UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

DESARROLLO DE UN VIDEOJUEGO EDUCATIVO COMO HERRAMIENTA PARA LA ENSEÑANZA DE LOS CONCEPTOS BÁSICOS DE LA PROGRAMACIÓN

Elaborado por:

Alejandra Tonito C.I. 26.383.611

Trabajo de Grado Presentado Ante la Universidad de Oriente Como Requisito Parcial para Optar al Título de:

INGENIERO EN COMPUTACIÓN

Barcelona, 2025.

UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

DESARROLLO DE UN VIDEOJUEGO EDUCATIVO COMO HERRAMIENTA PARA LA ENSEÑANZA DE LOS CONCEPTOS BÁSICOS DE LA PROGRAMACIÓN

Elaborado por:

Alejandra Tonito C.I. 26.383.611

TESIS REVISADA POR ASESOR ACADÉMICO

_____

Ing. Claudio Cortínez

Barcelona, 2025.

RESOLUCIÓN

De acuerdo con el Artículo 44 del Reglamento de Trabajos de Grado:

"Los trabajos son propiedad exclusiva de la Universidad de Oriente, y sólo podrán ser utilizados para otros fines con el consentimiento expreso del Consejo de Núcleo respectivo, quien participará al Consejo Universitario para su autorización

DEDICATORIA

A mi madre, por su amor incondicional, por creer en mí incluso en los momentos más difíciles y por enseñarme que el esfuerzo y la perseverancia siempre tienen su recompensa. Sin su apoyo, este logro no habría sido posible.

A mi hermano por su compañía, sus risas y su ánimo constante.

A mis amigos, esos faros de luz que iluminaron los días de estrés e incertidumbre. Gracias por escucharme, por empujarme a seguir adelante y por celebrar cada pequeño avance como si fuera suyo.

A cada persona que, de una forma u otra, contribuyó a que este sueño se hiciera realidad: este logro también es de ustedes.

AGRADECIMIENTOS

Agradezco profundamente a mi tutor de tésis, el profesor Claudio Cortínez, por su guía, paciencia y valiosas enseñanzas durante este proceso. Su conocimiento y dedicación fueron fundamentales para la culminación de este trabajo.

A mis profesores, por compartir su sabiduría y despertar en mí la pasión por el desarrollo de software. Cada clase y consejo contribuyó a mi crecimiento académico.

A mis amigos y compañeros de la universidad, especialmente a Katherine, Cristhian, y Daniel, por los debates, el apoyo mutuo y los momentos compartidos que hicieron de esta etapa una experiencia inolvidable.

A todos aquellos que, de cerca o de lejos, me brindaron su apoyo emocional o intelectual. Este logro no hubiera sido posible sin cada palabra de aliento, cada consejo oportuno y cada muestra de cariño.

Finalmente, a quienes me inspiraron a seguir adelante: gracias por ser mi motivación.

RESUMEN

Este proyecto desarrolla un videojuego educativo para enseñar los conceptos básicos de la programación de manera interactiva y accesible. El objetivo fue crear una herramienta que combine entretenimiento y educación, permitiendo a los usuarios aprender conceptos como estructuras de control, bucles y funciones a través de desafíos prácticos y retroalimentación inmediata. Dirigido a principiantes, el juego presenta los conceptos de manera gradual para facilitar el aprendizaje. El desarrollo se realizó utilizando el motor Godot y el lenguaje GDScript, seleccionados por su flexibilidad y facilidad de uso. Se siguió la metodología RUP (Proceso Unificado de Desarrollo de Software), que permitió una planificación estructurada y un enfoque iterativo. El videojuego incluye niveles que introducen conceptos de programación de manera progresiva, desde comandos básicos hasta estructuras más complejas. La interfaz es accesible y fácil de usar, con un panel de ayuda que proporciona información relevante sobre los conceptos tratados en cada nivel. Además, el juego permite ajustar configuraciones de sonido y guardar el progreso automáticamente.

ÍNDICE

RESOLUCIÓN iii
DEDICATORIA iv
AGRADECIMIENTOS v
RESUMEN vi
ÍNDICE vii
CAPÍTULO I 11
EL PROBLEMA 11
1.1 Planteamiento del problema 11
1.2 Objetivo general 13
1.3 Objetivos específicos 13
CAPÍTULO II 15
MARCO TEÓRICO 15
2.1 Antecedentes 15
2.3 Bases teóricas 18
2.3.1 Fundamentos de la programación 18
2.3.2 Pensamiento computacional 20
2.3.3 Video Juego 20
2.3.4 Juegos Serios 21
2.3.5 Gamificación 21
2.3.6 Proceso Unificado de Desarrollo Software (RUP) 22
2.3.7 Etapas del Proceso Unificado de Desarrollo Software 22

2.3.8 Diseño Instruccional de Galvis 24
2.3.9 Motor de Videojuegos 25
CAPÍTULO III 26
FASE DE INICIO 26
3.1 Modelo de Dominio 26
3.1.1 Glosario de términos 27
3.2 Requisitos 28
3.2.1 Requisitos Funcionales 28
3.2.2 Requisitos no Funcionales 29
3.3 Modelo de Casos de Uso 29
3.3.1 Identificación de los actores 30
3.3.2 Identificación de Casos de Uso 30
3.3.3 Descripción de los casos de uso 31
3.4 Identificación y Mitigación de Riesgos 33
CAPÍTULO IV 35
FASE DE ELABORACIÓN 35
4.1 Análisis 35
4.1.1 Diagramas de Clases de Análisis 37
4.1.2 Diagramas de Colaboración 40
4.1.3 Diagramas de Paquetes de Análisis 43
4.2 Diseño 45
4.2.1 Diagrama de Clase de Diseño 45
4.2.2 Diagrama de Capas 47

4.2.3 Diseño de Interfaz 48
4.3 Implementación 50
4.3.1 Diagrama de Componentes 51
CAPÍTULO V 53
FASE DE CONSTRUCCIÓN 53
5.1 Herramientas de Desarrollo 53
5.1.1 Escogencia del motor gráfico 53
5.1.2 Escogencia del lenguaje de desarrollo 53
5.2 Pruebas 53
5.2.1 Pruebas Unitarias 54
5.2.2 Pruebas de Integración 57
5.3 Código de los componentes 60
5.3.1 Interfaz.gd 60
5.3.2 Global.gd 75
5.3.3 Personaje.gd 78
5.3.4 Nivel.gd 83
5.3.5 MenuPrincipal.gd 85
CAPÍTULO VI 90
FASE DE TRANSICIÓN 90
Implementación 90
Pruebas 90
CONCLUSIONES 92
RECOMENDACIONES 93

BIBLIOGRAFÍA 94

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO: 96

CAPÍTULO I
EL PROBLEMA

1.1 Planteamiento del problema

Los juegos, según Koster (2004), son acertijos que resolver, como todo lo que encontramos en la vida. Están en el mismo nivel que aprender a conducir un automóvil, tocar la mandolina o aprender las tablas de multiplicar. Aprendemos los patrones subyacentes, los asimilamos completamente y los archivamos en nuestros cerebros para que podamos volver a ejecutarlos en el futuro según sea necesario. La única diferencia real entre los juegos y la realidad es que en los juegos los riesgos son menores (p. 34). Esta característica hace que los videojuegos, en particular, se conviertan en un medio viable para aplicar estos principios del aprendizaje de manera segura y controlada.

Los videojuegos educativos han surgido como una herramienta atractiva para la enseñanza de distintos conceptos en diferentes áreas educativas, combinando entretenimiento e interactividad con educación. Estos videojuegos presentan la ventaja de adaptarse a diversos niveles de habilidad y conocimiento, haciéndolos accesibles para jugadores de distintas edades y niveles educativos. Desde los fundamentos básicos hasta conceptos más avanzados, ofrecen una progresión gradual que permite a los jugadores aprender a su propio ritmo, proporcionando retroalimentación inmediata y orientación, facilitando la corrección de errores y el fortalecimiento de la comprensión de los conceptos de programación.

A pesar de los beneficios de los recursos educativos existentes, muchos no logran captar el interés de los principiantes de manera efectiva debido a que suelen ser poco intuitivos, lo que puede desmotivar a los usuarios principiantes. Otra limitación significativa es que muchos requieren una conexión a internet, reduciendo su accesibilidad para usuarios con acceso limitado o nulo a la red.

Por esta razón, surgió la necesidad de desarrollar un videojuego educativo que permita enseñar los conceptos fundamentales de la programación de una manera atractiva para principiantes. Un videojuego educativo bien diseñado puede abordar los desafíos específicos que enfrentan los individuos al comenzar su estudio de la programación, presentando ejercicios prácticos y ejemplos de código que les permitan aplicar los conceptos aprendidos de manera inmediata y relevante, el cual al poder utilizarse sin conexión a internet se hará más accesible a personas que quieran dar uso a esta herramienta educativa.

El contenido del videojuego se adaptó a niveles básicos y de principiantes, asegurando su accesibilidad y utilidad para jugadores con diversos antecedentes. Se diseñaron niveles y desafíos que abordaron desde los conceptos más básicos hasta temas intermedios, como estructuras de control y funciones, de una manera clara y comprensible. Lo cual permitirá guiar a los jugadores a medida que desarrollan su comprensión de la programación y adquieren habilidades prácticas que podrán aplicar en proyectos reales.

Al final, este enfoque integral permitió no solo la creación de un recurso educativo valioso, sino también la posibilidad de innovar en la forma en que se enseñan los conceptos de programación, haciendo que el aprendizaje sea una experiencia interactiva, envolvente y motivadora para los estudiantes.

1.2 Objetivo general

Desarrollar un videojuego educativo como herramienta para la enseñanza de los conceptos básicos de la programación.

1.3 Objetivos específicos

- Identificar los conceptos fundamentales de la programación que deben ser incluidos en el videojuego educativo.

- Recolectar información sobre las mejores prácticas en el diseño de videojuegos mediante el análisis de la literatura disponible.

- Diseñar la arquitectura del software, definiendo los módulos a ser usados para el videojuego educativo.

- Realizar el diseño de una interfaz gráfica de usuario intuitiva y sencilla de navegar.

- Implementar todos los módulos previamente diseñados utilizando el motor de software libre Godot.

- Verificar el correcto funcionamiento del videojuego educativo desarrollado.

CAPÍTULO II
MARCO TEÓRICO

2.1 Antecedentes

Zhao y Shute (2019) realizaron un estudio para comprender cómo los juegos diseñados específicamente para desarrollar habilidades de pensamiento computacional influyen en las habilidades y actitudes de los estudiantes de secundaria hacia la informática. Encontraron que estos juegos mejoran significativamente dichas habilidades en un corto periodo de tiempo. Sin embargo, identificaron limitaciones del juego que podrían haber afectado negativamente el aprendizaje y las actitudes de los estudiantes, como la perspectiva inversa del personaje que generaba una carga cognitiva adicional, la ausencia de un sistema de recompensas para aumentar la motivación y problemas técnicos durante el estudio. Estos hallazgos informan el diseño del juego en este proyecto, permitiendo implementar una estructura motivacional y fácil de entender.

Serrano (2019) realizó un análisis en el que identifica la necesidad de comprender el impacto del aprendizaje basado en juegos digitales en estudiantes de preescolar. Este análisis incluyó dieciséis estudios de investigación, dos metanálisis y dos revisiones de literatura, todos publicados entre 2011 y 2019. Los resultados concluyen que el aprendizaje basado en juegos digitales tiene un impacto positivo en los estudiantes cuando se utiliza junto con métodos tradicionales de enseñanza. Además, se observó que la presencia de elementos de diseño de juegos y estructuras instruccionales aumenta el interés y la participación de los estudiantes. Este análisis resalta la importancia de fundamentar el diseño de los juegos en teorías educativas

preexistentes y verificadas para maximizar la efectividad de los videojuegos educativos, así como la implementación de principios sólidos de diseño de juegos para ofrecer una experiencia más atractiva a los estudiantes.

En la investigación realizada por Gentry et al. (2019), se estudió la eficacia de la aplicación de juegos serios y la gamificación como técnicas educativas para abordar la escasez mundial de profesionales en el área de la salud. A partir de los 30 estudios considerados en esta investigación, se encontró evidencia de que estas técnicas pueden mejorar el conocimiento tanto de estudiantes como de profesionales. En algunos estudios, se encontró que estas técnicas pueden complementar la educación tradicional para el desarrollo de habilidades profesionales. Esta investigación proporciona evidencia sobre la efectividad de los juegos educativos como suplemento a la educación tradicional y también identifica las limitaciones de estudios previos. Además, sugiere la necesidad de implementar teorías educativas en estos estudios y considerar el contexto económico de los participantes para un uso efectivo de estas herramientas educativas.

En su estudio, Bado (2019) resalta la necesidad de comprender las prácticas pedagógicas involucradas en la integración de juegos digitales en la enseñanza y el aprendizaje, debido a la escasa documentación existente sobre las actividades instruccionales específicas que se implementan en las diferentes etapas del estudio, denominadas pre-juego, durante el juego y post-juego, y cómo estas actividades influyen en los resultados del aprendizaje. Bado llevó a cabo una revisión de artículos publicados en los 10 años anteriores a su estudio, identificando las actividades implementadas en cada etapa, las cuales se centraron principalmente en aspectos técnicos del juego,

su funcionamiento y la satisfacción de los estudiantes al finalizar el juego. Estos resultados proporcionan una guía para ajustar el contenido instruccional y la interfaz del juego con la finalidad de que sea accesible y fácil de entender.

Greipl et al. (2020) identifican en su estudio la necesidad de comprender tanto el potencial como las limitaciones del aprendizaje basado en juegos en el contexto educativo, y resalta la necesidad de encontrar maneras efectivas de integrar estas herramientas a los métodos de educación tradicional para mejorar el proceso de aprendizaje. Este estudio propone la utilización de un marco tridimensional que aborda los factores cognitivos, emocionales y sociales del aprendizaje basado en juegos, junto con las características del entorno de aprendizaje. Enfatiza que los juegos serios deben complementar y mejorar los enfoques educativos tradicionales, más no reemplazarlos. Además, subraya la importancia de recursos adecuados, marcos teóricos sólidos, y objetivos claros para implementar esta herramienta de manera efectiva.

Whang y Zheng (2020) buscaron comprender los efectos del aprendizaje basado en juegos, tanto digitales como no digitales, en el rendimiento académico y la autoeficacia de los estudiantes en la asignatura de ciencias. Aunque estudios previos han demostrado las ventajas del aprendizaje basado en juegos en comparación con el método tradicional, pocos han comparado las diferencias entre los juegos digitales y no digitales. En su estudio, llevaron a cabo un experimento para comparar el impacto de ambos tipos de juegos en el rendimiento académico de estudiantes de secundaria en China. Los resultados mostraron que ambos tipos de juegos tienen efectos positivos y significativos en el aprendizaje del contenido en

comparación con los métodos tradicionales. Sin embargo, no encontraron una diferencia significativa entre los grupos de juegos digitales y no digitales. Este estudio proporciona evidencia de que los juegos educativos, sean digitales o no digitales, pueden mejorar el rendimiento académico de los estudiantes, lo cual respalda la validez de este enfoque pedagógico.

2.3 Bases teóricas

2.3.1 Fundamentos de la programación

Algoritmo: Un algoritmo es una secuencia ordenada de pasos que se ejecutan en un tiempo finito para resolver un problema. Los problemas, en el contexto de un algoritmo, son cuestiones conceptuales o prácticas que pueden resolverse mediante la aplicación de un conjunto de instrucciones precisas y no ambiguas

La importancia de definir un problema como algo resoluble mediante un algoritmo radica en la capacidad de proporcionar una solución sistemática y eficiente. Al expresar un problema en términos algorítmicos, se facilita su comprensión y se ofrece una guía clara para su resolución.

Estructuras de datos: Una estructura de datos es una forma de organizar y almacenar datos en una computadora para permitir su acceso y manipulación de manera eficiente. Las estructuras de datos son fundamentales en la programación y son utilizadas por los programadores para organizar y manipular datos de manera efectiva. El uso adecuado de las estructuras de datos puede mejorar significativamente la eficiencia y el rendimiento del programa.

