UNIVERSIDAD CATÓLICA DE CUENCA

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

FACULTAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE REALIDAD VIRTUAL Y VIDEOJUEGOS

Desarrollo y evaluación de un sistema educativo interactivo en UNITY para mejorar la comprensión lógico-matemática aplicada a la programación de mecánicas de videojuego

PROYECTO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN REALIDAD VIRTUAL Y VIDEOJUEGOS

AUTORA: BRITNEY MARIUXY IÑIGUEZ NARVAEZ

DIRECTOR: ING. JHEYSON STEVEN GAONA PINEDA, MTR.

CUENCA - ECUADOR

2026

DIOS, PATRIA, CULTURA Y DESARROLLO

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

FACULTAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE REALIDAD VIRTUAL Y VIDEOJUEGOS

Desarrollo y evaluación de un sistema educativo interactivo en Unity para mejorar la comprensión lógico-matemática aplicada a la programación de mecánicas de videojuego

PROYECTO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN REALIDAD VIRTUAL Y VIDEOJUEGOS

AUTORA: BRITNEY MARIUXY IÑIGUEZ NARVAEZ

DIRECTOR: ING. JHEYSON STEVEN GAONA PINEDA, MTR.

CUENCA - ECUADOR

2026

DIOS, PATRIA, CULTURA Y DESARROLLO

Universidad Católica de Cuenca

DECLARATORIA DE AUTORÍA Y RESPONSABILIDAD

Declaratoria de Autoría y Responsabilidad

Britney Mariuxy Iñiguez Narvaez portador de la cédula de ciudadanía Nº **0107019010**. Declaro ser el autor de la obra: "**Desarrollo y evaluación de un sistema educativo interactivo en UNITY para mejorar la comprensión lógico-matemática aplicada a la programación de mecánicas de videojuego**", sobre la cual me hago responsable sobre las opiniones, versiones e ideas expresadas. Declaro que la misma ha sido elaborada respetando los derechos de propiedad intelectual de terceros y eximo a la Universidad Católica de Cuenca sobre cualquier reclamación que pudiera existir al respecto. Declaro finalmente que mi obra ha sido realizada cumpliendo con todos los requisitos legales, éticos y bioéticos de investigación, que la misma no incumple con la normativa nacional e internacional en el área específica de investigación, sobre la que también me responsabilizo y eximo a la Universidad Católica de Cuenca de toda reclamación al respecto.

Cuenca, 08/09/2026

F: .....

Britney Mariuxy Iñiguez Narvaez

C.I. **0107019010**

Universidad Católica de Cuenca

CERTIFICADO

Certificado

Certifico que la presente Propuesta Tecnológica fue desarrollado por **Britney Mariuxy Iñiguez Narvaez** portador(a) de la cedula de ciudadanía N.º **0107019010** con el tema "**Desarrollo y evaluación de un sistema educativo interactivo en UNITY para mejorar la comprensión lógico-matemática aplicada a la programación de mecánicas de videojuego**", bajo mi supervisión.

Cuenca, **08 de septiembre de 2026**

F: .....

Ing. Jheyson Steven Gaona Pineda, Mtr.

Docente - Tutor

Dedicatoria

Dedico este trabajo, con todo mi amor, a mis padres: Reina, mi hermosa y admirable madre, y Eugenio, mi padre fuerte y, al mismo tiempo, sensible. Gracias por confiar en mí, cuidarme y amarme incondicionalmente. Deseo que se sientan orgullosos de este logro y que sepan que cada uno de sus esfuerzos ha valido completamente la pena.

A mis hermanos, Ale y Vini, por sus palabras de aliento, sus cuidados y por preocuparse siempre por mí. Gracias por quererme tanto y por acompañarme en cada etapa de mi vida, así como yo siempre estaré para ustedes.

A mis mascotas, Nina, Neyca y Jui, por ser mis compañeras en los momentos buenos y también en los malos. Su lealtad, su cariño y esa forma tan especial de entenderme han sido un apoyo constante. A pesar de mis defectos, siempre me han brindado un amor puro y sincero.

A ti amor, Carlos. Por acompañarme en cada paso, apoyarme, animarme y amarme tal como soy. Gracias por caminar a mi lado y por compartir conmigo este proceso con el mismo amor que yo siento por ti.

Cada uno de ustedes me ha regalado tantos momentos que serían necesarias muchas páginas para recordarlos todos. Me siento profundamente feliz y agradecida por tenerlos en mi vida. Por ello, deseo dedicarles este logro, uno de los muchos que espero compartir con ustedes. Agradezco cada experiencia vivida, tanto las buenas como las difíciles, porque todas han contribuido a formar la persona que soy hoy.

Britney Iñiguez Narvaez

Agradecimiento

Agradezco, en primer lugar, a Dios por estar siempre conmigo, por su amor, su guía constante y por acompañarme en cada etapa de mi vida. Gracias por sostenerme, perdonarme y permitirme alcanzar tantas metas que hoy me llenan de felicidad y orgullo. Le pido que continúe acompañándome y orientando mi camino hasta el final.

A mis padres, por apoyarme en todo momento, por sus consejos y por cada esfuerzo y sacrificio que han realizado para que pudiera seguir adelante con cada meta y cada sueño. Este logro también les pertenece, porque su amor y dedicación han sido fundamentales para culminar mis estudios.

A mis hermanos, por cada granito de arena que han aportado a mi vida, por su apoyo constante y por los esfuerzos que también realizaron para ayudarme a cumplir mis objetivos. Agradezco profundamente todo lo que han hecho por mí y la forma en que siempre han estado presentes. A ti ñaña por ser mi ejemplo a seguir en muchos aspectos, gracias por todo.

A mi compañero de tesis, Beto, por el equipazo que logramos formar. Me siento muy feliz de haber desarrollado este proyecto juntos y de haber alcanzado grandes resultados a lo largo del proceso. Le deseo muchos éxitos tanto en su vida profesional como personal.

Finalmente, agradezco a mi tutor Jheyson, por acompañarnos en cada etapa, compartir sus conocimientos con generosidad y brindarnos siempre su preocupación, orientación y apoyo. Su guía fue fundamental para culminar esta etapa de la mejor manera. Le deseo igualmente muchos éxitos en cada uno de sus proyectos.

