
**Universidad de Valladolid**

## Escuela de Ingeniería Informática de Valladolid

### TRABAJO FIN DE GRADO

**Grado en Ingeniería Informática**

**Mención en Ingeniería del Software**

# Desarrollo de un videojuego educativo con agentes inteligentes en Godot

Autor:
**D. Sergio Rodríguez Casado**

Tutor:
**D. Luis Ignacio Jiménez Gil**

# Resumen

La principal contribución de este Trabajo Fin de Grado TFG ha sido el desarrollo de un videojuego educativo llamado *Maze2D* basado en laberintos que permitan ejemplificar los métodos de búsqueda. La motivación original de este desarrollo, parte de la solicitud del tutor de este trabajo de crear una herramienta que sirva para ejemplificar conceptos mostrados en la asignatura de Fundamentos de Inteligencia Artificial (FIA), impartida en el segundo curso del Grado de Ingeniería Informativa de la Universidad de Valladolid (UVa). En particular, el requisito principal fue la de ejemplificar el funcionamiento de los métodos de búsqueda en el espacio de estados de forma visual y atractiva para los estudiantes de esta asignatura, complementando los métodos educativos empleados en la enseñanza teórica y practica para este contenido temático. El desarrollo del videojuego se ha llevado a cabo siguiendo una planificación temporal que permite la administración recursos y ajuste a los periodos de entrega que se han establecido, reflejando la realización de cada una de las tareas. Además, se ha llevado a cabo el análisis del producto y el diseño de componentes y características propias del videojuego, determinando los objetivos y acciones necesarias para que el producto desarrollado sea completo y funcional. Como aspecto adicional, se ha optado utilizar *Godot*, un motor para el desarrollo de videojuegos gratuito y de código abierto que permite crear juegos 2D y 3D con un potente sistema de nodos, un lenguaje de *scripting* propio (*GDScript* basado en *Python*) y con soporte para múltiples plataformas. Esta elección está fundamentada en explorar otras opciones diferentes a *Unity Engine* como base para el desarrollo de juegos educativos para la plataforma *Gamispace*, proyecto del GRupo de Estudio en Innovación Docente en Informática (GREIDI) del Departamento de Informática de la UVa.

**Palabras clave:** Desarrollo de videojuegos, inteligencia artificial, búsqueda en el espacio de estados, gamificación, Godot Engine.

# Abstract

The main contribution of this Final Degree Project has been the development of an educational videogame called Maze2D based on labyrinths that allow the exemplification of the search methods. The original motivation for this development stems from the request of the tutor of this work to create a tool that serves to exemplify concepts shown in Fundamentos de Inteligencia Artificial (FIA) course, of the 2nd year of the degree of Computer Engineering at the Universidad de Valladolid. In particular, the main requeriment was to exemplify the technique of the states space search methods in a visual and atracctive way for the students of this subject, complementing the educational methods used in the theorical and practical teaching of this thematic content. The development of this videogame has been carried out following a time planning that allows the management of resources and adjustment to the delivery periods have been established reflecting the completion of each of the tasks. In addition, the product analysis and the design of the components and features of the videogames have been carried out, defining the objectives and actions necessary for the developed product to be complete and functional. As an additional feature, we have been dedided to use Godot, a free and open source videogame develompent that allows the creation of a 2D and 3D games with a powerful nodes system, a own spripting lenguaje (GDScript based on Python) and support for multiple platforms. This choice is based on exploring other options than Unity Engine as a basis for development of educational games for the Gamispace platform, a project for Grupo de Estudio en Innovación Docente en Informática (GREIDI) of the Department of Computer Science of the University of Valladolid.

Keywords: Game development, artificial intelligence, state space search, gamification, Godot Engine.

# Índice general

# Índice de figuras

# Índice de tablas

# Agradecimientos

Quiero agradecer a mi familia, tanto a mis padres Mari Carmen y Fausto como a mis hermanos Alberto y Sandra por apoyarme durante esta etapa, y a mi perro Kovu, por hacerme compañía durante tantas horas estos meses.

También quiero agradecer ami tutor por la ayuda ofrecida durante el desarrollo de este TFG.

# Glosario

**API** Interfaz de Programación de Aplicaciones.

**CU** Caso de Uso.

**ECS** Entidad-Componente-Sistema.

**FIA** Fundamentos de Inteligencia Artificial.

**GREIDI** GRupo de Estudio en Innovación Docente en Informática.

**GPS** Sistema de Posicionamiento Global, del inglés *Global Positioning System*.

**GUI** Interfaz Gráfica de Usuario, del inglés *Graphical User Interface*.

**IA** Inteligencia Artificial.

**MVC** Modelo-Vista-Controlador.

**NPC** Non Playable Character.

**RL** Aprendizaje por refuerzo, del inglés *Reinforcement Learning*.

**SVG** *Scalable Vector Graphics*.

**TFG** Trabajo Fin de Grado.

**TIC** Tecnologías de la Información y la Comunicación.

**UML** Lenguaje unificado de modelado, del inglés *Unified Modeling Language*.

**UVa** Universidad de Valladolid.

**VSC** *Visual Studio Code*.

# Capítulo 1

# Introducción

## 1.1. Contexto y Motivaciones

La Inteligencia Artificial (IA) es una disciplina científica que tiene como objetivo replicar las capacidades cognitivas humanas en sistemas computacionales, es decir, recrear programas informáticos, que a través de las matemáticas y la lógica simulan el razonamiento de las personas, permitiendo aprender nueva información y la toma de decisiones de forma autónoma[1]. Para lograr este propósito, se basa en el análisis de datos y en la experiencia acumulada. Las predicciones y acciones generadas por estos sistemas se fundamentan en el procesamiento de grandes conjuntos de datos utilizando los aciertos y errores previos como base para optimizar su aprendizaje y mejorar el desempeño generado. La IA desempeña un papel clave en la resolución de problemas complejos[2], así como la adaptación continua a nuevas situaciones, por lo que es muy útil en escenarios complejos y una herramienta fundamental en el ámbito tecnológico y científico[3].

En la actualidad, la IA[4] se ha convertido en un factor indispensable en nuestra vida cotidiana y el motor del futuro de las tecnologías. Su impacto se observa en una amplia gama de aplicaciones, desde herramientas utilizadas comúnmente a diario hasta sectores especializados y avances científicos. En el ámbito cotidiano se encuentran asistentes virtuales que ayudan a facilitar la gestión de las tareas diarias o las aplicaciones de navegación que optimizan las rutas y tiempos de viaje. Más allá de estas funciones, la influencia de la IA se extiende a sectores más especializados como son los avances en medicina[5] o el tratamiento de datos en el sector financiero. Asimismo, desempeña un papel crucial en tecnologías más avanzadas e investigación como pueden ser la conducción autónoma o simulaciones científicas. Aunque la IA es un concepto que se ha explorado desde la antigüedad y cuya finalizad era replicar la inteligencia humana, ha adquirido gran importancia especialmente en el último siglo. Esto ha sido posible a raíz de los grandes avances en informática, matemáticas y procesamiento de datos. Sin embargo, la verdadera popularización de la IA llegó recientemente con herramientas accesibles como chats inteligentes y los sistemas de generación de imágenes, basados en una IA generativa, del cual se puede ver un ejemplo de generación de imágenes en la aplicación *Recraft* en la Figura 1.1. Estas innovaciones mencionadas además de demostrar el potencial práctico de la IA, han acercado la tecnología al

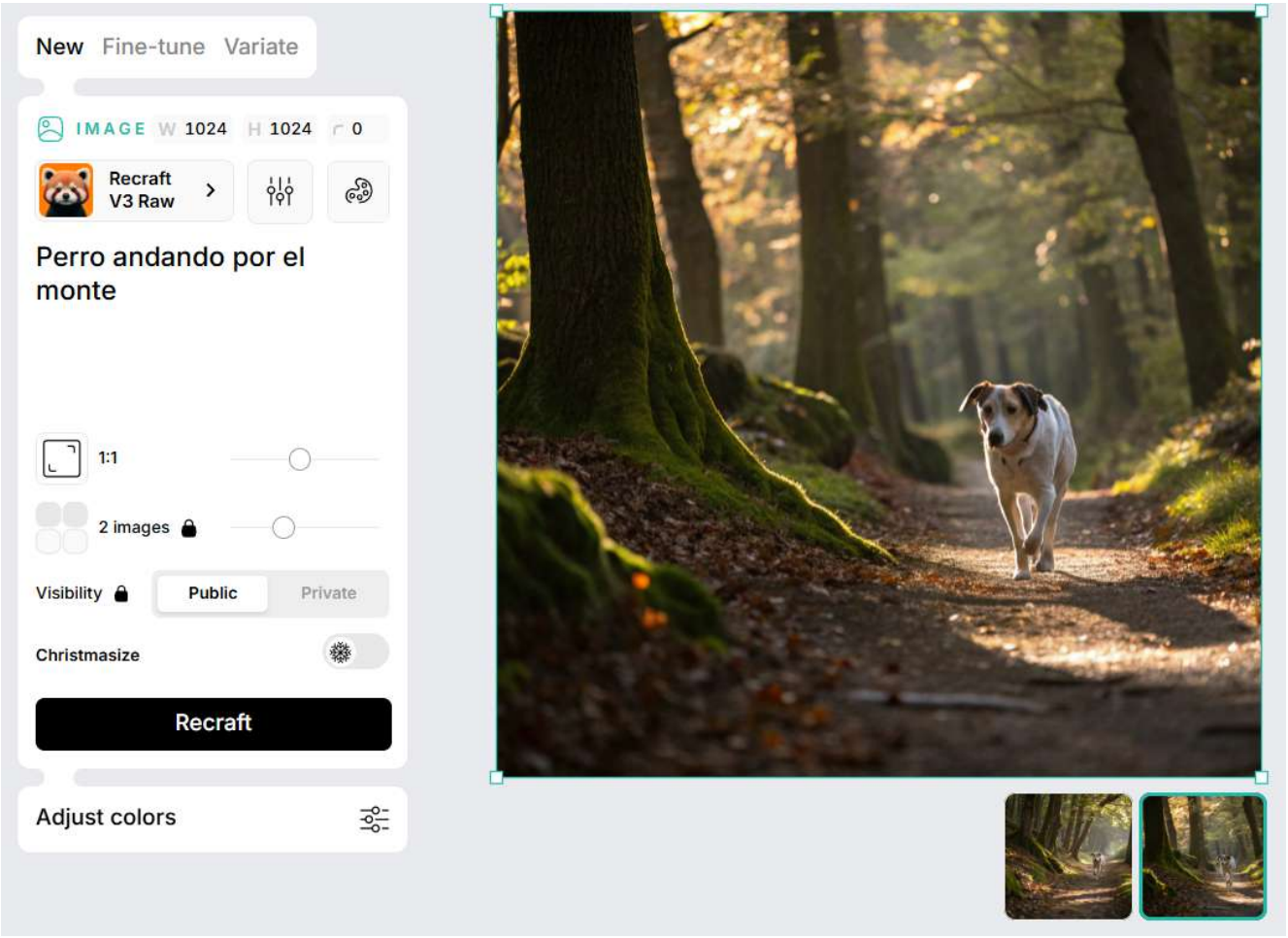


Figura 1.1: Ejemplo de la herramienta en línea Recraft, para un sistema de generación de imágenes basado en IA generativa.

público general, incluso personas sin conocimientos técnicos.

Es un hecho que la IA está presente en nuestro día a día, pues se estima que un 80 % de las personas utiliza la IA en sus actividades diarias, de las cuales solo la tercera parte es consciente de su uso. El mercado de aplicación de las tecnologías aportadas por la IA se encuentra en continuo desarrollo, estimándose una tasa crecimiento del 37.7 % entre 2023 y 2030[1], si bien en 2023, el 9,6 % de las empresas españolas usaban IA en sus trabajos cotidianos[6], en el año 2024 las empresas españolas que utilizan estas herramientas[2]ha aumentado hasta el 44 %. El uso de estas herramientas destacan en procesos como la automatización de flujos de trabajo, la ayuda en la toma de decisiones o en la identificación de personas u objetos, así como, en la mejora de los procesos de producción y en la seguridad TIC. Por otro lado, el 65 % de los estudiantes utiliza herramientas basadas en inteligencia artificial a nivel de usuario en la realización de las tareas escolares o consultas de información[7].

Las distintas aplicaciones de la IA mencionadas en los anteriores párrafos se sustentan de estructuras fundamentales llamadas algoritmos[3]. Los algoritmos son un conjunto de instrucciones y reglas definidas que desempeñan un papel crucial en la resolución de problemas complejos, la automatización de decisiones o el análisis de grandes cantidades de datos. En programación, un

[1] https://www.hostinger.es/tutoriales/estadisticas-y-tendencias-de-ia
[2] https://www.unlockingeuropesaipotential.com/spain
[3] https://www.sas.com/es_es/insights/analytics/algorithms.html

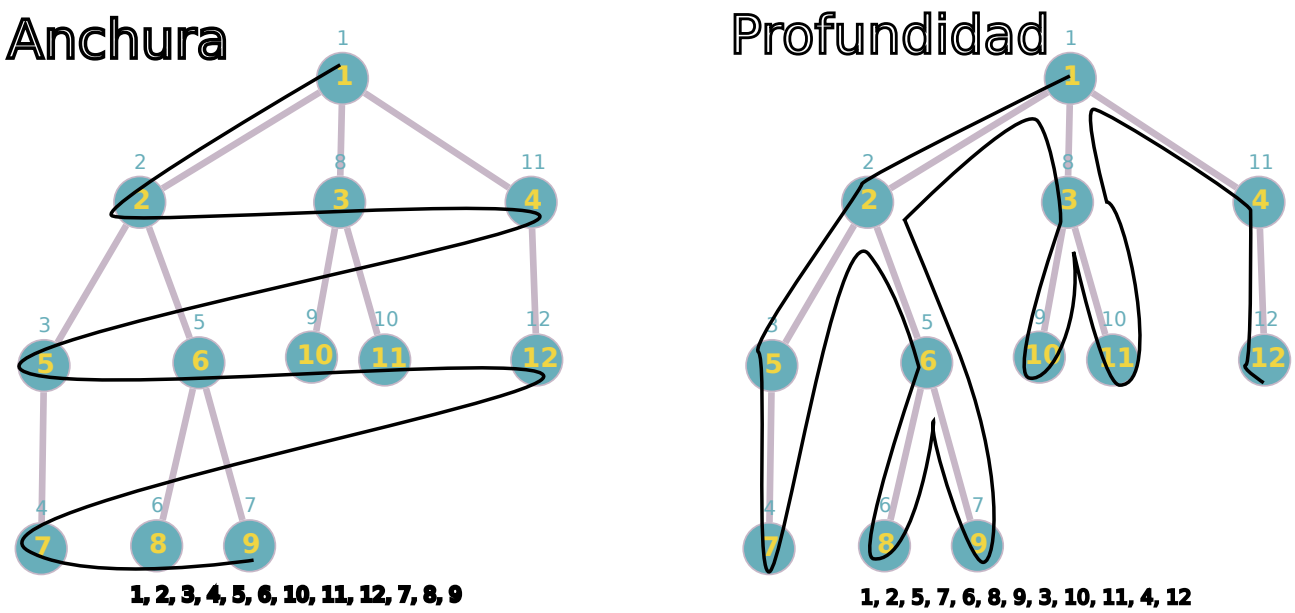


Figura 1.2: Ejemplo de exploración realizada en el espacio de estados de los algoritmos de Búsqueda del Primero en Anchura y Búsqueda del Primero en Profundidad.

algoritmo consiste en la sucesión de pasos lógicos diseñados para solucionar un problema de forma eficiente y precisa. Estos problemas pueden ser variados como el cálculo del máximo común divisor, la clasificación de datos en un sistema o incluso las instrucciones de uso de una lavadora. En este ámbito, los algoritmos constituyen un fundamento para abordar tareas cotidianas y científicas.

Entre las diversas categorías existentes[8], los algoritmos de búsqueda[9] ocupan un lugar destacado debido a su capacidad de exploración y análisis de diversas opciones que permiten obtener soluciones óptimas en un amplio espacio de posibilidades. Estos algoritmos[10] son esenciales en la resolución de problemas cuya incógnita radica en la identificación de elementos específicos, las rutas y configuraciones óptimas dentro de estructuras de datos establecidas[11]. Los algoritmos de búsqueda, mediante herramientas matemáticas y computacionales, permiten no solo la optimización de recursos y tiempo de procesamiento, sino que también son esenciales en aplicaciones de navegación GPS, en los motores de búsqueda en Internet[4] o la toma de decisiones de sistemas autónomos.

Ejemplificando dichas aplicaciones, los algoritmos son fundamentales para su correcto funcionamiento como en los sistemas de navegación, que buscan las rutas mas cortas y rápidas hasta un lugar determinado, o la toma rápida de decisiones en simulaciones científicas explorando grandes conjuntos de datos y configuraciones, e incluso en campos avanzados explorando nuevos modelos y parámetros con la intención de mejorar el rendimiento de los sistemas inteligentes. Algunos de los algoritmos de búsqueda destacables son la Búsqueda del Primero en Anchura[12], Búsqueda del Primero en Profundidad[13], el algoritmo de Dijkstra[14] o el algoritmo A*[15], presentando cada uno de ellos características y aplicaciones específicas que los hacen adecuados para diferentes contextos y están diseñados para actuar sobre estructuras más complejas, como pueden ser los grafos, ofreciendo un mejor rendimiento. Mientras que la búsqueda del Primero en Anchura recorre un grafo explorando los vecinos del nodo inicial antes de avanzar a niveles

[4] https://seoyweb.com/glossary/algoritmo-de-busqueda/

más profundos, la búsqueda del Primero en Profundidad recorre los nodos explorando un camino determinado por los nodos inferiores hasta alcanzar el final del recorrido. Estas estrategias se pueden observar en la Figura 1.2. Por su parte, el algoritmo de Dijkstra determina el camino más corto de un grafo ponderado recorriendo todos los nodos y acumulando los costos asociados a las aristas entre dos nodos para identificar la ruta más eficiente, mientras que el algoritmo A* combina la precisión del algoritmo de Dijkstra con una heurística definida para orientar la búsqueda, permitiendo encontrar el camino de menor coste.

En el ámbito educativo, la incorporación de herramientas didácticas resulta fundamental para facilitar la comprensión y explicación de conceptos complejos[16], especialmente en el área de la IA donde presenta gran complejidad de conceptos e interconexión entre la teoría y la práctica. Las metodologías de enseñanza tradicionales suelen ser unidireccionales, incluso comúnmente carecen del nivel de interacción necesario para captar el interés y compromiso de los estudiantes, dificultando así el proceso de aprendizaje. Por el contrario, la implementación de herramientas gamificadas transforman el aprendizaje en experiencias con mayor atractivo, motivando a los estudiantes y aumentando la participación activa en su propio desarrollo educativo.

Es en este contexto donde surge ***Maze2D***, un videojuego nacido de la necesidad planteada por el tutor de este TFG de formalizar el desarrollo de un juego educativo[17] para la asignatura FIA del segundo curso del grado de Ingeniería Informática de la UVa. A partir de una serie de premisas, este juego debe permitir, de manera didáctica y accesible por los estudiantes de la asignatura, la explicación de los diferentes algoritmos de búsqueda existentes por parte del profesor y su comprensión por parte de los alumnos. Estos algoritmos suelen percibirse de forma abstracta y compleja porque su funcionamiento resulta complicado sin acudir al uso de recursos visuales. La escasez de herramientas que permitan la comprensión de los algoritmos para personas cuyos conocimientos se encuentran en una fase inicial, motiva al desarrollo de esta aplicación con el objetivo de recrear gráfica y lúdicamente el proceso que realizan estos algoritmos.

Una de la motivaciones principales a la hora de desarrollar este trabajo fue la necesidad del tutor de obtener una herramienta capaz de ejemplificar los conceptos básicos de los algoritmos de búsqueda mediante un recurso visual, capaz de captar la atención de los estudiantes de la asignatura en la que se llevará a cabo las explicaciones pertinentes y como consecuencia fomente su interés por la materia. Este enfoque educativo busca no solo la difusión del conocimiento de los conceptos esenciales propios de los algoritmos de búsqueda, sino también reformar el formato de aprendizaje mediante nuevas dinámicas más efectivas y adaptadas al avance de las tecnologías disponibles en la actualidad. Incluso, se pretende plantear una experiencia atractiva no solo para estudiantes, sino para cualquier persona interesada en la IA, posicionando el videojuego como una herramienta útil tanto en el aula como fuera de ella.

*Maze2D* es una herramienta diseñada para hacer accesibles los conceptos relacionados con los algoritmos de búsqueda, al permitir a los usuarios observar de forma sencilla y atractiva como funcionan en tiempo real. A través de una representación interactiva, los usuarios pueden seguir el progreso de los algoritmos durante la exploración de distintas rutas con el fin de lograr alcanzar un objetivo predefinido en un entorno gráfico, lo que facilita la comprensión del proceso sin la necesidad de conocimientos previos avanzados. El diseño intuitivo de esta herramienta destaca gracias a su intención de mostrar claramente los caminos explorados por cada algoritmo, resaltando

el trayecto óptimo seleccionado una vez que se ha logrado encontrar el objetivo establecido, lo que la convierte en una herramienta educativa muy enriquecedora. *Maze2D* es un videojuego de laberintos implementado en el motor de desarrollo de videojuegos *Godot Engine*, con diferentes tamaños y disposición de elementos, que se compone principalmente de tres elementos, como son el Jugador (usuario), el Enemigo (rival) y la Moneda (recompensa).

En adicción, existe la posibilidad de integrar el juego, en futuras iteraciones, en una plataforma de educación gamificada que se encuentra en uso y desarrollo como parte de la investigación en innovación docente que se realiza en el Departamento de Informática por el grupo GREIDI. Esta plataforma, busca combinar herramientas educativas que permitan al usuario interactuar con ellas mediante mecánicas de juego que fomenten un aprendizaje más motivador y activo. El objetivo de esta integración es reforzar los conceptos acerca de los algoritmos de búsqueda mediante herramientas visuales e interactivas, al tiempo que proporciona a los estudiantes un entorno en el que se combina evaluación y progreso personalizado. Esto se lleva a cabo mediante la monitorización del desempeño del estudiante, utilizando la mecánica del juego para recopilar información como la puntuación obtenida. Este enfoque busca incrementar la motivación y el interés de los alumnos, permitiéndoles adquirir no solo conocimientos teóricos acerca de los algoritmos de búsqueda, sino participar activamente en el proceso de aprendizaje resolviendo problemas y superando los posibles desafíos presentados.

La enseñanza mediante este tipo de herramientas no solo supone la mejora del aprendizaje por parte de los estudiantes, sino también la transformación de los métodos de enseñanza, adaptándolos a los avances tecnológicos y a las necesidades educativas actuales. Este modelo podría aplicarse a otras disciplinas o áreas de enseñanza, sirviendo como base para el desarrollo de futuras estrategias educativas. Además, abre nuevas lineas de investigación sobre el impacto de estos juegos en la adquisición de conocimientos, permitiendo comparar su efectividad ante los métodos tradicionales mediante la implantación de sistemas de retroalimentación.

En este ámbito, el auge actual de la IA ofrece la posibilidad, no solo de la enseñanza de algoritmos de búsqueda y conceptos fundamentales, sino también su aplicación práctica en diferentes entornos, como el desarrollo de videojuegos[18]. Este proyecto se sitúa precisamente en ese espacio, mostrando cómo la IA puede ser una herramienta clave para trasladar conceptos teóricos al mundo del entretenimiento digital. En particular, la IA resulta esencial para dotar a los videojuegos de capacidades estratégicas, al simular la inteligencia humana y permitir decisiones basadas en la configuración del entorno y las acciones de los jugadores. La implementación de estas tecnologías no solo genera experiencias inmersivas y variadas, sino que también redefine la interacción entre el jugador, el entorno y los personajes. Entre las aplicaciones más comunes de la IA en videojuegos cabe destacar la creación de enemigos y personajes *Non Playable Character* (NPC) o la adaptación dinámica de la dificultad al desempeño realizado por el usuario o jugador. Ejemplos de estas implementaciones se encuentran en juegos como el *Minecraft*[5], *Invisible INC*[6] o *Assassin's Creed*[7]. Además, existen otros videojuegos que utilizan algoritmos de búsqueda como parte esencial de la mecánica del juego, optimizando decisiones para ejecutar la mejor jugada y evitar la victoria del contrincante. Estos algoritmos permiten evaluar múltiples posibilidades antes

[5] https://www.minecraft.net/es-es
[6] https://store.steampowered.com/app/243970/Invisible_Inc/
[7] http://ubisoft.com/es-es/game/assassins-creed

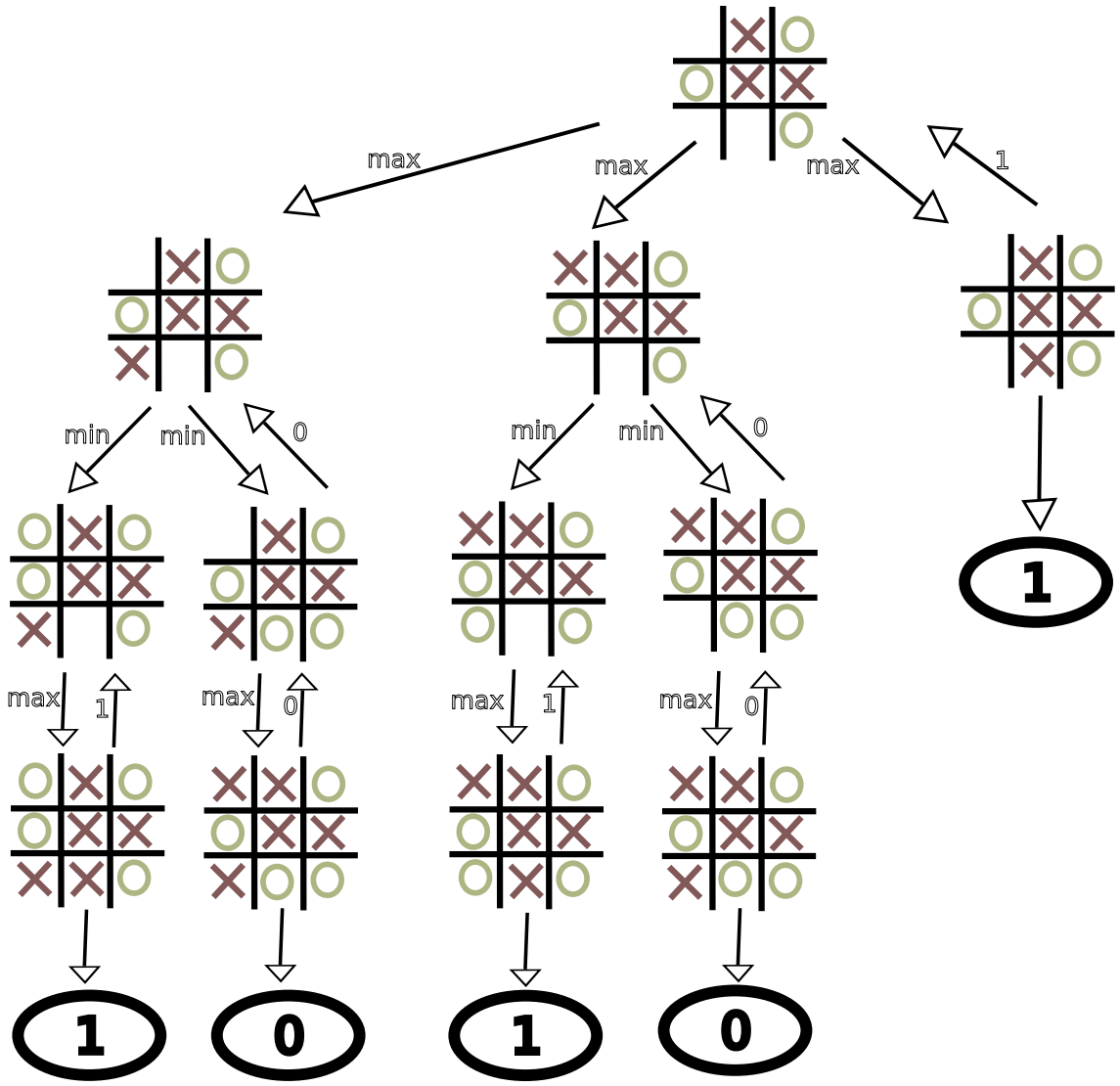


Figura 1.3: Ejemplo del algoritmo minmax aplicado al juego tres en raya para obtener la mejor jugada.

de tomar una decisión. Por ejemplo, el ajedrez y el tres en raya pueden hacer uso del algoritmo de búsqueda *MINIMAX*, diseñado para obtener el movimiento óptimo bajo la premisa de que el jugador contrario realiza elecciones óptimas, como se ve en el ejemplo descrito en la Figura 1.3. En contraste, el *Tetris* sería un juego más orientado a utilizar algoritmos heurísticos que analizan factores como las líneas completadas, casillas ocupadas o la altura acumulada para determinar la ubicación más eficiente. Asimismo, algunos videojuegos utilizan los algoritmos de búsqueda para explorar el entorno y encontrar las rutas óptimas como es el ejemplo del *Pac-Man*, en el que los fantasmas aplican estos algoritmos para localizar al jugador.

## 1.2. Objetivos

Este trabajo tiene como objetivo principal, el desarrollar una aplicación con carácter educativo que sirva de prototipo para la enseñanza de diferentes algoritmos de búsqueda. Este objetivo se materializará mediante el análisis, diseño e implementación de un videojuego desarrollado en *Godot Engine*, que combine simplicidad y la atracción visual como ejes fundamentales para la adquisición del conocimiento de estos algoritmos. Para alcanzar este objetivo general, es necesario detallar una serie de objetivos específicos que han sido listados a continuación.

- Analizar el rendimiento de los algoritmos de búsqueda, permitiendo observar su efectividad en función del tamaño y complejidad de los datos, proporcionando una herramienta para comprender mejor su funcionamiento.
- Diseñar interfaces accesibles e interactivas, que faciliten la interactuación con el videojuego y permitiendo la exploración con los diferentes algoritmos.
- Adoptar la metodología de desarrollo Scrum, con la finalidad de fortalecer los conocimientos sobre este método, permitiendo una planificación flexible además del cumplimiento de entregas en periodos cortos de tiempo.
- Modelar conocimientos teóricos adquiridos durante la etapa de análisis y diseño del proyecto.

s

## 1.3. Estructura del documento

Este documento se organiza en distintos capítulos, que dividen el proyecto en las diferentes etapas de planteamiento del problema, la reunión de necesidades para su posterior diseño y desarrollo del producto final, además de la descripción de conclusiones surgidas de los resultados obtenidos. La contribución individual de cada uno de los capítulos es detallada a continuación.

El análisis de proyectos existentes con características similares y como abordan las necesidades comparables a las planteadas en este trabajo se detallan en el Capitulo 2. El Capitulo 3, documenta la planificación inicial del proyecto, indicando las restricciones, la estimación de costes a los hace frente durante la ejecución del proyecto, además de las herramientas utilizadas para llevar a cabo su desarrollo, como también detalla la metodología aplicada para llevar a cabo las diferentes fases del proyecto. Por otra parte, el Capitulo 4, realiza el análisis y recopila los requisitos proporcionados por el cliente y las tareas identificadas para satisfacer el cumplimiento de dichos requisitos. Mientras que las decisiones tomadas para estructurar el proyecto, la arquitectura y patrones aplicados, así como la descripción del diseño de las interfaces que componen la aplicación y el flujo de ejecución de las tareas, se detallan en el Capitulo 5. En consecuencia, el Capitulo 6 describe la implementación del videojuego, detallando el uso de librerías y algoritmos que hacen posible el correcto funcionamiento del mismo, de igual manera que la configuración necesaria para

llevar a cabo la ejecución del proyecto. Finalmente, el análisis de las conclusiones obtenidas del desarrollo del videojuego, la reflexión acerca del cumplimiento de los objetivos y las lineas futuras se describen en el Capitulo 7. Adicionalmente, se incluyen una serie de Anexos con información adicional relativa a algunos apartados para mayor completitud.

# Capítulo 2

# Antecedentes y Estado del Arte

Los videojuegos tal y como los conocemos en la actualidad, surgieron a mediados del siglo XX, aunque el concepto de juego como forma de entretenimiento ya existía en antiguas civilizaciones a través de juegos de mesa y los primeros experimentos en electrónica[19]. Un videojuego, en la actualidad, se define como un software en el que uno o más jugadores interactúan con un dispositivo electrónico mediante periféricos de entrada o controladores, mostrando imágenes de vídeo en una pantalla. En las décadas de 1940 y 1950, cuando se abrieron paso los amplios avances en electrónica y computación dando lugar a la aparición de las primeras máquinas consideradas las predecesoras de los videojuegos. Estas máquinas suponían el inicio de la interacción de las personas con una máquina con el propósito de satisfacer su entretenimiento, siendo las precursoras de juegos surgidos en torno al año 1960 como es *Spacewar!*[1]. Posteriormente, aparecerían otros títulos dirigidos y ampliamente reconocidos al gran público como son *Pong* o *Asteroids*, así como las primeras consolas domésticas como son *Atari 2600*[2] y las primeras máquinas de arcade, popularizada gracias al videojuego *Space Invaders*[3]. En los años 90, con el salto a 16-bits, se introdujeron mejoras gráficas y las primeras experiencias tridimensionales. Además, aparecen nuevas consolas y juegos cada vez más avanzados, como son los juegos en línea y multijugador, así como plataformas móviles y portátiles, como son los teléfonos móviles inteligentes y las tabletas a finales de la década de los 2000.

Los primeros videojuegos carecían de estándares de desarrollo. En concordancia con el crecimiento de la industria, surge la necesidad de motores de desarrollo, plataformas que permiten la reutilización de código y la optimización de los videojuegos. Estos motores integran componentes como renderizado gráfico, simulación de física, detección de colisiones, animaciones, sonidos o inteligencia artificial. Tanto *Freescape* que está considerado como el primer motor de videojuegos e introdujo el concepto de diseño 3D, como también *Doom Engine* (id Tech 1)[4] considerado como uno de los primeros, revalidando el diseño 3D a través del mapeado de texturas, ambos introducen conceptos como los vértices, las aristas y los polígonos.

[1] https://store.steampowered.com/app/1086160/Space_War_Infinity/
[2] https://atari.com/
[3] https://www.space-invaders.com/home/
[4] https://store.steampowered.com/tags/en/DOOM+engine

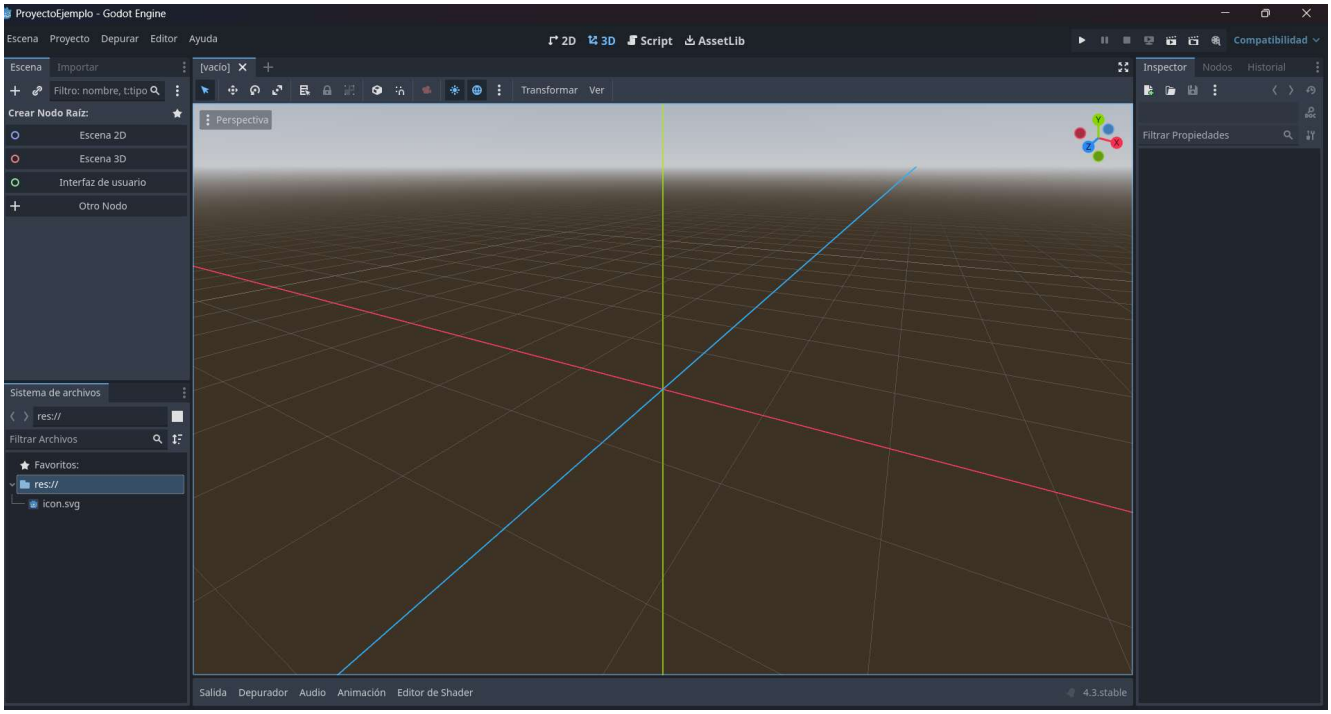


Figura 2.1: Estado inicial del entorno de desarrollo del motor de videojuegos Godot Engine.

A lo largo de los años, el mercado de los motores de videojuegos ha evolucionado significativamente con la aparición de herramientas que han transformado la industria, tanto para grandes compañías como para desarrolladores independientes. Entre estas plataformas, destacan *Unreal Engine*[5] o *Unity*[6], dos motores nacidos a inicios de los 2000, y el motor que ocupa este proyecto *Godot Engine*[7], una plataforma nacida hace apenas 10 años. Cada uno de estos motores de videojuegos presentan características únicas que los posicionan como referentes en diferentes ámbitos de mercado. En la Figura 2.1 se muestra un ejemplo de la configuración interfaz de inicial de un motor de videojuegos, en este caso usando Godot Engine.

Los motores de videojuegos son herramientas fundamentales en la industria, ya que proporcionan las bases técnicas necesarias para diseñar, construir y ejecutar videojuegos. Cada motor tiene un enfoque único, adaptándose a diferentes necesidades y perfiles de desarrolladores, desde grandes estudios con recursos avanzados hasta creadores independientes y proyectos educativos.

*Unreal Engine*, lanzado inicialmente para el diseño de juegos de disparos en primera persona para ordenadores personales, ha expandido su alcance a múltiples géneros y se ha adaptado por otras industrias como el cine y la televisión. Es reconocido por su gran capacidad de generar gráficos de alta fidelidad, destacando por su enfoque en la calidad visual y las físicas avanzadas, además, presenta un alto grado de portabilidad, compatible con muchas plataformas de escritorio, móviles o consolas. Aunque es gratuito para los primeros ingresos de un millón de dólares, después requiere un porcentaje de regalías, lo que puede influir en su adopción. Este motor no solo se utiliza en videojuegos, sino también en múltiples áreas como en avances farmacológicos a través de herramientas de realidad virtual, como también la ayuda en el diseño de edificios y estructuras

[5] https://www.unrealengine.com/
[6] https://unity.com/es
[7] https://godotengine.org/

o de automóviles, además de desarrollar plataformas de enseñanza y capacitación, por lo que se ha consagrado como una herramienta apta para la investigación e incluso la educación.

*Unity Engine*, por su parte, destaca por su accesibilidad y su enfoque en el desarrollo multiplataforma que permite a desarrolladores independientes el acceso a las herramientas necesarias de creación de videojuegos sin necesidad de grandes inversiones iniciales. Originalmente fue concebido para desarrollar proyectos para *Mac OS X*, y debido a su éxito, se expandió a múltiples plataformas, ofreciendo compatibilidad con diversas herramientas de modelado como *Blender*, *Maya* o *Adobe Photoshop*. Su soporte para múltiples lenguajes de programación, junto con una solución de control de versiones para almacenar los *assets* y *scripts* del juego llamada *Unity Asset Server*, además del uso de *PostgreSQL* como sistema gestor de bases de datos, lo convierten en un motor adaptable a usuarios de distintos niveles de experiencia. También destaca su amplia comunidad de soporte y de usuarios que hacen de ella una herramienta accesible para un gran número de públicos. Por último, este motor ha integrado herramientas de aprendizaje automático y simulación, expandiendo su uso al campo de la inteligencia artificial y permitiendo el uso de sus herramientas y de videojuegos creados con este motor para la enseñanza.

*Godot Engine*, en contraste, es un motor de juegos multiplataforma que ha ganado notoriedad en los últimos años debido a su filosofía de código abierto y su constante evolución para adaptarse a las necesidades del mercado. Permite crear juegos en 2D y 3D en varios lenguajes de programación, presenta un enfoque modular, interfaz intuitiva y utiliza nodos para facilitar la experiencia de desarrollo y herencia. Esta plataforma incluye un motor gráfico 2D y uno 3D independientes y combinables entre sí, e integra un lenguaje de programación propio, *GDScript*. Este motor está diseñado para ser accesible, ofreciendo una curva de aprendizaje más suave y eliminando las barreras de costos al ser completamente gratuito, sin necesidad de regalías ni licencias, lo cual la hacen una herramienta apta para el desarrollo de proyectos educativos. Aunque su soporte en consolas requiere de colaboración con empresas de terceros, debido a las restricciones a juegos cuya licencia es de código abierto. Además, cuenta con una gran comunidad de desarrolladores que está aumentando en consonancia con la evolución del motor, y que permite que sea una herramienta accesible.

En 2023, *Unity* anunció un cambio en su modelo de negocio, lo que supuso el aumento de críticas negativas y la pérdida de confianza de sus usuarios, generando la migración de miles de desarrolladores a otros motores de desarrollo, premiando a Godot por ser de código libre y con una gran comunidad, consolidando un gran aumento del número usuarios afines. El problema de esta polémica radica en la modificación en los planes de pago y la suscripción a su software, empezando a cobrar a empresas que alcancen una facturación mayor de doscientos mil dólares en el periodo de un año y al menos doscientas mil instalaciones del juego, cobrando una tarifa por instalación. Tras este anuncio, la polémica se encontraba en auge y muchos desarrolladores mostraron su descontento, por lo que la empresa decidió retractarse en algunas ideas, relajando las cuotas impuestas. Finalmente, en septiembre de 2024 *Unity* puso punto y final a la polémica eliminando completamente el cambio anunciado un año atrás y volviendo al estado anterior, donde la política de pagos no era tan estricta. A pesar de su eliminación, el descontento generado no ha terminado de convencer a algunos usuarios, que determinan una mala estrategia por parte de la empresa y una retracción tardía.

| Característica | Unreal | Unity | Godot |
|---|---|---|---|
| Aplicación ideal | Shooters o simulaciones realistas (presupuesto alto) | Realidad aumentada o móviles (presupuesto intermedio) | Puzzles o plataformas 2D (presupuesto limitado) |
| Nivel de dificultad | Avanzado | Intermedio | Principiante - Medio |
| Precio | Gratuito hasta cierto límite de ingresos | Gratuito con restricciones de ingresos y clientes | Totalmente gratuito |
| Compatibilidad | Enfocado en plataformas de alto rendimiento | Multiplataforma | Limitado en consolas |
| Lenguajes soportados | C++ o Blueprints | C# o ShaderLab | C#, C++ o GDScript |
| Ejemplos de juegos | Fortnite, Final Fantasy VII Remake | Among Us, Pokémon Go | Sonic Colors, Kingdoms of the Dump |

Tabla 2.1: Comparación de las principales características de los motores de videojuegos Unreal, Unity y Godot.

La Tabla 2.1 muestra un resumen de las características y comparación de cada uno de los motores de videojuegos mencionados previamente. En conjunto, estos tres motores representan diferentes enfoques en el desarrollo de videojuegos, mientras que *Unreal Engine* se orienta a producciones de alta calidad gráfica y desarrollos avanzados con un sistema de regalías, por su parte, *Unity Engine*, prioriza la accesibilidad y el alcance multiplataforma, estando formada de una gran diversidad de herramientas y restricciones de ingresos y clientes, y en contraposición, se encuentra *Godot Engine* que ofrece una solución libre y flexible para proyectos de presupuesto limitado y al igual que el alcance multiplataforma.

Se ha llevado a cabo una investigación de los productos existentes en el mercado cuyas características son similares los requisitos planteados para el desarrollo de *Maze2D*, con la finalidad de crear una herramienta diferenciada y única. Si bien existen numerosas aplicaciones diseñadas para representar visualmente los procesos llevados a cabo por los algoritmos para alcanzar un estado final, la mayoría carece un enfoque interactivo o de un objetivo explícitamente educativo, especialmente dirigido a estudiantes que se encuentran en una etapa inicial de aprendizaje sobre el tema. En el mundo actual, donde la tecnología presenta tanta importancia, es fundamental la adquisición habilidades digitales, incluso desde una edad temprana. Por ello, en los centros educativos se buscan maneras efectivas y entretenidas de enseñar los conceptos tanto básicos como avanzados de programación, dependiendo del nivel de enseñanza y ámbito en el que se requiere el uso de estas herramientas. Los juegos analizados no están orientados a la enseñanza, una se han creado con el objetivo de proporcionar entretenimiento al usuario. Aunque implementan algoritmos de búsqueda como parte de su mecánica de juego, tales como la búsqueda del jugador o la determinación de la posición óptima, estas implementaciones no son visualmente comprensibles. En otras palabras, no muestra gráficamente los procesos y métodos empleados en cada una iteración de los algoritmos, lo que dificulta su interpretación educativa.

Como ejemplo de videojuego donde un jugador debe escapar de uno o varios enemigos, y el


Figura 2.2: Ejemplo de la dinámica de juego del Pac-Man.

cual ha servido como inspiración parcial en el desarrollo de *Maze2D*, destaca el histórico *PacMan*[8], uno de los videojuegos arcade más icónicos e influyentes de la historia, marcando un antes y un después en la industria del entretenimiento interactivo, estableciendo bases fundamentales en el desarrollo de juegos posteriores con dinámicas similares. La Figura 2.2 es un ejemplo de la dinámica del juego *Pac-Man* clásico. Su diseño combina elementos de estrategia, habilidad y azar, desarrollados dentro de un laberinto cerrado donde el jugador debe recolectar todos los puntos mientras evita ser alcanzado por un conjunto de enemigos para avanzar al siguiente nivel. Cada nivel añade progresivamente dificultad, además, cuenta con un sistema de vidas que da al jugador oportunidades limitadas para alcanzar la victoria antes de reiniciar la partida.

En este proyecto, *Pac-Man* ha sido una inspiración directa, especialmente en la composición de la estructura de los laberintos y la mecánica de movimientos de los personajes, incluso, algunos aspectos de diseño visual del adversario están inspirados en el personaje protagonista. Además, la inteligencia aplicada al enemigo de *Maze2D* se basa en algoritmos de búsqueda que le permiten localizar y perseguir al jugador estratégicamente, un enfoque que presenta rasgos comunes a la dinámica realizada por los fantasmas de *Pac-Man*. Estos fantasmas están determinados por patrones de movimiento específicos, lo que introduce diversidad y colaboración en su comportamiento, algunos persiguen directamente al jugador, otros intentan bloquear caminos y otros presentan un comportamiento relativamente aleatorio en la exploración del mapa. La Figura 2.3 muestra el comportamiento característico de cada uno de los fantasmas, representando la trayectoria de cada uno en el color propio de cada fantasma.