Estructuras de control: Son aquellas que tienen por objeto marcar el orden de realización de los distintos pasos de un programa o algoritmo. Cada estructura tiene un punto de entrada y uno de salida, lo que facilita la depuración de posibles errores. Estas son de tres tipos:

1. **Estructuras selectivas**: Las estructuras selectivas en un programa se utilizan para tomar decisiones, de ahí que se suelen denominar también estructuras de decisión o alternativas. En estas estructuras se evalúa una condición, especificada mediante expresiones lógicas, en función de cuyo resultado, se realiza una opción u otra. En una primera aproximación, para esta toma de decisiones, podemos pensar en una variable interruptor o conmutador (switch), que representa un estado y por tanto puede cambiar de valor a lo largo de la ejecución regulando el paso a una u otra parte del programa, lo que supone una bifurcación en el flujo del programa, dependiendo del valor que tome el conmutador.

2. **Estructuras repetitivas**: El computador está especialmente diseñado para aplicaciones en las que una operación o un conjunto de ellas deben repetirse muchas veces. En este sentido, definiremos bucle o lazo (loop), como un segmento de un programa cuyas instrucciones se repiten bien un número determinado de veces o mientras se cumpla una determinada condición.

Funciones: Matemáticamente una función es una operación que a partir de uno o más valores, llamados argumentos, produce un valor denominado resultado o valor de la función, por medio de ciertas operaciones sobre los argumentos.

En el contexto de la informática, una función es un bloque de código con un nombre único que realiza una tarea específica. Proporciona modularidad, reutilización y abstracción en el desarrollo de programas, lo que facilita la organización y la eficiencia en la escritura de código.

2.3.2 Pensamiento computacional

El pensamiento computacional es un proceso mental que implica la resolución de problemas de manera lógica y sistemática, utilizando principios y técnicas de la informática. Este enfoque se basa en descomponer problemas complejos en partes más manejables, reconocer patrones, abstraer conceptos relevantes y diseñar algoritmos para resolver problemas de forma eficiente. El pensamiento computacional no se limita a la programación o la informática; es una habilidad interdisciplinaria que puede aplicarse en diversas áreas para mejorar la comprensión y la resolución de problemas.

2.3.3 Video Juego

Un videojuego es una forma de entretenimiento interactivo que requiere la interacción del jugador con una interfaz de usuario para generar retroalimentación visual en una pantalla. Esta interacción se logra a través de diversos dispositivos, como consolas de videojuegos, computadoras, teléfonos móviles o tablets. Los videojuegos están diseñados con sistemas de recompensa que motivan al jugador a cumplir con las tareas y desafíos establecidos en el juego. Estas recompensas pueden incluir puntos, niveles, trofeos, desbloqueo de contenido adicional, o mejoras en las habilidades del

personaje. Además, los videojuegos pueden ofrecer experiencias variadas, que van desde la simulación y la estrategia hasta la acción y la aventura, lo que permite a los jugadores explorar mundos virtuales y participar en narrativas interactivas.

2.3.4 Juegos Serios

Un juego serio es un tipo de juego diseñado con un propósito principal que va más allá del entretenimiento. Estos juegos incorporan elementos lúdicos y mecánicas de juego para cumplir objetivos educativos, formativos, terapéuticos o informativos. En el ámbito educativo, los juegos serios se utilizan para enseñar conocimientos específicos o desarrollar habilidades en diversas áreas como matemáticas, ciencias, historia, idiomas y habilidades técnicas.

2.3.5 Gamificación

La gamificación se define como la aplicación de elementos y principios propios del diseño de juegos en contextos no lúdicos con el propósito de aumentar la motivación y el compromiso de los individuos. Este enfoque utiliza mecanismos como la acumulación de puntos, niveles, recompensas, desafíos y retroalimentación inmediata para incentivar comportamientos específicos y mejorar la experiencia del usuario en diversas actividades. En el ámbito educativo, la gamificación se emplea para fomentar el interés y la participación activa de los estudiantes, facilitando el aprendizaje a través de un entorno más interactivo y atractivo.

2.3.6 Proceso Unificado de Desarrollo Software (RUP)

El proceso unificado de desarrollo software, o RUP por sus siglas en inglés, es un marco de trabajo para el desarrollo de software que proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es garantizar la producción de software de alta calidad que cumpla con las necesidades de los usuarios finales dentro de un presupuesto y tiempo estimado.

El RUP es iterativo e incremental, y se organiza en fases y disciplinas. Las fases representan el ciclo de vida del proyecto, mientras que las disciplinas representan las diferentes actividades que deben realizarse en cada iteración.

2.3.7 Etapas del Proceso Unificado de Desarrollo Software

El proceso unificado de desarrollo software divide su ciclo de vida en cuatro etapas: Inicio, Elaboración, Construcción y Transición. Cada fase está compuesta por una serie de iteraciones que permiten desarrollar el sistema de manera incremental.

La fase de inicio se centra en definir el alcance del proyecto y los requisitos básicos del sistema. En esta etapa, es fundamental establecer una visión general del proyecto. Las actividades principales incluyen la identificación de requisitos y restricciones, la evaluación de riesgos iniciales, la estimación de costos y tiempos, y la elaboración de un caso de negocio que justifique la viabilidad y el valor del proyecto. Esta fase también incluye la creación de un plan preliminar del proyecto que guiará las siguientes etapas.

En la fase de elaboración, el objetivo principal es desarrollar una arquitectura robusta y estable que sirva como base para el resto del proyecto. Durante esta etapa, se refinan y priorizan los requisitos del sistema y se mitigan los riesgos técnicos más importantes. Las actividades clave incluyen el refinamiento de los requisitos del sistema, la definición de la arquitectura del sistema y la implementación de prototipos o versiones preliminares de componentes críticos para validar dicha arquitectura. Esta fase también implica la actualización del plan del proyecto con estimaciones más precisas basadas en el conocimiento adquirido hasta el momento.

La fase de construcción se dedica al desarrollo completo del sistema de acuerdo con los requisitos y la arquitectura definidos en las fases anteriores. Las actividades clave incluyen la implementación de todas las funcionalidades y componentes del sistema, la realización de pruebas unitarias e integración para asegurar que los componentes funcionan correctamente juntos, y la documentación del sistema, incluyendo la creación de manuales de usuario. Las iteraciones permiten refinar y mejorar el sistema basado en la retroalimentación continua, asegurando que se cumplan los requisitos del cliente y se aborden los problemas emergentes de manera efectiva.

La fase de transición se centra en la entrega del sistema al usuario final y en asegurar su correcta implementación en el entorno operativo. Durante esta etapa, se realizan pruebas piloto y de aceptación del usuario para validar que el sistema cumple con los requisitos, se capacita a los usuarios finales y al personal de soporte técnico, y se implementa el sistema en el entorno operativo. Esta fase también incluye la resolución de problemas y la corrección

de errores detectados durante la implementación, así como la revisión y cierre del proyecto.

2.3.8 Diseño Instruccional de Galvis

El diseño instruccional de Galvis es una metodología educativa que se centra en la creación de sistemas de enseñanza eficaces y eficientes, considerando varios aspectos clave en el proceso de diseño y desarrollo. Esta metodología abarca tres componentes principales: educativo, comunicacional y computacional.

En el componente educativo, se determina el contenido educativo y el alcance del sistema, especificando los temas que se abordarán y los objetivos de aprendizaje. Los temas se seleccionan y organizan para facilitar la comprensión y la adquisición de conocimientos. Se establecen las metodologías y técnicas de enseñanza que se utilizarán para alcanzar los objetivos educativos. Esto puede incluir métodos interactivos, actividades prácticas, ejercicios y evaluaciones que refuercen el aprendizaje.

El componente comunicacional se enfoca en diseñar la interacción entre el usuario y la máquina, creando una interfaz y elementos interactivos que permitan a los usuarios interactuar eficazmente con el sistema educativo. Esto incluye la creación de menús, botones, gráficos y otros elementos visuales que faciliten la navegación y el uso del sistema. Asegura que la interfaz sea accesible y fácil de usar para todos los usuarios, independientemente de sus habilidades técnicas, mediante pruebas de usabilidad.

El componente computacional define las funciones y características técnicas que debe tener el sistema educativo para apoyar a los usuarios, tanto docentes como estudiantes. Esto incluye la especificación de algoritmos, bases de datos, sistemas de gestión de contenido y otras herramientas técnicas necesarias para el funcionamiento del sistema. Se implementan las soluciones técnicas necesarias para cumplir con los requisitos educativos y comunicacionales previamente definidos, lo que puede incluir la programación de software, la integración de multimedia y la configuración de plataformas tecnológicas.

2.3.9 Motor de Videojuegos

Un motor de videojuegos es una plataforma de software que proporciona el conjunto de herramientas y servicios necesarios para el desarrollo de videojuegos. Incluye componentes esenciales como motores gráficos, que renderizan las imágenes; motores de físicas, que simulan la física del mundo del juego; y motores de audio, que gestionan el sonido. Los motores de videojuegos suelen incorporar herramientas de scripting, editores de escenas, sistemas de animación y otras funcionalidades que permiten a los desarrolladores crear, modificar y optimizar los elementos del juego de manera eficiente. Su objetivo es facilitar el proceso de desarrollo al proporcionar una infraestructura sólida y reutilizable, lo que permite a los desarrolladores centrarse en el contenido y la jugabilidad del juego en lugar de tener que construir estas funcionalidades desde cero.

CAPÍTULO III
FASE DE INICIO

En esta fase se identifican las entidades fundamentales del sistema, se definen los requisitos tanto funcionales como no funcionales, se identifican los actores principales y sus interacciones mediante el modelo de casos de uso, y se realiza el análisis de los riesgos potenciales asociados al proyecto junto a la formulación de estrategias para su mitigación, con el fin de obtener una base clara a partir de la cual desarrollar el software.

3.1 Modelo de Dominio

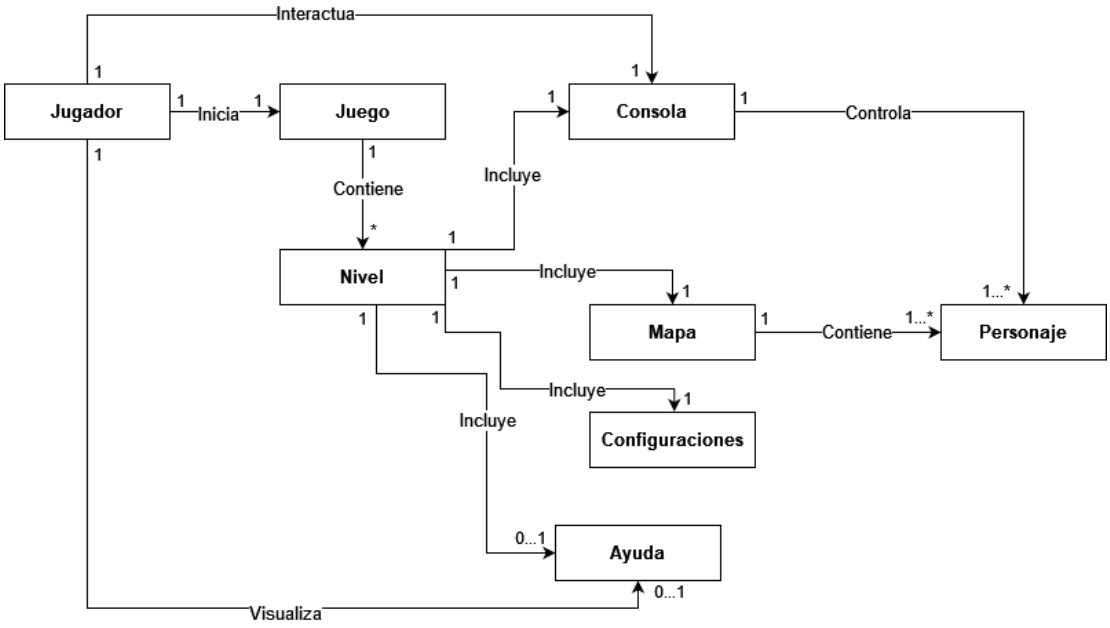


Figura 3.1: Diagrama del Modelo de Dominio

Fuente: Elaboración Propia

En el diagrama del modelo de dominio el jugador es la figura central que interactúa con el sistema. Este tiene la capacidad de ingresar a los niveles que tenga disponibles mediante el menú principal. La interfaz de los niveles está conformada por un mapa, una consola, un panel de configuración, y un panel de ayuda. Dentro del mapa se encuentran uno o más personajes, que serán controlados por las instrucciones que ingrese el jugador a la consola. El panel de ayuda ofrece información relevante en los niveles donde se introduzca un concepto de programación nuevo, mientras que el panel de configuraciones le permita al jugador manipular el volumen del sonido del juego.

3.1.1 Glosario de términos

- **Jugador**: Representa al usuario que interactúa con el juego.
- **Juego**: Representa el sistema en su totalidad.
- **Nivel**: Cada desafío dentro del juego.
- **Consola**: La interfaz en la que el jugador ingresa instrucciones para controlar el personaje. A través de la consola, el jugador puede ejecutar, editar, o reiniciar el código.
- **Mapa**: El entorno visual del nivel donde los personajes se mueven. Cada mapa incluye obstáculos y metas.
- **Personaje**: La entidad dentro del mapa que el jugador controla mediante las instrucciones introducidas en la consola.

- **Configuraciones**: Opciones que permiten al jugador ajustar el volumen del juego.

- **Ayuda**: Un panel accesible dentro de un nivel que proporciona información relevante al concepto de programación tratado en dicho nivel.

3.2 Requisitos

Los requisitos establecen las características y funcionalidades que se esperan del sistema con el objetivo de satisfacer los requerimientos de los usuarios finales. Se detallan los requisitos funcionales, que describen las acciones específicas que el sistema debe realizar, y los requisitos no funcionales, que definen características relacionadas con el rendimiento, la usabilidad y otros aspectos de calidad del sistema.

3.2.1 Requisitos Funcionales

- El jugador debe poder visualizar todos los niveles que ofrece el videojuego en el menú principal, junto a los niveles que ha desbloqueado y el puntaje que ha obtenido en ellos.

- El jugador debe poder seleccionar un nivel desbloqueado, lo que lo llevará a la interfaz del nivel seleccionado.

- El jugador debe poder ingresar y visualizar instrucciones en la consola mediante la lista de instrucciones disponibles.

- El jugador debe poder ejecutar su solución para controlar al personaje en el mapa.

- Si el jugador alcanza la meta, el nivel debe considerarse completado, y el siguiente será desbloqueado. Al final de este nivel se debe mostrar un panel con la puntuación obtenida por el jugador, al igual que un menú que le permita continuar al siguiente nivel, reintentar el nivel actual, o volver al menú principal.

- Cada vez que el jugador complete un nivel o realice un cambio en el panel de configuración, la información relevante debe de guardarse automáticamente en el archivo de guardado del videojuego.

3.2.2 Requisitos no Funcionales

- El juego debe cargar el menú principal y los niveles de forma rápida.

- La interfaz del juego debe ser fácil de usar para jugadores de cualquier nivel de experiencia en videojuegos.

- El sistema de ayuda debe ofrecer explicaciones claras y relevantes sobre los conceptos de programación abordados.

3.3 Modelo de Casos de Uso

El modelo de casos de uso detalla las interacciones principales entre el usuario y los procesos internos que un sistema es capaz de realizar, estas se presentan mediante la identificación de actores y los casos de uso, y la descripción de los casos de uso.

3.3.1 Identificación de los actores

En este sistema se identifica a un único actor principal:

ACTOR	DESCRIPCIÓN
Jugador	El jugador es la entidad externa que interactúa con el sistema. Es el usuario que ejecuta acciones como seleccionar niveles, ingresar instrucciones, y configurar el sonido.

