GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GAMIFICACION DE MICROPROCESADORES

Autor: Alejandra de la Rica Escudero

Director: Francisco Martin Martínez

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Gamificación de Microprocesadores

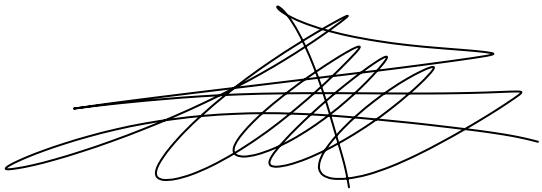
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/23 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.



Fdo.: Alejandra de la Rica Escudero Fecha: 04/06/2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado por MARTIN MARTINEZ
FRANCISCO MARIA - ***7410**
el día 05/06/2023
con un certificado emitido por AC
FNMT Usuarios

Fdo.: Francisco Martín Martínez Fecha: 04/06/2023

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GAMIFICACIÓN DE MICROPROCESADORES

Autor: Alejandra de la Rica Escudero

Director: Francisco Martín Martínez

Madrid

GAMIFICACIÓN DE MICROPROCESADORES

Autor: de la Rica Escudero, Alejandra.
Director: Martín Martínez, Francisco.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto está enfocado en la gamificación[1] de microprocesadores, explorando e implementando como convertir el aprendizaje de conceptos técnicos en un juego interactivo. Mediante el uso de herramientas software , se ha desarrollado un sistema que permite a los estudiantes experimentar de manera practica y entretenida el funcionamiento de los microprocesadores.

Palabras clave: Microprocesadores, Gamificación, Aprendizaje interactivo

1. Introducción

La enseñanza y aprendizaje de conceptos técnicos, como el funcionamiento de los microprocesadores, presenta desafíos en términos de comprensión y motivación para los estudiantes. Tradicionalmente, estos conceptos se abordan de forma teórica, lo que resulta poco atractivo y dificulta el aprendizaje efectivo. En este contexto, aparece la necesidad de explorar nuevas estrategias educativas que involucren a los estudiantes de manera más atractiva.

En respuesta a esto, el presente proyecto se centra en la gamificación de microprocesadores como una forma innovadora de enseñanza. La gamificación consiste en aplicar elementos propios del juego a entornos no lúdicos. En este caso, se pretende transformar el estudio de microprocesadores en una experiencia interactiva y entretenida.

2. Definición del proyecto

El objetivo principal de este proyecto es desarrollar un enfoque de gamificación para el aprendizaje de microprocesadores, ya que es una disciplina técnica y compleja.

El proyecto se basa en la idea de que al convertir el aprendizaje de microprocesadores en una experiencia lúdica e interactiva, los estudiantes pueden adquirir y comprender de manera más efectiva los conceptos y principios fundamentales de esta área.

Para lograrlo, se lleva a cabo un repaso exhaustivo de los temas clave de los microprocesadores identificando los puntos de dificultad y diseñando los desafíos de forma que ayuden a los estudiantes a aprender.

El desarrollo del proyecto implica la creación e implementación de un juego educativo centrado en los microprocesadores. El juego está diseñado de manera que los estudiantes pueden explorar y aplicar los conceptos de microprocesadores en un entorno simulado.

3. Descripción del sistema

El sistema desarrollado es un juego educativo en 2D utilizando el lenguaje de programación Java y el framework LibGDX[2]. El juego, tiene como objetivo ser

interactivo y que el jugador tenga que explorar un mapa, resolver diferentes pruebas y desafíos y que avance en la historia del juego al mismo tiempo que avanza en conceptos claves de los microprocesadores.

El modelo del juego sigue una arquitectura basada en el patrón de diseño Modelo-Vista-Controlador (MVC) [3] mostrado en la Figura 1:

- Modelo:
  En el modelo se implementan las clases y estructuras de datos necesarias para representar los elementos del juego. Estos son, por ejemplo, personajes, pruebas y objetos. El modelo también incluirá la lógica del juego y su ciclo interno de funcionamiento.
- Vista:
  La vista se encarga de la interfaz del juego, es decir los gráficos, animaciones y elementos visuales de este. Se utilizan para ello las capacidades graficas de LibGDX para el renderizado de objetos por pantalla.
- Controlador:
  El controlador gestiona la interacción del modelo y la vista. Responde a las acciones del jugador, como movimientos e interacciones con otros personajes y actualiza el estado del juego en consecuencia. Es decir, escucha y espera a acciones del usuario, comunica estas acciones con el modelo para actualizar el estado del juego, lo que cambia la vista del juego.

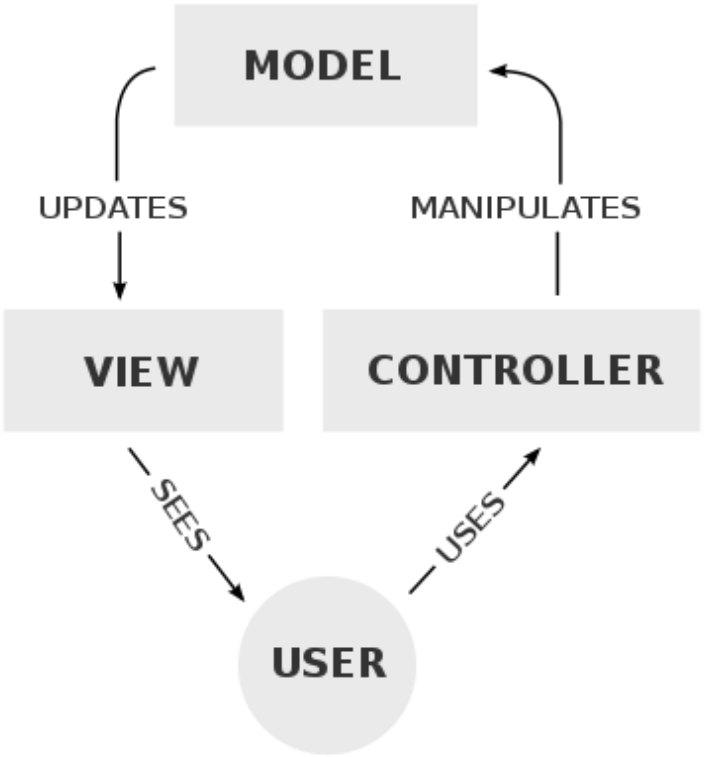


Figura 1: Patrón de diseño Modelo-Vista-Controlador [4]

Además, se utilizan otros patrones de diseño y principios de programación orientada a objetos para mantener un código limpio, modular y fácil de mantener, como la encapsulación, la herencia y el polimorfismo.

4. Resultados

El proyecto ha logrado desarrollar un juego educativo interactivo.

En términos de diseño, se han implementado distintas pantallas como la pantalla de inicio, la pantalla de juego o la pantalla de prueba. Cada una de estas ha sido diseñada de manera atractiva visualmente, utilizando elementos gráficos y de diseño adecuados para el contexto.

En cuanto a la funcionabilidad, se han implementado las funciones necesarias para que el jugador pueda explorar el mapa, interactuar con personajes secundarios, iniciar y completar pruebas, y gestionar su progreso en el juego. Se han creado sistemas de manejo de vidas, control de respuestas correctas e incorrectas en las pruebas, y guardado de progreso de juego para permitir a los jugadores retomar partidas guardadas.

En términos de accesibilidad, se ha tenido en cuenta la facilidad de uso y comprensión para el jugador. Se ha diseñado una interfaz de usuario clara y amigable para el usuario.

La jugabilidad del juego ha sido cuidadosamente diseñada para proporcionar desafíos y entretenimiento al jugador. Se han creado pruebas que requieren habilidades cognitivas y conocimiento del funcionamiento de microprocesadores, brindando una experiencia desafiante pero gratificante. El juego también ofrece una progresión de dificultad a medida que el jugador avanza, manteniendo su interés y motivación.

En cuanto a los resultados obtenidos, se ha logrado crear un juego completamente funcional y jugable, que cumple con los objetivos establecidos en el proyecto.

5. Conclusiones

En conclusión, el desarrollo de este juego educativo basado en la gamificación de microprocesadores ha demostrado ser exitoso en términos de diseño, funcionalidad, accesibilidad y jugabilidad. El juego ofrece una experiencia de aprendizaje interactiva y entretenida, que contribuye a mejorar la comprensión de los conceptos técnicos de microprocesadores por parte de los estudiantes.

Este proyecto sienta las bases para futuras investigaciones y mejoras en el ámbito de la gamificación educativa y el uso de juegos interactivos como herramientas de aprendizaje.

6. Referencias

[1] Educativa. "Gamificación: el aprendizaje divertido". https://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/. Visitado en: 25/11/2022

[2] LibGDX. "LibGDX". Fecha no especificada. https://libgdx.com/. Visitado en: 03/11/2022

[3] Junta de Andalucía. "Guía de buenas prácticas para el uso seguro y responsable de las tecnologías de la información y la comunicación (TIC). https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122. Visitado en: 19/04/2023

[4] Wikipedia. "Archivo:MVC-Process.svg". https://es.wikipedia.org/wiki/Archivo:MVC-Process.svg. Visitado en: 20/05/2023

MICROPROCESSOR GAMIFICATION

Author: de la Rica Escudero, Alejandra.
Supervisor: Martín Martínez, Francisco.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project focuses on the gamification 6 of microprocessors, exploring and implementing how to turn the learning of technical concepts into an interactive game. Through the use of software tools, a system has been developed that allows students to experience the functioning of microprocessors in a practical and entertaining way.

Keywords: Microprocessors, Gamification, Interactive Learning

1. Introduction

Teaching and learning technical concepts, such as the functioning of microprocessors, present challenges in terms of understanding and motivation for students. Traditionally, these concepts are approached in a theoretical manner, which is unappealing and hinders effective learning. In this context, there is a need to explore new educational strategies that engage students in a more attractive way.

In response to this, the present project focuses on the gamification of microprocessors as an innovative teaching approach. Gamification involves applying game elements to non-game environments. In this case, the aim is to transform the study of microprocessors into an interactive and entertaining experience.

2. Project definition

The main objective of this project is to develop a gamification approach for learning microprocessors, as it is a technical and complex discipline.

The project is based on the idea that by turning the learning of microprocessors into a playful and interactive experience, students can acquire and understand the fundamental concepts and principles of this area more effectively.

To achieve this, a comprehensive review of key microprocessor topics is carried out, identifying points of difficulty, and designing challenges that help students learn.

The project development involves the creation and implementation of an educational game centered around microprocessors. The game is designed in such a way that students can explore and apply microprocessor concepts in a simulated environment.

3. System description

The developed system is a 2D educational game using the Java programming language and the LibGDX framework [2]. The game aims to be interactive, requiring the player

to explore a map, solve different tests and challenges, and progress in the game's story while advancing key microprocessor concepts.

The game model follows an architecture based on the Model-View-Controller (MVC) design pattern [3] shown in Figure 2:

- Model:
  The model implements the necessary classes and data structures to represent the elements of the game. These include characters, tests, and objects. The model also includes the game logic and its internal functioning cycle.
- View:
  The view is responsible for the game's interface, including graphics, animations, and visual elements. The graphical capabilities of LibGDX are used for object rendering on the screen.
- Controller:
  The controller manages the interaction between the model and the view. It responds to player actions, such as movements and interactions with other characters, and updates the game's state accordingly. In other words, it listens and waits for user actions, communicates these actions to the model to update the game state, which changes the game's view.

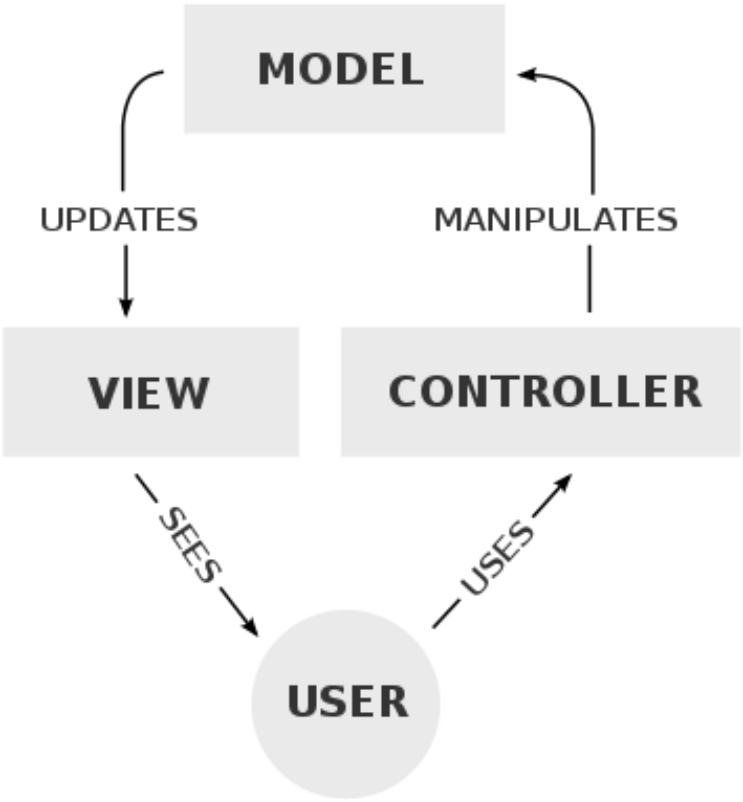


Figure 2: Model-View-Controller design pattern [4]

In addition, other design patterns and object-oriented programming principles are used to maintain clean, modular, and easy-to-maintain code, such as encapsulation, inheritance, and polymorphism.

4. Results

The project has successfully developed an interactive educational game. In terms of design, different screens have been implemented, such as the start screen, game screen, and test screen. Each of these screens has been visually designed attractively, using appropriate graphic and design elements for the context.

In terms of functionality, the necessary functions have been implemented to allow the player to explore the map, interact with secondary characters, initiate, and complete tests, and manage their progress in the game. Systems for managing lives, tracking correct and incorrect answers in tests, and saving game progress have been created to allow players to resume saved games.

In terms of accessibility, ease of use and understanding for the player have been considered. A clear and user-friendly interface has been designed.

The gameplay of the game has been carefully designed to provide challenges and entertainment to the player. Tests that require cognitive skills and knowledge of microprocessor functioning have been created, providing a challenging yet rewarding experience. The game also offers a progression of difficulty as the player advances, maintaining their interest and motivation.

Regarding the results obtained, a fully functional and playable game has been created, meeting the objectives set in the project.

5. Conclusions

In conclusion, the development of this educational game based on the gamification of microprocessors has been successful in terms of design, functionality, accessibility, and gameplay. The game offers an interactive and entertaining learning experience, contributing to improving students' understanding of microprocessor technical concepts.

This project lays the groundwork for future research and improvements in the field of educational gamification and the use of interactive games as learning tools.

6. References

[1] Educativa. "Gamificación: el aprendizaje divertido". https://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/. Visited on: 25/11/2022

[2] LibGDX. "LibGDX". Fecha no especificada. https://libgdx.com/. Visited on: 03/11/2022

[3] Junta de Andalucía. "Guía de buenas prácticas para el uso seguro y responsable de las tecnologías de la información y la comunicación (TIC).

https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122. Visited on: 19/04/2023

[4] Wikipedia. "Archivo:MVC-Process.svg". https://es.wikipedia.org/wiki/Archivo:MVC-Process.svg. Visited on: 20/05/2023

Índice de la memoria

Índice de figuras

Índice de tablas

1 INTRODUCCIÓN

La tecnología cada día está más presente en nuestras vidas. En nuestro día a día acudimos a ella para gestiones que hace años se hacían de otras formas como puede ser una reserva en un restaurante, la compra de una entrada para un concierto, e incluso transacciones monetarias.

Ahora me gustaría centrarme en dos partes de nuestras vidas en las que la tecnología está presente: el entretenimiento y la educación.

El entretenimiento y ocio en el siglo XXI es cada vez más tecnológico: videojuegos, aplicaciones de *streaming* de películas y series y descarga de música, entre otras muchas actividades, están ahora a nuestro alcance gracias a la evolución tecnológica.

Además, la tecnología está integrándose cada vez más en los colegios y universidades para complementarla y en muchos sentidos simplificarla. Hoy en día la mayoría de los colegios proporcionan *tablets* a los alumnos, existen clases con ordenadores integrados y los proyectores son un básico en nuestras aulas. Vemos entonces como en los últimos años y en parte gracias a la pandemia, la educación ha evolucionado tecnológicamente en múltiples sentidos. Esto ha permitido la aparición de aulas virtuales, plataformas de reuniones telemáticas, y diversas formas de aprendizaje como exámenes online.

Con todo esto llegamos a la gamificación[1], esta es una rama de la educación que se basa en el desarrollo de juegos y videojuegos en ámbitos no comunes como la formación educativa o profesional. De esta forma se fomenta la motivación y concentración de los estudiantes, consiguiendo así que se esfuercen todavía más y adquieran todos los conocimientos de forma fácil y efectiva, mejorando el rendimiento y los resultados académicos, entre otros aspectos.

1.1 MOTIVACIÓN

La motivación detrás del proyecto es fomentar la gamificación en el contexto educativo para mejorar la experiencia de aprendizaje de los estudiantes de la asignatura de microprocesadores.

En la actualidad, el uso de la gamificación en la educación superior sigue siendo limitado, a pesar de los múltiples beneficios que esta técnica puede ofrecer. Es posible que esto se deba a la falta de conocimiento sobre cómo aplicarla en el aula, a la falta de recursos para implementarla o incluso a la resistencia al cambio. Sin embargo, la gamificación puede ser una herramienta valiosa para mejorar el aprendizaje y la participación de los estudiantes universitarios.

Por otro lado, el uso de la gamificación en la educación superior puede marcar una diferencia significativa entre las universidades, permitiendo ofrecer una experiencia educativa más atractiva e innovadora. Además, la implementación de esta técnica puede mejorar la retención y la tasa de graduación de los estudiantes, lo que se traduce en una ventaja competitiva para las universidades que la adopten. Por lo tanto, es importante que las universidades consideren la implementación de la gamificación como una herramienta valiosa para mejorar la experiencia educativa de sus estudiantes y destacarse en el sector educativo.

Por ello se propone diseñar un juego que siga una línea temporal que refleje el calendario real de la asignatura de Microprocesadores para tratar los conceptos y temas de la asignatura. De esta manera, los estudiantes podrán aprender y repasar los conceptos mientras juegan, lo que resultaría en un aprendizaje más entretenido y atractivo.

1.2 RECURSOS EMPLEADOS

En este proyecto la mayoría de las herramientas necesarias son de software, ya que de hardware solo se requiere un ordenador con el que programar. Además, las herramientas de software utilizadas son:

- Java: Lenguaje de programación orientado a objetos y multiplataforma utilizado en desarrollo de software.
- IntelliJ IDEA Ultimate: IDE para programar el proyecto entero.
- LibGDX: *Game Engine*, un conjunto de herramientas de software diseñadas para ayudar en la creación y desarrollo de videojuegos para ayudarnos con la programación del juego y mapa.
- PixilArt: para el diseño de los personajes y los componentes del mapa.
- Tiled: para el diseño y creación del mapa.
- GitHub: para el control de versiones que permite gestionar y mantener un historial de cambios en proyectos del software.

2 DESCRIPCIÓN DE LAS TECNOLOGÍAS SOFTWARE

2.1 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

La programación orientada a objetos[3] (POO) es un paradigma de programación que se basa en el uso de objetos para desarrollar aplicaciones.

La POO ofrece una serie de ventajas[4] en el desarrollo de software, entre las que destacan las siguientes:

- La programación orientada a objetos se asemeja al mundo real, lo que hace que sea más fácil para los programadores entender y representar los problemas y soluciones que se están abordando.
- Favorece la reutilización y ampliación del código, lo que permite ahorrar tiempo y esfuerzo en el desarrollo de nuevas aplicaciones.
- Permite crear sistemas más complejos y escalables, ya que los objetos pueden ser combinados para formar estructuras más grandes y sofisticadas.
- Agiliza el desarrollo de software, ya que los objetos pueden ser creados y modificados con mayor facilidad.
- Facilita el trabajo en equipo, ya que los objetos pueden ser desarrollados y gestionados de manera independiente por diferentes programadores.

Resumidamente, la programación orientada a objetos proporciona una serie de conceptos y herramientas para representar el mundo real, lo que resulta en un desarrollo de software más eficiente y efectivo.

Para comprender el modelo de programación orientada a objetos es importante conocer dos conceptos básicos: clases y objetos.

Las clases son modelos o plantillas que definen las variables y métodos comunes a todos los objetos de cierto tipo. En otras palabras, son prototipos genéricos que permiten crear múltiples objetos con características similares. Por ejemplo, en el mundo real podemos tener muchos objetos del mismo tipo, como teléfonos móviles, que tienen características y

comportamientos comunes (marca, modelo, sistema operativo, hacer y recibir llamadas, enviar mensajes, etc.).

Los objetos son instancias de una clase, es decir, son los resultados de aplicar la plantilla de una clase para crear un objeto con características y comportamientos específicos.

Al mismo tiempo, este enfoque se fundamenta en cuatro principios básicos[5]: herencia, polimorfismo, abstracción y encapsulación.

2.1.1 HERENCIA

La herencia es un concepto fundamental que permite que una clase adquiera las variables y métodos de otra clase (conocida como la superclase o clase padre). Esto significa que una subclase, además de sus propias características y métodos, también incluye las características y métodos heredados de la superclase, creando así una jerarquía de herencia.

2.1.2 POLIMORFISMO

En la Programación Orientada a Objetos, el polimorfismo es un concepto clave que permite que un mismo identificador pueda tener varias formas diferentes. Este puede aplicarse a funciones, métodos, variables u objetos, y se refiere a la capacidad de adaptarse a diferentes situaciones y comportarse de manera diferente en función del contexto en que se utilice.

La finalidad del polimorfismo es implementar el envío de mensajes entre objetos, donde estos interactúan entre sí mediante llamadas a distintas funciones. De esta manera, se permite una mayor flexibilidad y adaptabilidad en el diseño del software, ya que los objetos pueden responder de manera distinta según la situación en la que se encuentren.

2.1.3 ABSTRACCIÓN

Este principio permite simplificar los conceptos y comportamiento más complejos mediante la creación de modelos o plantillas abstractas simplificadas. Los modelos abstractos se centran en los aspectos más relevantes del componente y eliminan los detalles confusos o

complicados. De esta manera se ayuda a mejorar la comprensión y mantenibilidad del programa.

2.1.4 ENCAPSULACIÓN

La encapsulación es un principio de la programación orientada a objetos que consiste en ocultar la complejidad interna de los objetos y exponer solo las interfaces necesarias para interactuar con ellos. En este sentido, se declara la visibilidad de los atributos de clase como *private* para que no puedan ser accedidos directamente desde fuera de la clase, lo que aumenta la seguridad y la integridad del objeto.

Para acceder a los valores de las variables de clase que han sido declaradas como *private*, se pueden crear métodos públicos de acceso conocidos como *getters* y *setters*. Los *getters* permiten obtener el valor de una variable, mientras que los *setters* permiten asignar un nuevo valor a la misma. De esta forma, se asegura que cualquier cambio en el estado interno del objeto se haga a través de sus métodos públicos, lo que permite un mayor control sobre su comportamiento y evita errores y conflictos en el código.

2.2 LIBGDX

LibGDX[6] es un *framework* de desarrollo de videojuegos de código abierto que utiliza el lenguaje de programación Java. Proporciona una amplia gama de herramientas y funciones para desarrollar juegos multiplataforma, lo que significa que se puede crear un juego una vez y luego distribuirlo en múltiples plataformas, como Android, iOS, PC, Mac y navegadores web. Es una herramienta muy popular en la industria de los videojuegos y se utiliza para crear juegos en 2D y 3D.

Una de las principales ventajas de utilizar LibGDX es que proporciona una serie de módulos y herramientas que permiten desarrollar juegos de manera eficiente y rápida. Por ejemplo, incluye soporte para gráficos, sonido, entrada de usuario, física, animación, entre otros.

Además, LibGDX es altamente personalizable y flexible, lo que significa que se puede adaptar la biblioteca a unas necesidades específicas. Por ejemplo, se puede trabajar con diferentes plataformas de manera sencilla y existe la libertad de elegir la herramienta de diseño y el motor de física que mejor se adapte al proyecto.

2.3 TILED

Tiled[8] es una herramienta de software libre y multiplataforma diseñada para ayudar en la creación de mapas y niveles para videojuegos. Se utiliza para crear y editar mapas de forma gráfica, colocar objetos y definir propiedades de estos, así como definir áreas de colisión y otros detalles importantes en el diseño de niveles para videojuegos.

Tiled admite diferentes tipos de mapas, como mapas ortogonales (2D) y mapas isométricos (3D), y puede exportar mapas a diferentes formatos de archivo, como XML, JSON y CSV, lo que lo hace muy versátil y fácil de integrar con diferentes motores de juego.