[8] https://www.pacman.com/en/

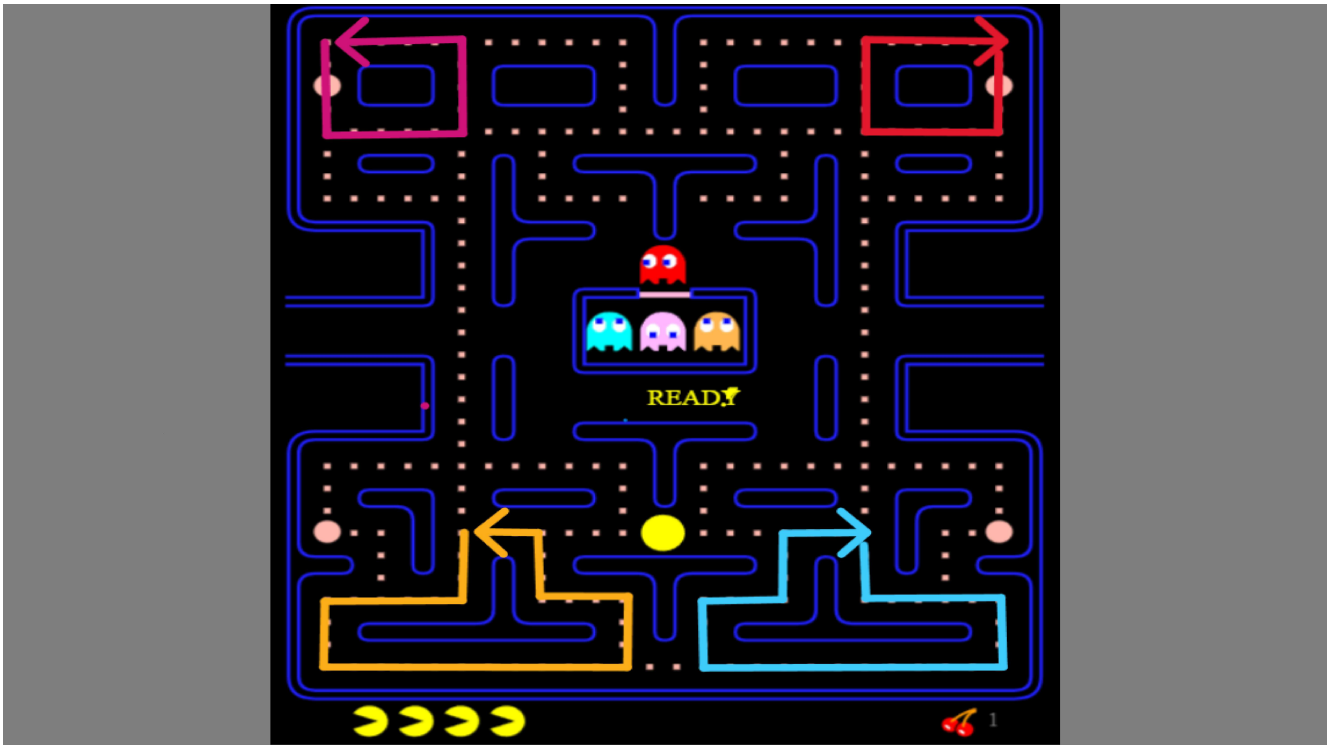


Figura 2.3: Ejemplo de la lógica de movimiento de los fantasmas en modo persecución en *Pac-Man*.

Este videojuego, donde los algoritmos subyacentes no son visibles para el usuario y están destinados únicamente al entretenimiento, lo convierte en un juego de escasa importancia en la enseñanza de estos conceptos mediante herramientas educativas que permitan la visualización de los procesos llevados a cabo. En cambio, *Maze2D* busca dar un paso más allá mostrando visualmente el proceso empleado por el personaje, permitiendo entender la toma de decisiones en cada momento.

A diferencia de este videojuego, donde los algoritmos de búsqueda empleados por los enemigos no son visibles para el usuario y están diseñados exclusivamente para maximizar la experiencia de entretenimiento, este proyecto busca una aplicación más educativa. La ausencia de herramientas que permitan visualizar los procesos y decisiones tomadas por los personajes convierte a *Pac-Man* en un juego de escasa utilidad en la enseñanza de conceptos relacionados con algoritmos y toma de decisiones. En contraste, *Maze2D* se propone ir un paso más allá al hacer visibles los procesos utilizados por el jugador, permitiendo al usuario comprender las decisiones estratégicas en tiempo real, como también ofrecer una plataforma para explorar y aprender los fundamentos detrás de los algoritmos de búsqueda. Por último, *Pac-Man* también destaca como un precursor en la simulación de dinámicas de cooperación entre agentes. La diversidad de comportamientos de los fantasmas, diseñada deliberadamente por su creador[21], introduce estrategias no lineales y repetitivas de persecución que enriquecen la interacción. Este enfoque colaborativo entre los enemigos no solo lo convierte en un referente clave para diseñar sistemas donde la interacción entre agentes mejora la experiencia del jugador, sino que proporciona una base conceptual para expandir el diseño de juegos y sistemas más complejos. Estos principios pueden expandirse a herramientas, donde la representación visual de la interacción y de los procesos ayuden a comprender estos conceptos, convirtiéndose en excelentes herramientas educativas.

De la investigación previa, se han analizado herramientas educativas concebidas con un carácter

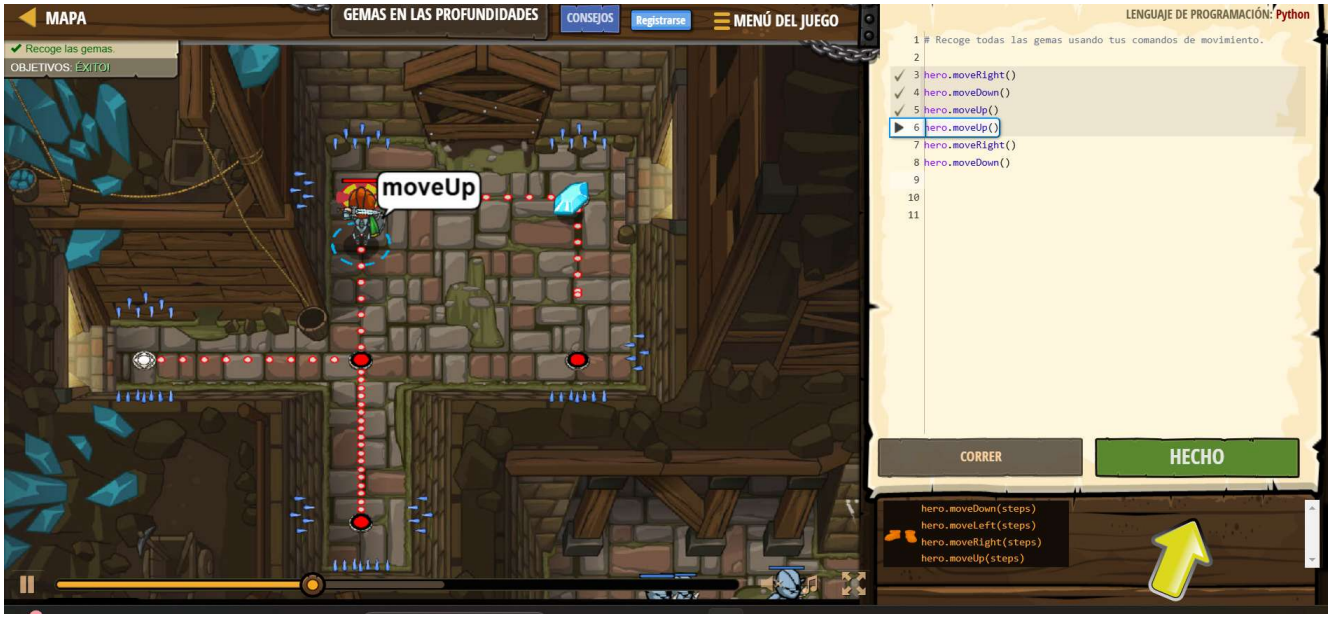


Figura 2.4: Ejemplo de uso del juego *Code Combat* en lenguaje *Python*.

lúdico, cuyo objetivo principal es enseñar conocimientos mediante un juego de entorno interactivo. Estas aplicaciones incluyen sistemas de retroalimentación que permiten a los usuarios identificar y corregir errores durante el proceso de aprendizaje. Los ejemplos analizados en esta sección presentan estrategias valiosas para enseñar conceptos complejos de manera accesible y entretenida.

## 2.1. Code Combat

*Code Combat*[9] es otro de los juegos educativos existentes en el mercado aptos para el aprendizaje de programación desde niveles más básicos hasta un nivel intermedio. Esta herramienta es un juego de rol en línea orientado al aprendizaje de la programación, abordando los fundamentos de los lenguajes como *Python* o *JavaScript*. En este juego, el usuario controla un personaje mediante la introducción de comandos e instrucciones determinando la secuencia de acciones que realiza el personaje. *Code Combat* ofrece versiones adaptadas tanto a usuarios mas jóvenes como a estudiantes avanzados, con niveles diseñados para aumentar gradualmente la complejidad. Este diseño permite la adquisición de conocimientos al ritmo de adaptación de las propias necesidades, combinando desafíos de codificación con una trama de aventuras que fomenta el interés y la motivación del usuario. Esta experiencia permite mantener el interés del alumno, transformando el aprendizaje de la programación en un proceso lúdico de mayor atractivo con el que involucrar al alumno de forma interactiva. Además, esta plataforma cuenta con una comunidad colaborativa donde los usuarios pueden discutir soluciones, formular preguntas y trabajar juntos en los problemas propuestos, lo que refuerza el aprendizaje mediante la interacción social. Por otra parte, esta herramienta ofrece determinadas funciones y niveles de forma no gratuita, es decir, se requiere el pago de una suscripción para completar algunas dinámicas del juego. Esto puede ser un inconveniente para aquellas personas que desean aprender gratuitamente, o no tienen la posibilidad de invertir en esa suscripción. En conclusión, este juego educativo permite el aprendizaje de programación y de algunos lenguajes de manera activa e enriquecedora, permitiendo

[9] https://codecombat.com/

la colaboración con otros jugadores, pero cuenta con limitaciones de las funciones que ofrece que requieren la suscripción, generando un impedimento de uso en un porcentaje del público objetivo. En la Figura 2.4 recrea el uso de este juego online en un nivel básico de enseñanza.

## 2.2. Blocky Games

*Blocky Games*[10] es una plataforma en línea compuesta por diversos juegos educativos diseñados para enseñar programación de forma sencilla e intuitiva, especialmente para niños y adolescentes sin conocimientos previos de programación que buscan la oportunidad de aprender ciencias de la computación. El usuario se familiariza con un lenguaje de programación visual basado en bloques, donde la tarea consiste en seleccionar y acoplar ordenadamente las instrucciones para ejecutar las acciones deseadas. Es una herramienta que destaca por su accesibilidad y enfoque lúdico, ideal para iniciarse en la programación. Esta herramienta ofrece diversos juegos organizados en niveles de dificultad, lo que hace que cada etapa sea más desafiante que la anterior. A través de estos juegos, se pueden aprender conceptos fundamentales de programación como bucles, condicionales, funciones y ecuaciones. Por ejemplo, el juego más básico, *Puzzle*, introduce la dinámica de los bloques y su correcta disposición. El juego del *Laberinto* profundiza en el uso de bucles y condicionales, mientras que en *Película* se introducen las ecuaciones, y en *Estanque JS* se explora el uso del lenguaje de programación *JavaScript*. Aunque esta herramienta es adecuada para un nivel inicial y permite el aprendizaje desde un nivel básico, su complejidad no alcanza un nivel avanzado, lo que la hace menos apta para su uso en contextos educativos superiores, como el ámbito universitario.

## 2.3. OpenAI Gym

*OpenAI Gym*[11] es un conjunto de herramientas diseñadas para facilitar el desarrollo y comparación de algoritmos de aprendizaje por refuerzo (RL). Emplea un agente que interactúa con un entorno y recibe una recompensa en función de sus acciones, para ello sigue el ciclo básico de la IA: observar, razonar y actuar. Esta herramienta proporciona una serie de escenarios configurados para el uso por un agente divididos en cuatro secciones: *Ejemplos de aprendizaje* para la adquisición de los conocimientos básicos de RL; *Algorítmicos* con ejercicios de complejidad gradual; *Atari* para sistemas que aprenden a jugar a juegos *Atari 2600*; y *Robots 2D y 3D* que se mueven bajo leyes físicas y restricciones temporales. La plataforma ofrece una API sencilla y consistente permitiendo la interacción con diversos entornos, programando agentes que envían acciones al entorno y reciben estados, recompensas y señales acerca de el logro del objetivo. Su diseño se enfoca en la investigación del comportamiento de los agentes en la toma de decisiones óptimas o erróneas en base a la repetida interacción con el entorno, ideal para explorar métodos de Q-learning, aprendizaje profundo o por políticas. Es una herramienta compatible con numerosos

[10] https://blockly.games/
[11] https://openai.com/index/openai-gym-beta/


Figura 2.5: Ejemplo de uso de la herramienta *OpenAI Gym* con el juego básico *CartPole*.

*frameworks* de aprendizaje profundo y bibliotecas de IA como *TensorFlow*[12], *Keras*[13] o *PyTorch*[14], facilitando su integración en gran cantidad de proyectos avanzados. A su vez, *OpenAI Gym* es una plataforma poderosa de enseñanza de conceptos del aprendizaje por refuerzo, algoritmos de optimización y simulación. Los estudiantes pueden programar los agentes con el objetivo de resolver problemas que van desde los más básicos hasta desafíos avanzados, lo que requiere de un conocimiento previo en programación en *Python* e incluso, acerca de los conceptos del aprendizaje por refuerzo. A medida que los agentes se entrenan, es posible observar como evolucionan sus estrategias de resolución en función del tiempo, ajustándose recompensas obtenidas. La Figura 2.5 muestra un ejemplo de uso en el juego más básico de esta herramienta llamado *CartPole*.

## 2.4. Unity ML-Agents

*Unity ML-Agents*[15] es una extensión del motor de videojuegos *Unity* que integra aprendizaje automático, específicamente para entrenar agentes de IA mediante técnicas de aprendizaje por refuerzo. En sintonía con *OpenAI Gym*, es posible la definición de entornos de aprendizaje permitiendo al agente realizar el entrenamiento ajustado a las recompensas y penalizaciones obtenidas, así como el continuo ajuste de los parámetros y variables de configuración. Debido a la popularidad de *Unity* en la industria de videojuegos, *ML-Agents* es una herramienta útil y atractiva para la enseñanza. Ofrece aplicaciones en áreas como la simulación de entornos realistas, el aprendizaje de IA, y la robótica, permitiendo a los estudiantes experimentar con conceptos complejos de forma práctica, sin embargo, requiere conocimientos previos en IA para su uso efectivo.

[12] https://www.tensorflow.org/
[13] https://keras.io/
[14] https://pytorch.org/
[15] https://docs.unity3d.com/es/2019.4/Manual/com.unity.ml-agents.html

## 2.5. Project Malmo

*Project Malmo*[16] es un proyecto de *Microsoft* diseñado para fomentar la investigación en IA mediante el uso del mundo de *Minecraft* como entorno de pruebas. Este modificación de *Minecraft* ofrece herramientas que permiten a los agentes interactuar y aprender en escenarios complejos, siendo programables en distintos lenguajes. Los objetivos incluyen la mejora de la colaboración entre humanos y agentes inteligentes, la adaptación a entornos dinámicos y la resolución de problemas mediante métodos como el aprendizaje por refuerzo y la ciencia cognitiva. Además de su enfoque en la investigación, *Project Malmo* tiene aplicaciones educativas, ya que facilita la enseñanza de conceptos avanzados de IA y programación de manera interactiva y visual. Ejemplos de actividades incluyen la navegación en laberintos, la colaboración entre agentes y la simulación de problemas del mundo real. Sin embargo, su uso requiere conocimientos previos, lo que puede limitar su accesibilidad para principiantes.

## 2.6. Pathfind Visualizer

*Pathfind Visualizer*[17] es una plataforma en línea diseñada para visualizar y comprender la implementación de los algoritmos de búsqueda en un laberinto, en concreto, permite observar el funcionamiento de cuatro algoritmos en tiempo real. En esta herramienta el tamaño del entorno es fijo, donde el usuario puede personalizar el entorno colocando obstáculos, seleccionando el lugar de inicio y fin, y eligiendo diferentes algoritmos como son Busqueda en Anchura, Busqueda en Profundidad, Dijkstra y A Estrella. El propósito principal de esta aplicación es educativo y demostrativo, proporcionando una comprensión visual y atractiva acerca de la lógica y los conceptos de los algoritmos proporcionados, lo que la convierte en una herramienta útil en la enseñanza de algoritmos e inteligencia artificial. La Figura 2.6 presenta un ejemplo de uso de esta aplicación, empleando el algoritmo de búsqueda en anchura y un mapa definido manualmente. Este enfoque presenta ventajas significativas, como la capacidad de emplear diversos algoritmos de búsqueda permitiendo comparar su efectividad y comprender su funcionamiento en tiempo real. También destaca por ofrecer una representación gráfica intuitiva, exponiendo diferentes opciones de configuración de los obstáculos y para ajustar la velocidad de procesamiento, facilitando la adaptación a contextos educativos. Gracias a esta herramienta, es posible la enseñanza y exploración de los algoritmos de búsqueda en un entorno interactivo. Sin embargo, también presenta ciertas limitaciones como herramienta educativa, como la experimentación en diferentes configuraciones que está restringida a un tamaño fijo del espacio de búsqueda. Por otra parte, aunque el diseño visual es funcional, puede no resultar suficientemente atractivo para los estudiantes, lo que generar descontento y falta de interés entorno a esta plataforma y los conocimientos ofrecidos.

[16] https://www.microsoft.com/en-us/research/project/project-malmo/
[17] https://pathfindout.com/

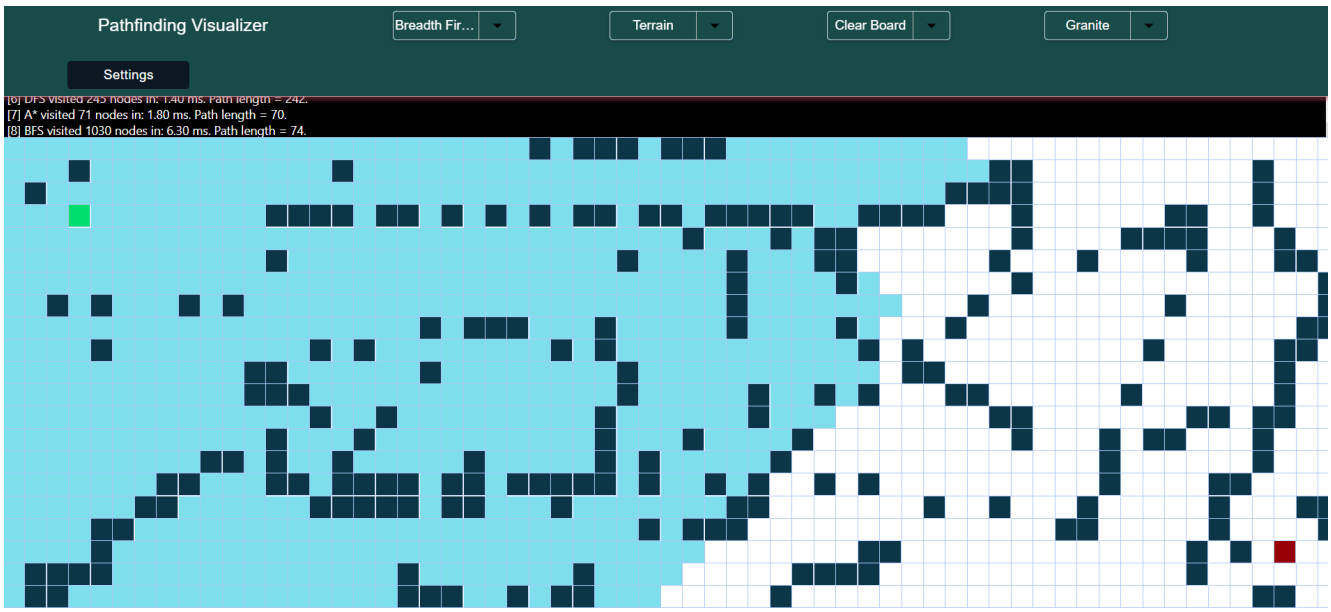


Figura 2.6: Ejemplo de la búsqueda en anchura utilizada en la aplicación *Pathfind Visualizer*.

# Capítulo 3

# Planificación y Metodología

En este trabajo, se han definido diversos objetivos cuya importancia y prioridad está basada en el cumplimiento de unos requisitos mínimos, que buscan obtener un producto acorde y funcional a las características solicitadas por el cliente interesado en este proyecto, en este caso, el tutor del Trabajo de Fin de Grado TFG. El objetivo principal que debe cumplir este trabajo es el desarrollo de una aplicación de carácter educativo y funcional basada en un videojuego, que sirva de prototipo para la enseñanza de diferentes algoritmos de búsqueda mediante una interfaz visual sencilla y atractiva para el usuario. Además, este objetivo se produce en sintonía con la búsqueda de los profesores de idear nuevas herramientas que permitan la enseñanza de asignaturas universitarias y sus conceptos teóricos y prácticos, e incluso, aplicable en otras enseñanzas externas, obteniendo mayor fidelidad e interacción de los estudiantes en los que está enfocada la aplicación.

A partir del análisis exhaustivo de los objetivos establecidos, se concluyen una serie de características clave que el proyecto debe contemplar y afrontar, para garantizar el desarrollo de un producto ajustado a las metas finales y las necesidades que satisfacen las expectativas planteadas por la parte solicitante. Estas características se formulan en términos de restricciones operativas que limitan las tareas a realizar, los riesgos potenciales surgidos durante el ciclo de vida del proyecto a los que se debe enfrentar reduciendo su impacto, los costes asociados al cumplimiento de los objetivos, y la planificación estricta que permita abordar las actividades en los periodos estipulados. Este planteamiento asegura tanto el cumplimiento de las necesidades planteadas, como proporcionar una base sólida para la gestión eficaz de los recursos, minimizar los riesgos y garantizar la calidad del producto final. En este capítulo se abordan los aspectos esenciales para lograr el correcto desarrollo y ejecución del proyecto, incidiendo en los métodos y alternativas empleadas para afrontar los posibles inconvenientes surgidos en el ciclo de vida del proyecto que permitan prevenir el impacto de los mismos, además de, la descripción de las herramientas utilizadas para la óptima implementación de cada una de las actividades requeridas. De igual modo, el capítulo proporciona una base para el posterior análisis de la solución planteada, así como el diseño e implementación de forma eficiente.

## 3.1. Restricciones

Las restricciones de un proyecto no se limitan únicamente al tiempo, el coste y el alcance, sino que abarcan también aspectos técnicos, legales y de recursos, que deben tenerse en cuenta previamente al inicio del proyecto, así como en su etapa de desarrollo, llevando a cabo acciones que permitan evitarlas y garanticen el éxito final del proyecto. Estas limitaciones son factores que condicionan el desarrollo de las tareas y el logro de los objetivos que forman el ciclo de vida del proyecto, por lo que se debe llevar a cabo una correcta gestión para obtener un exitoso resultado del producto final. Un factor clave es la conexión entre estas restricciones, donde el ajuste de una de ellas puede impactar negativamente en otras, lo que destaca la importancia de una gestión equilibrada de ellas, asegurando el cumplimiento de los objetivos establecidos.

Identificar y analizar las restricciones en el inicio del proyecto es crucial para anticipar riesgos, gracias al diseño de estrategias de migración. De acuerdo al estudio realizado previamente al desarrollo de este proyecto, el ciclo de vida del mismo está, en cierta medida, determinado por las siguientes restricciones o condicionantes:

- **Tiempo:** Los plazos de entrega del proyecto están limitados a los periodos lectivos acordados por los criterios de la Universidad, por lo que el tiempo disponible está limitado a la finalización del curso. Por otra parte, el desarrollo del producto está limitado en tiempo al compaginarse con otras actividades lectivas.

- **Conocimientos:** Los conocimientos adquiridos por el alumno, tanto en términos de tecnologías y herramientas utilizadas durante el desarrollo, como los fundamentos teóricos previos se encuentran en una etapa inicial, lo que implica un proceso de aprendizaje continuo a lo largo del proyecto.

- **Recursos:** Tanto los recursos hardware como software disponibles para el alumno son limitados, lo que implica que el acceso a recursos que requieren de una suscripción para su uso y de los que la universidad no dispone de una licencia educativa, son una limitación de uso de los mismos debido a las características y premisas de desarrollo de este proyecto educativo.

- **Alcance:** El objetivo del TFG debe estar definido claramente y bien delimitado para alcanzar resultados coherentes con las necesidades solicitadas, además de evitar desviaciones innecesarias de la planificación y eliminando esfuerzos dispersos.

Dadas estas restricciones, las medidas llevadas a cabo para reducir completa o parcialmente su impacto en el desarrollo, destaca tanto el diseño de una planificación detallada considerando estas restricciones que asigne de forma óptima los recursos y los plazos establecidos, como también la adquisición temprana de los conocimientos necesarios de las técnicas y de las bases teóricas que determinan un correcto desarrollo, además del monitoreo constante en términos del cumplimiento de las tareas definidas en el periodo asignado a cada una.

## 3.2. Riesgos

El ciclo de vida de un proyecto puede estar condicionado por una serie de factores que pueden provocar el retraso en las entregas, modificando la planificación, e incluso, afectando al cumplimiento de los requisitos establecidos inicialmente. Estos factores son los riesgos[22], referidos a cualquier circunstancia o evento imprevisto que puede impactar negativamente en el desarrollo del proyecto, lo que conlleva la definición de los posibles riesgos y los métodos de evitación y mitigación llevados a cabo con el propósito eliminar o reducir el impacto en el cumplimiento de los requisitos definidos. El análisis de riesgos es un factor clave en el proceso de planificación de un proyecto e implica tener una idea clara y concisa de los riesgos que amenazan el cumplimiento de los objetivos de acuerdo a los requerimientos de la parte interesada. Una correcta gestión de los riesgos implica la identificación de los posibles factores que pueden afectar la finalización del producto, y el análisis de la probabilidad y del impacto de estos riesgos en una etapa inicial, permitiendo así, evitar los efectos producidos gracias a los planes de minimización. En consecuencia, la rápida actuación ante la materialización de un riesgo está determinada por un exhaustivo análisis inicial de los riesgos, así como la importancia de definir una amplia y clara gestión de riesgos, como también el seguimiento de los riesgos durante la etapa de desarrollo del trabajo que garantice la mitigación de estos factores.

Dada la naturaleza del proyecto, el análisis llevado a cabo para determinar los posibles riesgos producidos durante la etapa de desarrollo de este trabajo, se manifiestan seis riesgos que pueden determinar la finalización del proyecto acorde a la planificación establecida y los objetivos definidos. La aparición de estos riesgos suponen un gran impacto, produciendo retrasos en la entrega final y ampliando notablemente los plazos establecidos, por lo que se requiere de una correcta gestión derivada de la elaboración de planes de mitigación.

En la Tabla 3.1 se destaca que la materialización de los riesgos identificados podría, en el peor de los escenarios, requerir posponer la entrega del proyecto a fechas posteriores, alineadas con un nuevo curso académico, lo que conlleva la necesidad configurar una nueva planificación de las etapas de desarrollo y afrontar nuevas limitaciones derivadas del cambio de plazos. Dado que el plazo de entrega de este proyecto está directamente relacionado con el calendario lectivo establecido por la UVa, la ocurrencia de estos riesgos con un alto impacto podría obligar a replantear el proyecto dentro de un nuevo marco temporal acorde a un nuevo curso lectivo, ajustando de nuevo los requisitos y los objetivos iniciales. En consecuencia, resulta crucial realizar una planificación adecuada de las etapas de desarrollo, complementada de la creación de un sólido plan gestión de riesgos, que permiten ajustarse a los plazos establecidos originalmente en la entrega del producto final, minimizando la necesidad de reformular los criterios y cronogramas, lo que a su vez evita inversiones adicionales de tiempo y recursos que pueden prevenirse con una estrategia más robusta desde el inicio.

| Descripción | Probabilidad | Impacto | Plan de Contingencia |
|---|---|---|---|
| La planificación definida no es realista | Media | Alto | Realizar un análisis exhaustivo de los objetivos y establecer una planificación inicial bien definida, contemplando compatibilidad con otras actividades y posibles limitaciones. |
| El tutor del TFG no es activo respondiendo dudas | Baja | Medio | Buscar información en otros recursos como Internet, libros o consultar con otros profesores de la universidad. |
| Incompatibilidad del desarrollo del proyecto por otras actividades prioritarias | Media | Alto | Planificar contemplando imprevistos a corto plazo y reorganizar el horario si los factores afectan a largo plazo, ajustando plazos a la fecha de finalización. |
| No se han superado todos los créditos necesarios en la fecha de presentación del TFG | Muy Bajo | Crítico | Matricularse en el TFG en un año en el que otras asignaturas no representen un impedimento. |
| Cambios en los requisitos del proyecto afectan la planificación | Media | Medio | Definir objetivos y requisitos claramente desde el inicio para establecer una planificación acorde. |
| Pérdida del proyecto o de archivos durante el desarrollo del TFG | Media | Alto | Utilizar copias de seguridad y repositorios para almacenar diferentes versiones, permitiendo la restauración en caso de pérdida. |

Tabla 3.1: Descripción de los riesgos identificados en el proyecto, su probabilidad de ocurrencia e impacto, y las medidas de contingencia planteadas para cada uno de ellos.

## 3.3. Coste

Los costes de un proyecto reflejan los gastos afrontados en la fase de ejecución de un proyecto, determinados por múltiples factores como el personal, los recursos materiales, las inversiones o los costes burocráticos, que contribuyen al presupuesto total del producto, influyendo en la rentabilidad y el éxito obtenidos. La administración de costos es fundamental para medir el rendimiento y determinar si el proyecto se ajusta a los recursos asignados y disponibles, así como mantener el proyecto dentro de los límites presupuestarios. Dado el carácter educativo de este trabajo, en esta sección se analizará los diferentes costes que supone el desarrollo del proyecto, tanto el coste real dado el ámbito en el que se realiza, como el coste que supondría a una empresa real y a su cliente, teniendo en cuenta los recursos software, el personal necesitado para llevar a cabo todas las tareas, los recursos hardware y herramientas que ha necesitado para poder completar dichas tareas, así como los costes indirectos que se han producido derivados del uso de Internet, luz, etc. Para este proyecto se tiene en cuenta que se ha realizado en un periodo de 300 horas y consta de diversos tipos de costes, que determinan el presupuesto final invertido para llevar a cabo el óptimo desarrollo de proyecto, como costes de software, de hardware, personal y costes externos. Para llevar a cabo el calculo de los costes del proyecto, se establece un periodo de seis

meses de desarrollo.

Aunque en este proyecto, se han utilizado herramientas de diseño cuya utilización es gratuita o se dispone de una licencia académicas ofrecida gratuitamente por los recursos proporcionados por la **uva!**, se considera el uso de cada una de ellas mediante la licencia de pago más básica de cada una para establecer los costes reales del proyecto a la empresa. Por un lado, la herramienta de prototipado *Figma*, la herramienta de creación de diagramas *Astah Profesional* también aportando las herramientas necesarias para la creación de diagramas. y finalmente varias herramientas proporcionadas por el paquete *Microsoft Office*.

En el contexto de recursos hardware empleados, se observa el desglose del presupuesto en la Tabla 3.2 estimado en un periodo de desarrollo de 6 meses. El primer recurso hardware utilizado es un ordenador portátil **HP-Laptop 15s-eq2xxx** personal adquirido 2022 que garantizan un probabilidad escasa de que se originen fallos graves que impacten en el desarrollo, con una vida útil a fecha final del proyecto de tres años. Por otro lado, se dispone de un monitor **MSI G274F** en la fecha de inicio de desarrollo del proyecto, por lo que se ha establecido como un coste directo y asociado al desarrollo de este proyecto.

En términos de costes de personal, la realización del proyecto ha sido llevada a cabo únicamente por el alumno, que ha cumplido todos los roles de desarrollo, pero la estimación del coste de personal se tiene en cuenta el sueldo medio un analista de sistemas y un desarrollador *full-stack*, que son los roles que más se ajustan a las competencias realizadas por el alumno. Para los dos tipos de desarrolladores que se necesitan para el desarrollo de este proyecto, se tiene en cuenta el sueldo medio anual de un analista de sistemas en España[1][2] para una jornada de 40 horas semanales. Además se considera lo que a la empresa le supone dichos trabajadores al año debido a las cotizaciones sociales (impuestos, seguridad social, etc) alrededor del 33 % adicional al salario bruto.

Por otra parte, los costes relacionados con Internet se tiene en cuenta el uso generalizado de una conexión de red personal de fibra óptica. En cuanto a los costes derivados de la luz se han tenido en cuenta el consumo estimado de tres elementos: el portátil, el monitor y un uso de luz por dos horas diarias. Por otro lado, los gastos en transportes realizados para acudir a reuniones presenciales con el tutor y gestiones relacionadas con el proyecto han supuesto la realización de cuatro viajes descartando este coste como asociado a los supuestos de una empresa. También, cabe destacar el coste de matriculación, que está determinado por la realización de dos matriculas en cursos diferentes, donde en el curso 2023-2024 se concedió una beca del Ministerio de Educación y Formación Profesional, afrontando el coste de esta matricula y la del curso 2024-2025, cuyo coste es similar al anterior, afectando unicamente a los costes reales. Por último, una hipotética empresa debe alquilar una oficina para llevar a cabo las tareas de desarrollo y cualquier otra que afecte al proyecto, suponiendo el alquiler de una oficina pequeña en Valladolid y añadiendo este coste a los asumidos por la empresa.

El coste total de este proyecto, junto con el desglose siguiendo los apartados mencionados anteriormente, se detalla en la Tabla 3.2. El presupuesto simula el desarrollo del producto en un

[1] https://www.jobted.es/salario/analista-sistemas
[2] https://es.talent.com/salary?job=desarrollador+full+stack

| Recurso | Unidades | Coste Unidad | Coste Real | Coste Empresa | Coste Cliente |
|---|---|---|---|---|---|
| Figma | 6 meses | 15 €/mes | 0 € | 90 € | |
| Astah Profesional | 6 meses | 11,99 €/mes | 0 € | 71,94 € | |
| Microsoft Office | 6 meses | 99 €/año | 0 € | 99 € | |
| **Total Software** | | | **0 €** | **260,94 €** | **365,32 €** |
| Portátil HP | 6 meses | 670 € | 0 € | 670 € | |
| Monitor MSI | 6 meses | 180 € | 180 € | 180 € | |
| **Total Hardware** | | | **180 €** | **850 €** | **1190 €** |
| Analista | 6 meses | 34.900 €/año | 0 € | 23.208,5 € | |
| Desarrollador Full-Stack | 6 meses | 33.000 €/año | 0 € | 21.945 € | |
| **Total Personal** | | | **0 €** | **44.156 €** | **61.818,4 €** |
| Internet | 6 meses | 30€/mes | 180 € | 180 € | |
| Luz | 0,815 kWh | 0,1417 €/kWh | 40 € | 40 € | |
| Transporte | 4 viajes | 15 €/viaje | 60 € | 0 € | |
| Matriculación | 1 ud. | 223 €/matr. | 223 € | 0 € | |
| **Total Indirectos** | | | **503 €** | **220 €** | **308 €** |
| Oficina | 1 oficina | 400 €/oficina | 0 € | 2400 € | |
| **Total Infraestructura** | | | **0 €** | **2400 €** | **3360 €** |
| Coste Total del Proyecto | | | 683 € | 47.886,94 € | 67.041,716 € |

Tabla 3.2: Desglose de los costes asociados al proyecto.

ámbito laboral para una supuesta empresa, junto con los costes reales adecuados al marco del proyecto educativo que han supuesto para el alumno el desarrollo de este trabajo. En cambio, el coste para el cliente debe ser mayor, ya que se debe suponer un beneficio para la empresa sobre los costes totales después de impuestos, por lo que se estima un beneficio en torno al 40 % sobre el coste que supone para la empresa, que permite asumir riesgos y costes de desarrollos posteriores debido a la extensión o modificación del plan inicial para el que se ha supuesto este presupuesto.

## 3.4. Planificación

Este proyecto ha tenido tres configuraciones diferentes de la planificación, que se han ido modificando debido a diferentes factores personales y externos que se comentarán a continuación. La primera planificación configurada para el este proyecto, se desarrolla en un contexto que requiere compatibilizar su realización con las prácticas curriculares y el estudio de dos asignaturas pendientes, las cuales deben ser aprobadas antes de la entrega final del TFG. Esta situación implica una planificación rigurosa y ajustada debido a las limitaciones de tiempo disponibles, como también adecuarla a la carga de trabajo de 300 horas estipuladas en la guía docente anexa al TFG.

Todas las planificaciones de este trabajo se han diseñado utilizando la estrategia de desarrollo *Scrum*[23], un marco de gestión de proyectos que facilita la estructuración y gestión del trabajo mediante una serie de principios y valores como la adaptabilidad, la transparencia y la mejora continua. Esta estrategia está basada en los conceptos de la metodología Ágil[24], un enfoque que organiza el trabajo en cortas iteraciones, promoviendo la colaboración entre los miembros del equipo y adaptándose rápidamente a los cambios, siguiendo un ciclo planificación, ejecución y evaluación. La estrategia utilizada implica la organización del proyecto en iteraciones o *sprints*, un

periodo de tiempo que permite determinar las prioridades y las estimaciones de esfuerzo necesarias para completar las tareas. Cada iteración o *sprint* tiene una duración fija, en este caso, un periodo de dos semanas para cada uno. Esto permite ajustar las prioridades, estimar esfuerzos, y realizar evaluaciones periódicas del progreso para garantizar el cumplimiento de los objetivos planteados. La planificación inicial planteada para este proyecto comprende 9 *sprints* detallados en la Tabla 3.3, donde se esclarece la duración de 2 semanas por cada uno. Esta periodicidad facilita el seguimiento del progreso y asegura que cada tarea cuente con un periodo de tiempo razonable para ser completada, evitando sobrecargas de agendas del alumno y el profesor, como también incompatibilidades con otras actividades. Al final de cada *sprint*, se realiza una reunión para evaluar los avances alcanzados y reajustar las prioridades si fuera necesario.

En la distribución de tareas planificadas para cada *sprint*, cada actividad incluye subtareas de duración comprendida entre uno y dos días, que determinan los hitos de los que se compone cada una, y buscan mantenerse dentro del marco temporal establecido, a excepción de dos tareas más complejas, la implementación y la memoria, cuya duración se extiende por tres semanas debido a su complejidad y naturaleza iterativa. Cada *sprint* se compone de una serie de actividades que se deben completar al final de cada uno. Cada actividad comprende diferentes hitos, y está determinado por un periodo de una o dos semanas, duración que se ajusta a la establecida para cada *sprint* (dos semanas), con la intención de establecer actividades sencillas que no se excedan en el tiempo, en cambio, hay dos actividades más compleja de tres semanas, la implementación y la memoria, que abarca múltiples subtareas con una duración de uno a dos días.

Teniendo en cuenta la duración estipulada para este proyecto y carga de trabajo que abarca cada tarea, basándose en la complejidad y la experiencia del alumno en cada una de ellas, se ha designado un periodo de horas determinado para cada actividad, permitiendo completar el proyecto en 300 horas, dedicando a cada tarea un tiempo diferente:

- **Conocimientos**: 25 horas para adquirir los conocimientos necesarios sobre las tecnologías y herramientas a emplear.
- **Modelos**: 75 horas para el diseño y creación de los modelos de negocio, dominio e interacción.
- **Implementación**: 100 horas para la implementación del código del proyecto, al ser una de las fases más exigentes.
- **Interfaces y pruebas**: 25 horas para el desarrollo de interfaces y las pruebas correspondientes.
- **Documentación memoria**: 75 horas dedicadas a la elaboración de la memoria, permitiendo documentar adecuadamente el trabajo realizado.

Esta planificación inicial busca abordar todas las actividades esenciales para cumplir los objetivos del proyecto en el tiempo estipulado. Según los cálculos realizados y considerando la necesidad de compaginar este proyecto con otras actividades, se plantea una dedicación de aproximadamente tres horas diarias durante los cuatro meses estimados de duración del proyecto, distribuidos en 25 días de trabajo efectivo cada mes. Además, esta planificación inicial busca

| Nombre del Sprint | Fecha Inicio | Fecha Fin |
|---|---|---|
| Sprint 1 | 26/02/24 | 10/03/24 |
| Sprint 2 | 11/03/24 | 24/03/24 |
| Sprint 3 | 25/03/24 | 07/04/24 |
| Sprint 4 | 08/04/24 | 21/04/24 |
| Sprint 5 | 22/04/24 | 05/05/24 |
| Sprint 6 | 06/05/24 | 19/05/24 |
| Sprint 7 | 20/05/24 | 02/06/24 |
| Sprint 8 | 03/06/24 | 16/06/24 |
| Sprint 9 | 17/06/24 | 30/06/24 |

Tabla 3.3: Organización de sprints de la primera planificación realizada desde Marzo hasta Junio.

asegurar la finalización de todas las actividades esenciales, y también establecer una base sólida para gestionar el tiempo y los recursos, facilitando la adaptación a potenciales cambios.

Debido a incompatibilidades derivadas de otras actividades externas, como por ejemplo el desarrollo en paralelo de las prácticas universitarias y las actividades pendientes relativas a las dos asignaturas no superadas en la fecha del desarrollo de este proyecto, la planificación sufrió una modificación, generando una segunda planificación en una fecha posterior y relativa a un nuevo curso lectivo. Estos inconvenientes surgen en una fase temprana del proyecto, por lo que permite establecer una planificación cuyas bases son idénticas a la planificación inicial. Ante esta nueva planificación, y de acuerdo a la reunión producida con la parte interesada (el tutor), se modifican los requisitos iniciales, estableciendo otros nuevos y fijando unos objetivos claramente definidos así como alcance, que permite realizar una nueva configuración más detallada y realista que permita lograr exitósamente los objetivos definidos acorde a las nuevas etapas marcadas. La nueva planificación utiliza la misma metodología seguida en la planificación inicial, *Scrum*, para garantizar el cumplimiento de cada uno de los hitos en las etapas establecidas. Sin embargo, esta configuración establece una nueva duración de los *sprints* en una semana, asegurando tanto el seguimiento detallado de las actividades realizadas, así como prevenir y actuar rápidamente ante potenciales cambios de los requisitos o incompatibilidades. Además, esta planificación tiene en cuenta periodos de vacaciones y no lectivos, ajustando las iteraciones a estas situaciones, e incluso adaptándose a la nueva disponibilidad del alumno, con una mayor dedicación a este proyecto.

La segunda planificación realizada en este proyecto, considera el inicio en base a los avances producidos en la anterior etapa de desarrollo que resultó interrumpida, por lo que se debe ajustar a las 300 horas estipuladas teniendo en cuenta, tanto las 100 horas aproximadas que se emplearon durante la anterior planificación, como también los avances realizados que evitan partir de cero y que garantizan una base bastante sólida para el desarrollo. En esta etapa, se consiguieron numerosos avances tanto en la fase de análisis del producto, como en las fases de diseño e implementación. A continuación, se describen los avances más importantes logrados en la primera etapa:

- **Requisitos, Casos de Uso y Dominio**: Se establecieron los requisitos definidos previamente, consolidando su alcance e importancia, y se determinaron los casos de uso

derivados de cada uno de ellos, permitiendo así un robusto conocimiento del dominio del proyecto. Estos avances permiten simplificar las tareas relacionadas con estas actividades, implicando unicamente la modificación de aquellos requisitos manifestados en la nueva definición del proyecto.

- **Implementación**: Las bases del juego como los movimientos, los personajes y su interacción con el entorno, o los resultados de cada partida se implementaron en la primera etapa, permitiendo continuar la implementación del videojuego sobre unos cimientos establecidos, requiriendo la puesta en marcha de los nuevos requisitos y de las actividades no logradas en la anterior planificación.

- **Interfaces**: Los elementos básicos de la interfaz del juego se definieron previamente a la interrupción de la primera iteración. Esto implica el desarrollo de interfaces referentes a la navegabilidad y las diferentes pantallas de los menús, descartando el diseño de la mayoría de elementos relacionados con la pantalla referente a la dinámica de juego.

- **Documentación memoria**: Todos los avances producidos durante el desarrollo del proyecto en la primera etapa, se plasmaron en el informe de la memoria, esclareciendo los progresos alcanzados y permitiendo retomar el proyecto con ideas claras sobre ese desarrollo obtenido.

Considerando el progreso alcanzado hasta la fecha de reestructuración de la planificación del proyecto, se procede a establecer una nueva configuración planificación determinada tanto por las actividades derivadas de los requisitos pendientes y de los nuevos, como de la asignación de cada actividad a cada *sprint*. Conforme al estudio realizado de cada una de las actividades pendientes, se ha determinado la necesidad de establecer 15 *sprints* semanales, en los que se limita el rango de actividades completadas al final de cada una de las iteraciones. El detalle de cada *sprint* se muestra en la Tabla 3.4, donde se considera, como se ha mencionado anteriormente, los periodos no lectivos y las vacaciones de la parte interesada y del alumno. Cabe mencionar, la existencia de un *sprint* extraordinario, establecido como ampliación y prevención ante nuevos imprevistos que pudieran comprometer el cumplimiento del plazo establecido, suponiendo la ampliación y modificación de esta planificación.

| Nombre del Sprint | Fecha Inicio | Fecha Fin |
|---|---|---|
| Sprint 1 | 05/08/24 | 11/08/24 |
| Sprint 2 | 19/08/24 | 25/08/24 |
| Sprint 3 | 26/08/24 | 01/09/24 |
| Sprint 4 | 02/09/24 | 08/09/24 |
| Sprint 5 | 23/09/24 | 29/09/24 |
| Sprint 6 | 30/09/24 | 06/10/24 |
| Sprint 7 | 07/10/24 | 13/10/24 |
| Sprint 8 | 14/10/24 | 20/10/24 |
| Sprint 9 | 21/10/24 | 27/10/24 |
| Sprint 10 | 28/10/24 | 03/11/24 |
| Sprint 11 | 04/11/24 | 10/11/24 |
| Sprint 12 | 11/11/24 | 17/11/24 |
| Sprint 13 | 18/11/24 | 24/11/24 |
| Sprint 14 | 25/11/24 | 01/11/24 |
| Sprint 15 | 02/11/24 | 08/12/24 |

Tabla 3.4: Organización de *sprints* de la segunda planificación realizada desde Agosto hasta Diciembre.