3.3.2 Identificación de Casos de Uso

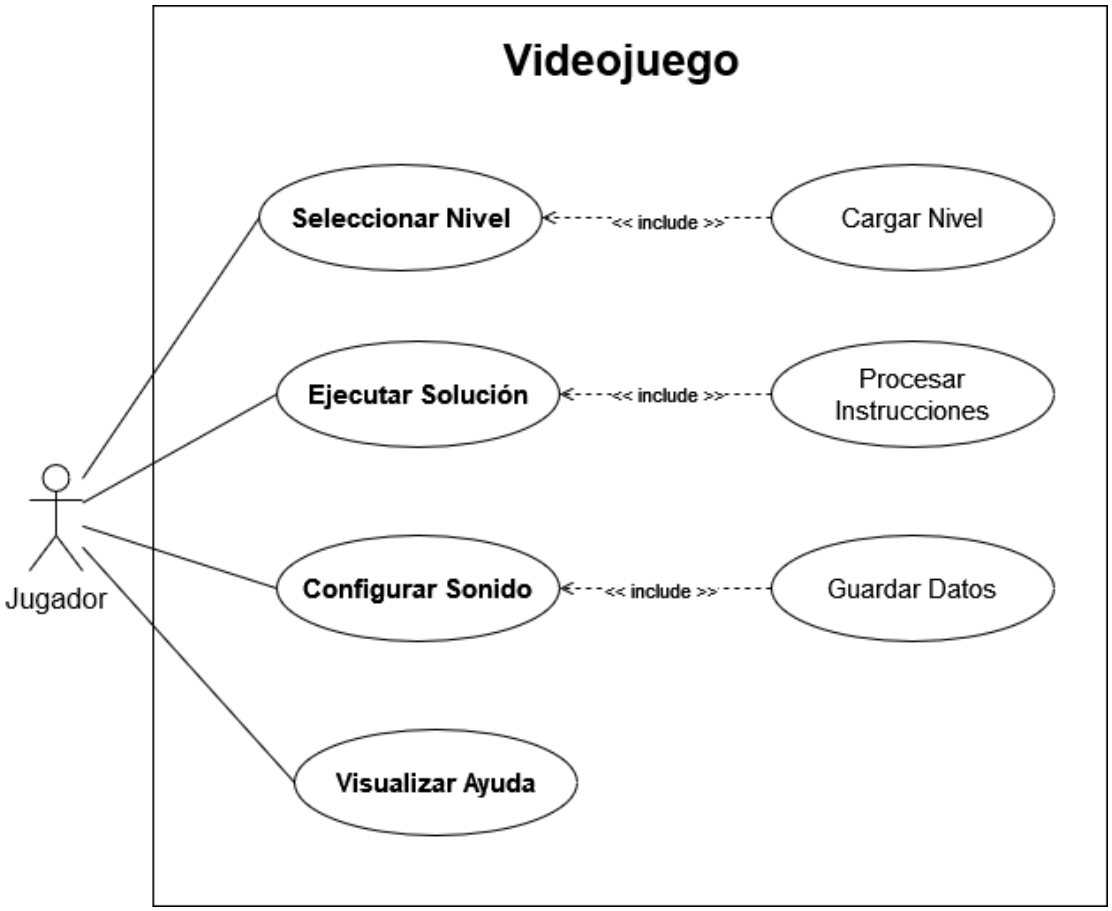


Figura 3.2: Diagrama de Casos de Uso

Fuente: Elaboración Propia

3.3.3 Descripción de los casos de uso

Caso de Uso: *Seleccionar Nivel*

Actor: Jugador

Descripción: El jugador selecciona un nivel desbloqueado desde el menú principal. El sistema carga el nivel seleccionado, incluyendo el mapa, el personaje y la interfaz del nivel.

Precondición: El jugador debe haber desbloqueado al menos un nivel y encontrarse en el menú principal.

Flujo Principal:

1. El jugador selecciona un nivel desbloqueado.
2. El sistema carga el nivel seleccionado.

Caso de Uso: *Ejecutar Solución*

Actor: Jugador

Descripción: El jugador presiona el botón de ejecución en la consola. El sistema procesa las instrucciones ingresadas y mueve al personaje en el mapa de acuerdo con la secuencia de instrucciones proporcionadas.

Precondición: El jugador debe haber ingresado al menos una instrucción válida a la consola.

Flujo Principal:

1. El jugador presiona el botón de ejecutar.
2. El sistema procesa las instrucciones ingresadas.
3. El personaje en el mapa realiza las acciones correspondientes.

Caso de Uso: *Configurar Sonido*

Actor: Jugador

Descripción: El jugador accede al panel de configuración de sonido y ajusta los niveles de volumen global, música y efectos de sonido.

Precondición: El jugador debe estar en el menú principal o dentro de un nivel para acceder a las configuraciones.

Flujo Principal:

1. El jugador selecciona la opción de configuración de sonido.
2. El jugador ajusta los niveles de volumen global, música y efectos de sonido.
3. El sistema guarda los ajustes en el archivo de configuración.

Caso de Uso: *Visualizar Ayuda*

Actor: Jugador

Descripción: El jugador accede al panel de ayuda desde el menú de opciones.

Precondición: El jugador debe estar dentro de un nivel que ofrezca la opción de visualizar el panel de ayuda.

Flujo Principal:

1. El jugador selecciona la opción de ayuda.
2. El sistema muestra el panel con la información necesaria.

3.4 Identificación y Mitigación de Riesgos

En cualquier proyecto de desarrollo de software, es esencial identificar posibles riesgos que puedan afectar el éxito del producto final. En esta sección

se analizan los riesgos asociados al sistema y se proponen estrategias de mitigación para reducir su impacto.

RIESGO	DESCRIPCIÓN	MITIGACIÓN
Error de lógica	Los jugadores podrían ingresar instrucciones que no funcionan correctamente o que rompen la lógica del juego.	Implementar una validación robusta de las instrucciones antes de su ejecución e incluir ejemplos en el panel de ayuda para familiarizar al jugador con la sintaxis de las instrucciones.
Mecánica de juego confusa	Los jugadores pueden tener dificultades para comprender cómo jugar.	Incluir tutoriales que expliquen de manera clara y progresiva las mecánicas del juego y los conceptos de programación.
Contenido inadecuado	El contenido del videojuego podría ser mal interpretado por los usuarios.	Revisar el contenido para asegurar que los conceptos y desafíos estén correctamente presentados

CAPÍTULO IV
FASE DE ELABORACIÓN

La fase de elaboración tiene como objetivo principal construir la base sobre la cual se desarrollará el sistema final. Durante esta fase, se trabajan los flujos de trabajo de análisis y diseño, enfocados en definir y refinar los elementos fundamentales de la arquitectura del sistema. En este proceso, se identifican y clarifican los procesos internos del sistema, se establecen las capas de software, y se organizan los paquetes y subsistemas, asignando responsabilidades claras a cada componente. Además, se depuran y refinan las interfaces tanto internas, que gestionan la interacción entre los componentes del sistema, como las externas, que manejan la comunicación con los actores principales, como el jugador.

A lo largo de la fase de elaboración, los modelos creados en la fase de inicio se revisan y ajustan, generando una versión más detallada y precisa de la solución. Este refinamiento permite que la arquitectura evolucione, tomando en cuenta los requisitos más significativos y asegurando que el diseño esté alineado con los objetivos del proyecto. El resultado de esta fase es una línea base del diseño sobre la cual se construirá y se implementará el videojuego en la fase de construcción.

4.1 Análisis

Esta etapa se basa en estudiar con más profundidad los casos de uso identificados previamente, con el fin de modelar las interacciones entre el jugador y el sistema, visualizando los comportamientos del sistema en respuesta a las acciones del usuario. Este proceso implica la identificación y estructuración de las clases de análisis, que representan los componentes

esenciales que el software debe gestionar. Se identifican principalmente un conjunto de clases de análisis clasificadas en tres categorías: clases de interfaz, clases de control y clases de entidad.

Clases de Interfaz

Las clases de interfaz son responsables de gestionar la interacción entre el jugador y el sistema, actuando como los medios de comunicación visuales y de entrada de datos.

- **UI Menú Principal**: Muestra las opciones principales del videojuego al jugador, como seleccionar nivel o configurar el sonido.
- **UI Nivel**: Muestra el entorno gráfico del nivel al jugador, como el panel de configuración, el panel de ayuda, el mapa, los objetos y obstáculos, el personaje a controlar, la consola, y la lista de comandos.

Clases de Control

Las clases de control se encargan de la lógica del sistema. Actúan como intermediarias entre las clases de interfaz y las de entidad, coordinando las interacciones entre el jugador y los elementos del sistema.

- **Controlador del Juego**: Gestiona el flujo general del juego, como el inicio, gestión de niveles, gestión del progreso del jugador, y configuración del sonido.
- **Controlador del Nivel**: Maneja la lógica relacionada con los niveles.

- **Controlador de la Consola**: Se encarga de procesar los comandos ingresados en la consola y convertirlos en acciones que el personaje puede realizar.
- **Controlador del Personaje**: Ejecuta las acciones del personaje en el mapa, basándose en las instrucciones procesadas que recibe de la consola.

Clases de Entidad

Las clases de entidad representan los objetos y datos del sistema. Estas clases encapsulan la información necesaria para la ejecución del juego.

- **Datos del Juego**: Representa los atributos del juego como los niveles desbloqueados por el jugador, su puntaje, y los cambios que se han realizado en el panel de configuraciones.
- **Datos del Nivel**: Contiene los datos y estados asociados a cada componente que conforma un nivel.

4.1.1 Diagramas de Clases de Análisis

Para cada uno de los casos de uso mencionados, se generan diagramas de clases específicos que detallan cómo las clases de interfaz, control y entidad colaboran entre sí. Estos diagramas se centran en los objetos clave dentro del flujo de trabajo de cada caso de uso, sus relaciones y las responsabilidades que asumen dentro del sistema.

4.1.1.1 Diagrama de Clases de Análisis para el caso de uso “Seleccionar Nivel”

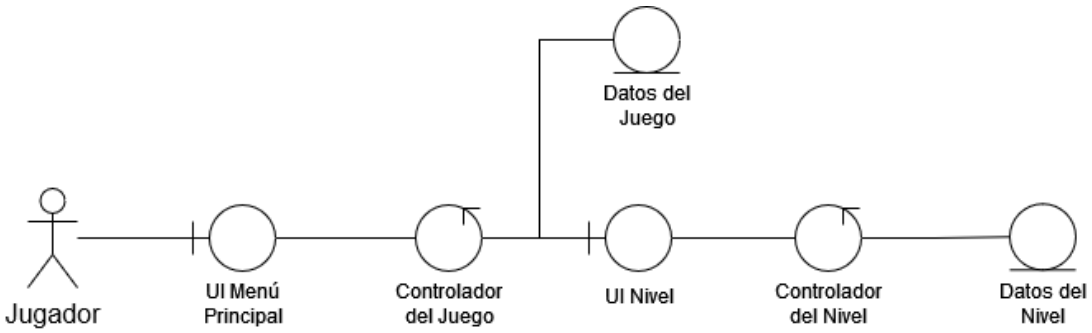


Figura 4.1: Diagrama de Clases de Análisis para el caso de uso “Seleccionar Nivel”

Fuente: Elaboración propia

4.1.1.2 Diagrama de Clases de Análisis para el caso de uso “Ejecutar Solución”

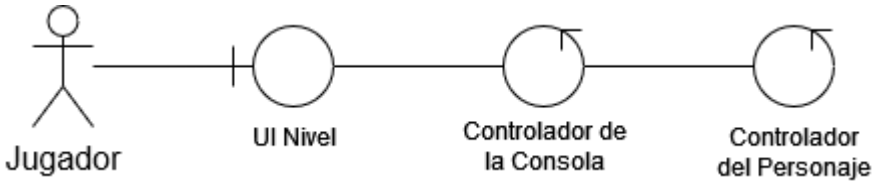


Figura 4.2: Diagrama de Clases de Análisis para el caso de uso “Ejecutar Instrucciones”

Fuente: Elaboración propia

4.1.1.3 Diagrama de Clases de Análisis para el caso de uso “Configurar Sonido”

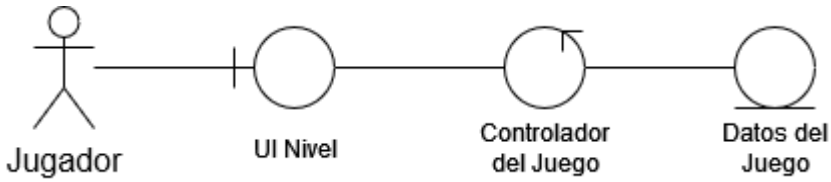


Figura 4.3: Diagrama de Clases de Análisis para el caso de uso “Configurar Sonido**”**

Fuente: Elaboración propia

4.1.1.4 Diagrama de Clases de Análisis para el caso de uso “Visualizar Ayuda**”**

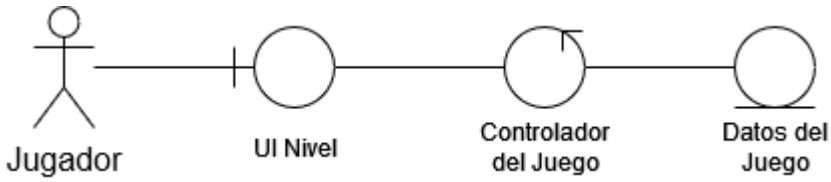


Figura 4.4: Diagrama de Clases de Análisis para el caso de uso “Visualizar Ayuda”

Fuente: Elaboración propia

4.1.2 Diagramas de Colaboración

Los diagramas de colaboración muestran cómo los objetos del sistema interactúan para realizar un caso de uso. Se centran en las relaciones y comunicaciones entre los objetos mientras ejecutan una función específica del sistema. Estos diagramas permiten visualizar el flujo de mensajes entre los actores y los objetos del videojuego, mostrando claramente cómo se intercambian datos y colaboran para cumplir con los requisitos del sistema.

4.1.2.1 Diagrama de Colaboración para el caso de uso “Seleccionar Nivel”

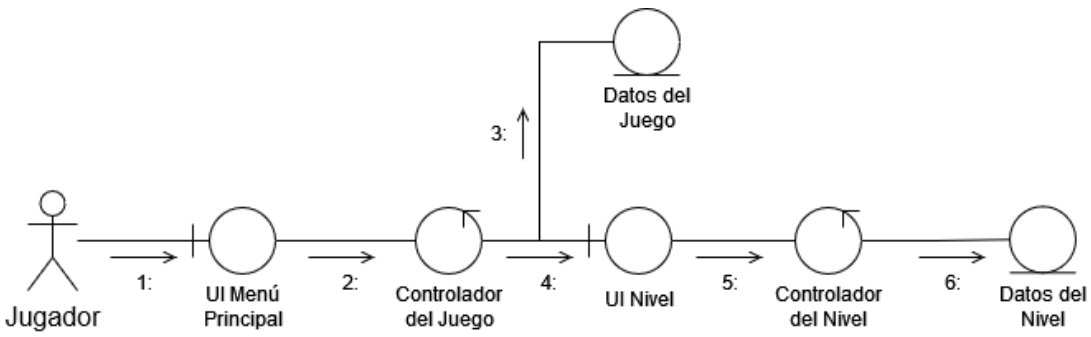


Figura 4.5: Diagrama de Colaboración para el caso de uso “Seleccionar Nivel”

Fuente: Elaboración propia

Leyenda:

1. Solicitud de inicio de juego
2. Solicitud para cargar los datos del juego
3. Solicitud de datos a la entidad “Datos del Juego”
4. Carga la interfaz del nivel seleccionado

5. Solicitud para cargar los datos del nivel
6. Solicitud de datos a la entidad “Datos del Nivel”

4.1.2.2 Diagrama de Colaboración para el caso de uso “Ejecutar Solución”

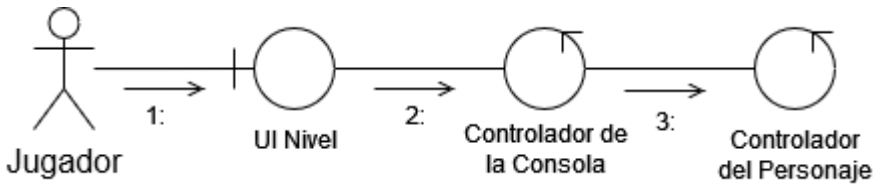


Figura 4.6: Diagrama de Colaboración para el caso de uso “Ejecutar Solución”

Fuente: Elaboración propia

Leyenda:

1. Solicitud para ejecutar solución ingresada en la consola
2. Envío de datos a procesar al controlador de la consola
3. Envío de datos procesados al controlador del personaje

4.1.2.3 Diagrama de Colaboración para el caso de uso "Configurar Sonido"

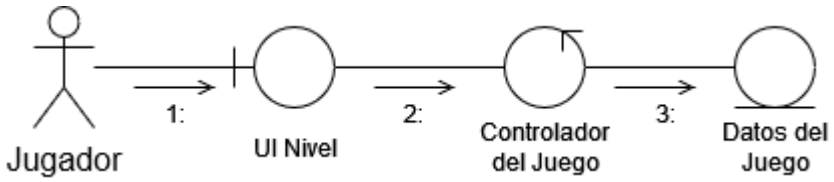


Figura 4.7: Diagrama de Colaboración para el caso de uso "Configurar Sonido"

Fuente: Elaboración propia

Leyenda:

1. Solicitud para visualizar el panel de configuración
2. Envío de cambios del sonido
3. Envío de datos actualizados

4.1.2.4 Diagrama de Colaboración para el caso de uso "Visualizar Ayuda"

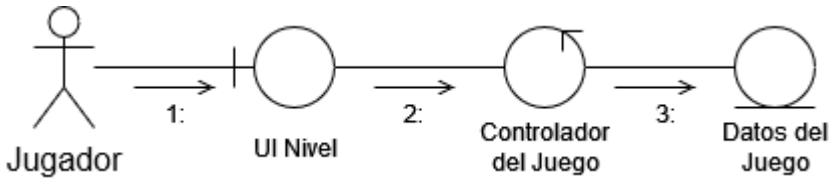


Figura 4.8: Diagrama de Colaboración para el caso de uso "Visualizar Ayuda"

Fuente: Elaboración propia

Leyenda:

1. Solicitud de visualización del panel de ayuda
2. Solicitud para visualizar el texto de ayuda
3. Solicitud de datos del texto de ayuda

4.1.3 Diagramas de Paquetes de Análisis

Los diagramas de paquetes sirven para organizar y simplificar el diseño de un sistema, dividiéndolo en agrupaciones o paquetes lógicos. Estos paquetes agrupan elementos como clases de análisis y casos de uso, lo que facilita la gestión y comprensión del sistema. La idea es mantener una alta cohesión dentro de cada paquete y reducir la dependencia entre ellos, para que el sistema sea más modular y fácil de manejar.

