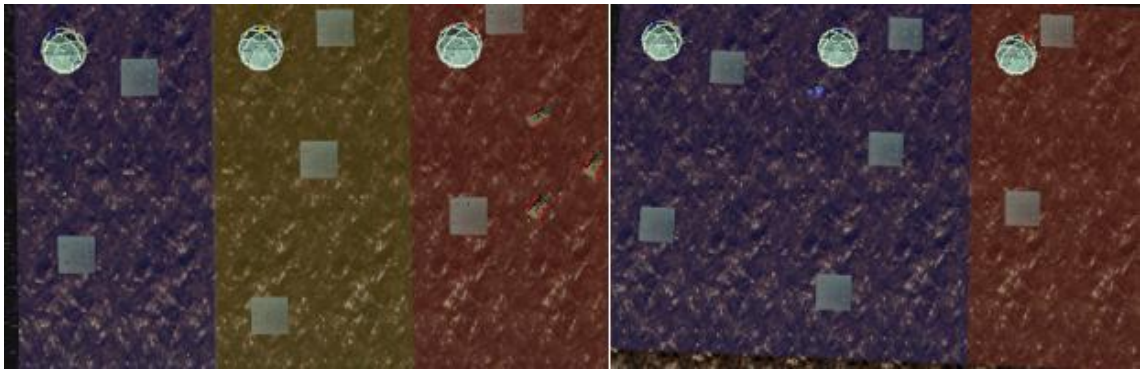
Todas las unidades que se encuentran en la zona *check* son elegibles para optar a apoderarse de la base, asignándola al jugador que controla estas unidades. Este progreso puede ser observado a través de un indicador de captura. Las condiciones son las siguientes:

- Si solo hay unidades de una sola facción, el contador aumenta.
- Cuando hay unidades de dos o más facciones, la captura se detiene.
- Si una facción que estaba liderando la captura y no tiene unidades dentro de la zona de verificación, su progreso disminuye. Si este contador vuelve a 0 otra facción presente puede optar por liderar la captura.
- Cuando una facción logra llenar el indicador, completa la captura y se asigna la base a su equipo.
- Si no hay ninguna facción presente, el contador simplemente regresará a 0.

Una vez una facción ha logrado completar el proceso de captura, la base pierde el estado capturable y puede usar la zona de influencia para construir.

Figura 22

Demostración visual del resultado del usuario participando en el proceso de captura



En la Figura 22, se aprecia una vista aérea del mapa. En esta situación específica, el usuario cuya facción tiene el color azul ha capturado la base neutral de color amarillo, aumentando su número total de posiciones.

5.5 Comportamientos generales de las unidades

La estructura de datos elegida para respaldar el sistema estadístico se fundamenta en la utilización de *scriptable objects*. Esta configuración ofrece un elevado nivel de uniformidad ya que todas las entidades comparten las mismas estadísticas fundamentales. Esto realiza la idea planteada de que si un valor no se utiliza se le asigne 0. Esto permite evitar errores relacionados con valores nulos inesperados, asegurando que todas las solicitudes generen al menos un valor válido.

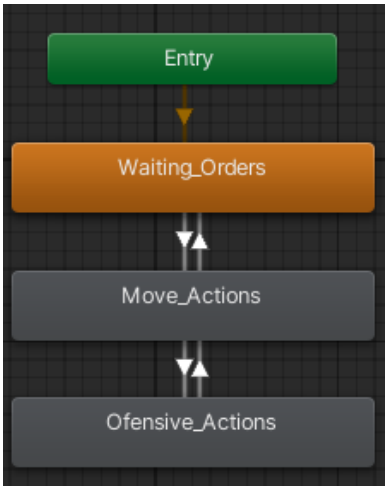
Figura 23

Scriptable object presente en todas las unidades, con valores particularizados

Como se muestra en la Figura 23, esto permite acceder rápidamente y almacenar, de manera ordenada, la información de todas las unidades. Además, al centralizar la información, nos evita tener que predefinir valores en otras mecánicas de la unidad, como seria el rango de ataque que utiliza la unidad en la máquina de estados.