3 ESTADO DE LA CUESTIÓN

Tal y como hemos visto, basar el aprendizaje en juegos lo convierte en un proceso más entretenido y fácil, alejando al aprendizaje de lo tradicional al mismo tiempo que se consiguen resultados extraordinarios.

Sin embargo, los beneficios no se quedan aquí, y es que la gamificación es la manera perfecta de fortalecer la comunicación, colaboración, creatividad y pensamiento crítico. Asimismo permite a los estudiantes tener la libertad de equivocarse sin que haya consecuencias negativas, lo que se relaciona directamente con el desarrollo de una mayor tolerancia a la frustración y al fracaso, convirtiéndose así en personas resilientes y luchadoras.

Hoy en día ya existen millones de plataformas y aplicaciones basadas en la gamificación que permiten el aprendizaje de forma didáctica.

3.1 EJEMPLOS DE GAMIFICACIÓN EN LA EDUCACIÓN

Aplicar la gamificación en la educación tiene multitud de ventajas para su rendimiento, y lo mejor es que puede aplicarse en cualquier tipo de área de conocimiento o asignatura.

3.1.1 APRENDIZAJE PUNTUAL

Una de las plataformas más conocidas de gamificación es Kahoot![10], una herramienta muy útil para profesores y estudiantes para aprender y repasar conceptos de forma entretenida, como si fuera un concurso. Esta es una plataforma de tipo Trivial, que recompensa a quienes progresan en las respuestas con una mayor puntuación que les catapulta a lo más alto del ranking. Además Kahoot!, te da la posibilidad de competir de forma individual, o por equipos.

La Figura 3 representa un ejemplo de una pantalla de Kahoot!

Figura 3: Pantalla de Kahoot! [11]

La idea es la misma que hemos oído hablar tantas veces: aprender divirtiéndote. Sin embargo, es más una herramienta de refuerzo, pues la naturaleza de las preguntas es demasiado corta como para entrar en demasiado detalle. Esto la convierte en una plataforma de aprendizaje puntual.

Con características parecidas encontramos Socrative[12], una herramienta similar al Kahoot!, pero más formal. Esta herramienta ofrece a los alumnos cuestionarios online, de tipo test, que proporciona al alumnado y al instructor un *feedback* en tiempo real de las respuestas de los alumnos. Socrative, aun siendo un ejemplo de gamificación, no tiene el elemento del juego tan integrado como el Kahoot!, por lo que para los estudiantes es menos llamativo para el alumnado.

La Figura 4 muestra un ejemplo de lo que seria un cuestionario de Socrative.

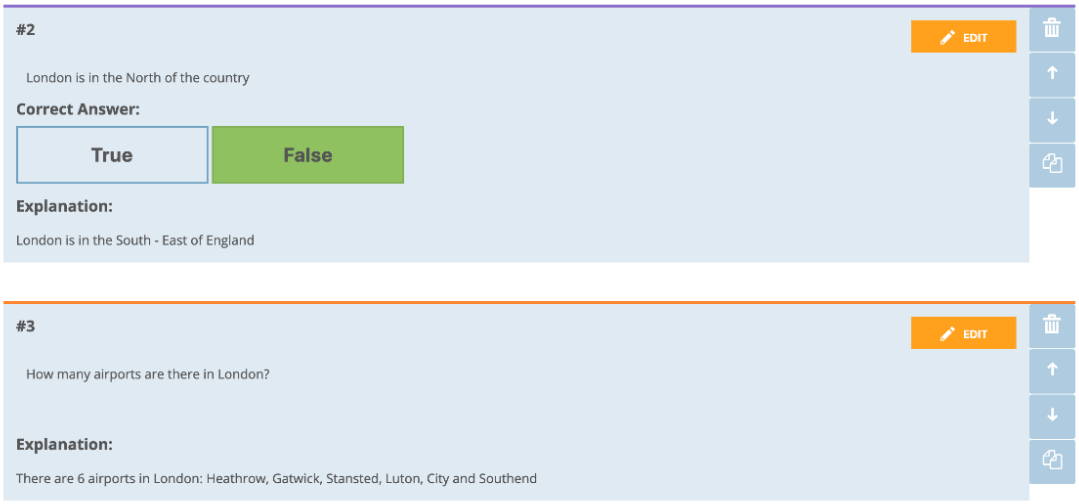


Figura 4: Pantalla de Socrative [13]

3.1.2 APRENDIZAJE PROGRESIVO Y CONTINUADO

En esta sección destaca iCuadernos[14], una herramienta desarrollada por la editorial Rubio, dirigida a niños de entre 3 y 12 años. Basada en un sistema intuitivo y fácil de manejar. Esta aplicación les ayuda a mejorar sus conocimientos de manera divertida y sin ayuda de los mayores: practicarán las Matemáticas, se iniciarán en la escritura y el reconocimiento de las letras mayúsculas y minúsculas, y encontrarán una extensa variedad de actividades para desarrollar conceptos básicos, lo cual les ayudará en su desarrollo intelectual y motriz.

La idea es que la evolución sea progresiva y continuada, cada niño irá avanzando según sus capacidades y una vez superado un nivel empiece el siguiente. El objetivo es que el niño use la aplicación habitualmente y no como algo puntual. Esta aplicación de aprendizaje tiene multitud de tipos de pruebas, pero se basa en replicar el uso de los tradicionales cuadernillos Rubio, por lo que el niño ira dibujando en el dispositivo las respuestas a las pruebas.

Un ejemplo de la herramienta puede observarse en la Figura 5.

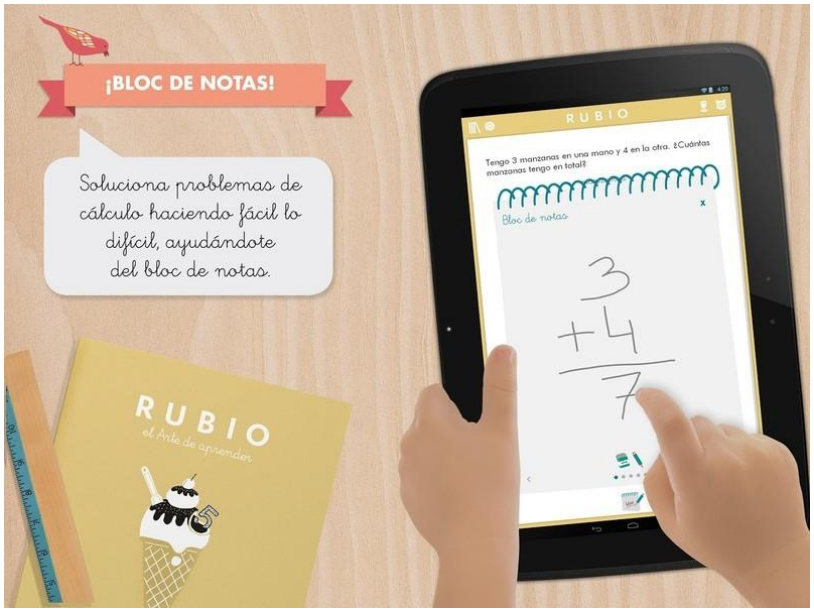


Figura 5: Ejemplo de iCuadernos [15]

Otro ejemplo de este tipo de juego es ClassCraft[16], un juego RPG. Los juegos de RPG (Role-Playing Game) son lo más parecido a videojuegos que encontramos en la gamificación, ya que en estos, el jugador, tiene un personaje con el que se identifica y controla.

Con ClassCraft, el aula se convierte en un escenario en el que, de forma completamente virtual, los alumnos juegan a ser Guerreros, Magos y Curanderos. Conforme van avanzando las clases, ellos van adquiriendo más poderes y facultades ganados a partir de su desempeño dentro del aula (participación, entrega de tareas, realización de trabajos, etcétera).

Este juego se basa en la cooperación y la participación ya que el juego se jugará en equipos y con esto la participación aumenta y la colaboración con sus compañeros también, ya que se van ayudando a no caer en batalla o a incrementar su nivel.

Se puede ver una pantalla del juego en la Figura 6.

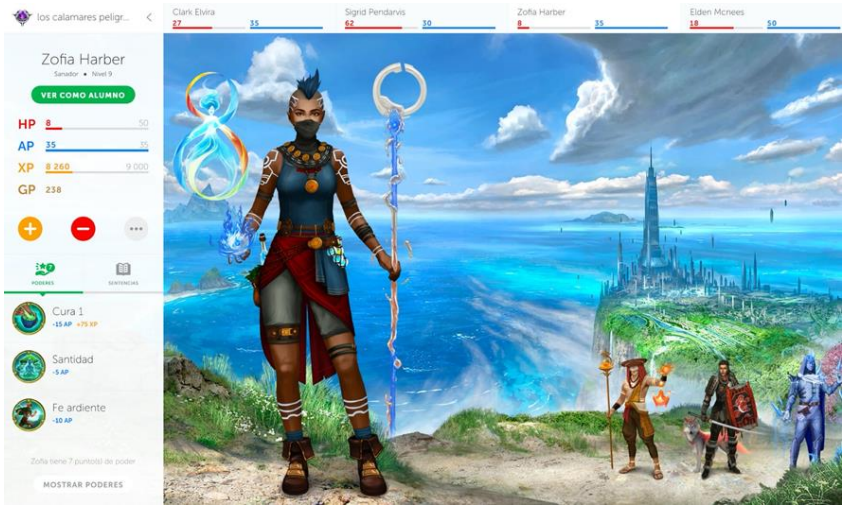


Figura 6: Pantalla de ClassCraft [16]

Por último, como otro ejemplo RPG, aunque de forma explícita, están las presentaciones de tipo *Escape Room* que pueden crearse en Canva, Prezi, Google Forms o Google Slides. Con presentaciones, se pueden crear salas de escape colaborativas e interactivas para ayudar a los estudiantes a completar un nuevo concepto o revisarlo.

En estas, el estudiante debe asumir el rol de detective encerrado para escapar. Para desbloquear cada cerradura y abrir cada puerta, será necesario utilizar el ingenio y los conocimientos de aquellas áreas que se vayan a estudiar en numerosas pruebas que llevaran a escapar.

3.2 EJEMPLOS DE GAMIFICACIÓN EN LA PROGRAMACIÓN

La gamificación es una gran herramienta para que los alumnos aprendan, sin embargo uno de los usos de la gamificación más importantes de mencionar para este proyecto, es el aprendizaje de la programación. La gamificación, es una herramienta fantástica para que los alumnos aprendan programación e incluso robótica de forma más amena y natural.

Hay muchas ventajas que derivan de la aplicación de la gamificación en el proceso de aprendizaje de la programación. Y es que hay varias habilidades que se adquieren casi sin

esfuerzo gracias a esta metodología, que son imprescindibles para un programador, como por ejemplo el pensamiento computacional.

La mayoría de los juegos gamificados de programación se basan en un aprendizaje continuado, por lo que no dividiremos los ejemplos en esas secciones.

Un ejemplo claro de juego RPG gamificado es CodeMonkey[22].

CodeMonkey es un entorno divertido y educativo basado en juegos dirigidos a niños, con los que aprenden a codificar sin ninguna experiencia previa. La dinámica que propone CodeMonkey es simple y entretenida, ya que se basa en un juego con una serie de retos que los niños tienen que superar. En cada prueba, el niño tiene que guiar al mono para que siempre obtenga sus plátanos, y para ello habrá que superar desafíos mientras se van mostrando conceptos básicos de programación.

Cada prueba consta de una pantalla dividida en dos partes, una que muestra al mono y los plátanos, y otra en la que hay un cuadro de texto. Usando ambas pantallas, el jugador debe guiar al mono hacia los plátanos, escribiendo en el cuadro de texto lo que el mono debe hacer (girar a la derecha, andar x pasos, etcétera). Esto puede observarse en la Figura 7 que presenta un ejemplo de una pantalla del juego. Por tanto, el juego necesita de un compilador detrás que compruebe el código y mueva al mono en función del texto escrito.

Figura 7: Pantalla de CodeMonkey [23]

Prácticamente igual a este, pero con un público distinto, esta CodeCombat[17], ya que este está dirigido a personas más mayores e idealmente con alguna base en programación.

CodeCombat es un juego de rol online en el cual se pueden aprender los fundamentos de lenguajes de programación basados en texto tan populares como Python y JavaScript (entre otros). En este juego, hay una progresión enorme de principio a fin ya que a medida que el jugador avanza, se sale de la monotonía del principio y el jugador avanza por desiertos y bosques a través de los que obtiene atajos al desarrollo web, fortaleciendo a los jugadores con las habilidades en HTML y scripting. Otros planos enseñan lo esencial sobre desarrollo de juegos, permitiendo crear niveles propios. Al igual que CodeMonkey, este necesita de un compilador para su correcto funcionamiento.

Se puede observar una pantalla de ejemplo de CodeCombat en la Figura 8.

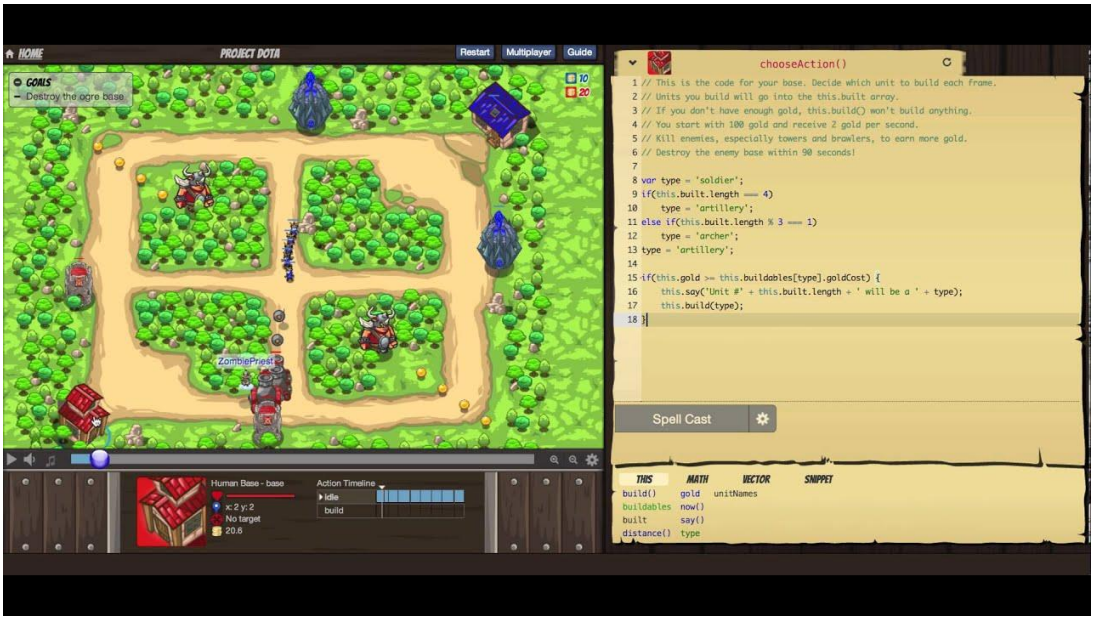


Figura 8: Pantalla de CodeCombat [18]

Otro ejemplo, distinto a los anteriores, es while True: learn()[19]. Este es un juego de simulación y puzles que trata algo todavía más enigmático: el aprendizaje automático, las redes neuronales, los macrodatos y la IA. Pero, lo más importante, es que trata de intentar entender a tu gato.

En el juego eres un programador que, sin quererlo, descubre que a su gato se le da extremadamente bien la programación, pero no es capaz de hablar el lenguaje de los humanos. Como programador que eres, debes aprender cómo funciona el aprendizaje automático y usar la programación visual para crear un sistema de reconocimiento de voz gato-humano.

En este juego no se escribe código, sino que con las piezas que el juego te ofrece, creas un árbol/red que poco a poco ira creando el sistema ya comentado. La Figura 9 muestra un ejemplo de la pantalla de este juego.

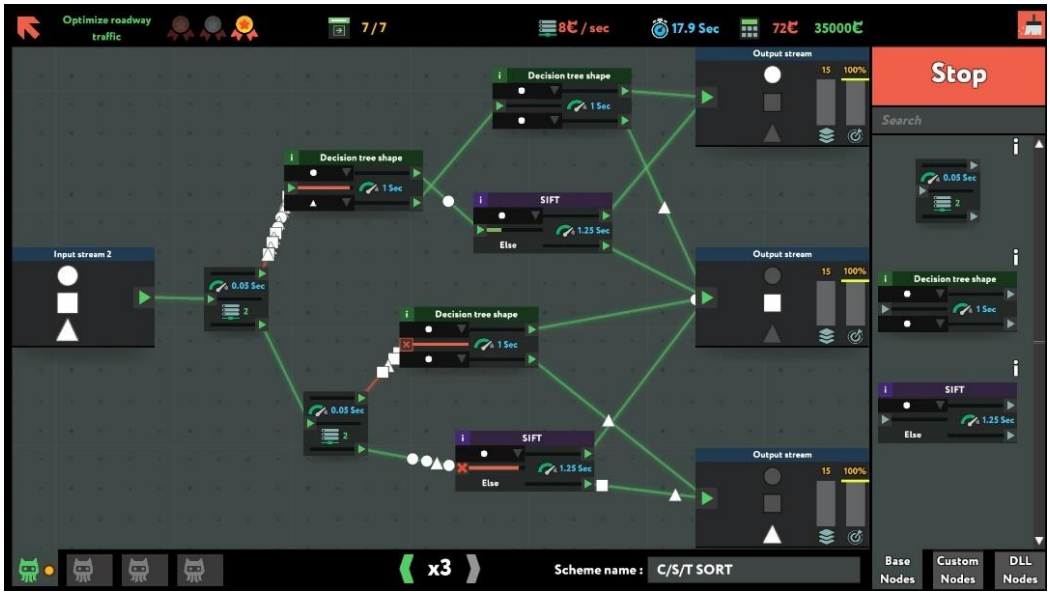


Figura 9: Pantalla de While true: learn() [19]Comparación final

Con todos estos ejemplos de la educación gamificada se presenta una comparación final (Tabla 1) de estos, añadiendo nuestro proyecto para identificar todas las diferencias y similitudes entre estos:

JUEGO	**APRENDIZAJE CONTINUADO**	**RPG**	**TIPO**	**MAPA**	**PUBLICO/TIPO DE JUGADOR**	**MODOS DE JUEGO**
Kahoot!	✗	✗	Trivial	✗	Todos los públicos/ Ambiciosos	Individual y equipos
Socrative	✗	✗	Test	✗	Todos los públicos/ Triunfadores	Individual
iCuadernos	✓	✗	---	✗	Niños/ Triunfadores	Individual

ClassCraft	✓	✓	---	✓	Niños y adolescentes/ Ambiciosos	Equipos
EscapeRoom	✗	✗	Escape Room	✗	Todos los públicos/ Exploradores	Equipos
Codemonkey	✓	✓	Escribir Texto	✓	Niños/ Triunfadores	Individual
CodeCombat	✓	✓	Escribir Texto	✓	Adolescentes y adultos/ Triunfadores	Individual

While true: learn()	✓	✓	Puzle/ Drag and Drop	X	Adolescentes y adultos/ Triunfadores	Individual
Gamificación de microprocesa-dores	✓	✓	Trivial, test y puzle	✓	Adolescentes y adultos/ Triunfadores	Individual

Tabla 1: Comparación final del estado de la cuestión

En este proyecto, el juego tiene un personaje (por lo que será un juego RPG) al que el jugador controla por un mapa basado en la planta de laboratorios de ICAI. El jugador tiene que explorar el mapa para buscar a otros personajes por el mapa los cuales le presentan las pruebas que tiene que resolver para poder desbloquear nuevas pruebas y nuevos personajes.

El objetivo es que el aprendizaje sea progresivo y continuado durante el curso escolar y de forma individual para que cada alumno sea consciente de su nivel en la asignatura de Microprocesadores.

Las pruebas serán de tipo Trivial (preguntas con huecos en blanco que deberá rellenar el estudiante), de tipo test (con cuatro opciones), o de tipo puzle (drag and drop/drop down).

Este estará destinado principalmente a estudiantes de la universidad de ICAI cursando la asignatura de Microprocesadores, pero también pueden hacer uso del adolescentes y adultos fuera del sistema educativo con interés en aprender de microprocesadores.

4 DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

La realización de este proyecto es esencial para cubrir una falta de uso de la gamificación en las universidades y estudios superiores.

Además, el juego propuesto sigue una línea temporal que refleja el calendario real de la asignatura, lo que permite fortalecer los conocimientos de los estudiantes de manera progresiva y continuada durante el curso escolar. El juego también ofrece diferentes tipos de pruebas para evaluar los conocimientos de los estudiantes, lo que lo hace atractivo para el mercado educativo y para aquellos interesados en aprender sobre microprocesadores fuera del sistema educativo.

El proyecto ofrece una solución innovadora y atractiva para mejorar el aprendizaje y el compromiso de los estudiantes de Microprocesadores.

4.2 OBJETIVOS

El objetivo del proyecto es el desarrollo y la implantación de un juego basado en la gamificación, que permita y complemente el aprendizaje del área de los microprocesadores alineándose temporalmente con el calendario de la asignatura de Microprocesadores impartida en ICAI, de manera que éste pueda ser utilizado por los estudiantes para reforzar sus conocimientos. Por tanto un listado de los objetivos del proyecto seria:

- Diseño del juego para hacerlo llamativo y entretenido para los estudiantes a la vez que aprenden.
- Reforzar con las pruebas todo el contenido de la asignatura de Microprocesadores.
- Alinear las pruebas con el calendario de la asignatura para obtener un aprendizaje progresivo y continuado.
- Implementar el juego para que pueda ejecutarse desde cualquier PC, sin causar problemas.

- Obtener un juego completamente funcional, para poder subirlo a la plataforma de ICAI y que los alumnos le den un uso real.

4.3 METODOLOGÍA

El proyecto está compuesto de tres partes principales:

- El diseño del GUI (*Graphical User Interface*) del juego, como son los personajes, el mapa, y el formato de cada prueba. Esto incluye la programación básica del mapa y los personajes.
- El diseño de las pruebas del juego (que temas se van a preguntar y de qué manera, cuantas pruebas va a haber). Al mismo tiempo que se diseña se irán programando las pruebas.
- La programación e implementación del videojuego al completo integrando ambas partes anteriores.

Para todo esto se hará uso del lenguaje de programación Java, usando Git para la utilización de Control de versiones.

El juego a desarrollar está basado en el Pokémon Esmeralda[21] de la *GameBoy Advance*, por lo que el GUI es muy parecido. El personaje aparece en una esquina del mapa y puede moverse por él, buscando a los distintos personajes que están en este.

Para que las pruebas se realicen en orden del temario de la asignatura, solo se puede realizar la prueba de tal personaje si se ha resuelto la anterior correctamente. En caso de no haberla superado, todos los personajes menos este o los de temas anteriores, te mandan a la prueba necesaria, consiguiendo así un orden estricto de aprendizaje.

Finalmente, el jugador podrá presentarse al examen final, en el que se enfrentará a el Boss final, con una prueba completa de la asignatura. En caso de haber superado todas las pruebas con éxito, el jugador gana y termina el juego.

5 DISEÑO DEL JUEGO

5.1 FUNCIONAMIENTO DEL JUEGO

De cara al jugador y en vistas generales el funcionamiento del juego sería el siguiente, el cual está representado visualmente en la Figura 10:

El jugador inicia el juego y se le muestra la pantalla de inicio. Si existe una partida ya guardada el jugador puede ir a la pantalla de juego directamente; de lo contrario el jugador pasa por una pantalla de selección de personaje y se crea una nueva partida llevándolo a la pantalla de juego.

Una vez en la pantalla de juego, el jugador debe explorar el mapa para encontrar al personaje asociado a la siguiente prueba correspondiente. Puede hablar con los personajes que encuentre en el mapa, los cuales le proporcionan direcciones para encontrar la prueba.

Una vez encuentra al personaje, este le presenta la prueba al jugador, quien debe superarla sin fallos. Si no lo logra, pierde una vida, y puede volver a intentar la prueba hasta quedarse sin vidas, lo que le llevaría a perder el juego (*Game Over*). Si supera la prueba, y aún quedan pruebas por resolver, el jugador repite el ciclo de búsqueda hasta completar la totalidad de las pruebas. En este caso se considera que ha ganado el juego.