Britney Iñiguez Narvaez

Resumen

El aprendizaje de programación de mecánicas de videojuego requiere comprender la relación entre el código, los fundamentos lógico-matemáticos y el comportamiento observable dentro de un entorno interactivo. Sin embargo, en los procesos de formación puede existir una separación entre la explicación teórica y la aplicación práctica, lo que dificulta que los estudiantes interpreten, modifiquen y desarrollen mecánicas de manera autónoma. Ante esta situación, el proyecto tuvo como objetivo desarrollar un sistema educativo e interactivo que se integra a un entorno guiado en Unity y un libro didáctico complementario para apoyar la comprensión de la programación de mecánicas de videojuego.

El desarrollo se organizó mediante el modelo en cascada, considerando las fases de análisis de requisitos, diseño técnico y narrativo, implementación, pruebas y ajustes finales. La solución se estructuró en cuatro módulos progresivos: (i) movimiento y sprint, (ii) salto y dash, (iii) zonas del entorno y (iv) máquina de estados. El entorno en Unity incorporó instrucciones y validaciones paso a paso, mientras que el libro explicó la lógica de programación, los fundamentos matemáticos y el comportamiento de las mecánicas.

La validación se realizó mediante una encuesta con escala de Likert aplicada a una muestra de 20 participantes pertenecientes al público objetivo. Los resultados reflejaron una valoración favorable del sistema, especialmente en la claridad de las instrucciones, la comprensión de las mecánicas y la utilidad de las validaciones durante el proceso. Se concluyó que la integración del entorno guiado y el recurso didáctico orienta de mejor manera al usuario y reduce dificultades durante la implementación de las actividades.

Palabras Clave: *Entorno guiado, Mecánicas de videojuegos, Lógica matemática, Unity.*

Abstract

Learning to program game mechanics requires understanding the relationship between code, logical-mathematical fundamentals, and observable behavior within an interactive environment. However, in the training processes, there can be a separation between theoretical explanation and practical application, which makes it difficult for students to interpret, modify, and develop mechanics autonomously. Given this situation, the project aimed to develop an educational and interactive system that integrates into a guided environment in Unity and a complementary didactic book to support the understanding of video game mechanics programming. The development was organized using the waterfall model, considering the phases of requirements analysis, technical and narrative design, implementation, testing, and final adjustments. The solution is structured into four modules: (i) movement and sprint, (ii) jump and dash, (iii) environment zones, and (iv) state machine. The environment in Unity incorporates step-by-step instructions and validations, while the book explains the programming logic, mathematical foundations, and mechanics. The validation is carried out through a Likert scale survey applied to a sample of 20 participants belonging to the target audience. The results reflect a favorable evaluation of the system, especially in the clarity of the instructions, the understanding of the mechanics, and the usefulness of the validations during the process. It is concluded that the integration of the guided environment and the educational resource better guides the user and reduces difficulties during the implementation of the activities.

Keywords: *Guided environment, Video game mechanics, Mathematical logic, Unity.*

Índice

DECLARATORIA DE AUTORÍA II
CERTIFICADO III
Dedicatoria IV
Agradecimiento V
Resumen VI
Abstract VII
Introducción 1
Capítulo I 3
1. Marco Referencial 3
1.1. Contextualización del problema 3
1.2. Planteamiento del problema 4
1.3. Formulación del problema 6
1.4. Justificación 6
1.5. Objetivo general 8
1.6. Objetivos específicos 9
1.7. Alcance del proyecto 9
1.8. Delimitaciones 11
1.9. Beneficiarios 12
1.10. Metodología 13
1.11. Resultados esperados 15
Capítulo II 17
2. Marco Teórico 17
2.1. Antecedentes de investigación 17
2.1.1. Interactive Tutorial in Unity 17
2.1.2. Using Unity to Teach Game Programming 21
2.1.3. GameDevDojo 25
2.1.4. Cuadro comparativo de trabajos relacionados 30
2.2. Bases teóricas 32
2.2.1. Evolución del aprendizaje de programación mediante videojuegos 32
2.2.2. Estructura narrativa y didáctica del aprendizaje guiado 34
2.2.3. Estructura y secuencia didáctica del libro complementario 36
2.2.4. Aporte educativo y tecnológico 37
2.3. Bases metodológicas 39
2.3.1. Cascada 39
2.3.2. Scrum 42
2.3.3. Extreme Programming (XP) 44
2.3.4. Cuadro comparativo de metodologías de desarrollo 48

2.4. Bases tecnológicas 50
2.4.1. Motores de desarrollo de videojuegos 50
2.4.2. Lenguajes de programación 62
2.4.3. Plataformas de repositorios para el control de versiones 72
2.5. Arquitecturas de software 84
2.5.1. Modelo–Vista–Controlador 85
2.5.2. Modelo–Vista–Controlador 87
2.5.3. Comparación de arquitecturas de software 89
2.6. Herramientas de desarrollo 90
2.7. Marco conceptual 92
Capítulo III 94
3. Manual De Usuario 94
3.1. Presentación y perfil del usuario 94
3.2. Acceso e inicio 95
3.3. Navegación y controles 98
3.4. Funciones e interacciones disponibles 101
3.4.1. Validación y progreso 102
3.4.2. Solución de problemas frecuentes 104
3.5. Evidencia de uso 104
Capítulo IV 107
4. Manual Del Programador 107
4.1. Arquitectura general del sistema 107
4.2. Entorno tecnológico y organización técnica 108
4.2.1. Tecnologías y herramientas utilizadas 108
4.2.2. Estructura de carpetas y recursos 109
4.2.3. Organización de la escena y elementos reutilizables 110
4.2.4. Organización del código fuente 111
4.3. Arquitectura de clases y componentes 112
4.3.1. Relación entre clases y componentes 112
4.4. Proceso de desarrollo de los módulos educativos 113
4.4.1. Desarrollo técnico de las mecánicas 114
4.4.2. Adaptación al entorno guiado y validación 115
4.4.3. Elaboración del libro didáctico 115
4.5. Programación de las mecánicas 115
4.5.1. Módulo 1: movimiento y sprint 115
4.5.2. Módulo 2: salto y dash 118
4.5.3. Módulo 3: zonas del entorno 120
4.5.4. Módulo 4: máquina de estados 122
4.6. Sistema de tutoriales, validación y progreso 124