Figura 24

Representación de los estados de la unidad mediante la herramienta animator de Unity



En la fase de diseño se consideró abordar el comportamiento de las unidades como máquinas de estados finitos. Puede observarse en la Figura 24, una herramienta con la que cuenta Unity denominada "*animator*", que además de poder asistir en la gestión de estos estados, nos permite añadirles funcionalidades, como animaciones o efectos visuales, para la transición entre los estados.

En este caso específico, dentro del estado *waiting_orders,* hasta que se reciba una orden, la unidad está leyendo el sistema de colisiones para detectar objetivos, lo que se podría interpretar como un estado de alerta. Cuando se detectan unidades enemigas que deben ser consideradas como objetivos, o se les asigna un objetivo desde una orden externa, se activa el estado *move_actions.*

Tal y como se comentó en el apartado de diseño, para crear ordenes más complejas, se combinan permisos con los estados. El estado *Move_actions*, es el estado mayormente afectado por estos. Modificando sus permisos desde desplazarse libremente a limitar el rango de movimiento de una forma específica, se crean conductas.

Si no se limita, por defecto la actitud es perseguir. Si, por ejemplo, se limita con la prohibición de alejarse, de más de una distancia especifica de un punto, se ha añadido la instrucción de mantener una ubicación. Esto significa que, si se necesita ocupar o proteger un área, solo atacará a otras unidades, siempre que respete la distancia máxima.

Estas conductas son importantes para el flujo de la partida. Si no se puede ordenar que las unidades mantengan una posición, sería imposible utilizar el sistema de captura, pues este requiere mantener unidades en puntos específicos, para realizar las comprobaciones de qué unidades están presentes.

El estado *ofensive_actions* gestiona el ataque, realizando las llamadas al *script* que gestiona los proyectiles asignándoles la información necesaria, tanto la posición relativa del emisor y receptor, los valores para los cálculos de daño y los comportamientos del proyectil. La forma de interactuar con este estado es lo que define la mayoría de las diferencias entre unidades, tanto en su forma de rango, trayectoria,

caída de proyectil entre otros y permite desarrollar comportamientos específicos que caractericen a las unidades.

5.5.1 Ejemplo de comportamientos específicos de unidad

Modificando las características de la unidad, se trata de formar tipos de conductas especiales. Un ejemplo de este tipo de comportamiento diseñado es el comportamiento de artillería.

En su implementación, se mezcla un enfoque basado tanto en las en estadísticas como en las mecánicas. Se les proporciona a estas unidades un rango mayor al habitual, pero el tiempo en el aire del proyectil también es mayor al habitual, lo que permite a las unidades particularmente rápidas esquivarlos. Esto genera una segunda capa de efectividades entre unidades, que incentiva la interactividad por parte del usuario.

Para lograr este este efecto, se ha experimentado con proporcionarle a las unidades de artillería un ataque que realice una curva en forma de parábola, lo que hace que el tiempo adicional de impactar contra el objetivo tenga visualmente sentido. Se requiere mencionar que para asegurar el aspecto de parábola se necesita condicionar al menos un aspecto, ya sea una distancia mínima con el objetivo o predefinir la altura a la que llegará el proyectil, aunque esto signifique que puede comprimirse.

5.6 Economía

La gestión del sistema de recursos requiere que cada participante tenga un objeto específico para la gestión de su economía. Este se asigna al inicio de la generación del mapa.