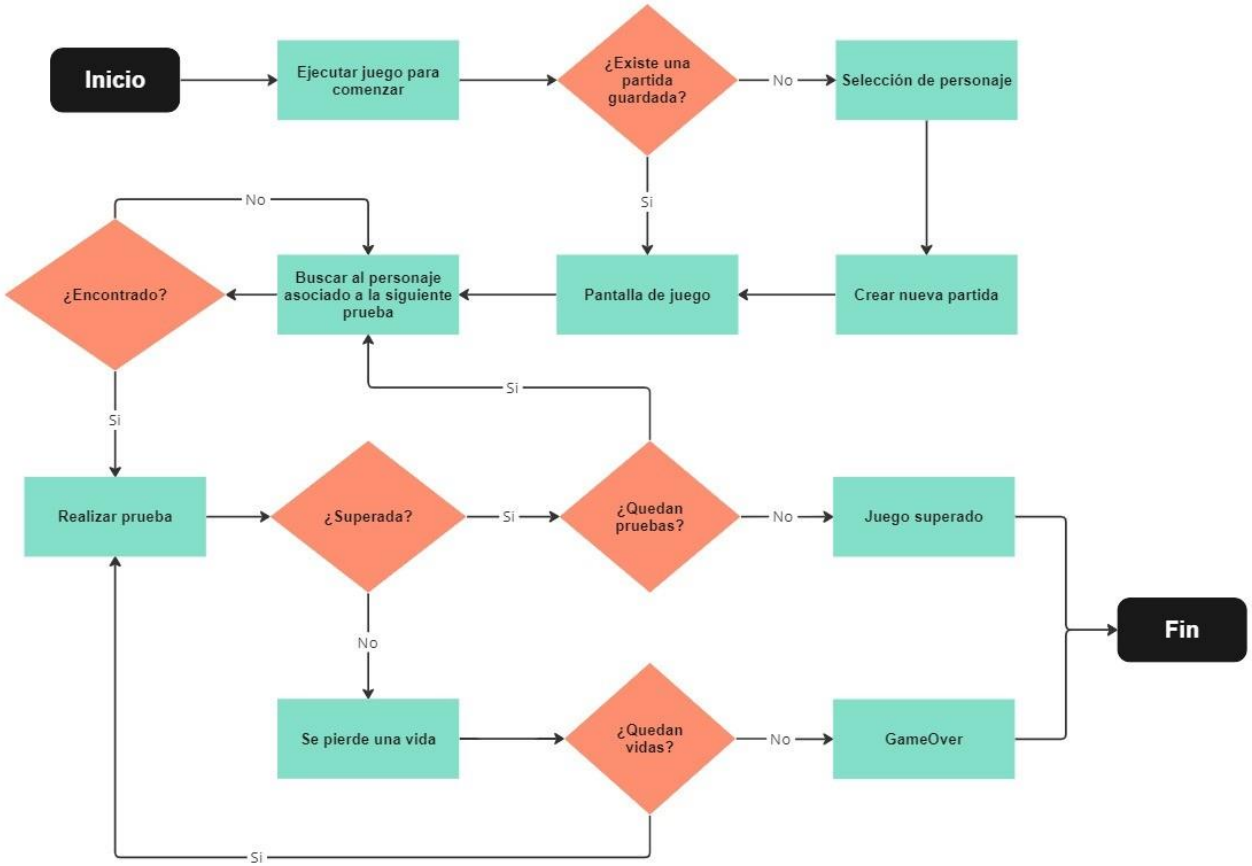


Figura 10: Diagrama de flujo del funcionamiento del juego

5.2 MAPAS

Los mapas en el juego se basan en distintas zonas de ICAI usando componentes del popular juego Pokémon Esmeralda de la *Gameboy Advance*. Para crear estos mapas, utilizamos la herramienta, *Tiled*, que permite crear mapas detallados y complejos de manera eficiente.

Cada mapa consta de cuatro capas diferentes: suelos, colisiones, decoración y extras. El correcto orden de estas es crucial, ya que, por ejemplo, no queremos que los suelos se dibujen encima de las paredes. Por ello en la Figura 11 se muestra la forma y orden en la que debemos crear las capas de nuestros mapas.

Figura 11: Diagrama piramidal de capas

La capa de suelos se emplea para definir el tipo de terreno en cada área del mapa, es decir, el suelo por el que el jugador se moverá, mientras que la capa de colisiones se utiliza para definir las áreas en las que el jugador no puede pasar, como paredes y obstáculos. La capa de decoración se agrega para añadir detalles visuales al mapa, como cuadros, libros y otros objetos. Finalmente, la capa de extras se utiliza para agregar elementos adicionales encima de algún objeto de capas anteriores.

Para crear los mapas, se empieza definiendo el tamaño y la forma del mapa. Luego, se agrega la capa de suelos y se define el tipo de terreno en cada área. A continuación, se añade la capa de colisiones y se identifican y marcan las áreas por las que el jugador no puede pasar.

Una vez que se han definido las capas de suelos y colisiones, se agrega la capa de decoración y se incorporan detalles visuales al mapa. Esto incluyó agregar cuadros, bebidas, plantas, máquinas expendedoras, libros y otros objetos para dar vida al entorno.

Finalmente, hay que agregar la capa de extras para añadir elementos adicionales al mapa, que requieren de una capa más.

Una vez completado el diseño del mapa, se prueba para asegurar de que sea lo más parecido a la realidad.

Para ver entender el concepto del diseño de las capas, en la Figura 12 se muestran las distintas capas de uno de los mapas del juego:

Figura 12: Capas del mapa secundario El Gruñidor

En el lado izquierdo se presenta únicamente la capa de suelos. A su derecha se muestra también la capa de colisiones superpuesta en la capa de suelos, la cual es esencial para limitar el movimiento del jugador en el juego. Después, se muestra además la capa de decoración, donde se pueden apreciar todas las decoraciones existentes en la cocina, mesas y en la barra.

Finalmente, se presenta la capa de extras, que aunque tiene pocos cambios, es necesaria para tener decoraciones adicionales encima de las existentes en la capa de decoración. La Figura 13 muestra una vista ampliada del mapa de la Figura 12 en la que se observa mejor la diferencia entre las dos últimas capas. Por ejemplo, la tablas de cortar de la cocina estaría en la capa de decoración, mientras que los componentes encima de las tablas deben estar en una capa adicional, es decir, la capa de extras.

Figura 13: Diferencia entre las capas de decoración y extras en el mapa secundario de El Gruñidor

5.2.1 ICAI

La imagen presentada muestra el mapa principal del juego, el cual está basado en una planta del edificio de ICAI, específicamente la planta de laboratorios o segunda planta. La elección de este mapa se debe a la diversidad de aulas que ofrece, incluyendo tanto aulas convencionales como laboratorios de distintos tipos.

Para facilitar la navegación del jugador en el mapa, se ha incluido un letrero en la entrada de cada una de las aulas, despachos o laboratorios, indicando el nombre correspondiente. De esta manera, el jugador puede identificar rápidamente la ubicación exacta en la que se encuentra.

En la siguiente figura (Figura 14) se puede ver como es este mapa, el mapa principal del juego.

Figura 14: Mapa principal del juego, segunda planta de ICAI

Cabe destacar que el mapa ha sido diseñado con un enfoque en la usabilidad y la accesibilidad para el jugador, lo que se refleja en la inclusión de elementos como los letreros en las entradas de las aulas y en la elección de un entorno familiar para muchos jugadores, como lo es el edificio de ICAI.

Este mapa es el escenario principal para la realización de la mayoría de las pruebas del juego, lo que significa que la mayoría de los personajes secundarios estarán presentes en este mapa y el jugador pasará una cantidad significativa de tiempo explorándolo.

Dado que este mapa es tan importante para el desarrollo del juego, se ha prestado especial atención a los detalles y características del diseño. Esto incluye elementos visuales atractivos y la creación de entornos desafiantes y diversos que mantengan al jugador comprometido y entretenido.

Además, se ha trabajado para asegurar que el mapa sea fácil de navegar y que el jugador pueda encontrar rápidamente los lugares y personajes que necesita para avanzar en el juego. Es decir, se ha hecho un esfuerzo consciente para crear un mapa que sea tanto atractivo como funcional, con el objetivo de proporcionar al jugador una experiencia de juego satisfactoria y emocionante.

5.2.2 EL GRUÑIDOR

El mapa presentado es uno de los dos mapas secundarios del juego, y está basado en un bar muy conocido por los alumnos de ICAI llamado El Gruñidor[24]. Aunque en este mapa sólo se desarrolla una de las pruebas del juego, esta puede considerarse una de las más importantes, por lo que hemos querido darle una ubicación distinta y especial.

El diseño de este mapa se ha cuidado al detalle, y se ha trabajado para lograr la mayor fidelidad posible con la realidad del lugar. Se han incluido elementos decorativos y detalles que hacen que el jugador sienta que está en el local real. Todo esto, junto con la importancia de la prueba que se lleva a cabo en este mapa, hace que la experiencia de juego sea emocionante y auténtica.

La Figura 15 muestra el mapa de El Gruñidor.

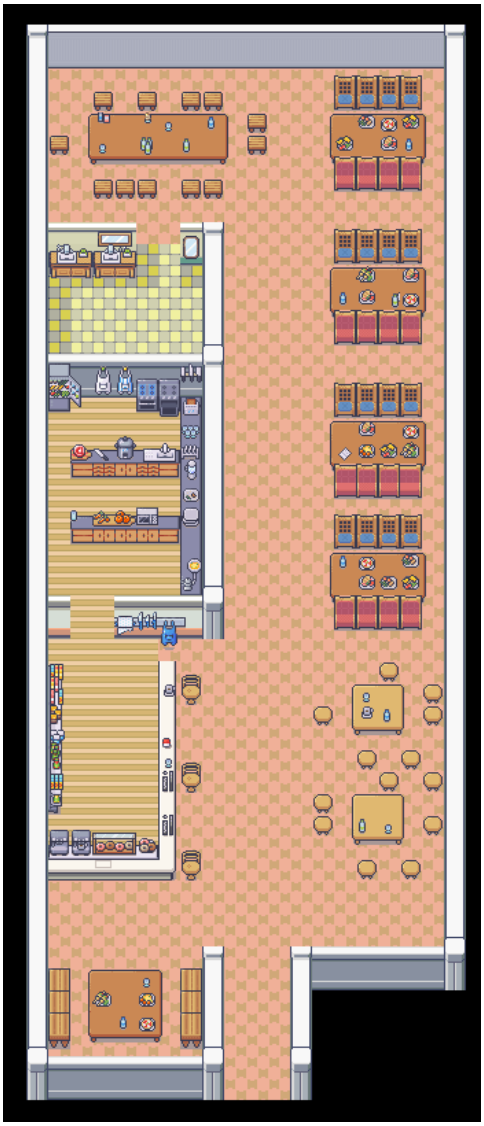


Figura 15: Mapa secundario del juego, El Gruñidor

Es importante destacar que para acceder a este mapa desde el mapa principal, el jugador deberá bajar por unas escaleras especificas (Escaleras 1). Esto es un detalle importante que se ha incluido para guiar al jugador y asegurarse de que sepa cómo llegar a este mapa secundario.

5.2.3 AULA 0-204 ICADE

El mapa presentado es el otro mapa secundario del juego, y está basado en una de las aulas más grandes de ICADE, la O-204, también conocida como "Maracaná". En este mapa se

desarrollará la prueba final del juego, una decisión tomada porque muchos de los exámenes finales de ICAI se han llevado a cabo en esta aula.

La siguiente figura (Figura 16) muestra como se ve este mapa.

Figura 16: Mapa secundario del juego, Maracaná

Esta aula se caracteriza por ser muy grande y fría, sin mucha decoración y con muchas mesas para que los alumnos realicen sus exámenes. Hemos querido reflejar esto en nuestro mapa, cuidando los detalles para que el jugador sienta que realmente está en el aula real. La simplicidad del diseño y la falta de decoración en el mapa se han utilizado para crear un ambiente tenso y emocionante, que se ajusta perfectamente a la importancia de la prueba final del juego que se llevará a cabo en este mapa.

Como en el mapa anterior, el jugador deberá bajar por unas escaleras especificas (Escaleras 2) del mapa de ICAI para acceder a este mapa secundario.

5.3 PERSONAJES

Los personajes son una parte importante de cualquier videojuego ya que ayudan a crear una experiencia más completa e inmersiva.

Los personajes se dibujan encima del mapa, es decir, componen lo que sería la última capa del mapa. Esto se muestra en la Figura 17 y se debe a que deben dibujarse por encima de todas las capas, ya que en caso contrario el personaje no se vería.

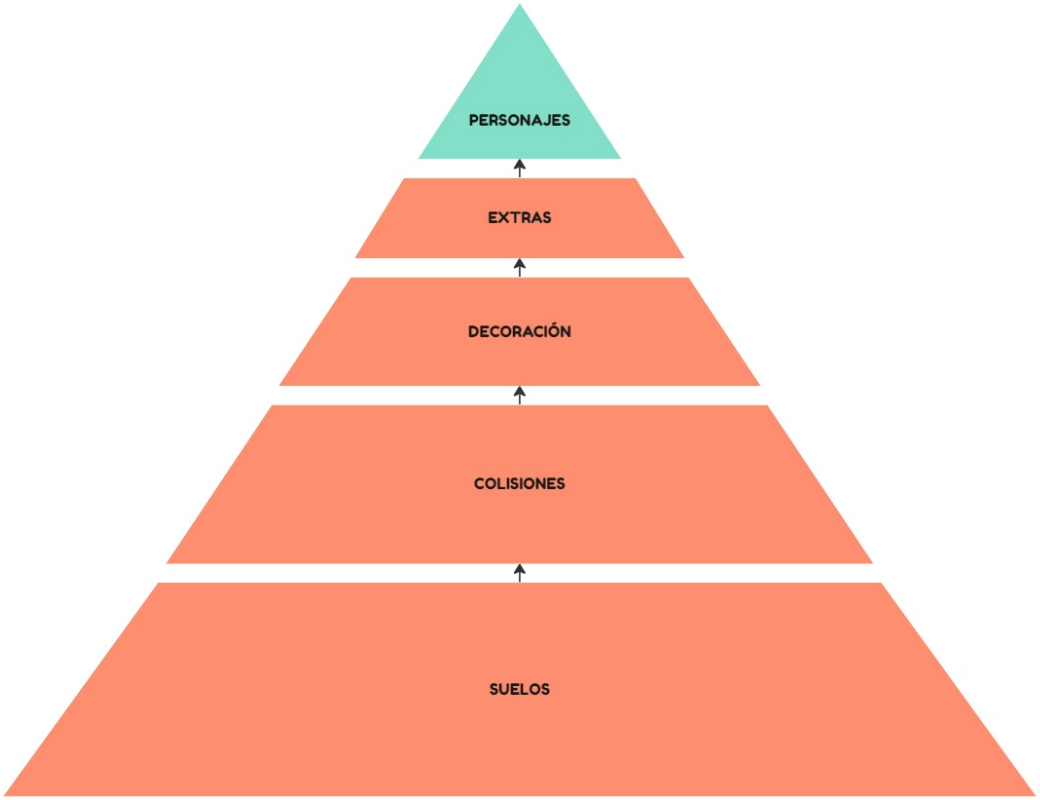


Figura 17: Diagrama piramidal de capas de un mapa y personajes

Estos personajes están diseñados a mano con PixilArt[25], lo que significa que cada uno de ellos es único y tiene su propia personalidad y características. PixilArt es una página web gratuita que permite a los usuarios crear y editar dibujos en formato de píxeles. Ofrece una variedad de herramientas y opciones para la creación de dibujos, como el uso de distintos pinceles, la selección de colores y la posibilidad de trabajar con capas.

Se han diseñado todos los personajes con un retrato incluido para darle más cercanía a los personajes y que el jugador pueda crear un vínculo con ellos más fácilmente.

5.3.1 PERSONAJES SECUNDARIOS

En este juego, los personajes secundarios desempeñan un papel fundamental, ya que son los que guían al jugador a través de las distintas pruebas y desafíos que se presentan en el juego.

Los personajes secundarios se conocen como NPCs, y se dividen en dos categorías: aquellos que tienen una prueba asociada y los que no la tienen. Todos ellos han sido diseñados con el objetivo de enriquecer la trama y proporcionar una experiencia de juego más completa y satisfactoria para el jugador.

5.3.1.1 Personajes asociados a una prueba

Los personajes secundarios con pruebas asociadas tienen la función de servir como guías al jugador durante el progreso del juego. Al interactuar con ellos en el momento oportuno, mostrarán al jugador la prueba correspondiente para superarla. En caso contrario, los personajes guiarán al jugador hacia la prueba que le corresponde en ese momento.

La Figura 18 incluye a todos los personajes secundarios con pruebas asociadas, y sus retratos.

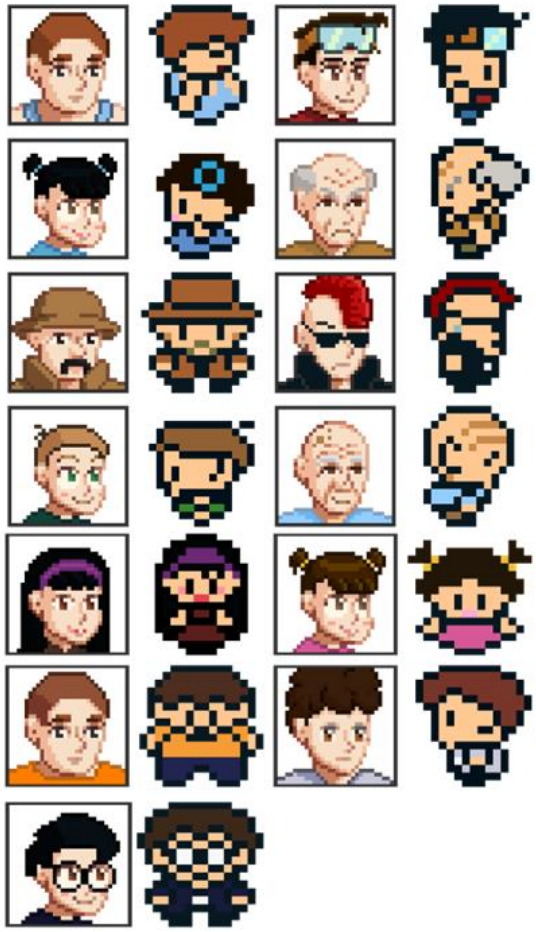


Figura 18: Personajes secundarios asociados a una prueba

5.3.1.2 Personajes no asociados a una prueba

Los personajes no asociados a pruebas son parte de la ambientación del juego y se encuentran distribuidos por todo el mapa. Estos personajes tienen un papel importante en la creación de la atmósfera del juego, ya que le dan vida y realismo al mundo que rodea al jugador.

A diferencia de los personajes asociados a pruebas, los NPCs sin pruebas asociadas tienen un diálogo limitado y siempre le dirán lo mismo al jugador al interactuar con ellos. Sin embargo, este diálogo está diseñado para que coincida con la ubicación y la personalidad del personaje. Por ejemplo, en el bar se puede encontrar a un camarero atendiendo a clientes, mientras que en las aulas se pueden encontrar profesores trabajando.

La inclusión de estos personajes secundarios no solo añade más elementos al juego, sino que también lo hace más inmersivo para el jugador, creando un mundo más realista y coherente. Además, el jugador puede explorar el mapa y descubrir a estos personajes que, aunque no tienen un papel fundamental en la trama, pueden enriquecer la experiencia del juego.

En Figura 19 se muestran todos los personajes secundarios sin prueba asociada.

Figura 19: Personajes secundarios no asociados a una prueba

5.3.2 EL JUGADOR

El juego también cuenta con el personaje principal, el jugador. Este personaje es el que el jugador controla a lo largo del juego, y es el encargado de superar todas las pruebas y desafíos que se presentan en su camino.

El personaje del jugador tiene la habilidad de moverse en cualquier dirección en el mapa y, para interactuar con los personajes secundarios, debe estar enfrente de ellos y mirando en su dirección.

El personaje puede ser representado de dos maneras en función de la elección del jugador al iniciar una nueva partida. Esta elección determina la imagen que representa al jugador en todo momento durante el transcurso del juego. En la Figura 20 se muestran los dos posibles aspectos que puede tomar el jugador.

Figura 20: El jugador

5.3.2.1 Movimientos y colisiones

El jugador puede moverse en todas las direcciones mediante la pulsación de teclas específicas (AWSD). Cuando se presiona una de estas teclas, se realiza una comprobación de colisión en la dirección correspondiente para determinar si el jugador puede o no avanzar en esa dirección. Si hay un obstáculo en el camino, como paredes, mesas, sillas, etc., el jugador girará en esa dirección, pero no se moverá. Por otro lado, si no hay obstáculos, el jugador se moverá en la dirección deseada.

Las colisiones se gestionan mediante el uso de capas. Es decir, se realiza una comprobación de colisión en cada capa para determinar si el jugador puede avanzar en la dirección seleccionada. Si la ubicación a la que el jugador desea moverse está en la capa de colisiones, entonces no se le permitirá avanzar, ya que se considera que hay una colisión en esa ubicación.

La Figura 21 representa el funcionamiento del movimiento del jugador, el cual se gestiona desde un metodo llamado update, por lo que el diagrama de estados (Figura 21) muestra el funcionamiento de este metodo.

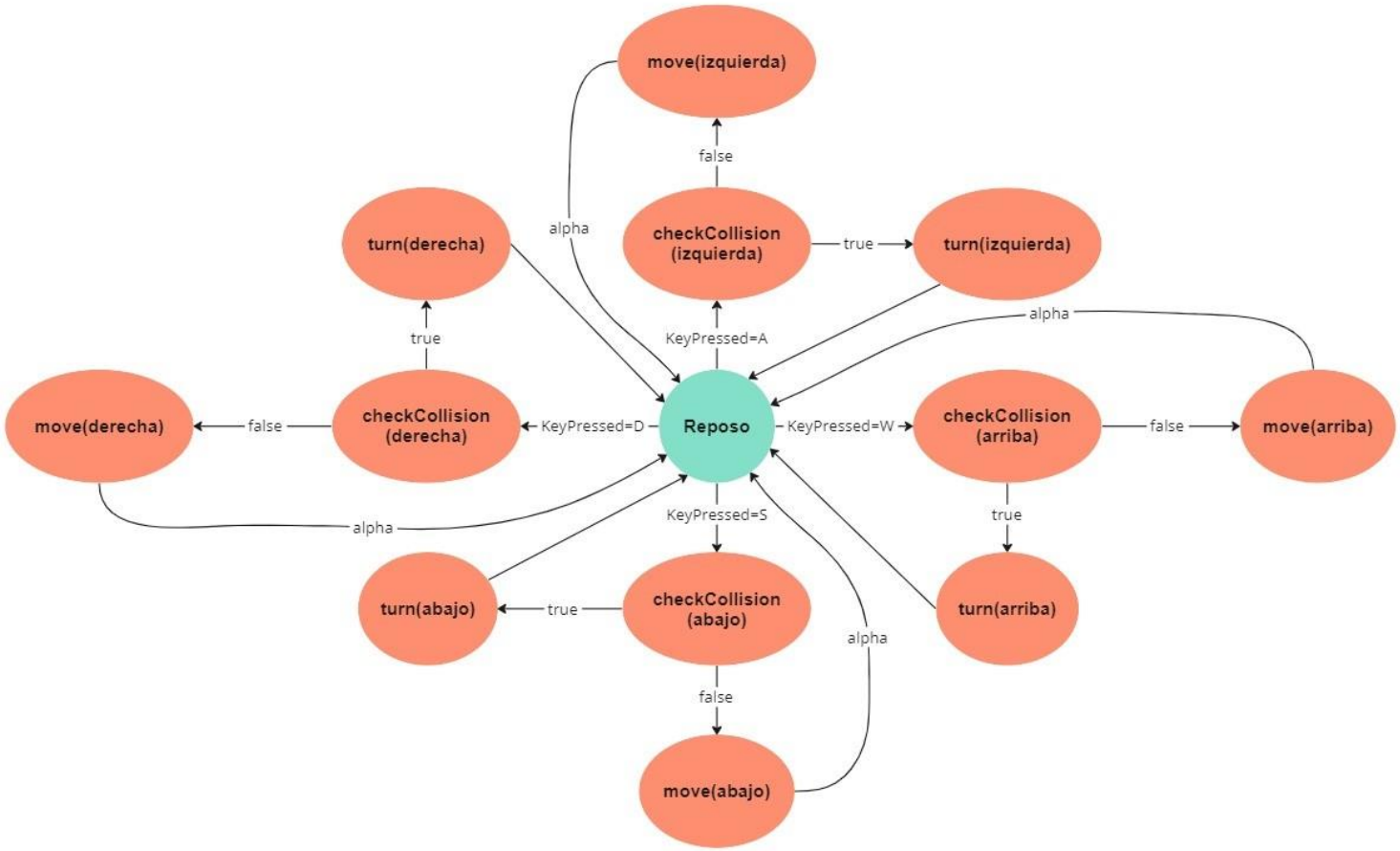


Figura 21: Diagrama de estados del movimiento del jugador

La función *update* es un método que se encarga de actualizar el estado del juego en cada cuadro o *frame*. En este caso, se encarga de manejar los movimientos del jugador y las colisiones con el mapa y los objetos en el mismo.

Primero, se comprueba si el jugador se encuentra en determinadas coordenadas del mapa y en ese caso se le teletransporta a otra ubicación para moverse entre los distintos mapas. Esto no queda representado en la Figura 21.