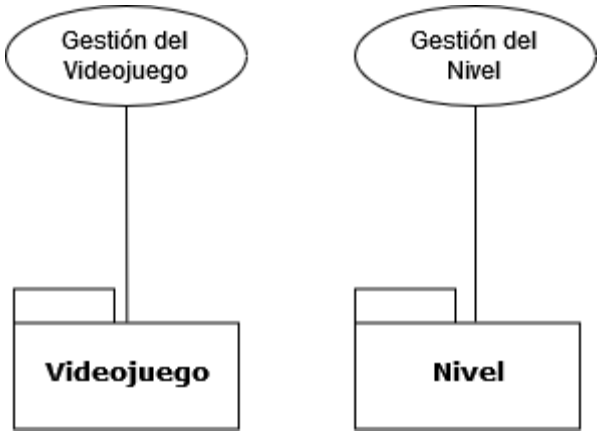


Figura 4.9: Diagrama de paquetes del sistema

Fuente: Elaboración propia.

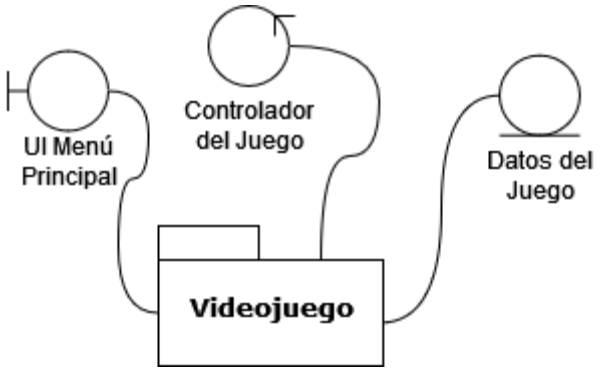


Figura 4.10: Diagrama de paquete de módulo Videojuego

Fuente: Elaboración propia.

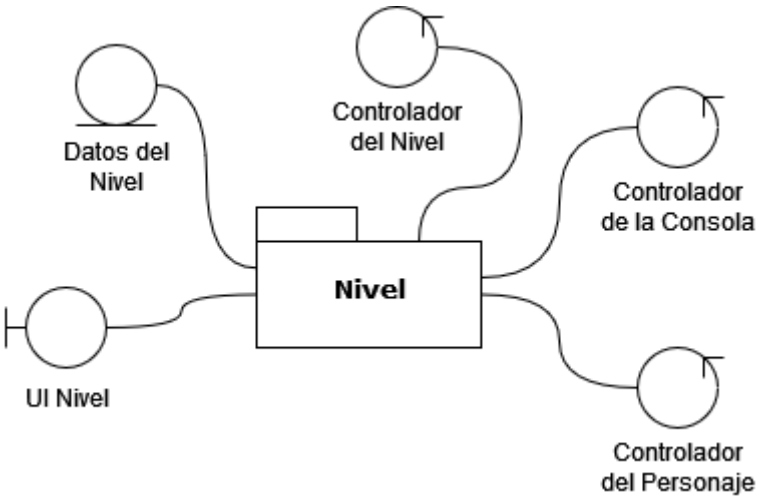


Figura 4.11: Diagrama de paquete del módulo Nivel

Fuente: Elaboración propia.

4.2 Diseño

En la fase de elaboración, el flujo de trabajo de diseño tiene como objetivo principal desarrollar una arquitectura detallada del sistema que soporte todos los requisitos establecidos, incluyendo tanto los funcionales como los no funcionales. Durante esta etapa, se avanza hacia una representación más concreta del sistema, asegurando que la arquitectura pueda manejar las restricciones y las necesidades identificadas en fases anteriores.

El diseño busca obtener una visión clara y estructurada de la arquitectura, modelando los componentes clave del sistema y estableciendo cómo se relacionan entre sí para implementar los casos de uso. Este proceso permite realizar la transición de los modelos abstractos a una solución técnica más definida, estableciendo la base para la implementación física del sistema. Así, el flujo de trabajo de diseño en esta fase se enfoca en depurar y refinar la estructura del sistema, preparando el camino para su construcción en fases posteriores.

4.2.1 Diagrama de Clase de Diseño

Este diagrama detalla las clases necesarias para implementar el sistema, especificando sus atributos, métodos, y las relaciones entre ellas. Aquí se incluyen las clases específicas de la lógica del juego, las opciones y la consola, y su relación con la aplicación y el motor de desarrollo. Es fundamental para definir cómo se implementarán los casos de uso y cómo interactuarán las diferentes partes del sistema a nivel de código.

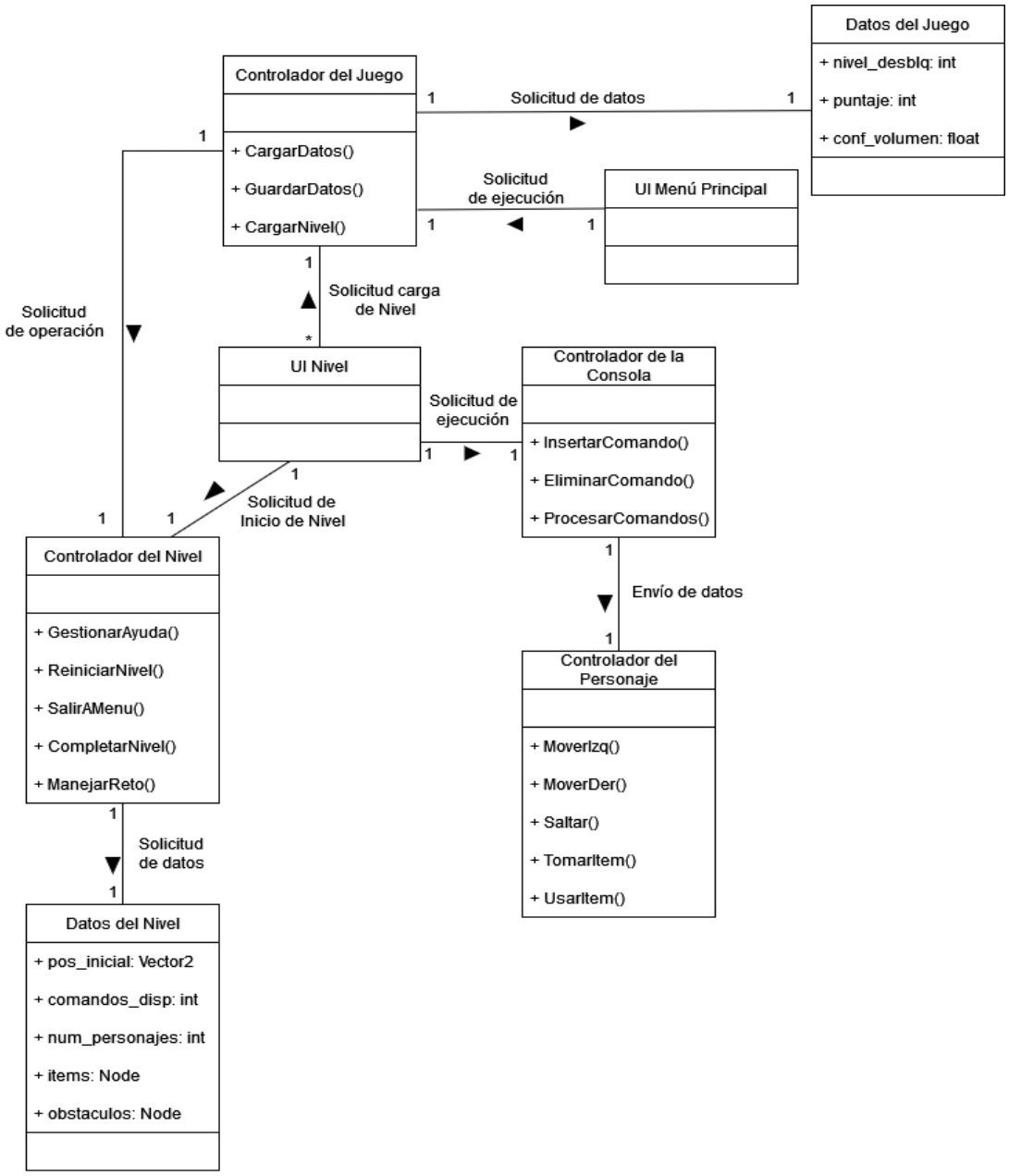


Figura 4.12: Diagrama de Clase de Diseño.

Fuente: Elaboración Propia.

4.2.2 Diagrama de Capas

Este diagrama define la estructura de la arquitectura del sistema, mostrando las diferentes capas y cómo interactúan entre sí. Facilita la comprensión de la separación de responsabilidades entre las diferentes capas del sistema y ayuda a mantener una arquitectura modular.

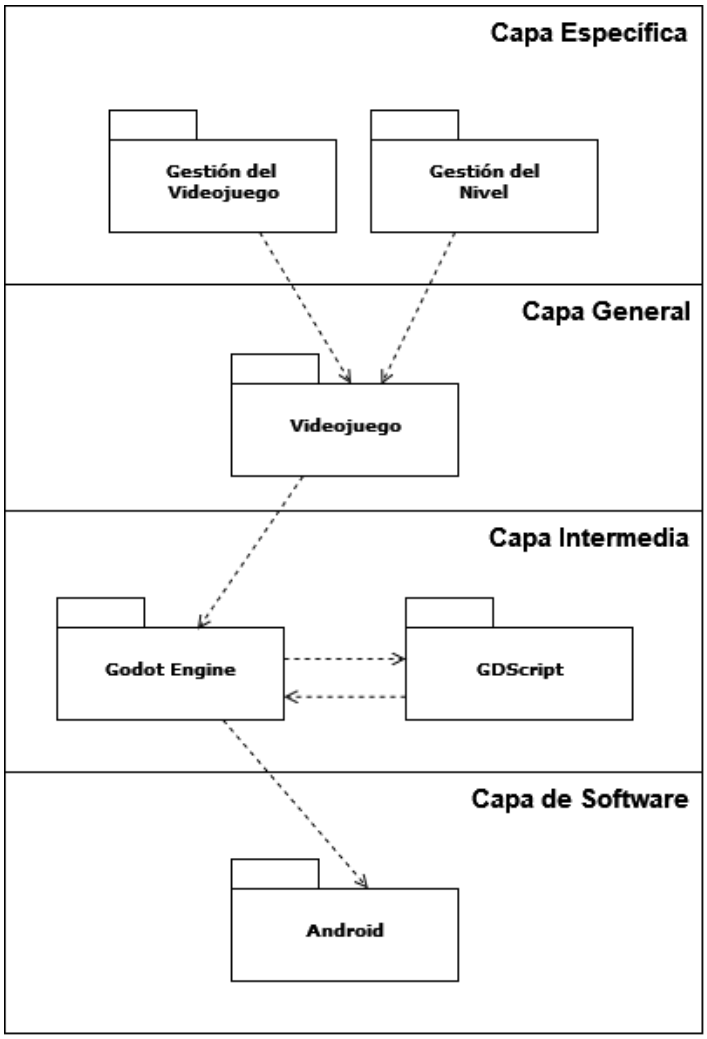


Figura 4.13: Diagrama de Capas

Fuente: Elaboración Propia

4.2.3 Diseño de Interfaz

El diseño de las interfaces se ha planteado considerando que el objetivo principal es que sea accesible y fácil de entender para la mayoría de los usuarios de cualquier habilidad en videojuegos. La interfaz inicial, presentada al cargar el juego, muestra el menú principal, desde el cual el jugador puede acceder a los niveles desbloqueados, las configuraciones de sonido y los créditos del juego.

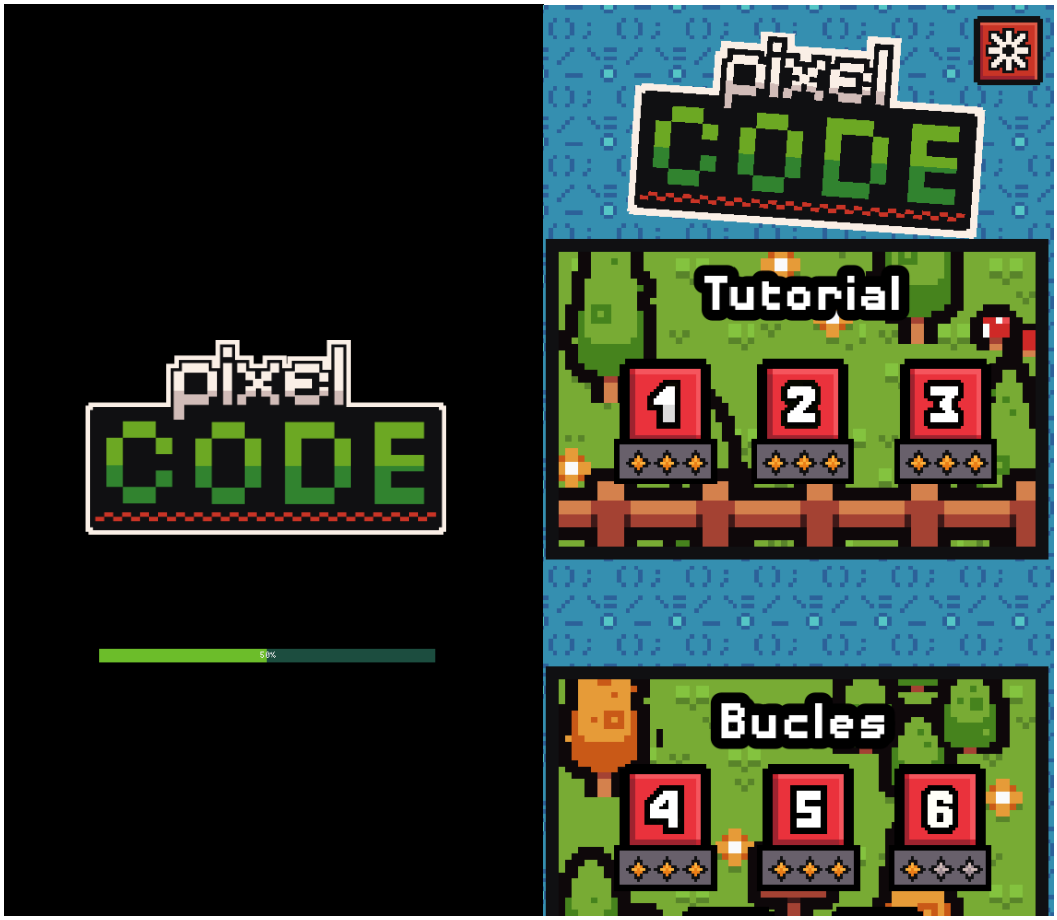


Figura 4.14: Pantalla de carga y pantalla de menú principal

Fuente: Elaboración Propia

La interfaz del nivel aparece una vez que el jugador selecciona un nivel. Esta interfaz está compuesta por varios elementos clave:

- Un panel de comandos con los botones necesarios para insertar instrucciones como "`moverDer()`", "`saltar()`", "`repetir`" y "definir función". Estos comandos son esenciales para que el jugador construya su solución al nivel presentado.
- Una consola donde se ingresan las soluciones mediante los comandos seleccionados, permitiendo que el jugador vea cómo se ejecuta su código.
- Botones para solicitar ayuda, ver los requisitos para obtener estrellas en el nivel y ajustar el sonido.