4.6.1. Organización de las páginas y orientación contextual 124
4.6.2. Validación de las actividades 125
4.6.3. Seguimiento durante Play Mode 126
4.6.4. Progreso, desbloqueo y reinicio 127
4.7. Integración del libro didáctico con los módulos 128
4.7.1. Flujo integrado de aprendizaje 129
4.8. Pruebas técnicas y verificación del funcionamiento 130
4.8.1. Pruebas funcionales de las mecánicas 130
4.8.2. Verificación del entorno guiado 130
4.8.3. Estabilidad técnica y compilación 131
4.9. Conclusiones y recomendaciones 132
4.9.1. Conclusiones 132
4.9.2. Recomendaciones 132

Índice de tablas

Tabla 1 Valoración de la experiencia con GameDevDojo en una escala de 1 a 5. 29
Tabla 2 Comparación de los trabajos relacionados. 30
Tabla 3 Responsabilidades del Equipo Scrum. 43
Tabla 4 Eventos del marco de trabajo Scrum. 43
Tabla 5 Artefactos y compromisos de Scrum. 43
Tabla 6 Prácticas principales de Extreme Programming. 46
Tabla 7 Comparación de enfoques para el desarrollo de software. 48
Tabla 8 Comparación de motores de desarrollo de videojuegos. 60
Tabla 9 Comparación de lenguajes de programación utilizados en motores de videojuegos. 71
Tabla 10 Comparación de plataformas de repositorios para el control de versiones. 83
Tabla 11 Comparación entre MVC y MVP 89
Tabla 12 Conceptos técnicos aplicados al proyecto. 92
Tabla 13 Recursos y requisitos para descargar e instalar el proyecto. 95
Tabla 14 Ventas de Unity con su uso en el recorrido. 98
Tabla 15 Cómo se observan los módulos y su comprobación. 101
Tabla 16 Posibles problemas y sus acciones a tomar. 104
Tabla 17 Tecnologías principales del proyecto. 109
Tabla 18 Estructura principal de carpetas. 109
Tabla 19 Responsabilidad de las clases principales. 112
Tabla 20 Complementación entre el libro y el Unity guiado. 128

Índice de ilustraciones

Ilustración 1 Fases del modelo en cascada aplicado al desarrollo del proyecto. 14
Ilustración 2 Arquitectura básica de los subsistemas de un videojuego. 18
Ilustración 3 Flujo básico de datos dentro de un sistema de videojuego. 19
Ilustración 4 Interfaz de selección de niveles del tutorial interactivo en Unity. 20
Ilustración 5 Vista general del escenario utilizado en los tutoriales de programación en Unity. 22
Ilustración 6 Estructura general de los documentos tutoriales. 22
Ilustración 7 Secuencia sugerida para el desarrollo de los tutoriales. 24
Ilustración 8 Diseño de un nivel de GameDevDojo con escenario, objetivo, vista de componentes y controles. 27
Ilustración 9 Vista de ayuda para identificar componentes faltantes o configurados incorrectamente. 28
Ilustración 10 Fases del modelo de desarrollo en cascada. 41
Ilustración 11 *Ciclos de planificación y retroalimentación en Extreme Programming.* 46
Ilustración 12 Ventanas principales del Editor de Unity. 51
Ilustración 13 Áreas principales del Editor de Godot Engine. 54
Ilustración 14 Áreas principales del Editor de Unreal Engine. 57
Ilustración 15 Programación visual mediante Blueprints en Unreal Engine. 59
Ilustración 16 Estructura de una clase programada en C++ para Unreal Engine. 64
Ilustración 17 Creación de un script de movimiento mediante C# en Unity. 66
Ilustración 18 Configuración de las variables de un script C# desde el Inspector de Unity. 67
Ilustración 19 Ejemplo de programación mediante GDScript en Godot Engine. 69
Ilustración 20 Creación y selección de ramas dentro de un repositorio de GitHub. 73
Ilustración 21 Ejecución de un flujo de trabajo automatizado mediante GitHub Actions. 75
Ilustración 22 *Revisión y registro de cambios mediante GitHub Desktop.* 76
Ilustración 23 Seguimiento de solicitudes de integración y revisiones en GitLab. 77
Ilustración 24 *Representación de trabajos y etapas en una canalización de GitLab CI/CD.* 78
Ilustración 25 Seguimiento de una pull request asignada para revisión en Bitbucket. 81
Ilustración 26 Seguimiento de las etapas y resultados de una canalización en Bitbucket Pipelines. 82
Ilustración 27 Estructura general del patrón Modelo–Vista–Controlador. 86
Ilustración 28 Estructura general del patrón Modelo–Vista–Presentador. 88
Ilustración 29 Procedimiento de inicio del proyecto Lógica en Juego en Unity. 96
Ilustración 30 Pantalla inicial del entorno guiado. 97
Ilustración 31 Acceso al primer módulo del recorrido. 97
Ilustración 32 Elemento solicitado y resaltado dentro del Editor. 99
Ilustración 33 Configuración de la actividad en el Inspector. 100
Ilustración 34 Controles utilizados durante las pruebas del proyecto. 100
Ilustración 35 Desarrollo general de la actividad. 102
Ilustración 36 Actividad ejecutada en Play Mode. 103
Ilustración 37 Validación correcta de la actividad. 103
Ilustración 38 Registro del progreso y continuidad del recorrido. 105
Ilustración 39 Arquitectura general de Lógica en Juego. 108
Ilustración 40 Organización de carpetas y recursos del proyecto. 110
Ilustración 41 Organización de Niveles.unity y sus elementos principales. 111
Ilustración 42 Relación entre las clases de ejecución y las herramientas educativas. 113
Ilustración 43 Proceso de desarrollo de los módulos mediante la metodología en cascada. 114
Ilustración 44 Relación entre la configuración del objeto Player y las mecánicas de movimiento y sprint.. 116
Ilustración 45 Flujo general de las mecánicas de movimiento y sprint. 118
Ilustración 46 Relación entre la configuración del objeto Player y las mecánicas de salto y dash. 119
Ilustración 47 Flujo general de las mecánicas de salto y dash. 120
Ilustración 48 Relación entre la configuración de ZoneEffectArea y la interacción del jugador con el entorno. 121
Ilustración 49 Flujo general de interacción con las zonas del entorno. 122
Ilustración 50 Relación entre PlayerMovement, PlayerStateMachine y el estado del jugador. 123
Ilustración 51 Estados y transiciones principales del jugador. 124
Ilustración 52 Organización de las páginas y orientación contextual del entorno guiado. 125