Luego, se comprueba si el usuario ha pulsado alguna de las teclas correspondientes a los movimientos del jugador (arriba, abajo, izquierda, derecha). Para comprobar si el jugador está bloqueado en una posición, se utiliza el método *checkCollision*. Este método comprueba si la próxima celda en la que se moverá el jugador está bloqueada (es decir, contiene un obstáculo en el mapa), y si es así, también comprueba si hay algún NPC (personaje no-jugador) en esa posición. Si la próxima celda está bloqueada o si hay un NPC en esa posición, se devuelve true, lo que indica que el jugador no puede moverse en esa dirección.

Si el jugador no está bloqueado en la dirección en la que intenta moverse, se le permite moverse en esa dirección utilizando el método *move* de la clase *Player*. En caso contrario, se invoca al método *turn* para que el jugador gire en esa dirección en lugar de moverse en ella.

Además, se utiliza un concepto de tiempo *alpha* para realizar la comprobación de teclas pulsadas. Esto implica que cada cierto tiempo (*alpha*), se vuelve a reposo y se comprueba qué tecla se ha presionado iniciando el proceso nuevamente.

Este sistema de movimiento y colisiones garantiza que el jugador no pueda pasar a través de objetos o paredes y que solo pueda avanzar en áreas permitidas dentro del mapa.

5.3.2.2 Animación de movimiento

La animación de movimiento del jugador se logra utilizando doce imágenes diferentes que se dividen en cuatro filas, cada una correspondiente a una dirección del movimiento (arriba, abajo, izquierda y derecha). En la Figura 22 se muestran las imágenes (también llamadas

texturas) que se han utilizado para la animación del jugador, cada fila tiene tres imágenes que se utilizan en diferentes momentos del movimiento para lograr una transición fluida entre los *frames*.

Figura 22: Imágenes, llamadas texturas, utilizadas en la animación del jugador

La duración del movimiento está determinada por un valor de tiempo llamado *alpha*, que indica el tiempo de una animación completa. El método utilizado para animar el movimiento del jugador es *animate*, el cual comprueba el valor de *alpha* en relación con el tiempo total de movimiento y establece la textura del jugador en consecuencia.

En caso de por ejemplo estar moviéndose el jugador hacia la derecha, el funcionamiento del método es el siguiente:

- Si *alpha* es menor a 1/3, la textura del jugador se establecerá en andar_derecha_1.png (siendo esta la imagen en la fila 2 columna 2 de la Figura 22).
- Si *alpha* está entre 1/3 y 2/3, la textura del jugador se establecerá en derecha.png (imagen en la fila 2 columna 1 de la Figura 22).

- Si *alpha* está entre 2/3 y 1, la textura del jugador se establecerá en andar_derecha_2.png (imagen en la fila 2 columna 3 de la Figura 22).

Una vez *alpha* sea igual a 1, la animación y el movimiento terminan, volviendo a reposo como describe la Figura anterior. De esta manera, se logra una animación fluida y realista del movimiento del jugador.

5.3.2.3 Interacciones y diálogos

En esta sección se explica cómo funcionan las interacciones del jugador con los distintos NPCs.

El método *gameCycle* es el bucle principal del juego, se ejecuta continuamente mientras el juego está en marcha y es el encargado de escuchar a interacciones. El método utiliza la instancia de la clase *gameUpdater* para actualizar la lógica del juego y comprueba si el jugador está interactuando con algún NPC mediante el método *checkInteraction*.

La Figura 23 muestra un diagrama de flujo del funcionamiento del método *checkInteraction*, que gestiona las interacciones del jugador con los personajes secundarios.

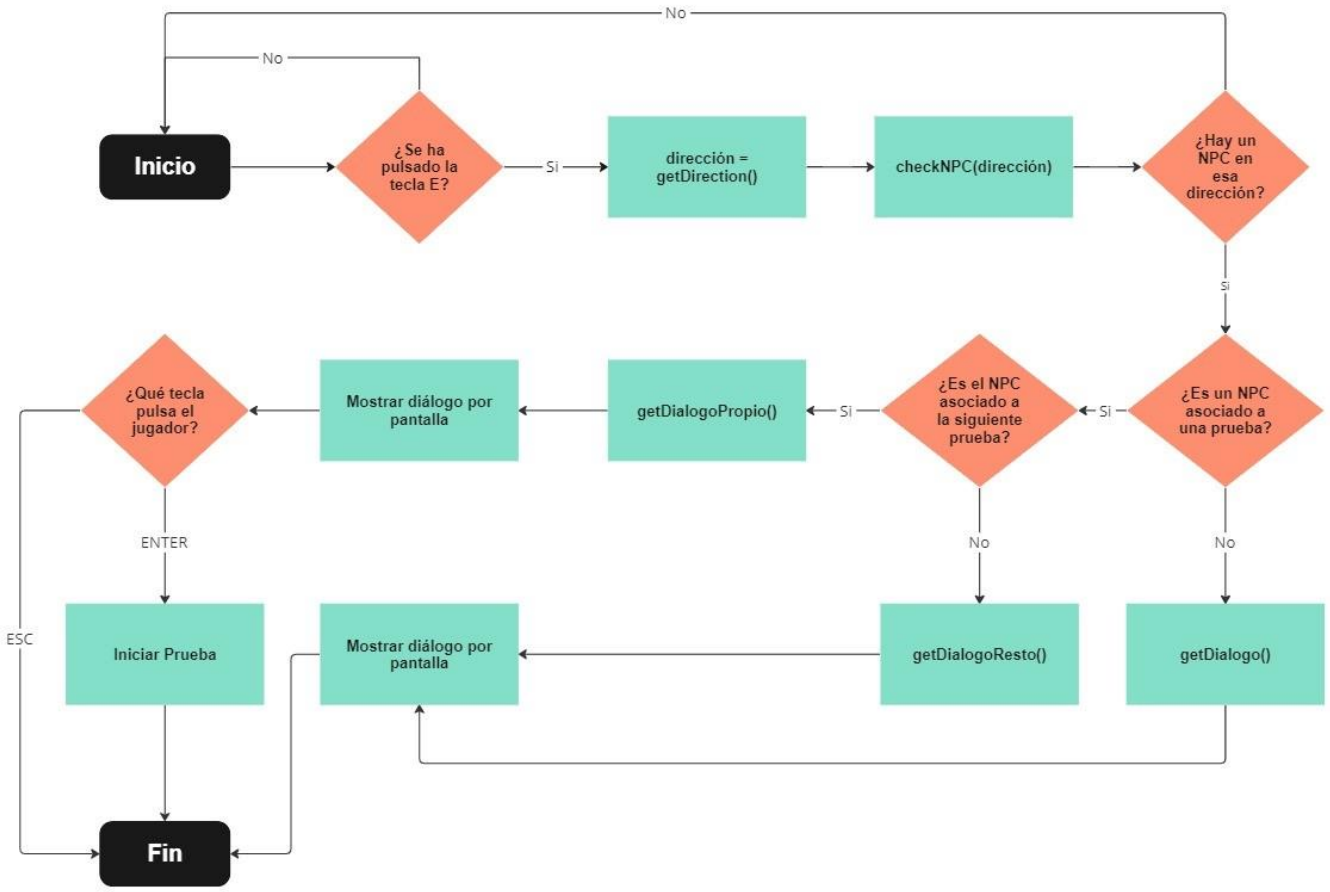


Figura 23: Diagrama de flujo de las interacciones del jugador con los personajes secundarios

El método *checkInteraction* comprueba si el jugador ha presionado la tecla "E" en el teclado y si es así, busca un NPC en la dirección en la que está mirando el jugador mediante el método *checkNPC*. El método *checkNPC* busca en la lista de NPCs si hay algún NPC en las coordenadas especificadas, devolviendo el NPC correspondiente si lo encuentra y nulo si no lo hace.

Si el jugador está interactuando, es decir, hay un NPC en esa dirección, se establece la variable de estado *interacting* en true.

Luego, el método determina el tipo de NPC con el que el jugador está, y muestra la imagen del retrato del NPC y su diálogo correspondiente. Si el NPC tiene una prueba asociada, el método también verifica si el jugador está en la prueba correcta antes de mostrar el diálogo del NPC.

Después de mostrar el diálogo del NPC, el método se encarga de escribir el texto letra por letra hasta que todo el texto se muestre.

Si el jugador presiona *Enter*, el juego comienza la prueba asociada con el NPC si tiene una prueba asociada. Si el jugador presiona *Esc*, la interacción con el NPC finaliza y se reinicia el cuadro de diálogo.

5.4 LAS PRUEBAS

Las pruebas del juego tienen como objetivo reforzar los conocimientos adquiridos en la asignatura de Microprocesadores a través de una serie de trece pruebas diseñadas específicamente para cubrir todo el temario. Estas pruebas se presentan en un orden que coincide con el calendario de la asignatura, permitiendo a los estudiantes usar el juego como una herramienta adicional de estudio.

Cada una de las pruebas se enfoca en un tema específico, desde la introducción a la programación en C hasta la organización de un sistema completo. Además, cada prueba está

asociada a un personaje distinto en el juego, lo que requiere que el jugador navegue a través del mapa para encontrar a los personajes y superar todas las pruebas en el orden establecido.

Las pruebas están diseñadas para involucrar a los estudiantes en una variedad de tareas prácticas, incluyendo el uso de puertos de entrada y salida del PIC32, temporizadores, interrupciones, comunicación serie asíncrona, PWM, buses I2C y SPI, conversión AD, y organización de sistemas. La culminación del juego es un examen final que abarca todo el temario y permite al estudiante evaluar su conocimiento y comprensión de la materia.

Este es el orden de las pruebas:

1) Introducción a la programación en C
2) Puertos de entrada y salida del PIC32
3) Temporizadores y remapeo de pines
4) Laboratorio de temporizadores
5) Interrupciones
6) Laboratorio de interrupciones
7) Comunicación serie asíncrona
8) PWM
9) Repaso para el examen de laboratorio
10) Buses I2C y SPI
11) Conversar AD
12) Organización de un sistema
13) Examen final

Este orden de las pruebas se ha decidido en función de un cronograma aproximado de la asignatura. La Tabla 2 muestra el cronograma orientativo real de la asignatura de microprocesadores impartida en ICAI en el grado de GITT (Grado de Ingeniería de Telecomunicaciones).

CRONOGRAMA ORIENTATIVO DE MICROPROCESADORES GITT														
								I				SS		
PROGRAMA DE TEORÍA	**18-1**	**25-1**	**1-2**	**8-2**	**15-2**	**22-2**	**1-3**	**8-3**	**15-3**	**22-3**	**29-3**	**5-4**	**12-4**	**19-4**
Tema 1. Introducción a los microcontroladores														
Tema 2. Lenguaje C para programación en bajo nivel														
Tema 3. Puertos de entrada y salida digital														
Tema 4. Temporizadores														
Control de C y puertos														
Tema 5. Arquitectura del MIPS32														
Tema 6. Soporte de interrupciones en la familia PIC32MX														
Tema 7. Comunicación serie asíncrona														
Resolución exámenes pasados/Corrección examen														
Control														
Tema 8. Buses I2C y SPI														
Tema 9. Conversor Analógico Digital														
Tema 10. Modulación por ancho de pulso (PWM)														
Tema 11. Organización de un sistema digital basado en microprocesador														

Tabla 2: Cronograma orientativo de la asignatura Microprocesadores

Cada una de las pruebas esta guardada en un JSON. El JSON contiene información como preguntas, opciones de respuesta y la respuesta correcta correspondiente a cada pregunta.

Como se describió en la sección 5.3.2.3, tras la interacción del jugador con un personaje secundario, en caso de ser el asociado a la prueba siguiente, se iniciaría el proceso de la prueba. Con lo que llegamos al funcionamiento de este proceso. La Figura 24 muestra un resumen de este proceso, el cual está dividido en tres partes principales:

1. Iniciar la prueba
2. Flujo de la prueba
3. Terminar la prueba

Figura 24: Diagrama de flujo del proceso general de prueba

Primero, cuando el jugador interactúa con un personaje asociado a una prueba, se llama a la función *startTest* en la clase *Game*, pasando como parámetro el número de la prueba a la que se quiere acceder. Esta es la primera parte del proceso completo de prueba, es decir el Inicio de prueba que está representado en la Figura 24 como Iniciar Prueba.

Esta parte del proceso se muestra detalladamente en un diagrama de flujo (Figura 25), que representa el funcionamiento del método *startTest*.

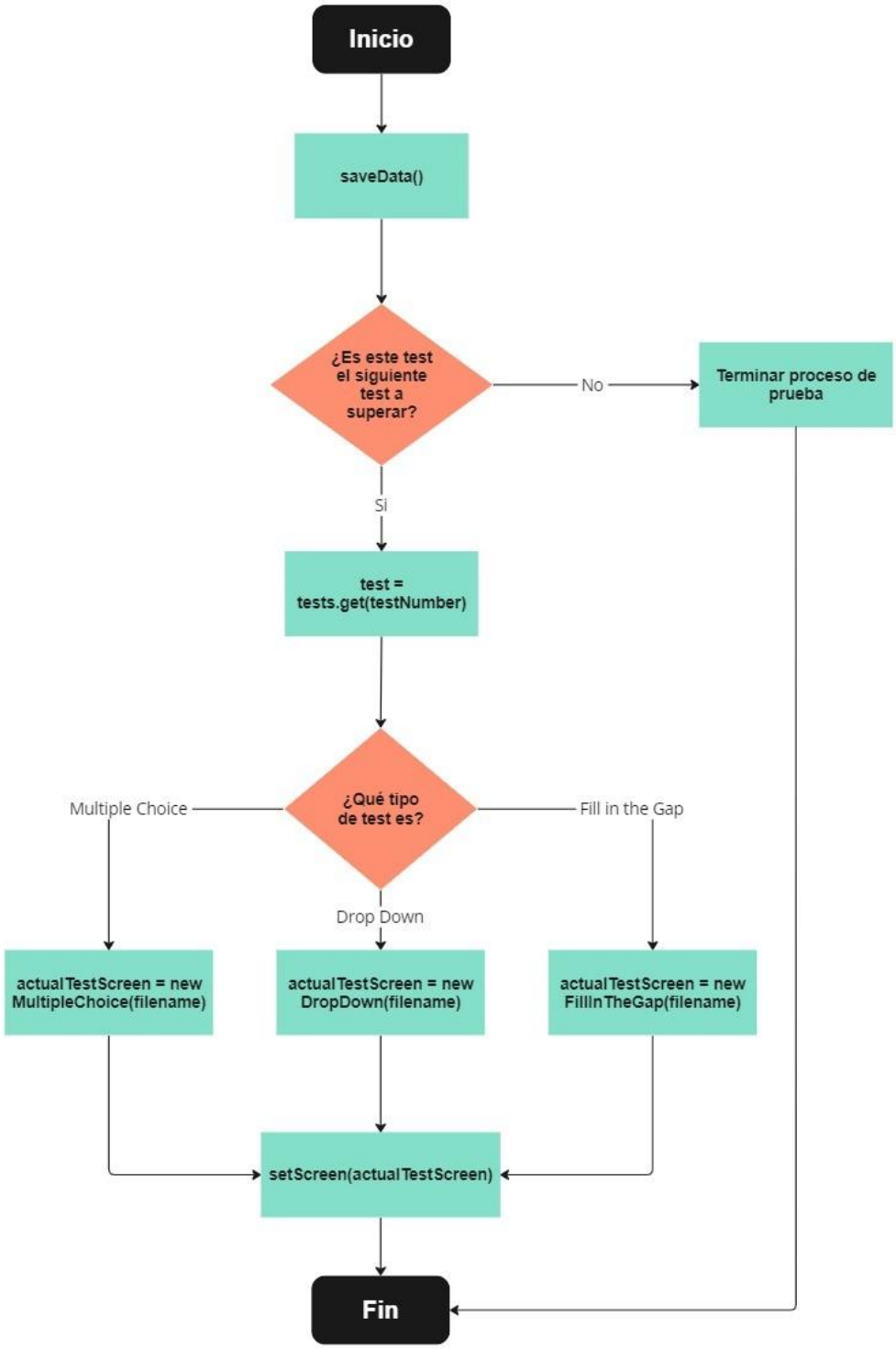


Figura 25: Diagrama de flujo del proceso de Iniciar Prueba

Dentro del método *startTest*, se guarda la partida actual y en caso de estar en la prueba correspondiente se obtiene la información de la prueba a partir de la variable *tests*, que es una lista que contiene la información de cada prueba en un formato específico.

Dependiendo del tipo de prueba que se va a crear, se llama al constructor correspondiente para crear el objeto *actualTestScreen*. Los tres tipos de prueba disponibles son: *MultipleChoice*, *DropDown* y *FillInTheGap*.

Una vez creada la pantalla de la prueba, se establece como pantalla principal mediante la función *setScreen* en la clase *Game*. La pantalla de la prueba se muestra al jugador, quien debe completarla correctamente para avanzar en el juego.

Así comienza el flujo interno de la prueba (Figura 26). Que, en ámbitos generales es igual para los tres tipos de pruebas. Las especificaciones de cada tipo se describen más adelante en este capítulo.

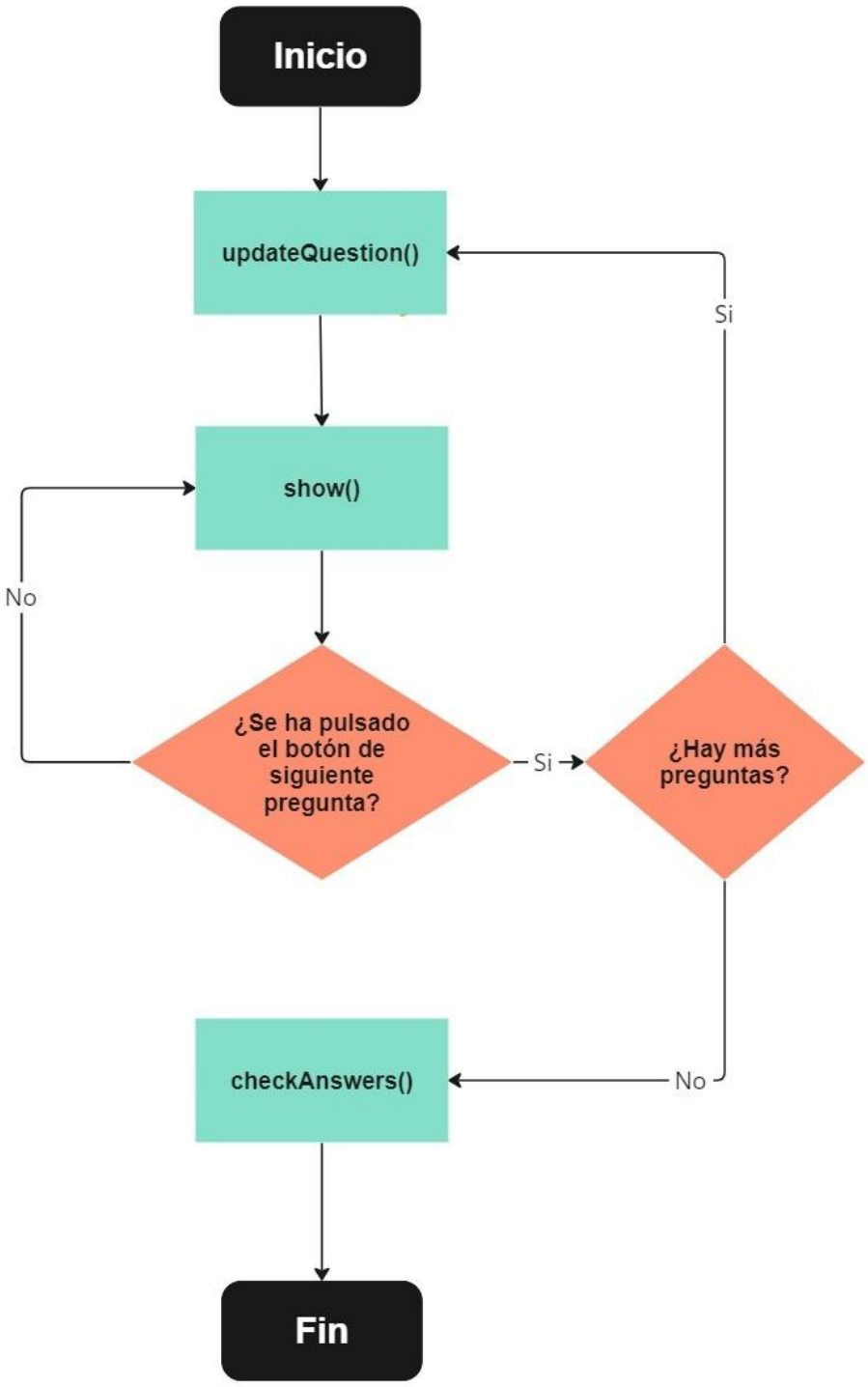


Figura 26: Diagrama de flujo del Flujo Interno de la Prueba

El diagrama (Figura 26) representa el flujo de funcionamiento interno de una prueba en el juego. El primer paso es la carga de la primera pregunta o enunciado de la prueba en el método *updateQuestion,* y seguidamente el flujo pasara por el método *show* donde se muestra por pantalla al jugador ese primer problema a resolver.

A continuación, el jugador debe seleccionar una respuesta, dependiendo del tipo de pregunta que se esté presentando y pulsar el botón de siguiente pregunta. Si hay más preguntas a resolver se vuelve al método *updateQuestion* que actualizaría la pregunta a la siguiente, volviendo otra vez a mostrarla con el método *show*.

En caso de haber terminado con todas las preguntas de la prueba, se inicia el método *checkAnswers* que comprueba si el jugador ha superado la prueba o no.

Tras comprobar las respuestas comienza la terminación de la prueba, representado en la Figura 27, llamando al método *endTest* al que pasamos como parámetro un *boolean* indicando si la prueba ha sido superada (true) o no (false).

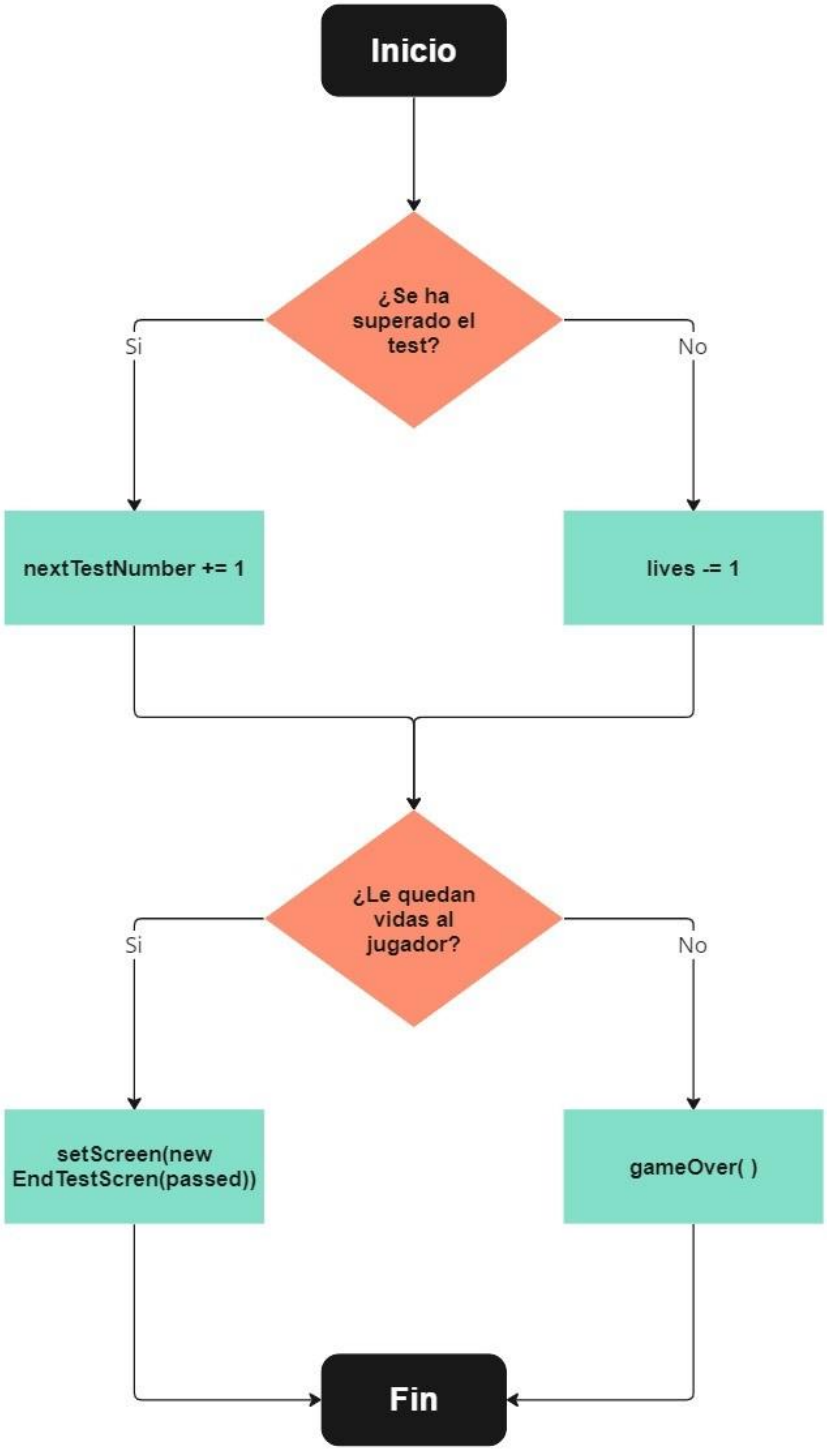


Figura 27: Diagrama de Flujo del proceso Terminar Prueba

El método *endTest* finaliza la prueba actual. Si la prueba fue superada, el siguiente número de prueba se incrementa en 1, de lo contrario se resta una vida. Seguidamente, si todavía quedan vidas, la pantalla principal se establece en la pantalla final de la prueba y se muestra el resultado de la prueba. Si no quedan vidas, el juego termina y se muestra la pantalla de *Game Over*.