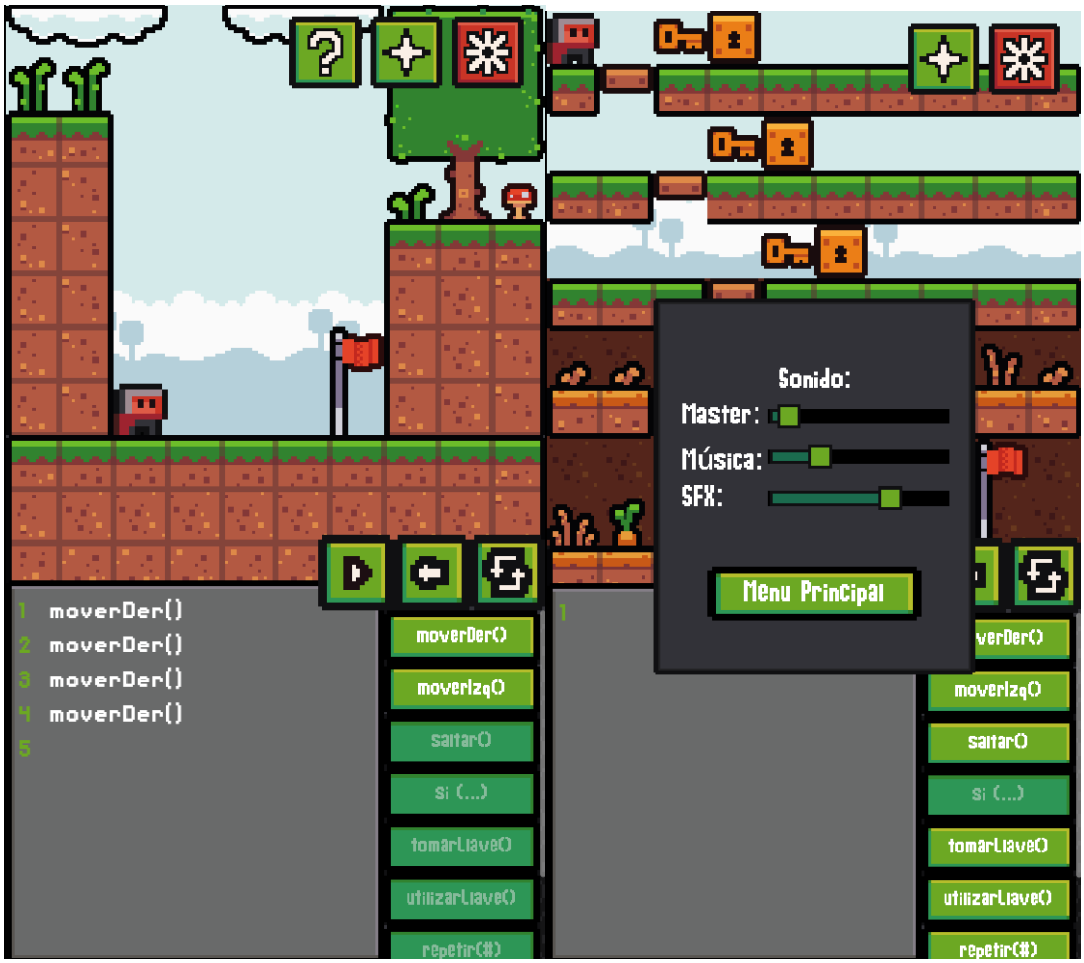


Figura 4.15: Interfaz de nivel, Panel de Opciones

Fuente: Elaboración Propia

4.3 Implementación

Este flujo de trabajo tiene como objetivo fundamental transformar el diseño del sistema en código funcional, siguiendo los lineamientos establecidos durante la etapa de diseño. En este punto del desarrollo, se lleva a cabo la creación de los componentes de software, basándose en las

arquitecturas y diagramas previamente definidos, como los diagramas UML de componentes y clases.

El enfoque de la implementación está en asegurar que cada componente se construya de manera modular, permitiendo su integración con otros elementos del sistema y garantizando que se cumplan tanto los requisitos funcionales como no funcionales. Se desarrollan las clases, métodos y servicios necesarios para cumplir con los casos de uso identificados, y se integran las bibliotecas o herramientas externas que soportan el funcionamiento del videojuego educativo.

4.3.1 Diagrama de Componentes

El diagrama de componentes se utiliza para identificar las relaciones y las dependencias técnicas que deben implementarse. Este diagrama muestra qué módulos necesitan ser desarrollados, cómo interactúan entre ellos y cuáles deben estar listos primero para garantizar una implementación funcional y coherente.

Al tener una vista clara de los componentes del sistema y sus interacciones, se facilita el proceso de construcción, permitiendo identificar de manera temprana cualquier posible problema de integración, riesgos técnicos o complejidades en la implementación de la arquitectura del juego.

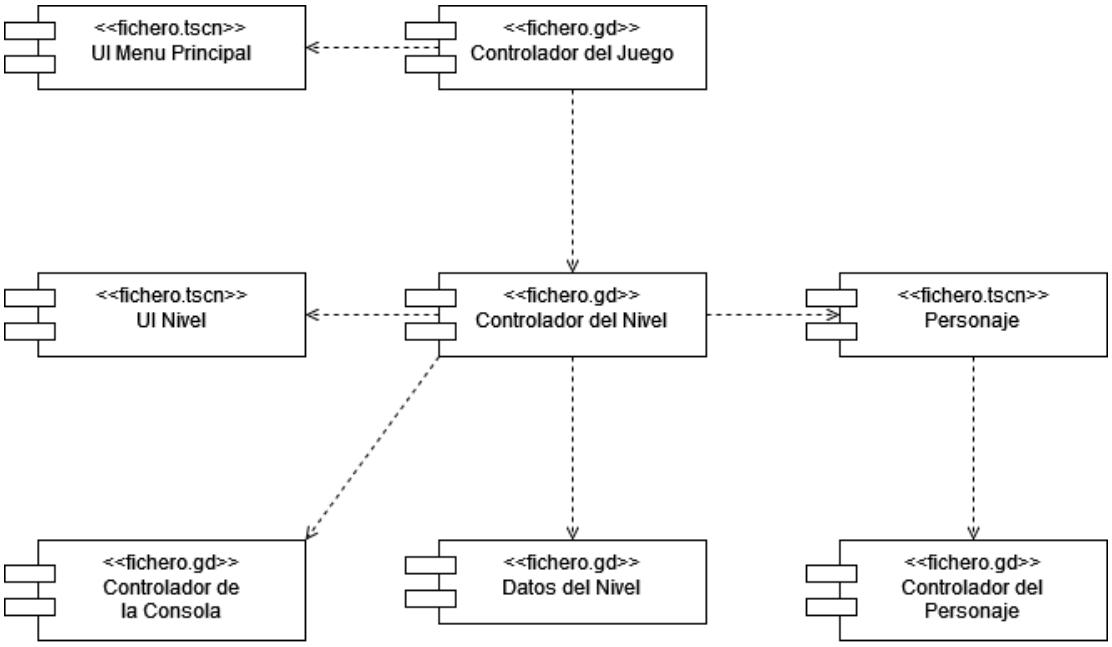


Figura 4.16: Diagrama de Componentes para el caso de uso “Seleccionar Nivel”

CAPÍTULO V
FASE DE CONSTRUCCIÓN

La fase de construcción tiene como objetivo principal lograr una versión funcional del software. En esta etapa, todos los componentes, características y requisitos identificados en las fases previas se implementan y adaptan según las capacidades de la herramienta de desarrollo seleccionada, asegurando que el software funcione de acuerdo con las especificaciones establecidas.

5.1 Herramientas de Desarrollo

5.1.1 Escogencia del motor gráfico

Se seleccionó Godot como motor de desarrollo para este proyecto debido a su flexibilidad, facilidad de uso, y la amplia gama de herramientas integradas que ofrece para la creación de videojuegos.

5.1.2 Escogencia del lenguaje de desarrollo

Se decidió usar GDScript ya que está diseñado para el desarrollo de videojuegos en este motor gráfico, y facilita el uso de sus herramientas y funciones integradas.

5.2 Pruebas

Las pruebas son esenciales para asegurar que el diseño de software cumpla con los requisitos funcionales establecidos. A través de estas pruebas, se valida que tanto los componentes individuales como el sistema en su conjunto funcionen correctamente. Estas pruebas están divididas en pruebas

unitarias, que se enfocan en la verificación de componentes individuales, y pruebas de integración, que comprueban la interacción correcta entre las distintas clases y módulos del sistema

5.2.1 Pruebas Unitarias

Las pruebas unitarias realizadas se centraron en verificar el funcionamiento de los componentes internos del sistema. El objetivo principal fue garantizar que cada elemento del sistema, de manera aislada, tradujera correctamente los comandos seleccionados en secuencias de instrucciones que luego serían ejecutadas por otros módulos

5.2.1.1 Prueba procesamiento de bucles

Esta prueba verifica que el sistema procese correctamente un comando `repetir`, traduciéndolo a una secuencia interna de instrucciones repetidas el número de veces especificado, con el fin de que simule correctamente un bucle.

```
repetir 3:
    moverIzq()
```

Algoritmo 5.1

Prueba:

Si se ingresa el Algoritmo 5.1 a la consola, la función correspondiente a la simulación del bucle debe traducir las instrucciones ingresadas a la siguiente secuencia: `moverIzq() moverIzq() moverIzq().`

Resultado:

La función tradujo el algoritmo correctamente a la secuencia esperada, almacenando internamente el comando `moverIzq()` tres veces seguidas.

5.2.1.2 Prueba de procesamiento de bucles anidados

Esta prueba verifica que el sistema pueda procesar correctamente un bucle anidado, es decir, un bucle dentro de otro bucle.

Algoritmo 5.2

```
repetir 2:
    saltar(der)
    repetir 3:
         moverDer()
```

Prueba:

Si se ingresa el Algoritmo 5.2 a la consola, la función correspondiente a la simulación del bucle debe generar la secuencia: `saltar(der) moverDer() moverDer() moverDer() saltar(der) moverDer() moverDer() moverDer()`

Resultado:

La función tradujo el algoritmo correctamente a la secuencia esperada, y fue almacenada internamente en la lista de comandos a procesar.

5.2.1.3 Prueba de creación de funciones

Esta prueba valida que el sistema procese correctamente la creación de una función definida por el jugador, con el fin de garantizar que el conjunto de instrucciones asociados a una función se almacene correctamente en el sistema.

Algoritmo 5.3

```
def func1():
    saltar(arriba)
    repetir 4:
         moverIzq()
```

Prueba:

Si se ingresa el Algoritmo 5.3 en la consola, el sistema debe de almacenar las instrucciones internas dentro de un diccionario con el nombre y el contenido correcto, para así asegurar su uso a futuro.

Resultado:

La función fue almacenada correctamente dentro de un diccionario, cuya clave es el nombre de la función, y su valor la serie de instrucciones definidas por el jugador.

5.2.1.4 Prueba de Ejecución de Funciones

Esta prueba valida que el sistema procese correctamente el llamado de una función definida por el jugador.

```
func1()
```

Algoritmo 5.4

Prueba:

Si se ingresa el Algoritmo 5.4 en la consola, el sistema debe tomar la serie de instrucciones asociadas a la función desde el diccionario encargado de almacenar las funciones definidas previamente por el jugador.

Resultado:

El sistema almacenó todos los comandos asociados a la función dentro de la lista de comandos a ejecutar de forma correcta. En este caso son las siguientes instrucciones: `saltar(arriba) moverIzq() moverIzq() moverIzq() moverIzq()`.

5.2.2 Pruebas de Integración

Las pruebas de integración se enfocan en evaluar la interacción entre los diferentes módulos del sistema, asegurando que la comunicación entre la consola de comandos, el controlador del personaje, y el sistema de guardado funcionara sin problemas. Estas pruebas fueron clave para garantizar que, una

vez integrados, los componentes individuales trabajen de manera coherente y fluida.

5.2.2.1 Control del movimiento del personaje a través de comandos

Esta prueba verifica que el sistema procese los comandos ingresados en la consola y los comunique correctamente al controlador del personaje, para que este ejecute los movimientos en el mapa.

Algoritmo 5.4

```
moverDer()
moverDer()
saltar(der)
moverDer()
```

Prueba:

Si se ingresa el Algoritmo 5.4 a la consola y se presiona el botón “Ejecutar” en la interfaz, el personaje deberá moverse de la forma que indiquen las instrucciones ingresadas. En este caso deberá moverse dos veces a la derecha, saltar hacia la derecha, y finalmente moverse a la derecha una vez más dentro del mapa.

Resultado:

El personaje se mueve correctamente de la forma especificada en la consola.

5.2.2.2 Progreso de niveles

Esta prueba valida que el sistema gestione correctamente el progreso entre niveles al completar un nivel. El objetivo es asegurar que, al finalizar un nivel, el siguiente se desbloquee correctamente y el progreso del jugador se guarde en el archivo de datos. Se verifica también que, al reiniciar el juego, el progreso del jugador se mantenga intacto y el nuevo nivel desbloqueado esté disponible en el menú principal.

Prueba:

Se selecciona un nivel desbloqueado desde el menú principal y se completa el nivel siguiendo las instrucciones necesarias. Al alcanzar la meta, el sistema marca el nivel como completado, desbloquea el siguiente nivel, y guarda el progreso del jugador en el archivo de datos automáticamente. Posteriormente, se reinicia el juego, y al volver al menú principal, el sistema carga el progreso desde el archivo de datos y debe mostrar el nuevo nivel desbloqueado en el menú principal.

Resultado:

El sistema desbloqueó correctamente el siguiente nivel, guardó el progreso en el archivo de datos, y tras reiniciar el juego, el nuevo nivel se mantuvo marcado como desbloqueado en el menú principal.