Ilustración 53 Relación entre la actividad guiada, la configuración y su validación. 126
Ilustración 54 Seguimiento y validación de una mecánica durante Play Mode. 127
Ilustración 55 Registro del progreso, desbloqueo de módulos y reinicio del entorno guiado. 128
Ilustración 56 Integración entre el contenido del libro, la actividad guiada y el resultado en Unity. 129
Ilustración 57 Verificación conjunta de las mecánicas y del entorno guiado. 131
Ilustración 58 Pregunta y resultado 1..... 137
Ilustración 59 Pregunta y resultado 2..... 138
Ilustración 60 Pregunta y resultado 3..... 138
Ilustración 61 Pregunta y resultado 4..... 138
Ilustración 62 Pregunta y resultado 5..... 139
Ilustración 63 Pregunta y resultado 6..... 139
Ilustración 64 Pregunta y resultado 7..... 139
Ilustración 65 Pregunta y resultado 8..... 140
Ilustración 66 Pregunta y resultado 9..... 140
Ilustración 67 Pregunta y resultado 10..... 140

Introducción

El desarrollo de videojuegos reúne conocimientos de programación, matemáticas y diseño de sistemas interactivos. Dentro de este campo, la programación de mecánicas exige comprender cómo las variables, las condiciones y las operaciones aplicadas en el código producen respuestas concretas en el entorno de juego. Sin embargo, durante el proceso de aprendizaje puede ocurrir que el estudiante logre ejecutar una funcionalidad sin comprender con claridad la lógica que la sostiene, lo que dificulta su capacidad para modificarla, adaptarla o construir una solución propia.

Esta dificultad adquiere importancia en entornos como Unity, donde el resultado de una instrucción puede observarse de forma inmediata. Aunque esta característica facilita la práctica y la experimentación, también puede llevar a que la atención se concentre únicamente en el comportamiento visual de la mecánica. Por esta razón, el aprendizaje requiere relacionar la ejecución del código con los fundamentos lógicos y matemáticos que intervienen en su funcionamiento, de manera que el estudiante pueda interpretar el sistema y no limitarse a repetir una serie de pasos. Esta necesidad se vincula con el desarrollo del pensamiento computacional, la resolución de problemas y la organización de soluciones dentro de la programación de videojuegos.

A partir de este contexto, el presente trabajo aborda el desarrollo de un sistema educativo interactivo orientado a la programación de mecánicas de videojuego. La propuesta integra un entorno guiado en Unity y un libro didáctico complementario. Ambos recursos fueron planteados para trabajar de forma conjunta: el entorno permite realizar actividades prácticas y comprobar cada paso, mientras que el libro explica la lógica de programación, los

fundamentos matemáticos y la relación entre el código y el comportamiento observable del sistema.

La estructura del proyecto se organiza mediante módulos progresivos relacionados con el movimiento del jugador, el salto, el dash, las zonas del entorno y la máquina de estados. Esta organización permite avanzar desde mecánicas básicas hasta sistemas que requieren una mayor relación entre condiciones y comportamientos. Además, el contenido didáctico mantiene una secuencia narrativa que acompaña al usuario durante el aprendizaje, presenta las instrucciones de manera ordenada y conecta cada explicación con su aplicación dentro de Unity.

El trabajo se desarrolla en el área tecnológica de programación e ingeniería de videojuegos. Su importancia radica en la necesidad de contar con recursos que no se limiten a mostrar cómo implementar una mecánica, sino que también ayuden a comprender por qué funciona de una determinada manera. En este sentido, la propuesta busca reducir la separación entre teoría y práctica, facilitar la interpretación de los procesos internos y ofrecer una experiencia de aprendizaje más clara para estudiantes en formación.

Capítulo I

1. Marco Referencial

Este capítulo reúne los elementos que permiten delimitar y orientar el proyecto. Primero se presenta la problemática relacionada con la comprensión de los fundamentos lógicos y matemáticos aplicados a la programación de mecánicas de videojuego. Luego se explica la justificación, se establecen los objetivos y el alcance, además, se describe la metodología utilizada durante el desarrollo y la evaluación del sistema educativo interactivo.

1.1. Contextualización del problema

La programación de mecánicas de videojuego requiere comprender la relación entre las instrucciones escritas en el código y el comportamiento que se produce dentro de un entorno interactivo. En motores como Unity[1], una mecánica depende de elementos como entradas del usuario, variables, condiciones, operaciones matemáticas, eventos y cambios de estado[1], [2]. Por esta razón, el aprendizaje no debería limitarse a conseguir que una función se ejecute correctamente, sino que también debería permitir interpretar por qué ocurre determinado comportamiento y cómo puede modificarse de manera controlada[3].

En los procesos de formación en programación, algunos estudiantes consiguen implementar soluciones básicas mediante la repetición de instrucciones, ejemplos o fragmentos de código previamente elaborados. Sin embargo, esta práctica no siempre garantiza una comprensión clara de los principios que sostienen el funcionamiento del sistema[3]. Kazimoglu señala que el uso de entornos interactivos puede contribuir al aprendizaje de programación y al desarrollo del pensamiento computacional[2] cuando los conceptos se relacionan con las actividades realizadas dentro de la experiencia[4]. De manera

[1] Unity: motor utilizado para desarrollar videojuegos y aplicaciones interactivas.
[2] Pensamiento computacional: forma de resolver problemas mediante pasos lógicos y ordenados.

similar, otros estudios relacionan la comprensión de la programación con la capacidad de descomponer problemas, establecer relaciones lógicas y construir soluciones que puedan adaptarse a situaciones diferentes[5], [6].

Esta dificultad se vuelve más evidente en el desarrollo de videojuegos, debido a que cada mecánica integra varios procesos que deben funcionar de manera coordinada. Por ejemplo, el movimiento de un personaje no depende únicamente de presionar una tecla, sino de interpretar una entrada, definir una dirección, aplicar una velocidad y actualizar su posición dentro del espacio[1]. Mecánicas como el salto, el dash[3] o la interacción con zonas también requieren comprender conceptos como impulso, duración, tiempo de reutilización, detección de condiciones y modificación temporal de propiedades.

Unity facilita la experimentación porque permite observar de forma inmediata los cambios producidos por el código dentro de una escena. Kuznetsov destaca su utilidad como herramienta para la enseñanza del desarrollo de videojuegos[7], mientras que Moiseienko señala que la interacción directa con el motor favorece la comprensión práctica del proceso de implementación[8].