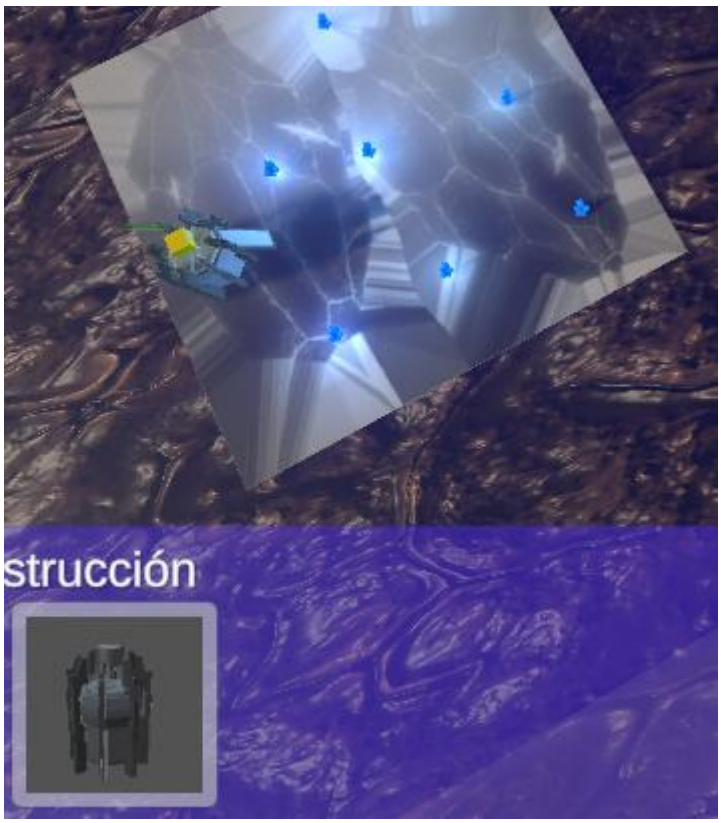
Este objeto se utiliza mediante un *script* que maneja los recursos. Los elementos se alojan en una lista de recursos, junto con las funciones necesarias para gestionar estos elementos. Entre dichas funciones se encuentran aquellas encargadas de añadir recursos, verificar si se poseen los recursos suficientes para realizar compras y, en caso de realizar una compra, consumir los recursos correspondientes.

Para el jugador humano, este también contiene las utilidades necesarias para mostrar los cambios de recursos en la interfaz gráfica. Cada vez que se añaden o sustraen recursos, estos se reflejan en la parte superior de la interfaz de usuario.

La interacción con el aspecto de la economía de recursos se realiza mediante edificios de la categoría de “edificios económicos”. Estos contienen las funciones necesarias para generar recursos al jugador. Dependiendo del edificio especifico, puede incluir desde ingresos pasivos a lo largo del tiempo a la construcción y coordinación de unidades civiles que extraigan los recursos.

Figura 25

Ejemplo visual del usuario, seleccionando dónde situar el edificio de tipo mina



Como observamos en la Figura 25, para interactuar con este sistema, el usuario debe acceder al menú de edificios económicos y seleccionar el del edificio deseado en el menú de construcción haciendo clic en el icono. También se puede apreciar que el edificio aún está en modo selección, porque el indicador de facción aún tiene los colores neutrales.

Dentro de este modo de selección, cuando el usuario haga *click* con el botón izquierdo sobre una superficie válida, que en este caso corresponde al plano con textura cristalizada, adquiere los colores del usuario e inicia la construcción. En este caso específico de la mina, vinculándose a ella mientras el edificio esté presente.

Una vez se complete la construcción, al seleccionar la estructura, se activará el icono que permite producir unidades que extraen los minerales. La vinculación con el plano permite leer valores como el tamaño asignado al plano. Este valor determina la cantidad de mineros que pueden trabajar simultáneamente en esta mina. Otro aspecto importante es que cada plano de mineral solo admite una mina a la vez.

Figura 26

Interfaz de creación de unidades que se muestra al seleccionar una mina funcional

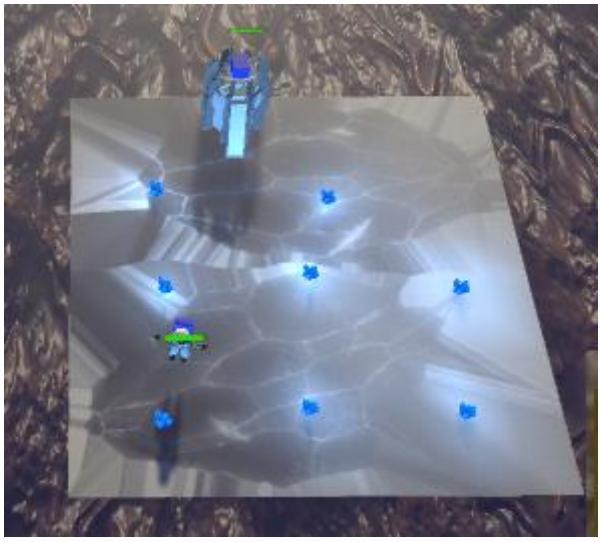
Nota: icono para representar a las unidades económicas extraídas de la página web Freepik, con referencia "Jackhammer Icon"

En la Figura 26 se aprecia el menú que se muestra en la interfaz de usuario. Este permite generar las unidades denominadas mineros si se dispone de los recursos necesarios para ello.