Los *sprints* en esta segunda planificación, son de una semana, lo que conlleva la organización en un mayor número de tareas con periodos de tiempo más cortos. La división de tareas en esta planificación tiene en cuenta los avances alcanzados anteriormente, en el primera planificación, por lo que en esta nueva organización, se incluye una etapa de ajuste de la implementación del juego tanto de los nuevos requisitos fijados en esta versión, como también de las etapas previas de análisis y modelado realizadas con un alto detalle de diseño. Estas actividades se han fijado, en la mayoría de casos, con una duración de una semana en coincidencia con la duración de los *sprints*, aunque existen excepciones en tareas de mayor complejidad en las que se ha establecido la duración en dos semanas. A su vez, estas actividades, están organizadas en subtareas diarias, estableciendo el objetivo esperado de cada una y permitiendo realizar un seguimiento más activo que da la posibilidad de una reacción temprana ante cualquier inconveniente. Las tareas de más complejidad hacen referencia al contexto de la implementación y desarrollo del videojuego, debido a la identificación tras el análisis inicial de que estas actividades suponen una alta carga de trabajo favorecida por la complejidad que suponen. También, la nueva planificación separa el diseño de cada etapa de modelado en diferentes actividades, permitiendo desacoplar estas tareas y la realización independiente de cada una de ellas.

La segunda planificación, se realizó ajustando a los periodos de vacaciones previstos por el estudiante, pero a estos periodos se les sumó otra etapa de aproximadamente dos semanas de vacaciones no previstas, a lo que se les añadió una serie de problemas personales acontecidos durante las fechas de desarrollo del proyecto, que provocaron la realización de una tercera y última planificación en Septiembre de 2024. Esta nueva planificación presenta características muy

| Nombre de la actividad | Sprint | Fecha Inicio Estimada | Fecha Fin Estimada |
|---|---|---|---|
| Casos de Uso | Sprint 1 | 23/09/24 | 29/09/24 |
| Modelo de Dominio | Sprint 2 | 30/10/24 | 06/10/24 |
| Modelo de Interacción | Sprint 3,4 | 07/10/24 | 20/10/24 |
| Modelo de Diseño | Sprint 4,5 | 21/10/24 | 03/11/24 |
| Ajustar nueva implementación | Sprint 6,7 | 04/11/24 | 17/11/24 |
| Enfrentamientos y algoritmos | Sprint 8,9 | 18/11/24 | 01/12/24 |
| Interfaces | Sprint 10 | 02/12/24 | 08/12/24 |
| Bases de Datos | Sprint 11 | 09/12/24 | 15/12/24 |
| Pruebas | Sprint 12 | 16/12/24 | 22/12/24 |
| Redacción Memoria | Sprint 13,14 | 23/12/24 | 05/01/25 |
| Ajustes | Sprint Extra | 06/01/25 | 19/01/25 |

Tabla 3.5: Desglose de tareas de la tercera planificación realizada entre Septiembre de 2024 y Enero de 2025, con periodos comprendidos entre 1una y dos semanas.

similares a la segunda planificación en cuanto a términos de tareas y sprints realizados se refiere. La modificación se basa en retrasar los periodos establecidos para cada tarea en aproximadamente un mes, para las tareas posteriores a Agosto, estableciendo el fin de esta planificación en la segunda semana de Enero de 2025. Por lo que, se tiene en cuenta las tareas realizadas y completadas hasta dicha fecha, como son la realización del modelo de negocio y del modelo de requisitos. La Tabla 3.5 describe la nueva planificación realizada en Septiembre de 2024, indicando los sprints creados y las actividades relativas a cada uno de los *sprints* establecidos.

La planificación de las tareas, como se detalla visualmente en el diagrama de Gantt de la Figura 3.1, tienen en cuenta vacaciones y otros periodos de ausencia, completando un total de cuatro meses de trabajo estimado para llevar a cabo la finalización de este proyecto. De estos cuatro meses de duración, el trabajo efectivo está comprendido en 15 semanas coincidiendo con el número de *sprints* fijado en la estrategia de planificación establecida.

La planificación establecida para llevar a cabo las actividades del proyecto contempla un total de 200 horas dedicadas a la ejecución directa de tareas específicas, complementadas con 100 horas adicionales dedicadas a la realización de la primera etapa del proyecto, ambas destinadas a alcanzar las 300 horas fijadas como objetivo general. A la planificación semanal establecida se añade el ajuste a periodos de trabajo de cinco días semanales, y en consecuencia dos días de descanso. Esta distribución tiene en cuenta no solo la limitación de tiempo disponible y la estimación de completar el proyecto en un periodo aproximado de tres meses y medio de trabajo efectivo, sino también la mayor disponibilidad del alumno durante el desarrollo del proyecto gracias a la eliminación de cargas de trabajo paralelas e independientes de este trabajo, y también a la seguridad consolidada de evitar riesgos derivados con asuntos académicos, como son la presencia de asignaturas no completadas o la realización de las prácticas, con lo que estos riesgos suponen. Este incremento en la disponibilidad ha permitido ajustar la carga de trabajo diaria a un promedio de tres horas, optimizando así el uso del tiempo y garantizando un progreso constante en la consecución de los objetivos planteados.

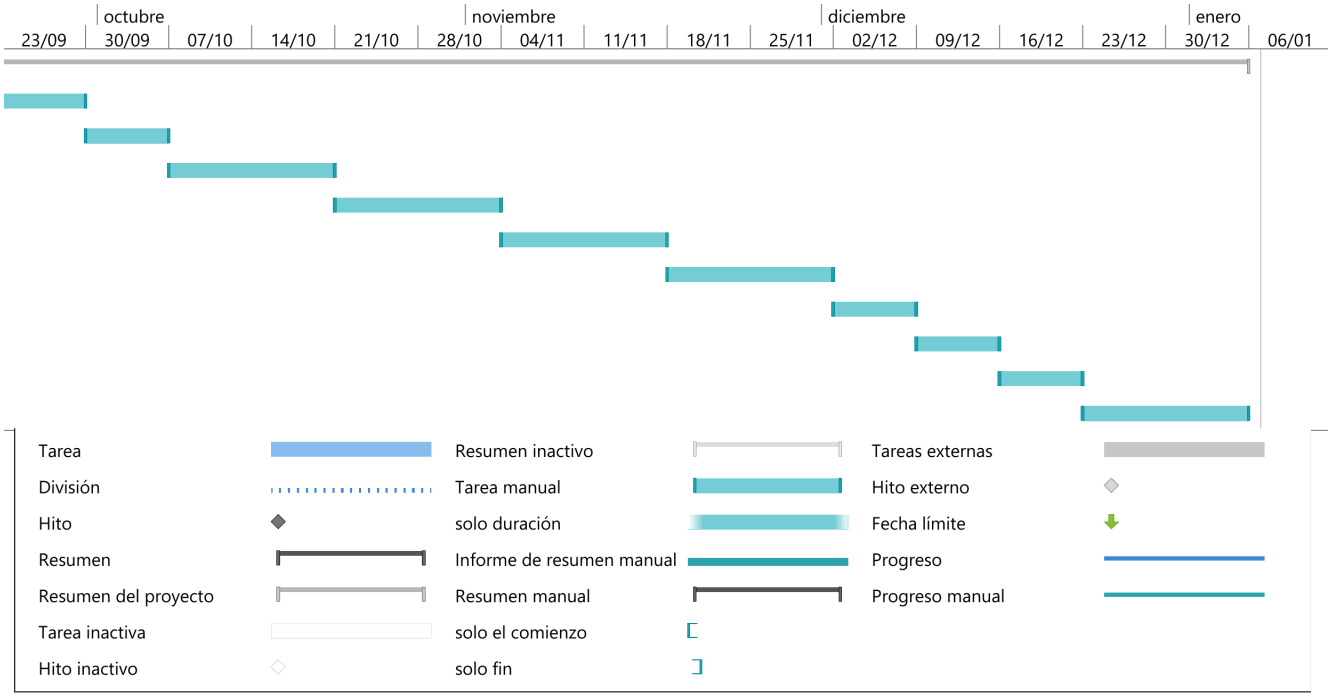


Figura 3.1: Diagrama de Gantt de la primera planificación entre Septiembre 2024 y Enero 2025 con el desglose del periodo empleado en cada una de las tareas.

## 3.5. Variación de planificación

La existencia de un *sprint* extraordinario en la Tabla 3.5, está creado con la intención de abarcar cualquier imprevisto en una duración de aproximadamente dos semanas, cubriendo corrección de errores simples y pequeños detalles de la memoria. Sin embargo, este *sprint* tiene en consideración una etapa de vacaciones (Navidad), que implica la finalización y entrega real en fechas posteriores a estas vacaciones, como también la limitación de las reuniones con el tutor en este periodo, lo que puede suponer un impedimento y generar retrasos en las correcciones finales debido a la falta de comunicación rápida consecuencia del horario no lectivo.

De acuerdo a la tercera y última planificación planteada, se ha debido hacer uso del *sprint* extraordinario para ampliar el plazo de entrega a una fecha posterior de la fijada en Enero de 2025. Esta última variación se debe a las incompatibilidades de reunión con el tutor, al coincidir con un periodo de exámenes en la Universidad, y otros asuntos personales tanto del tutor como del alumno. Durante este nuevo periodo, el trabajo realizado en este proyecto se ha limitado a la redacción del informe y corrección de leves errores observados en la aplicación basados en las múltiples pruebas realizadas, pretendiendo ajustarse a los criterios establecidos por los requerimientos del informe, basados en la guía docente referente a la entrega del TFG. En resumen, la planificación establecida ha sufrido una variación y ampliación de aproximadamente dos semanas, teniendo en cuenta el periodo de vacaciones acontecido durante esta nueva fecha, y ajustándose a las limitaciones de tiempo del tutor durante este nuevo periodo, lo que ha provocado una comunicación menos continua.

## 3.6. Herramientas utilizadas

La ejecución de este proyecto ha requerido el uso de una amplia variedad de herramientas que han sido seleccionadas cuidadosamente para abordar las diferentes etapas de trabajo de manera eficiente y eficaz. Estas herramientas han sido elegidas en función de criterios como la facilidad de uso, compatibilidad con las tecnologías empleadas, la disponibilidad de versión gratuita, o en su defecto, de una licencia académica, y de la capacidad para optimizar la planificación y seguimiento del progreso. La combinación de las herramientas tanto técnicas como de gestión ha permitido asegurar un flujo de trabajo continuo y sencillo, maximizando los resultados dentro del marco temporal y los recursos disponibles. Entre estas, se encuentran herramientas con una finalidad muy dispar, tanto herramientas de diseño de interfaces y elementos, plataformas de desarrollo de videojuegos o mantenimiento y codificación, como también diseño de diagramas, gestión de bases de datos o control de versiones.

El diseño de los diagramas y las tablas de requisitos derivadas de los modelos de análisis e implementación empleados se han realizado mediante una herramienta de licencia sujeta a suscripción de pago, de la que se dispone de una clave académica que permite el uso gratuito e ilimitado. Esta herramienta es *Astah*[3], una plataforma que permite realizar el diseño de diagramas de casos de uso, de secuencia o de estados, entre otros, mediante el lenguaje de modelado UML[25], un lenguaje estándar utilizado para visualizar un plan arquitectónico de elementos reflejado en actividades, procesos de negocio y esquemas de datos. En este proyecto aborda un producto derivado de Astah llamado *Astah Profesional*[4], la herramienta que se ha utilizado para el diseño de los diagramas de modelado necesarios para la correcta implementación, como son el modelo de dominio general, el de la base de datos, los casos de uso, los diagramas de secuencia de cada caso de uso, los diagramas de estados y de actividad y los diagramas de despliegue. También se ha usado para definir los requisitos tanto funcionales como los no funcionales, y los riesgos del proyecto.

Para este proyecto, se han utilizado diversas herramientas especializadas que, en conjunto, cubren las necesidades de diseño en distintas fases del desarrollo, desde la creación de gráficos pixelados hasta la elaboración de prototipos interactivos, estas herramientas han permitido establecer una línea visual coherente y funcional, como el diseño de las interfaces y sus elementos, los prototipos o los bocetos. Estas herramientas facilitan la creación, edición y visualización de componentes gráficos esenciales, asegurando una experiencia de usuario intuitiva y estéticamente, cada una con un propósito específico que ha contribuido al desarrollo de las interfaces y elementos gráficos. A continuación se describen cada una de las herramientas utilizadas, detallando el uso que se ha producido de cada una de ellas en este proyecto:

- **Figma**[5]: es una atractivo e innovador editor de gráficos vectoriales diseñado para paginas web y aplicaciones, utilizada por diseñadores software y agencias de comunicación para el diseño de sus *apps*. Esta plataforma[26] es muy popular en el ámbito de diseño gráfico de páginas web e interfaces de aplicaciones, e incluso, creación de presentaciones, que permite

[3] https://astah.net/
[4] https://astah.net/products/astah-professional/
[5] https://www.figma.com/

la colaboración simultánea entre varios miembros de un equipo. Es una herramienta gratuita en su versión básica, y dispone de una versión de pago, utilizada en este proyecto mediante la licencia académica disponible para los estudiantes. Figma es la herramienta utilizada en la creación de los prototipos de las interfaces iniciales, con un nivel de detalle muy aproximado a los resultados finales.

- **Adobe Color**[6]: es una herramienta online que permite experimentar con la teoría del color[27] para conocer la combinación perfecta entre varios colores en base a diferentes reglas cromáticas. En esta aplicación se pueden crear y establecer paletas de colores en base a estas reglas definidas, permitiendo la modificación de los tonos, la saturación o armonías entre colores. En este proyecto, *Adobe Color* ha sido utilizada para la creación de la paleta de colores que componen los diferentes elementos de la interfaz de navegabilidad y de la interfaz del juego.

- **Pixelorama**[7]: Es una potente herramienta de código abierto para crear *pixel art*[28], que permite la creación de diversas expresiones artísticas como *sprites*, mosaicos o animaciones. Es una plataforma de gran capacidad en la que desarrollar formas, personajes y otros elementos mediante el diseño pixel de forma sencilla y una gran cantidad de mecanismos que permiten el diseño de multiples capas y animaciones. Esta herramienta permite la exportación a formatos de archivo como *png* o *jpg* *Pixelorama* ha sido determinante en este proyecto en la realización de las apariencias de los personajes (jugador y enemigo), sus animaciones y secuencias de movimiento basadas en obras realizadas previamente por otros artistas, y además, el diseño de *tiles* o elementos que componen el mapa del juego (muros). También ha favorecido el diseño de los componentes de las interfaces de navegación.

- **Inkscape**[8]: es un editor de gráficos vectoriales de código abierto que utiliza el formato de archivos Scalable Vector Graphics[29] (SVG) con una interfaz sencilla y fácil de usar a la vez que una potente herramienta de creación de imágenes y diseños vectoriales. En complementación a *Pixelorama*, *Inkscape* también ha sido determinante en el diseño de los menús y los componentes de las interfaces de navegación de este proyecto. El uso de esta herramienta ha permitido el desarrollo de los menús tanto de la pantalla principal como de las pantalla de creación y carga de partidas, o de las configuración del la pantalla que presenta la dinámica del juego.

- **Recraft**[9]: es una innovadora herramienta impulsada por IA que permite la creación de gráficos vectoriales de increíble calidad a partir de la introducción de texto. Es una plataforma online y gratuita, que destaca por su facilidad de uso y la gran capacidad intuitiva con gran potencial en la generación de ilustraciones, imágenes 3D, vectores, iconos o logos muy precisas. Esta aplicación se ha utilizado como herramienta de generación de imágenes 2D, que permiten complementar las interfaces de usuario del videojuego con los otros componentes diseñados cuidadosamente.

- **Soundtrap**[10]: es una herramienta trabajo de audio digital basada en la nube que está

[6] https://color.adobe.com/
[7] https://orama-interactive.itch.io/pixelorama
[8] https://inkscape.org/
[9] https://www.recraft.ai/
[10] https://www.soundtrap.com/

reinventando la creación de música. Con esta plataforma, diseñada por productores musicales, compositores y expertos en audio, permite dar rienda suelta a todo el potencial creativo del usuario. Esta herramienta se ha utilizado para el desarrollo de pistas música que acompañaran el curso del juego permitiendo establecer una inmersión de usuario dentro del mismo.

Por otro lado, este trabajo cuenta con plataformas fundamentales para el desarrollo del videojuego, utilizadas como herramientas de programación e implementación de las funcionalidades del juego, como también un sistema de control de versiones que asegura un registro detallado de los cambios realizados y facilita la recuperación en caso de errores. Estas herramientas facilitan la escritura y depuración de código, garantizando tanto su calidad como una organización clara y estructurada de las modificaciones realizadas. El proyecto ha requerido del uso de una plataforma dedicada exclusivamente al desarrollo de videojuegos, apoyada de una aplicación para la mejora y estructuración de los archivos del proyecto, y de una herramienta online que permite mantener un registro de las diferentes versiones y modificaciones producidas a lo largo de la etapa de vida del proyecto. Las tres herramientas utilizadas en la implementación de código se describen a continuación:

- **Godot**[11]: es un motor de juegos multiplataforma[12] de código abierto y en constante evolución para adaptarse a las necesidades del mercado, que permite juegos en 2D y 3D en varios lenguajes de programación, presenta un enfoque modular, interfaz intuitiva y utiliza nodos para facilitar la experiencia de desarrollo y herencia. Esta plataforma incluye un motor gráfico 2D y uno 3D independientes y combinables entre si, e integra un lenguaje de programación propio, *GDScript* que integra una sintaxis ligera y derivada del lenguaje *Python*. Godot integra un editor de juego completo con herramientas para satisfacer las necesidades más comunes como animaciones, mapas de mosaicos y sombras, depuración, un sistema de física o soporte para gráficos 3D, entre otros. Esta herramienta es la plataforma fundamental del desarrollo de este proyecto, pues se ha utilizado para el desarrollo integral del videojuego en consonancia con las interfaces y sus elementos diseñados acorde a los prototipos que complementan el diseño del juego en dos dimensiones, con todas las interfaces necesarias para ofrecer un videojuego intuitivo de gran calidad. Cada interfaz hace referencia a una escena diferente desarrollada en este motor de juegos, en la que se ha implementado los diferentes elementos de los que se compone esta plataforma, como los *tiles* para la creación del mapa de juego (el laberinto), o los *canvas layer* para el diseño de mensajes y configuración de los elementos. También se han creado animaciones de los personajes para recrear sensación de movimiento y transiciones que aportan gran armonía de navegabilidad.
- **Visual Studio Code**[13] (**VSC**): es un editor[14] de código abierto para desarrolladores que permite múltiples funcionalidades como implementación de código, depuración, control integrado de *Git*, resaltado de sintaxis, refactorización de código o estructuración de proyectos. Además, permite la personalización y modificación del estilo del editor,

[11] https://godotengine.org/
[12] https://docs.godotengine.org/es/4.x/index.html
[13] https://code.visualstudio.com/
[14] https://code.visualstudio.com/docs

modificando el tema y atajos de teclado al gusto del usuario. Esta herramienta es compatible con infinitud de lenguajes de programación lo que la hace una plataforma accesible para la mayor parte de desarrolladores, mejorando la codificación de aplicaciones, web y cualquier otro desarrollo e implementación. En este proyecto se ha utilizado VSC en la estructuración del proyecto y como herramienta de conexión con el sistema de control de versiones *Git*, así como un apoyo en la edición de código y depuración gracias a su diseño intuitivo y fácil de usar, y la familiaridad del alumno con este entorno.

- **GitHub**[15]: es un sistema de control de versiones[30] que brinda la oportunidad de alojar proyectos en un repositorio con la capacidad de colaboración entre diferentes miembros del equipo y comparación de versiones del programa. Es una plataforma[31] de software libre que admite el almacenamiento de múltiples tipos de proyectos y archivos, y cuenta con diferentes ramas de almacenamiento así como un control de permisos que da la oportunidad de controlar la actividad de los usuarios, evitando modificaciones imprevistas o con potenciales errores. Esta herramienta se ha utilizado para el control de versiones del videojuego, permitiendo visualizar la estructura y los cambios producidos a lo largo del ciclo de vida del proyecto, y manteniendo una oportunidad de recuperación de versiones en caso de perdida de datos.

A las herramientas de implementación de código, se les añade otras herramientas y tecnologías dedicadas a la administración de bases de datos, adaptadas a las necesidades especificas del videojuego que permiten mantener la correcta gestión de las bases de datos gracias a facilidad de creación y modificación que presentan. La gestión de datos es un componente fundamental en el desarrollo de cualquier aplicación, especialmente en un proyecto como este, donde es necesario almacenar y acceder a información de las partidas y los usuarios manera eficiente y estructurada. Los sistemas de gestión de bases de datos permiten administrar grandes volúmenes de datos de forma organizada, facilitando operaciones como la consulta, actualización y eliminación de registros. Además, las herramientas especializadas para crear y modificar bases de datos simplifican el diseño y mantenimiento de estas estructuras. A continuación se describen las herramientas relacionadas con bases de datos utilizadas en este proyecto:

- **SQLite**[16]: es una biblioteca o base de datos ligera e independiente, compatible con multiples aplicaciones que destaca por su ausencia de una estructura cliente-servidor. Es el sistema de bases de datos más extendido y utilizado de tipo relacional, cuya funcionalidad[32] se basa en la llamada a subrutinas y funciones reduciendo la latencia de accesos a las bases de datos, e implementa la mayor parte del estándar *SQL-92*[33] e incluye un tipado dinámico. Varios procesos o hilos pueden acceder a la misma base de datos sin problemas, permitiendo múltiples accesos de lectura pueden ser servidos en paralelo, pero solo un acceso de escritura puede ser servido simultáneamente. Los tipos de datos admitidos son *integer* para datos numéricos y enteros, *real* para todo tipo de datos numérico, *text* para las cadenas de texto, *blob* para datos de gran tamaño y *null* para valores nulos. En este proyecto se ha utilizado debido a su sencillez, y de forma local para el guardado de las partidas y recuperación posterior, como también la implementación de un sistema de registro y sesión de usuarios que permita desacoplar las partidas de cada usuario y ofrecer una experiencia personalizada.

[15] https://github.com/
[16] https://www.sqlite.org/

- **DB Browser**[17]: DB Browser for SQLite es una aplicación gratuita y de código abierto diseñada para facilitar la creación y administración de las bases de datos que utilizan *SQLite*. Es una herramienta que combina una interfaz muy clara y sencilla de utilizar, basada en tablas de manera que tanto usuarios sin mucha experiencia en la creación y administración de bases de datos, como los desarrolladores más avanzados puedan trabajar cómodamente con sus bases de datos. Esta aplicación permite crear, definir y modificar tablas, indices o archivos, como también editar y buscar entradas y la importación a archivos de múltiples formatos. Esta herramienta ha facilitado la creación y modificación de tablas y la gestión de la base de datos de este proyecto otorgando una experiencia sencilla de manejo de los datos persistentes del videojuego, permitiendo la visualización de los datos de las partidas y usuarios.

Por ultimo, cabe destacar y mencionar las herramientas utilizadas para una correcta y eficiente gestión de la planificación del proyecto, que permiten organizar las tareas, supervisar el progreso y resultados asegurando que se cumplan los objetivos dentro del tiempo y recursos disponibles. Para este proyecto ha sido clave el uso de dos herramientas de gestión, mediante una licencia académica, que facilitan la visualización y el control de las diferentes fases de trabajo. *Microsoft Project*[18] ha sido utilizado como herramienta principal para estructurar y planificar las etapas del proyecto, gracias a sus capacidades para crear diagramas de Gantt, asignar recursos y analizar dependencias entre tareas. Por otro lado, *Microsoft Excel*[19] ha complementado esta planificación generando reportes personalizados de los avances diarios y semanales, además de reflejar una retroalimentación semanal que permita el seguimiento de las etapas del proyecto. La combinación de estas herramientas ha permitido mantener una visión clara del progreso y ajustar la planificación según las necesidades emergentes.

[17] https://sqlitebrowser.org/
[18] https://www.microsoft.com/es-es/microsoft-365/project/project-management-software
[19] https://www.microsoft.com/es-es/microsoft-365/excel

# Capítulo 4

# Análisis

En este capítulo, se realiza el planteamiento de la solución que va a resolver el problema descrito en este proyecto. El análisis software consiste en analizar el sistema, en este caso el videojuego del laberinto, resuelto mediante algoritmos de búsqueda y aprendizaje por refuerzo, y plantear la solución de manera que el software y las técnicas utilizadas permitan obtener el éxito en cuestiones de calidad y el logro de los objetivos requeridos.

Para realizar el análisis de un sistema software, se comienza realizando el estudio de sus requisitos que involucran a cada actor que interactúa con el sistema, reflejando parte de los requisitos funcionales en los casos de uso del el sistema. En las siguientes secciones se va a presentar detalladamente los requisitos funcionales y no funcionales, los casos de uso ajustados a los requisitos y los actores que interactúan con los casos de uso del sistema, y finalmente, el modelo de dominio derivado del análisis de las secciones anteriores.

## 4.1. Actores

Tras el análisis del problema planteado, en el sistema desarrollado para el videojuego del laberinto se extraen varios actores, donde se distinguen entre tres tipos de actores principales y un actor secundario:

- **El actor Usuario** es el actor principal, encargado de interactuar directamente con el sistema y con las diferentes interfaces de las que se compone el videojuego, como crear partidas, establecer configuraciones o indicar los movimientos que realiza el jugador.
- **El actor Jugador** es el actor encargado de tratar de resolver el laberinto cuando alcanza la moneda mediante el método más rápido y óptimo posible para obtener la victoria. También se encarga de evitar ser alcanzado por el enemigo en partidas con Modo Enfrentamiento habilitado.
- **El actor Enemigo** encargado de alcanzar al jugador evitando que este alcance la moneda y por ello no resuelva el laberinto, otorgando la victoria al enemigo.

- **El actor BaseDatos**, el actor secundario, tratará los datos persistentes almacenados en una base de datos. En ella se podrá guardar las partidas con sus configuraciones del entorno y los datos de progreso.

El usuario es el actor principal que se encarga de crear partidas, indicar sus configuraciones, o llevar a cabo las diferentes funcionalidades que permite realizar durante el transcurso de la partida. A su vez, comparte relación de casos de uso con el único actor secundario del sistema, BaseDatos, que está relacionado con los casos de uso referentes a almacenar y recuperar partidas de la base de datos, proporcionando el servicio externo de la persistencia de datos y tratamiento de los mismos.

## 4.2. Requisitos

Un requisito es la propiedad que ha de exhibir el software a desarrollar o adaptar para resolver un problema o conseguir un objetivo determinado, conectando el dominio del problema con el dominio de la solución. Se diferencia entre requisitos del usuario, sin detalles técnicos y requisitos del sistema, con una descripción detallada y el punto de partida para el diseño del sistema. Los requisitos del sistema, a su vez, están compuestos por dos tipos, los requisitos funcionales que determinan los servicios que el sistema debe proporcionar y los requisitos no funcionales que son restricciones que afectan a los servicios proporcionados

### 4.2.1. Requisitos funcionales

Los requisitos funcionales (RF) describen la funcionalidad que debe proporcionar el sistema. De acuerdo al estudio realizado, se han extraído los siguientes requisitos funcionales de este sistema:

- **RF01:** El sistema debe permitir al usuario crear y configurar los datos de la partida.
- **RF02:** El sistema debe permitir al usuario iniciar un nuevo juego al crear una nueva partida o tras finalizar el juego anterior.
- **RF03:** El sistema debe permitir al usuario seleccionar entre modos de juego (Modo Solitario o Modo Enfrentamiento).
- **RF04:** El sistema debe permitir al usuario seleccionar el algoritmo de búsqueda mediante el cual se resolverá el laberinto.
- **RF05:** El sistema debe permitir al usuario seleccionar el modo interacción (Modo Usuario o Modo Simulación).
- **RF06:** El sistema debe permitir al usuario seleccionar la dificultad del juego.
- **RF07:** El sistema debe permitir al usuario seleccionar el nivel del laberinto.
- **RF08:** El sistema debe permitir al usuario finalizar la partida permitiendo guardar el resumen del progreso obtenido.

- **RF09:** El sistema debe permitir al usuario guardar la partida durante el transcurso de la misma, almacenando el progreso obtenido y la configuración del entorno.
- **RF10:** El sistema debe permitir al usuario continuar el progreso de una partida almacenada anteriormente.
- **RF11:** El sistema debe permitir al usuario reiniciar la partida durante el transcurso de la misma eliminando progreso alcanzado.
- **RF12:** El sistema debe permitir al usuario cambiar el modo de interacción entre Modo Usuario y Modo Simulación durante el transcurso de la partida.
- **RF13:** El sistema debe permitir al usuario cambiar la dificultad del juego durante el transcurso de la partida.
- **RF14:** El sistema debe permitir al usuario modificar el algoritmo de búsqueda entre los disponibles para la resolución del laberinto durante el transcurso de la partida.
- **RF15:** El sistema debe permitir al jugador buscar la moneda y crear las trayectorias que lo conectan mediante algoritmos de búsqueda.
- **RF16:** El sistema debe permitir al jugador desplazarse para alcanzar la moneda y evitar al enemigo en base a los cuatro movimientos posibles.
- **RF17:** El sistema debe permitir al enemigo buscar al jugador y crear las trayectorias que lo conectan mediante algoritmos de búsqueda.
- **RF18:** El sistema debe permitir al enemigo desplazarse para alcanzar al jugador en base a los cuatro movimientos posibles (arriba, abajo, derecha o izquierda).
- **RF19:** El sistema debe permitir al usuario controlar al jugador mediante entradas de teclado cuando el juego se encuentra en Modo Usuario.
- **RF20:** El sistema debe incluir una moneda situada en una posición estática durante toda la partida.
- **RF21:** El sistema debe permitir al usuario navegar a través los menús e interfaces pudiendo seleccionar opciones utilizando las teclas de flecha y la tecla 'Enter' del teclado.
- **RF22:** El sistema debe incluir un menú principal que permita al usuario seleccionar entre las opciones de 'Crear Partida', 'Opciones de juego', 'Instrucciones', y 'Salir'.
- **RF23:** El sistema debe permitir al usuario la elección de una apariencia para el enemigo y para el jugador antes de iniciar la partida.

### 4.2.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) describen las restricciones de la funcionalidad del sistema. De acuerdo al estudio realizado, se han extraído los siguientes requisitos no funcionales para este sistema:

- **RNF01:** El sistema estará implementado mediante el entorno de desarrollo Godot en su versión 4.0, o posterior.
- **RNF02:** El sistema se compondrá de elementos desarrollados mediante el entorno de diseño Pixelorama.
- **RNF03:** El sistema tendrá un tamaño pre-establecido para cada nivel del laberinto.
- **RNF04:** El sistema estará compuesto dos modos de interacción (Usuario o Simulación).
- **RNF05:** El sistema tendrá definido dos modos de juego (Enfrentamiento o Solitario).
- **RNF06:** El sistema se deberá desarrollar en lenguaje GDscript.
- **RNF07:** El sistema establecerá un tiempo límite para el desarrollo de cada juego.
- **RNF08:** El sistema deberá permitir al jugador realizar la búsqueda de la moneda y del enemigo en base a diferentes algoritmos.
- **RNF09:** El sistema debe componerse de los siguientes elementos principales: un jugador, un adversario, una moneda y un entorno de juego.
- **RNF10:** El sistema debe finalizar el juego cuando el jugador alcance la moneda o sea alcanzado por el enemigo.
- **RNF11:** El sistema deberá obtener una respuesta al movimiento en un tiempo menor a 1 segundo.
- **RNF12:** El sistema debe permitir al usuario configurar fácilmente el entorno.
- **RNF13:** El sistema debe permitir al usuario interactuar con el sistema de forma clara y sencilla.
- **RNF14:** El sistema se compondrá de elementos desarrollados en formato pixel y vectorial.

## 4.3. Casos de Uso

En este sistema se ha diferenciado dos escenarios principales, compuestos por diferentes actores. El primer escenario, es el referente al actor principal *Usuario* y al actor secundario *Base de Datos*. En la Figura 4.1 se puede observar los casos de uso en los que interactúan ambos actores, que son los relativos a la interacción con las interfaces de control de las partidas, como la creación, el guardado o la selección de apariencias. En el caso del guardado y carga interviene el actor Base de Datos permitiendo la conexión a la base de datos y tratamiento de los datos persistentes.

Por otro lado, el sistema está formado por otro escenario referente a las acciones llevadas a cabo por el jugador y por el enemigo, que representan la dinámica del juego. El *jugador* y el *enemigo* son los actores principales que interactuan con el sistema, desempeñando roles fundamentales en el desarrollo del juego. El jugador tiene como objetivo principal alcanzar la moneda para lograr la victoria, lo que implica tareas como desplazarse por el laberinto, buscar y recolectar la moneda, y evitar al enemigo durante su desplazamiento. Por su parte, el enemigo realiza tareas destinadas a obstaculizar al jugador, como desplazarse por el laberinto y seguir al jugador para impedir que alcance su objetivo. Ambos actores comparten el caso de uso "Desplazarse", que abarca la lógica y las mecánicas de movimiento en el entorno del laberinto, que implican el uso de previo de los algoritmos de búsqueda que permiten encontrar el camino que conecta con sus objetivos. En la Figura 4.2, se presentan los casos de uso específicos para los actores Jugador y Enemigo.

A continuación se describen en detalle los casos de uso más importantes del sistema del videojuego, describiendo la secuencia de acciones que llevan al éxito del caso de uso, y las excepciones potenciales de cada uno. En el Anexo 8 se describen detalladamente el resto de CU.

El primer caso a describir, es el de *Crear Partida*, donde se podrá configurar el entorno de juego con sus diferentes modos. Al crear una partida, se configura el entorno estableciendo el nivel, la dificultad, el modo de juego si se desea jugar contra un rival o en solitario (Modo Enfrentamiento o Modo Solitario), el modo de interacción eligiendo el Modo Usuario o Modo Simulación, o el numero de juegos (1, 3 o 5). En el caso de elegir Modo Simulación o Modo Enfrentamiento se podrá establecer el algoritmo de búsqueda utilizado por el jugador y el usado por el enemigo, respectivamente. El nivel establece el tamaño del laberinto y la disposición de los personajes y la moneda, pudiendo elegir entre 3 niveles predefinidos y un nivel generado aleatoriamente. En la Tabla 4.1 se puede observar la descripción detallada de este CU.

Cuando se crea e inicia una nueva partida, se crea un juego con las configuraciones establecidas, llevando a cabo el CU *Nuevo Juego*. A su vez, cuando se ha establecido el numero de juegos mayor a 1, se crea un nuevo juego después de finalizar el anterior debido a la finalización del tiempo, el alcance de de la moneda por parte del jugador, o el alcance del enemigo al jugador antes de lograr su objetivo. En la Tabla 4.2 se puede observar la descripción detallada de este CU.

Otro caso de uso destacable es *Guardar Partida*. El usuario podrá llevar a cabo el guardado de la partida a lo largo del periodo en que esté iniciada. Además será posible guardar la partida al salir o cuando se requiera un cambio en la configuración del entorno. En caso de finalizar la partida, como se ha mencionado anteriormente, se guardara los resultados de la misma, pero solo se pueden recuperar para su posterior visualización. En la Tabla 4.3 se puede observar la descripción

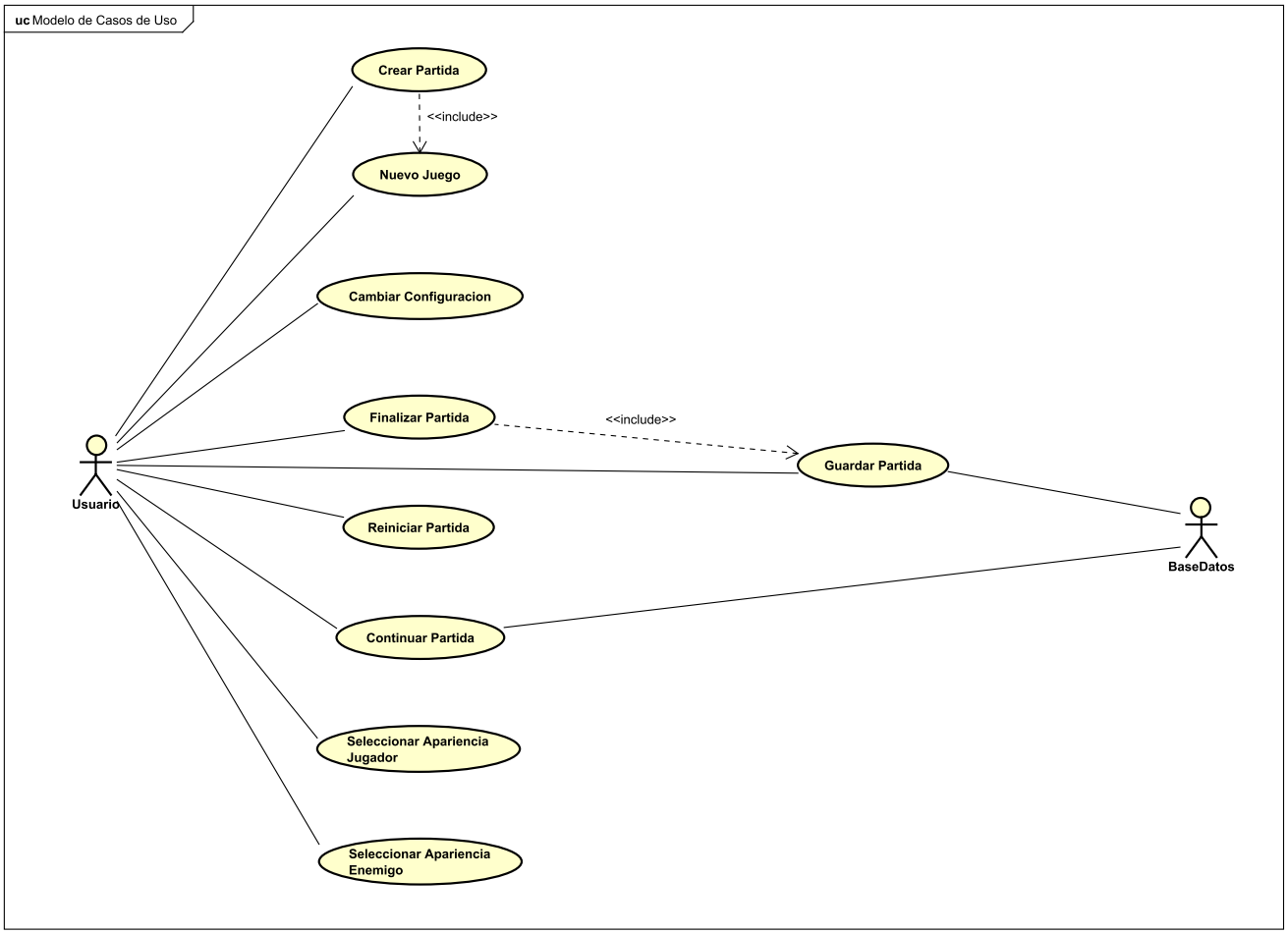


Figura 4.1: Casos de uso referentes actores que intervienen en el videojuego Usuario y BaseDatos.

detallada de este CU, donde se realiza la conexión con la base de datos para almacenar los datos relativos a la partida, que permitan recrear de nueva el entorno con la configuración establecida.

Las partidas guardadas cuya esta no se encuentra finalizado, permiten recuperar el progreso alcanzado anteriormente, y continuar el juego en el punto de partida almacenado. Este es el CU *Continuar Partida*, que consiste en listar las partidas guardadas que pertenecen al usuario que mantiene la sesión activada en la aplicación. Además, una partida se podrá continuar y guardar tantas veces como se desee. En la Tabla 4.4 se puede observar la descripción detallada de este CU.

Si la partida se encuentra en el modo de juego Modo Simulación, el jugador busca la moneda utilizando el algoritmo de búsqueda definido en la configuración, por lo que se lleva a cabo el CU *Recolectar Moneda*. Al buscar la moneda, crea un camino para recorrer posteriormente desplazándose mediante los movimientos permitidos (izquierda, derecha, arriba o abajo), en cambio, cuando el juego se encuentra en Modo Enfrentamiento, utiliza el mismo algoritmo para buscarlo y crear un camino mediante el cual desplazarse intentando evitar la trayectoria que sigue el enemigo. En la Tabla 4.5 se puede observar la descripción detallada de la secuencia que sigue este CU.

El jugador tiene como objetivo principal alcanzar y recolectar lo moneda mediante el CU *Buscar Moneda*, para ello debe obtener el camino o los caminos existentes hasta ella mediante el uso de un algoritmo de búsqueda establecido en la configuración de la partida. Estos caminos son utilizados posteriormente para desplazarse hasta la moneda de la forma más rápida y óptima,

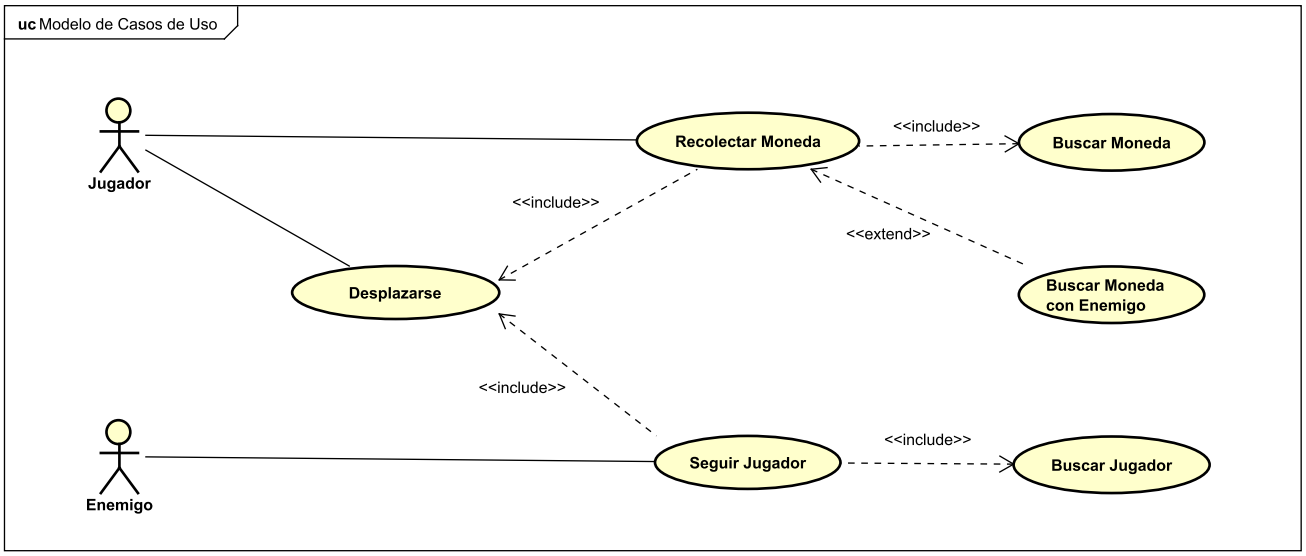


Figura 4.2: Casos de uso correspondientes a los actores que intervienen en el videojuego Jugador y Enemigo.

incluso evitando al enemigo mediante la combinación de los caminos obtenidos que conectan al jugador y la moneda, y las trayectorias existentes para evitar al enemigo. En la secuencia de este CU mostrada detalladamente en la Tabla 4.6 se puede observar como comprueba el algoritmo de búsqueda establecido, simulando el recorrido del laberinto desde la posición del jugador hasta la posición de la moneda, que le permite obtener y almacenar los caminos existentes estableciendo el orden recorrido por el algoritmo. En cambio, cuando hay un enemigo presente en el juego, el jugador debe modificar su estrategia para evitar un enfrentamiento, cuyo proceso se basa en la planificación de un camino seguro hacia la moneda, esquivando la posición del enemigo. Para lograrlo, el jugador calcula los posibles caminos desde su posición hasta la moneda mediante el uso de los algoritmos de búsqueda establecidos y evalúa cuáles de ellos no atraviesan con la posición del enemigo. Entre estas rutas, se selecciona la más óptima según el criterio del algoritmo empleado, como la menor distancia o menor costo, permitiendo que el jugador se desplace evitando al enemigo mientras cumple su objetivo principal.

Tanto el jugador como el enemigo pueden realizar desplazamientos que se realizan de forma paralela, a través del CU *Desplazarse*. En Modo Simulación, el jugador realiza el movimiento tras realizar la búsqueda de la moneda y usando el camino recreado mediante el algoritmo de búsqueda, en cambio, en Modo Usuario, el desplazamiento se realizará en función de los movimientos indicados por el usuario mediante las entradas de teclado indicando la dirección del movimiento (arriba, abajo, derecha o izquierda). En Modo Enfrentamiento, el enemigo se desplazará hacia una de las direcciones posibles tras la búsqueda del jugador, recorriendo el caminos creado. En la Tabla 4.7 se puede observar la descripción detallada de la secuencia que sigue este CU.

## 4.4. Modelo de dominio

El modelo de dominio es la representación visual de la estructura que forman los conceptos del problema a resolver, analizados los casos de uso y los requisitos funcionales y no funcionales,

| **Caso de Uso** | Crear Partida |
|---|---|
| Descripción | El usuario podrá crear la partida creando un nuevo juego y configurando el nivel, la dificultad, el numero de juegos, el modo de enfrentamiento, el modo de interacción y en su defecto el algoritmo de búsqueda utilizado para resolver el laberinto. |
| **Actores** | Usuario |
| **Escenario Principal** | 1. El Usuario solicita crear una nueva partida.<br>2. El Sistema solicita el nivel, dificultad, numero de juegos, modo de juego y modo de interacción de la partida.<br>3. El Usuario indica el nivel, dificultad, numero de juegos, modo de juego y modo de interacción de la partida.<br>4. El Sistema comprueba que se ha indicado el Modo Simulación o el Modo Enfrentamiento.<br>5. El Sistema solicita el algoritmo de búsqueda deseado.<br>6. El Usuario indica el algoritmo de búsqueda.<br>7. El Sistema solicita confirmación de los datos de configuración.<br>8. El Usuario confirma los datos de configuración de la partida.<br>9. El Sistema almacena la configuración de la partida.<br>10. Se realiza el caso de uso Nuevo Juego.<br>11. Finaliza el caso de uso Crear Partida. |
| **Flujos alternativos** | 3.a. 6.a. 8.a. El usuario solicita finalizar el caso de uso.<br>- 1. El sistema cancela la configuración de la partida.<br>- 2. Finaliza el caso de uso Configurar Entorno.<br>4.a El Sistema comprueba que no se ha indicado el Modo Simulación ni el Modo Enfrentamiento.<br>- 1. Se continua en el paso 13. El Sistema solicita confirmación |

Tabla 4.1: Descripción de la secuencia de acciones del CU Crear Partida.

se lleva a cabo el diseño del modelo de dominio plasmando en diferentes entidades los conceptos y objetos que resuelven los problemas planteados en los casos de uso. En la Figura 4.3 se puede observar el modelo de dominio completo definido para este sistema, sin incluir las operaciones propias de cada entidad, y ajustado a los tipos de datos propios del lenguaje de programación *GDScript* utilizado para el desarrollo e implementación del dominio de este proyecto. Los tipos de datos propios de este lenguaje de desarrollo de videojuegos utilizado que determinan el tipo de variables de cada entidad que compone el dominio están descritos en la Tabla 4.8, donde se detalla cada tipo de dato utilizado y su descripción especifica. En el Anexo 8 se encuentra el diagrama del modelo de dominio ampliado, con las operaciones específicas de cada entidad.