1.2. Planteamiento del problema

Cuando estos elementos se aprenden como pasos aislados, puede resultar difícil identificar la causa de un error, modificar el comportamiento o integrar la mecánica con otros sistemas[2]. No obstante, la visualización inmediata del resultado no garantiza por sí sola que el estudiante comprenda la estructura interna de la mecánica. Cuando la atención se concentra únicamente en obtener un comportamiento funcional, pueden pasar desapercibidas las relaciones entre variables, métodos, condiciones y componentes del sistema[9].

[3] *Dash:* desplazamiento rápido y breve del personaje.

Una de las causas de esta situación es la separación que puede existir entre la explicación teórica y la práctica dentro del motor. En algunos recursos educativos se presenta el código como una solución terminada y se explica qué debe escribirse, pero no siempre se desarrolla con suficiente claridad el razonamiento que conecta cada instrucción con el resultado observable[10]. Mathew sostiene que el aprendizaje de programación requiere actividades que fortalezcan la resolución de problemas y no solamente la ejecución de procedimientos[3]. Por ello, la ausencia de explicaciones progresivas puede provocar que el estudiante complete una actividad sin comprender completamente la lógica que utilizó.

Otra dificultad se relaciona con la organización del código. La programación de mecánicas exige distribuir responsabilidades entre diferentes componentes, controlar dependencias y evitar que varias acciones entren en conflicto. Esto se vuelve especialmente importante al integrar comportamientos como movimiento, salto, dash, zonas y estados dentro de un mismo sistema. Nystrom explica que el uso de patrones y estructuras organizadas permite reducir la complejidad y facilita el mantenimiento del código. Sin una base conceptual clara, el estudiante puede construir una mecánica funcional, pero presentar dificultades al momento de ampliarla, reutilizarla o relacionarla con otros sistemas[2].

Las consecuencias de este problema se reflejan en la autonomía del estudiante. La reproducción de una solución puede permitir completar una tarea puntual, pero resulta insuficiente cuando se requiere interpretar un error, adaptar una mecánica o construir una variante propia. También puede afectar la capacidad para relacionar los fundamentos matemáticos con el comportamiento del personaje y para comprender cómo una condición o una variable modifica la respuesta del sistema. Esta limitación dificulta el desarrollo de soluciones organizadas y reduce la posibilidad de aplicar lo aprendido en otros proyectos o situaciones[3], [10].

El problema se delimita en el aprendizaje de programación de mecánicas de videojuego mediante Unity, específicamente en estudiantes o desarrolladores en formación que necesitan comprender la relación entre el código, los fundamentos lógico-matemáticos y el comportamiento observable de un sistema interactivo. Dentro de este contexto, se identifica la necesidad de contar con una propuesta educativa que vincule la explicación conceptual con la implementación práctica, organice el aprendizaje de manera progresiva y permita comprobar cada parte del proceso antes de avanzar hacia mecánicas de mayor complejidad.

1.3. Formulación del problema

¿De qué manera un sistema educativo interactivo compuesto por un entorno guiado en Unity y un libro didáctico complementario puede apoyar el aprendizaje de la programación de mecánicas de videojuego en estudiantes o desarrolladores en formación?

1.4. Justificación

El desarrollo de este proyecto responde a la necesidad de contar con recursos educativos que permitan relacionar la programación de mecánicas de videojuego con los fundamentos lógicos y matemáticos que intervienen en su funcionamiento. Aunque un estudiante puede conseguir que una mecánica funcione siguiendo instrucciones o utilizando un código previamente elaborado, comprender las decisiones que producen ese comportamiento resulta necesario para modificarla, corregirla e integrarla con otros sistemas[2], [10].

Desde el ámbito educativo, la propuesta adquiere importancia porque articula la explicación conceptual con la práctica dentro de Unity. El aprendizaje de programación requiere desarrollar la capacidad de analizar un problema, dividirlo en partes y construir

soluciones mediante procesos organizados[5]. En este sentido, la interacción con un entorno funcional puede facilitar la observación de los resultados, pero debe estar acompañada por explicaciones que permitan interpretar la relación entre las variables, las condiciones, los métodos y el comportamiento final de la mecánica[7].

El proyecto también presenta un aporte académico al abordar la programación de videojuegos como un proceso de comprensión y no únicamente de ejecución. La organización progresiva de los contenidos permite avanzar desde conceptos básicos, como la dirección y la velocidad, hasta estructuras que requieren coordinar diferentes comportamientos, como las zonas y la máquina de estados[4]. Esta secuencia busca que los conocimientos adquiridos en un módulo sirvan como base para las actividades posteriores, favoreciendo una comprensión acumulativa del sistema[11].

En el ámbito tecnológico, el uso de Unity permite implementar y visualizar las mecánicas dentro de una escena controlada. Kuznetsov y Moiseienko destacan la utilidad de este motor en procesos de formación relacionados con el desarrollo de videojuegos, debido a que facilita la experimentación directa con componentes y comportamientos[7], [8]. Sin embargo, para aprovechar esta característica resulta necesario organizar el código de manera clara y establecer relaciones comprensibles entre sus elementos. La aplicación de principios de modularidad y separación de responsabilidades contribuye a reducir la complejidad, facilita el mantenimiento y permite integrar nuevas funciones de forma ordenada[2].

El entorno guiado incorpora instrucciones y validaciones que permiten comprobar cada actividad antes de continuar. Esta característica brinda al usuario una orientación constante durante la implementación y ayuda a detectar errores en el momento en que se

[4] Máquina de estados: sistema que organiza comportamientos en estados y controla sus cambios.

producen[12]. De manera complementaria, el libro didáctico explica los fundamentos de cada mecánica, la función de sus componentes y el efecto que generan dentro de Unity. La integración de ambos recursos permite que la práctica no se limite a copiar instrucciones, sino que se acompañe de una explicación sobre lo que se está desarrollando y la razón de cada decisión[10].

El componente narrativo y didáctico también resulta relevante, ya que organiza el contenido mediante una secuencia clara y progresiva[11]. Los cuatro capítulos del libro mantienen correspondencia con los cuatro módulos del entorno, lo que facilita la continuidad entre la explicación y la actividad práctica. Además, el uso de ejemplos, fragmentos de código, diagramas y explicaciones permite presentar los procesos de una forma más comprensible para estudiantes en formación[13].