Estas unidades civiles son lo suficientemente distintas para ser unidades especiales, pero siguen teniendo los mismos tres estados de las unidades, aunque modificados. En su estado *waiting_orders*, en lugar de esperar una orden de ataque o buscar objetivos hostiles, buscan en su rango, como objetivo, una mina la cual tenga espacio en su contador de unidades simultaneas.

Figura 27

Unidad de tipo civil económica desplazándose para recolectar recursos



Como permite observar la Figura 27, la unidad minera se encuentra en su estado move_Actions que le hace desplazase hasta uno de los puntos destacados del plano. Una vez se encuentre adyacente a unos de los puntos brillantes, se detendría frente a este, para realizar su equivalente a *ofensive_actions.* Aquí en lugar de atacar a otra unidad, comienza a minar hasta que su contador interno se llene de recursos.

Una vez el contador de recursos se encuentra en el valor máximo, regresa al estado move_Actions, el cual, por la condición del contador, en lugar de dirigirse a otro

punto de recursos, volverá a la mina a vaciar su reserva. Una vez complete esto, se repetirá el ciclo.

Figura 28
Ejemplo de edificio de tipo económico denominado "Panel solar"

Para finalizar la sección se destaca que no todos los edificios económicos operan de la misma manera. Tal y como puede apreciarse en la Figura 28, el edificio denominado panel solar usa las normas estándar de construcción, lo que implica que no se ha construido sobre un terreno especial. De igual manera, no requiere unidades civiles para ser operado.

5.7 Sistema de utilidad en la toma de decisiones de los jugadores no humanos

La forma en que se lleva a cabo la implementación consiste en segmentar la toma de decisiones en varios pares de *scripts*. Estos puntos se denominan núcleos, ya que funcionan como centros para consolidar funciones específicas. Los elementos de estos pares son los siguientes:

El primer elemento es un script que monitoriza unos elementos específicos del juego. Esto incluye entre otros, leer variables directas de los jugadores o determinar cuál es la base con menos unidades hostiles en el mapa. Además, este elemento también se encarga de seleccionar los elementos que cumplirán las ordenes como, por ejemplo, seleccionar lar unidades que no tienen una orden asignada.

El segundo elemento es una lista de decisiones. Este se encarga de evaluar la deseabilidad de las decisiones, aplicando la información extraída del script de monitorización a las ecuaciones de consideración que contiene cada decisión.

En esta implementación, solo está presente un núcleo, correspondiente a aspectos tácticos, el cual se encarga de mover las unidades por el mapa con el fin de alcanzar objetivos. En una posible versión definitiva, se requerirían núcleos contemplando otras funciones, como, por ejemplo, uno enfocado a decisiones económicas.

5.7.1 Aplicación a una decisión especifica

Antes de realizar la acción, se requiere que las utilidades le proporcionen la información necesaria para asignar el valor de la decisión.

Cada curva de consideración realiza las peticiones de los valores que toma en cuenta, con una llamada *get value.* En este caso, se usa la fórmula dedicada a la función de ataque, basada en la diferencia de potencial entre facciones en base a la cantidad de población.

```
counter = ResourceManagerPlayer.Instance[i].get_specific_resource(i);
players_army[i] += counter;
```

Con los valores necesarios se hace el cálculo. Una vez realizado, si resulta la opción ganadora, se ejecuta la respuesta.

```
public override void Next_Action(Data data)
{
    Data.Tactic_Data.Get_attack_Points();
    Data.Tactic_Data. Call_units()
}
```

Una vez ha sido calculado y se tiene que se encuentra en una situación de fortaleza, se comprueba cuál es la base óptima para ser atacada.

```
public void Get_attack_Points()
{
    this.vulnerable_Base = Manage.Is_Defended(2,out vulnerable_number);
}
```