5.3 Código de los componentes

5.3.1 Interfaz.gd

```
extends Control
var running = false
var current_scene
var character_1 = $"../Control/Character1"
var character_2 = ""
var character_3 = ""
@onready var console_text = $Console/ConsoleBG/ConsoleText
@onready var popup_panel = $Console/ScrollContainer/Commands/PopupPanel

@onready var finish_panel = $Finish_Panel

@onready var master_slider =
$Volume/Panel/MarginContainer/VBoxContainer/GridContainer/MasterSlider
@onready var music_slider =
$Volume/Panel/MarginContainer/VBoxContainer/GridContainer/MusicSlider
@onready var sfx_slider =
$Volume/Panel/MarginContainer/VBoxContainer/GridContainer/SFXSlider
@onready var sfx_button = $sfx_button

@onready var text_edit = $Console/ConsoleBG/TextEdit

@onready var run_button = $Console/Edit/run
@onready var delete_button = $Console/Edit/delete
@onready var restart_button = $Console/Edit/restart

@onready var mover_der_button = $"Console/ScrollContainer/Commands/moverDer()"
@onready var mover_izq_button = $"Console/ScrollContainer/Commands/moverIzq()"
@onready var saltar_button = $"Console/ScrollContainer/Commands/saltar()"
@onready var tomar_llave_button = $"Console/ScrollContainer/Commands/tomarLlave()"
```

```
@onready var utilizar_llave_button = $"Console/ScrollContainer/Commands/utilizarLlave()"
@onready var saltar_arriba_button =
$"Console/ScrollContainer/Commands/PopupPanel/HBoxContainer/saltar(arriba)"
@onready var saltar_izq_button =
$"Console/ScrollContainer/Commands/PopupPanel/HBoxContainer/saltar(izq)"
@onready var saltar_der_button =
$"Console/ScrollContainer/Commands/PopupPanel/HBoxContainer/saltar(der)"
@onready var saltar_abajo_button =
$"Console/ScrollContainer/Commands/PopupPanel/HBoxContainer/saltar(abajo)"
@onready var rep_2 = $"Console/ScrollContainer/Commands/repetir_popup/HBoxContainer/reps_2"
@onready var rep_3 = $"Console/ScrollContainer/Commands/repetir_popup/HBoxContainer/reps_3"
@onready var rep_4 = $"Console/ScrollContainer/Commands/repetir_popup/HBoxContainer/reps_4"
@onready var rep_5 = $"Console/ScrollContainer/Commands/repetir_popup/HBoxContainer/reps_5"
@onready var rep_6 = $"Console/ScrollContainer/Commands/repetir_popup/HBoxContainer/reps_6"
@onready var rep_7 = $"Console/ScrollContainer/Commands/repetir_popup/HBoxContainer/reps_7"
@onready var rep_8 = $"Console/ScrollContainer/Commands/repetir_popup/HBoxContainer/reps_8"
@onready var rep_9 = $"Console/ScrollContainer/Commands/repetir_popup/HBoxContainer/reps_9"
@onready var repetir_popup = $Console/ScrollContainer/Commands/repetir_popup
@onready var repetir___ = $"Console/ScrollContainer/Commands/repetir(#)"
@onready var repetir_button = $"Console/ScrollContainer/Commands/repetir(#)"
@onready var funcion_crear = $Console/ScrollContainer/Commands/funcionCrear
@onready var func_1_popup = $Console/ScrollContainer/Commands/func1_popup
@onready var crear_1 = $"Console/ScrollContainer/Commands/func1_popup/HBoxContainer/crear1"
@onready var crear_2 = $"Console/ScrollContainer/Commands/func1_popup/HBoxContainer/crear2"
@onready var crear_3 = $"Console/ScrollContainer/Commands/func1_popup/HBoxContainer/crear3"
@onready var crear_4 = $"Console/ScrollContainer/Commands/func1_popup/HBoxContainer/crear4"
@onready var crear_5 = $"Console/ScrollContainer/Commands/func1_popup/HBoxContainer/crear5"
@onready var funcion_usar = $Console/ScrollContainer/Commands/funcionUsar
@onready var func_2_popup = $Console/ScrollContainer/Commands/func2_popup
@onready var usar_1 = $"Console/ScrollContainer/Commands/func2_popup/HBoxContainer/usar1"
@onready var usar_2 = $"Console/ScrollContainer/Commands/func2_popup/HBoxContainer/usar2"
@onready var usar_3 = $"Console/ScrollContainer/Commands/func2_popup/HBoxContainer/usar3"
@onready var usar_4 = $"Console/ScrollContainer/Commands/func2_popup/HBoxContainer/usar4"
@onready var usar_5 = $"Console/ScrollContainer/Commands/func2_popup/HBoxContainer/usar5"
```

```
@onready var hint_button = $Options/hint
var level_scene = get_parent()

var instructions_list = []
var instructions_colors = {
	"rojo":[],
	"azul":[],
	"verde":[]
}
var colors = ["rojo","verde","azul"]
var level_hint = null
var no_hint_levels = ["Level5","Level6", "Level8", "Level10", "Level11", "Level12", "Level13",
"Level15", "Level16", "Level17", "Level18"]
var curr_level_has_no_hint = null
var buttons = []

func check_hints():
	if curr_level_has_no_hint:
		hint_button.visible = false
	else:
		level_hint = $"../Hint"
var text_index = 0

func if_logic(if_line):
	var next_line = text_edit.get_line(text_index + 1)
	var indent_level = text_edit.get_indent_level(text_index + 1)
	var commands = []
	var color = ""
	var equal_sign = true
	if if_line.contains("!="):
		equal_sign = false
	if if_line.contains("rojo"):
		color = "rojo"
	elif if_line.contains("azul"):
```

```
                color = "azul"
        elif if_line.contains("verde"):
                color = "verde"
        while text_edit.get_indent_level(text_index + 1) == indent_level:
                var empty = next_line.is_empty() or next_line.ends_with("\t")
                if next_line.contains("repetir"):
                        text_index += 1
                        commands+= loop_logic(next_line)
                elif next_line.contains("def func"):
                        def_func_logic(next_line)
                elif next_line.contains("func") and !next_line.contains("def func"):
                        var func_name = "func" + next_line[-3]
                        if functions.has(func_name):
                                commands += functions[func_name]
                        text_index += 1
                elif !empty:
                        commands.append(next_line.lstrip("\t"))
                        text_index += 1
                else:
                        text_index += 1
                if text_index < text_edit.get_line_count():
                        next_line = text_edit.get_line(text_index + 1)
        equal_or_not(equal_sign, color, commands)
        print(instructions_colors)


func equal_or_not(equal_sign, color, commands):
        if equal_sign:
                instructions_colors[color] += commands
        else:
                for color_aux in level_scene.active_colors:
                        if color_aux != color:
                                instructions_colors[color_aux] += commands
```

```
func loop_logic(loop_line):
	var repeats = str_to_var(loop_line.lstrip("\t")[8])
	var next_line = text_edit.get_line(text_index + 1)
	var unraveled_commands = []
	var aux_list = []
	var indent_level = text_edit.get_indent_level(text_index + 1)
	while text_edit.get_indent_level(text_index + 1) == indent_level:
		var empty = next_line.is_empty() or next_line.ends_with("\t")
		if next_line.contains("repetir"):
			text_index += 1
			unraveled_commands += loop_logic(next_line)
		elif next_line.contains("def func"):
			def_func_logic(next_line)
		elif next_line.contains("func") and !next_line.contains("def func"):
			var func_name = "func" + next_line[-3]
			if functions.has(func_name):
				unraveled_commands += functions[func_name]
			text_index += 1
		elif next_line.contains("si color"):
			text_index += 1
			if_logic(next_line)
		elif !empty:
			unraveled_commands.append(next_line.lstrip("\t"))
			text_index += 1
		else:
			text_index += 1
		if text_index < text_edit.get_line_count():
			next_line = text_edit.get_line(text_index + 1)
	var j = 0
	while j < repeats:
		aux_list += unraveled_commands
		j += 1
	return aux_list
```

```
var functions = {}

func def_func_logic(def_func_line):
	var func_name = "func" + def_func_line[-4]
	var func_body = []
	var next_line = text_edit.get_line(text_index + 1)
	var indent_level = text_edit.get_indent_level(text_index + 1)

	while text_edit.get_indent_level(text_index + 1) == indent_level:
		var empty = next_line.is_empty() or next_line.ends_with("\t")

		if next_line.contains("repetir"):
			text_index += 1
			func_body += loop_logic(next_line)
		elif next_line.contains("def func"):
			text_index += 1
			def_func_logic(next_line)
		elif !empty:
			func_body.append(next_line.lstrip("\t"))
			text_index += 1
		else:
			text_index += 1
		if text_index < text_edit.get_line_count():
			next_line = text_edit.get_line(text_index + 1)
	functions[func_name] = func_body

func add_global_command(command):
	for color in level_scene.active_colors:
		instructions_colors[color] += command

func line_to_array():
	var total_lines = text_edit.get_line_count() - 1
	while text_index in range(total_lines):
		var curr_line = text_edit.get_line(text_index)
```

```
                var empty = curr_line.is_empty() or curr_line.ends_with("\t")
                if curr_line.contains("repetir"):
                        add_global_command(loop_logic(curr_line))
                elif curr_line.contains("def func"):
                        def_func_logic(curr_line)
                elif curr_line.contains("func") and !curr_line.contains("def func"):
                        var func_name = "func" + curr_line[-3]
                        if functions.has(func_name):
                                add_global_command(functions[func_name])


                elif curr_line.contains("si color"):
                        if_logic(curr_line)
                elif !empty:
                        add_global_command([curr_line])
                text_index += 1

func buttons_pressed():
        for button in buttons:
                var command = ""
                var current_name = ""+button.name
                if current_name.contains("reps_"):
                        command = "repetir " + current_name[-1] + ":"
                elif current_name.contains("funcionCrear"):
                        command = "def func" #+ var_to_str(function_number) + "():"
                elif current_name.contains("usar"):
                        command = "func" + current_name[-1] + "()"
                else:
                        command = button.name
                button.pressed.connect(self.add_command.bind(command))
func _on_jump_button_pressed():
        popup_panel.set_position(saltar_button.get_global_position())
        popup_panel.popup()

func _on_repetir_pressed():
```

```
	repetir_popup.set_position(repetir_button.get_global_position())
	repetir_popup.popup()

func _on_funcion_crear_pressed():
	pass

func _on_funcion_usar_pressed():
	func_2_popup.set_position(funcion_usar.get_global_position())
	func_2_popup.popup()

var function_number = 1
func automatic_func_naming():
	function_number = 1
	var index = 0
	var total_lines = text_edit.get_line_count() - 1

	while index in range(total_lines):
		if text_edit.get_line(index).contains("def func"):
			function_number += 1
		index += 1

func indent_creator(index):
	var indent = ""
	var i = 0
	while i in range(index):
		indent += "\t"
		i += 1
	return indent

func add_command(command):
	if not running:
		automatic_func_naming()
		if command.contains("def func"):
			if function_number > 5:
```

```
                    return
                command = command + var_to_str(function_number) + "():"

            var line_number = text_edit.get_caret_line()
            if line_number < level_scene.max_input:
                var curr_line = text_edit.get_line(line_number)
                var requires_indent = curr_line.contains("repetir") or
curr_line.contains("def func") or curr_line.contains("si color")
                var empty = curr_line.is_empty() or curr_line.ends_with("\t")
                var indent_level = text_edit.get_indent_level(line_number)/4
                var indents_to_add = ""

                if indent_level > 5:
                    return

                if !empty:
                    line_number += 1
                    if requires_indent:
                        indents_to_add = "\t" + indent_creator(indent_level)
                if indent_level >= 1 and !requires_indent:
                    indents_to_add = indent_creator(indent_level)

                text_edit.insert_line_at(line_number,indents_to_add+command)

                if command.contains("repetir") or command.contains("def func") or
command.contains("si color"):
                    indents_to_add += "\t"
                    text_edit.insert_line_at(line_number + 1, indents_to_add)
                text_edit.set_caret_line(line_number+1)

func run_commands():
    if "active_colors" in level_scene:
                for color in level_scene.active_colors:
                    match color:
```

```
                                        "rojo":

        character_1.parse_command(instructions_colors.rojo)
                                        "verde":

        character_2.parse_command(instructions_colors.verde)
                                        "azul":

        character_3.parse_command(instructions_colors.azul)
                                    prints("color active: ", color)
func _on_run_pressed():
        if not running:
                if "active_colors" in level_scene:
                        instructions_colors = {
                                "rojo":[],
                                "azul":[],
                                "verde":[]
                        }
                global.reset_level(level_scene, "key_flags" in level_scene)
                text_index = 0

                running = true
                global.data.level_solutions[current_scene][0] = text_edit.get_line_count() - 1
                global.save_game()

                line_to_array()
                print("running commands")
                await run_commands()if=multiple_chars ?
                running = false

func _on_delete_pressed():
        if not running:
                var total_length = text_edit.get_line_count()
                var curr_line = text_edit.get_caret_line()
```

```
                var prev_line = curr_line
                var prev_line_length = 0
                var caret_position = text_edit.get_caret_column()
                if total_length-1 == curr_line:
                        if total_length >1:
                                text_edit.remove_text(curr_line-1,0,curr_line,0)
                elif curr_line == 0 and total_length > 1:
                        text_edit.remove_text(curr_line,0,curr_line+1,0)

                elif total_length > 1:
                        if curr_line >=1:
                                prev_line = curr_line-1
                                prev_line_length = text_edit.get_line(prev_line).length()
                        text_edit.remove_text(prev_line,prev_line_length, curr_line,
caret_position)
func _on_restart_pressed():
        var level = level_scene
        if "characters" in level:
                for character in level.characters:
                        var temp_str = "character_" + str(character+1)
                        var temp_post = "position_" + str(character+1)
                        level[temp_str].flip(false)
                        level[temp_str].position = level[temp_post]
        else:
                level.character_1.flip(false)
                level.character_1.position = level.initial_position
        instructions_list = []
        get_tree().reload_current_scene()


@onready var transition = $transition

func _ready():
        transition.play("fade_in")
```

```
	level_scene = get_parent()
	if "active_colors" in level_scene:
		if "azul" in level_scene.active_colors:
			character_3 = $"../Control/Character3"
		if "verde" in level_scene.active_colors:
			character_2 = $"../Control/Character2"

	curr_level_has_no_hint = level_scene.name in no_hint_levels
	buttons = [mover_der_button, mover_izq_button, tomar_llave_button,
utilizar_llave_button,saltar_arriba_button,saltar_izq_button,saltar_der_button,saltar_abajo_button,
rep_2, rep_3, rep_4, rep_5, rep_6, rep_7, rep_8,
rep_9,funcion_crear,usar_1,usar_2,usar_3,usar_4,usar_5]
	buttons_pressed()
	if_buttons = [rojo,verde,azul,equal,not_equal]
	if_variables_pressed()
	var buttons_main = [mover_der_button, mover_izq_button, saltar_button,
tomar_llave_button, utilizar_llave_button, repetir_button, funcion_crear, funcion_usar,
si_loop_button]

	text_edit.virtual_keyboard_enabled = false
	text_edit.scroll_past_end_of_file = false
	text_edit.caret_blink = true
	text_edit.draw_tabs = true
	master_slider.value = db_to_linear(global.data.volume_settings.master)
	music_slider.value = db_to_linear(global.data.volume_settings.music)
	sfx_slider.value = db_to_linear(global.data.volume_settings.sfx)
	current_scene = get_tree().current_scene.name

	for index in level_scene.active_buttons:
		buttons_main[index].disabled = false
	check_hints()
func _process(_delta):
	var line_idx = text_edit.get_caret_line()
	var line_length = text_edit.get_line(line_idx).length()
```

```
		text_edit.set_caret_line(line_idx)
		text_edit.set_caret_column(line_length)
		if level_scene.panel_up:
			finish_panel.visible = true


@onready var volume = $Volume
@onready var MASTER_BUS_ID = AudioServer.get_bus_index("Master")
@onready var MUSIC_BUS_ID = AudioServer.get_bus_index("Music")
@onready var SFX_BUS_ID = AudioServer.get_bus_index("SFX")

func _on_hint_pressed():
		sfx_button.play()
		level_hint.visible = !level_hint.visible
		if level_scene.has_method("reset_hint"):
			level_scene.reset_hint()
		if level_hint.visible:
			volume.visible = false
			rating.visible = false

func _on_volume_pressed():
		sfx_button.play()
		volume.visible = !volume.visible
		if volume.visible:
			rating.visible = false
		if !curr_level_has_no_hint:
			level_hint.visible = false


func _on_master_slider_value_changed(value):
		global.data.volume_settings.master = linear_to_db(value)
		AudioServer.set_bus_volume_db(MASTER_BUS_ID, global.data.volume_settings.master)
		AudioServer.set_bus_mute(MASTER_BUS_ID, value < 0.05)
		global.save_game()
```

```
func _on_music_slider_value_changed(value):
	global.data.volume_settings.music = linear_to_db(value)
	AudioServer.set_bus_volume_db(MUSIC_BUS_ID, global.data.volume_settings.music)
	AudioServer.set_bus_mute(MUSIC_BUS_ID, value < 0.05)
	global.save_game()

func _on_sfx_slider_value_changed(value):
	global.data.volume_settings.sfx = linear_to_db(value)
	AudioServer.set_bus_volume_db(SFX_BUS_ID, global.data.volume_settings.sfx)
	AudioServer.set_bus_mute(SFX_BUS_ID, value < 0.05)
	global.save_game()

func _on_main_menu_pressed():
	get_tree().change_scene_to_file("res://Scenes/main_menu.tscn")

@onready var rating = $Rating
func _on_stars_pressed():
	sfx_button.play()
	rating.visible = !rating.visible
	if rating.visible:
		volume.visible = false
	if !curr_level_has_no_hint:
		level_hint.visible = false

@onready var si_popup = $si_popup
@onready var si_loop_button = $Console/ScrollContainer/Commands/siLoop
@onready var equal_button = $si_popup/MarginContainer/HBoxContainer/equalButton
@onready var color_button = $si_popup/MarginContainer/HBoxContainer/colorButton
@onready var confirm_button = $si_popup/MarginContainer/HBoxContainer/ifConfirm
@onready var equal_popup = $equal_popup
@onready var color_popup = $color_popup
@onready var rojo = $color_popup/HBoxContainer/rojo
```

```
@onready var verde = $color_popup/HBoxContainer/verde
@onready var azul = $color_popup/HBoxContainer/azul
@onready var equal = $equal_popup/HBoxContainer/equal
@onready var not_equal = $equal_popup/HBoxContainer/not_equal
var short_colors = ["R", "V", "A"]
var if_buttons = []
func if_variables_pressed():
	for button in if_buttons:
		button.pressed.connect(self.if_button_handler.bind(button.text))
func if_button_handler(button_name):
	print(button_name)
	if button_name in short_colors:
		color_button.text = button_name
		color_popup.visible = false
	else:
		equal_button.text = button_name
		equal_popup.visible = false

func _on_si_loop_pressed():
	si_popup.set_position(si_loop_button.get_global_position()+Vector2(-250,-50))
	si_popup.visible = true

func _on_equal_button_pressed():
	equal_popup.set_position(equal_button.get_global_position()+Vector2(0,0))
	equal_popup.visible = !equal_popup.visible
	color_popup.visible = false

func _on_color_button_pressed():
	color_popup.set_position(color_button.get_global_position()+Vector2(0,0))
	color_popup.visible = !color_popup.visible
	equal_popup.visible = false

func _on_if_confirm_pressed():
	if color_button.text in short_colors:
```

```
            var full_color = ''
            match color_button.text:
                "R":
                    full_color = "rojo"
                "V":
                    full_color = "verde"
                "A":
                    full_color = "azul"
            add_command("si color " + equal_button.text + " " + full_color + ":")
            si_popup.visible = false


func _on_si_popup_focus_exited():
    print("popup lost focus")
    si_popup.visible = false
    color_popup.visible = false
    equal_popup.visible = false
```