Por estas razones, el proyecto se justifica como una propuesta educativa y tecnológica que integra programación, fundamentos lógico-matemáticos y narrativa didáctica. Su importancia radica en ofrecer una guía estructurada para comprender cómo el código produce determinados comportamientos dentro de Unity. De esta manera, se busca apoyar al usuario durante la implementación de mecánicas y favorecer el desarrollo de una mayor autonomía al analizar, modificar y construir soluciones propias.

1.5. Objetivo general

Desarrollar y evaluar un sistema educativo interactivo compuesto por un entorno guiado en Unity y un libro didáctico complementario, mediante la integración progresiva de fundamentos lógicos y matemáticos aplicados a la programación de mecánicas de videojuego, con el propósito de favorecer la comprensión de la relación entre el código y el comportamiento del sistema en estudiantes en formación.

1.6. Objetivos específicos

- Diseñar la estructura técnica y narrativa del sistema educativo interactivo, articulando el entorno guiado en Unity con el contenido del libro didáctico para relacionar la explicación conceptual con la aplicación práctica.
- Implementar un entorno guiado en Unity organizado en módulos progresivos, que incorpore mecánicas de movimiento, sprint[5], salto, dash, zonas del entorno y máquina de estados, junto con instrucciones y validaciones paso a paso.
- Elaborar e integrar un libro didáctico que explique la lógica de programación, los fundamentos matemáticos, la estructura de los scripts[6] y su relación con el comportamiento observable de las mecánicas desarrolladas.
- Evaluar la percepción del público objetivo sobre la claridad, utilidad y facilidad de uso del sistema educativo interactivo, mediante una encuesta con escala de Likert[7] aplicada después de la interacción con el entorno guiado y el libro didáctico.

1.7. Alcance del proyecto

El presente proyecto comprende el desarrollo de un sistema educativo interactivo orientado al aprendizaje de la programación de mecánicas de videojuego en Unity. La propuesta está compuesta por un entorno guiado y un libro didáctico complementario, ambos organizados en cuatro módulos progresivos que relacionan los fundamentos lógicos y matemáticos con la implementación práctica de mecánicas dentro de una escena controlada.

En el área de programación, el entorno guiado incluye cuatro módulos funcionales. El primer módulo aborda la configuración inicial del proyecto, el movimiento base del

[5] *Sprint:* aumento temporal de la velocidad del personaje.
[6] Script: archivo de código que controla funciones o comportamientos.
[7] Escala de Likert. escala utilizada para medir niveles de opinión o valoración.

jugador y el sprint, mediante el uso de entradas, vectores de dirección, velocidad y normalización.[8] El segundo módulo incorpora el salto y el dash, considerando aspectos como impulso, dirección, duración y tiempo de reutilización. El tercer módulo desarrolla un sistema de zonas del entorno capaces de modificar temporalmente el comportamiento del jugador, mediante condiciones de entrada, permanencia y salida. El cuarto módulo integra las mecánicas anteriores a través de una máquina de estados, encargada de organizar los comportamientos y controlar las transiciones entre ellos.

El entorno también incluye un sistema de instrucciones y validaciones paso a paso. Cada actividad debe completarse correctamente antes de continuar, lo que permite guiar al usuario durante la implementación y reducir errores dentro del proceso. Además, los módulos se encuentran integrados en una misma experiencia, de manera que los conocimientos adquiridos en las primeras actividades se utilizan posteriormente en sistemas de mayor complejidad.

En el componente narrativo y didáctico, el proyecto contempla la elaboración de un libro estructurado en cuatro capítulos, cada uno correspondiente a uno de los módulos implementados en Unity. El primer capítulo explica el movimiento y el sprint; el segundo desarrolla el salto y el dash; el tercero aborda las zonas del entorno; y el cuarto presenta la máquina de estados y la integración de las mecánicas. Cada capítulo incluye explicaciones conceptuales, fundamentos matemáticos, interpretación de variables, condiciones y métodos, fragmentos de código comentados, diagramas de flujo y actividades relacionadas con el entorno guiado.

[8] Normalización: ajuste de un vector para conservar su dirección con magnitud igual a uno.

La narrativa del libro sigue una secuencia progresiva que acompaña al usuario desde la configuración inicial hasta la integración de los sistemas. Su función consiste en explicar qué se realiza, por qué se utiliza cada elemento y cómo el código modifica el comportamiento observable dentro de Unity. De este modo, el contenido no se limita a mostrar instrucciones, sino que busca facilitar la comprensión de la lógica interna de las mecánicas.

El alcance también incluye la validación del sistema mediante una encuesta con escala de Likert aplicada a una muestra de 20 participantes pertenecientes al público objetivo. La evaluación consideró la claridad de las instrucciones, la utilidad del contenido didáctico, la facilidad de uso del entorno y el apoyo proporcionado por las validaciones durante el desarrollo de las actividades.

1.8. Delimitaciones

El proyecto se delimita funcionalmente al desarrollo de un sistema educativo interactivo compuesto por un entorno guiado en Unity y un libro didáctico complementario. El contenido se restringe a cuatro módulos: movimiento y sprint, salto y dash, zonas del entorno y máquina de estados. De igual manera, el libro se organiza en cuatro capítulos correspondientes a dichos módulos. Por tanto, no se incorporan mecánicas ni contenidos de programación adicionales a los establecidos dentro de esta estructura.

En el ámbito tecnológico, el entorno guiado fue desarrollado en Unity mediante programación en C#. La propuesta se limita a una experiencia educativa controlada, orientada a la implementación y validación progresiva de mecánicas. No contempla el desarrollo de un videojuego comercial completo, un mundo abierto, exploración libre, sistemas multijugador, inteligencia artificial avanzada, generación procedural ni la publicación del producto en plataformas digitales. Tampoco incluye como parte del aporte técnico y narrativo la creación

visual de personajes, escenarios, modelos, texturas o ilustraciones. Estas exclusiones ya se encuentran establecidas en el alcance final del proyecto.

En el ámbito metodológico, el desarrollo se organizó mediante el modelo en cascada, considerando las fases de análisis, diseño técnico y narrativo, implementación, pruebas, validación y ajustes finales. La comprobación técnica se concentró en el funcionamiento de las mecánicas, la secuencia de los módulos, las instrucciones y las validaciones requeridas para avanzar dentro del entorno. La evaluación con usuarios se limitó a una encuesta con escala de Likert aplicada a 20 participantes pertenecientes al público objetivo, mediante la cual se valoraron la claridad, utilidad, facilidad de uso y apoyo proporcionado por el sistema.