Al decidir llevar a cabo el ataque, se identifica cuál es la base con menor defensa para optimizar el resultado. Para esto, se revisa el listado de todas las bases y se selecciona como objetivo aquella que tenga menos unidades defendiéndola.

```
public Transform Is_Defended(int my_faction, out int num)
{
    int count=0;
    int this_defense = 0;
    int defended = 999;
    Transform least_defended = null;
    foreach (var listaInterna in Capture_List.ClaimedBy)
    {
        if (count != my_faction)
        {
            foreach (var elemento in listaInterna.Capturable)
            {
                this_defense =
elemento.transform.parent.GetComponent<Capture_Handler>().Check_Oposition(my_faction);
                if (this_defense < defended)
                {
                    defended = this_defense;
                    least_defended = elemento.transform;
                }}}
```

```
        count++;
    }
    num = defended;
    return least_defended;
}
```

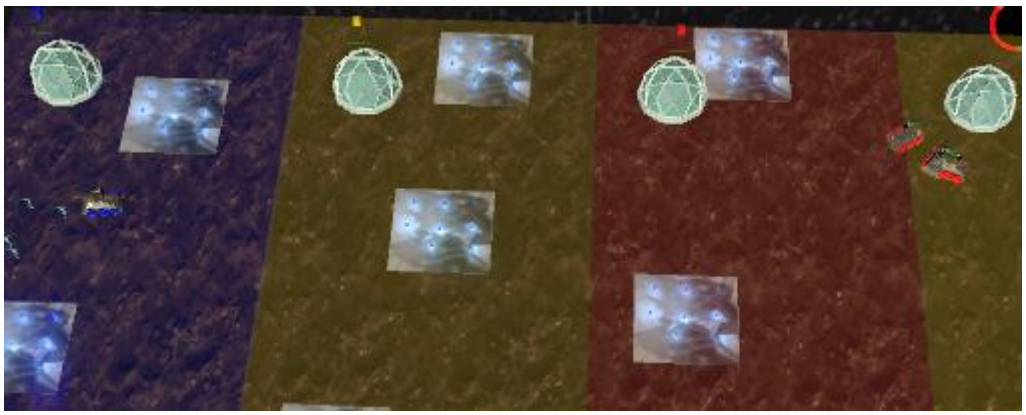
Una vez está decidida la base, son movilizadas las unidades que no estén ocupadas realizando otra función. Esto se lleva a cabo proporcionando la orden a las unidades que no tengan un objetivo asignado.

```
public void Call_units()
{
    foreach (Transform unit in MyUnits)
    {
        if (!(unit.GetComponent<AttackController>() == null) && vulnerable_Base != null)
        {
            if (unit.GetComponent<AttackController>().targetToAttack == null &&
unit.GetComponent<AttackController>().target_Order== null)
            {
                unit.GetComponent<AttackController>().Atack_Order(vulnerable_Base);
            }
        }
    }
}
```

Se realiza una comprobación visible en la Figura 29. La prueba dio como resultado que la decisión tomada por la lógica fue desplazar las unidades iniciales y tomar una de las bases desprotegidas. Esto se repetiría mientras el usuario no moviese las unidades de su zona, dándole la consideración de poco defendida.

Figura 29

Comprobación de la decisión tomada por la lógica de facción y su ejecución



6 Debug

Se han incluido dos *scripts* de pruebas para realizar las comprobaciones de forma ágil y rápida.

El primero contendría el equivalente a formas extremadamente desbalanceadas de interactuar con el entorno, para comprobar si se dan interacciones no deseadas, a modo de ejemplo, se muestran las utilidades de recursos.

En el siguiente *script*, al pulsar una tecla definida, se proporcionaría la cantidad de recursos determinada.

```
public class DebugUtilites : MonoBehaviour
{
    [SerializeField] public bool DEBUG_MODE = true;
    [SerializeField] private KeyCode key_AddResources = KeyCode.Insert;
    [SerializeField] private int[] resources2add = { 500, 500, 500, 500};

    void Update()
    {
        int p1 = (int)PlayerAdmin.players_type.Player_1;
        if (DEBUG_MODE)
        {
            if (Input.GetKeyDown(key_AddResources))
            {
                ResourceManagerPlayer.Instance[p1].add_resources_array(resources2add);
            }
        }
    }
}
```