5.4.1 PRUEBA TIPO MULTIPLE CHOICE

La primera categoría de pruebas que se presenta es conocida como Multiple Choice (prueba de selección múltiple). Estas pruebas se componen de una pregunta con una única respuesta correcta, aunque se ofrecen 4 opciones diferentes para que el jugador elija la correcta. Estas opciones se diseñan de manera que desafíen al jugador y no resulten evidentes o fáciles de determinar.

5.4.1.1 Formato JSON

El formato del JSON de Multiple Choice es una estructura de datos compuesta por un conjunto de preguntas, donde cada pregunta tiene una serie de opciones y una respuesta correcta.

La estructura se compone de una serie de objetos, donde cada objeto representa una pregunta, y dentro de ese objeto se encuentran los siguientes campos:

- "PREGUNTA": una cadena de texto que representa la pregunta en sí.
- "OPCION 1" a "OPCION 4": una cadena de texto que representa cada una de las opciones propuestas para responder la pregunta.
- "CORRECTA": una cadena de texto que indica la opción correcta entre las propuestas.

Figura 28 muestra cómo se estructura un objeto JSON (una pregunta) de esta prueba.

Figura 28: Formato JSON de una pregunta Multiple Choice

5.4.1.2 Lectura del JSON

Para cargar las pruebas de Multiple Choice, se debe leer el JSON. Para ello se ha creado una clase llamada *ReadMultipleChoice* que es la que permite leer y cargar preguntas de opción múltiple desde un archivo JSON. La clase contiene un método llamado *loadQuestions* (Figura 29) que toma el nombre de un archivo JSON como parámetro y devuelve una lista de objetos de la clase *MultipleChoiceQuestion*.

El método utiliza la biblioteca LibGDX para cargar el archivo JSON. Luego, itera sobre los objetos JSON en el archivo y extrae las preguntas y opciones de respuesta para crear objetos *MultipleChoiceQuestion* y los agrega a la lista de preguntas.

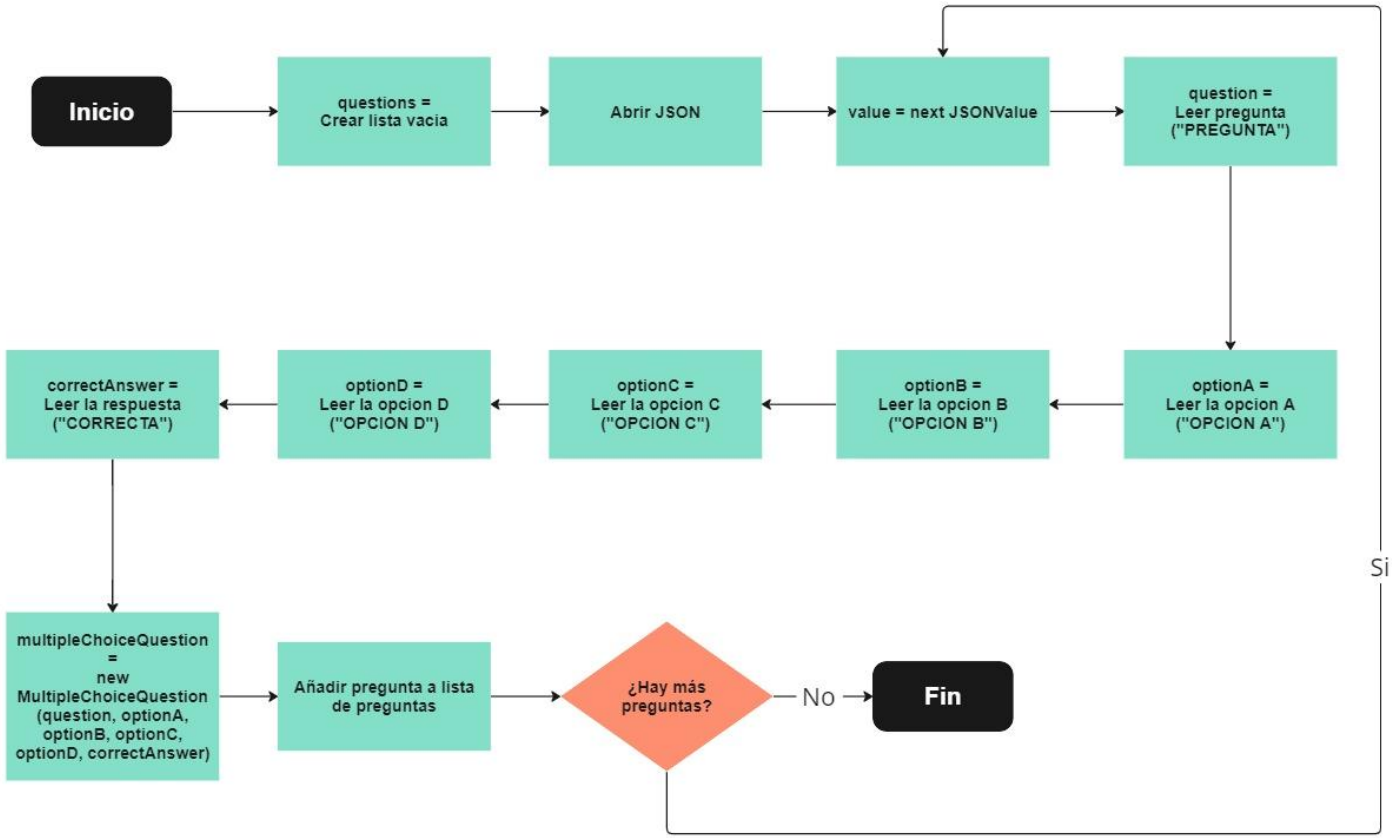


Figura 29: Diagrama de flujo del proceso de lectura de un JSON de una prueba tipo Multiple Choice

5.4.1.3 Display

La pantalla de la prueba tipo Multiple Choice consta de varios elementos:

- Pregunta: La pregunta se muestra en la parte superior de la pantalla, y se presenta en un formato claro y fácil de leer para que el jugador pueda entenderla claramente.
- Opciones de respuesta: Las opciones de respuesta se muestran debajo de la pregunta, en una lista ordenada, y el jugador debe seleccionar la opción que considera correcta. Las opciones incorrectas están diseñadas para hacer pensar al jugador y deben ser plausibles. Estas opciones están formadas por un grupo de botones por lo que solo se puede seleccionar uno.
- Botón de siguiente pregunta: El botón de siguiente se encuentra debajo de las opciones de respuesta y se utiliza para guardar la respuesta seleccionada por el jugador y pasar a la pregunta siguiente.

La Figura 30 muestra como se ha diseñado la pantalla con una pregunta del tipo Mutiple Choice, esta figura representa los componentes que componen a esta pantalla.

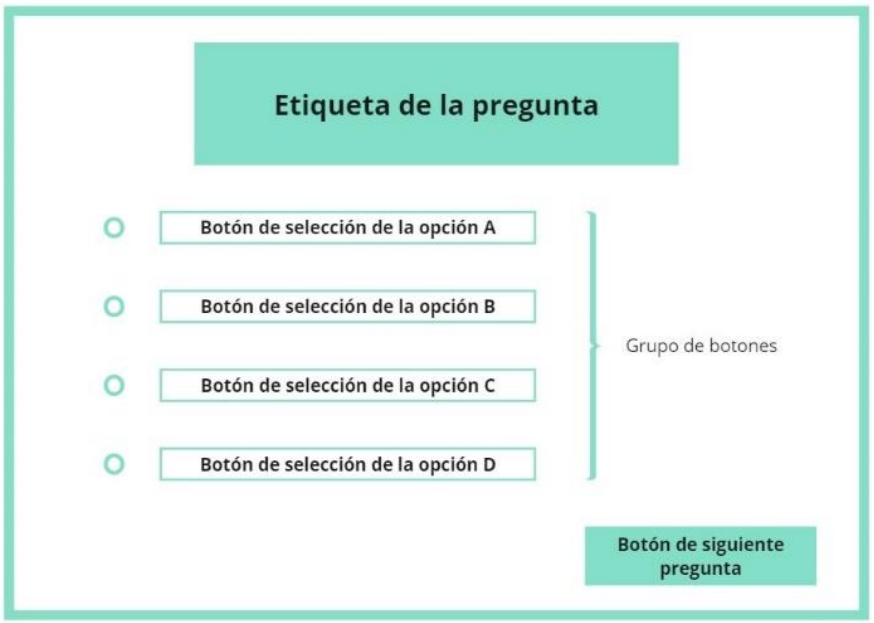


Figura 30: Diseño de una pregunta Multiple Choice en pantalla

El resultado de la pantalla de esta prueba se puede ver en la Figura 31. Esta muestra una captura de pantalla de una prueba real del juego.

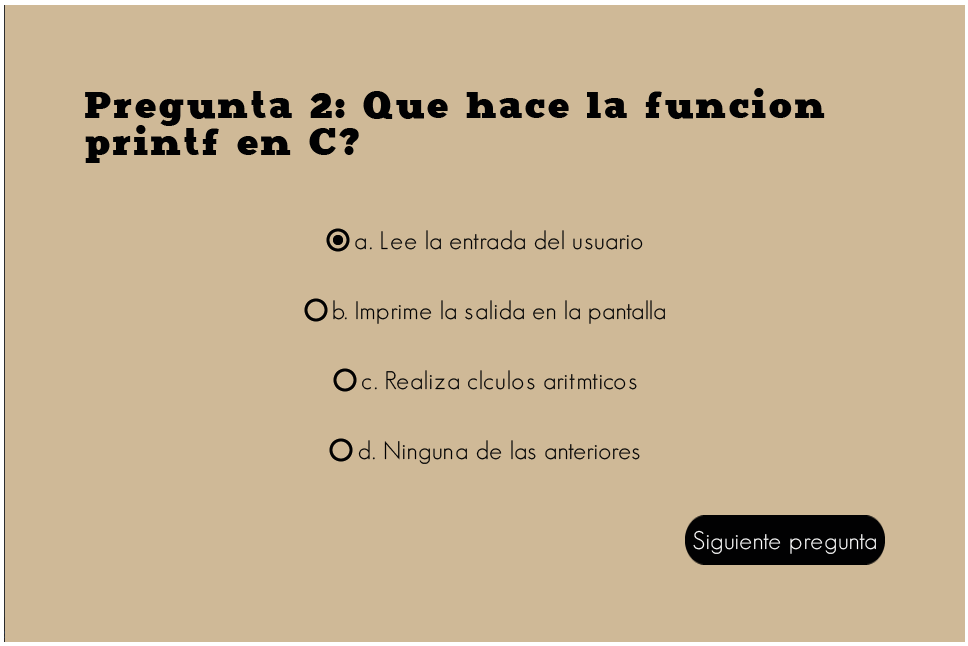


Figura 31: Captura de pantalla de prueba Multiple Choice

5.4.2 PRUEBA TIPO DROP DOWN

En esta sección, se explica la implementación de la prueba de opción desplegable. La prueba de tipo Drop Down (prueba de opción despegable) es un tipo de pregunta comúnmente utilizado en cuestionarios y evaluaciones en línea. Este tipo de pregunta se presenta como una pregunta con uno o varios espacios en blanco, cada uno de los cuales se puede llenar con una opción de respuesta de una lista desplegable.

Los estudiantes deben seleccionar la opción correcta para cada espacio en blanco. Este tipo de pregunta puede ser particularmente útil en situaciones donde se desea evaluar la capacidad del estudiante para aplicar un conocimiento específico.

5.4.2.1 Formato JSON

En la prueba Drop Down, cada objeto del JSON tiene las siguientes propiedades:

- "PREGUNTA": Es la pregunta que se presenta al usuario.
- "HUECOS": Es la cantidad de espacios en blanco que se deben llenar con las respuestas correctas.
- "OPCIONES": Es la cantidad de opciones que se presentan por cada hueco al usuario.
- "OPCION A", "OPCION B", "OPCION C" : Son las opciones que se presentan al usuario. El número de opciones variara dependiendo del valor de "OPCIONES".
- "ENTRADA 1", "ENTRADA 2", "ENTRADA 3", "ENTRADA 4": Son las respuestas correctas que se deben ingresar en los espacios en blanco de la pregunta. El número de respuestas correctas o campos de "ENTRADA" es igual al valor de "HUECOS".

El JSON de esta prueba debería tener una estructura como la que se muestra en la Figura 32.

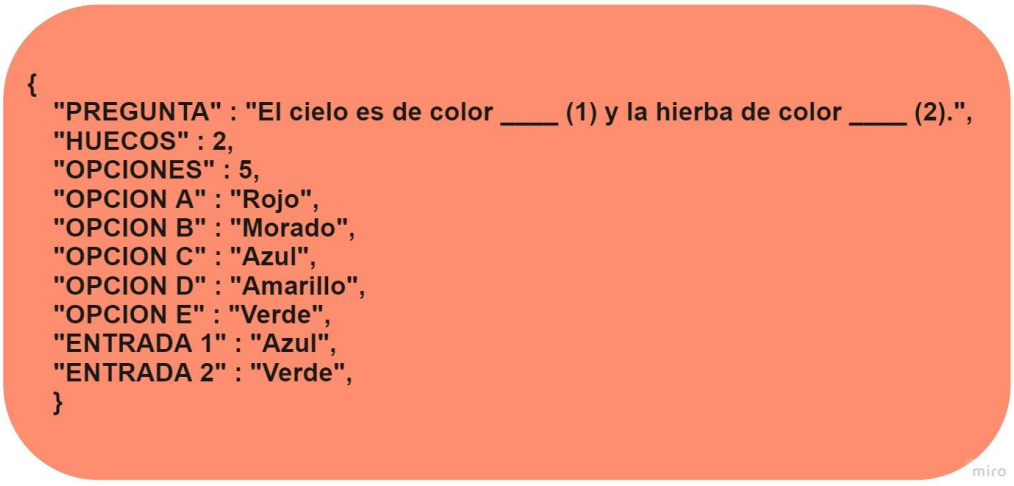


Figura 32: Formato JSON de una pregunta Drop Down

5.4.2.2 Lectura del JSON

Para la lectura de este tipo de prueba se crea la clase *ReadDropDown*. Esta contiene, al igual que en el resto de los tipos de pruebas, un método *loadQuestions* que toma el nombre de un

archivo JSON como parámetro y devuelve una lista de preguntas de la clase *DropDownQuestion*. Este método está representado en la Figura 33 en forma de diagrama de flujo.

El método carga el archivo JSON e itera sobre los objetos JSON en el archivo. Dentro del bucle, se extrae la pregunta, el número de espacios en blanco y el número de opciones para la pregunta, y se crea un objeto *DropDownQuestion*.

Luego, se extraen las opciones y las respuestas correctas para la pregunta y se agregan al objeto *DropDownQuestion* correspondiente y se agrega a la lista. Una vez que se han procesado todas las preguntas, la lista de preguntas se devuelve como resultado.

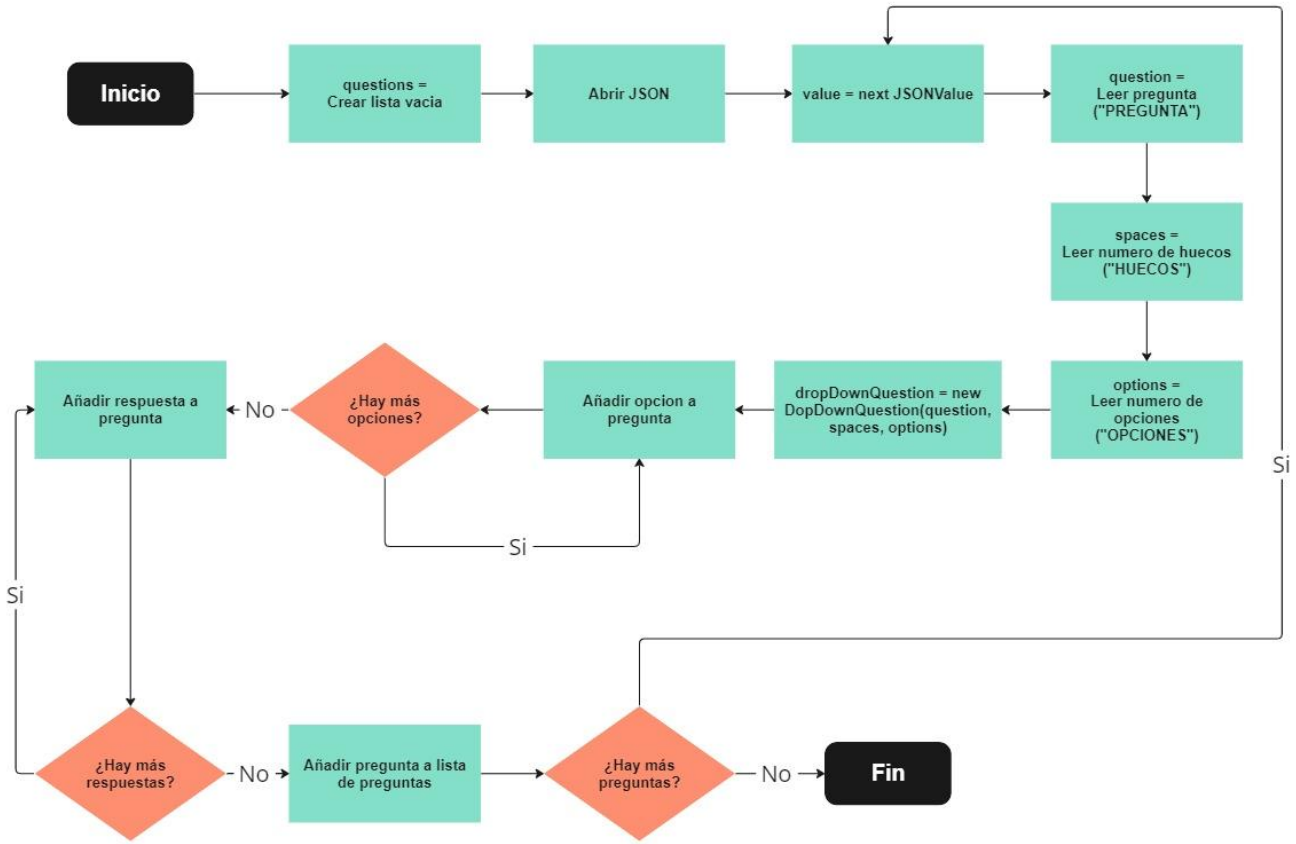


Figura 33: Diagrama de flujo del proceso de lectura de un JSON de una prueba Drop Down

5.4.2.3 Display

La pantalla de la prueba tipo Drop Down está compuesta de varios elementos:

- Título: El titulo se muestra en la parte superior e indica el número de pregunta en el que está el jugador.
- Pregunta: La pregunta se muestra en la parte central de la pantalla, debajo del título, y se presenta en un formato claro y fácil de leer para que el jugador pueda entenderla claramente. Estas preguntas tienen huecos en ellas con un numero marcado a su derecha que indica que numero de hueco es.
- Cajas de selección múltiple: Las cajas de opciones se muestran debajo de la pregunta, y el jugador debe seleccionar la opción que considera correcta para cada uno de los huecos.
- Botón de siguiente pregunta: El botón de siguiente se encuentra debajo de las opciones de respuesta y se utiliza para guardar la respuesta seleccionada por el jugador y pasar a la pregunta siguiente.

La siguiente figura (Figura 34) representa el diseño de la pantalla con una pregunta del tipo Drop Down con todos sus componentes marcados.

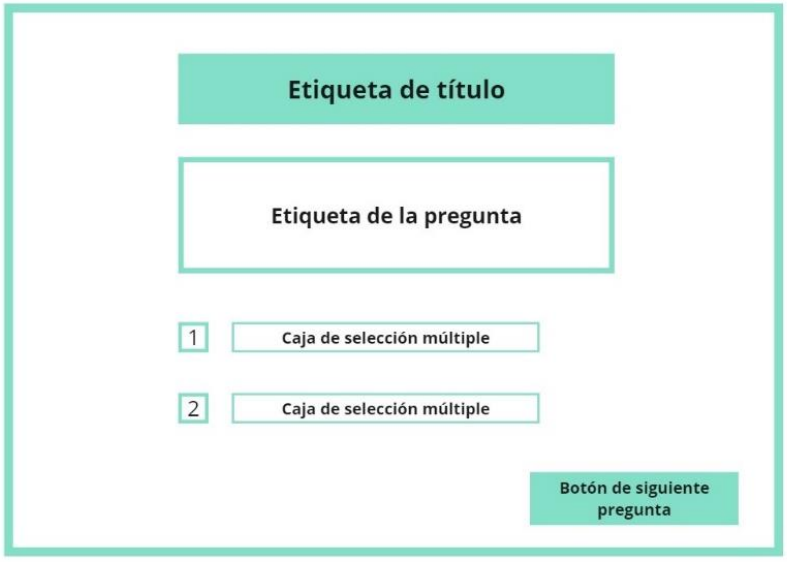


Figura 34: Diseño de una pregunta Drop Down en pantalla

El resultado del diseño mostrado en la Figura 34 en una prueba real de tipo Drop Down puede verse en la Figura 35.

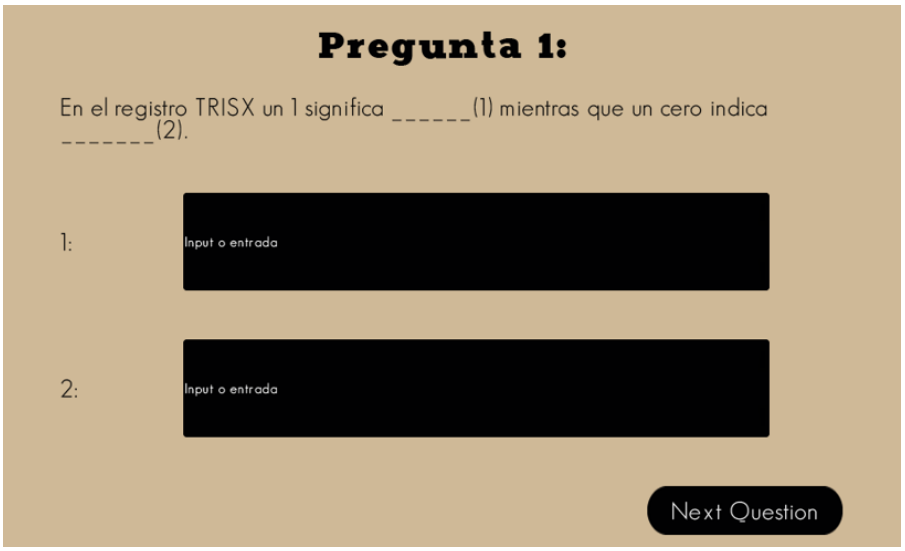


Figura 35: Captura de pantalla de una prueba Drop Down

5.4.3 PRUEBA TIPO FILL IN THE GAP

Por último están las pruebas Fill in the Gap (prueba de rellenar los huecos en blanco). En este tipo de prueba, se presentan frases, oraciones o párrafos con espacios en blanco que deben ser completados con la palabra o palabras adecuadas. Esta prueba los estudiantes deben demostrar su capacidad para utilizar el lenguaje de forma precisa y efectiva, lo que puede ser útil en una variedad de contextos, incluyendo la comunicación cotidiana, el ámbito académico o en este caso la programación.

5.4.3.1 Formato JSON

Cada objeto (pregunta) dentro del JSON tiene las siguientes claves:

- "PREGUNTA": Es el texto que representa la pregunta.
- "HUECOS": Es el número que indica cuántos huecos tiene la pregunta.
- "RESPUESTA 1", "RESPUESTA 2", "RESPUESTA 3": Es la respuesta correcta del hueco correspondiente. Las respuestas están numeradas en orden ascendente,

empezando por "RESPUESTA 1" y terminando en "RESPUESTA N", donde N es igual al número de huecos de la pregunta.

La estructura de un objeto JSON de este tipo de prueba debe verse como el del ejemplo representado en la Figura 36.