El modelo de dominio esta compuesto por una clase principal *Partida*, relacionada con la entidad que establece la lógica del videojuego llamada *Videogame*, en cambio la entidad Partida, hace referencia a la secuencia de interacciones con el *Laberinto* que realiza el *Usuario* y se ve reflejado en los diferentes objetos de los que se compone. Además, hay dos tipos de *Personaje*, el *Jugador* y el *Enemigo*, con características comunes descritas como clases, por ejemplo *Camino*, que contiene la trayectoria desde cada personaje hasta su objetivo, o *Algoritmo de Búsqueda*, permitiendo la búsqueda de los respectivos objetivos de cada uno. Por otro lado, la partida cuenta con otro objeto llamado *Moneda*. Cada partida se compone de uno o más *Juegos*, que hacen referencia a cada una de las partes en las que se divide una partida. El modelo de dominio esta

| **Caso de Uso** | Nuevo Juego |
|---|---|
| **Descripción** | El usuario inicia un nuevo juego en la partida. |
| **Actores** | Usuario |
| **Pre-condición** | El usuario ha configurado un entorno de juego. |
| **Escenario Principal** | 1. El Usuario solicita iniciar un nuevo juego.<br>2. El Sistema muestra la configuración del entorno.<br>3. El Sistema inicia un juego nuevo con la configuración almacenada.<br>4. Finaliza el caso de uso Nuevo Juego. |

Tabla 4.2: Descripción de la secuencia de acciones del CU Nuevo Juego.

| **Caso de Uso** | Guardar Partida |
|---|---|
| **Descripción** | El usuario guarda la partida almacenando el progreso obtenido y las configuraciones de la partida establecidas.. |
| **Actores** | Usuario, BaseDatos |
| **Pre-condición** | El usuario ha iniciado una partida. |
| **Post-condición** | Se podrá continuar la partida con el progreso guardado. |
| **Escenario Principal** | 1. El usuario solicita el guardado de la partida.<br>2. El sistema muestra el progreso de la partida.<br>3. El sistema solicita confirmación del guardado de la partida.<br>4. El usuario confirma el guardado de la partida.<br>5. El sistema comprueba que si la partida estaba guardada.<br>6. Base de Datos almacena el progreso de la partida.<br>7. Finaliza el caso de uso Guardar Partida. |
| **Flujos alternativos** | 4.a. El usuario solicita finalizar el caso de uso.<br>- 1. El sistema cancela el guardado de la partida.<br>- 2. Finaliza el caso de uso Guardar Partida.<br>5.a. El sistema comprueba que la partida no estaba guardada.<br>- 1. El sistema almacena la partida sobrescribiendo los datos.<br>- 2. Finaliza el caso de uso Guardar Partida. |

Tabla 4.3: Descripción de la secuencia de acciones del CU Guardar Partida.

compuesto por las entidades descritas, a continuación se detalla cada una de ellas:

- **Partida:** Es la entidad principal, comprendida por la puntuación o el nombre de la partida, es decir, almacena la información relativa a la partida.
- **Videogame:** Compuesta por los datos de configuración de la partida, y encargada de conectar con la API que lleva a cabo la conexión con la base de datos.
- **Laberinto:** Contiene tanto las posiciones de los personajes y elementos del laberinto, como su tamaño y tiempo máximo de resolución.
- **Usuario:** Son los datos del usuario identificado en el sistema, mediante email y contraseña.
- **Personaje:** Es una generalización de los tipos de personajes que componen el juego, donde se diferencian dos tipos descritos a continuación. Se compone de los atributos propios de las entidades «CharacterBody2D» como la posición, la velocidad, o la aceleración, y también de

| **Caso de Uso** | Continuar Partida |
|---|---|
| **Descripción** | El Usuario retomar una partida guardada anteriormente con las configuraciones establecidas y el progreso alcanzado. |
| **Actores** | Usuario, BaseDatos |
| **Pre-condición** | El usuario tiene al menos una partida guardada. |
| **Post-condición** | La partida se puede guardar posteriormente. |
| **Escenario Principal** | 1. El usuario solicita continuar una partida.<br>2. El sistema comprueba que existen partidas guardadas.<br>3. El sistema muestra la lista de partidas guardadas.<br>4. El sistema solicita la partida a continuar.<br>5. El usuario indica la partida que desea continuar.<br>6. El sistema muestra la configuración y el progreso de la partida.<br>7. El sistema inicia y continua la partida con el progreso almacenado.<br>8. Finaliza el caso de uso Continuar Partida. |
| **Flujos alternativos** | 2.a. El sistema comprueba que no hay partidas guardadas.<br>- 1. El sistema muestra que no hay partidas guardadas.<br>- 2. Finaliza el caso de uso Continuar Partida.<br>5.a. El usuario solicita finalizar el caso de uso.<br>- 1. El sistema cancela la continuación de la partida.<br>- 2. Finaliza el caso de uso Continuar Partida. |

Tabla 4.4: Descripción de la secuencia de acciones del CU Continuar Partida.

otros atributos definidos referentes al manejo de la posición o el movimiento, y la apariencia del personaje que permite visualizar al personaje con diferentes interfaces.

- **Jugador:** Es un tipo de personaje que puede realizar movimientos en el laberinto mediante entradas de teclado o a través de los caminos obtenidos por su algoritmo de búsqueda, hereda los atributos de la entidad Personaje y se compone de una señal que emite cuando finaliza el movimiento.

- **Enemigo:** Es un tipo de personaje que puede realizar movimientos en el laberinto unicamente a través de los caminos obtenidos por su algoritmo de búsqueda, hereda los atributos de la entidad Personaje y se compone de una señal que emite cuando finaliza el movimiento.

- **Camino:** Comprende el array con el camino obtenido para cada personaje hasta su objetivo, ademas de los atributos referentes al nodo inicial y al nodo final de dicha trayectoria.

- **Algoritmo de Búsqueda:** Es un atributo propio de cada personaje, que permite la búsqueda y obtención del camino entre el personaje y su objetivo, en el caso del jugador busca la moneda y evita al enemigo para permitir resolver el laberinto, y en el caso del enemigo,busca al jugador para evitar que este alcance la moneda. Se compone de grafo que determina la composición del laberinto, la heuristica de cada nodo, y las trayectorias de cada personaje.

- **Moneda:** Es un elemento del laberinto con una posición fija, objetivo del jugador para resolver y obtener la victoria en el juego. Se compone de una señal emitida al ser alcanzada

| **Caso de Uso** | Recolectar Moneda |
|---|---|
| **Descripción** | El jugador recolecta la moneda buscando y creando una trayectoria mediante los algoritmos de búsqueda, a su vez, evita al enemigo y avanzando hacia la moneda mediante desplazamientos. |
| **Actores** | Jugador |
| **Pre-condición** | Se ha iniciado una partida y se encuentra en Modo Simulación. |
| **Escenario Principal** | 1. El jugador solicita alcanzar la moneda.<br>2. El sistema comprueba que la partida está en Modo Solitario.<br>3. Se realiza el caso de uso Buscar Moneda.<br>4. El sistema comprueba que existe un camino solución calculado para desplazarse hasta la moneda.<br>5. El sistema obtiene la siguiente posición a la que se desplazará el enemigo.<br>6. Se realiza el caso de uso Desplazarse.<br>7. El sistema comprueba si el jugador ha alcanzado la moneda.<br>8. El sistema indica que la partida ha finalizado victoriosa al haber alcanzado la moneda y actualiza la puntuación.<br>9. Finaliza el caso de uso Recolectar Moneda. |
| **Flujos alternativos** | 2.a. El sistema comprueba que la partida esta en Modo Enfrentamiento.<br>- 1. Se realiza el caso de uso Buscar Moneda con Enemigo.<br>- 2. Se continua en el paso 4. El sistema comprueba que existe...<br>4.a. El sistema comprueba que no existe un camino resultado para recolectar la moneda.<br>- 1. El sistema muestra que no existe camino para recolectar la moneda.<br>- 2. El sistema finaliza la partida.<br>- 3. Finaliza el caso de uso Recolectar Moneda.<br>7.a. El sistema comprueba que el jugador no ha alcanzado la moneda.<br>- 1. El sistema muestra que no ha alcanzado la moneda.<br>- 2. Se continua en el paso 2. El sistema comprueba que la partida... |

Tabla 4.5: Descripción de la secuencia de acciones del CU Recolectar Moneda.

por el jugador.

- **Juegos:** Determina el número de ejecuciones o juegos que comprende una partida, sucedidos tras la finalización del juego anterior y finalizados cuando el jugador alcanza la moneda o es alcanzado por el enemigo. El número de juegos posibles en una partida se compone de tres opciones diferentes (al mejor de uno, de tres o de cinco) .

Los tipos de datos establecidos como «ennumeration» determinan los estados de la partida y del juego, y los modos y configuraciones de la partida. Antes de iniciar la partida, se debe realizar la configuración de las diversas opciones que determinaran el flujo del juego. Esta configuración se refleja en los atributos de la clase partida. Ante una ampliación del número de opciones de configuración se plantea la existencia de una nueva entidad que refleje todas ellas. Las opciones iniciales son las siguientes:

- **Modo de juego:** Indica el modo de juego en el que se diferencia entre Modo Solitario y Modo Enfrentamiento, para elegir juego con o sin enemigo.

| **Caso de Uso** | Buscar Moneda |
|---|---|
| **Descripción** | El jugador realiza la búsqueda de la moneda mediante el uso del algoritmo definido anteriormente. |
| **Actores** | Jugador |
| **Escenario Principal** | 1. El jugador solicita realizar la búsqueda de la moneda.<br>2. El sistema comprueba el algoritmo de búsqueda definido.<br>3. El sistema busca la moneda aplicando dicho algoritmo.<br>4. El sistema almacena el camino obtenido para alcanzar la moneda.<br>5. Finaliza el caso de uso Buscar Moneda. |

Tabla 4.6: Descripción de la secuencia de acciones del CU Buscar Moneda.

| **Caso de Uso** | Desplazarse |
|---|---|
| **Descripción** | El jugador/enemigo realiza el desplazamiento en el laberinto en base a los cuatro movimientos disponibles. |
| **Actores** | Jugador, Enemigo |
| **Pre-condición** | Se ha iniciado una partida. |
| **Post-condición** | La posición del jugador/enemigo se modifica a una casilla adyacente en el laberinto. |
| **Escenario Principal** | 1. El actor solicita desplazarse indicando la siguiente posición del laberinto.<br>2. El sistema comprueba que la dirección del movimiento se realiza hacia una posición válida y sin colisión.<br>3. El sistema actualiza la posición del actor.<br>4. El sistema muestra la nueva posición del actor.<br>5. Finaliza el caso de uso Desplazarse. |
| **Flujos alternativos** | 2.a El sistema comprueba que la dirección de movimiento no es válida o tiene colisión.<br>- 1. El sistema muestra que no la dirección de movimiento no es válida y mantiene la posición actual.<br>- 2. Finaliza el caso de uso Desplazarse. |

Tabla 4.7: Descripción de la secuencia de acciones del CU Desplazarse.

- **Modo de interacción:** Indica el modo de interacción con el juego, pudiendo elegir entre Modo Usuario y Modo Simulación, en el modo usuario, es el usuario el que indica los movimientos a realizar por el jugador, y en el Modo Simulación esta controlado mediante la computadora.
- **Nivel:** En el nivel se diferencian tres niveles preestablecidos en tamaño y posiciones iniciales y uno aleatorio, que determinan el tamaño del laberinto.
- **Algoritmo:** Es el algoritmo de búsqueda utilizado tanto por el jugador como por el enemigo.
- **Dificultad:** La cercanía inicial entre el jugador y el enemigo, o la distancia a la moneda.
- **Juegos:** Indica el número de juegos o secciones en las que se divide la partida.
- **Estado Partida:** La partida dispone de 4 estados (Iniciada, en curso, guardada o finalizada).
- **Estado Juego:** El juego, puede encontrarse en tres estados (En curso, finalizado o en pausa).

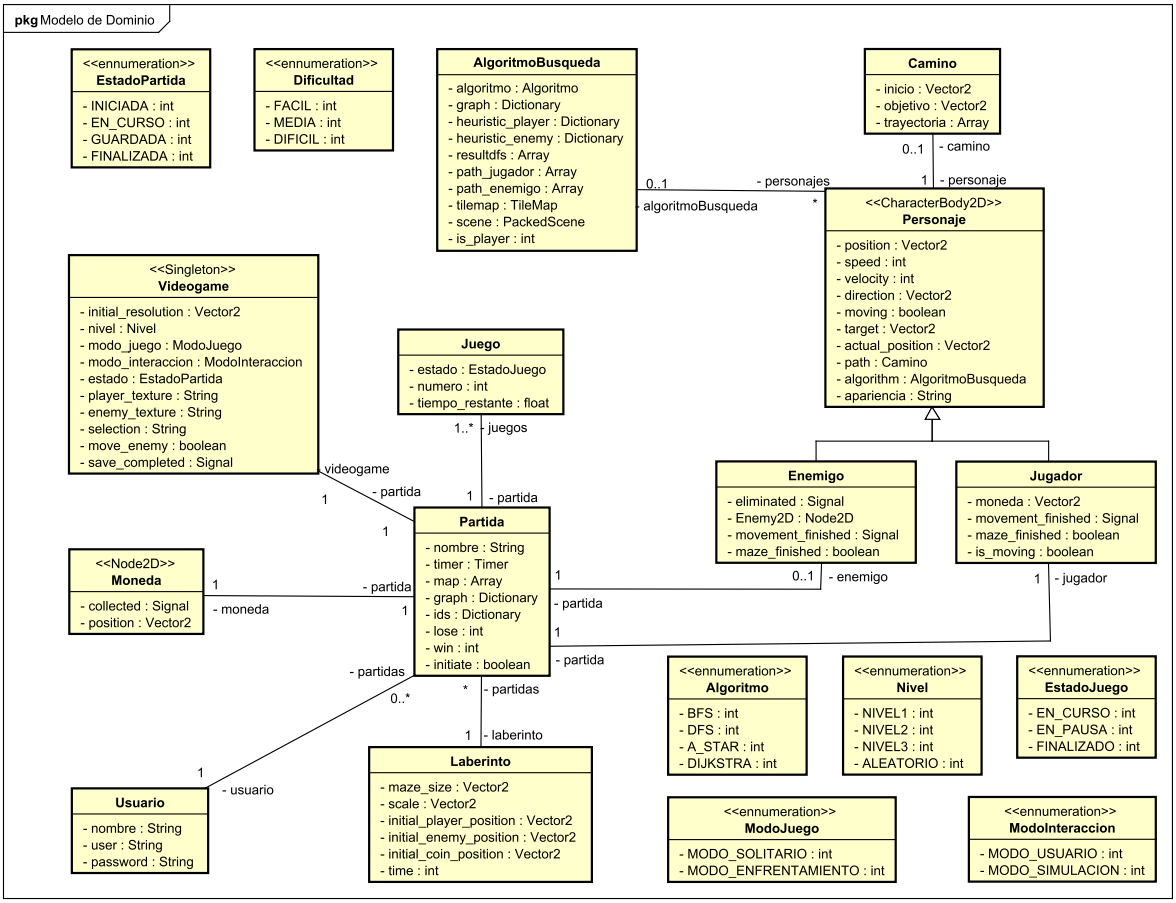


Figura 4.3: Diagrama del modelo de dominio simplificado donde se incluyen las entidades y atributos del sistema.

| Tipo de dato | Descripción |
|---|---|
| String | Cadenas de caracteres |
| int | Valores numéricos de tipo entero |
| float | Valores numéricos de tipo flotante o decimal |
| Array | Conjunto o secuencia de datos de diferente tipo |
| Dictionary | Conjunto de datos de diferente tipo compuestos de pares clave-valor |
| boolean | Valores verdadero o falso únicamente |
| Vector2 | Vector de coordenadas en 2 dimensiones y valores de tipo flotante |
| Signal | Representa una señal emitida por un objeto |
| TileMap | Nodo para mapas basados en mosaicos 2D. |
| PackedScene | Una abstracción de una escena serializada. |

Tabla 4.8: Descripción de los tipos de datos utilizados en el proyecto por los atributos que componen las entidades del modelo de dominio.

# Capítulo 5

# Diseño

Este capítulo se limita a la definición de la arquitectura, componentes, los detalles técnicos y la forma de interaccionar entre los actores y sistema con el fin de resolver los casos de uso planteados en el capítulo 4. En la fase de diseño se definen las estructuras de datos y su comportamiento teniendo en cuenta las limitaciones establecidas por los requisitos del sistema. Así mismo, define las instrucciones que se debe llevar a cabo en la fase de desarrollo e implementación.

## 5.1. Modelo dinámico

El modelo dinámico está constituido a su vez por tres elementos, el modelo de interacción, las máquinas de estados y la vista de actividades, que representan el comportamiento del sistema a través del tiempo y cómo se comunican los elementos que lo componen. En las siguientes secciones se describen en detalle cada uno de estos elementos de los que se compone el modelo dinámico, especificando los casos de uso principales del sistema a los que hacen referencia.

### 5.1.1. Modelo de interacción

El modelo de interacción describe cómo colaboran entre sí grupos de objetos para conseguir un fin, y se pueden expresar en forma de diagramas de secuencia o diagramas de comunicación. Para el desarrollo de este proyecto, nos centraremos en los diagramas de secuencia. Se detallan los más relevantes, los cuales son los referentes a la dinámica de juego y el control de menús accesibles antes de crear la partida y durante el transcurso de la misma. El sistema se comunica repetidamente con la clase partida, que es la clase principal del dominio y encargada de comunicarse con todos los demás objetos del sistema.

En el menú principal disponemos de diversas opciones que corresponden a los diversos casos de uso descritos en el sistema. Todas ellas llevadas a cabo por el actor Usuario y entre ellas encontramos Continuar Partida, Seleccionar Apariencias o Crear Partida. A continuación se describen los diagramas de secuencia referentes a los casos de uso *Crear Partida*, *Nuevo Juego*,

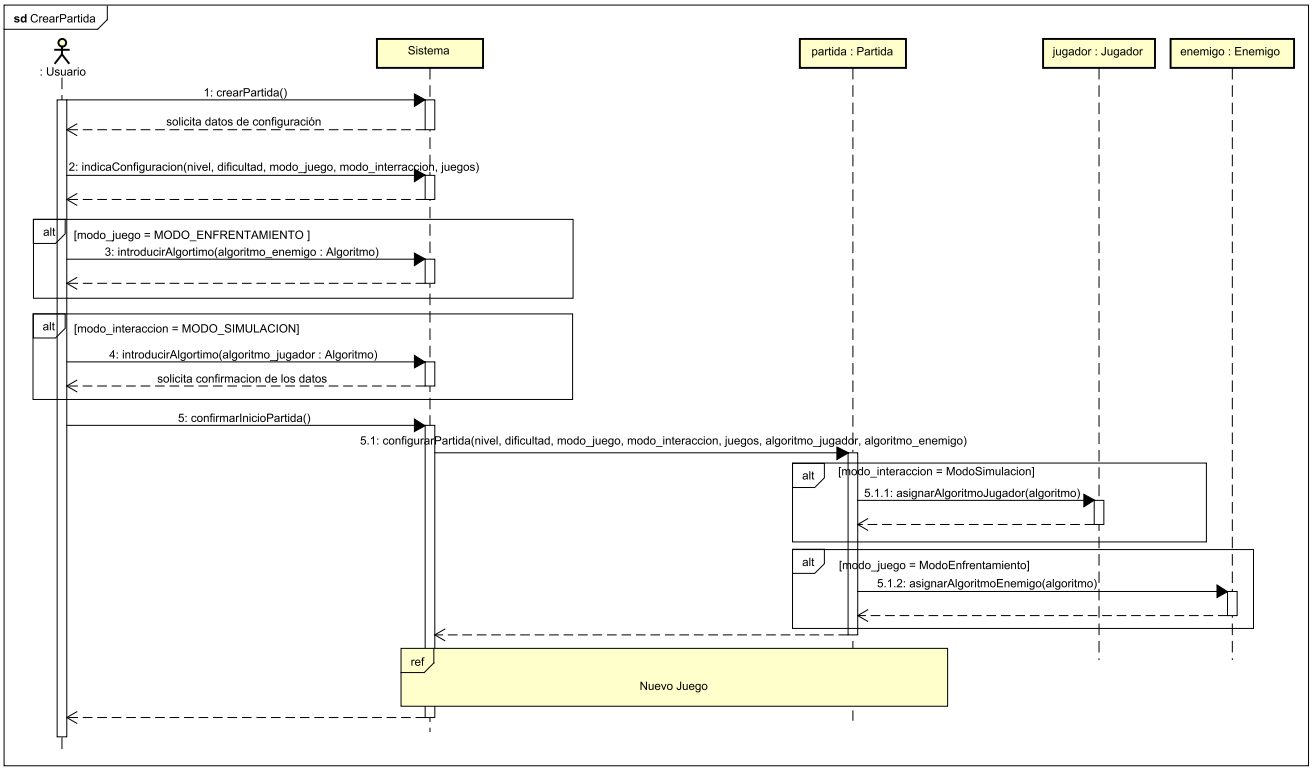


Figura 5.1: Diagrama de secuencia Crear Partida.

*Guardar Partida*, *Recolectar Moneda*, *Buscar Moneda* y *Desplazarse*. En el Anexo 9 se describen detalladamente el resto de diagramas de secuencia referentes a los casos de uso no descritos a continuación.

El caso de uso Crear Partida, define el proceso mediante el cual se configura el entorno del videojuego, especificando características como el tamaño, la dificultad y los modos del laberinto. La secuencia de acciones comienza con la selección del nivel de laberinto, seguida por la elección del modo de juego y del modo de interacción, y en caso de que el usuario seleccione el Modo Enfrentamiento o el Modo Simulación, se incluye una subsecuencia adicional para elegir el algoritmo de búsqueda correspondiente. Durante este proceso, el sistema registra las entradas proporcionadas por el usuario y, al finalizar la configuración, establece la partida en estado «INICIADA». Esto da inicio automáticamente a un nuevo juego, activando el caso de uso asociado para esta secuencia, y marcando el estado de la partida como «EN CURSO». A partir de este punto, el sistema permite el desarrollo de acciones relacionadas con la resolución del laberinto, estableciendo el estado del juego como «EN CURSO». La Figura 5.1 ilustra esta secuencia mediante un diagrama de secuencia que detalla las interacciones y pasos definidos para el caso de uso Crear Partida.

Una vez iniciada la partida, se genera un nuevo juego, lo que permite la ejecución de acciones propias del desarrollo del laberinto, como el desplazamiento tanto del jugador como del enemigo (si aplica), así como el uso de algoritmos configurados previamente. Cada partida puede contener uno o varios juegos, según lo establecido en su configuración (uno, tres o cinco juegos), donde al finalizar cada uno, el sistema crea automáticamente el siguiente hasta alcanzar el número indicado. Por esta razón, la secuencia de creación de un juego se detalla de forma independiente del proceso inicial de creación de la partida, y se detalla en la Figura 5.2, donde se describe cómo el sistema

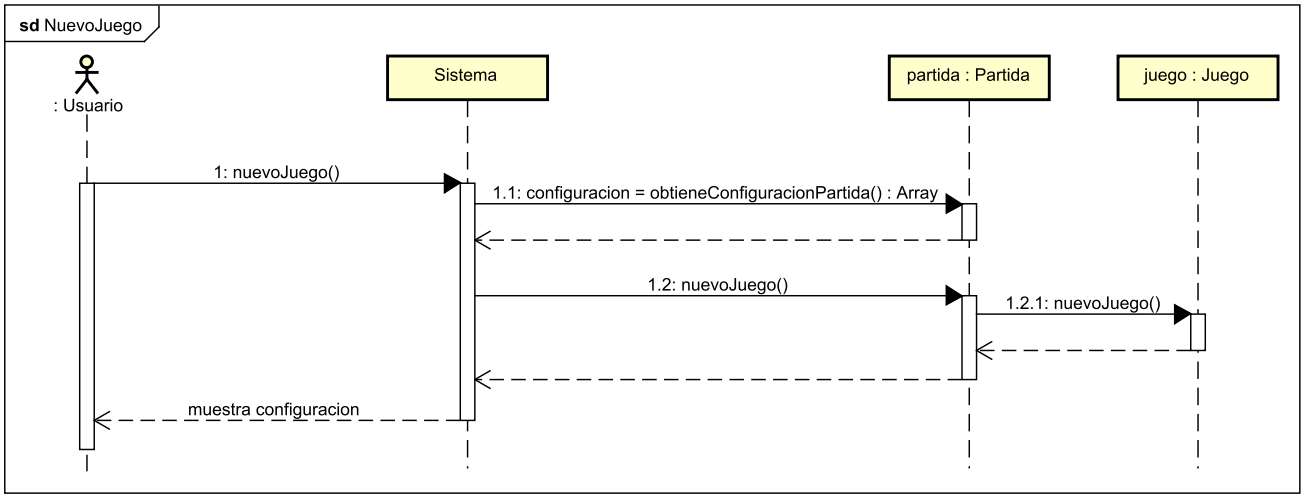


Figura 5.2: Diagrama de secuencia Nuevo Juego.

obtiene los parámetros de configuración, solicita la confirmación del usuario para crear un nuevo juego, y lo añade a la lista de juegos de la partida en curso.

El caso de uso Guardar Partida permite al usuario registrar el progreso de la partida en la base de datos repetidamente durante el transcurso de la partida hasta alcanzar el final. Esta acción puede realizarse de manera autónoma o ser requerida por otros casos de uso, como Finalizar Partida, donde el guardado incluye información relevante como puntuación, estadísticas (victorias y derrotas) y configuraciones del entorno que se ha establecido al inicio de la partida. El progreso almacenado permite continuar la partida desde el último punto guardado, siempre y cuando esta no haya sido finalizada, en caso contrario, solo podrán recuperarse las estadísticas y visualizarlas. La Figura 5.3 muestra cómo el sistema recopila el progreso, conecta con la base de datos y almacena la información, y si ya existía un progreso anterior, este es sobrescrito por los datos más recientes.

El desarrollo de juego está representado mediante dos casos de uso con secuencias muy similares, aplicables tanto al jugador como al enemigo. El jugador realiza acciones como recolectar la moneda y evitar al enemigo durante este proceso, mientras que el enemigo tiene como objetivo seguir al jugador y obstaculizar su progreso impidiendo que alcance su objetivo. Ambas acciones comparten una estructura común: se inicia obteniendo la posición del objetivo correspondiente, seguida por la búsqueda del mismo y la creación de trayectorias posibles utilizando casos de uso específicos de búsqueda, una vez identificadas estas trayectorias, se verifica si existe al menos una ruta válida para continuar y, en caso afirmativo, el personaje procede a desplazarse hacia la siguiente posición. La Figura 5.4 muestra la secuencia detallada para la recolección de la moneda por parte del jugador, mientras que la Figura 9.8 ilustra el proceso de seguimiento del enemigo al jugador.

Cada caso de uso asociado a la búsqueda de un objetivo incluye la generación de trayectorias posibles desde la posición actual del actor hasta su destino. Estos procesos se implementan a través de los casos de uso Buscar Jugador y Buscar Moneda, aunque las secuencias son similares, difieren en la posición inicial, el objetivo, y el actor que emplea las trayectorias calculadas. La separación en dos casos de uso responde a estas diferencias, donde el procedimiento comienza seleccionando el algoritmo de búsqueda correspondiente, que calcula las rutas posibles hacia el objetivo, creando así el camino asignado al personaje. En la Figura 5.5 se presenta el proceso relativo a la búsqueda de la moneda, mientras que la Figura 9.7 detalla el proceso de búsqueda del jugador.

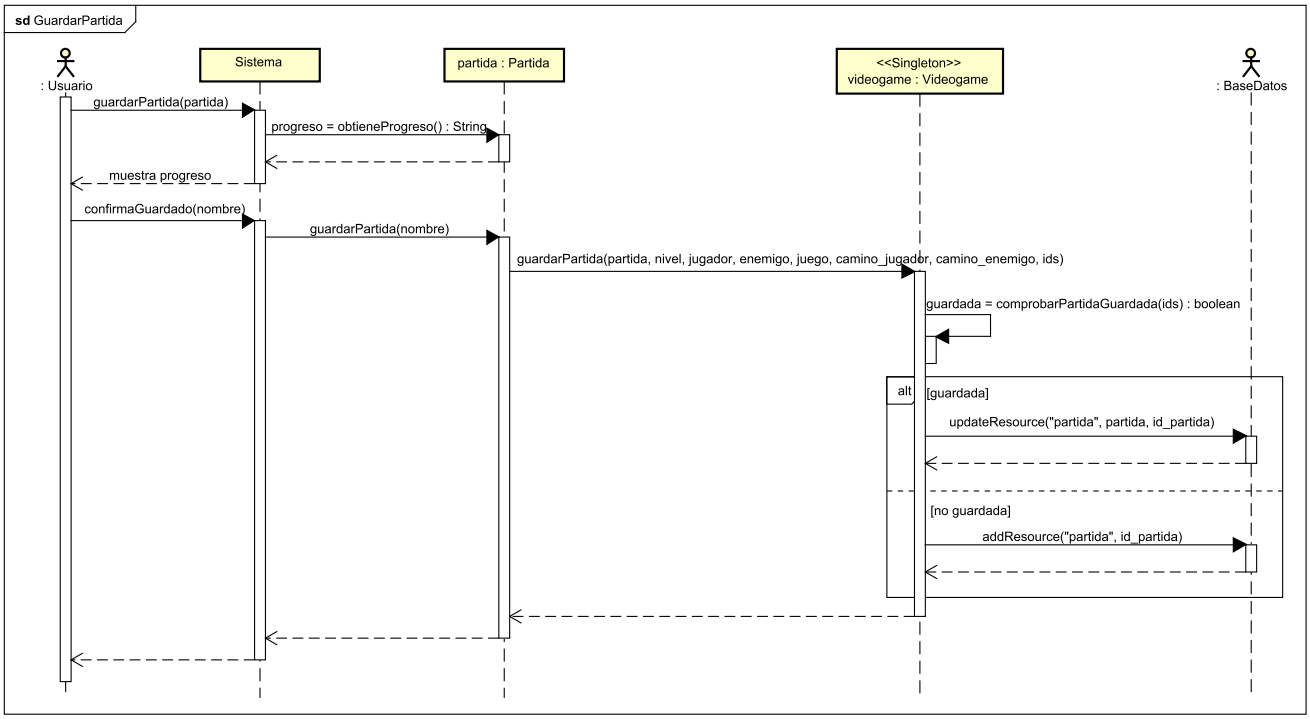


Figura 5.3: Diagrama de secuencia Guardar Partida.

Con las trayectorias calculadas, ambos personajes se mueven a la siguiente posición determinada por estas rutas. Este proceso incluye una verificación para asegurar que el movimiento indicado es válido, tras lo cual se actualiza la posición del personaje, cuyas secuencias varían según el tipo de personaje: la Figura 5.6 describe el desplazamiento del jugador, en cambio, que la Figura 9.9 ilustra el movimiento del enemigo, donde la diferencia radica en el personaje al que se va a actualizar la posición.

### 5.1.2. Maquinas de estados

En esta sección se van a describir las secuencias de estados que sufren los objetos que interactúan con los casos de uso *Crear Partida*, *Nuevo Juego*, *Continuar Partida*, *Recolectar Moneda* y *Buscar Moneda*.

Debido a la inclusión del caso de uso Nuevo Juego dentro del caso Crear Partida, ambos diagramas de estados se describen de manera simultánea. El proceso de creación de la partida comienza con la introducción de datos de configuración preestablecidos, como se muestra en la Figura 5.7, donde se observa cómo los estados iniciales representan la espera de cada uno de los datos proporcionados por el usuario para configurar el entorno. Una vez completada esta etapa, se crea la partida y se asignan los algoritmos correspondientes al jugador y al enemigo según los modos seleccionados. A continuación, se inicia el proceso de creación del juego, descrito en la Figura 5.8, que transita desde el estado «solicitado» a «creado», y este proceso abarca los estados intermedios de «en configuración» y «en creación» propios del caso de uso Crear Juego.

El caso de uso Continuar Partida, representado en la Figura 5.9, detalla las transiciones de estados para una partida guardada que el usuario desea reanudar. El flujo comienza con la recuperación de la partida desde la base de datos y su posterior selección, una vez confirmada,

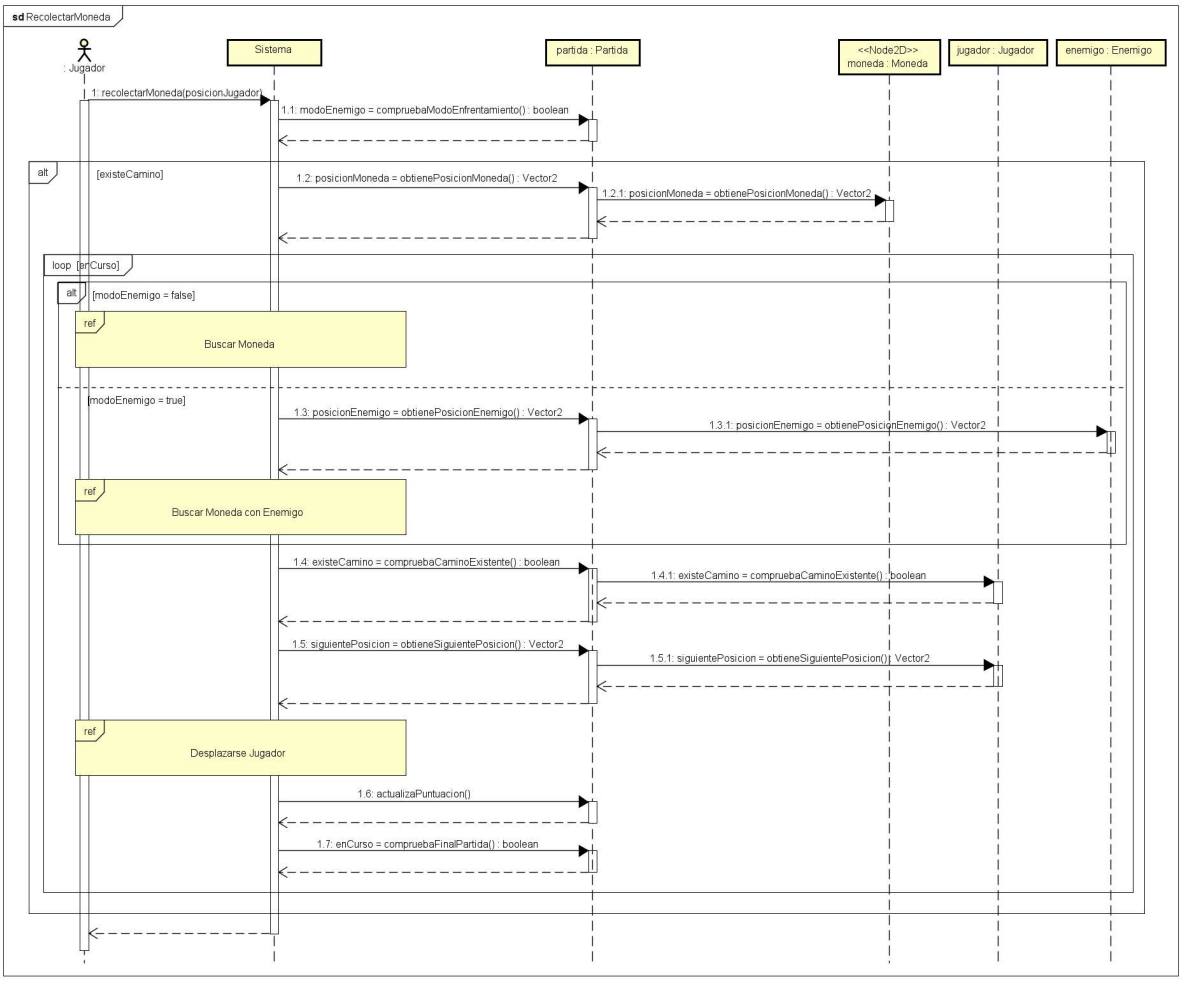


Figura 5.4: Diagrama de secuencia Recolectar Moneda.

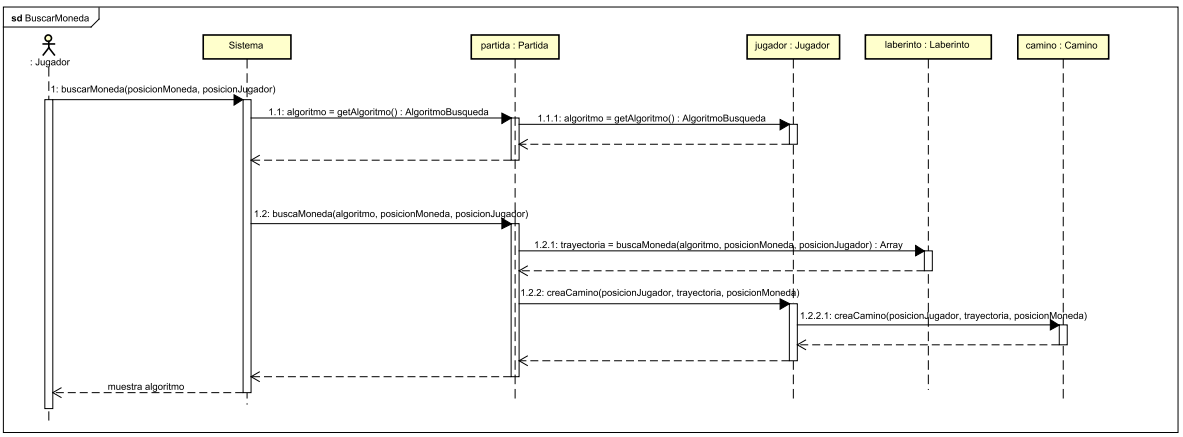


Figura 5.5: Diagrama de secuencia Buscar Moneda.

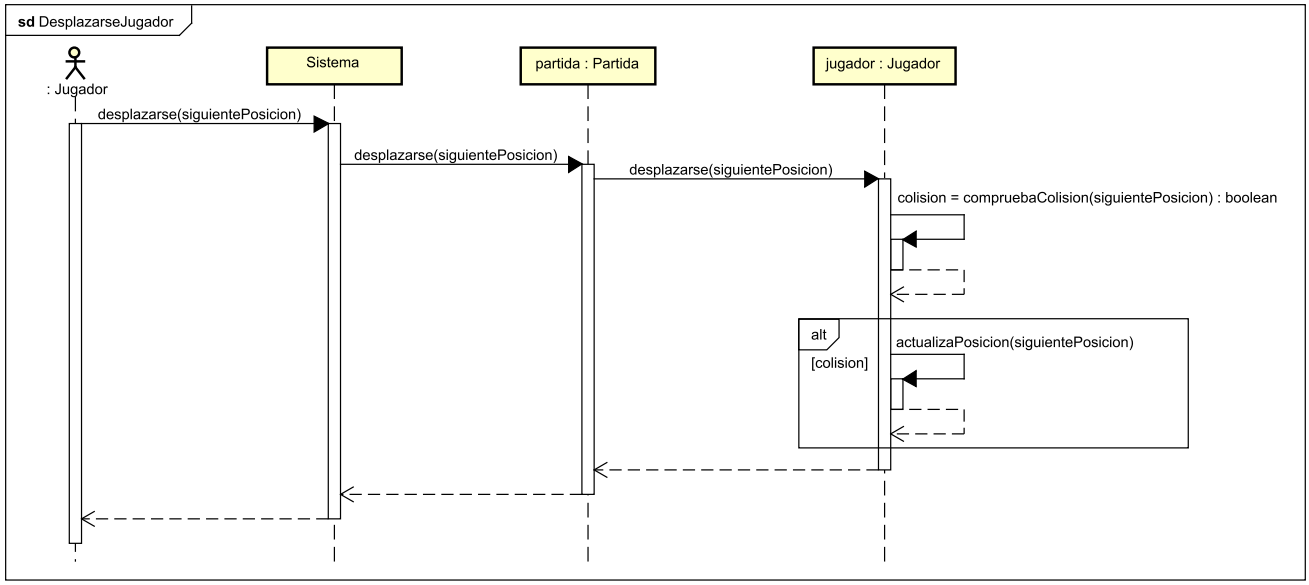


Figura 5.6: Diagrama de secuencia Desplazarse Jugador.

la partida se establece en el estado «En curso», permitiendo que el jugador continúe el progreso previamente guardado.

En relación con las acciones del jugador, las máquinas de estados asociadas a los casos de uso Recolectar Moneda y Buscar Moneda se describen en la Figura 5.10 y en la Figura 5.11, respectivamente. Estas muestran las transiciones de los estados de la moneda y del propio juego, desde «en curso» hasta «finalizado», dependiendo de si el jugador alcanza la moneda o es interceptado por el enemigo. Además, se detallan las actualizaciones y cálculos de la posición del jugador durante el desplazamiento. La moneda transita del estado «en búsqueda» a «encontrada» como resultado del caso de uso Buscar Moneda, que incluye la creación del camino con las trayectorias que la conectan con el jugador.

### 5.1.3. Diagramas de actividad

Siguiendo la estructura descrita en el apartado anterior, se analizan los casos de uso Crear Partida, Nuevo Juego, Continuar Partida, Recolectar Moneda y Buscar Moneda. Los diagramas de estados presentan una estructura similar a los diagramas de actividad, lo que implica que la secuencia y composición de cada uno guardan muchas similitudes. Por ello, en las explicaciones siguientes se destacan los estados por los que transitan los diferentes objetos involucrados en las secuencias de los casos de uso mencionados.

En los diagramas mostrados en la Figura 5.12 y en la Figura 5.13 se observa cómo se establecen las configuraciones y el algoritmo en función de los modos seleccionados. Cuando el nodo de decisión identifica el modo como «enfrentamiento» o «simulación», se configura la partida. Una vez creada, su estado cambia a «Iniciada». Posteriormente, al generar el nuevo juego, tanto la partida como el juego adoptan el estado «en curso».

Para continuar una partida, es necesario conectarse a la base de datos y recuperar las partidas guardadas asociadas al usuario. El objeto denominado Partidas, representado en la Figura 5.14,

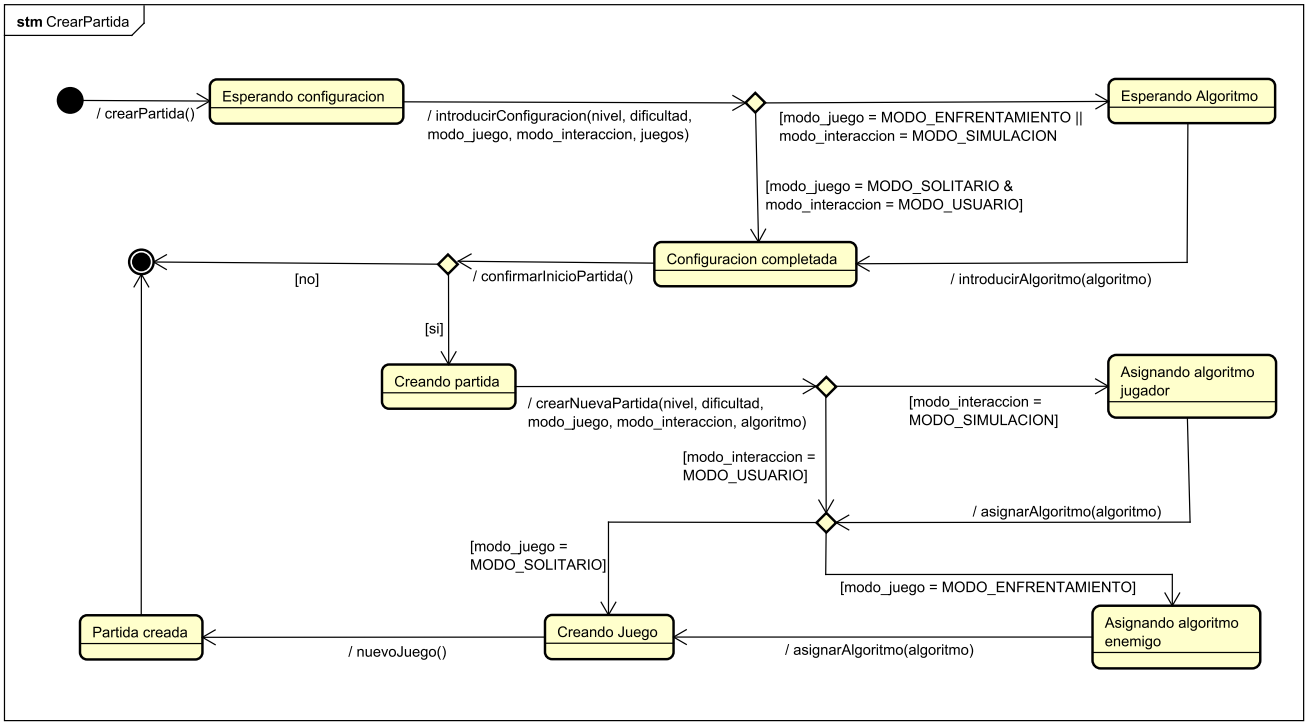


Figura 5.7: Diagrama de estados referente a Crear Partida.

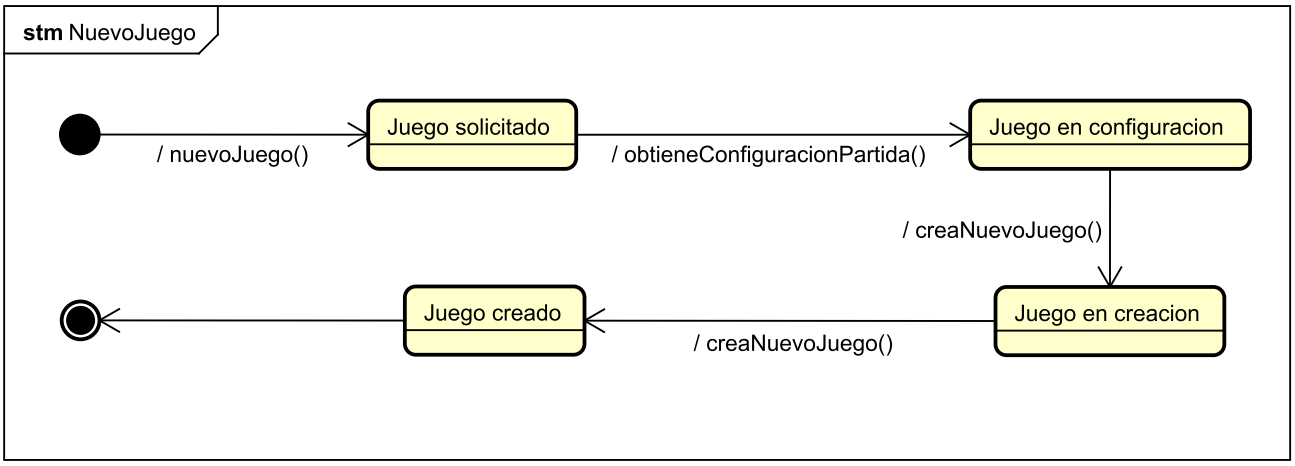


Figura 5.8: Diagrama de estados referente a Nuevo Juego.

corresponde a un objeto de tipo «datastore» que contiene todas las partidas almacenadas en la base de datos, sin importar el usuario. Al mostrarse la lista de partidas, el usuario selecciona una en estado «guardada», que transiciona a «en curso» tras confirmar su continuación.

En cuanto a la combinación de los diagramas relacionados con la moneda, se describe la transición de su estado desde «búsqueda pendiente» a «en búsqueda», y finalmente a «encontrada», una vez completado el caso de uso Buscar Moneda. Durante este proceso, el camino se establece como «creado» al generar las trayectorias que conectan al jugador con la moneda. Posteriormente, se realizan las verificaciones de finalización del juego: este concluye con el estado «finalizado» si el jugador alcanza la moneda o si es interceptado por el enemigo.