El segundo tipo de comprobación es más específica, para poner estrés sobre las comprobaciones de las funcionalidades, para asegurar que actúan correctamente. En este caso, el siguiente *script* pretende construir un edificio en una localización aleatoria.

```
public class Building_test : MonoBehaviour
{
    [SerializeField]
    public GameObject objetoPrefab;
    public GameObject objetoPlaceHolder;
    public KeyCode teclaInstanciar = KeyCode.K;
    public float rangoMinXZ = 0f;
    public float rangoMaxXZ = 100f;

    private void Update()
    {
        if (Input.GetKeyDown(teclaInstanciar))
        {
            while (!Instanciar_en_posicion_Aleatoria()) ;
            actualizar_NavMesh(1f);
```

```
        }
    }

    private bool Instanciar_en_posicion_Aleatoria()
    {
        Vector3 posicionAleatoria = ObtenerPosicionAleatoria();
        bool construible = Posicion_valida(posicionAleatoria, objetoPlaceHolder);
        if (construible)
        {
GameObject objetoInstanciado = Instantiate(objetoPrefab, posicionAleatoria,
Quaternion.identity);
        }
        return construible;
    }
    private Vector3 ObtenerPosicionAleatoria()
    {
        float x = Random.Range(rangoMinXZ, rangoMaxXZ);
        float z = Random.Range(rangoMinXZ, rangoMaxXZ);
        return new Vector3(x, 0, z);
    }
}
```

7 Conclusiones

Para llevar a cabo el diseño e implementación del prototipo de un nivel de videojuego *RTS* en la plataforma Unity, se ha realizado un estudio sobre este tipo de producto. El objetivo ha sido analizar las necesidades de elementos esenciales, como unidades, edificios, sistemas de recursos, rivales y otros sistemas que influyen en el flujo de la partida, como el sistema de captura específico en este juego, así como proponer soluciones para su implementación.

Uno de los factores que ha llevado a elegir Unity sobre otras opciones comerciales es la experiencia previa adquirida con esta herramienta durante la carrera. Aunque no se ha planteado como un objetivo, se ha tratado de utilizar elementos estándar comunes en la mayoría de los motores de desarrollo. Esto permite que estos elementos sean fácilmente aplicables en otros motores y evita la situación en la que, si la compañía deja de ofrecer soporte al motor, algunas partes se vuelvan obsoletas. Un ejemplo de estas herramientas es el paquete NavMesh, utilizado para la navegación de unidades, que presenta equivalentes directos en otros motores de desarrollo, como “Unreal”, “CryEngine” y “Godot”, los cuales cuentan con complementos de navegación pathfinding mediante el uso de un NavMesh.

Adicionalmente, para las mecánicas más sencillas, la opción tomada fue implementar las soluciones directamente mediante código, en lugar de utilizar funcionalidades descargables. Esta decisión se ha tomado, en primer lugar, para tener una ocasión de aplicar los conocimientos adquiridos en la carrera, y, en segundo lugar, para mantener el proyecto lo más replicable posible. Un ejemplo de esto es la implementación de las unidades como máquinas de estados, diseñando cómo interactúan con los sistemas del juego en lugar de descargar un paquete de terceros que ya incluya estos comportamientos y funcionalidades.

Respecto a la posibilidad de ampliar las ideas mostradas en este prototipo, en caso de desear elevarlo al estándar de un producto comercializable, existen considerables limitaciones que podrían superarse con un equipo multidisciplinario. Esto incluiría aumentar el cuerpo de programación para añadir funcionalidades que mejorarían significativamente la calidad, así como contar con personal con habilidades artísticas para mejorar el aspecto visual. Otra opción valiosa sería incorporar personal con capacitad para adaptar y comercializar el producto a nivel global.

Finalmente, a nivel personal, este proyecto ha supuesto un reto mayor al esperado inicialmente. En esta implementación, los elementos presentes interactúan entre sí de manera mucho más interconectada a la que se estimó, lo que ha generado problemas en la gestión de interacciones no deseadas entre mecánicas. Un ejemplo de esto es la interacción entre el sistema de navegación y el sistema de construcción, ya que se debe tener en cuenta la posibilidad de que el estado del escenario puede cambiar considerablemente en muy poco tiempo.