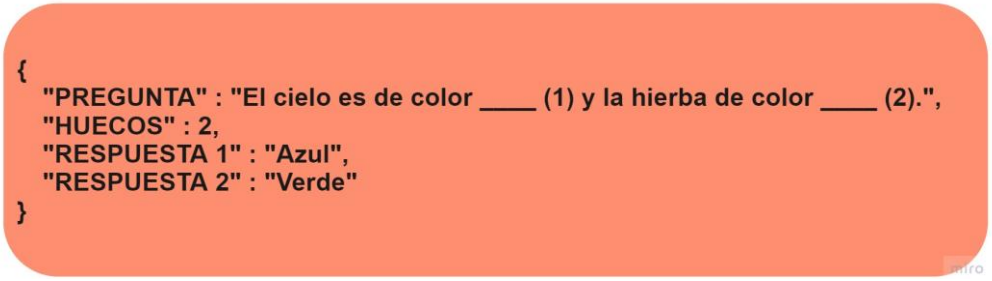


Figura 36: Formato JSON de una pregunta Fill In The Gap

5.4.3.2 Lectura del JSON

Para la lectura de estas pruebas se utiliza la clase *ReadFillInTheGap* y su método *loadQuestions*. El método *loadQuestions* (Figura 37) recibe el nombre del archivo JSON a de la prueba y carga el archivo desde el sistema de archivos interno.

Luego, itera a través de cada objeto JSON, obteniendo la pregunta y el número de espacios en blanco creando un objeto *FillInTheGapQuestion*. Además carga las respuestas correctas para cada hueco de la pregunta y lo añade al objeto FillInTheGapQuestion. Por último, agrega cada objeto *FillInTheGapQuestion* a la lista de preguntas y devuelve la lista completa.

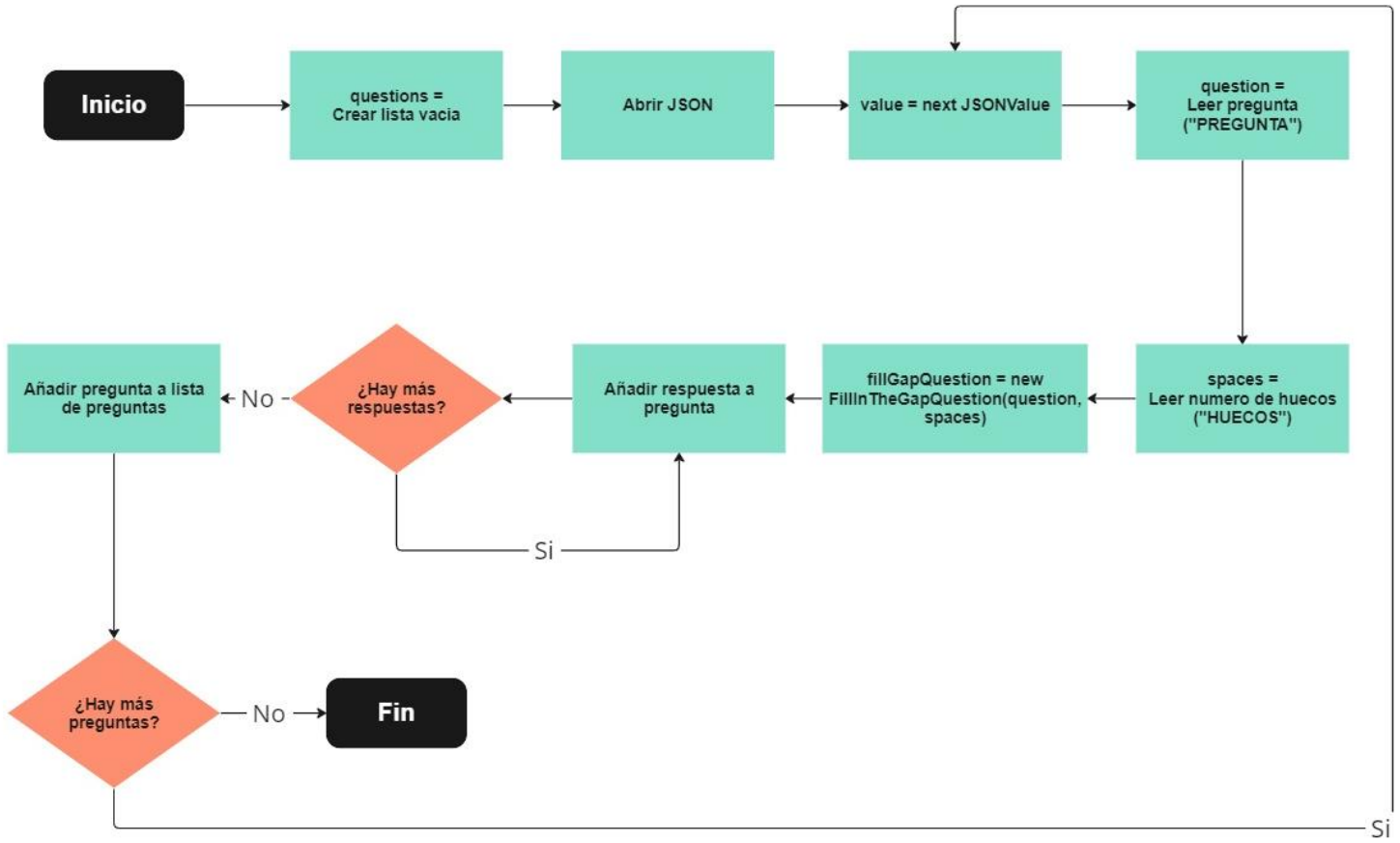


Figura 37: Diagrama de flujo del proceso de lectura de un JSON de una prueba Fill In The Gap

5.4.3.3 Display

Los componentes principales de una pregunta Fill In The Gap son:

- El título: Este título está centrado y se encuentra en la parte superior de la pantalla. Indica el numero de la pregunta en la que se encuentra el jugador.
- La pregunta: Es el texto que contiene uno o varios espacios en blanco que deben ser completados por el jugador.
- Los campos de texto: Es la zona en la que el jugador debe escribir la respuesta para el hueco correspondiente.
- Botón de siguiente pregunta: El botón de siguiente se encuentra debajo de las opciones de respuesta y se utiliza para guardar la respuesta seleccionada por el jugador y pasar a la pregunta siguiente.

Este diseño puede observarse en la Figura 38, con los sus componentes nombrados.

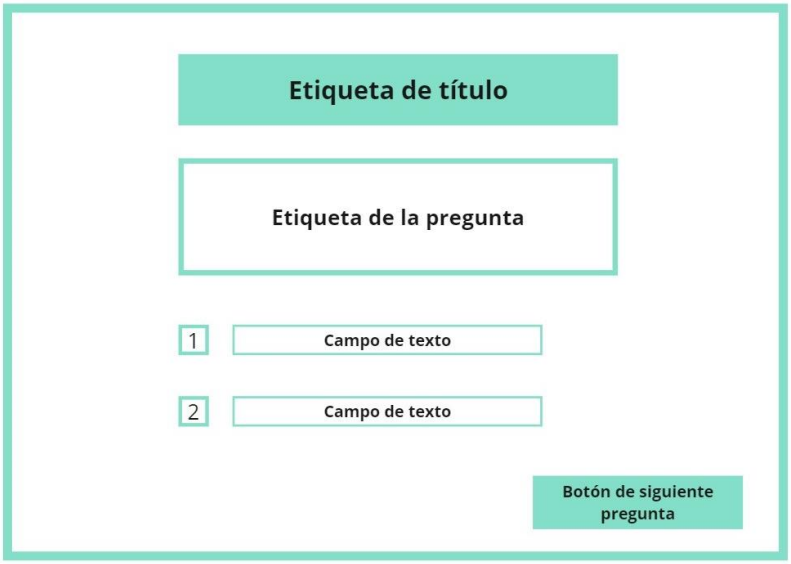


Figura 38: Diseño de una pregunta Fill In The Gap en pantalla

Este diseño puede verse implementado en una prueba real en la Figura 39.

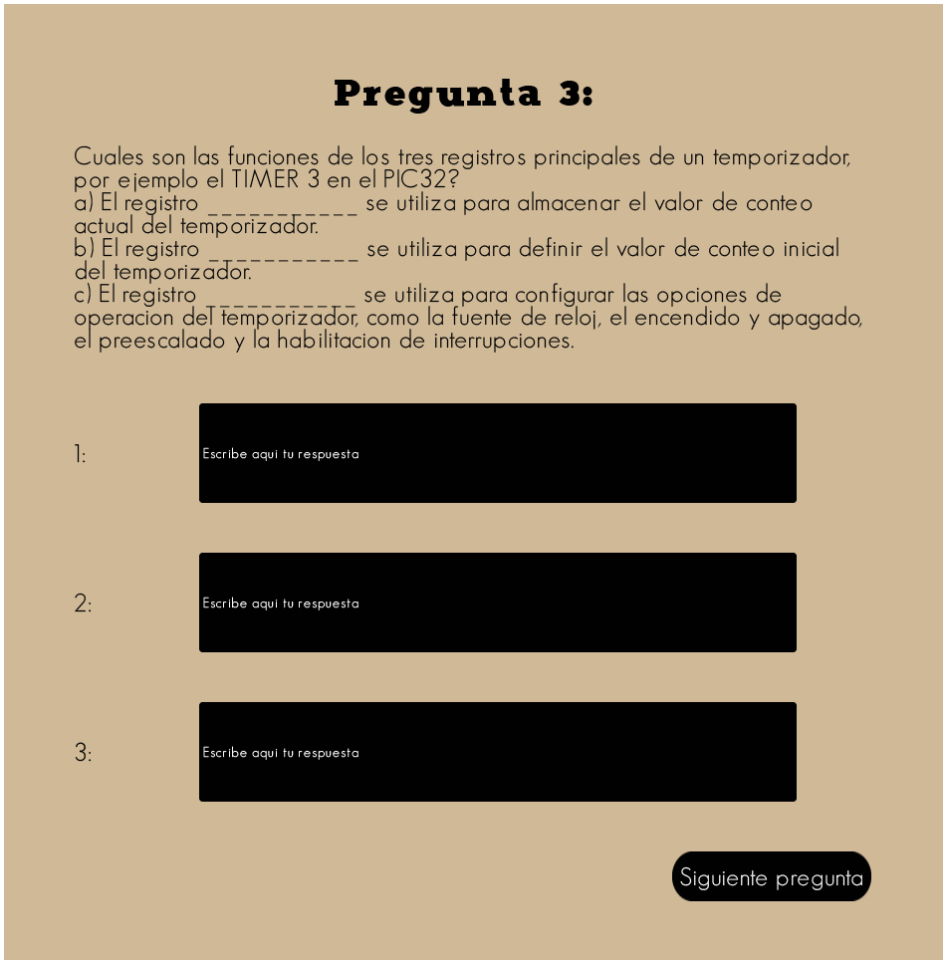


Figura 39: Captura de pantalla de una prueba Fill In The Gap

5.5 PROGRESO Y GUARDADO

El progreso y guardado del juego se gestiona desde la clase Game. Al crear el juego mediante el método *create* de la clase Game, se realiza una búsqueda del progreso del jugador, si es que existe. El método *create* funciona de la siguiente manera (Figura 40):

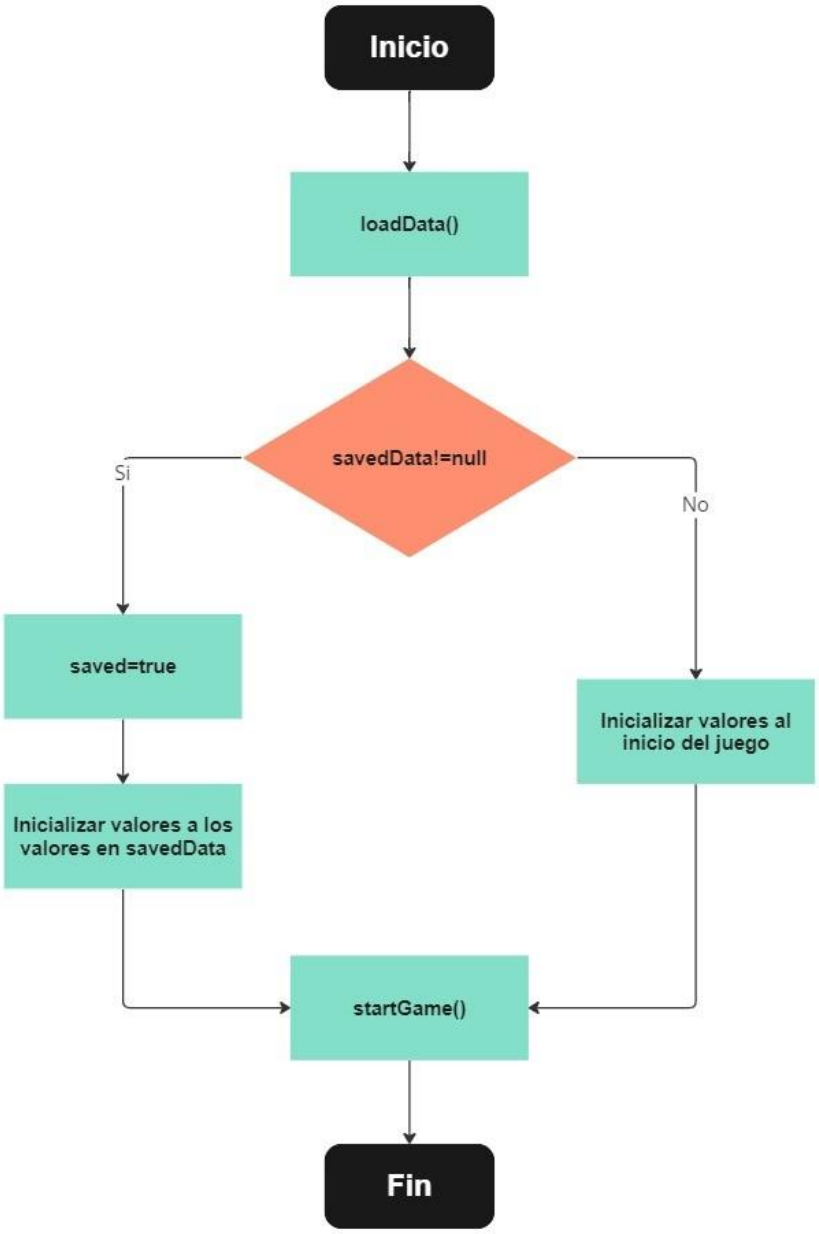


Figura 40: Diagrama de flujo del proceso de inicio del juego

En este método, se carga la información previa del juego a partir de las claves y los datos guardados. Si existe un juego previo guardado, se establece la bandera *saved* como verdadera y se cargan los valores de *nextTestNumber* y *lives* desde los datos guardados. De lo contrario, si no hay un juego previo guardado, se establecen los valores iniciales de *nextTestNumber* y *lives* como 1 y 3 respectivamente. Tras esto se invoca al método *startGame* el que se encarga

de determinar la posición, género y nombre del jugador en función de si había un juego guardado anterior o no.

El guardado de los datos de progreso también se gestiona desde la clase *Game*, en el método *saveData*. Su funcionamiento es el siguiente (Figura 41):

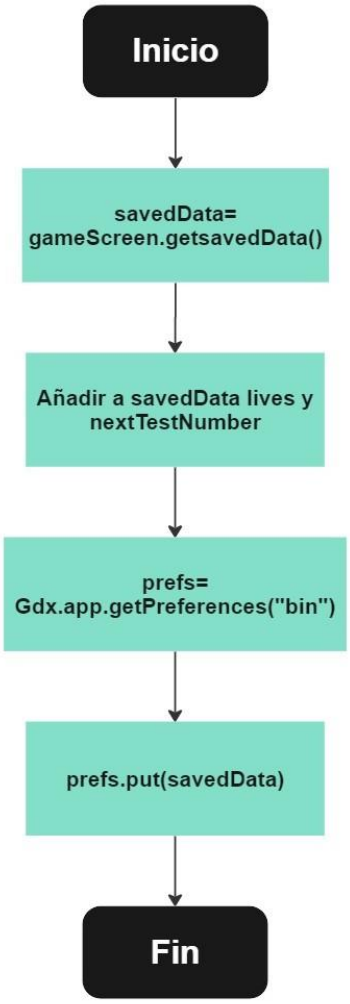


Figura 41: Diagrama de flujo del proceso de guardado de datos

En primer lugar, recoge los datos de la pantalla de juego (*gameScreen*) con el método *getSavedData*. Luego actualiza los valores de *nextTestNumber* y *lives.*

A continuación accede a un archivo local utilizando la clase *Preferences* de LibGDX llamado *bin* (Se le asigna el nombre *bin* para ocultarlo a los jugadores y evitar la manipulación de datos de progreso en el juego.). Si no existe todavía, lo crea. Finalmente se guardan los datos del progreso (*savedData*) en el archivo local.

6 ESTRUCTURA DEL JUEGO

En esta sección se describe en detalle en la estructura del software del juego. Este juego ha sido programado siguiendo los principios SOLID[28]. En ingeniería de software, SOLID es un acrónimo mnemónico introducido por Robert C. Martin que representa cinco principios básicos de la programación orientada a objetos y el diseño de software.

Estos cinco principios indican como se deben organizar funciones y estructuras de datos en componentes y como estos deben estar interconectados. Los principios son los siguientes (Figura 42):

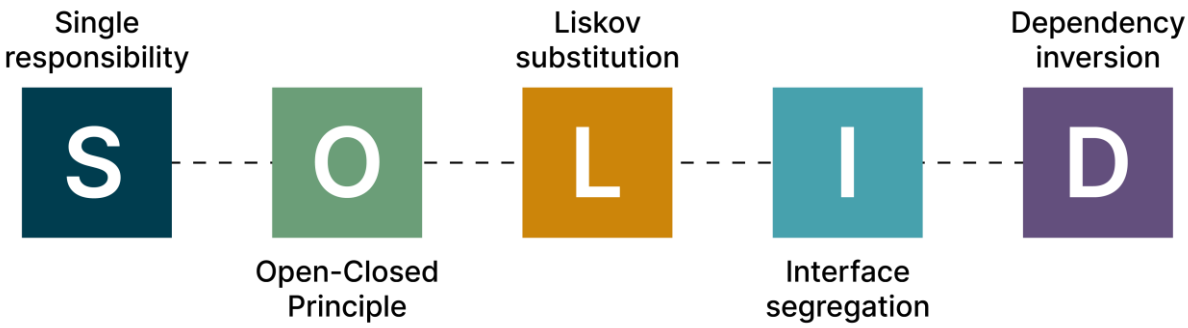


Figura 42: Los cinco principios SOLID [28]

- **Single Responsibility o Responsabilidad única**

  "Una clase debe tener una y solo una única razón para cambiar, lo que significa que una clase debe tener un solo trabajo". Es esto, precisamente, "razón para cambiar", lo que Robert C. Martin identifica como "responsabilidad".

- **Open/Closed o Abierto/Cerrado**

  "Los objetos o entidades de software deben estar abiertos para la extensión, pero cerrados para la modificación". Este principio requiere que las entidades deben estar programadas de forma que puedan ampliarse sin necesidad de modificarlas.

- **Liskov substitution o Sustitución de Liskov**
  "Cada subclase o clase derivada debe ser sustituible por su clase base o padre". Cuando una clase hereda de otra, la clase hija debe poder sustituir al padre sin que haya un mal funcionamiento

- **Interface segregation o Segregación de interfaz**
  "Los clientes de un programa solo deberían conocer de éste aquellos métodos que realmente usa, y no aquellos que no necesitan utilizar". El principio de segregación de interfaces nos indica que es mejor tener muchas interfaces más pequeñas y específicas para ciertas funciones que algunas más grandes y generales.

- **Dependency inversion o Inversión de dependencia**
  "El módulo de alto nivel no debe depender del módulo de bajo nivel, sino que deben depender de abstracciones". Martin señala además, que "las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones". Este principio requiere que el código sea escrito de forma que el módulo de alto nivel (con el que interactúa el usuario) sea independiente del código que se ejecuta a bajo nivel, y que estos se relacionen de manera abstracta.

6.1 PAQUETES

El código de este juego está dividido en distintos paquetes. Los paquetes[29] son un mecanismo de Java para facilitar la modularidad del código. La funcionalidad de una aplicación Java se implementa en múltiples clases, entre las que existen diferentes relaciones. Estas clases se agrupan en unidades de un nivel superior, los paquetes, que actúan como contenedores de tipos.

Este proyecto está dividido en siete paquetes:

- model: En este paquete se encuentran todas las clases relacionadas con los personajes.
- IO: En este paquete están todos los módulos que interactúan con archivos de input u output.
- questionTypes: En este paquete están las clases que describen los tipos de preguntas de las distintas pruebas.
- gameHelpers: Este paquete contiene clases adicionales al juego, que ayudan con procesos más complejos.
- testScreens: En este paquete se encuentran todas las pantallas relacionadas con las distintas pruebas.
- gameScreens: Este paquete tiene a todas las pantallas relacionada con el juego.
- gameApp: El paquete gameApp contiene a la clase aplicación del juego.

La estructura de paquetes (con las clases que contiene cada uno) queda representada en la Figura 43.

Esta estructura de paquetes sigue los principios SOLID que hemos visto antes:

- SRP: Cada paquete tiene una única responsabilidad. Esto implica que las clases dentro de cada paquete están relacionadas y son responsables de un conjunto coherente de funcionalidades. Al agrupar las clases relacionadas por su responsabilidad en un paquete, se facilita la comprensión y el mantenimiento del código.
- OCP: Esta estructura permite la extensión sin modificación. Los cambios o adiciones de nuevas funcionalidades son posibles mediante creación de nuevos paquetes sin necesidad de modificar los ya existentes.
- LSP: La estructura de paquetes sigue el principio de sustitución de Liskov, donde las clases derivadas puede ser utilizadas en lugar de las clases base sin alterar el comportamiento esperado del programa.
- ISP: Los módulos de cada paquete están diseñados para ser utilizadas únicamente por las clases que requieren esa funcionalidad especifica

- DIP: Los paquetes definen interfaces y clases que pueden ser utilizadas por otros paquetes sin conocer detalles de implementación.

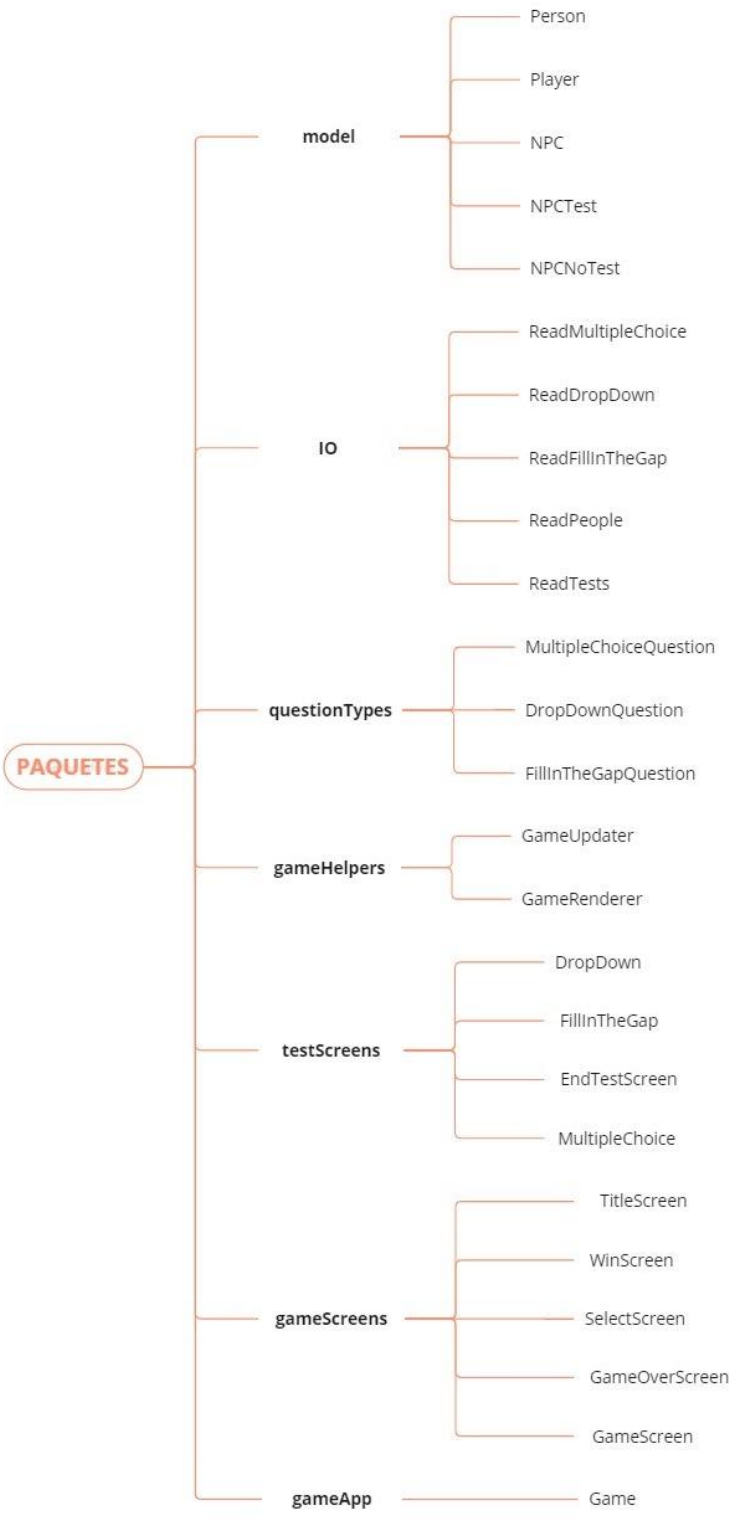


Figura 43: Estructura de paquetes del proyecto

6.2 CLASES

En esta sección se realiza un análisis en detalle de las clases que conforman el juego. Se explora la funcionalidad y responsabilidades de cada una de ellas. Este desglose proporciona una comprensión exhaustiva de la arquitectura y estructura del juego.