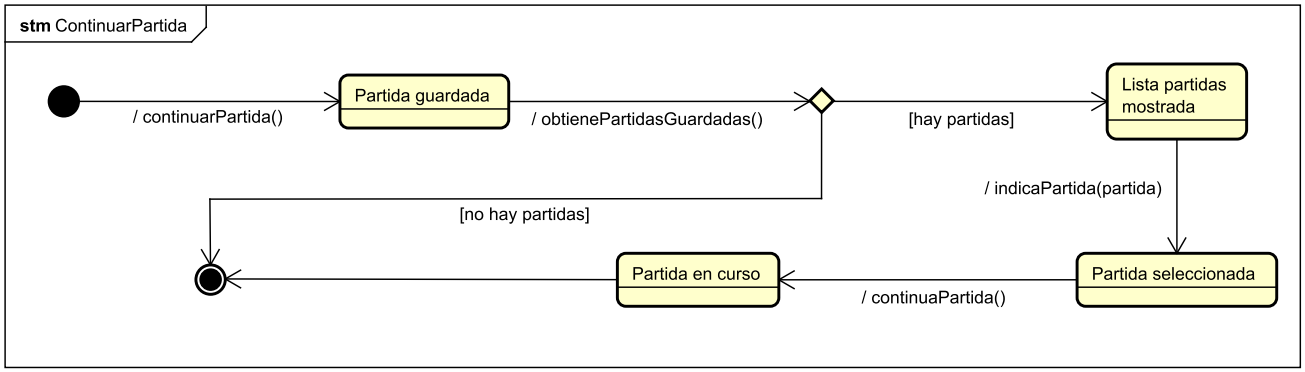


Figura 5.9: Diagrama de estados referente a Continuar Partida.

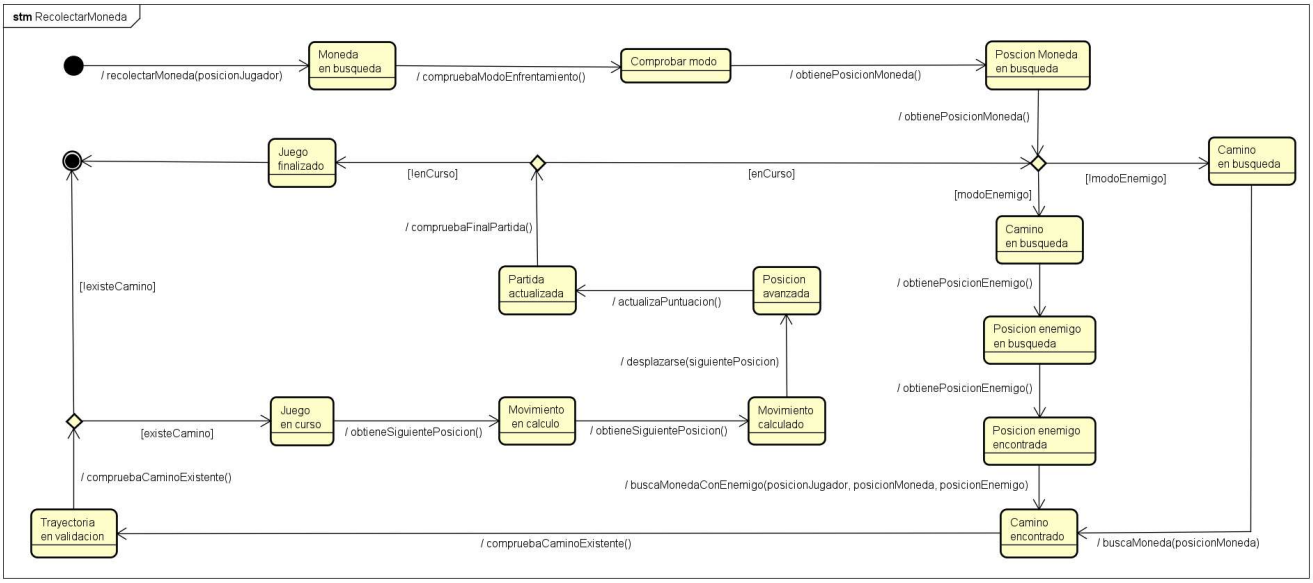


Figura 5.10: Diagrama de estados referente a Recolectar Moneda.

## 5.2. Modelo de Implementación

### 5.2.1. Diagrama de componentes

La Figura 5.17 ilustra un diagrama simplificado de los componentes del proyecto. El videojuego cuenta con dos interfaces distintas, dependiendo del entorno de ejecución. Si el juego se ejecuta a través de un navegador web, utiliza la interfaz proporcionada por el componente «HTML5». Por otro lado, cuando el videojuego se ejecuta en su versión de escritorio, emplea la interfaz proporcionada por el componente «.exe». Otro componente destacado es la «Base de Datos», que se encarga de ofrecer la interfaz necesaria para gestionar y administrar los datos de persistencia. Este componente es esencial para el manejo adecuado del almacenamiento y recuperación de información en el videojuego.

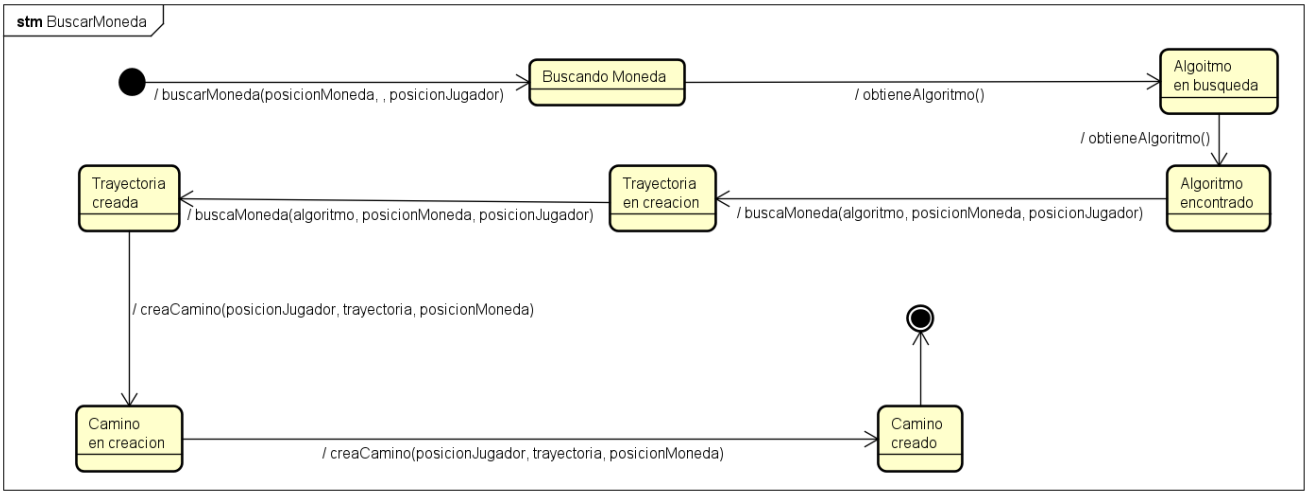


Figura 5.11: Diagrama de estados referente a Buscar Moneda.

### 5.2.2. Diagrama de despliegue

En la Figura 5.18 se presenta el diagrama de despliegue del sistema desarrollado, el cual está compuesto por dos nodos principales que representan los elementos clave del entorno de ejecución del proyecto:

- **Dispositivo:** Este nodo simboliza el entorno donde los usuarios interactúan directamente con la aplicación, que se subdivide en dos entornos de ejecución. El *Navegador Web*, representa la ejecución del videojuego en navegadores como *Chrome*, *Firefox*, o similares, este entorno utiliza el componente *HTML5*, que actúa como la tecnología base para la exportación del proyecto a una versión web, el cual proporciona compatibilidad multiplataforma y acceso sin necesidad de instalación adicional, facilitando la experiencia del usuario. *Aplicación de Escritorio* o *Desktop*, en este caso, el videojuego se ejecuta localmente en formato de aplicación, como un archivo ejecutable, un entorno está representado por el componente *.exe*, que hace referencia a la extensión comúnmente utilizada para archivos ejecutables en sistemas operativos como *Windows*, donde la ejecución en este formato permite un rendimiento optimizado y un control total sobre los recursos del sistema.

- **Base de Datos:** Este nodo es crucial para garantizar la persistencia de la información. Está diseñado para almacenar y gestionar datos como las partidas guardadas y la autentificación de usuarios, entre otros. El componente central de este nodo es SQLite, un sistema gestor de bases de datos ligero y eficiente, ideal para aplicaciones locales y de tamaño reducido. SQLite permite manejar el almacenamiento y la recuperación de datos de manera rápida y confiable, proporcionando una solución robusta para las necesidades del proyecto.

Esta arquitectura de despliegue permite una implementación flexible y eficiente del sistema, asegurando una experiencia de usuario consistente tanto en entornos web como de escritorio.

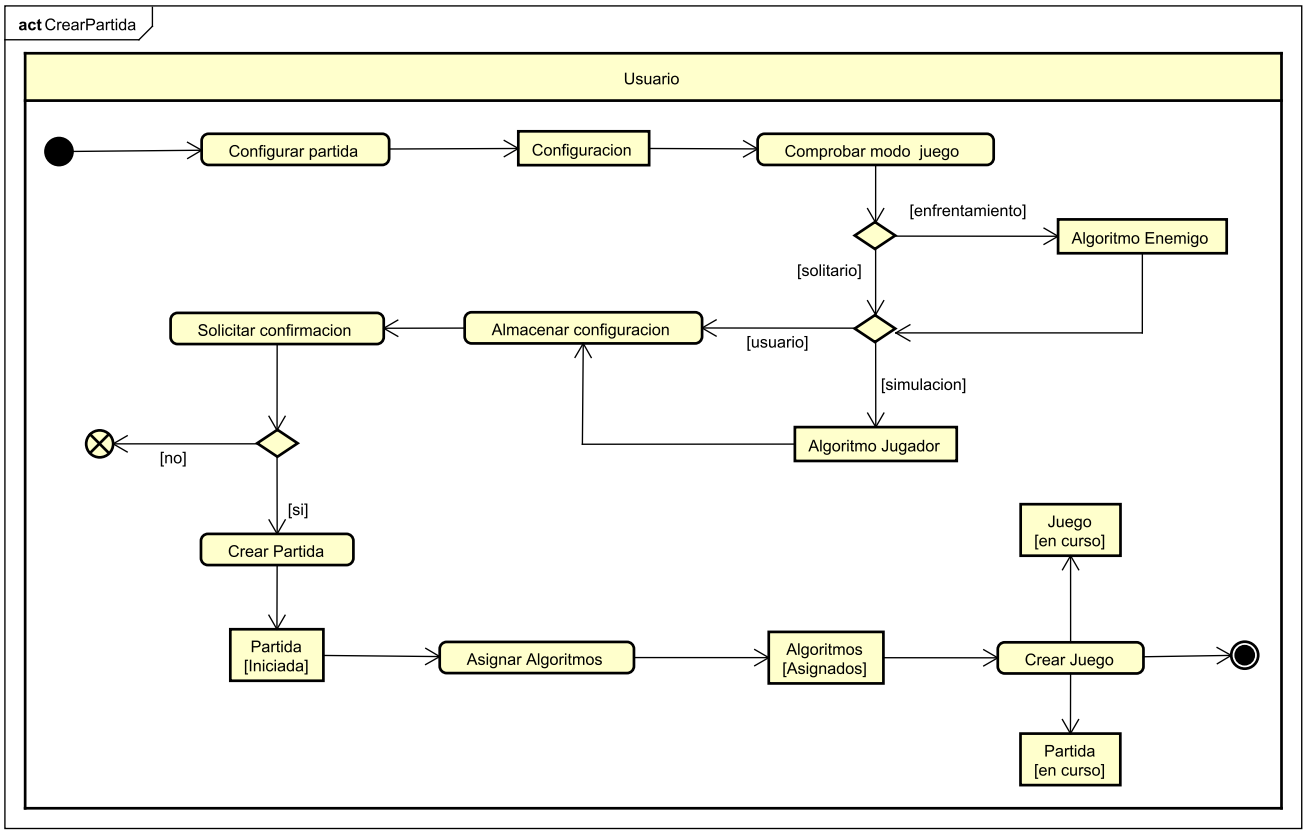


Figura 5.12: Diagrama de actividad referente a Crear Partida.

## 5.3. Arquitectura

En este proyecto, se ha utilizado el patrón Modelo-Vista-Controlador (MVC) como muestra la Figura 5.19, para organizar la estructura del videojuego, compuesta por 3 capas estrictas y 1 relajada. Este enfoque divide las responsabilidades en tres componentes principales: el Modelo, que gestiona los datos y la lógica del sistema; la Vista, que se encarga de la representación visual y la interacción con el usuario; y el Controlador, que actúa como intermediario, procesando las entradas y actualizando la Vista en función del Modelo.

### 5.3.1. Patrones de diseño

Un patrón arquitectónico muestra el esquema organizacional de un sistema software describiendo un problema de diseño particular mediante sus componentes, relaciones y formas en las que interactúan. Existen diferentes patrones de diseño, de los que destacan:

- **Cliente-Servidor:** basado en la existencia de un servidor y uno o mas clientes. Utilizado en servicios web y online.
- **Por capas:** divide la aplicación en capas que representan una tarea y un nivel de abstracción diferente.
- **Modelo-Vista-Controlador (MVC):** divide la aplicación en tres capas, separando la lógica, del modelo de datos y de la vista, y es muy utilizado en aplicaciones web y en

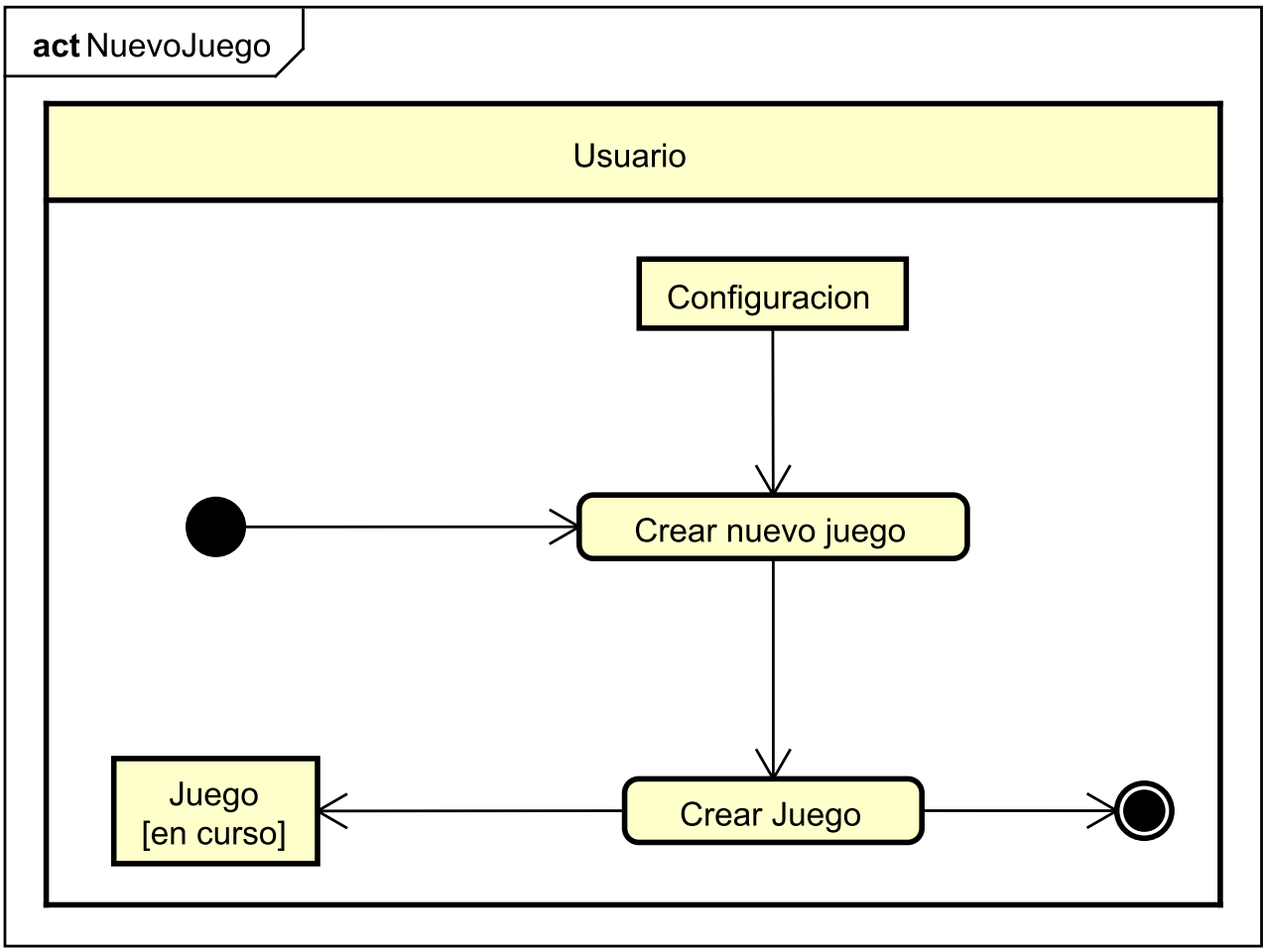


Figura 5.13: Diagrama de actividad referente a Nuevo Juego.

programación orientada a objeto.

- **Entidad-Componente Sistema (ECS):** esta creado específicamente para el desarrollo de videojuegos y divide la aplicación también en tres partes separando datos de lógica.

Aunque, ECS es el patrón específico creado para el desarrollo de videojuegos como es el caso de este proyecto, por lo que sería el más adecuado para aplicar a este problema, se ha llevado a cabo la implementación del videojuego mediante la aplicación del patrón arquitectónico MVC. Esto se debe a que *Godot Engine*, el motor de videojuegos utilizado en este proyecto no está diseñado ni basado específicamente[1] para el desarrollo de videojuegos mediante el patrón arquitectónico ECS. En cambio, este motor de videojuegos, utiliza programación orientada a objetos proporcionando nodos que contienen tanto la lógica como los datos, como el uso de composición y herencia, ya que realiza composiciones a un nivel superior al de un ECS tradicional. Aunque existen implementaciones externas al propio *Godot Engine*[2], que permiten adaptar la lógica de este motor al patrón ECS, el trabajo requerido para llevar a cabo esta implementación es altamente denso y extenso, lo que provocaría una carga de trabajo en tiempo y recursos excesiva en consecuencia con los límites establecidos para este proyecto. También, en este motor, la aplicación de ECS es recomendada para proyectos que requieran procesar la lógica del juego miles objetos, como en el caso de este

[1] https://godotengine.org/article/why-isnt-godot-ecs-based-game-engine/
[2] https://github.com/GodotECS/godex

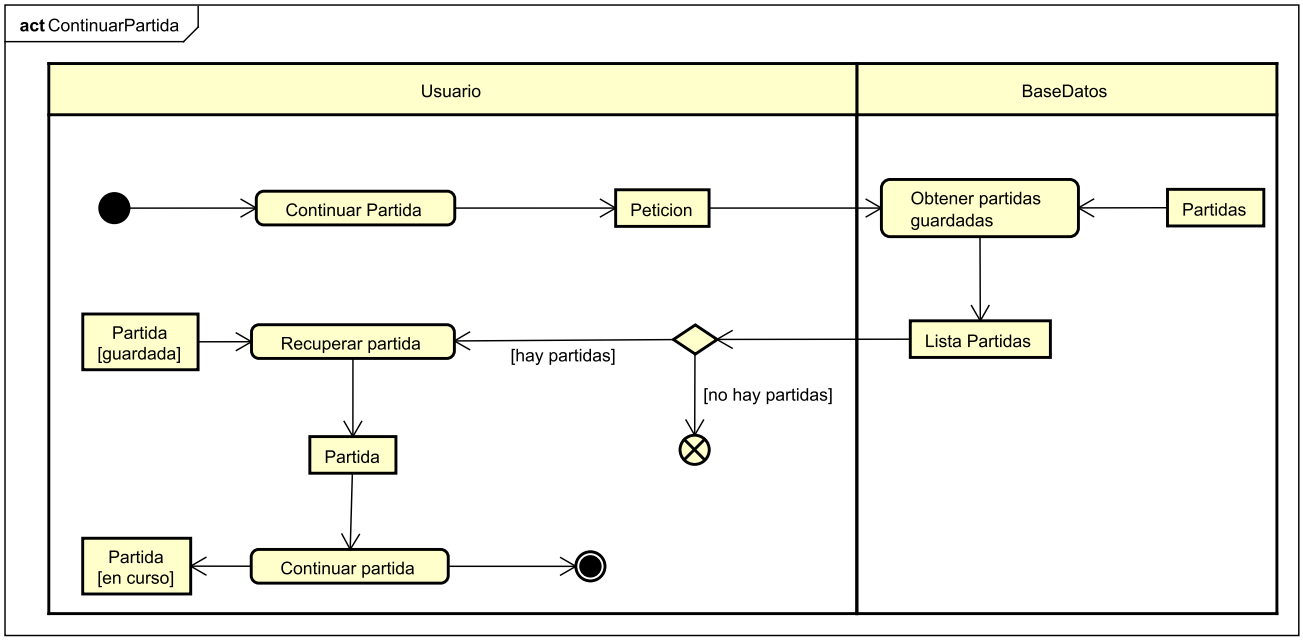


Figura 5.14: Diagrama de actividad referente a Continuar Partida.

proyecto la complejidad de este sistema no es muy alta ni se precisa del tratamiento de la lógica de un número excesivo de objetos, también ha servido como característica para la elección de MVC frente a ECS. Otro de los criterios que han decantado esta elección es la amplia familiaridad con MVC que presenta el estudiante encargado de desarrollar este proyecto. Por lo que, gracias a todos estos criterios descritos, se ha implementado el patrón arquitectónico MVC para el desarrollo de este proyecto, permitiendo separar la lógica, datos y la vista, y generando una estructuración sencilla.

### 5.3.2. Modelo-Vista-Controlador (MVC)

El patrón MVC es una arquitectura de software que se compone de tres capas fundamentales: el Modelo, la Vista y el Controlador. Este patrón organiza los componentes de la aplicación de manera que cada capa tiene una responsabilidad bien definida, lo que permite una estructura más clara, mantenible y escalable. El principio básico de MVC es el desacoplamiento, donde cada capa se enfoca en un aspecto específico de la aplicación, lo que minimiza el acoplamiento entre ellas y mejora la organización general del código. En este enfoque, el Modelo es responsable de gestionar los datos, la lógica de negocio y el comportamiento del sistema. La Vista se encarga de la presentación de la información al usuario, y el Controlador actúa como intermediario entre el Modelo y la Vista, interpretando las entradas del usuario y actualizando el Modelo o la Vista según sea necesario. Además, el Modelo no tiene conocimiento directo de la Vista, lo que garantiza que los componentes de la lógica del juego y la interfaz de usuario estén desacoplados, lo que facilita la evolución del sistema sin afectar directamente a cada capa.

En el contexto del videojuego de laberinto desarrollado en este proyecto, el uso de este patrón ha sido clave para permitir un diseño modular, flexible y mantenible. Aunque el proyecto no tiene una complejidad tan alta como otros juegos, la elección de MVC sigue siendo adecuada porque permite que el desarrollo se mantenga ordenado y facilite futuras modificaciones o ampliaciones.

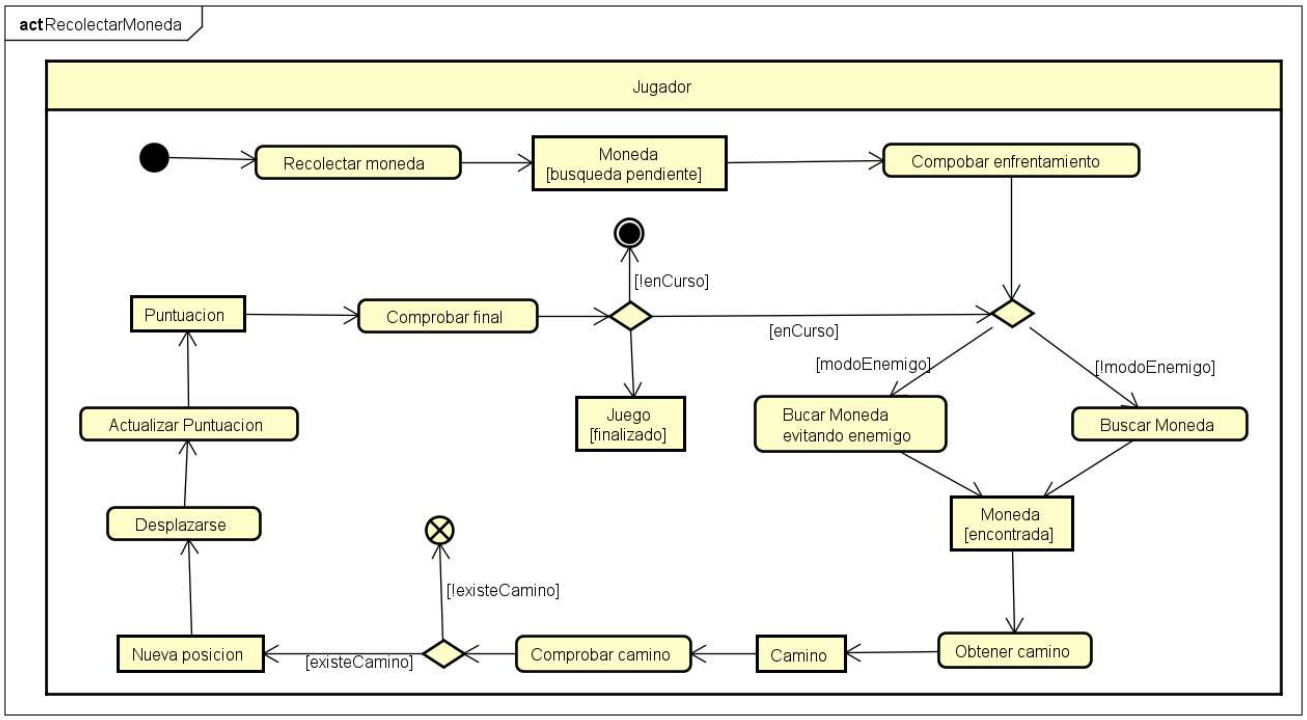


Figura 5.15: Diagrama de actividad referente a Recolectar Moneda.

Para mejorar aún más la estructura, se han implementado tres capas estrictas según el patrón MVC clásico, y se ha añadido una capa adicional para la Persistencia, que se considera una capa relajada debido a su naturaleza flexible, un capa adicional que se encarga de la gestión de la base de datos y la persistencia de los datos del usuario y las partidas guardadas. La estructura determinada de aplicación del patrón MVC en este proyecto es la siguiente:

- **Modelo:** El modelo es el componente central del videojuego, donde se gestionan los datos fundamentales como el estado del laberinto, las posiciones del jugador y los enemigos, los algoritmos de búsqueda (por ejemplo, para encontrar el camino hacia la moneda), y otros elementos del dominio del juego. El Modelo también contiene la lógica que define cómo deben interactuar estos elementos y cómo se comporta el laberinto y, además, el gestiona la actualización del estado del juego, que luego es reflejado en la Vista.

- **Vista:** La Vista se encarga exclusivamente de presentar los datos del Modelo de una manera que el usuario pueda comprender. En este videojuego, la Vista incluye la visualización del laberinto, las posiciones del jugador y los enemigos, y los menús de configuración, además de las interfaces de navegación. Esta capa no tiene ninguna lógica relacionada con la funcionalidad del juego, solo recibe los datos del Modelo y los presenta de forma adecuada. Esto permite que la interfaz gráfica sea completamente independiente de la lógica interna del juego, lo que facilita cambios en la interfaz sin modificar la lógica subyacente.

- **Controlador:** El Controlador actúa como el intermediario entre el Modelo y la Vista, recibiendo las entradas del usuario como las teclas de movimiento del jugador o la configuración del juego y las traduce en comandos que afectan al Modelo. El Controlador se asegura de que las acciones del jugador como mover al jugador o actualizar el estado del juego se transmitan correctamente al Modelo. Además, cuando el Modelo cambia, por

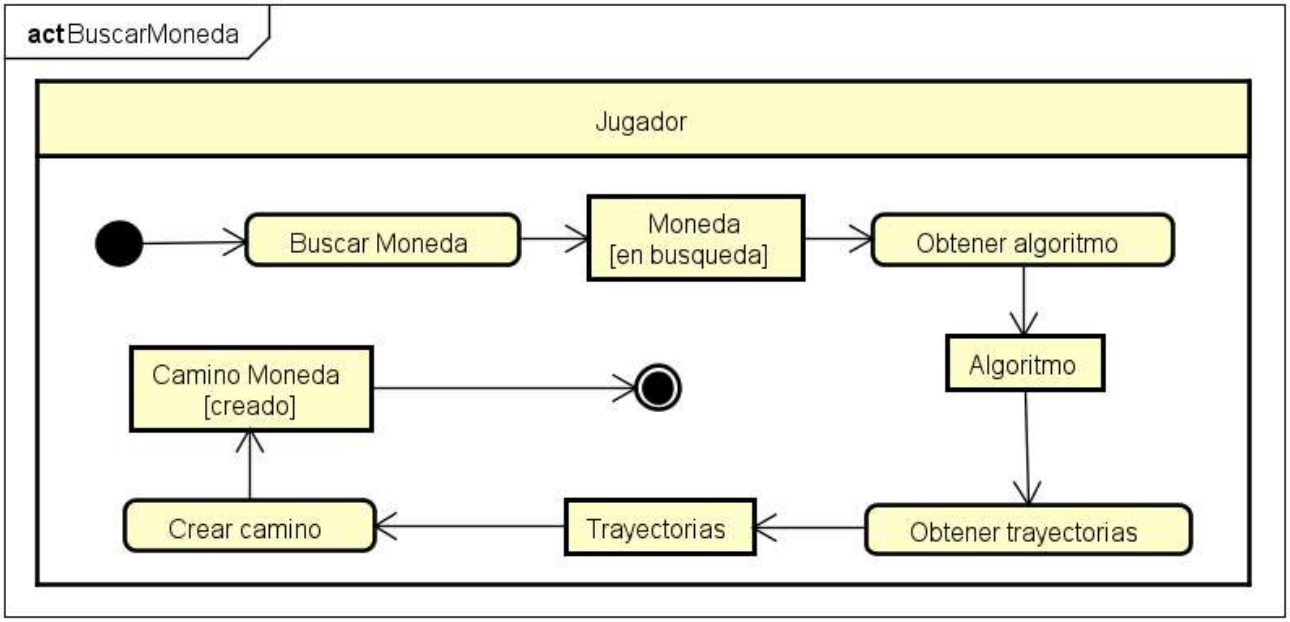


Figura 5.16: Diagrama de actividad referente a Buscar Moneda.

ejemplo, después de que el jugador se mueva o el estado del juego se actualice, el Controlador también es responsable de actualizar la Vista para reflejar esos cambios.

- **Persistencia:** La capa de Persistencia es la encargada de gestionar el almacenamiento y la recuperación de datos relacionados con el progreso del juego, como las partidas guardadas y los datos de usuario. Esta capa es relajada porque el Modelo puede interactuar directamente con ella sin necesidad de pasar por el Controlador, lo que permite una mayor flexibilidad y eficiencia. La Persistencia incluye la integración con la base de datos, utilizando SQLite para guardar los estados de las partidas y la autenticación de los usuarios, lo que garantiza que cada jugador tenga acceso solo a sus propias partidas guardadas.

### 5.3.3. Singleton

El patrón de diseño *Singleton*[34], descrito en la literatura de patrones de diseño como un patrón de instancia única, limita la creación de una clase a una única instancia, asegurando un control centralizado de recursos compartidos, como bases de datos o configuraciones globales. Este patrón permite un acceso global a la instancia única mediante un método estático, lo que asegura que dicha instancia no pueda ser sobrescrita por otra parte del programa, por lo que, para evitar que se creen nuevas instancias, el constructor de la clase se declara como privado o protegido. Sin embargo, el patrón *Singleton* tiene una desventaja importante, pues viola el Principio de Responsabilidad Única, ya que la clase tiene tanto la responsabilidad de gestionar su propia instancia como de cumplir su propósito principal, lo que puede dificultar la prueba y la extensión de la clase, especialmente en aplicaciones complejas.

En este proyecto, el patrón *Singleton* se ha aplicado principalmente para gestionar la comunicación con la base de datos, a través de una única instancia de la clase *Singleton*, se establece la conexión con la Interfaz de Programación de Aplicaciones (API) de la base de datos diseñada

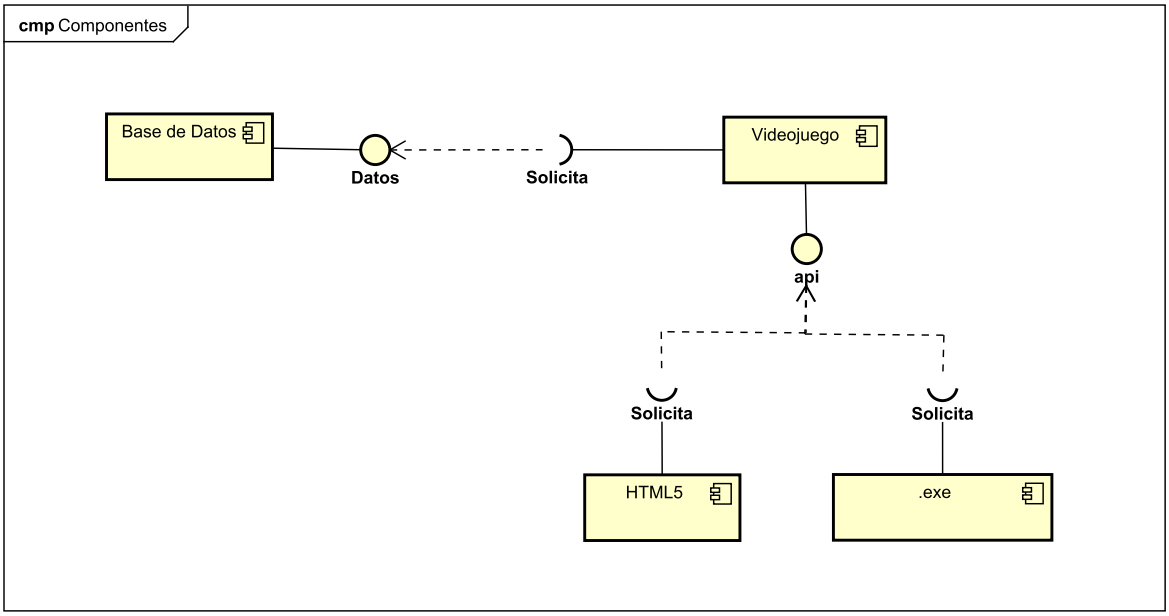


Figura 5.17: Diagrama de componentes.

para el juego, centralizando procesos como el inicio de sesión, el registro de usuarios, y la gestión de partidas guardadas y cargadas. Este enfoque garantiza que no haya múltiples instancias intentando acceder simultáneamente a los mismos recursos, reduciendo errores y mejorando la eficiencia. Además, el *Singleton* se emplea en este proyecto para mantener una única fuente de verdad sobre la configuración de la partida, donde la instancia almacena variables globales relacionadas con la apariencia seleccionada para los personajes y las configuraciones específicas de cada partida como pueden ser el modo de juego, el nivel o los algoritmos de búsqueda, permitiendo que sean accesibles desde cualquier punto del programa sin necesidad de múltiples referencias o instancias adicionales.

Gracias a la aplicación de este patrón, se obtienen ciertas ventajas como la centralización de recursos, la facilidad de acceso o eficiencia y reducción en la creación de múltiples instancias. Aunque el uso de este patrón puede suponer la sobrecarga con múltiples responsabilidades y comprometer la eficiencia y seguridad de la aplicación debido al acceso desde múltiples clases a esta instancia, se ha elegido Singleton cuidadosamente, evitando realizar múltiples interacciones con esta clase y evitando sobrecargar sus responsabilidades. El nombre de la clase elegida en este proyecto para actuar como clase *Singleton* se llama *Videogame*.

### 5.3.4. Observer

El patrón de diseño *Observer*, establece una relación publicador-suscriptor que permite definir un mecanismo de suscripción para notificar a los objetos sobre cualquier evento que le suceda al objeto que están observando. Es un patrón muy útil en el desarrollo de videojuegos porque permite que diferentes partes del sistema estén sincronizadas sin depender directamente unas de otras. Este patrón sugiere que añadas un mecanismo de suscripción a la clase notificadora para que los objetos individuales puedan suscribirse o cancelar su suscripción a un flujo de eventos que proviene de esa notificadora, y cuando le sucede un evento importante al notificador, recorre sus

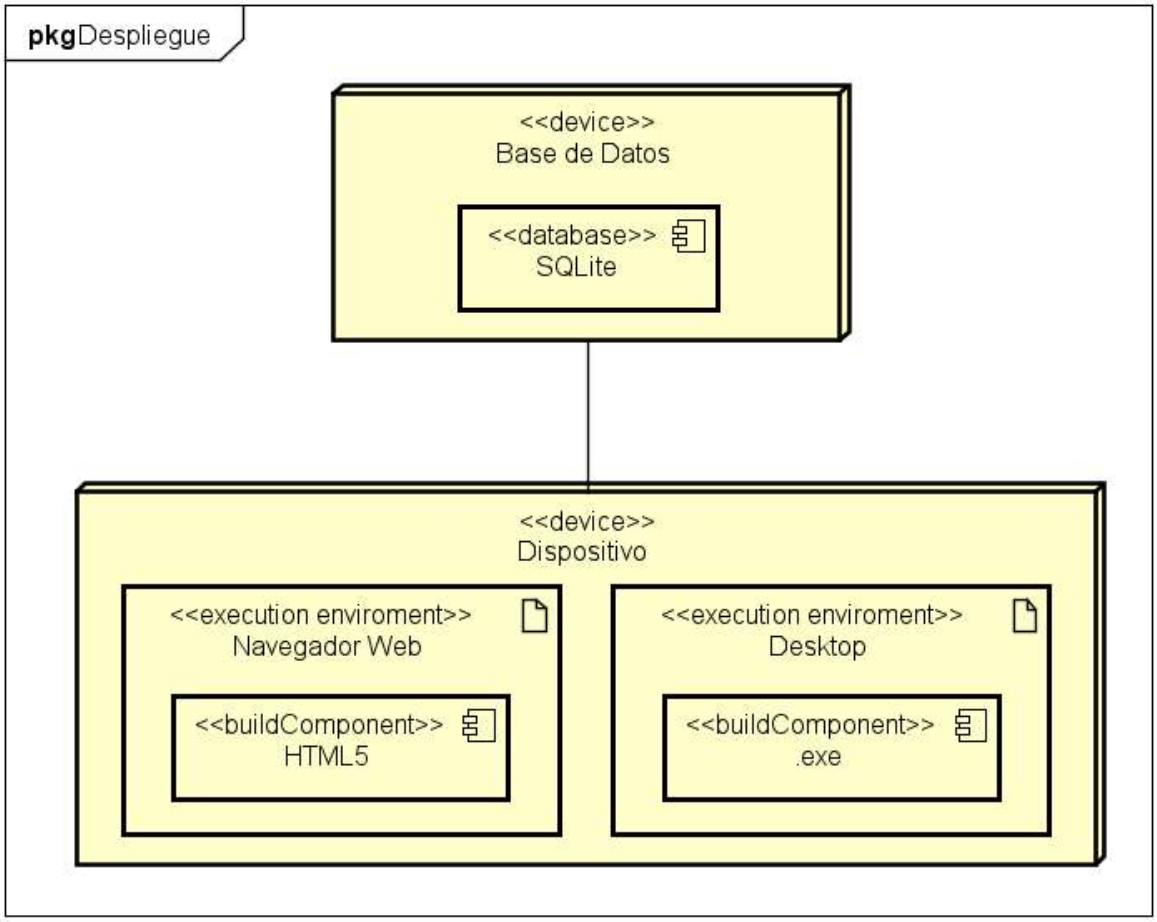


Figura 5.18: Diagrama de despliegue.

suscriptores y llama al método de notificación específico de sus objetos.

El motor Godot Engine implementa este patrón mediante el uso de señales o *signals*, una herramienta incorporada en el propio motor. En este proyecto se ha hecho uso de estas señales para notificar diferentes sucesos ocurridos entre los personajes y elementos del laberinto para poder implementar una lógica de juego coherente e instantánea. El uso de estas señales se da en este proyecto en ejemplos como la notificación por parte de la moneda de ser recolectada por el jugador, o también la notificación por parte tanto del jugador como del enemigo de que han finalizado su movimiento, e incluso en las notificaciones de inicio de sesión y registro del usuario en la aplicación.

## 5.4. Diseño e Interfaz de Usuario

En todo desarrollo de una aplicación, uno de los aspectos fundamentales es el diseño y la configuración de la Interfaz Gráfica de Usuario (GUI), ya que esta es crucial para la retención de los usuarios potenciales y para fomentar su fidelidad, al ofrecerles una experiencia satisfactoria. El propósito principal de las interfaces de usuario es proporcionar una interacción sencilla y agradable, facilitando una navegación intuitiva y eficiente para el usuario.

En las primeras etapas del desarrollo de una aplicación, es necesario plantear el diseño deseado,

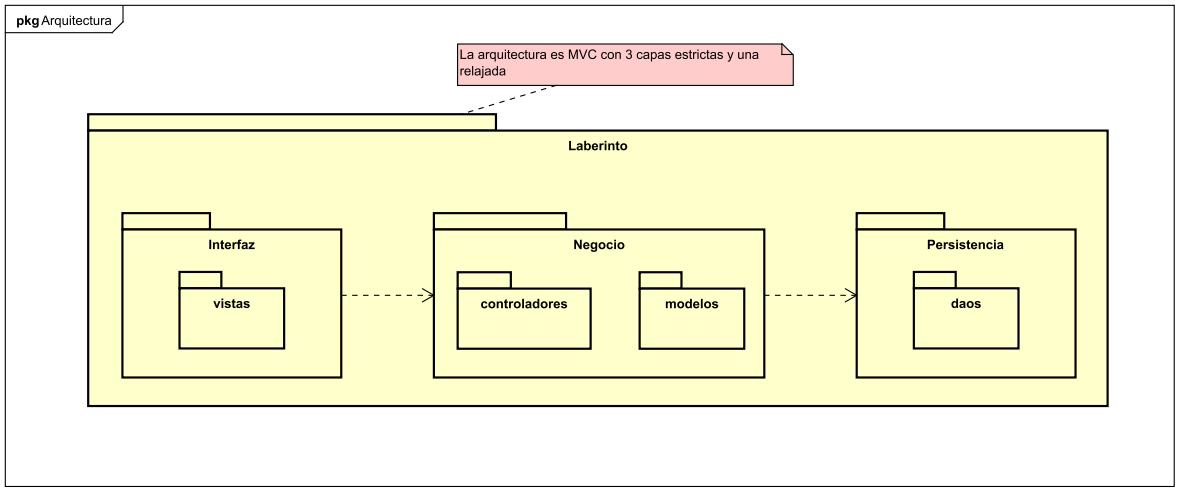


Figura 5.19: Diagrama de arquitectura.

lo cual se lleva a cabo mediante el proceso de prototipado. Este proceso se compone de varias fases: el boceto, el esquema, la maqueta y, finalmente, el prototipo. El *boceto* es un dibujo inicial, poco detallado, del diseño de la aplicación, cuyo objetivo es visualizar por primera vez los elementos que formarán parte de la misma. El *esquema* o *wireframe* representa de manera esquemática la estructura básica de los elementos y su organización. La *maqueta* o *mockup* es una representación más detallada, que incorpora aspectos visuales como imágenes o colores, con el fin de evaluar la viabilidad de su desarrollo. El *prototipo* es una representación que refleja de manera bastante precisa la interfaz del producto final, incluyendo las interacciones entre sus elementos. El proceso de creación del prototipado requiere la configuración de diversas características, siendo las más relevantes para un diseño adecuado de la interfaz de usuario las siguientes: Paleta de colores, Estética de los elementos, Disposición de los elementos, Tipografía, y Navegabilidad y usabilidad.

### 5.4.1. Boceto, esquema y maqueta

El *boceto* y *esquema* muestran las diferentes interfaces de forma poco SWMP y clara, pues su intención es unicamente mostrar de forma muy abstracta y sencilla la disposición de los diferentes elementos que compondrán cada una de ellas. Se usa en la fase temprana de desarrollo del proyecto. La *maqueta* muestra de forma más detallada la organización y forma de los elementos que componen las diferentes pantallas. Permite tener una noción más clara de la distribución del producto final, por lo que se utiliza en etapas intermedias del desarrollo.

A continuación se muestran los bocetos y esquemas de las principales pantallas del videojuego en el siguiente orden: Iniciar Sesión, Menú Principal, Seleccionar Apariencia, Crear Partida, Continuar Partida, Laberinto y Configuraciones del Laberinto y la maqueta de estas interfaces puede visualizarse en la Figura 5.20.

### 5.4.2. Paleta de colores

La sensación de atracción producida por los colores utilizados por una aplicación determina una buena experiencia por parte del usuario tras su uso, lo que se refleja en una fidelidad por

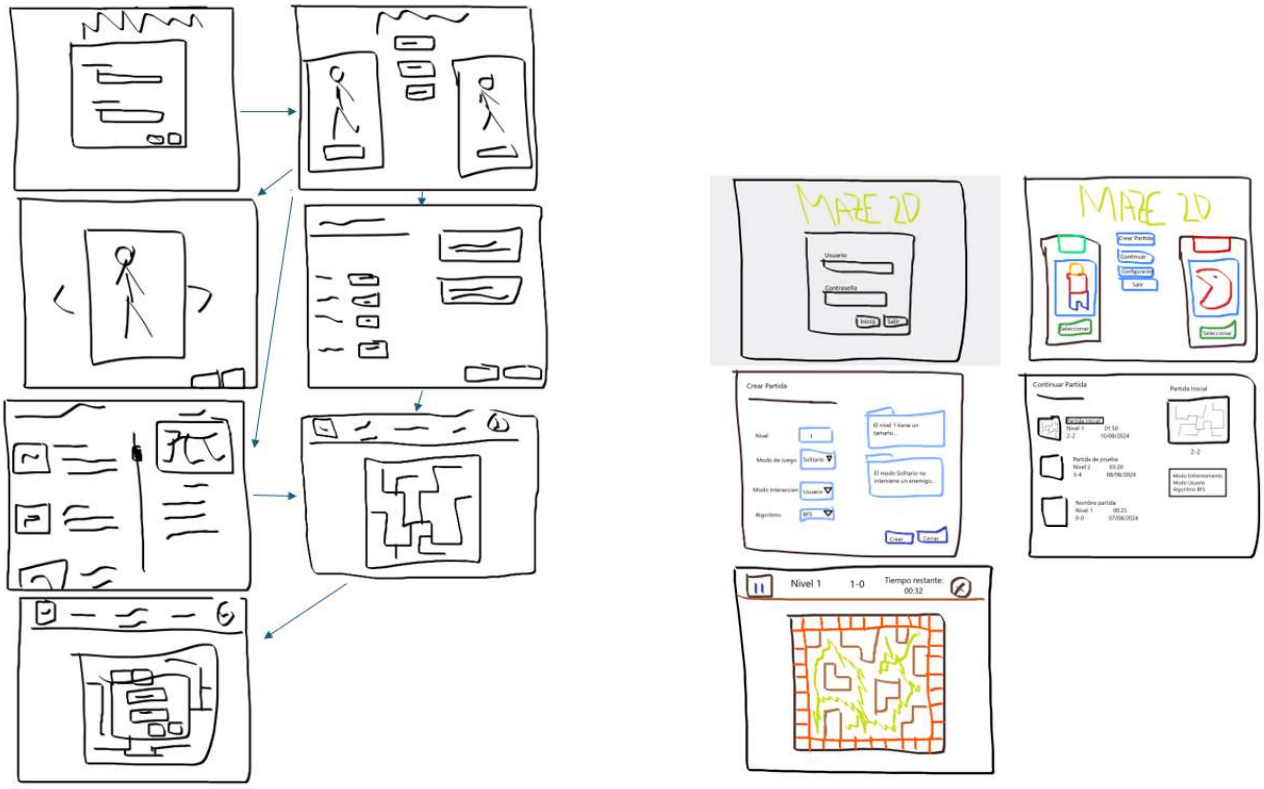


Figura 5.20: Representación del boceto y el esquema de las interfaces del videojuego.

parte del cliente ante el producto. Por ello, una acción importante en el diseño de interfaces de usuario, es la elección de la paleta de colores que forma la aplicación.