8 Bibliografía

Age of Empires II: Definitive Edition en Steam. (s. f.). https://store.steampowered.com/app/813780/Age_of_Empires_II_Definitive_Edition/?l=spanish

Apperley, T. (2006). Genre and Game Studies: Toward a Critical Approach to Video Game Genres. Simulation and Gaming. https://doi.org/10.1177/1046878105282278

Ben-Ari, Mordechai & Mondada, Francesco. (2018). Finite State Machines. *Elements of robotics*, 55-61.

Definition of **strategy** from the Cambridge Advanced Learner's Dictionary & Thesaurus © Cambridge University Press

Freepik. (2020, 4 junio) Jackhammer [*Jackhammer Icon*]. Flaticon. https://www.flaticon.com/free-icon/jackhammer_3061424

Kamiński, B., Jakubczyk, M., & Szufel, P. (2018). A framework for sensitivity analysis of decision trees. *Central European journal of operations research*, *26*(1), 135–159. https://doi.org/10.1007/s10100-017-0479-6

Kei Ito and Masataka Okabe, August 2002) Color Universal Design (CUD) - How to make figures and presentations that are friendly to Colorblind people - https://jfly.uni-koeln.de/color/

Poly Angel - Space Pack | 3D Space | Unity Asset Store. (2025, 15 junio). Unity Asset Store. https://assetstore.unity.com/packages/3d/vehicles/space/poly-angel-space-pack-267010

StarCraft II. (s. f.). https://starcraft2.blizzard.com/en-us/

SteamDB. (2019). Age of Empires II: Definitive Edition. Retrieved (July 2, 2024), from https://steamdb.info/app/813780/

Supreme Commander 2 on Steam. (s. f.). https://store.steampowered.com/app/40100/Supreme_Commander_2/

Świechowski, M., Lewiński, D., & Tyl, R. (2021, December). Combining utility ai and mcts towards creating intelligent agents in video games, with the use case of tactical troops: Anthracite shift. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1-8). IEEE.

Translucent Crystals | 3D Fantasy | Unity Asset Store. (2022, 8 octubre). Unity Asset Store. https://assetstore.unity.com/packages/3d/environments/fantasy/translucent-crystals-106274

Try Evidence Research Team (2021, 15 10). *REPORT: Women in Games–Insights on Female Gamer Dynamics https://tryevidence.com/blog/lreport-women-in-games-insights-on-female-gamer-dynamics/*

Unity Technologies. Publication (2019.4)
https://docs.Unity3d.com/es/2019.4/Manual/nav-BuildingNavMesh.html

Yughues Free Metal Materials | 2D Metals | Unity Asset Store. (2013, 22 noviembre). Unity Asset Store. https://assetstore.unity.com/packages/2d/textures-materials/metals/yughues-free-metal-materials-12949

9 Anexo: Relación del trabajo propuesto con los ODS

Tabla 4

Tabla referente a los objetivos de desarrollo sostenible y su relación con este proyecto

	Alto	Medio	Bajo	No Procede
Fin de la pobreza				X
Hambre cero				X
Salud y bienestar				X
Educación de calidad				X
Igualdad de género				X
Agua limpia y saneamiento				X
Energía asequible y no contaminante				X
Trabajo decente y crecimiento económico			X	
Industria, innovación e infraestructuras			X	
Reducción de las desigualdades		X		
Ciudades y comunidades sostenibles				X
Producción y consumo responsables				X
Acción por el clima				X
Vida submarina				X
Vida de ecosistemas terrestres				X
Paz, justicia e instituciones sólidas				X
Alianzas para lograr objetivos				X
Descripción de la alineación del TFG/M con los ODS				X

Nota: Tabla extraída del formulario requerido sobre los ODS para la publicación del TFG, proporcionada por la Universidad Politécnica de Valencia.

Se sitúa en la categoría “bajo” de los ODS, “Trabajo decente y crecimiento económico” e “Industria, innovación e infraestructuras” puesto que, de la misma forma que se ha mencionado en la sección de introducción del TFG; se ha planteado en la industria del videojuego como un generador de oportunidades de empleo y como un motor del I+D+I por las inversiones que se generan.

Se sitúa en la categoría “media” de los ODS, “Reducción de las desigualdades”, ya que el proyecto contempla formas de acomodar a usuarios con un perfil de necesidades distinto.