6.2.1 GAME

La clase *Game* se encuentra en el paquete *gameApp*. Esta clase es la clase principal y es la encargada de gestionar las diferentes pantallas del juego y el estado del juego en sí. También gestiona la carga y almacenamiento del progreso del juego, la reproducción de música de fondo y la transición entre pantallas.

Esta clase proporciona métodos para iniciar el juego, iniciar una prueba, finalizar una prueba, gestionar el final del juego o reiniciarlo.

6.2.2 GAMESCREEN

Esta clase se encuentra en el paquete *gameScreens*. Esta clase es responsable de iniciar la pantalla de juego, renderizar el mundo del juego y manejar la lógica interna de este. Carga el mapa, se encarga del jugador, los NPCs, la cámara y las colisiones. Además proporciona métodos para interacciones con NPCs, mostrar diálogos y administrar las vidas del jugador.

También utiliza clases de ayuda para el renderizado y actualización del juego. Y por último permite al usuario cambiar ciertas configuraciones del juego como el control del volumen y la capacidad de guardar y cerrar el juego.

6.2.3 TITLESCREEN

La clase *TitleScreen* se encuentra en el paquete *gameScreens*. Esta clase representa la pantalla de inicio del juego. En esta pantalla se muestra una imagen de título, un botón de inicio y un botón de reinicio de juego si existe un juego guardado previamente.

Esta pantalla está pensada para mejorar la experiencia del jugador al presentarle un diseño visual y atractivo.

6.2.4 SELECTSCREEN

Esta clase está en el paquete *gameScreens*. *SelectScreen* es una clase que representa la pantalla de creación de personaje del juego. Permite al jugador elegir su género y su nombre antes de iniciar el juego. Esto permite al jugador personalizar un poco a su personaje.

El botón de inicio del juego solo está habilitado si se ha ingresado un nombre, lo que evita que el jugador inicie el juego sin proporcionar información necesaria.

6.2.5 GAMEOVERSCREEN

Esta clase se encuentra en el paquete *gameScreens*. Esta clase representa la pantalla de *Game Over* del juego. Esta pantalla se muestra cuando el jugador pierde todas sus vidas. Contiene un botón de reinicio del juego un botón de cierre y una etiqueta que indica al jugador que ha perdido.

6.2.6 WINSCREEN

Esta clase se encuentra en el paquete *gameScreens*. Esta clase representa la pantalla de final del juego. Esta pantalla se muestra cuando el jugador supera todas las pruebas. Contiene un botón de reinicio del juego un botón de cierre y una etiqueta que indica al jugador que ha ganado.

6.2.7 GAMEUPDATER

La clase *GameUpdater* se encuentra en el paquete *gameHelpers*. Esta clase se encarga de actualizar el estado del juego en función de la entrada del jugador y su posición. Esta clase se inicializa dentro de la clase *GameScreen*, y es la que gestiona movimientos del jugador, colisiones e interacciones.

6.2.8 GAMERENDERER

Esta clase está en el paquete *gameHelpers*. Esta clase se encarga de renderizar el mapa y los personajes en la pantalla de juego. Al igual que la clase *GameUpdater*, esta clase se inicializa dentro de *GameScreen*.

6.2.9 PERSON

La clase *Person* se encuentra en el paquete *model*. *Person* es una clase abstracta que representa a una persona en el juego. Contiene información como la posición de la persona y tu textura y permite dibujar a la persona en el mapa con el método *draw*. Esta clase proporciona una estructura o esqueleto base para crear personajes en el juego.

Al ser una clase abstracta, se puede heredar de ella para crear diferentes tipos de personajes con comportamientos y características específicas.

6.2.10 PLAYER

Esta clase está contenida en el paquete *model*. Player es una clase que representa al jugador. Hereda de la clase *Person* y le agrega funcionalidades específicas, como moverse, cambiar de dirección y una animación del movimiento.

Esta clase representa al personaje controlado por el jugador. Es esencial para el funcionamiento del juego, ya que representa la interacción directa del jugador con el entorno del juego.

6.2.11 NPC

La clase *NPC* está en el paquete *model*. Esta clase abstracta representa a un personaje no jugador en el juego. *NPC* hereda de la clase *Person*, y añade una propiedad adicional (el nombre del personaje). Al ser una clase abstracta, proporciona una base común para la creación de diferentes tipos de NPCs con comportamientos y características específicas.

6.2.12 NPCTEST

Esta clase se encuentra en el paquete *model*. *NPCTest* representa a los NPCs que tienen una prueba asociada. Esta clase hereda de *NPC* y agrega propiedades adicionales como el número de prueba asociada o el dialogo del NPC.

6.2.13 NPCNOTEST

Esta clase se encuentra en el paquete *model*. Esta clase representa a un personaje no jugable que no tiene una prueba asociada. La clase *NPCNoTest* también hereda de *NPC* y agrega una propiedad que guarda el dialogo del NPC.

6.2.14 MULTIPLECHOICE

La clase *MultipleChoice* está contenida en el paquete *testScreens*. Esta clase representa un juego de pregunta de opción múltiple. Esta clase se encarga de cargar las preguntas y respuestas desde un archivo y mostrarlas por pantalla. Cuando se responde a todas las preguntas se verifican las respuestas para comprobar resultados.

6.2.15 DROPDOWN

Esta clase está en el paquete *testScreens*. Esta clase representa un juego de preguntas de opción en formato de menú despegable. Se encarga de cargar las preguntas y mostrarlas por pantalla para que el usuario pueda ir respondiendo a las preguntas. Una vez terminada la prueba se verifican las respuestas y se llega a un resultado.

6.2.16 FILLINTHEGAP

Esta clase se encuentra en el paquete *testScreens*. Esta case representa un juego de completar espacios en blanco. Esta clase carga las preguntas desde un archivo y las muestra en la pantalla. Al finalizar de contestar a todas las preguntas se chequean las respuestas llegando a un resultado de la prueba.

6.2.17 ENDTESTSCREEN

Esta clase está en el paquete *testScreens*. *EndTestScreen* representa la pantalla de final de prueba, su finalidad es mostrar un mensaje de felicitación si el jugador ha superado la prueba o mostrar un mensaje indicando que el jugador ha perdido una vida junto con los resultados de la prueba.

6.2.18 MULTIPLECHOICEQUESTION

La clase *MultipleChoiceQuestion* se encuentra en el paquete *questionTypes*. Esta clase representa una pregunta de opción múltiple con cuatro opciones y una respuesta correcta. Permite almacenar la pregunta, las opciones de respuesta y la respuesta correcta. También proporciona métodos para acceder a estos datos.

6.2.19 DROPDOWNQUESTION

Esta clase se encuentra en el paquete *questionTypes*. Esta clase representa una pregunta de selección desplegable en un juego de preguntas o cuestionario. Permite construir preguntas de selección desplegable con diferentes configuraciones de espacios en blanco y opciones, y proporciona métodos para acceder y modificar los datos de la pregunta.

6.2.20 FILLINTHEGAPQUESTION

Esta clase se encuentra en el paquete *questionTypes*. *FillInTheGapQuestion* representa una pregunta de completar espacios en blanco en un juego de preguntas o cuestionario. Contiene el texto de la pregunta con los espacios en blanco representados por guiones bajos, y una lista de respuestas correctas para la pregunta. Permite construir preguntas de completar espacios en blanco con diferentes configuraciones de espacios y proporciona métodos para acceder y modificar los datos de la pregunta.

6.2.21 READMULTIPLECHOICE

La clase *ReadMultipleChoice* está dentro del paquete IO. Esta clase es responsable de cargar un conjunto de preguntas de opción múltiple (*MultipleChoiceQuestion*) desde un archivo JSON. Esta clase se inicializa desde la clase *MultipleChoice*.

6.2.22 READDROPDOWN

La clase *ReadDropDown* se encuentra en el paquete IO. Esta clase es responsable de cargar un conjunto de preguntas de selección de opciones desplegables (*DropDownQuestion*) desde un archivo JSON. Esta clase se inicializa desde la clase *DropDown*.

6.2.23 READFILLINTHEGAP

La clase *ReadFillInTheGap* está dentro del paquete IO. Esta clase es responsable de cargar un conjunto de preguntas de rellenar espacios en blanco (*FillInTheGapQuestion*) desde un archivo JSON. Esta clase se inicializa desde la clase *FillInTheGap*.

6.2.24 READPEOPLE

Esta clase se encuentra dentro del paquete *IO*. Esta clase es responsable de cargar los personajes (NPCs) desde un archivo JSON y devolverlos. Los NPCs se cargan en un *ArrayList* de objetos NPC y este se devuelve como resultado. Además, la clase también crea objetos *NPCNoTest* que son NPCs sin pruebas asociadas y los agrega al *ArrayList*.

6.2.25 READTESTS

Esta clase se encuentra dentro del paquete *IO*. *ReadTests* gestiona la carga de las pruebas desde un archivo de texto y devolverlas. Lee el archivo de texto línea por línea y divide cada línea en campos separados por comas. Los campos contienen información sobre el número de prueba, el tipo de prueba y el nombre de la prueba. Esta información se almacena en un *HashMap*, donde la clave es el número de prueba y el valor es un *ArrayList* que contiene el tipo y el nombre de la prueba.

6.3 UML

En esta sección se presentan una serie de diagramas UML que explican las relaciones entre las distintas clases o tipos de clases de este proyecto. Los diagramas UML son herramientas visuales que permiten comprender la estructura, el comportamiento y el diseño de un proyecto software.

Cada uno de los diagramas que se incluyen en esta sección están enfocados en un aspecto específico del sistema. Estos diagramas están simplificados para ser claros y concisos, y su objetivo es facilitar la comprensión de las relaciones entre los componentes y el diseño del proyecto.

Para entender esta sección se debe conocer el significado de alguna de las relaciones entre clases y su representación gráfica[26]. Las cuatro más importantes de este proyecto son las siguientes (Figura 44):

- **Asociación:**
  Indica que una clase tiene una referencia a una instancia (o instancias) de otra clase a través de una propiedad.
- **Composición:**
  Representa la relación entre el todo y la parte de la clase, y el total y la parte tienen una duración constante. Es decir una clase necesita de la otra para existir.
- **Herencia:**
  También conocida como generalización, describe la relación entre las clases padre e hijo. En la herencia, la subclase hereda todas las funciones de la clase principal y la clase principal tiene todos los atributos, métodos y subclases.
- **Dependencia:**
  Indica que un cambio en la clase A provoca un cambio en la clase B, lo que significa que la clase B depende de la clase A. Las dependencias se reflejan en los métodos de una clase que utilizan el objeto de otra clase como parámetros.

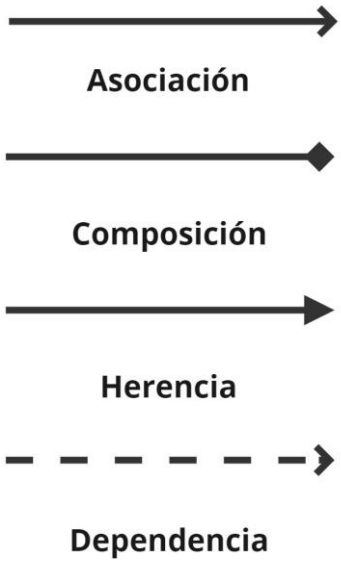


Figura 44: Relaciones de clase en un UML

En la Figura 45 se representa el UML entre la clase principal *Game*, y las clases del paquete *gameScreens*. Todas estas clases se comportan igual por lo que el diagrama esta resumido de forma que la clase *Screen* se utiliza para representar todas las clases del paquete *gameScreens* de una manera generalizada.

En el diagrama (Figura 45) se muestra como la clase *Game* tiene una relación de composición con *Screen*, lo que significa que *Game* depende de la existencia de *Screen* para funcionar correctamente, ya que *Game* es el encargado de la transición entre las distintas pantallas, y por ello necesita conocerlas para llevar a cabo esta tarea.

Asimismo, la relación de composición también aplica en la dirección opuesta, ya que *Screen* necesita conocer a *Game* para poder acceder a métodos de este como el guardado de datos o el cerrado del juego. La clase *Screen* utiliza la instancia de *Game* para invocar estos métodos y llevar a cabo acciones importantes relacionadas con el funcionamiento global del juego.

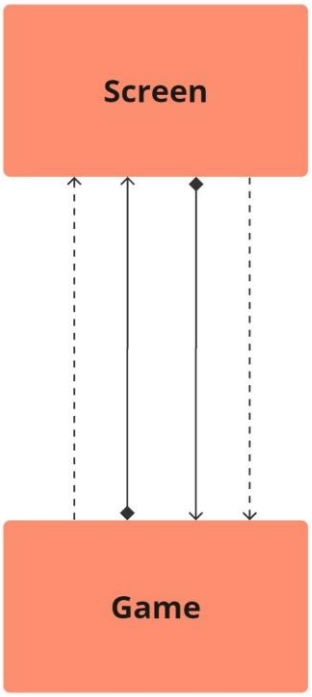


Figura 45: Diagrama UML de la clase Game y el paquete gameScreens

En la Figura 46 se muestra un diagrama UML que representa las relaciones entre la clase *GameScreen* (la pantalla de juego) y todas las clases que contiene el paquete *model*. Este diagrama muestra como la clase *Person* actúa como clase Padre del paquete *model* y el resto heredan de ella. Específicamente, las clases *NPC* y *Player* son clases hijas de *Person*, y a su vez, la clase *NPC* actúa como clase padre para las clases *NPCTest* y *NPCNoTest*.

Además, la pantalla de juego (*GameScreen*), contiene instancias de todas estas clases ya que al contener a un jugador de la clase *Player*, también implica la inclusión de la clase *Person*. Esto sigue el principio LSP (*Liskov Substitution Principle*) de los principios SOLID.

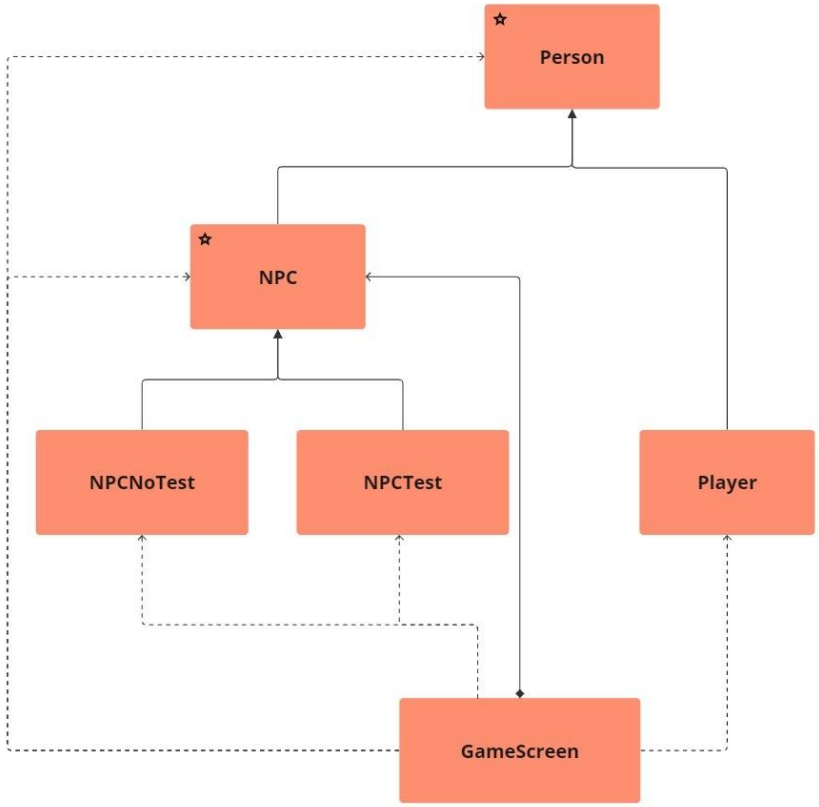


Figura 46: Diagrama UML de la clase GameScreen y el paquete model

En el siguiente diagrama (Figura 47), se representa un UML de la clase *GameScreen* y el paquete *gameHelpers*. Este diagrama es de naturaleza simple ya que consta de tan solo tres clases. *GameScreen* requiere de la existencia de ambas clases del paquete para funcionar, y por ello están en un paquete llamado *gameHelpers*, debido a que brindan soporte y asistencia a la clase *GameScreen* para llevar a cabo sus funciones de manera correcta.

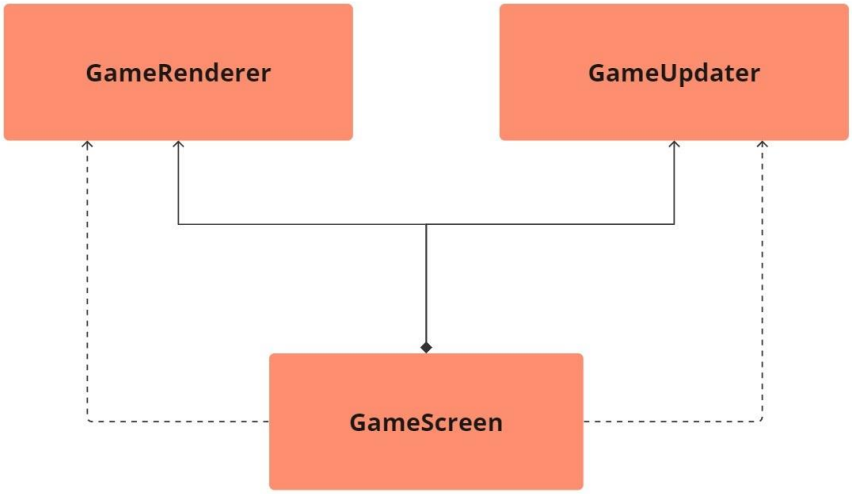


Figura 47: Diagrama UML de la clase GameScreen y el paquete gameHelpers

Por último, en la Figura 48 está representado el diagrama UML de la clase *Game* y todas las clases relacionadas con las pruebas, es decir el paquete *testScreens* y *questionTypes* al completo, y parte del paquete *IO*.

Para simplificar el diagrama, se ha condensado la representación los tres tipos de prueba (*MultipleChoice*, *DropDown* y *FillInTheGap*) en un solo ejemplo llamado *Test* ya que comparten las mismas relaciones con Game y entre sí.

La relación entre *Game* y *EndTestScreen* es similar a la relación entre *Game* y *Screen* que se mostró en el primer diagrama, ya que *EndTestScreen* es otra pantalla del sistema.

En cuanto a la clase *Game* solo existe la relación de composición en un sentido con *Test*. *Test* necesita la existencia de *Game* para existir y realizar su función al completo. A su vez *Test* necesita de *TestQuestion* para existir, ya que una prueba sin preguntas carece de sentido.

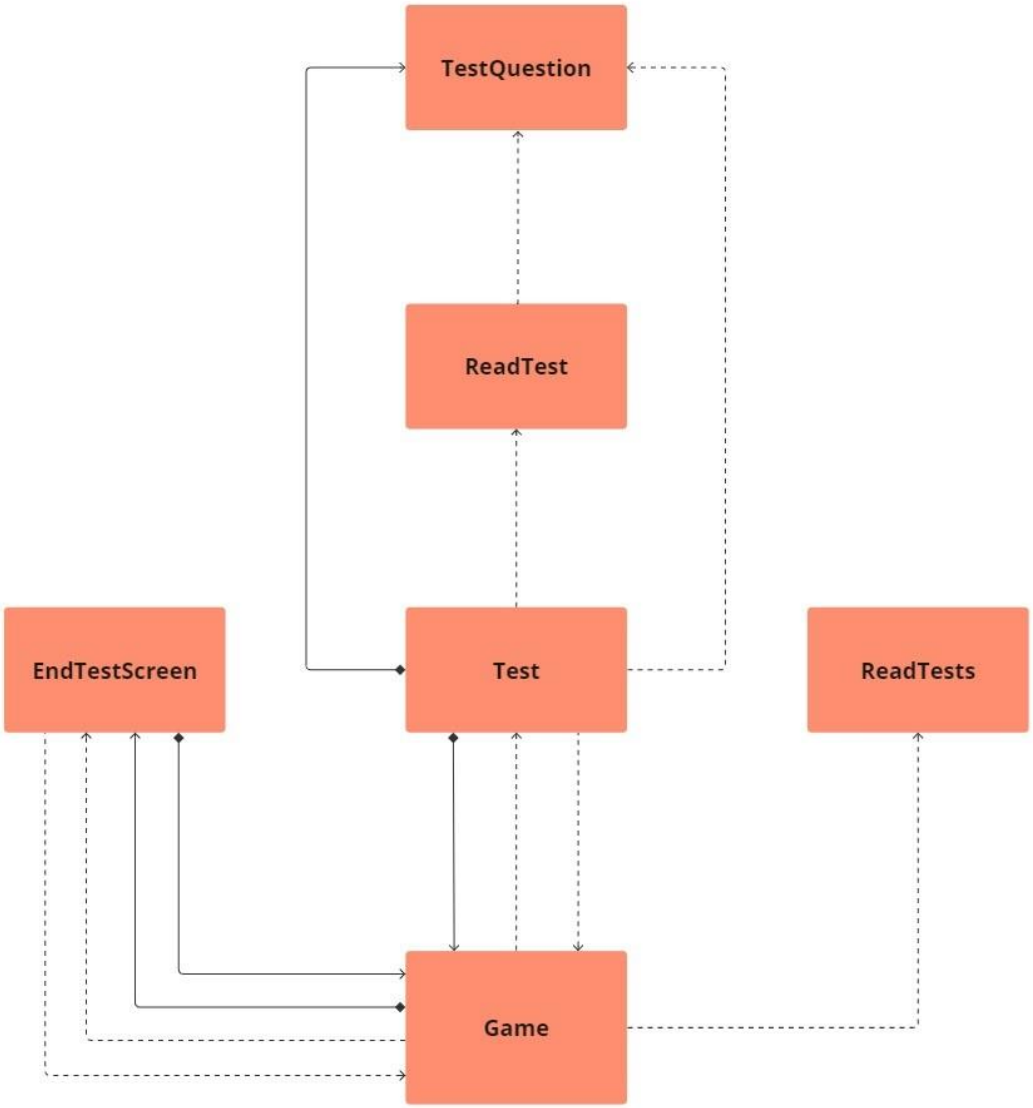


Figura 48: Diagrama UML de la clase Game y los paquetes testScreens, questionTypes e IO

7 RESULTADOS

La sección final de resultados del proyecto desarrollado brinda una visión general del juego. Se presentan las diversas pantallas que componen la experiencia del jugador. A través de esta recopilación de pantallas, se pueden apreciar los resultados obtenidos en el proyecto y los avances logrados en términos de diseño, funcionalidad, accesibilidad y jugabilidad.

Así se puede evaluar el rendimiento del juego y observar cómo se han materializado los elementos y conceptos previamente descritos

Con el inicio del juego, se presenta la pantalla de inicio del juego (Figura 49).

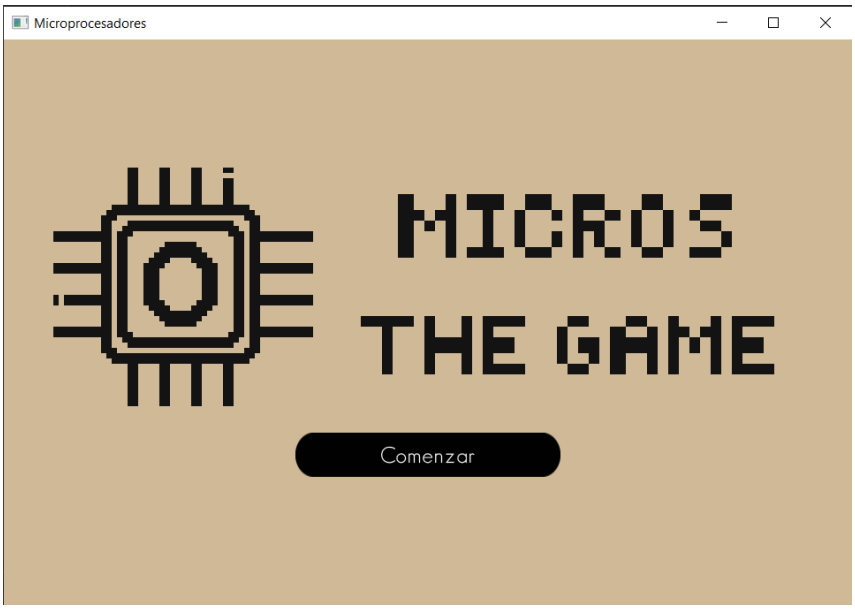


Figura 49: Pantalla de inicio del juego

Tras pulsar el botón de comenzar, se muestra una pantalla de selección (Figura 50), que se utiliza para iniciar una nueva partida para el usuario. En esta pantalla el jugador debe introducir su nombre y el género de su personaje dentro del juego.