En este caso, se ha utilizado el generador de paletas de colores proporcionado por Adobe Color[3], una herramienta que ofrece diferentes combinaciones de colores que permiten la armonía del conjunto de elementos que componen la aplicación. La paleta seleccionada para esta aplicación se compone de tonos cálidos, incluyendo algunos tonos intermedios que permiten la diferenciación y destacar de parte de los elementos del diseño. Esta configuración surge de la regla cromática del cuadrado compuesta por colores complementarios dos a dos (combinación de dos armonías de color complementarias). Su elección se debe a la necesidad de existencia de diversidad de colores, para el diseño de elementos del laberinto con tonos y formas impactantes y cierta extravagancia (colores más fríos y saturados, con cierta luminosidad), y una navegación entre menús mas tranquilas y accesibles (tonos más intermedios y de menor saturación).

[3] https://color.adobe.com/es/create/color-wheel

El uso en este proyecto, en su mayoría de colores azules y verdes radica en el significado de cada uno de ellos. El color azul representa estabilidad, confianza y sabiduría, pues se pretende dar una sensación de estabilidad y confianza al usuario. El color verde es el color del optimismo y de la buena suerte y se relaciona con la naturaleza de ahí su elección al relacionar un laberinto con naturaleza. La Figura 5.21 representa los colores utilizados en este proyecto, extraídos y configurados mediante la herramienta mencionada.

### 5.4.3. Estética de los elementos

Elegida la paleta de colores que compondrá los diferentes elementos de la interfaz del usuario, se determina la forma y estética de dichos elementos. Los diferentes items que componen el videojuego son elaboración propia gracias al uso de estas tres herramientas software:

- **Inkscape:** Diseño de elementos de navegabilidad como menús, cabeceras o botones vectoriales.
- **Pixelorama:** Diseño de componentes del laberinto como muros, personajes o animaciones en formato pixel.
- **Recraft:** Creación de imágenes de forma artificial como fondo de las diferentes pantallas de las que se compone el videojuego.

En los elementos que componen los diferentes menús de navegación, sus botones e iconos presentan una tendencia hacia tonos fríos de color azul o morado, variando sus opacidades y su saturación en los diferentes estados de los que se componen los botones (Normal, Focus, Hover o Pressed) mostrado en la Figura 5.22. El primero hace referencia al diseño del panel de la pantalla inicial donde se presentan las apariencias de los dos personajes del juego cuyo diseño trata de reflejar formas 3D y sensación de fondo. El resto de paneles donde se muestra información relativa a los diferentes eventos de la aplicación como es el guardado, detalles de configuración o el inicio de sesión reflejan ventanas de notificación sencillas. Los botones tienen un diseño enfocado a destacar en cada interfaz aquellos que representan las acciones propias de cada pantalla, como son la creación, selección o cancelación, frente a los botones de configuración de los diferentes elementos.

Por otra parte, se encuentra el diseño de los personajes del videojuego y otros elementos como son las apariencias del jugador, del enemigo y de la moneda, o de los elementos del laberinto *muros* o el *césped* por donde se desplazaran los personajes. Estos elementos también son una elaboración propia, diseñados en el entorno Pixelorama usando el formato pixel en una escala de 32x32 pixels. Estos diseños reflejan personajes abstractos e inventados o basados en creaciones de otros videojuegos o dibujos ya existentes. En el caso del jugador se compone de 4 apariencias (una persona, un diseño basado en pikachu, un perro y un pájaro) como se ve en la Figura 5.23. En el caso del enemigo se compone también de 4 apariencias diferentes (un diseño basado en el enemigo del pac-man, un payaso, y dos zombies) presentados en la Figura 5.23. La moneda tiene una única apariencia que presenta una animación de movimiento, simulando el giro sobre el eje vertical de una moneda.

Por último, se encuentra el diseño de los *Tiles* o elementos del laberinto, en los que se ha diseñado los muros exteriores e interiores del laberinto el césped, basándose en otros diseños de videojuegos, en el caso de los muros interiores se pretende simular paredes de ladrillo, por el contrario los exteriores reflejan una sensación del terror, mientras que el césped hace referencia a los elementos por los que se pueden desplazar los personajes mostrando una sensación más agradable observado en la Figura 5.24.

La diferencia de diseño con el formato pixel para personajes o *tiles* y el formato vectorial para botones, menús o elementos de navegación se ha decidido con el motivo mostrar dicha diferencia de forma clara, y más atractiva para el usuario. Otro de los motivos es demostrar que diferentes tipos de diseños pueden convivir en una misma aplicación respetando la facilidad de uso y de forma intuitiva en armonía con la paleta de colores elegida.

### 5.4.4. Disposición de los elementos

La organización de los diferentes elementos que componen la aplicación, proporcionan un diseño intuitivo y de fácil interacción. Pues se presentará en el centro de la pantalla los elementos con los que se interactúa en la ejecución de cada escena como pueden ser los personajes y los elementos del laberinto, o los botones y campos de configuración del juego (Generales como la resolución o volumen, o del Entorno como son el Nivel, Dificultad...) . En los extremos se presentan los botones cuya función es llevar a cabo acciones de cambios de escena o presentación de nuevos menús (botones de Crear Partida, Cancelar o Iniciar Sesión). Las ventanas emergentes (menús) aparecen en el centro de la pantalla, pues el usuario debe centrar la vista e interacción en dichos menús.

En la cabecera o en los extremos superiores (el izquierdo en su mayoría) muestran información de la interfaz mostrada en ese instante mediante campos de texto como puede ser Crear Partida en la pantalla de creación de una partida o la cabecera incluida en cada uno de los diferentes laberintos, donde se muestra las puntuaciones, tiempos y opciones de configuración entre otros. El diseño se ha pensado y desarrollado de forma adaptativa para una resolución de pantalla inicial de 1920x1080, adaptándose de forma responsiva a otros tamaños de resolución.

### 5.4.5. Tipografía

El tipo de fuente y el tamaño del texto determina la legibilidad de los diferentes campos de texto que componen la interfaz, una buena legibilidad aumenta la usabilidad y la inhabilidad con la aplicación. La tipografía, obtenida en Google Fonts[4], para este proyecto está determinada por el tipo de letra *Changa One* en todos los botones y campos de texto. En cambio en el titulo de la pagina principal se ha usado una fuente llamada *Arbutus* que se asemeja a un diseño de pixeles y hace referencia al estilo de diseño utilizado en parte de este proyecto.

El tamaño esta determinado en pixeles, el que se ha utilizado para el texto de los botones es de 20 pixels dispuestos en el centro de la ventana, un tamaño de 16 pixels para el texto de

[4] https://fonts.google.com/

los botones de los menús de configuración emergentes durante la partida, y un tamaño de 30 pixels para los botones dispuestos en los extremos inferiores para mostrar de forma más clara las acciones a realizar en cada pantalla. Los campos de texto inmutables se ha utilizado un tamaño de 20 pixels acordes al tamaño de los botones. Por ultimo, el diseño del titulo, como se ha mencionado anteriormente tiene un estilo diferente, y su tamaño se ha establecido en 200 pixels.

### 5.4.6. Navegabilidad y Usabilidad

La *navegabilidad* determina la relación entre las diferentes pantallas e interfaces existentes. La navegabilidad repercute en la facilidad con la que el usuario se desplaza por la aplicación, por eso debe ser sencilla y comprensible para que el usuario logre una rápida autonomía. La navegabilidad de este proyecto esta determinada por la interacción con los diferentes botones de los que dispone cada una de las interfaces, pues la transición de una pantalla a otra se produce en su mayoría mediante la pulsación de los botones dispuestos en el extremo inferior derecho de la pantalla. Se compone del botón de cancelación o atrás que redirecciona a la pantalla anterior, y del botón principal que determina la acción que debe completar y llevar a cabo cada pantalla (Iniciar Sesión, Crear Partida, Continuar Partida...).

La *usabilidad* analiza como los usuarios pueden interactuar con la aplicación de forma fácil, cómoda e intuitiva. Su objetivo es lograr la facilidad de aprendizaje y lograr una eficiencia de uso mientras mantiene la seguridad. Para lograr una buena usabilidad se ha contado con diferentes características como es la adaptación a diferentes resoluciones y el mantenimiento de la información de forma visual de manera clara y legible. Por otro lado, se mantiene una protección sobre los datos de los usuarios que permiten identificarse en la aplicación permitiendo cumplir con la seguridad de los datos. Y por último, se mantiene una coherencia y claridad en los enlaces entre ventanas y menús, mostrando claramente la intención de cada uno de los botones que proporcionan la navegabilidad.

### 5.4.7. Jugabilidad

La jugabilidad determina la interacción entre el jugador y el entorno virtual. En este proyecto, la jugabilidad ha sido diseñada cuidadosamente para ofrecer una experiencia inmersiva y desafiante, en la que el jugador debe navegar por diferentes niveles de un laberinto, evitando toparse con obstáculos y enemigos, mientras se dirige a alcanzar la moneda. En el videojuego, se han creado tres laberintos de tamaño y composición fijos, y un laberinto generado aleatoriamente con un tamaño delimitado tanto en el número mínimo como el máximo de casillas que lo componen. Cada laberinto presenta una estructura diferente que fomenta la exploración, la estrategia y la toma de decisiones, por lo que la aplicación de los algoritmos de búsqueda está determinada a la configuración de cada uno de ellos.

El primer nivel tiene un tamaño de 15x15 casillas, permitiendo al usuario familiarizarse con el entorno, aprendiendo el funcionamiento del videojuego y las experiencias que ofrece, presentando diferentes recorridos claramente definidos entre el jugador y la moneda. El segundo nivel aumenta

la dificultad del juego, puesto que su tamaño es relativamente mayor, de 25x20 casillas y se ha pensado para un usuario más experimentado en el juego, es decir, que ya ha navegado y conoce el funcionamiento. Por último, un tercer nivel de 50x25 casillas para usuarios más expertos con múltiples trayectorias entre el jugador y la moneda, y entre el enemigo y el jugador. Por otro lado, existe un nivel generado aleatoriamente, es decir, cuyo tamaño y conjunto de elementos no está predefinido antes de iniciarse el juego. Este laberinto tendrá un tamaño entre 15x15 y 40x25 casillas, elegido de manera aleatoria, como también las posiciones de los personajes jugador y enemigo, o de la moneda. Con las posiciones y el tamaño definidas, se crea un mapa de elementos asegurando que existen trayectorias que conecten a todos los personajes mediante la aplicación del algoritmo de búsqueda en profundidad.

En el ámbito de la jugabilidad, también cabe destacar la posibilidad del usuario de indicar los movimientos que realiza el jugador cuando crea la partida estableciendo el modo de interacción en modo usuario. Cuando se establece este modo, el usuario puede indicar con las flechas del teclado los movimientos que realiza el jugador, permitiendo el desplazamiento en las direcciones izquierda, derecha, arriba y abajo. La indicación de estos movimientos se pueden realizar en el momento que el jugador ha finalizado el desplazamiento, es decir, cuando ha llegado a la posición objetivo.

En adicción, para mejorar la jugabilidad de este videojuego se ha diseñado un panel incluido en la interfaz propia de la dinámica del juego que muestra el resultado de las victorias y derrotas obtenidas por el jugador, determinadas por el alcance de la moneda (las victorias) y el alcance del enemigo o la finalización del tiempo (las derrotas). Este panel además, muestra el tiempo restante para la ejecución del actual juego. Para aumentar la percepción de estos resultados, tanto de la victoria como de la derrota, se ha incluido un panel que incluye una etiqueta (*label*) con el texto relativo al resultado obtenido en cada juego, mostrado en el centro de la pantalla. A estas etiquetas se le añade otra que indica el final del juego, mostrada en la zona inferior central de la pantalla.

### 5.4.8. Otras características de diseño

En otros estilos utilizados, destacan las fotografías utilizadas como *background* de las interfaces iniciales para mejorar la apariencia visual y transmitir un entorno más agradable. El diseño de estas imágenes se ha realizado mediante la utilización de una herramienta para la generación de imágenes a través de inteligencia artificial (llamada Recraft). Estás imágenes representan videojuegos de basados en el juego del laberinto y se añaden a la decoración de los diferentes menús cuya observación en segundo plano permite, sin invadir los elementos principales, el aumento de la atracción visual por parte del usuario.

### 5.4.9. Prototipado

Definidos los criterios de diseño deseados en la aplicación final, se comienza el desarrollo del prototipo cuya intención es proporcionar una visión bastante realista del producto final. Se ha utilizado la herramienta Figma para el desarrollo de prototipos. A continuación se presentan en la Figura 5.25 y la Figura 5.26, de forma general, el prototipo de las diferentes pantallas de las

que se compondrá el videojuego en su fase de desarrollo y que será mejorado para obtener un producto completamente intuitivo y usable en la fase de implementación. Estos prototipos serán muy similares a lo obtenido en el producto final. En el Anexo 10 se puede observar más clara y detalladamente cada uno de los prototipos siguientes.

### 5.4.10. Música

Para ambientar el videojuego y enriquecer la experiencia del jugador, se han creado diversas melodías utilizando la herramienta en línea Soundtrap, especializada en la creación de música. Estas composiciones tienen como objetivo no solo acompañar el desarrollo del juego, sino también sumergir al usuario en una atmósfera creativa y atractiva. Para llevar a cabo este objetivo, se han producido dos pistas principales. La primera es una melodía basada en un estilo de piano llamado *Dark Chi*, que proporciona un tono enigmático y relajante. Esta pieza está acompañada por una base rítmica del estilo *Bass*, cuyo propósito es animar y dar dinamismo a la navegación por las interfaces del juego, como el menú principal, la creación de partidas y la selección de opciones. Por otro lado, la música de la interfaz principal del juego está ambientada con sonidos que evocan el estilo retro del mundo pixel art. Para lograrlo, se han combinado elementos de los estilos *Synth* y *Drums*, que añaden un toque vibrante y característico a la experiencia sonora. Estas pistas no solo cumplen con la función de ambientar, sino que también contribuyen a crear una experiencia inmersiva, tanto a nivel visual como auditivo, haciendo que el jugador se sienta más involucrado en el juego. El enfoque en la ambientación sonora, en complementación con la visual, busca destacar la identidad del juego, permitiendo establecer una conexión emocional con el usuario y mejorando la calidad de la experiencia global, ayudando a captar y mantener la atención del jugador en el videojuego.

## 5.5. Modelo de datos (DB)

El modelo de datos muestra la estructura lógica que compone la base de datos (su tipo y relaciones), además de las condiciones de integridad (restricciones y limitaciones) y las operaciones que permiten la manipulación de los mismos. A continuación se puede observar el diagrama referente a las entidades almacenadas en la base de datos del sistema. La Figura 5.27 muestra el diagrama del modelo de datos con las entidades y atributos que se almacenan en la base de datos del sistema del videojuego.

Se observa un diagrama similar al modelo de dominio pero bastante simplificado. La clase principal se mantiene *Partida* que además de todos los datos de configuración de la partida almacena el id del usuario, del jugador y del enemigo permitiendo acceder a los datos almacenados de cada uno de ellos. A los datos de configuración de la partida, se suman los datos de configuración del *Nivel* que determinan el tamaño y las posiciones iniciales de cada uno de los elementos así como el mapa que forma cada *muro* y cada *césped* del laberinto. El *Usuario* mantiene sus atributos y permitiendo la autentificación en el sistema mediante «user» y «password» y cuenta con una lista de las partidas guardadas.

Por otro lado, la partida contiene dos personajes *Jugador* y *Enemigo* que almacena de forma independiente los datos referentes a los mismos. Estos datos son el la posición que ocupa en el laberinto, el algoritmo, la apariencia y el id del camino propio de cada uno. En el caso del jugador y el enemigo se dispone de otra entidad llamada *Camino* que almacenan los atributos con los que cuenta en el modelo de dominio general. Por último, la partida dispone del *Juego* almacena el numero del juego actual, el estado y el tiempo restante.


Figura 5.21: Paleta de colores utilizada en los elementos que componen las interfaces del videojuego.

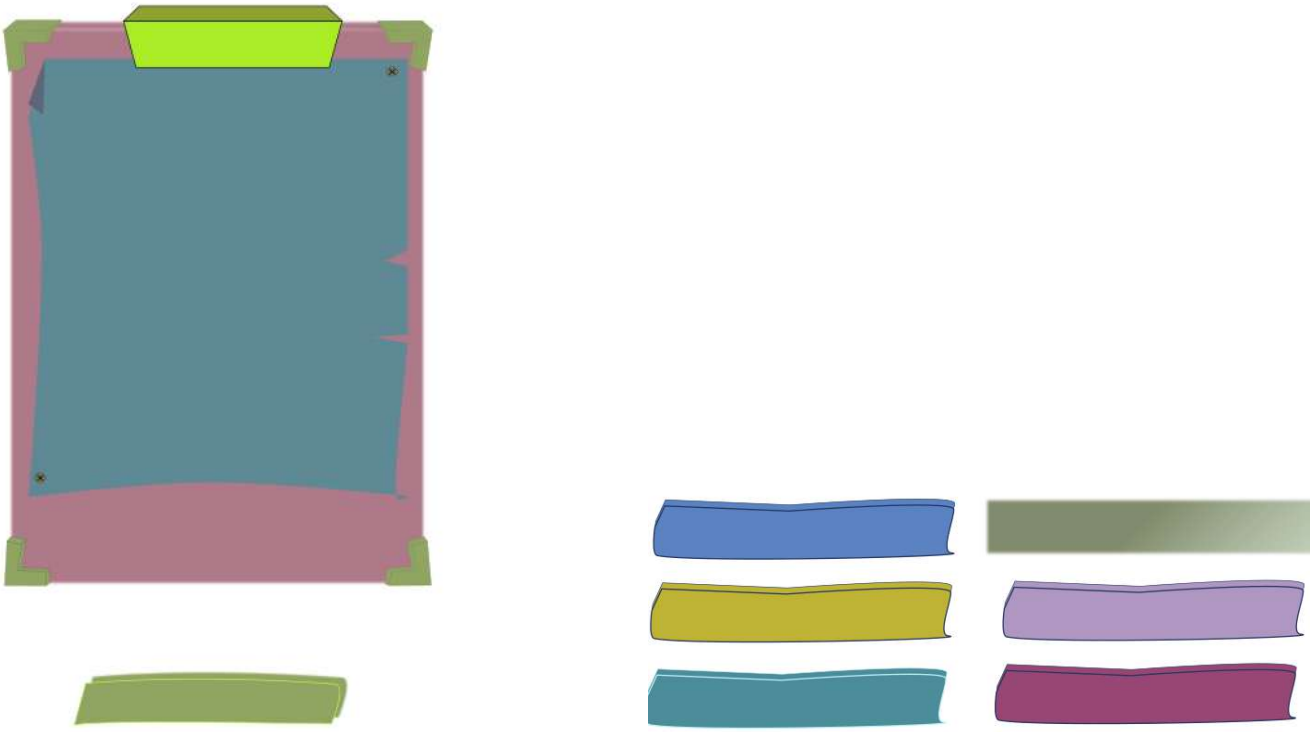

Figura 5.22: Estética creada en Inkscape para los paneles y los botones.

Figura 5.23: Apariencias creadas en Pixelorama del jugador y del enemigo del videojuego.

Figura 5.24: Apariencias creadas en Pixelorama de los tiles del videojuego.

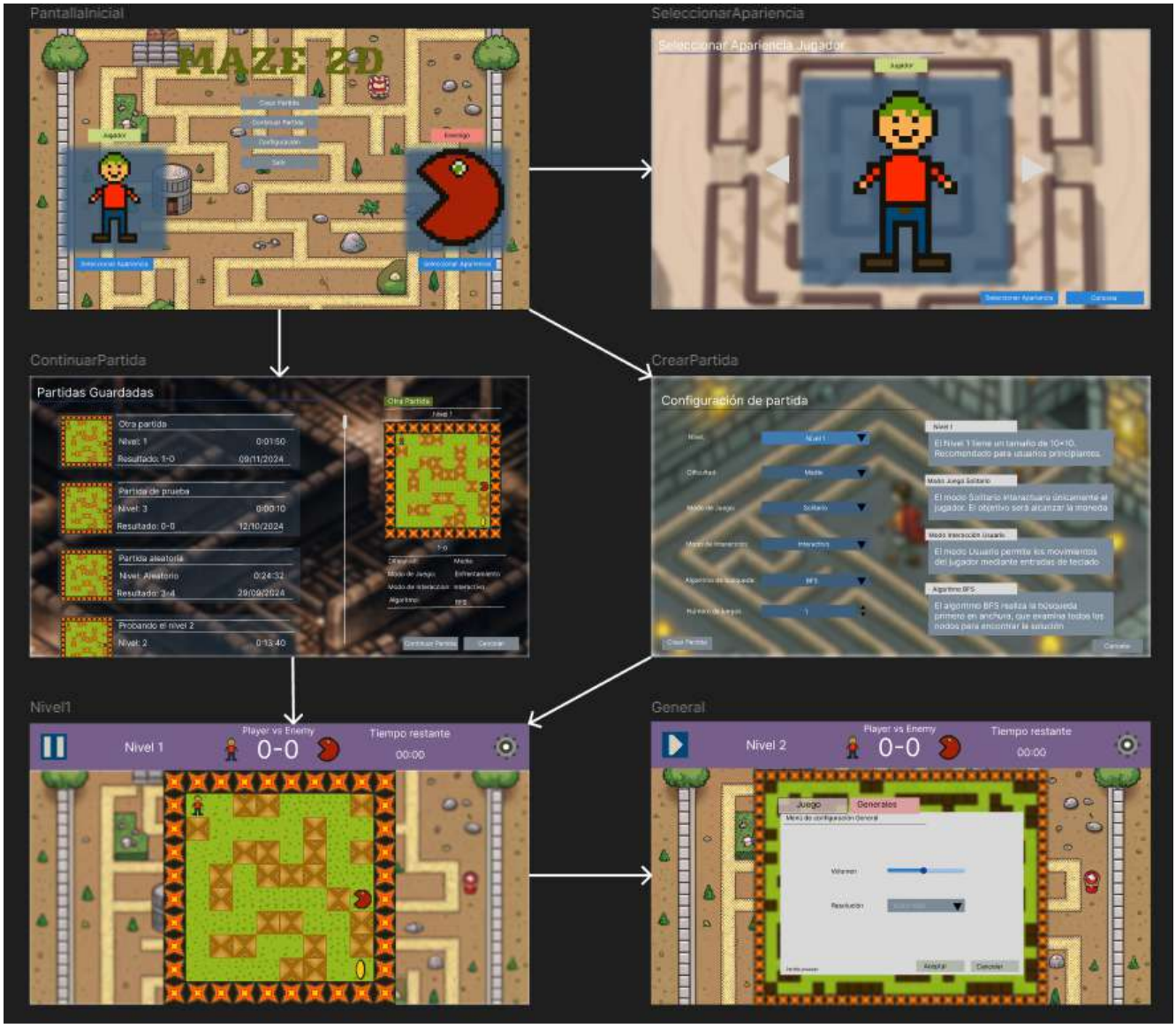


Figura 5.25: Diseño del prototipo inicial con las interfaces referentes a la creación y carga del juego.

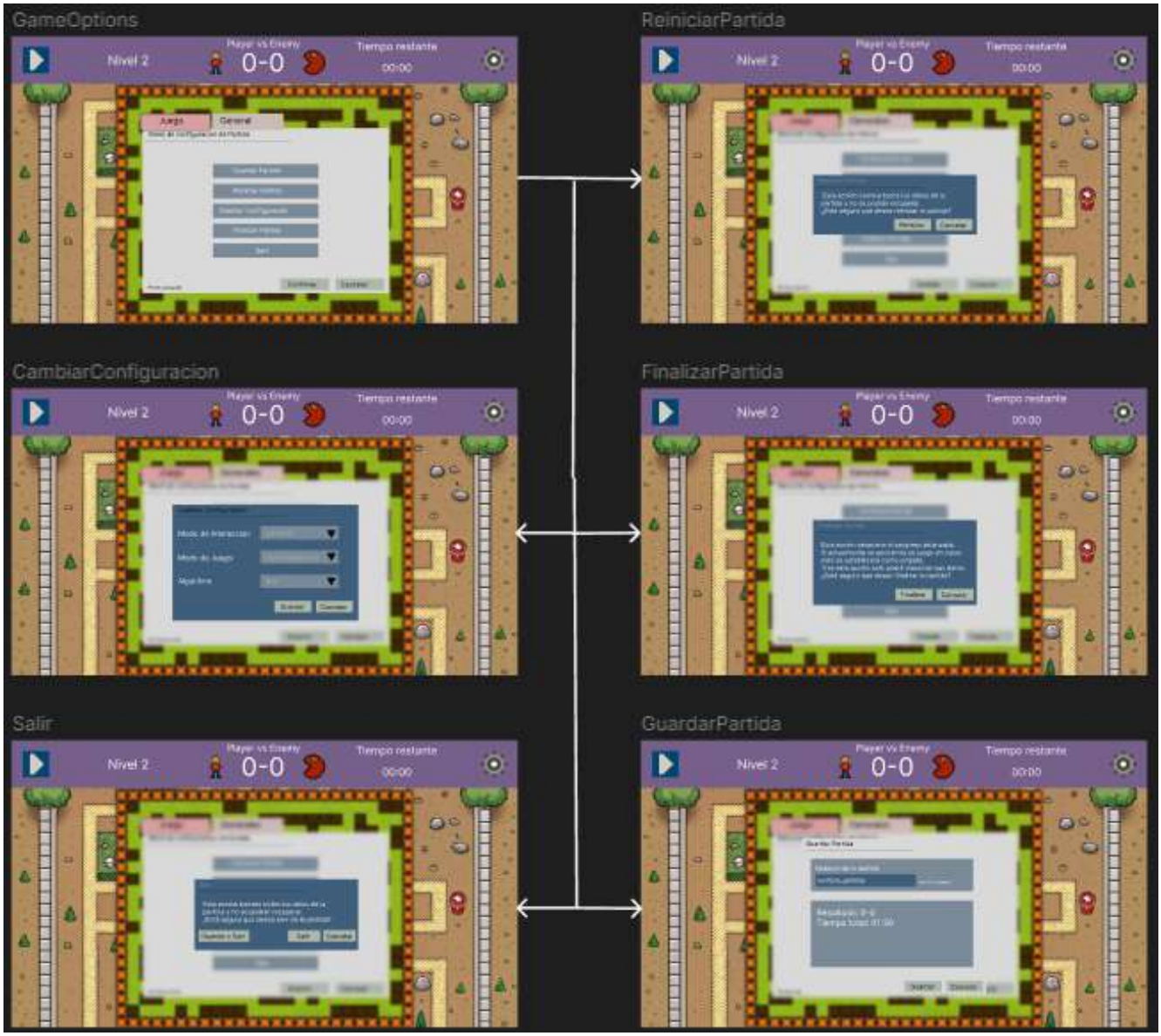


Figura 5.26: Diseño del prototipo inicial con las interfaces referentes a la configuración del juego.

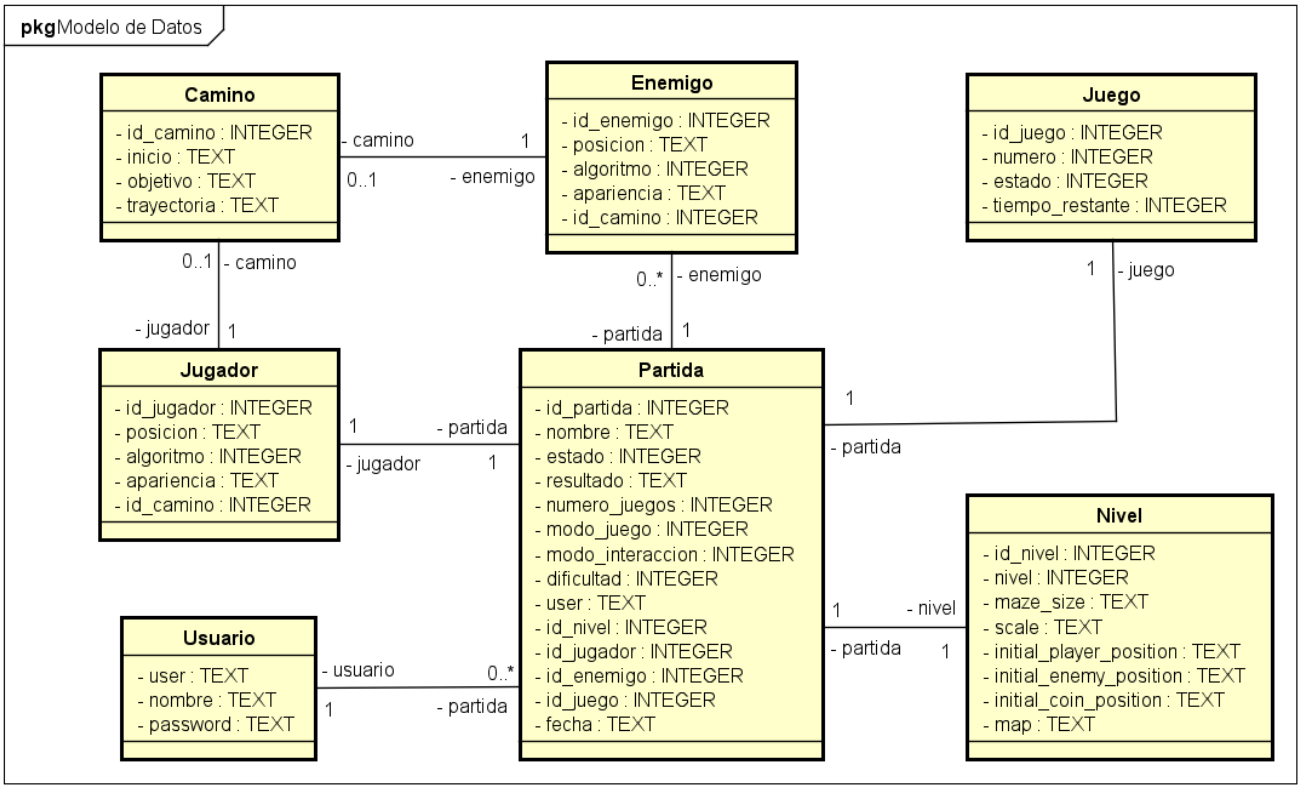


Figura 5.27: Modelo de dominio de la base de datos del videojuego.

# Capítulo 6

# Desarrollo

En este capítulo se realiza la explicación de las implementaciones llevadas a cabo para ajustarse a las soluciones propuestas en la fase de análisis y de diseño tratadas en los dos capítulos anteriores. A continuación, se describe el ajuste a través de herramientas e implementación de código como el ajuste a la arquitectura definida. También se detalla los algoritmos utilizados para las búsqueda y fragmentos de código relevante empleado para crear el Videojuego.

## 6.1. Configuración y Persistencia

En consonancia con la arquitectura definida y el patrón MVC utilizado en este proyecto, el sistema sigue una organización de tres capas estrictas y una relajada. Para ejecutar algunos de los casos de uso y completar diversas acciones, es fundamental establecer la conexión con la base de datos, gestionar adecuadamente los datos y realizar consultas eficientes en cada situación. Específicamente, los casos de uso que requieren conexión a la base de datos son: Identificarse, Registrarse, Guardar Partida y Continuar Partida. En cada uno de ellos, se accede a la base de datos para obtener y almacenar los datos correspondientes. La administración correcta de los datos persistentes se lleva a cabo mediante la herramienta *DB Browser*, utilizada para gestionar bases de datos *SQLite*. Esta biblioteca fue seleccionada para el proyecto debido a su sencillez, amplia compatibilidad con el motor de desarrollo utilizado y facilidad de uso como base de datos local. Gracias a esto, cada ordenador en el que se ejecute el videojuego puede operar de manera independiente, ya que se genera una base de datos local en cada instancia. La base de datos implementada, denominada *db_Maze2D.db*, es accesible a través de una API específica (Maze/DatabaseAPI/database.gd), que facilita la conexión y las operaciones con la base de datos.

```
database = SQLite.new()
var path = "user://db_Maze2D.db"
database.open_db()
```

Listing 6.1: Apertura de la conexión con la base de datos de SQLite llamada db_Maze2D.db.

Las conexiones con la base de datos se establecen durante la ejecución de los casos de uso relacionados con el usuario. Las consultas a la base de datos se implementan mediante cadenas de texto y los métodos proporcionados por la librería de SQLite integrada en el motor de desarrollo Godot, una integración que simplifica y optimiza la ejecución de consultas, permitiendo especificar los parámetros necesarios para realizar cualquier tipo de operación sobre la base de datos. Además, las entidades definidas en el modelo de dominio se representan mediante tablas básicas, cada una con un identificador único, lo que facilita su gestión y manipulación. Al establecer la conexión, si la base de datos no ha sido previamente creada en el entorno de ejecución, se generan automáticamente las tablas definidas en el modelo de dominio. Este proceso se lleva a cabo en el método *createTables()*, que ejecuta las consultas necesarias verificando la existencia de cada tabla. Por ejemplo, la tabla usuario se crea definiendo el campo *user* como clave única mediante las sentencias *TEXT NOT NULL UNIQUE* y *PRIMARY KEY(user)*.

```
CREATE TABLE IF NOT EXISTS "usuario" (
  "user"  TEXT NOT NULL UNIQUE,
  "nombre"  TEXT NOT NULL,
  "password"  TEXT NOT NULL,
  PRIMARY KEY("user")
    )
```

Listing 6.2: Creación de tablas con sus atributos y caracteristicas en la primera ejecución del videojuego.

Una vez establecida la conexión y configuradas correctamente las tablas correspondientes a las entidades del sistema, la API queda disponible para realizar operaciones de inserción, obtención, actualización o eliminación de registros. Estas operaciones son fundamentales para los casos de uso definidos, como la verificación de usuarios registrados o la recuperación de partidas y recursos asociados a un usuario. Al iniciar el juego, el sistema solicita la identificación del usuario o, en su defecto, permite su registro proporcionando un nombre, un *nickname* y una contraseña. En ambos casos, es necesario consultar la base de datos para verificar la existencia del usuario en la tabla *usuario*. Además, en el proceso de inicio de sesión, se debe validar que la contraseña ingresada coincida con la almacenada en la base de datos. Para ello, la consulta que recupera los datos del usuario incluye una cláusula *WHERE* que filtra los resultados según el *nickname*, cuyo atributo en la base de datos se denomina *user*. La ejecución de esta consulta se realiza mediante el método *query_with_bindings(String query_string, Array param_bindings )*, permitiendo una recuperación segura y eficiente de la información del usuario.

```
SELECT * FROM usuario WHERE user = ?
database.query_with_bindings(query, [user])
```

Listing 6.3: Obtención de un usuario de la base de datos mediante la busqueda del atributo user.

Para llevar a cabo las operaciones de inserción, actualización y borrado de recursos en *SQLite*, es necesario especificar el nombre de la tabla que se desea modificar junto con los datos actualizados de la entidad correspondiente. Estas acciones se ejecutan mediante los métodos proporcionados por la librería de SQLite de la siguiente manera.

```
database.insert_row(name_collection, collection)
database.update_rows(name_collection, "id_" + name_collection +
  " = " + str(id), collection)
database.delete_rows(name_collection, "id_" + name_collection +
  " = " + str(id))
```

Listing 6.4: Inserción actualización y borrado de un recurso de la base de datos.

A su vez, la inserción de un nuevo recurso de tipo partida en la base de datos requiere los identificadores de las demás entidades asociadas, ya que estos completan los atributos de la clase *partida* y sirven como referencia a las entidades correspondientes a cada partida almacenada, facilitando así su posterior recuperación. Para obtener estos identificadores y agregarlos correctamente a la clase *partida*, se recupera el identificador del último elemento insertado en la base de datos. Esto se logra mediante una consulta que emplea la función predefinida de *SQLite last_insert_rowid()*, la cual devuelve el *rowid* de la última fila insertada dentro de la conexión de base de datos que la invocó.

```
SELECT last_insert_rowid() AS last_id
```

Listing 6.5: Obtención del identificador del último recuros insertado en la base de datos.

Otra acción que requiere acceso a la base de datos es la carga de las partidas asociadas a un usuario. Para ello, es necesario que el usuario haya iniciado sesión en el sistema, de manera que su dirección de correo electrónico pueda ser utilizada en la cláusula *WHERE* para filtrar los resultados y recuperar únicamente las partidas correspondientes a dicho usuario. Esta consulta implica la combinación de múltiples tablas con la tabla *partida*, ya que es necesario recuperar información de diversas entidades para mostrarla en la pantalla de carga, cuya combinación de datos se logra mediante el uso de las cláusulas *JOIN* y *LEFT JOIN*. La primera cláusula vincula la tabla *partida* con las tablas *juego* y *nivel*, permitiendo obtener el tiempo restante del juego y el nivel asociado a la partida, respectivamente. Esta relación se establece a través del identificador almacenado en la entidad *partida*, que referencia a las dos entidades mencionadas. La segunda cláusula vincula la tabla *partida* con las tablas *jugador* y *enemigo* para recuperar los algoritmos asignados a cada una de estas entidades, también mediante la relación de su identificador en la entidad *partida*. Además, se incluye una cláusula *ORDER BY* con el criterio *DESC* para ordenar las partidas en orden descendente según la fecha de guardado. La sentencia SQL resultante, junto con la ejecución del método *query_with_bindings()*, se presenta a continuación.

```
SELECT partida.*, juego.tiempo_restante, nivel.nivel, jugador.algoritmo as
  algoritmo_jugador, enemigo.algoritmo as algoritmo_enemigo
FROM partida
JOIN juego ON partida.id_juego = juego.id_juego
JOIN nivel ON partida.id_nivel = nivel.id_nivel
LEFT JOIN jugador ON partida.id_jugador = jugador.id_jugador
LEFT JOIN enemigo ON partida.id_enemigo = enemigo.id_enemigo
WHERE user = ?
ORDER BY partida.fecha DESC
```

Listing 6.6: Obtención de las partidas pertenencientes al jugador registrado en el videojuego.

Por último, si el usuario selecciona una partida específica, es necesario recuperar todas las entidades referenciadas en dicha partida para reconstruir la configuración del juego y permitir su continuación desde el punto de progreso guardado. La obtención de estos recursos se basa en el identificador de cada entidad, cuyos valores están almacenados en los atributos de la tabla *partida*. El proceso de recuperación de estas entidades sigue una estructura similar, diferenciándose únicamente en el nombre de la tabla referenciada en la cláusula *FROM* y en el valor utilizado en la cláusula *WHERE*, correspondiente al identificador de dicha entidad. A continuación, se muestra un ejemplo de consulta para recuperar los datos de la entidad *juego*, análoga para las demás entidades.

```
SELECT * FROM juego WHERE id_juego = ?
database.query_with_bindings(query, [id])
```

Listing 6.7: Obtención de un juego dado su identificador.

## 6.2. Algoritmos

Este trabajo tiene como finalidad el cumplimiento de un objetivo enfocado en el desarrollo de una aplicación con carácter educativo que sirva de prototipo para la enseñanza de los diferentes algoritmos de búsqueda, que se materializa mediante el análisis, diseño e implementación de un videojuego desarrollado en *Godot Engine*. Un algoritmo de búsqueda es un conjunto de instrucciones que están diseñadas para localizar un elemento con ciertas propiedades dentro de una estructura de datos, es decir, buscar un estado concreto dentro de un conjunto de datos llamado espacio de estados mediante una estrategia determinada por el algoritmo aplicado. La búsqueda puede ser mediante algoritmos no informados o ciegos, más ineficientes en tiempo y memoria que otros métodos, o algoritmos informados, más eficientes y divididos en heurísticos o de búsqueda con adversario. El videojuego está desarrollado implementando 4 de estos algoritmos de búsqueda, el algoritmo primero en anchura, el algoritmo primero en profundidad, el algoritmo de Dijkstra o el algoritmo A*.

El algoritmo *Primero en Anchura* es un algoritmo de búsqueda no informada que permite recorrer y buscar elementos en un grafo, usualmente utilizado en arboles, y supone que el recorrido se realice por niveles. La búsqueda comienza en el nodo inicial o raíz, y se continua con la exploración de todos los nodos vecinos a ese nodo, donde una vez explorado todos los nodos, se realiza la exploración de los nodos adyacentes a cada uno de los vecinos explorados, realizando esta secuencia continuamente hasta recorrer todos los nodos del grafo o árbol, o en su defecto, se llegue a un nodo establecido como objetivo. Este algoritmo expande y examina todos los nodos de un árbol sucesivamente hasta encontrar una solución, omitiendo el uso de estrategias heurísticas y garantizando encontrar una solución incluso para grafos infinitos. En términos de complejidad computacionales, este algoritmo se expresa como *O(|vértices |+ |aristas |)*, en cambio, la complejidad en el aspecto de memoria es de *O(|vértices |)*. La implementación de este algoritmo en el videojuego se lleva a cabo mediante el método *AlgorithmController.bfsSearch()*, y utiliza una cola de tipo *Array* para añadir los nodos pendientes de visitar, y una para almacenar los nodos visitados, y un diccionario (*Dictionary*), para almacenar los nodos vecinos de un nodo,

respectivamente. Con estas variables, el algoritmo recorre la lista de nodos hasta recorrer todos o encontrar el nodo objetivo, visitando los nodos vecinos no visitados, y guardando los nodos adyacentes de cada nodo que permite recrear el recorrido desde el nodo inicial hasta el nodo final. La secuencia descrita, implementada en el videojuego es la siguiente.

```
func bfsSearch(start_node: Vector2, end_node: Vector2):
  var queue = []
  var parent = {}
  visited = []
  queue.append(start_node)
  visited.append(start_node)
  parent[start_node] = null

  while queue:
    var current_node = queue.pop_front()
    if current_node == end_node:
      await printSearchTiles()
      return await createPath(start_node, end_node, parent)

    for neighbor in get_neighbors(current_node):
      if neighbor not in visited:
        queue.append(neighbor)
        visited.append(neighbor)
        parent[neighbor] = current_node
```

Listing 6.8: Implementación del algoritmo primero en anchura empleada en el videojuego.

El algoritmo *Primero en Profundidad* es un algoritmo de búsqueda no informada que permite recorrer y buscar elementos o nodos en un grafo, usualmente utilizado en arboles, y sin importancia en los niveles de los que dispone cada nodo. La búsqueda comienza en el nodo inicial o raíz, y se continua el primer nodo vecino de cada nodo, expandiendo todos y cada uno de los nodos que va localizando de forma recurrente en un camino, hasta alcanzar el nodo final del camino, donde regresa hasta encontrar un nuevo camino disponible para recorrer. Este algoritmo no asegura la optimización, pues puede encontrar una solución más profunda que otra que no ha sido aún expandidas. La implementación de este algoritmo en el videojuego, se basa en una búsqueda recursiva mediante el método *AlgorithmController.recursiveDFS()*, que utiliza una cola de tipo (*Array*), para almacenar los nodos visitados que se pasa como parámetro en cada ejecución de la función recursiva. El algoritmo recorre recursivamente el primer nodo hijo del nodo inicial llegar al final de ese recorrido, cuando realiza el *backtracking* para recorrer otro camino diferente disponible. Este algoritmo algoritmo hace uso de una variable global para almacenar los nodos y sus adyacentes una vez recorridos, que sera la que retorne una vez finalizado el algoritmo. Esta secuencia descrita y llevada acabo por el algoritmo primero en anchura, se implementa en el videojuego de la siguiente forma.

```
func dfsSearch(start_node: Vector2, end_node: Vector2):
  visited = []
  resultdfs = []
  recursiveDFS(start_node, end_node)
  await printSearchTiles()
  return resultdfs

func recursiveDFS(start: Vector2, end: Vector2):
  if start not in visited:
    visited.append(start)
    if start == end:
      return true

    for neighbor in get_neighbors(start):
      if recursiveDFS(neighbor, end):
        resultdfs.push_front(neighbor)
        return true

  return false
```

Listing 6.9: Implementación del algoritmo primero en profundidad empleada en el videojuego.

El algoritmo de *Dijkstra*, o de caminos mínimos, es un método de búsqueda en grafos que determina el camino más corto desde un vértice origen hasta el resto de los vértices en un grafo con pesos positivos en sus aristas. La búsqueda comienza en el nodo inicial y explora iterativamente los caminos más cortos hasta alcanzar el nodo objetivo. Su complejidad se expresa como $O(|vértices|\hat{2} + |aristas|)$ cuando se utiliza una cola de prioridad. En el videojuego, la implementación de este algoritmo se lleva a cabo mediante el método *AlgorithmController.dijkstraSearch()*. Se emplea una cola de prioridad de tipo *Array* para gestionar los nodos pendientes de visitar y dos diccionarios (*Dictionary*): uno para almacenar los nodos vecinos y otro para registrar las distancias acumuladas. Además, se usa un tercer diccionario para inicializar los pesos de cada arista con un valor neutro, mediante el método *AlgorithmController.inicializeNodesWeigth()*. El algoritmo recorre los nodos hasta visitar todos o encontrar el objetivo. Para cada nodo, explora sus vecinos no visitados, calcula la nueva distancia en función del peso de la arista con el método *AlgorithmController.getCost()* y actualiza el valor si encuentra una distancia menor. También almacena los nodos adyacentes, permitiendo reconstruir el recorrido desde el nodo inicial hasta el final. La implementación de este algoritmo en el videojuego es la siguiente.

```
func dijkstraSearch(start_node: Vector2, end_node: Vector2,
          avoid_position: Vector2):
  var distances = {}
  var parent = {}
  var heap = []
  visited = []

  var asign = inicializeNodesWeigth(start_node, distances, parent)
  distances = asign[0]
  parent = asign[1]
  pushHeap(heap, [0, start_node])

  while heap:
    var node = popHeap(heap)
    var accumulated_cost = node[0]
    var current_node = node[1]

    if current_node in visited:
      continue

    visited.append(current_node)

    if current_node == end_node:
      await printSearchTiles()
      return await createPath(start_node, end_node, parent)

    for neighbor in get_neighbors(current_node):
      var cost = getCost(neighbor, avoid_position)
      var new_cost = accumulated_cost + cost

      if new_cost < distances[neighbor]:
        distances[neighbor] = new_cost
        parent[neighbor] = current_node
        pushHeap(heap, [new_cost, neighbor])

  return []
```

Listing 6.10: Implementación del algoritmo dijkstra empleada en el videojuego.