5.3.2 Global.gd

```
extends Node
var data = {
    "volume_settings": {
        "master" = 1,
        "music" = 1,
        "sfx" = 1
    },
    "unlocked_levels": 1,
    "level_solutions": {
        "Level1" = [0, [5,6,10],0, false],
        "Level2" = [0, [5,6,10],0, false],
        "Level3" = [0, [12,15,20],0, false],
        "Level4" = [0, [5,5,10],0, false],
```

```
                "Level5" = [0, [8,13,17],0, false],
                "Level6" = [0, [12,15,20],0, false],
                "Level7" = [0, [9,9,11],0, false],
                "Level8" = [0, [11,13,15],0, false],
                "Level9" = [0, [8,9,13],0, false],
                "Level10" = [0, [21,23,25],0, false],
                "Level11" = [0, [20,20,23],0, false],
                "Level12" = [0, [14,15,17],0, false],
                "Level13" = [0, [12,15,17],0, false],
                "Level14" = [0, [4,8,15],0, false],
                "Level15" = [0, [14,18,20],0, false],
                "Level16" = [0, [13,15,20],0, false],
                "Level17" = [0, [7,13,15],0, false],
                "Level18" = [0, [15,20,25],0, false],
        }
}
var path = "user://save.json"
const TILE_SIZE = Vector2(72,72)
const tilesize = 18*4
func save_game():
        var file = FileAccess.open(path, FileAccess.WRITE)
        if file:
                file.store_string(JSON.stringify(data))
                file.close()
        else:
                print("Could not save data.")

func load_game():
        var file = FileAccess.open(path, FileAccess.READ)
        if file:
                global.data = str_to_var(file.get_as_text())
                file.close()
        else:
                print("No save file found.")
```

```
func wall_up(level):
	var tween = get_tree().create_tween()
	var target_position = level.wall_initial_position + (Vector2.UP * 2) * TILE_SIZE
	tween.tween_property(level.wall, "position", target_position, 0.35)
	await tween.finished
	level.wall.position = target_position
	level.wall_up = true

func reset_level(level, key_flags):
	if key_flags:
		for key in level.keys:
			level.keys[key] = false
		for box in level.boxes:
			level.boxes[box] = false
		level.reset_visibility()
		level.wall_up  = false
		level.wall.position = level.wall_initial_position
		level.inside_key = false
		level.inside_keybox = false
		level.box_solved = false
	if "characters" in level:
		for character in level.characters:
			var temp_str = "character_" + str(character+1)
			var temp_post = "position_" + str(character+1)
			level[temp_str].flip(false)
			level[temp_str].position = level[temp_post]
	else:
		level.character_1.flip(false)
		level.character_1.position = level.initial_position

func calculate_score(level_name):
	if data.level_solutions[level_name][0] <= data.level_solutions[level_name][1][0]:
```

```
            if data.level_solutions[level_name][2] < 3:
                data.level_solutions[level_name][2] = 3
            return 3
        elif data.level_solutions[level_name][0] == data.level_solutions[level_name][1][1]:
            if data.level_solutions[level_name][2] < 2:
                data.level_solutions[level_name][2] = 2
            return 2
        else:
            if data.level_solutions[level_name][2] <=1:
                data.level_solutions[level_name][2] = 1
            return 1
```

5.3.3 Personaje.gd

```
extends CharacterBody2D

const SPEED = 600.0
const JUMP_VELOCITY = -250.0
var detected_area = null

@onready var sprite_1 = $sprite1
@onready var ray_cast_2d_down = $RayCast2D_DOWN
@onready var ray_cast_2d_right = $RayCast2D_RIGHT
@onready var ray_cast_2d_up = $RayCast2D_UP
@onready var ray_cast_2d_left = $RayCast2D_LEFT
@onready var ray_cast_2d_diag_r = $RayCast2D_DIAG_R
@onready var ray_cast_2d_diag_l = $RayCast2D_DIAG_L
@onready var ray_cast_2d_diag_rd = $RayCast2D_DIAG_RD
@onready var ray_cast_2d_diag_ld = $RayCast2D_DIAG_LD
@onready var sfx_jump = $sfx_jump
@onready var sfx_use = $sfx_use
@onready var sfx_pickup = $sfx_pickup
```

```
var level_scene = ''

func _ready():
	level_scene = get_parent().get_parent()
var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")

func _physics_process(delta):
	if not is_on_floor():
		velocity.y += (gravity*2) * delta
	move_and_slide()

func raycast_update():
	ray_cast_2d_down.force_raycast_update()
	ray_cast_2d_left.force_raycast_update()
	ray_cast_2d_right.force_raycast_update()
	ray_cast_2d_up.force_raycast_update()
	ray_cast_2d_diag_l.force_raycast_update()
	ray_cast_2d_diag_r.force_raycast_update()
	ray_cast_2d_diag_rd.force_raycast_update()
	ray_cast_2d_diag_ld.force_raycast_update()

func parse_command(instructions_list):

	while !is_on_floor():
		await get_tree().create_timer(0.1).timeout
	sprite_1.play("walk")
	prints("in command parse, this is the list: ", instructions_list)
	for command in instructions_list:
		raycast_update()
		match command:
			"moverDer()":
				if !ray_cast_2d_right.is_colliding():
					flip(false)
					await mover(Vector2.RIGHT,1,0.35)
```

```
                    "moverIzq()":
                        if !ray_cast_2d_left.is_colliding():
                            flip(true)
                            await mover(Vector2.LEFT,1,0.35)
                    "saltar(izq)":
                        flip(true)
                        await saltar("diag_left")
                    "saltar(der)":
                        flip(false)
                        await saltar("diag_right")
                    "saltar(arriba)":
                        await saltar("up")
                    "saltar(abajo)":
                        await saltar("down")
                    "tomarLlave()":
                        await pickUp()
                    "utilizarLlave()":
                        await utilizarLlave()

            await get_tree().create_timer(0.2).timeout
            while not is_on_floor():
                await get_tree().create_timer(0.2).timeout
        sprite_1.stop()
        sprite_1.play("default")

func pickUp():
    if level_scene.inside_key:
        sfx_pickup.play()
        detected_area.visible = false
        level_scene.keys[level_scene.curr_key] = true
        detected_area = null

func utilizarLlave():
    var key = level_scene.find_key()
```

```
        if key:
                if level_scene.inside_keybox and level_scene.keys[key]:
                        if detected_area:
                                sfx_use.play()
                                detected_area.visible = false
                                print("key used!")
                                level_scene.keys[key] = false


                                level_scene.boxes[level_scene.curr_box] = true
                                level_scene.all_boxes_solved()
        else: print("no key in hand.")

func get_area(area):
        detected_area = area

func flip(flag):
        sprite_1.flip_h = flag

func saltar(direction=null):
        sprite_1.play("walk")
        sfx_jump.play()
        var target: Vector2
        while !is_on_floor():
                await get_tree().create_timer(0.1).timeout
        var ray_diag_up
        var ray_col
        if direction == "diag_right" or direction == "diag_left":
                if !ray_cast_2d_up.is_colliding():


                        match direction:
                                "diag_right":
                                        ray_diag_up = ray_cast_2d_diag_r
                                        ray_col = ray_cast_2d_right
                                        target = Vector2(1,1)
```

```
                    "diag_left":
                        ray_diag_up = ray_cast_2d_diag_l
                        ray_col = ray_cast_2d_left
                        target = Vector2(-1,1)
                if !ray_diag_up.is_colliding() and !ray_col.is_colliding():
                    await mover(target*Vector2(0.5,-0.5),1, 0.15)
                    await mover(target*Vector2(0.5,0.5),1, 0.15)
                elif !ray_diag_up.is_colliding() and ray_col.is_colliding():
                    await mover(target*Vector2(0.15,-0.5),1, 0.1)
                    await mover(target*Vector2(0.15,-0.5),1, 0.05)
                    await mover(target*Vector2(0.20,-0.5),1, 0.05)
                    await mover(target*Vector2(0.25,0.25),1, 0.08)
                    await mover(target*Vector2(0.25,0.25),1,0.08)
                else:
                    await mover(Vector2.UP,1, 0.1)

        elif direction == "down":
            await mover(Vector2.DOWN,0.1,0.1)
        else:
            if !ray_cast_2d_up.is_colliding():
                await mover(Vector2.UP,1, 0.1)
        while !is_on_floor():
            await get_tree().create_timer(0.1).timeout

func mover(direction: Vector2, distance, time):
    raycast_update()
    var tween = get_tree().create_tween()
    var target_position = position + direction * (global.TILE_SIZE * distance)
    tween.tween_property(self, "position", target_position, time)
    await tween.finished
    position = target_position
```

5.3.4 Nivel.gd

```
extends Control
@onready var pages = $Hint/Panel/pages
@onready var prev = $Hint/Panel/prev
@onready var next = $Hint/Panel/next
@onready var character_1 = $Control/Character1
@onready var exit = $Hint/Panel/exit
@onready var hint = $Hint


var position_1 = Vector2(global.tilesize * 0, global.tilesize * 9)

var panel_up = false
var max_input = 20
var active_buttons = [0,1,2,5]
var flags_entered = 0
var characters = 1
var active_colors = ["rojo"]

var total_pages = 3
var curr_page = 1

func page_flip(number):
	match number:
		1:
			if curr_page >= total_pages:
				return
		-1:
			if curr_page <= 1:
				return
	var page_name = "page" + str(curr_page)
	var page_node = page_handler(page_name)
	page_node.visible = false
```

```
        curr_page = curr_page + number
        page_name = "page" + str(curr_page)
        page_node = page_handler(page_name)
        page_node.visible = true

func page_handler(page_name):
        for page in pages.get_children():
                if page.name == page_name:
                        return page

func _on_next_pressed():
        page_flip(1)
        validate_buttons()

func _on_prev_pressed():
        page_flip(-1)
        validate_buttons()


func validate_buttons():
        match curr_page:
                1:
                        prev.visible = false
                        next.visible = true
                        exit.visible = false
                total_pages:
                        next.visible = false
                        exit.visible = true
                        if curr_page == total_pages:
                                prev.visible = true
                _:
                        next.visible = true
                        prev.visible = true
                        exit.visible = false
```

```
func reset_hint():
	curr_page = 1
	var page_name = "page" + str(curr_page)
	for page in pages.get_children():
		if page.name == page_name:
			page.visible = true
		else:
			page.visible = false
	prev.visible = false
	next.visible = true
	exit.visible = false

func _on_exit_pressed():
	hint.visible = false

func _ready():
	global_audio.play_music_level("forest")
```

5.3.5 MenuPrincipal.gd

```
extends Control
@onready var MASTER_BUS_ID = AudioServer.get_bus_index("Master")
@onready var MUSIC_BUS_ID = AudioServer.get_bus_index("Music")
@onready var SFX_BUS_ID = AudioServer.get_bus_index("SFX")
@onready var master_slider = $UI/PanelBG/Panel/MarginContainer/GridContainer/MasterSlider
@onready var music_slider = $UI/PanelBG/Panel/MarginContainer/GridContainer/MusicSlider
@onready var sfx_slider = $UI/PanelBG/Panel/MarginContainer/GridContainer/SFXSlider
@onready var logo_anim = $logoAnim
@onready var tutorial = $LevelScrolls/VertBox/Tutorial
@onready var bucles = $LevelScrolls/VertBox/Bucles
@onready var funciones = $LevelScrolls/VertBox/Funciones
@onready var condicionales = $LevelScrolls/VertBox/Condicionales
```

```
var level_panels = []
var level_buttons = []
var level_scores = []

func get_level_buttons():
	for panel_obj in level_panels:
		for child in panel_obj.get_children():
			if str_to_var(child.name) in range(19):
				level_buttons.append(child)
func get_level_scores():
	for panel_obj in level_panels:
		for child in panel_obj.get_children():
			if child.name.contains("Level"):
				level_scores.append(child)
func _ready():
	global.load_game()
	level_panels = [tutorial, bucles, funciones, condicionales]
	get_level_buttons()
	get_level_scores()
	logo_anim.play("sway")
	AudioServer.set_bus_volume_db(MASTER_BUS_ID, global.data.volume_settings.music)
	AudioServer.set_bus_volume_db(MUSIC_BUS_ID, global.data.volume_settings.music)
	global_audio.play_music_level("main")
	connect_levels(level_buttons)
	master_slider.value = db_to_linear(global.data.volume_settings.master)
	music_slider.value = db_to_linear(global.data.volume_settings.music)
	sfx_slider.value = db_to_linear(global.data.volume_settings.sfx)
@onready var transition = $transition

func connect_levels(container):
	for level in container:
		if str_to_var(level.name) in range(global.data.unlocked_levels+1):
			level.disabled = false
			level.pressed.connect(self.change_level.bind(level.name))
```

```
                load_level_score(level.name)
            else:
                level.disabled = true
                load_level_score(level.name)

func load_level_score(level_number):
    var stars = 0
    var level_name = "Level"+str(level_number)
    for score in level_scores:
        if score.name == level_name:
            stars = global.data.level_solutions[level_name][2]
            match stars:
                1:
                    score.get_node("1star").visible = true
                    score.get_node("2star").visible = false
                    score.get_node("3star").visible = false
                    score.get_node("Panel").visible = true
                2:
                    score.get_node("1star").visible = false
                    score.get_node("2star").visible = true
                    score.get_node("3star").visible = false
                    score.get_node("Panel").visible = true
                3:
                    score.get_node("1star").visible = false
                    score.get_node("2star").visible = false
                    score.get_node("3star").visible = true
                    score.get_node("Panel").visible = true
                _:
                    score.get_node("1star").visible = false
                    score.get_node("2star").visible = false
                    score.get_node("3star").visible = false
                    score.get_node("Panel").visible = false

func change_level(level_name):
```

```
        transition.play("fade_out")
        await transition.animation_finished
        get_tree().change_scene_to_file("res://Scenes/level_" + level_name +  ".tscn")

@onready var panel_bg = $UI/PanelBG
@onready var panel = $UI/PanelBG/Panel
@onready var credits = $UI/PanelBG/Credits

func _on_settings_pressed():
        panel_bg.visible = true

func _on_panel_bg_gui_input(event):
        if event is InputEventScreenTouch and event.is_pressed():
                panel_bg.visible = false
                panel.visible = true
                credits.visible = false
func _on_credits_pressed():
        panel.visible = false
        credits.visible = true

func _on_master_slider_value_changed(value):
        AudioServer.set_bus_volume_db(MASTER_BUS_ID, global.data.volume_settings.master)
        AudioServer.set_bus_mute(MASTER_BUS_ID, value < 0.05)
        global.data.volume_settings.master = linear_to_db(value)
        global.save_game()


func _on_music_slider_value_changed(value):
        AudioServer.set_bus_volume_db(MUSIC_BUS_ID, global.data.volume_settings.music)
        AudioServer.set_bus_mute(MUSIC_BUS_ID, value < 0.05)
        global.data.volume_settings.music = linear_to_db(value)
        global.save_game()

func _on_sfx_slider_value_changed(value):
```

```
        AudioServer.set_bus_volume_db(SFX_BUS_ID, global.data.volume_settings.sfx)
        AudioServer.set_bus_mute(SFX_BUS_ID, value < 0.05)
        global.data.volume_settings.sfx = linear_to_db(value)
        global.save_game()

func _on_logo_anim_animation_finished(_anim_name):
        logo_anim.play("sway")
```

CAPÍTULO VI
FASE DE TRANSICIÓN

La fase de transición es el paso final de la metodología RUP. Su propósito es garantizar la entrega de un prototipo completamente funcional que refleje los requerimientos establecidos al inicio del proyecto.