Figura 50: Pantalla de selección del juego

Cuando el usuario introduce las entradas necesarias y pulsa el botón de Iniciar Juego, se inicia la pantalla de juego (Figura 51), creando una nueva partida.

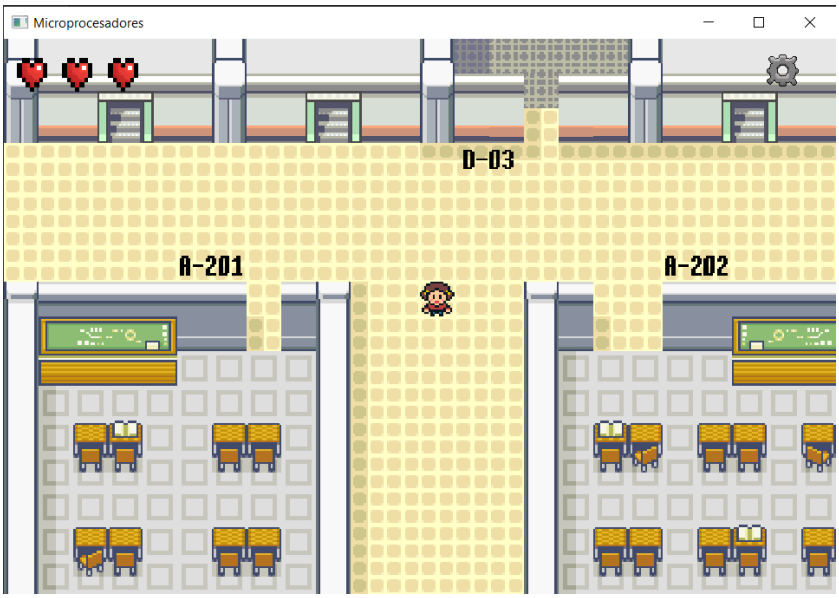


Figura 51: Pantalla de juego

En el caso de que exista ya una partida creada anteriormente, es decir, haya un progreso de juego ya guardado, la pantalla de inicio seria distinta (Figura 52). Esta pantalla permite al

jugador tanto seguir con su partida anterior, o iniciar una nueva partida eliminando su progreso anterior.

Figura 52: Pantalla de inicio con progreso guardado

El siguiente paso es interactuar con los personajes secundarios. Cuando el jugador interactúa con un personaje, este inicia un cuadro de dialogo por pantalla (Figura 53).

Figura 53: Pantalla de juego con cuadro de diálogo

Si el jugador decide iniciar la prueba, esta se muestra en pantalla (Figura 54).

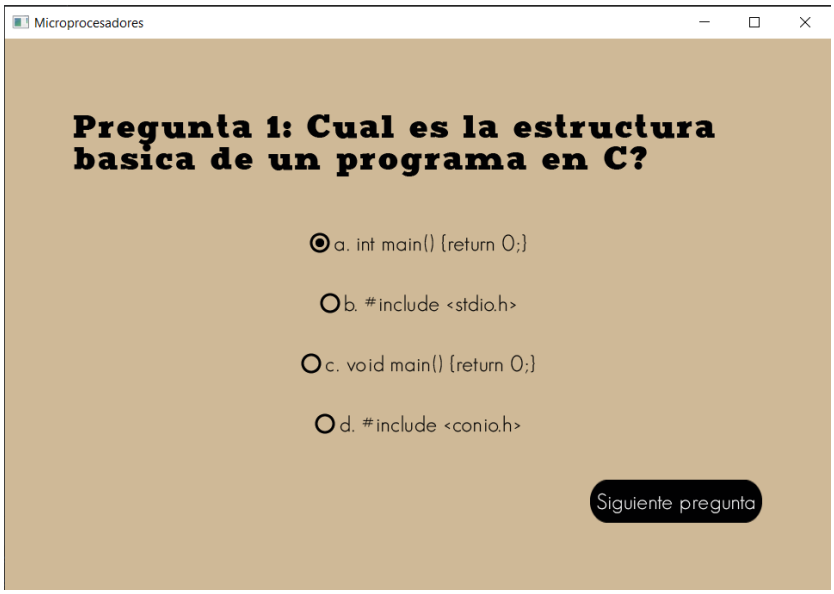


Figura 54: Pantalla de prueba

Tras terminar la prueba, existen dos posibles alternativas:

- El jugador supera la prueba (Figura 55).

Figura 55: Pantalla de prueba superada

- El jugador falla alguna de las preguntas de la prueba llevándolo a perder una vida, y a una pantalla de fin de prueba (Figura 56) en la que se le muestran los resultados de la prueba realizada.

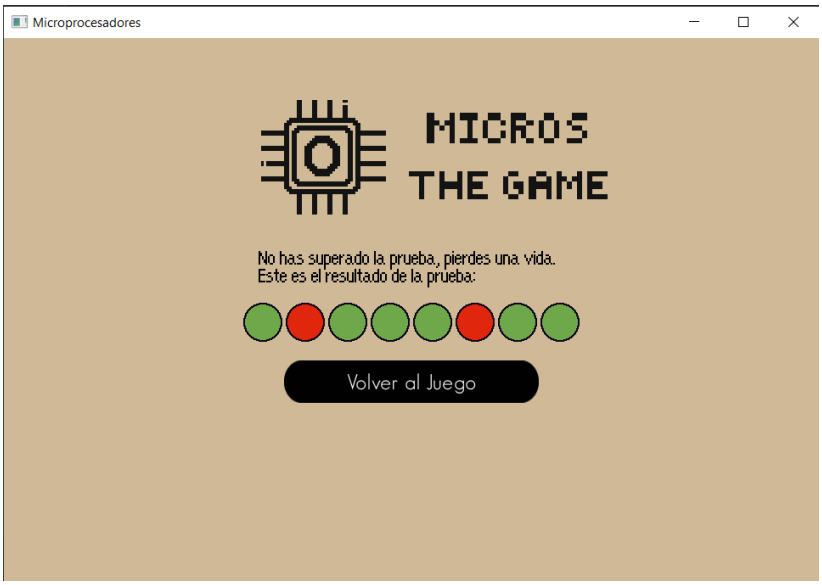


Figura 56: Pantalla de prueba fallida

Así va avanzando el usuario en el juego. Al fallar una prueba, el jugador pierde una vida (de las 3 con las que empieza el juego), y en caso de perderlas todas, perdería el juego (Figura 57), llevándolo a tener que iniciar la partida desde el inicio y volver a superar las pruebas desde la primera.

Figura 57: Pantalla de GameOver

Finalmente, si el jugador supera todas las pruebas, sin quedarse sin vidas, el jugador gana el juego (Figura 58).

Figura 58: Pantalla victoria, fin del juego

El resultado final entonces es un juego completamente funcional. El siguiente video () muestra una demo del juego para comprender y apreciar mejor el resultado:

DEMO.mp4

Archivo 1: Demo en video del juego

8 CONCLUSIONES Y TRABAJOS FUTUROS

El desarrollo del proyecto ha sido un éxito en términos de implementación de un juego funcional ya que cumple con su objetivo principal.

Durante el desarrollo del juego, una de las principales dificultades técnicas ha sido la implementación de un sistema de detección de colisiones preciso y confiable ya que se probaron multitud de maneras distintas antes de llegar a la correcta.

Otro aspecto a destacar ha sido la estructura del juego y la organización de las tareas. Para gestionar esto se dividieron las tareas en distintas etapas o partes del proyecto. Esto garantizo un progreso constante y un enfoque centrado en cumplir los objetivos marcados, evitando retrasos y asegurando la finalización exitosa del juego.

El juego desarrollado ha logrado cumplir con los requisitos y objetivos establecidos demostrando un funcionamiento sólido. Se reconoce y agradece la contribución del director del proyecto, cuya orientación y apoyo han sido fundamentales para lograr estos resultados.

En cuanto a los posibles trabajos futuros, se plantea la posibilidad de aplicar esta idea de proyecto en otras áreas/signaturas académicas, para así aprovechar el potencial educativo del juego desarrollado. Además, se sugiere la ampliación del juego, incluyendo NPCs más realistas, pruebas más elaboradas y la implementación de características adicionales como la posibilidad de volver a jugar pruebas anteriores o la inclusión de misiones secundarias que te permitan recuperar vidas.

9 BIBLIOGRAFÍA

[1] Educación 3.0. "Gamificación: qué es y objetivos". Educación 3.0, 27 de mayo de 2023. https://www.educaciontrespuntocero.com/noticias/gamificacion-que-es-objetivos/. Visitado en: 25/11/2022

[2] Fundación Aquae. "¿Qué es gamificación?". https://www.fundacionaquae.org/wiki/que-es-gamificacion/ Visitado en: 25/11/2022

[3] BySpel. "Programación orientada a objetos en Java (POO en Java)". https://byspel.com/programacion-orientada-a-objetos-en-java-poo-en-java/. Visitado en: 10/01/2023

[4] Asociación AEPI. "Programación orientada a objetos en Java". https://asociacionaepi.es/programacion-orientada-a-objetos-en-java/. Visitado en: 10/01/2023

[5] Web Design Cusco. "Los 4 principios fundamentales de la Programación Orientada a Objetos (POO)". https://webdesigncusco.com/los-4-principios-fundamentales-de-la-programacion-orientada-a-objetos-poo/. Visitado en: 10/01/2023

[6] LibGDX. "LibGDX". https://libgdx.com/

[7] Code and Coke. "Apuntes: LibGDX". https://multimedia.codeandcoke.com/apuntes:libgdx Visitado en: 24/03/2023

[8] Tiled Map Editor. "Tiled Map Editor". https://www.mapeditor.org/

[9] Genbeta. "Tiled Map Editor, el editor de mapas libre". https://www.genbeta.com/desarrollo/tiled-map-editor-el-editor-de-mapas-libre Visitado en: 24/03/2023

[10] Kahoot. https://kahoot.com/

[11] Xataka. "Kahoot: qué es, para qué sirve y cómo funciona". https://www.xataka.com/basics/kahoot-que-es-para-que-sirve-y-como-funciona Visitado en: 20/11/2022

[12] Socrative. https://www.socrative.com/

[13] Educación 3.0. "¿Qué es Socrative?" https://www.educaciontrespuntocero.com/recursos/que-es-socrative/ Visitado en: 20/11/2022

[14] iCuadernos. http://www.icuadernos.com/

[15] Compartir Palabra Maestra. "iCuadernos: la aplicación infantil que convierte el aprendizaje en diversión". https://www.compartirpalabramaestra.org/recursos/herramientas-tic/icuadernos-la-aplicacion-infantil-que-convierte-el-aprendizaje-en-diversion Visitado en: 20/11/2022

[16] Classcraft. "Classcraft". https://www.classcraft.com/es-es/

[17] CodeCombat. https://codecombat.com/

[18] La Mirada del Replicante. "Aprende a programar jugando con CodeCombat". La Mirada del Replicante, 30 de enero de 2015. https://lamiradadelreplicante.com/2015/01/30/aprende-a-programar-jugando-con-codecombat/ Visitado en: 20/11/2022

[19] Luden.io. "What The Func?". https://luden.io/wtl/ Visitado en: 20/11/2022

[20] GitHub. https://github.com/

[21] Wikipedia. "Pokémon Emerald". https://en.wikipedia.org/wiki/Pok%C3%A9mon_Emerald Visitado en: 02/12/2022

[22] CodeMonkey. https://www.codemonkey.com/

[23] EduTech. "CodeMonkey: Una herramienta para aprender a programar". EduTech, 27 de enero de 2022. https://www.edutechca.com/soluciones/2022-01-27-codemonkey/ Visitado en: 20/11/2022

[24] Yelp. "El Gruñidor, Madrid". https://www.yelp.com/biz/el-gru%C3%B1idor-madrid Visitado en: 01/06/2023

[25] Pixilart. "Pixilart - Online Pixel Drawing Tool". https://www.pixilart.com/draw#

[26] Universidad de Granada. "Relaciones en Java". Departamento de Ciencias de la Computación e Inteligencia Artificial (DECSAI). http://elvex.ugr.es/decsai/java/pdf/3c-relaciones.pdf Visitado en: 13/05/2023

[27] Avantgarde IT. "5 principios del desarrollo SOLID que debes conocer". https://avantgardeit.es/5-principios-desarrollo-solid-que-debes-conocer/ Visitado en: 13/05/2023

[28] Fusiona. "Principios SOLID: ¿Por qué usarlos en tu desarrollo de software?". https://fusiona.cl/blog/tecnologia/principios-solid-por-que-usarlos-en-tu-desarrollo-de-software#:~:text=Los%20Principios%20Solid%20indican%20c%C3%B3mo,s%C3%B3lo%20sean%20aplicables%20a%20ellas Visitado en: 13/05/2023

[29] CampusMVP. "Paquetes en Java: ¿Qué son, para qué se utilizan y cómo se usan?". https://www.campusmvp.es/recursos/post/paquetes-en-java-que-son-para-que-se-utilizan-y-como-se-usan.aspx#:~:text=Los%20paquetes%20son%20el%20mecanismo,distribuy%C3%A9ndose%20habitualmente%20como%20un%20archivo. Visitado en: 15/05/2023

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Este proyecto de desarrollo de un juego interactivo se alinea con varios de los Objetivos y Metas del Desarrollo Sostenible (ODS):

1. Educación de calidad (Figura 59): El juego desarrollado es una herramienta educativa que fomenta el aprendizaje y la adquisición de habilidades académicas por parte de los jugadores. A través de la resolución de pruebas, los jugadores desarrollan habilidades cognitivas, lógicas y de resolución de problemas.

Figura 59: Objetivo del Desarrollo Sostenible número 4

2. Industria, innovación e infraestructura (Figura 60): La creación del juego, implica el uso de tecnología y el desarrollo de infraestructura digital. El proyecto ha requerido de utilización de herramientas software lo que promueve la innovación y el avance de la industria tecnológica.

Figura 60: Objetivo del Desarrollo Sostenible número 9

3. Producción y consumo responsables (Figura 61): Durante el desarrollo del juego se han adoptado prácticas de optimización de recursos como la eficiencia en el uso de la memoria y el rendimiento del juego.

Figura 61: Objetivo del Desarrollo Sostenible número 12

ANEXO II: MANUAL DEL JUGADOR

El juego sumerge a los jugadores en una serie de pruebas que abarcan los temas fundamentales impartidos en el curso de Microprocesadores. Estas pruebas se encuentran estratégicamente dispersas por todo el mapa virtual y deben ser completadas a través de interacciones con los personajes no jugadores (NPC). Cada prueba debe ser superada en un orden secuencial. Al completar una prueba, se desbloquea la siguiente en la secuencia. La culminación exitosa de todas las pruebas es requisito indispensable para alcanzar el final del juego. Este enfoque secuencial brinda a los jugadores una experiencia de aprendizaje estructurada y desafiante, que les permite adquirir un dominio profundo de los conceptos y habilidades relacionados con el mundo de los microprocesadores. Para ejecutar el juego abre el CMD y ejecuta esta línea de código donde tengas guardado el archivo JAR del juego (Archivo 2):

```
java -jar gamificacionDeMicroprocesadores.jar
```


gamificacionDeMicroprocesadores.jar

Archivo 2: Archivo jar del juego

Pantalla Inicial

Cuando inicies el juego, se te presentará la pantalla inicial, donde tendrás la oportunidad de personalizar tu experiencia. En primer lugar, podrás asignar un nombre a tu personaje, permitiéndote darle una identidad única y personal. Además, se te proponen dos opciones de género: "Chico" o "Chica", brindándote la libertad de elegir la representación que más te identifique.

Es importante tener en cuenta que una vez que hayas tomado la decisión de tu género y hayas comenzado el juego, no será posible cambiar ni el género de tu personaje ni su nombre. Por lo tanto, te recomendamos considerar cuidadosamente tus preferencias antes de iniciar tu aventura en el mundo de Microprocesadores. Una vez hayas tomado todas las decisiones pertinentes, estarás listo para adentrarte en el emocionante desafío que te espera. ¡Que comience la exploración y el aprendizaje!

Controles

A continuación, se presenta una descripción de los controles disponibles en el juego:

- Movimiento del personaje:
  Al ingresar al juego, tu personaje se ubicará en el centro de la pantalla. Para desplazarte en las cuatro direcciones cardinales, puedes utilizar las teclas W, A, S y D en tu teclado. La tecla W te permitirá moverte hacia arriba, la tecla A hacia la izquierda, la tecla S hacia abajo y la tecla D hacia la derecha. Esto puede verse mejor en la Tabla 3.

TECLA	**DIRECCIÓN**
A	DERECHA
W	ARRIBA
S	ABAJO
D	IZQUIERDA

Tabla 3: Teclas de movimiento del jugador

- Interacción con los NPCs:
  Para interactuar con los personajes no jugadores (NPC) dentro del juego, utiliza la tecla E. Al acercarte a un NPC, mirar en su dirección y presionar la tecla E, podrás entablar una conversación o iniciar una interacción con dicho personaje. Una vez que

hayas interactuado con un NPC y desees finalizar la interacción, tienes dos opciones. Puedes presionar la tecla Esc (Escape) para salir de la interacción sin realizar la prueba que el NPC pueda ofrecerte en ese momento. Por otro lado, si estás interesado en realizar la prueba propuesta por el NPC, puedes presionar la tecla Enter. Esto queda representado en la Tabla 4.

TECLA	**ACCIÓN**
E	INICIAR INTERACCION
ESC	CERAR INTERACCION
ENTER	INICIAR PRUEBA

Tabla 4: Teclas de interacción del jugador

Es importante tener en cuenta que, si el NPC no tiene una prueba disponible para ofrecerte en ese momento, puedes presionar tanto la tecla Esc como la tecla Enter para avanzar en el juego sin realizar una prueba específica. Recuerda que estos controles te permitirán explorar el mundo del juego, interactuar con los NPC y progresar en las pruebas de Microprocesadores de manera fluida.

Pruebas

El núcleo del juego se centra en la superación de pruebas. En el encontrarás tres tipos diferentes de pruebas que pondrán a prueba tus conocimientos. A continuación, se describen los tipos de pruebas disponibles:

- Prueba de opción múltiple:
  Este tipo de prueba requerirá que elijas una opción correcta de entre las presentadas. Se te presentarán varias alternativas y deberás seleccionar la que consideres correcta.

Analiza cuidadosamente las opciones antes de tomar tu decisión, ya que solo una de ellas será la respuesta correcta.

- Prueba de completar el espacio en blanco:
  En este tipo de prueba, se te presentará una oración, una pregunta o un enunciado con espacios en blanco. Tu objetivo será completar los espacios vacíos escribiendo la solución correspondiente. Deberás utilizar tus conocimientos en Microprocesadores para determinar la respuesta adecuada y escribirla en el espacio proporcionado.

- Prueba de selección desplegable:
  Esta prueba requiere que respondas utilizando opciones desplegables. Se te presentará una pregunta o un enunciado con una lista de opciones desplegables. Deberás seleccionar la opción correcta de la lista proporcionada para responder correctamente a la pregunta.

Si fallas una prueba, el juego te proporcionará una retroalimentación sobre las preguntas que has respondido incorrectamente. Esta función te permitirá identificar las áreas en las que necesitas mejorar y te brindará la oportunidad de fortalecer tus conocimientos.

Una vez que hayas completado exitosamente una prueba, se desbloqueará la siguiente y tendrás que encontrarla en el mapa del juego. Interactuar con los NPCs te proporcionará pistas sobre la ubicación de la siguiente prueba. Ten en cuenta que, si fallas en completar una prueba, perderás una vida. Por lo tanto, es importante estar preparado y utilizar tus habilidades y conocimientos en Microprocesadores para tener éxito en cada prueba.

Vidas

Al comenzar tu aventura, dispondrás de tres vidas. Estas se pueden ver en la esquina superior izquierda de la pantalla. Sin embargo, ten en cuenta que cada vez que falles en completar una prueba, perderás una vida.

Es fundamental saber que no hay manera de recuperar vidas perdidas durante el juego. Por lo tanto, es importante tomar decisiones estratégicas y realizar tu mejor esfuerzo para responder correctamente a las preguntas y superar las pruebas.

Si en algún momento pierdes las tres vidas disponibles, lamentablemente tendrás que reiniciar el juego desde el principio. Esta mecánica de vidas agrega un nivel adicional de dificultad y te motiva a mantener un enfoque constante y preciso durante todo el juego.

Ajustes

En la esquina superior derecha del juego, encontrarás un botón que te permitirá acceder a un menú con varias opciones útiles. A continuación, se describen las funciones disponibles en este menú:

- Guardar:
  Esta opción te permitirá guardar tu progreso en el juego. Es importante destacar que el progreso solo se guarda automáticamente al completar una prueba. Si deseas guardar tu progreso entre pruebas, deberás hacerlo manualmente seleccionando esta opción en el menú. Al guardar, asegúrate de seleccionar un espacio de guardado adecuado para almacenar tu progreso actual.

- Guardar y cerrar:
  Esta opción te permitirá guardar tu progreso y salir del juego de manera segura. Al seleccionar esta opción, se guardará automáticamente tu progreso actual y el juego se cerrará. La próxima vez que inicies el juego, podrás continuar desde el último punto guardado.

- Ajustar volumen:
  Esta opción te brindará la capacidad de ajustar el volumen del juego según tus preferencias. Podrás aumentar o disminuir el volumen para adaptarlo a tus necesidades o preferencias auditivas.

Recuerda utilizar el botón en la esquina superior derecha con precaución y acceder al menú cuando sea necesario. Asegúrate de guardar tu progreso regularmente para no perder tus logros y avances en el juego.

ANEXO III: MANUAL DEL PROGRAMADOR

Pasos para visualizar y ejecutar el código del proyecto desde GitHub:

1. Accede a la URL del proyecto en GitHub: https://github.com/aleedelarica/Gamificacion-de-Microprocesadores
2. En la página del repositorio, encontrarás el botón "Code" o "Clone" (clonar). Haz clic en él y se abrirá un menú desplegable.
3. En el menú desplegable, selecciona la opción "Download ZIP" (descargar ZIP) para descargar el código fuente del proyecto en formato ZIP.
4. Una vez descargado el archivo ZIP, descomprímelo en una ubicación conveniente de tu computadora.
5. Comprueba que tienes JAVA descargado y con una versión superior o igual a la 17.
6. Abre tu IDE preferido (o descárgalo si no lo tienes), como Visual Studio o IntelliJ.
7. En el IDE, selecciona la opción "Open" o "Open Project" (abrir proyecto) desde el menú principal.
8. Navega hasta la ubicación donde descomprimiste el archivo ZIP y selecciona la carpeta principal del proyecto. El IDE cargará y abrirá el proyecto.
9. Para ejecutar el proyecto has de ejecutar una tarea especifica de Gradle. Navega hasta la pestaña de Gradle (en VS code has de instalar la extensión de Gradle) y ejecuta la tarea desktop/Tasks/other/run.
10. El IDE compilará y ejecutará el proyecto, y podrás ver los resultados en la interfaz gráfica.
11. Si solo se desea ver el juego se puede ejecutar un archivo JAR (Archivo 2Archivo 3) mediante la siguiente línea de código en el CMD (has de tener el CMD abierto en la ubicación del JAR): `java -jar gamificacionDeMicroprocesadores.jar`

gamificacionDeMicroprocesadores.jar

Archivo 3: Archivo jar del juego

Con estos pasos, podrás acceder al código fuente del proyecto, visualizarlo, clonarlo en tu computadora local y ejecutarlo desde tu IDE preferido. Esto te permitirá analizar el código según tus necesidades y explorar el funcionamiento del juego en detalle.

Al abrir el proyecto en un IDE veras una estructura que a primeras parece muy compleja. nosotros solo hemos de centrarnos en 3 áreas, *assets*, *core* y desktop.

Assets contiene todos los extras usados en el código, ya sean las texturas de los personajes, el mapa o la música, y todo se encuentra organizado en sub carpetas.

Core contiene todo el código que se ha diseñado para este proyecto. Core está dividido en los diferentes paquetes del proyecto.

Finalmente la carpeta *desktop* tiene una carpeta llamada *build* que contiene una carpeta llamada *libs*, dentro de ella se encuentra el archivo JAR del juego.