Debido al tipo de instrumento aplicado, los resultados permiten conocer la percepción de los participantes sobre la experiencia, pero no demostrar una mejora objetiva del aprendizaje. No se realizó una comparación previa y posterior, no se utilizó un grupo de control y tampoco se efectuó un seguimiento prolongado para determinar la retención o transferencia de los conocimientos adquiridos. Por esta razón, las conclusiones deben limitarse a la valoración del sistema y al apoyo percibido durante su uso, sin atribuirle efectos educativos generales o permanentes.

1.9. Beneficiarios

Los beneficiarios directos del proyecto son los estudiantes y desarrolladores en formación interesados en aprender la programación de mecánicas de videojuego mediante Unity. La propuesta se dirige especialmente a usuarios que necesitan relacionar las instrucciones escritas en el código con los fundamentos lógico-matemáticos y el comportamiento observable dentro de una escena. Estos usuarios disponen de un entorno guiado para implementar y comprobar las mecánicas de manera progresiva, acompañado por

un libro didáctico que explica la función de las variables, condiciones, métodos y operaciones utilizadas.

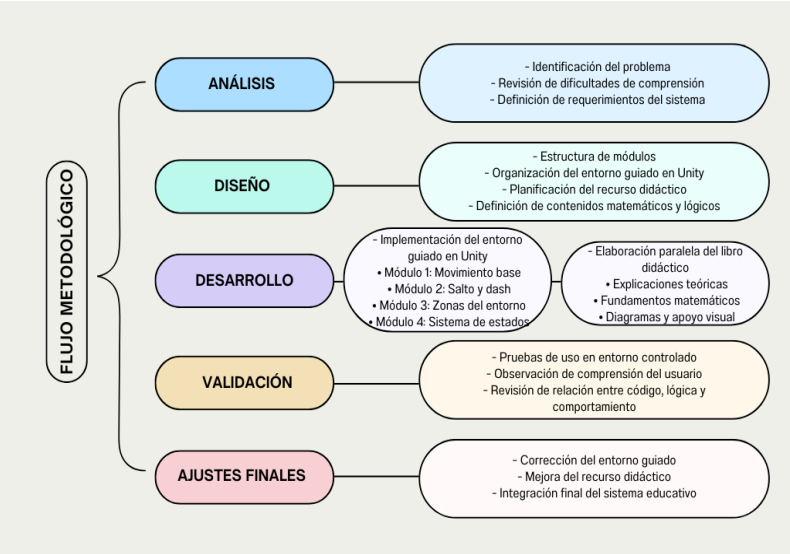
De manera indirecta, pueden beneficiarse docentes, tutores y responsables de asignaturas relacionadas con programación y desarrollo de videojuegos, quienes podrían utilizar el entorno y el libro como recursos complementarios para organizar actividades prácticas. También pueden beneficiarse programas académicos vinculados con el desarrollo de software interactivo, al disponer de una propuesta estructurada que relaciona la explicación conceptual con la implementación dentro de Unity.

1.10. Metodología

El desarrollo del proyecto se organizó mediante el modelo en cascada, debido a que permitió avanzar de forma ordenada a través de etapas previamente definidas. Esta metodología facilitó la planificación del sistema educativo interactivo, la organización de los contenidos del libro y la implementación de los módulos en Unity, manteniendo una secuencia clara desde la identificación de los requerimientos hasta la validación del producto final.

Ilustración 1

Fases del modelo en cascada aplicado al desarrollo del proyecto.



Nota. Elaboración propia.

En la fase de análisis se revisó la problemática relacionada con la comprensión de la programación de mecánicas y se definieron las necesidades del público objetivo. También se establecieron los cuatro módulos del sistema, los fundamentos lógicos y matemáticos que debían abordarse y la relación entre el entorno guiado y el libro didáctico.

Durante la fase de diseño se organizó la estructura técnica y narrativa del proyecto. En el entorno de Unity se definieron el flujo de actividades, las instrucciones, las validaciones y la integración de las mecánicas. De forma paralela, se estableció la organización de los cuatro capítulos del libro, la secuencia de las explicaciones y la forma en que cada contenido se relacionaría con las actividades prácticas.

La fase de implementación comprendió el desarrollo progresivo de los módulos de movimiento y sprint, salto y dash, zonas del entorno y máquina de estados. Además, se elaboraron los contenidos del libro, incluyendo explicaciones sobre lógica de programación,

fundamentos matemáticos, interpretación de scripts, diagramas y actividades vinculadas con el funcionamiento del entorno guiado.

En la fase de pruebas, se verificó el funcionamiento de las mecánicas, la secuencia de los módulos y las validaciones paso a paso. También se revisó la integración entre el contenido didáctico y las actividades desarrolladas en Unity. La validación con el público objetivo se realizó mediante una encuesta con escala de Likert aplicada a una muestra de 20 participantes, con el propósito de conocer su percepción sobre la claridad, utilidad y facilidad de uso del sistema.

Finalmente, la fase de ajustes permitió corregir errores, mejorar el funcionamiento de las validaciones y realizar cambios en las explicaciones del libro. Este proceso permitió consolidar una versión funcional en la que la programación y la narrativa didáctica se integran para acompañar al usuario durante el aprendizaje de las mecánicas.

1.11. Resultados esperados

Al concluir el proyecto, se espera obtener un sistema educativo interactivo funcional compuesto por un entorno guiado en Unity y un libro didáctico complementario. El entorno deberá integrar cuatro módulos progresivos correspondientes al movimiento y sprint, salto y dash, zonas del entorno y máquina de estados. Cada módulo incorporará instrucciones, actividades y validaciones que permitan comprobar la configuración realizada antes de continuar con el proceso.

Como segundo producto, se espera disponer de un libro didáctico organizado en cuatro capítulos relacionados directamente con los módulos del entorno. Este recurso incluirá explicaciones conceptuales, fundamentos matemáticos, interpretación de variables,

condiciones y métodos, fragmentos de código comentados, diagramas de flujo y actividades vinculadas con la implementación práctica de las mecánicas.