Implementación

En esta etapa, se empleó Godot Engine para empaquetar el proyecto en un archivo APK compatible con dispositivos Android. A través del menú de exportación, se configuraron los parámetros necesarios, incluyendo una clave de firma digital en modo debug, junto con ajustes como el identificador del paquete y el número de versión, esenciales para la instalación en dispositivos reales. Posteriormente, se generó el archivo APK y se verificó su integridad mediante pruebas de instalación y ejecución en diversos dispositivos Android.

Pruebas

La instalación de la APK permitió confirmar su correcto funcionamiento en un entorno real. Se validó que el juego se ejecutara sin errores, que las interfaces fueran accesibles y que las mecánicas operaran conforme a los requerimientos del proyecto. No se identificaron problemas de rendimiento o compatibilidad que requirieran ajustes adicionales.

Esta fase finalizó con la documentación de las actividades realizadas, los resultados obtenidos y las observaciones pertinentes, dejando un registro que respalda los objetivos alcanzados y sirve como referencia para futuros desarrollos.

CONCLUSIONES

- Los objetivos planteados al inicio del proyecto se lograron exitosamente gracias a la implementación de la metodología RUP, la cual proporcionó un enfoque estructurado y consistente a lo largo de todo el desarrollo.

- Los desafíos diseñados en el videojuego facilitaron la introducción de conceptos fundamentales de programación de manera gradual y comprensible. Este enfoque incremental permitió que los usuarios asimilaran cada concepto a su propio ritmo, lo cual genera una experiencia de aprendizaje más accesible y efectiva.

- La interfaz del videojuego fue desarrollada para garantizar una navegación e interacción simples e intuitivas, que les permite a los jugadores concentrarse plenamente en los desafíos propuestos sin distracciones innecesarias.

- Finalmente, se demostró que herramientas de software libre como Godot Engine son altamente efectivas para proyectos académicos. Su carácter gratuito, combinado con su facilidad de uso y un lenguaje accesible como GDScript, la convierten en una plataforma ideal tanto para desarrolladores experimentados como para principiantes, destacando su versatilidad en iniciativas educativas.

RECOMENDACIONES

- Desarrollar niveles adicionales, mecánicas más complejas o elementos narrativos que incrementen el valor educativo del juego.

- Adaptar el videojuego a plataformas como Windows o navegadores web para aumentar su disponibilidad y alcance.

- Traducir el contenido del videojuego a diferentes idiomas podría ampliar su alcance y hacerlo accesible a una audiencia más diversa.

BIBLIOGRAFÍA

1. Bado, N. (2019). Game-based learning pedagogy: a review of the literature. *Interactive Learning Environments*, *30*(5), 936–948. https://doi.org/10.1080/10494820.2019.1683587

2. Gentry, S. V., Gauthier, A., Ehrstrom, B. L., Wortley, D., Lilienthal, A., Car, L. T., Dauwels-Okutsu, S., Nikolaou, C. K., Zary, N., Campbell, J., & Car, J. (2019). Serious Gaming and Gamification Education in Health Professions: Systematic review. *JMIR. Journal of Medical Internet Research/Journal of Medical Internet Research*, *21*(3), e12994. https://doi.org/10.2196/12994

3. Greipl, S., Moeller, K., & Ninaus, M. (2020). Potential and limits of game-based learning. *International Journal of Technology Enhanced Learning*, *12*(4), 363. https://doi.org/10.1504/ijtel.2020.110047

4. *Jacobson, I., Booch, G., &Rumbaugh, J., (2000). El proceso unificado de desarrollo de software (1era Edición).Madrid: Pearson Educación.*

5. Koster, R., & Wright, W. (2004). *A theory of fun for game design*. http://ci.nii.ac.jp/ncid/BB15618405

6. Maldonado, J. J. C., Macho, L. K. G., & Casallas, E. C. (2023). La investigación aplicada y el desarrollo experimental en el fortalecimiento de las competencias de la sociedad del siglo XXI. *Tecnura*, *27*(75), 140–174. https://doi.org/10.14483/22487638.19171

7. Serrano, K. (2019). The effect of digital game-based learning on student learning: A literature review. *Graduate Research Papers*. https://scholarworks.uni.edu/cgi/viewcontent.cgi?article=1909&context=grp

8. Tamayo, M. T. Y. (2001). *El proceso de la investigación científica*. Editorial Limusa.

9. Wang, M., & Zheng, X. (2020). Using Game-Based Learning to Support Learning Science: A Study with Middle School Students. *The*

10. *Asia-Pacific Education Researcher*, *30*(2), 167–176. https://doi.org/10.1007/s40299-020-00523-z

11. Zhao, W., & Shute, V. J. (2019). Can playing a video game foster computational thinking skills? *Computers and Education/Computers & Education*, *141*, 103633. https://doi.org/10.1016/j.compedu.2019.103633

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

TÍTULO	Desarrollo de un videojuego educativo como herramienta para la enseñanza de los conceptos básicos de la programación
SUBTÍTULO	

AUTOR(ES):

APELLIDOS Y NOMBRES	**CVLAC / E_MAIL**	
Tonito R., Alejandra C	**CVLAC:**	26.383.611
	E_MAIL	alejandratonito@gmail.com
APELLIDOS Y NOMBRES	**CVLAC / E_MAIL**	

PALÁBRAS O FRASES CLAVES:

Videojuego educativo
aprendizaje
desarrollo software
software libre
RUP
UML

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO:

ÁREA	SUBÁREA
Escuela de Ingeniería y Ciencias Aplicadas	Ingeniería en Computación

RESUMEN (ABSTRACT):

En este proyecto desarrolla un videojuego educativo para enseñar los fundamentos básicos de la programación de manera interactiva y accesible. El objetivo fue crear una herramienta que combine entretenimiento y educación, permitiendo a los usuarios aprender conceptos como estructuras de control, bucles y funciones a través de desafíos prácticos y retroalimentación inmediata. Dirigido a principiantes, el juego presenta los conceptos de manera gradual para facilitar el aprendizaje. El desarrollo se realizó utilizando el motor Godot y el lenguaje GDScript, seleccionados por su flexibilidad y facilidad de uso. Se siguió la metodología RUP (Proceso Unificado de Desarrollo de Software), que permitió una planificación estructurada y un enfoque iterativo. El videojuego incluye niveles que introducen conceptos de programación de manera progresiva, desde comandos básicos hasta estructuras más complejas.

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO

CONTRIBUIDORES

APELLIDOS Y NOMBRES	**ROL / Código CVLAC / E_MAIL**	
Pedro Dorta	**ROL**	**CA** | **AS** | **TU** | JU **X**
	CVLAC:	12.914.617
	E_MAIL	dortap22@gmail.com
Victor Mujica	**ROL**	**CA** | **AS** | **TU** | **JU X**
	CVLAC:	14.054.907
	E_MAIL	vmujica0102@gmail.com
Claudio Cortínez	**ROL**	**CA** | **AS** | **TU X** | **JU X**
	CVLAC:	12.115.334
	E_MAIL	cl_cortinez@hotmail.com

FECHA DE DISCUSIÓN Y APROBACIÓN:

2025	03	06
AÑO	**MES**	**DIA**

LENGUAJE: SPA.

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO

ARCHIVO (S):

NOMBRE DE ARCHIVO	TIPO MIME
NZZTTG_TRAC2025	Application/msword

ALCANCE

ESPACIAL: inespacial

TEMPORAL: intemporal

TÍTULO O GRADO ASOCIADO CON EL TRABAJO:

Ingeniero en Computación

NIVEL ASOCIADO CON EL TRABAJO:

Pregrado

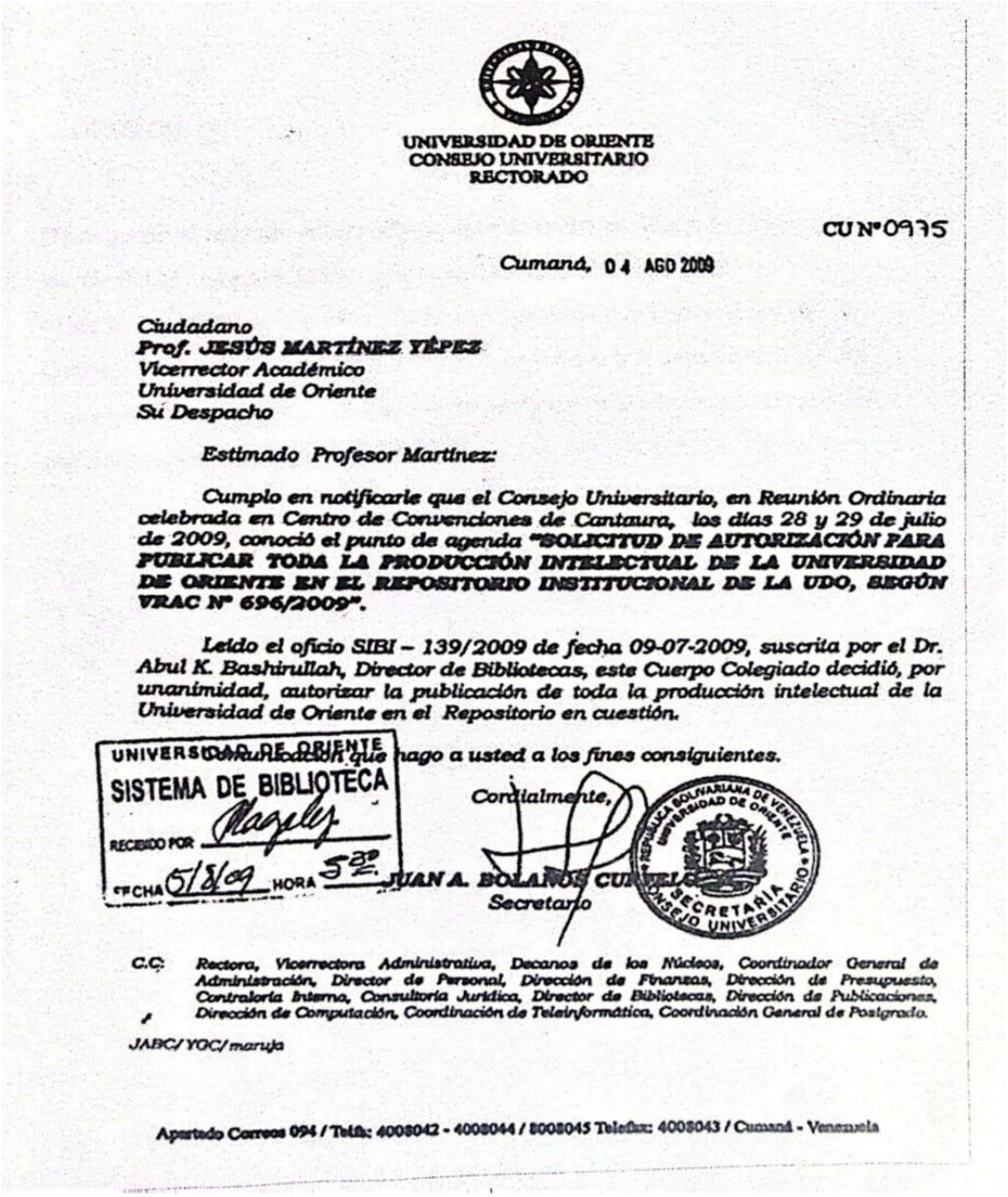
ÁREA DE ESTUDIO:

Departamento de Computación y Sistemas

INSTITUCIÓN:

Universidad de Oriente/Núcleo de Anzoátegui.

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO



UNIVERSIDAD DE ORIENTE
CONSEJO UNIVERSITARIO
RECTORADO

CU Nº 0975

Cumaná, 04 AGO 2009

Ciudadano
Prof. JESÚS MARTÍNEZ YÉPEZ
Vicerrector Académico
Universidad de Oriente
Su Despacho

Estimado Profesor Martínez:

Cumplo en notificarle que el Consejo Universitario, en Reunión Ordinaria celebrada en Centro de Convenciones de Cantaura, los días 28 y 29 de julio de 2009, conoció el punto de agenda **"SOLICITUD DE AUTORIZACIÓN PARA PUBLICAR TODA LA PRODUCCIÓN INTELECTUAL DE LA UNIVERSIDAD DE ORIENTE EN EL REPOSITORIO INSTITUCIONAL DE LA UDO, SEGÚN VRAC Nº 696/2009"**.

Leído el oficio SIBI – 139/2009 de fecha 09-07-2009, suscrita por el Dr. Abul K. Bashirullah, Director de Bibliotecas, este Cuerpo Colegiado decidió, por unanimidad, autorizar la publicación de toda la producción intelectual de la Universidad de Oriente en el Repositorio en cuestión.

Comunicación que hago a usted a los fines consiguientes.

UNIVERSIDAD DE ORIENTE
SISTEMA DE BIBLIOTECA
RECIBIDO POR
FECHA 5/8/09 HORA 5:20

Cordialmente,

JUAN A. BOLAÑOS CURIEL
Secretario

REPÚBLICA BOLIVARIANA DE VENEZUELA · UNIVERSIDAD DE ORIENTE · SECRETARÍA · CONSEJO UNIVERSITARIO

C.C: Rectora, Vicerrectora Administrativa, Decanos de los Núcleos, Coordinador General de Administración, Director de Personal, Dirección de Finanzas, Dirección de Presupuesto, Contraloría Interna, Consultoría Jurídica, Director de Bibliotecas, Dirección de Publicaciones, Dirección de Computación, Coordinación de Teleinformática, Coordinación General de Postgrado.

JABC/ YOC/ maruja

Apartado Correos 094 / Telfs: 4008042 - 4008044 / 8008045 Telefax: 4008043 / Cumaná - Venezuela

METADATOS PARA TRABAJOS DE GRADO, TESIS Y ASCENSO

Derechos

De acuerdo al artículo 41 del reglamento de trabajos de grado (Vigente a partir del II Semestre 2009, según comunicación CU-034-2009)

“Los Trabajos de grado son exclusiva propiedad de la Universidad de Oriente y solo podrán ser utilizadas a otros fines con el consentimiento del consejo de núcleo respectivo, quien lo participara al Consejo Universitario, para su autorización”

Alejandra Tonito
26.383.611
AUTOR

Claudio Cortínez	**Pedro Dorta**	**Victor Mujica**
12.115.334	12.914.617	14.054.907
TUTOR	**JURADO**	**JURADO**

Prof. Claudio Cortínez

CORDINADOR(A)

COMISION DE TRABAJO DE GRADO