El algoritmo *A** es un método de búsqueda informada o heurística que permite recorrer y buscar elementos en un grafo y garantiza obtener siempre el camino de menor coste desde un vértice origen hasta un objetivo en un grafo con pesos positivos en sus aristas. Su funcionamiento se basa en una estrategia heurística definida antes de la ejecución. En el videojuego, la heurística se genera mediante la función *AlgorithmController.createHeuristic()*, diferenciándose para el jugador y el enemigo. La heurística del jugador se basa en la distancia de cada nodo a la moneda: cuanto mayor sea la distancia, mayor será su valor heurístico. En cambio, la del enemigo depende de la posición del jugador, asignando valores menores a los nodos más cercanos a él. Para calcular estos valores, se utiliza *la distancia de Manhattan* [35], sumando las diferencias absolutas de las coordenadas y dividiendo entre 32, que es el numero de pixels que se desplaza en cada movimiento, es decir, la distancia entre cada nodo colindante.

```
func createHeuristic(x_size: int, y_size:int,
          heuristic_position: Vector2 = Vector2(0,0)):
  var heuristic = {}
  penalty = (x_size + y_size) / 10

  for node in graph.keys():
    if heuristic_position != Vector2(0,0):
      heuristic[node] = (abs(heuristic_position.x - node.x) +
      abs(heuristic_position.y - node.y))/ pixels_move

  return heuristic
```

Listing 6.11: Creación de la heuristica de los personajes basada en la distancia de Manhattan.

El algoritmo A* comienza desde el nodo inicial y explora iterativamente los caminos más cortos, considerando la heurística asignada a cada nodo. Su implementación en el videojuego es similar a la del algoritmo de Dijkstra y se realiza mediante el método *AlgorithmController.aStarSearch()*. Se emplea una cola de prioridad para gestionar los nodos pendientes de visitar y dos diccionarios: uno para almacenar los nodos vecinos y otro para registrar las distancias acumuladas. Inicialmente, los pesos de las aristas se establecen en un valor neutro mediante *AlgorithmController.inicializeNodesWeigth()*. Durante la búsqueda, el algoritmo visita nodos no explorados, calcula la distancia entre vértices sumando la distancia acumulada en el nodo actual, el peso asignado al nodo vecino con *AlgorithmController.getCost()* y su valor heurístico, cumpliendo así $f(n) = g(n) + h(n)$. Si la nueva distancia es menor que la acumulada, se actualiza y se almacenan los nodos adyacentes, permitiendo reconstruir el camino óptimo. La implementación detallada de este algoritmo se muestra a continuación.

```
func aStarSearch(start_node: Vector2, end_node: Vector2, heuristic: Dictionary,
        avoid_position: Vector2):
  var distances = {}
  var parent = {}
  var heap = []
  visited = []

  var asign = inicializeNodesWeigth(start_node, distances, parent)
  distances = asign[0]
  parent = asign[1]
  heap.append([0, start_node])

  while heap:
    var node = popHeap(heap)
    var accumulated_cost = node[0]
    var current_node = node[1]

    if current_node in visited:
      continue
    visited.append(current_node)

    if current_node == end_node:
      await printSearchTiles()
      return await createPath(start_node, end_node, parent)

    for neighbor in get_neighbors(current_node):
      var cost = getCost(neighbor, avoid_position)
      var movement_cost = cost + heuristic[neighbor]
      var new_cost = accumulated_cost + movement_cost

      if new_cost < distances[neighbor]:
        distances[neighbor] = new_cost
        parent[neighbor] = current_node
        pushHeap(heap, [new_cost, neighbor])

  return []
```

Listing 6.12: Implementación del algoritmo a estrella empleada en el videojuego.

Las implementaciones de los algoritmos Dijkstra y A* utilizan una cola de prioridad para explorar los nodos pendientes de visitar, ordenados de forma ascendente según el coste de desplazamiento desde el nodo inicial. Para gestionar la inserción y extracción de elementos en esta cola, se han definido los métodos *AlgorithmController.pushHeap()* y *AlgorithmController.popHeap()*. El método de inserción ordena la cola utilizando la función *AlgorithmController.sortAscending()*, garantizando que los nodos con menor coste sean procesados primero.

```
func pushHeap(heap, element):
  heap.append(element)
  heap.sort_custom(sortAscending)

func sortAscending(a, b):
  if a[0] < b[0]:
    return true
  return  false

func popHeap(heap):
  if heap.is_empty():
    return null
  else:
    return heap.pop_front()
```

Listing 6.13: Métodos de inserción extracción y ordenamiento de nodos de la cola de prioridad.

Por otro lado, ambos algoritmos hacen uso de la función *AlgorithmController.getCost()* para obtener el coste que supone desplazarse desde un nodo hasta otro, que tiene en cuenta la posición que debe evitar el personaje, generando una penalización y aumento del coste cuanto mayor es la cercanía a dicha posición. Estas funciones se describen a continuación.

```
func getCost(next_position: Vector2, avoid_position: Vector2):
  var default_cost = 1
  var distance_to_enemy = abs(avoid_position.x - next_position.x) +
  abs(avoid_position.y - next_position.y)
  var penalty_cost = max(penalty - distance_to_enemy, default_cost)
  return default_cost + penalty_cost
```

Listing 6.14: Método de calculo del peso de movimiento entre dos nodos.

En cada algoritmo definido anteriormente, está implementada la representación gráfica y visual del recorrido que realiza cada uno de los algoritmos en la búsqueda del nodo final, desde la posición inicial, permitiendo ajustarse al objetivo inicial, de permitir la visualización y comprensión del funcionamiento de cada uno de los algoritmos. Esta representación es realizada unicamente en la ejecución producida por los algoritmos llevados a cabo por el jugador, para simplificar la visualización y desacoplar la ejecución de ambos. Esta representación utiliza las coordenadas del *TileMap* gracias al método *local_to_map()* para obtener la posición del nodo y reemplaza el tile anterior, por un tile definido para este supuesto mediante la función *set_cell()*, ambos proporcionados por el entorno de desarrollo Godot. La representación del funcionamiento de estos algoritmos se lleva a cabo de la siguiente manera, unicamente siendo visible para el funcionamiento del jugador.

```
func printSearchTiles():
  var size = visited.size()
  var time_await: float = 1.0/(size*1000000)

  for node in visited:
    if is_player:
      var cell = tilemap.local_to_map(node)
      var atlas_coords = Vector2i(0, 0)
      tilemap.set_cell(0, cell, 7, atlas_coords)
      await scene.get_tree().create_timer(time_await).timeout
```

Listing 6.15: Método de representación grafica en el mapa del laberinto del funcionamiento de los algoritmos.

Finalizado la ejecución de los algoritmos, y obtenido el grafo con los nodos vecinos de cada nodo, recrea el camino solución desde el nodo inicial hasta el nodo final mediante la función *AlgorithmController.createPath()*. Este método recorre en reversa el grafo, desde el nodo final, agregando al principio del array resultado cada nodo que recorre hasta alcanzar el nodo inicial.

```
func createPath(start_node: Vector2, end_node: Vector2, parent: Dictionary):
  var path = []
  var current_node = end_node

  while current_node != start_node:
    path.insert(0, current_node)
    current_node = parent[current_node]

  return path
```

Listing 6.16: Recreación de la trayectoria en base a los datos obtenidos de la aplicación del algoritmo de busqueda.

El uso de estos algoritmos en el videojuego presenta claras diferencias. Mientras que los algoritmos de *búsqueda en anchura* y *búsqueda en profundidad*, se ejecutan unicamente en primera instancia al inicio del juego, obteniendo el camino desde el inicio hasta el final, y ejecutando los movimientos en base a ese camino inicial encontrado, sin presentar modificaciones hasta el final del recorrido. En cambio, los algoritmos de *Dijkstra* y *A Estrella*, se ejecutan en cada interacción de movimiento producido por los personajes, es decir, crean el camino inicialmente al comienzo del juego, y posterior a cada movimiento producido por los personajes, permitiendo recrear nuevas trayectorias, modificando la calidad de la trayectoria, como por ejemplo, el caso de que el enemigo obtenga una nueva trayectoria que no atraviese el recorrido obtenido por el enemigo.

La implementación de estos algoritmos requiere la creación de un grafo con los nodos hijos de cada nodo, que permite crear el árbol de nodos y recorrerlo para obtener las trayectorias de cada personaje hasta su objetivo. La creación del grafo esta condicionada por el siguiente orden de adicción de nodos vecinos de un nodo en el mapa del laberinto correspondiente: Primero, se añade el nodo inmediatamente a la derecha; Después, se añade el nodo inmediatamente abajo del nodo, A continuación, se añade el nodo inmediatamente superior del nodo; Por ultimo, se añade el nodo inmediatamente a la izquierda del nodo. Previamente a la creación del grafo, se crea un array

multidimensional con el identificador del *tile* correspondiente a cada nodo, siendo 0 el identificador del tile de césped, donde esta permitido el movimiento, y por el contrario el 1,2 y 3 corresponden a los identificadores de tiles de muros, donde no está permitido el movimiento.

```
func createMap(x_size: int, y_size:int):
  for i in range(80, y_size+64, pixels_move):
    var row = []
    for j in range(80, x_size+64, pixels_move):
      var cell = tilemap.local_to_map(Vector2(j, i))
      var id = tilemap.get_cell_source_id(0, cell)
      row.append(id)
    map.append(row)
  createGraph(x_size, y_size)

func createGraph(xSize: int, ySize: int):
  var childs = []
  for i in range(1, ySize/pixels_move - 1):
    for j in range(1, xSize/pixels_move - 1):
      if map[i][j] == 0:
        if map[i][j+1] == 0:
          childs.append(Vector2(pixels_center + pixels_move*(j+1) +
           pixels_offset, pixels_center + pixels_move*i + pixels_offset))
        if map[i+1][j] == 0:
          childs.append(Vector2(pixels_center + pixels_move*j +
           pixels_offset, pixels_center + pixels_move*(i+1) + pixels_offset))
        if map[i-1][j] == 0:
          childs.append(Vector2(pixels_center + pixels_move*j +
             pixels_offset, pixels_center + pixels_move*(i-1) + pixels_offset))
        if map[i][j-1] == 0:
          childs.append(Vector2(pixels_center + pixels_move*(j-1) +
           pixels_offset, pixels_center + pixels_move*i + pixels_offset))
        graph[Vector2(pixels_center + pixels_move*j + pixels_offset,
        pixels_center + pixels_move*i + pixels_offset)] = childs
        childs = []
```

Listing 6.17: Creación del mapa y del grafo correspondiente a la disposición de los elementos del laberinto.

## 6.3. Explicación de código relevante

Este videojuego se compone una secuencia de acciones cuya importancia destaca sobre otras. Estas acciones son la dinámica de juego, el calculo de caminos mediante los algoritmos de búsqueda, y la dinámica de movimiento de los personajes. El calculo de caminos está descrito en la sección anterior, por lo que esta sección se limitará a describir las secuencias referentes a la dinámica del juego y de movimientos.

En primer lugar, la dinámica del juego consiste en inicializar las posiciones de los personajes y elementos que componen el laberinto y posteriormente realizar los búsquedas y movimientos en función de las configuraciones establecidas al crear la partida. El juego comienza con la ejecución

del método *initGame()*, que a su vez ejecuta la inicialización de variables mediante la función *setupData()* y donde comienza un nuevo de juego estableciendo las posiciones iniciales en el método *nuevoJuego()*. La función *setupData()* inicializa las variables referentes al estado del juego y de la partida, como también conecta la señal que permite detectar cuando el jugador alcanza la moneda, además de crear el mapa de juego y establecer el algoritmo del jugador en caso de que se haya establecido. En cambio, el método *nuevoJuego()* ejecuta el método *initPositions()* donde se inicializa las posiciones del jugador a la posición establecida para ese laberinto, como también la posición de la moneda, por otra parte, en este método se comprueba que el modo de juego sea de tipo *Enfrentamiento*, y genera al personaje del enemigo en el laberinto, para posteriormente inicial el contador de tiempo y ejecutar el método *gameProcess()*, que es el encargado de llevar a cabo el proceso del juego, ejecutando las búsquedas mediante los algoritmos definidos e indicar los movimientos de los personajes en el laberinto.

El método *gameProcess()* crea la heuristica siguiendo el método descrito en la anterior sección, y se divide en varias secciones. La primera sección corresponde al modo de juego *Solitario* y el modo de interacción *Usuario*, donde no realiza ninguna búsqueda, y unicamente ejecuta los movimientos en base a las entradas de teclado introducidas por el usuario:

```
if Videogame.modo_interaccion ==
  VideogameConstants.ModoInteraccion.MODO_USUARIO
and Videogame.modo_juego == VideogameConstants.ModoJuego.MODO_SOLITARIO:
  return
```

Listing 6.18: Proceso del juego referente a la sección con los modos de juego Solitario y Usuario.

La siguiente sección es la referente al modo de juego *Enfrentamiento* y el modo de interacción *Usuario*, donde se realiza la búsqueda del camino unicamente para el enemigo mediante *searchPath()*. Si el algoritmo es *Anchura* o *Profundidad*, se realiza la búsqueda una vez, y mientras el camino encontrado, no haya llegado a la posición objetivo se realiza el método *moveOneStep()*, que indica al enemigo el movimiento a realizar en cada ejecución.

```
elif Videogame.modo_interaccion ==
VideogameConstants.ModoInteraccion.MODO_USUARIO
and Videogame.modo_juego == VideogameConstants.ModoJuego.MODO_ENFRENTAMIENTO:
  await searchPath(algorithm, trayectory, scene)

  while initiate:
    if maze.enemigo.enemy.path.trayectoria.size() > 0:
      await moveOneStep(algorithm)
      Videogame.move_enemy = false
      if Videogame.algoritmo_enemigo == VideogameConstants.Algoritmo.DIJKSTRA
      or Videogame.algoritmo_enemigo == VideogameConstants.Algoritmo.A_STAR:
        var path_enemigo = await newSearch(Videogame.algoritmo_enemigo, ...)
        if !path_enemigo.is_empty():
          await maze.enemigo.setPath(maze.enemigo.position, ...)
    else:
      Videogame.move_enemy = false
      break
```

Listing 6.19: Proceso del juego referente a la sección con los modos de juego Enfrentamiento y Usuario.

Otra sección es la que hace referencia el modo de interacción *Simulación*, cuando el algoritmo configurado para el jugador es *Anchura* o *Profundidad*, siguiendo una dinámica similar a la anterior, realizando la búsqueda unicamente al principio de la interacción, y realiza la búsqueda para el enemigo (si se encuentra en el modo de juego *Enfrentamiento*) también dependiendo del algoritmo definido. Se realiza el bucle de movimiento de jugador, y en su defecto, enemigo hasta que el jugador llega a su posición final del camino creado mediante el algoritmo de búsqueda.

```
elif Videogame.algoritmo_jugador == VideogameConstants.Algoritmo.BFS
 or Videogame.algoritmo_jugador == VideogameConstants.Algoritmo.DFS:
  await searchPath(algorithm, trayectory, scene)

  while initiate:
    if Videogame.modo_juego ==
    VideogameConstants.ModoJuego.MODO_ENFRENTAMIENTO:
      if maze.enemigo.enemy.path.trayectoria.size() > 0:
        await moveOneStep(algorithm)
        var path_enemigo = await newSearch(Videogame.algoritmo_enemigo, ...)
        if !path_enemigo.is_empty():
          await maze.enemigo.setPath(maze.enemigo.position, ...)
      else:
        Videogame.move_enemy = false
        break
    else:
      await moveOneStep(algorithm)
```

Listing 6.20: Proceso del juego referente a la sección con el modo Simulación y los algoritmos DFS y BFS.

La siguiente sección está condicionada por la elección del algoritmo *Anchura* o *Profundidad* para el personaje del enemigo, y el algoritmo *Dijkstra* o *A Estrella* para el personaje del jugador.

En este caso, el enemigo solo realizara una búsqueda del jugador al principio de la ejecución, mientras que el jugador, realiza la búsqueda en cada iteración.

```
elif Videogame.algoritmo_enemigo == VideogameConstants.Algoritmo.BFS
or Videogame.algoritmo_enemigo ==   VideogameConstants.Algoritmo.DFS:
  await searchPath(algorithm, trayectory, scene)
  algorithm.setValueIsPlayer(true)

  while initiate:
    if maze.enemigo.enemy.path.trayectoria.size() > 0:
      await moveOneStep(algorithm)
      var path_jugador = await newSearch(Videogame.algoritmo_jugador, ...)
      if !path_jugador.is_empty():
        await maze.jugador.setPath(maze.jugador.position, ...)
    else:
      Videogame.move_enemy = false
      break
```

Listing 6.21: Proceso del juego referente a la sección con el modo Enfrentamiento y los algoritmos DFS y BFS.

La última sección se ejecuta cuando los algoritmos del jugador y del enemigo son *Dijkstra* o *A Estrella*. En este caso, tanto el jugador como el enemigo realizan la búsqueda de su objetivo mediante el algoritmo establecido en cada iteración del bucle, es decir, después de cada movimiento realizado por ambos, realizan una nueva búsqueda generando un nuevo camino que recorrer.

```
else:
  while initiate:
    await searchPath(algorithm, trayectory, scene)
    await moveOneStep(algorithm)
    createHeuristic(algorithm, heuristic)
```

Listing 6.22: Proceso del juego referente a la sección con los algoritmos Dijkstra y A Estrella.

Por otro lado, una sección de gran importancia en el desarrollo de este proyecto es la dinámica de movimiento de los personajes, que debe realizarse en sincronía y en dependencia entre sí. En primera instancia, el jugador puede realizar los movimientos en el laberinto condicionado por las entradas de teclado del usuario (mediante las flechas del teclado), que indican movimientos derecha, izquierda, arriba y abajo, o por el contrario, su movimiento puede estar condicionado por el camino creado a partir de la búsqueda mediante el algoritmo configurado. La primera opción esta determinada por el método *PlayerController.input()*, que captura el evento del movimiento indicado, y en función de la dirección del movimiento, asigna una posición objetivo al jugador y almacena la posición actual como referencia del nodo inicial en la función *PlayerController.asign_values()*. En cambio, la segunda opción está determinada por el método *PlayerController.desplazarse()*, invocado desde *MazeController.moveOneStep()*, y consiste en obtener, mientras el jugador no está en movimiento, el siguiente nodo de desplazamiento almacenado en el camino recreado por el algoritmo, y almacenar ese nodo como la posición objetivo, como también guardar la posición inicial. Una vez que se ha calculado el nodo objetivo, comienza el desplazamiento siguiendo las físicas propias de Godot, mediante el uso del

método *move_and_collide()*, que ejecuta el movimiento en base a la velocidad y la aceleración especificadas, y comprueba las posibles colisiones de realizar dicho movimiento. En el caso de producirse una colisión, se mantiene la posición inicial, deteniendo el movimiento y emitiendo la señal de desplazamiento finalizado. En cambio, si el movimiento no genera ninguna colisión, y la distancia al objetivo es menor a 1 pixel, se actualiza la posición a la posición objetivo, finalizando el movimiento. A continuación está descrito el proceso de movimiento del jugador:

```
func _process(delta):
  await get_tree().physics_frame
  if player.maze_finished:
    return

  if player.position != player.target and is_moving:
    player.direction = (player.target - player.position).normalized()
    velocity = player.direction * player.speed
    var collision = move_and_collide(velocity * delta)

    if collision:
      actualizaPosicion(player.actual_position)
      emit_signal("movement_finished")
      velocity = Vector2()
    elif position.distance_to(player.target) < 1:
      actualizaPosicion(player.target)
      velocity = Vector2()

    if position == player.target:
      is_moving = false
      emit_signal("movement_finished")
```

Listing 6.23: Proceso de la dinámica del movimiento del jugador hasta la posición objetivo.

La dinámica de movimiento del enemigo guarda multiples similitudes con la del jugador, exceptuando la imposibilidad por parte del enemigo de realizar movimientos condicionados por entradas del teclado por parte del usuario. El enemigo unicamente realiza desplazamientos en base a los caminos creados por el algoritmo configurado, cuya posición objetivo se establece obteniendo el siguiente nodo de dicho camino mediante el método *EnemyController.desplazarse()*, invocado desde la función *MazeController.moveOneStep()*, y que a su vez almacena la posición actual. El movimiento del enemigo también se realiza siguiendo las físicas del motor Godot,y en este caso el método *move_and_collide()* se utiliza para determinar si la colisión producida es ante el personaje del jugador, o ante un muro. En el primer caso, emite la señal *eliminated*, que indica la eliminación del jugador debido al alcance por parte del enemigo, en caso contrario, detiene el movimiento por colisión. A su vez, en caso de no existir colisión, se actualiza la posición cuando la distancia es menor a 1 pixel. A continuación está descrito el proceso de movimiento del enemigo:

```
func _process(delta):
  await get_tree().physics_frame

  if !enemy.maze_finished and Videogame.move_enemy
    and enemy.position != enemy.target:
    enemy.direction = (enemy.target - enemy.position).normalized()
    velocity = enemy.direction * enemy.speed
    var collision = move_and_collide(velocity * delta)

    if collision:
      if collision.get_collider() != null and
        collision.get_collider().name == "Jugador":
        emit_signal("eliminated")
        else:
        actualizaPosicion(enemy.actual_position)
        velocity = Vector2()
      elif position.distance_to(enemy.target) < 1:
        actualizaPosicion(enemy.target)
        velocity = Vector2()

    if position == enemy.target:
      emit_signal("movement_finished")
      Videogame.move_enemy = false
```

Listing 6.24: Proceso de la dinámica del movimiento del enemigo hasta la posición objetivo.

## 6.4. Manual de usuario

En el Anexo 11 se describe brevemente los pasos a seguir para llevar a la experiencia de inicio de sesión, y creación de partidas del videojuego Maze2D.

# Capítulo 7

# Conclusiones y Líneas futuras

En esta sección se analiza brevemente el cumplimiento de los objetivos establecidos inicialmente con el cliente (el tutor), acorde al desarrollo de la aplicación final, así como el ajuste a las planificación temporales establecidas. Además, se tratara de forma breve las implementaciones no completadas en su totalidad, y las lineas futuras de las nuevas propuestas establecidas y surgidas en el transcurso de este trabajo, para versiones futuras del proyecto.

Como conclusiones de este TFG se expone que se ha realizado los objetivos establecidos inicialmente que pretendían cumplir una serie de propósitos que garantizasen la obtención de un resultado acorde a las expectativas iniciales de las partes interesadas del proyecto. Entre estos objetivos propuestos, destaca un objetivo principal y otros cuatro de menor grado de importancia. El principal objetivo estaba enfocado en el desarrollo de una aplicación con carácter educativo que sirva de prototipo para la enseñanza de los diferentes algoritmos de búsqueda materializado mediante el análisis, diseño e implementación de un videojuego desarrollado en Godot Engine combinando simplicidad y la atracción visual para la adquisición del conocimiento de estos algoritmos. Y además, buscando innovar en los métodos de enseñanza ofrecidos en la universidad, promoviendo un enfoque más atractivo y dinámico tanto para la adquisición de conocimientos por parte del alumno como para la enseñanza por parte del profesor. Por otro lado, los objetivos de menor importancia buscaban analizar el rendimiento de los algoritmos de búsqueda, diseñar interfaces de usuario accesibles e interactivas y la modelación de conocimientos teóricos, todos ellos cumpliendo el objetivo de adoptar de la metodología de desarrollo Scrum. Ante estos objetivos planteados, y una vez desarrollado el proyecto, se realiza un análisis del cumplimiento de objetivos y obtener una aproximación del ajuste a los propósitos iniciales. Tras realizar este análisis, determina un ajuste muy aproximado a los propósitos iniciales, gracias al desarrollo completo de la aplicación con carácter educativo y visual requerido por el objetivo principal, como también por el cumplimiento de los requisitos secundarios, para los que se han proporcionado y desarrollado interfaces de usuario destacables por su facilidad de uso y su atracción visual, modelando los cuatro algoritmos de búsqueda propuestos inicialmente y aportando un método de visualización de su lógica. Incluso, de este análisis se ha observado un correcto ajuste, en gran medida, al marco de desarrollo Scrum planteado inicialmente, con leves variaciones de tiempo en ciertas tareas, y ampliación del periodo total de desarrollo.

En contraste, algunos requisitos propuestos inicialmente para el desarrollo de este proyecto, se han visto afectados en la manera que han sido completados parcial o nulamente, por lo que la implementación total de estos requisitos se ha propuesto como trabajo futuro, en nuevas versiones posteriores a la finalización de este proyecto. Simultáneamente, durante el trabajo realizado han surgido nuevas propuestas establecidas, también, como implementaciones realizadas en futuras versiones del proyecto. Tanto las implementaciones requeridas inicialmente no completadas, como las propuestas a futuro más destacables son:

- **Seleccionar apariencias:** Como se ha mencionado, las limitaciones de tiempo y recursos han generado propuestas no completadas acorde a los requisitos iniciales, y una de ellas es la selección de las apariencias, tanto de jugador como enemigo. En la versión de la aplicación desarrollada se ha creado una escena de selección de apariencias, pudiendo observar las multiples existentes para cada personaje, y queda como mejora a futuro la selección e implementación de esta apariencia en el desarrollo del juego.
- **Diferenciar partidas:** Otra propuesta no completada, es la diferenciación en la pantalla de carga de partidas, de las partidas que ya están finalizadas y no se pueden continuar y las que están guardadas y está permitido continuar desde el progreso guardado. En la versión actual, solamente se muestran las partidas no finalizadas.
- **Varios enemigos:** Como método para aumento de la dificultad del juego al jugador, se propone en nuevas versiones, la inclusión de multiples enemigos, dificultando la victoria al jugador, generando la necesidad de aplicación de nuevas estrategias.
- **Seleccionar posición del enemigo:** La selección por parte del usuario de la posición inicial que ocupa el personaje del enemigo en el laberinto, permite explorar el uso y lógica empleado por el algoritmo de búsqueda empleado por el jugador, para encontrar la moneda y evitar el usuario en función de su posición. Por lo que esta mejora, esta planteada en una próxima versión del trabajo implementado.
- **Crear laberintos manualmente:** En consonancia con la anterior propuesta, la elección por parte del usuario de los elementos del laberinto, es decir, los muros y césped, o las posición del jugador o la moneda, permite la exploración de los algoritmos con multiples configuraciones, adecuadas a la necesidad del usuario, ayudando a adquirir el conocimiento de la lógica empleada por cada algoritmo en diferentes situaciones.
- **Añadir algoritmos relevantes:** En adicción a los algoritmos implementados, se plantea en nuevas versiones de la aplicación, el uso de nuevos algoritmos de búsqueda, que permitan aumentar las posibilidades de juego, permitiendo visualizar la lógica empleada por cada uno de estos algoritmos. Algunos de los algoritmos planteados para futuras implementaciones puede ser el algoritmo de Tremaux, o la implementación de algoritmos basados en aprendimiento por refuerzo y la implementación de *Q-Learning* y *Deep-Learning*.

# Capítulo 8

# Anexo I. Modelo de Dominio y Casos de Uso

En este Anexo, se detalla inicialmente el modelo de dominio ampliado del sistema, mostrando las entidades que componen este diagrama y las operaciones propias de cada entidad. También, se describen las secuencias que corresponden a los casos de uso de menor importancia que no han sido descritas en los capítulos anteriores.

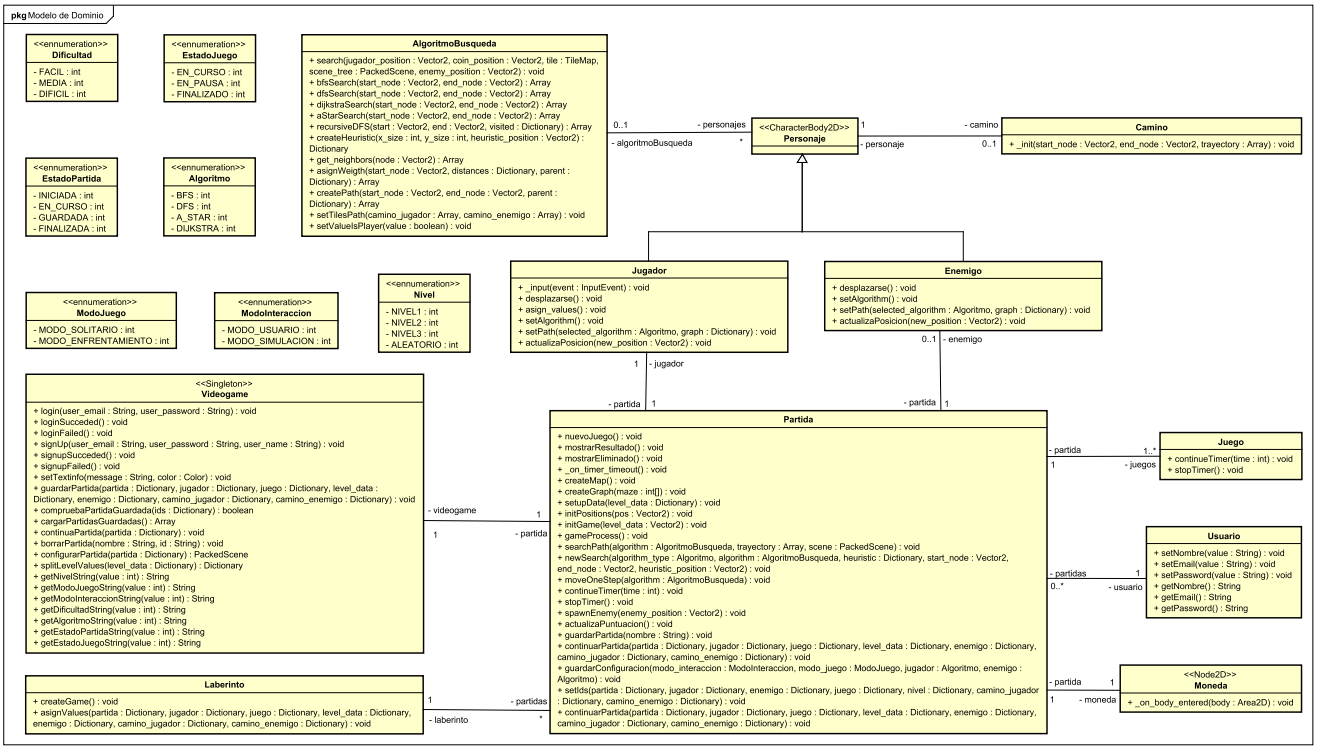


Figura 8.1: Diagrama del modelo de dominio ampliado donde se incluyen las entidades y operaciones del sistema

| **Caso de Uso** | Finalizar Partida |
|---|---|
| **Descripción** | El usuario finaliza la partida pudiendo guardar el progreso obtenido. |
| **Actores** | Usuario |
| **Pre-condición** | El usuario ha iniciado una partida |
| **Escenario Principal** | 1. El usuario solicita la finalización de la partida en curso.<br>2. El sistema solicita confirmación del guardado de la partida.<br>3. El usuario confirma el guardado de la partida.<br>4. Se realiza el caso de uso Guardar Partida.<br>5. El sistema finaliza la partida y vuelve al menú principal.<br>6. Finaliza el caso de uso Finalizar Partida. |
| **Flujos alternativos** | 3.a. El usuario cancela la finalización de la partida.<br>- 1. El sistema anula el guardado y finalización de la partida.<br>- 2. El sistema continua la partida.<br>- 3. Finaliza el caso de uso Finalizar Partida. |

Tabla 8.1: Descripción de la secuencia de acciones del CU Finalizar Partida.

| **Caso de Uso** | Reiniciar Partida |
|---|---|
| **Descripción** | El Usuario reinicia la partida durante su transcurso de la misma eliminando la puntuación y avance obtenido hasta ese momento. |
| **Actores** | Usuario, BaseDatos |
| **Pre-condición** | El usuario ha iniciado una partida. |
| **Post-condición** | La partida no se podrá recuperar tras el reinicio de la misma. |
| **Escenario Principal** | 1. El usuario solicita el reinicio de la partida.<br>2. El sistema solicita confirmación del reinicio de la partida.<br>3. El usuario confirma el reinicio de la partida.<br>4. El sistema muestra la configuración inicial.<br>5. El sistema inicia nueva partida con la configuración inicial.<br>6. Finaliza el caso de uso Reiniciar Partida. |
| **Flujos alternativos** | 3.a. El usuario solicita finalizar el caso de uso.<br>- 1. El sistema cancela el reinicio de la partida.<br>- 2. Finaliza el caso de uso Reiniciar Partida. |

Tabla 8.2: Descripción de la secuencia de acciones del CU Reiniciar Partida.

| **Caso de Uso** | Cambiar Configuracion |
|---|---|
| **Descripción** | El usuario cambia la configuración del entorno de juego. |
| **Actores** | Usuario |
| **Pre-condición** | El usuario ha iniciado una partida. |
| **Escenario Principal** | 1. El usuario solicita cambiar la configuración de la partida creada.<br>2. El sistema solicita al usuario que indique los cambios que desea realizar.<br>3. El usuario indica los cambios que desea realizar.<br>4. El sistema comprueba los cambios indicado por el usuario.<br>5. El sistema almacena la nueva configuración.<br>6. Se realiza el caso de uso Nuevo Juego.<br>7. Finaliza el caso de uso Cambiar Configuracion. |
| **Flujos alternativos** | 3.a. El usuario solicita finalizar el caso de uso.<br>- 1. El sistema cancela la continuación de la partida.<br>- 2. Finaliza el caso de uso Cambiar Configuracion. |

Tabla 8.3: Descripción de la secuencia de acciones del CU Cambiar Configuracion.

| **Caso de Uso** | Buscar Moneda Con Enemigo |
|---|---|
| **Descripción** | El jugador realiza la búsqueda de la moneda mediante el uso del algoritmo definido anteriormente y evitando al enemigo. |
| **Actores** | Jugador |
| **Escenario Principal** | 1.El jugador solicita realizar la búsqueda de la moneda.<br>2.El sistema comprueba el algoritmo de búsqueda definido.<br>3.El sistema busca la moneda aplicando dicho algoritmo evitando la posición del enemigo.<br>4.El sistema almacena el camino obtenido para alcanzar la moneda.<br>5.Finaliza el caso de uso Buscar Moneda. |

Tabla 8.4: Descripción de la secuencia de acciones del CU Buscar Moneda Con Enemigo.

| **Caso de Uso** | Buscar Jugador |
|---|---|
| **Descripción** | El enemigo busca al jugador mediante el algoritmo de búsqueda definido. |
| **Actores** | Enemigo |
| **Pre-condición** | El usuario ha iniciado una partida en Modo Simulación. |
| **Escenario Principal** | 1. El enemigo solicita realizar la búsqueda del jugador.<br>2. El sistema comprueba el algoritmo de búsqueda definido.<br>3. El sistema busca al jugador aplicando el algoritmo indicado.<br>4. El sistema almacena el camino obtenido para alcanzar al jugador.<br>5. Finaliza el caso de uso Buscar Jugador. |

Tabla 8.5: Descripción de la secuencia de acciones del CU Buscar Jugador.

| **Caso de Uso** | Seguir Jugador |
|---|---|
| **Descripción** | El enemigo sigue al jugador mediante uno de los caminos que los conectan. |
| **Actores** | Enemigo |
| **Pre-condición** | Se encuentra en Modo Enfrentamiento. |
| **Escenario Principal** | 1. El enemigo solicita seguir al jugador.<br>2. Se realiza el caso de uso Buscar Jugador.<br>3. El sistema comprueba que existe un camino que conecte con el jugador.<br>4. El sistema obtiene la siguiente posición a la que se desplazará el enemigo.<br>5. Se realiza el caso de uso Desplazarse.<br>6. El sistema comprueba si el enemigo ha alcanzado al jugador.<br>7. El sistema indica que la partida ha finalizado con derrota para el jugador al haber sido alcanzado por el enemigo.<br>8. Finaliza el caso de uso Seguir Jugador. |
| **Flujos alternativos** | 4.a El sistema comprueba que no existe un camino resultado para desplazarse hasta el jugador.<br>- 1. El sistema muestra que no existe un camino resultado para desplazarse hasta el jugador.<br>- 2. Finaliza el caso de uso Seguir Jugador.<br>6.a El sistema comprueba que el enemigo no ha alcanzado al jugador.<br>- 1. El sistema muestra que no ha alcanzado al jugador.<br>- 2. Se continua en el paso 2. Se realiza el caso de uso Buscar Jugador. |

Tabla 8.6: Descripción de la secuencia de acciones del CU Seguir Jugador.

# Capítulo 9

# Anexo II. Diagramas de Secuencia

En este Anexo se describen las secuencias que corresponden a los casos de uso de menor importancia que no han sido descritas en los capítulos anteriores mediante los diagramas de secuencia propios de cada CU especificando las operaciones y entidades que actúan ante cada uno de ellos.

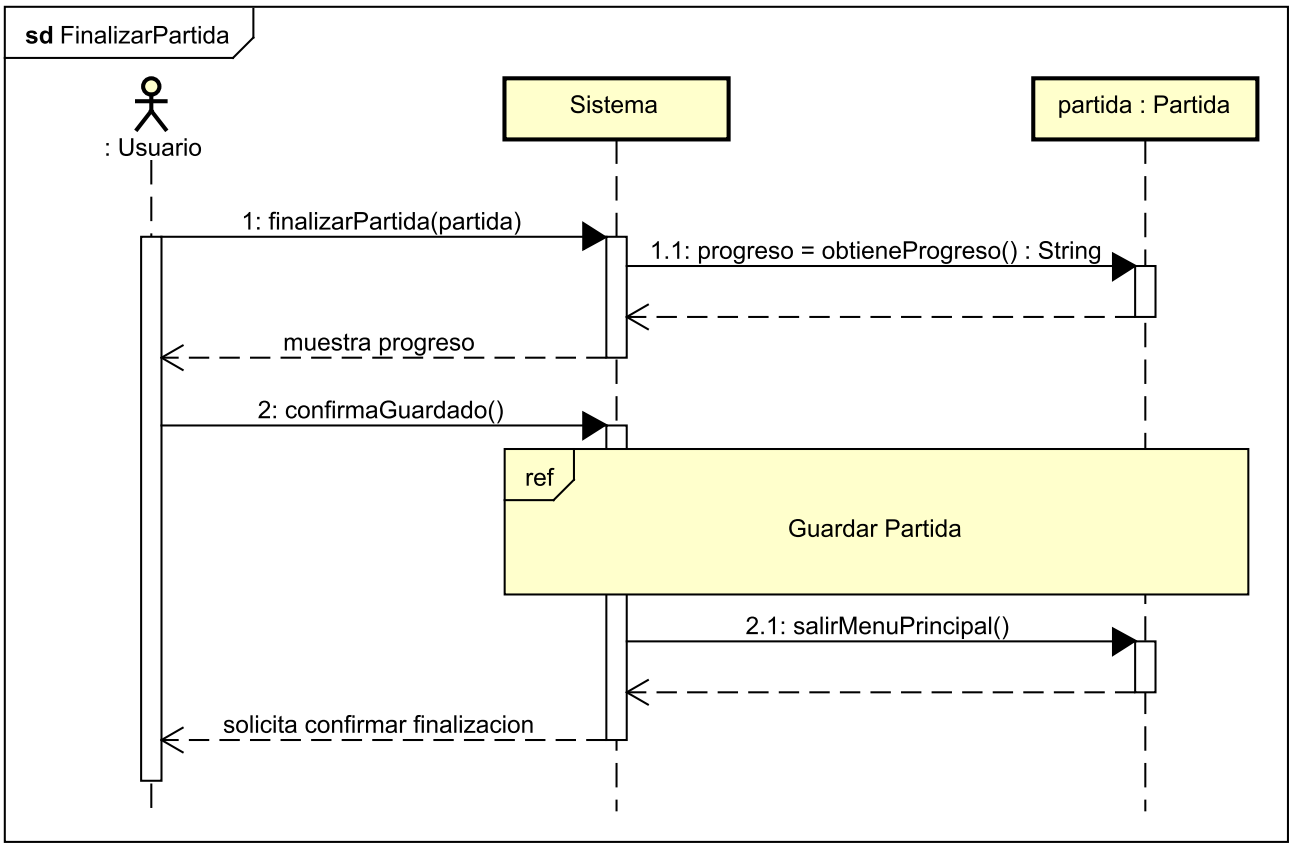


Figura 9.1: Diagrama de secuencia Finalizar Partida.

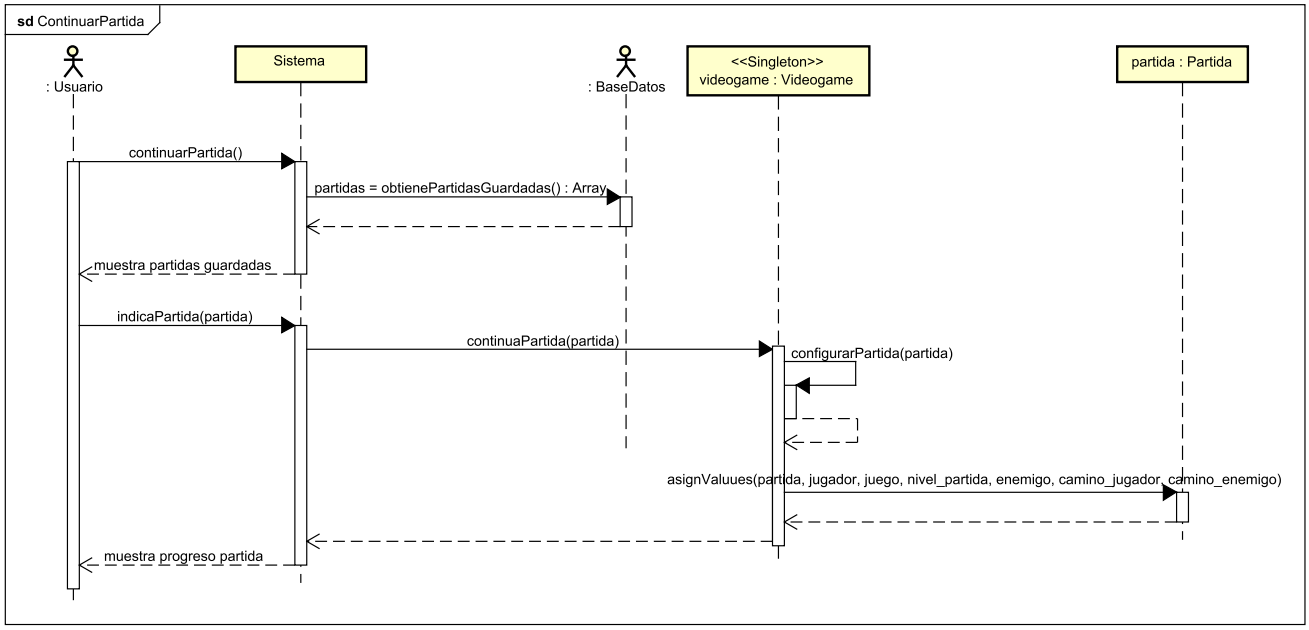


Figura 9.2: Diagrama de secuencia Continuar Partida.

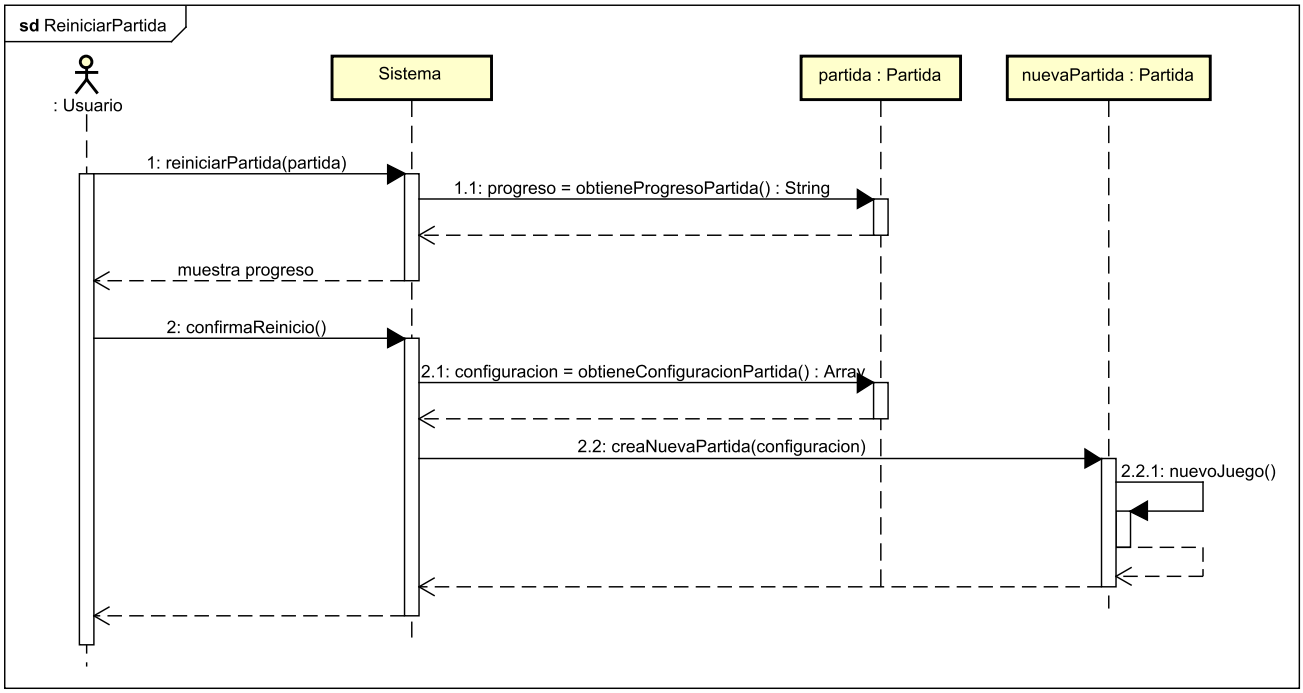


Figura 9.3: Diagrama de secuencia Reiniciar Partida.

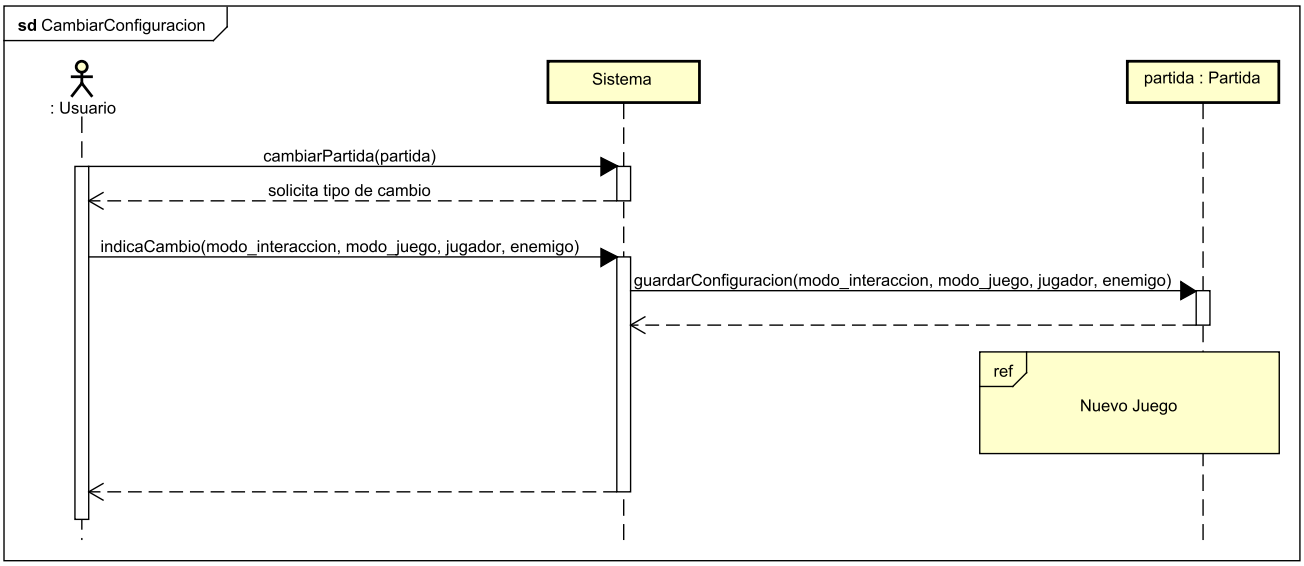


Figura 9.4: Diagrama de secuencia Cambiar Configuracion.

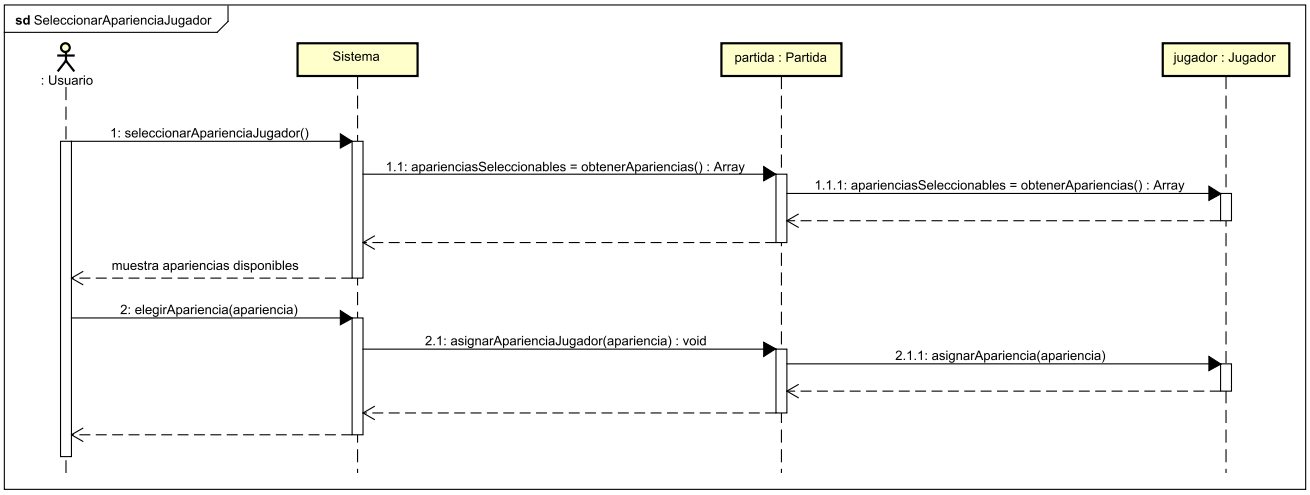


Figura 9.5: Diagrama de secuencia Seleccionar Apariencia Jugador.

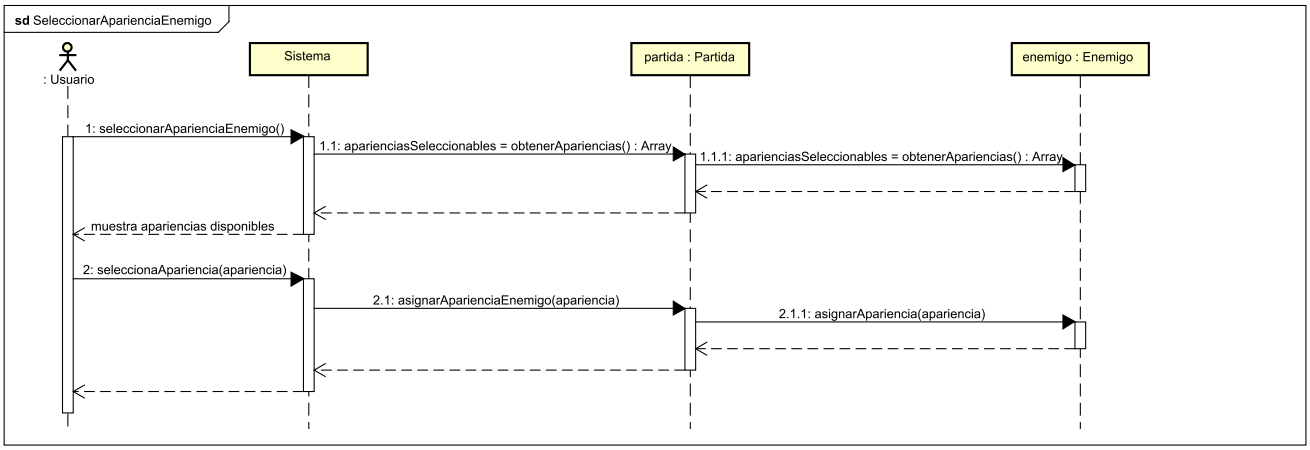


Figura 9.6: Diagrama de secuencia Seleccionar Apariencia Enemigo.

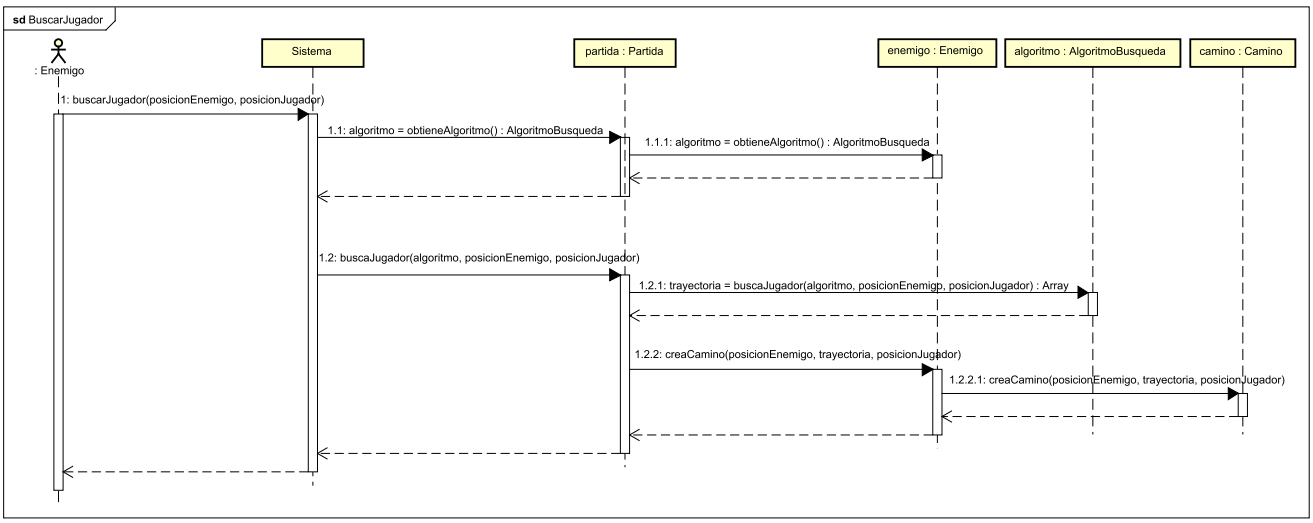


Figura 9.7: Diagrama de secuencia Buscar Jugador.

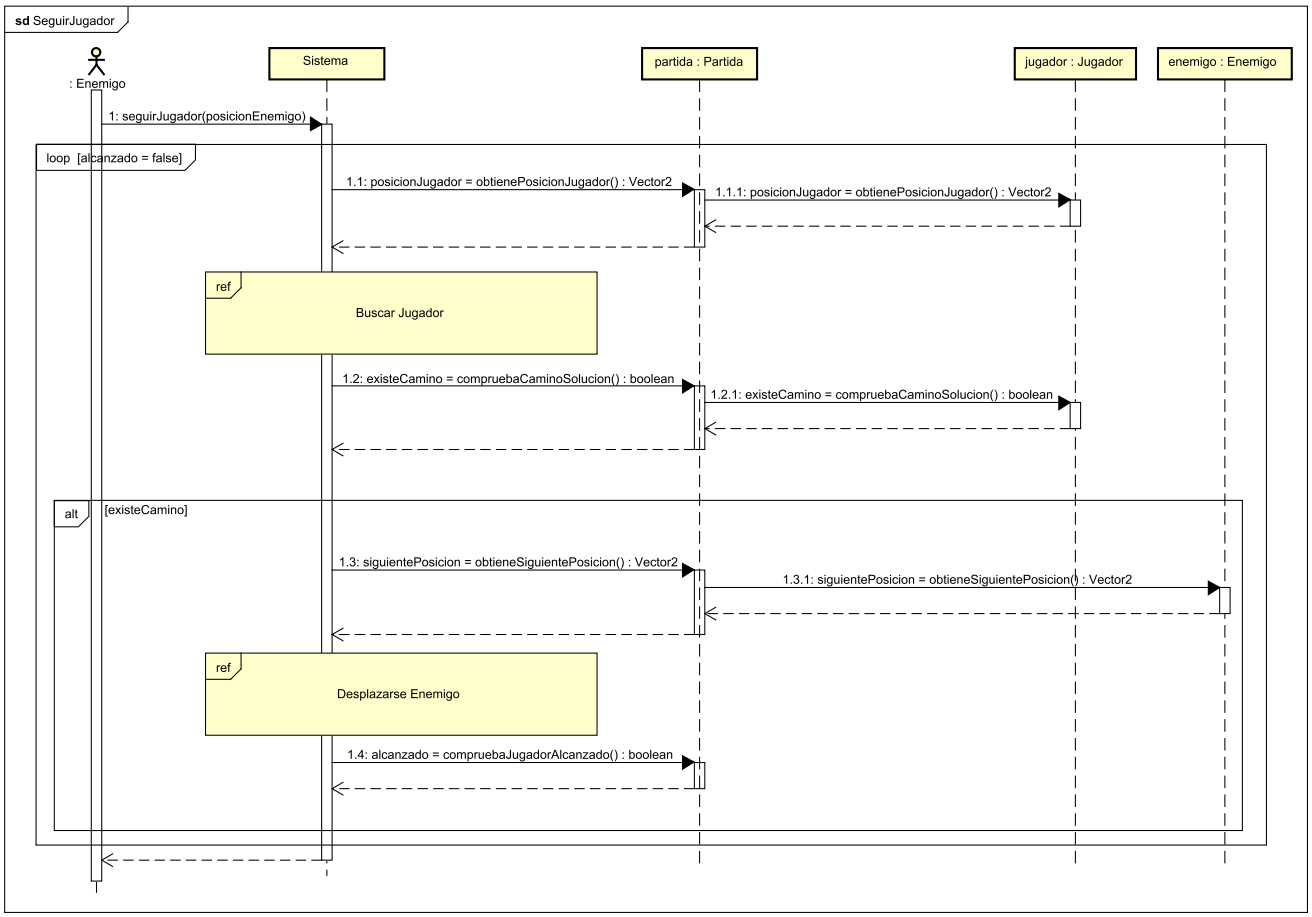


Figura 9.8: Diagrama de secuencia Seguir Jugador.

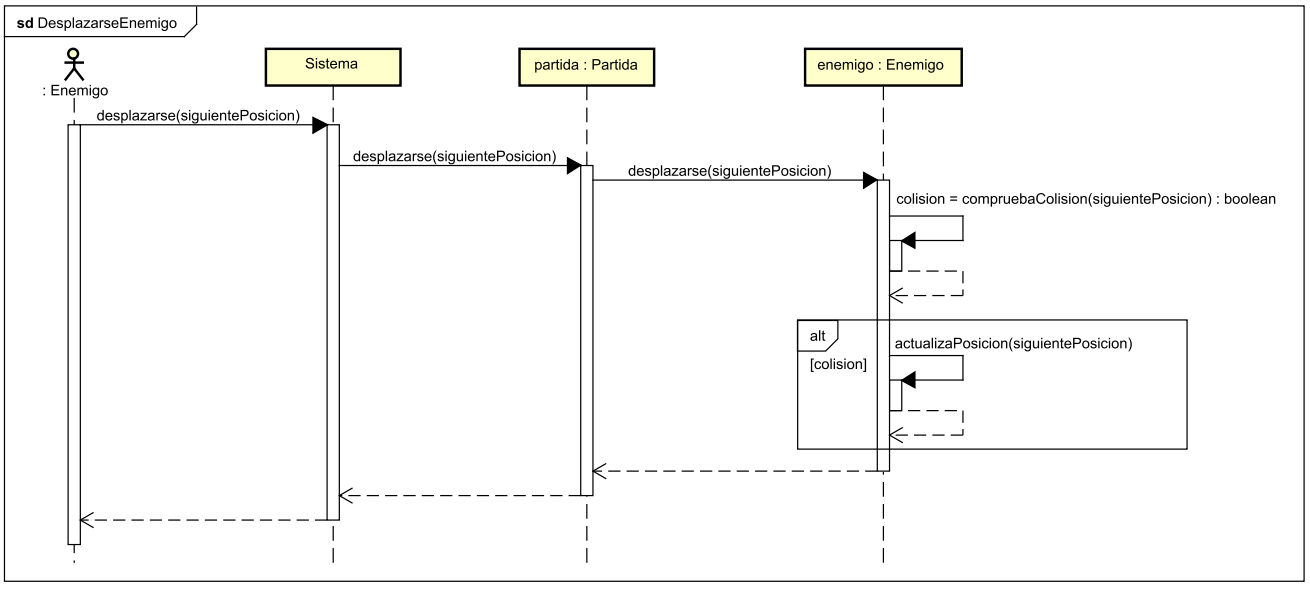


Figura 9.9: Diagrama de secuencia Desplazarse Enemigo.

# Capítulo 10

# Anexo III. Vistas de la aplicación

En este Anexo se incluyen los prototipos creados y mencionados a lo largo de la etapa de diseño. Estas figuras corresponden a cada uno de los prototipos de las interfaces realizadas previamente a la implementación del videojuego.


Figura 10.1: Diseño del prototipo para la interfaz de la pantalla inicial.

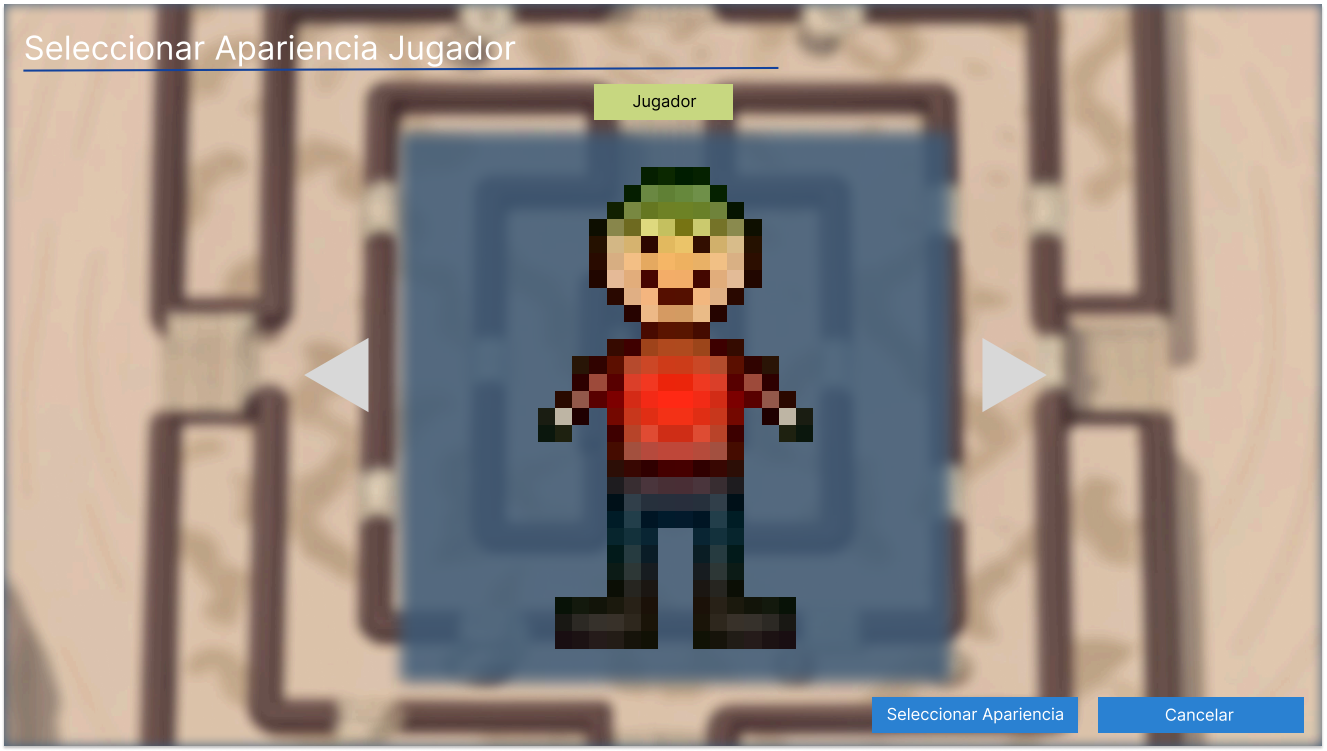


Figura 10.2: Diseño del prototipo para la interfaz de seleccionar apariencia.


Figura 10.3: Diseño del prototipo para la interfaz de continuar partida.

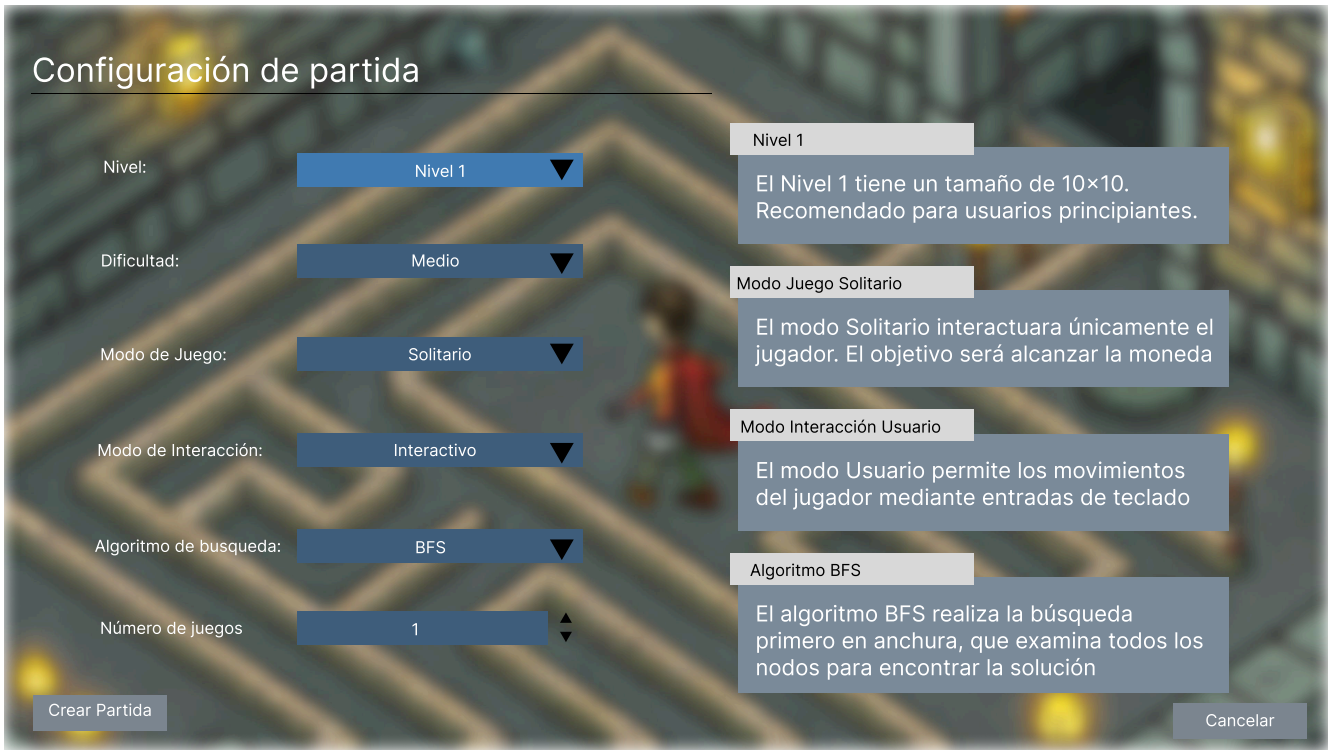


Figura 10.4: Diseño del prototipo para la interfaz de crear partida.


Figura 10.5: Diseño del prototipo para la interfaz del laberinto en el nivel inicial.


Figura 10.6: Diseño del prototipo para la interfaz de las opciones de configuración generales.


Figura 10.7: Diseño del prototipo para la interfaz de las opciones de configuración del juego.

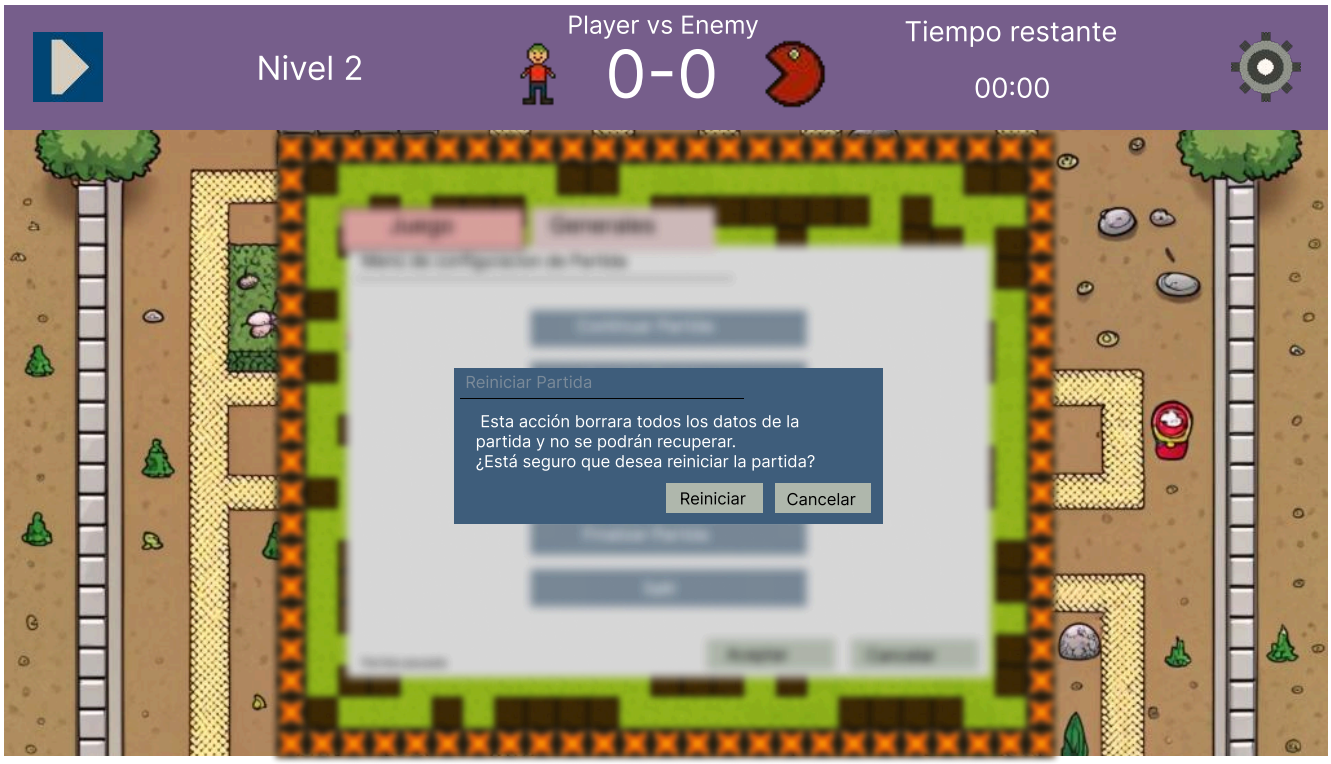


Figura 10.8: Diseño del prototipo para la interfaz de reiniciar partida.


Figura 10.9: Diseño del prototipo para la interfaz de cambiar configuración.


Figura 10.10: Diseño del prototipo para la interfaz de finalizar partida.

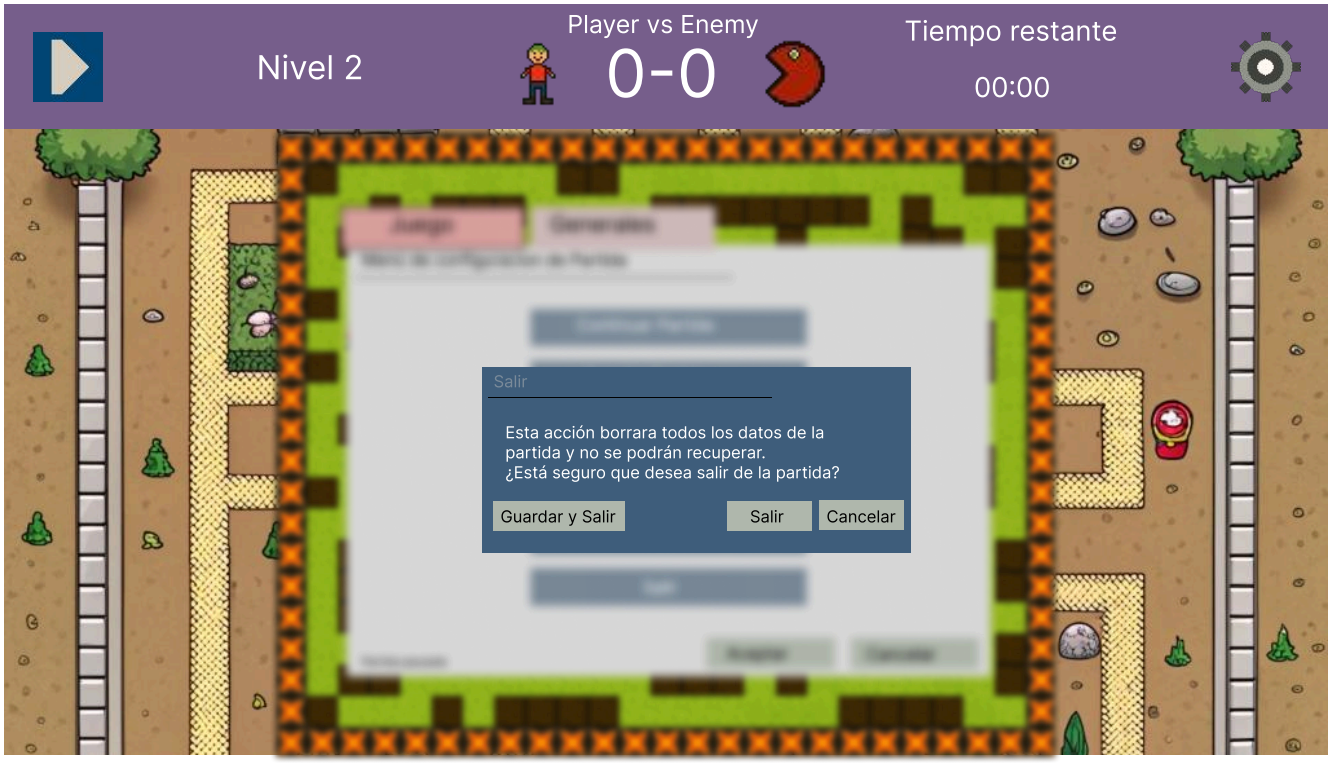


Figura 10.11: Diseño del prototipo para la interfaz de salir al menú principal.

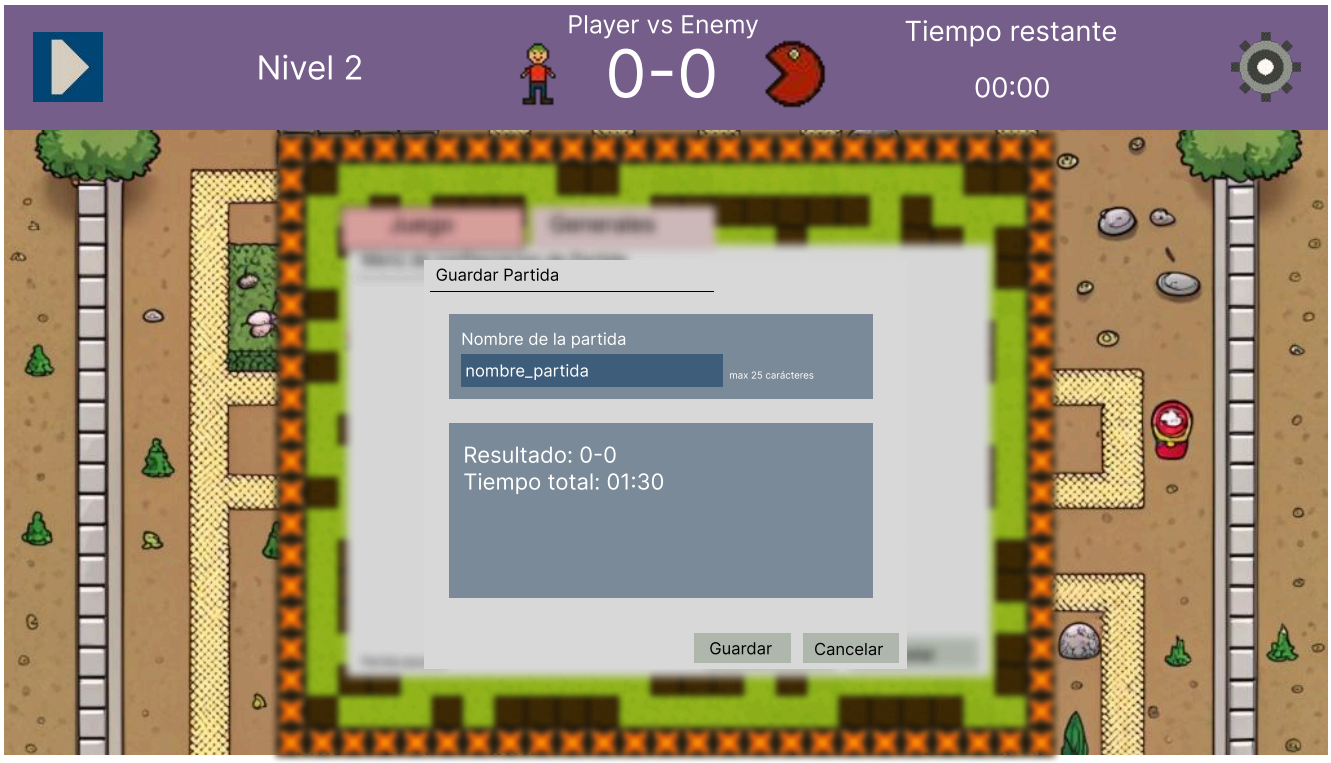


Figura 10.12: Diseño del prototipo para la interfaz de guardar partida.

# Capítulo 11

# Anexo IV. Manual de usuario

El videojuego Maze2D está disponible en el repositorio de GitHub https://github.com/SergioMollo/Maze2D, donde se puede descargar y ejecutar desde la sección Releases para la versión 1.0 https://github.com/SergioMollo/Maze2D/releases, disponible para el sistema operativo *Windows* en la versión de 64 bits. Para su correcto funcionamiento, es necesario tener descargados los archivos .exe y .dll que se encuentran en la carpeta llamada *Maze2D_x64_Windows.zip*, y asegurándose de ubicarlos en la misma carpeta. Además, se recomienda tener instalado el componente más reciente de Microsoft Visual C++ Redistributable desde la pagina oficial https://www.microsoft.com/es-es/download/details.aspx?id=48145, los controladores gráficos actualizados y un sistema operativo basado en 64 bits. Para mayor información consultar el archivo *README.md*.

La primera interfaz de usuario que aparece al ejecutar el videojuego, es la interfaz de inicio de sesión, donde se debe introducir un nombre de usuario y una contraseña registrados en el sistema para proceder a la pantalla principal del videojuego. La primera ejecución del videojuego, requiere de la creación de un nuevo usuario, pues no existe ninguno registrado previamente debido al uso de una base de datos local. Para registrar un usuario, se pulsa el botón *Registrarse*, donde aparece la pantalla de Registro, y se introducen los datos del usuario (nombre, usuario y contraseña), con los que posteriormente se podrá iniciar sesión. Una vez registrado al usuario, redirige, de nuevo, a la pantalla inicial, e introduciendo el usuario y contraseña creados, aparece la pantalla principal del videojuego.

La pantalla principal del videojuego, muestra varias opciones, como son *Crear Partida*, *Cargar Partida*, *Configuración* y *Salir*. Cada una de ellas corresponde a las siguientes acciones:

- **Crear Partida**: Redirige a la pantalla de creación de la partida, donde se configuran y establecen los datos del juego como el tamaño del laberinto, el numero de juegos, o los algoritmos del jugador y del enemigo.
- **Cargar Partida**: Muestra la pantalla de continuación de partida, con la lista de todas las partidas guardadas por el usuario que ha iniciado sesión en el juego anteriormente, permitiendo continuar una partida desde el punto de progreso alcanzado .

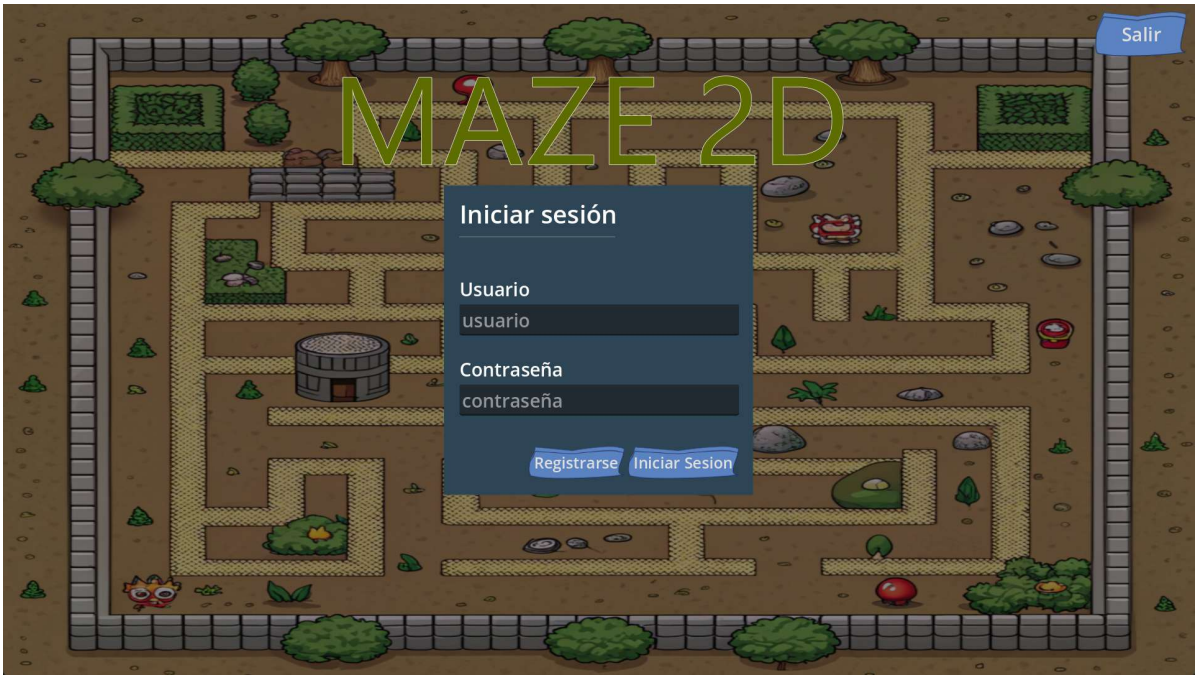


Figura 11.1: Interfaz del videojuego de la pantalla de Inicio de Sesión.

- **Configuración**: Muestra el menú de configuración de resolución de pantalla y volumen de música.
- **Salir**: Cierra completamente el juego.

Por otro lado, en la pantalla principal aparecen dos menús con las interfaces que representan a cada personaje durante el juego (jugador y enemigo), permitiendo acceder a la pantalla de selección de la apariencia de cada uno.

Para iniciar una partida, se pulsa en el botón de *Crear Partida* de la pantalla principal, donde aparecerá las opción de configuración del juego. En esta pantalla se configura inicialmente el tamaño del laberinto, donde se dispone de tres niveles predefinidos, y un nivel aleatorio, cuyo tamaño se encuentra entre 15x15 y 40x40 casillas. Por otro lado, se configura el modo de juego, permitiendo establecer un juego con o sin rival, y en caso de determinar la existencia del enemigo, se establece el algoritmo de búsqueda empleado por el rival, Además, permite elegir entre el modo de iteración habilitado o el modo simulación, donde será controlado por la maquina (generando una simulación de los movimientos), para lo cual se podrá establecer un algoritmo de búsqueda. Por último, se establece el numero de juegos permitido entre 1, 3 o 5.

Tras configurar los datos de la partida, y pulsando el botón de *Crear Partida*, se inicia el juego, mostrando el laberinto junto a los personajes, el tiempo y la puntuación. En este momento, el juego esta listo e iniciado, por lo que el usuario podrá interactuar con el.


Figura 11.2: Interfaz del videojuego de la pantalla de Registro.

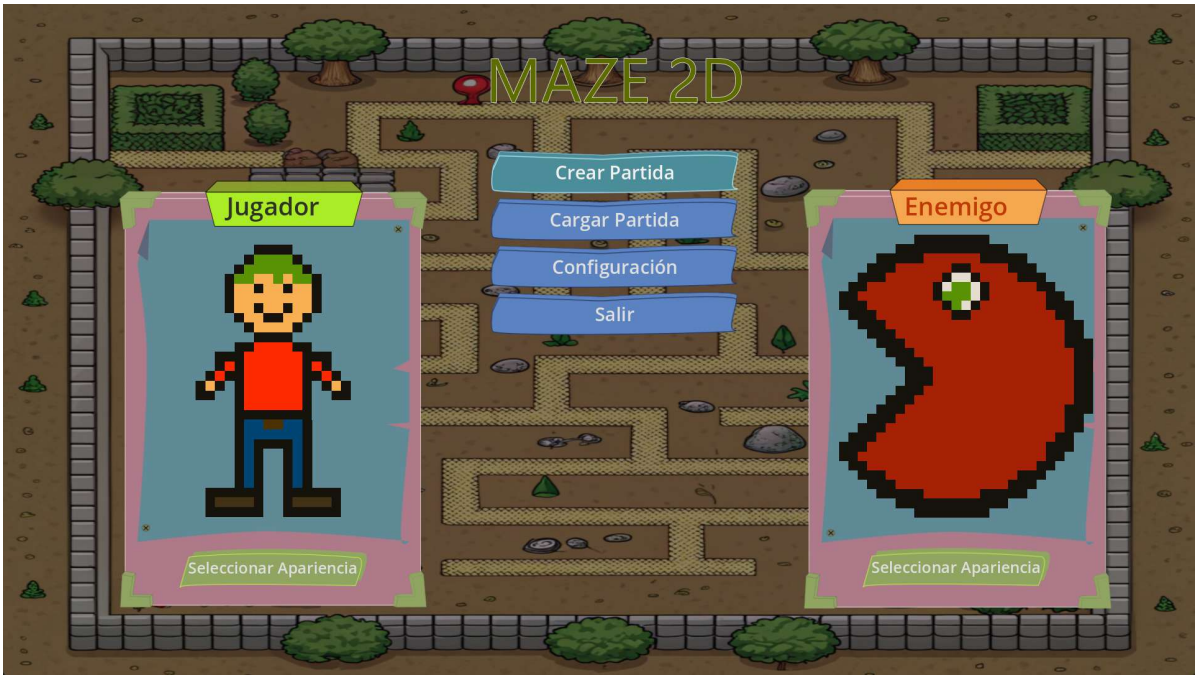


Figura 11.3: Interfaz del videojuego de la pantalla principal.


Figura 11.4: Interfaz del videojuego de la pantalla de Creación de Partida.

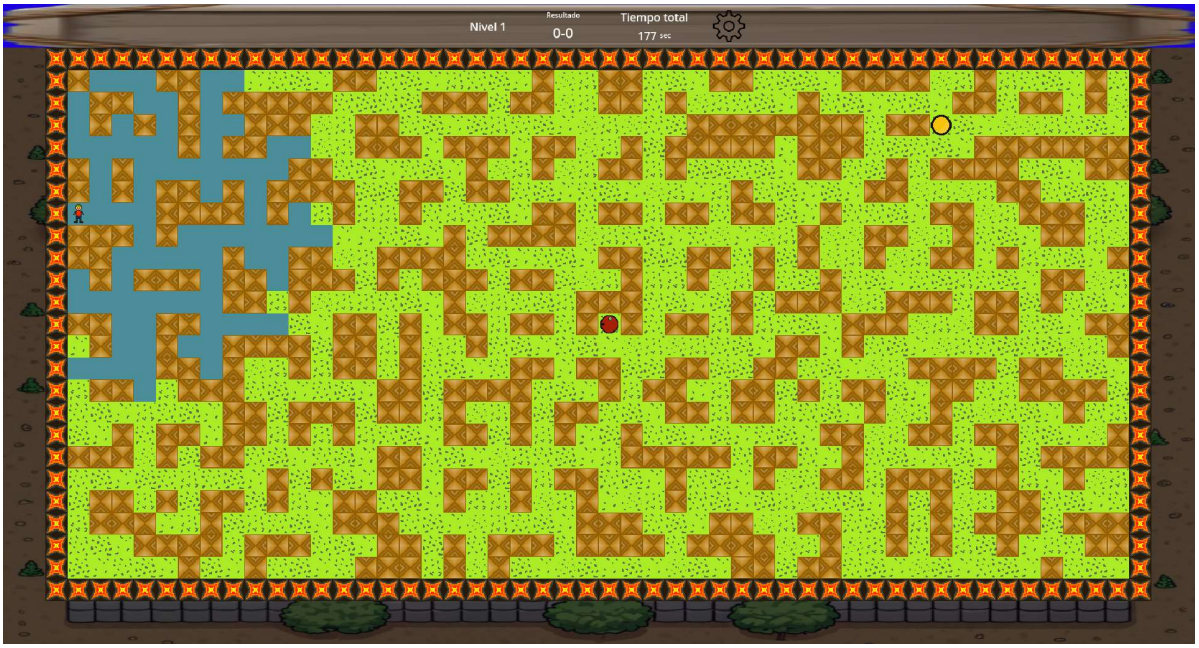


Figura 11.5: Interfaz del videojuego de la pantalla de juego una vez iniciado.

# Bibliografía


[1] Lasse Rouhiainen. Inteligencia artificial. *Madrid: Alienta Editorial*, pages 20–21, 2018.

[2] Pedro Ponce. *Inteligencia artificial: con aplicaciones a la ingeniería*. Alpha Editorial, 2010.

[3] Peter Lee, Carey Goldberg, and Isaac Kohane. *Inteligencia artificial: con aplicaciones a la ingeniería*. Anaya Multimedia, 2024.

[4] Jose Francisco Ávila Tomás, Miguel Angel Mayer-Pujadas, and Victor Julio Quesada-Varela. La inteligencia artificial y sus aplicaciones en medicina ii: importancia actual y aplicaciones prácticas. *Atencion primaria*, 53(1):81–88, 2021.

[5] Pavel Hamet and Johanne Tremblay. Artificial intelligence in medicine. *metabolism*, 69:S36–S40, 2017.

[6] Observatorio Nacional de Tecnología y Sociedad. Indicadores de uso de inteligencia artificial en las empresas española 2023. *ONTSI*, page 43, 2024.

[7] Ral Darío Moreno Padilla. La llegada de la inteligencia artificial a la educación. *Revista de Investigación en Tecnologías de la Información: RITI*, 7(14):260–270, 2019.

[8] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[9] Francisco Herrera. Introducción a los algoritmos metaheurísticos. 2017.

[10] Michael JD Powell. Direct search algorithms for optimization calculations. *Acta numerica*, 7:287–336, 1998.

[11] Oscar Meza and Maruja Ortega. Grafos y algoritmos, 2006.

[12] Scott Beamer, Krste Asanović, and David Patterson. Direction-optimizing breadth-first search. *Scientific Programming*, 21(3-4):137–148, 2013.

[13] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[14] Yong Deng, Yuxin Chen, Yajuan Zhang, and Sankaran Mahadevan. Fuzzy dijkstra algorithm for shortest path problem under uncertain environment. *Applied Soft Computing*, 12(3):1231–1237, 2012.

[15] Gang Tang, Congqiang Tang, Christophe Claramunt, Xiong Hu, and Peipei Zhou. Geometric a-star algorithm: An improved a-star algorithm for agv path planning in a port environment. *IEEE access*, 9:59196–59210, 2021.

[16] Alan Amory, Kevin Naicker, Jacky Vincent, and Claudia Adams. The use of computer games as an educational tool: identification of appropriate game types and game elements. *British Journal of Educational Technology*, 30(4):311–321, 1999.

[17] Jairo Andrés Montes González, Solanlly Ochoa-Angrino, David S Baldeón Padilla, and Mariana Bonilla Sáenz. Videojuegos educativos y pensamiento científico: análisis a partir de los componentes cognitivos, metacognitivos y motivacionales. *Educación y Educadores*, 21(3):388–408, 2018.

[18] Javier Alcalá. Inteligencia artificial en videojuegos. *Ciclo de conferencias Game Spirit*, 2, 2011.

[19] Simone Belli and Cristian López Raventós. Breve historia de los videojuegos. *Athenea Digital. Revista de pensamiento e investigación social*, (14):159–179, 2008.

[20] Y. Lebihan. *Historia de Los Videojuegos: Todo Lo Que Necesitas Saber Desde Sus Inicios Hasta Principios del Siglo XXI*. Look Series. Redbook Ediciones, 2019.

[21] Susan Lammers. *Programmer's At Work*. Microsoft Press, Redmond, WA, 1986.

[22] Tzvi Raz and Erez Michael. Use and benefits of tools for project risk management. *International journal of project management*, 19(1):9–17, 2001.

[23] Ken Schwaber. Scrum development process. In *Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings 16 October 1995, Austin, Texas*, pages 117–134. Springer, 1997.

[24] Martin Fowler, Jim Highsmith, et al. The agile manifesto. *Software development*, 9(8):28–35, 2001.

[25] Craig Larman. Uml y patrones. 2003.

[26] ZE Ferdi Fauzan Putra, Hamidillah Ajie, and Ika Anwar Safitri. Designing a user interface and user experience from piring makanku application by using figma application for teens. *IJISTECH (International Journal of Information System and Technology)*, 5(3):308–315, 2021.

[27] George A Agoston. *Color theory and its application in art and design*, volume 19. Springer, 2013.

[28] José Luis Maravall Llagaria and José Vicente Martán Martánez. Pixel art: Estética de la necesidad o elogio del medio. *Arte y poláticas de identidad*, 12:145–168, 2015.

[29] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson. *Scalable vector graphics (SVG) 1.0 specification*. iuniverse Bloomington, 2000.

[30] Scott Chacon and Ben Straub. *Pro git*. Springer Nature, 2014.

[31] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pages 1277–1286, 2012.

[32] Michael Owens. *The definitive guide to SQLite*. Springer, 2006.

[33] Jim Melton and Alan R Simon. *Understanding the new SQL: a complete guide*. Morgan Kaufmann, 1993.

[34] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7*, pages 406–431. Springer, 1993.

[35] Wei-Yu Chiu, Gary G Yen, and Teng-Kuei Juan. Minimum manhattan distance approach to multiple criteria decision making in multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 20(6):972–985, 2016.