También se espera consolidar la correspondencia entre ambos componentes, de manera que cada capítulo del libro sirva como apoyo conceptual para las actividades realizadas en Unity. Esta integración deberá permitir que el usuario consulte la explicación de una mecánica, la implemente dentro del entorno, compruebe su funcionamiento y continúe hacia contenidos de mayor complejidad. El resultado no será un videojuego orientado al entretenimiento, sino una experiencia educativa controlada centrada en la programación de mecánicas.

Finalmente, se espera obtener información sobre la percepción del público objetivo mediante una encuesta con escala de Likert aplicada a 20 participantes. Esta evaluación permitirá conocer la valoración de los usuarios respecto a la claridad de las instrucciones, la utilidad del contenido didáctico, la facilidad de uso y el apoyo ofrecido por las validaciones. Los resultados servirán como evidencia de aceptación y como referencia para identificar ajustes, sin atribuir al sistema una mejora objetiva del aprendizaje que no haya sido comprobada mediante un diseño experimental.

Capítulo II

2. Marco Teórico

Este capítulo reúne las bases teóricas y técnicas que sustentan el sistema educativo interactivo. Primero se revisan los antecedentes relacionados con la enseñanza de programación mediante videojuegos y entornos interactivos. Luego se analizan las características técnicas y didácticas de estos sistemas, la organización del contenido narrativo y los trabajos relacionados que sirven como referencia. También se investigan y comparan las tecnologías, herramientas y metodologías disponibles, con el propósito de determinar cuáles se adaptan mejor a las necesidades del proyecto.

2.1. Antecedentes de investigación

En este apartado se revisan tres trabajos relacionados con la enseñanza de programación y el desarrollo de videojuegos mediante entornos interactivos. La selección considera propuestas que incorporan actividades guiadas, recursos didácticos, programación de mecánicas o sistemas de validación. De cada trabajo se analizan su propósito, funcionamiento, aportes técnicos y forma de evaluación, para obtener referentes y fortalecer las bases de la investigación. Al final se presenta un cuadro comparativo que permite reconocer sus principales similitudes, diferencias y oportunidades de mejora.

2.1.1. Interactive Tutorial in Unity

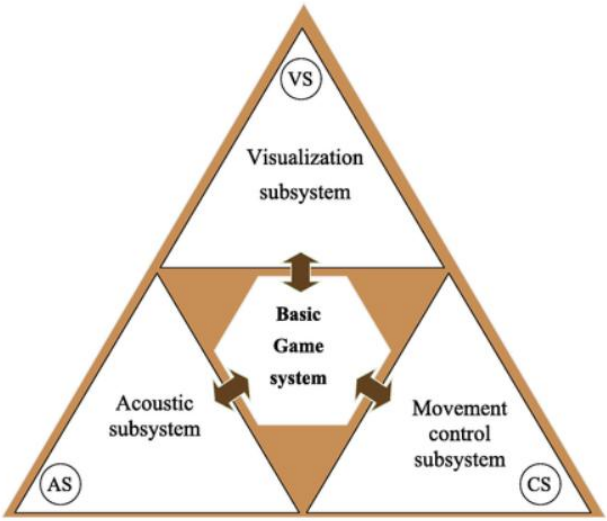
Sobota y Pietriková estudiaron el uso de motores de videojuegos como herramientas para la enseñanza de desarrollo de juegos y ciencias de la computación. Su trabajo revisó diferentes motores y presentó experiencias prácticas utilizadas en cursos universitarios. Entre ellas se encuentra un tutorial interactivo desarrollado en Unity, creado para familiarizar a los

estudiantes con las funciones principales del motor mediante explicaciones y actividades realizadas dentro del propio proyecto[14].

Desde el área de programación, los autores explicaron que un videojuego puede organizarse mediante varios subsistemas conectados con un sistema central. En su esquema incluyeron el control del movimiento, la visualización y el sonido. El esquema permite observar que el funcionamiento de un juego no depende de un único componente, sino de la comunicación entre sistemas con responsabilidades diferentes[14]. La Ilustración 2 identifica estos subsistemas mediante las siglas VS, AS y CS, y muestra su relación con el sistema básico de juego ubicado en el centro.

Ilustración 2

Arquitectura básica de los subsistemas de un videojuego.



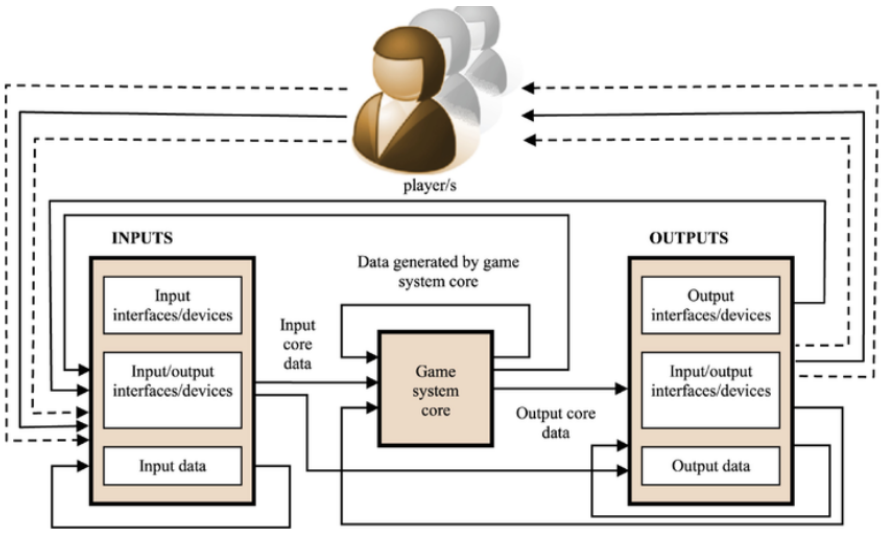
Nota. Sobota y Pietriková

Los autores también representaron el funcionamiento de un videojuego como un flujo continuo de datos. Las acciones del jugador ingresan mediante dispositivos o interfaces, se convierten en datos de entrada y son procesadas por el núcleo del sistema. Después, este núcleo genera información de salida que se envía a los dispositivos encargados de producir

respuestas visuales, sonoras o interactivas. La Ilustración 3 añade la idea de retroalimentación, ya que las salidas regresan al jugador y pueden generar nuevas entradas, formando un ciclo continuo[14].

Ilustración 3

Flujo básico de datos dentro de un sistema de videojuego.



Nota. Sobota y Pietriková

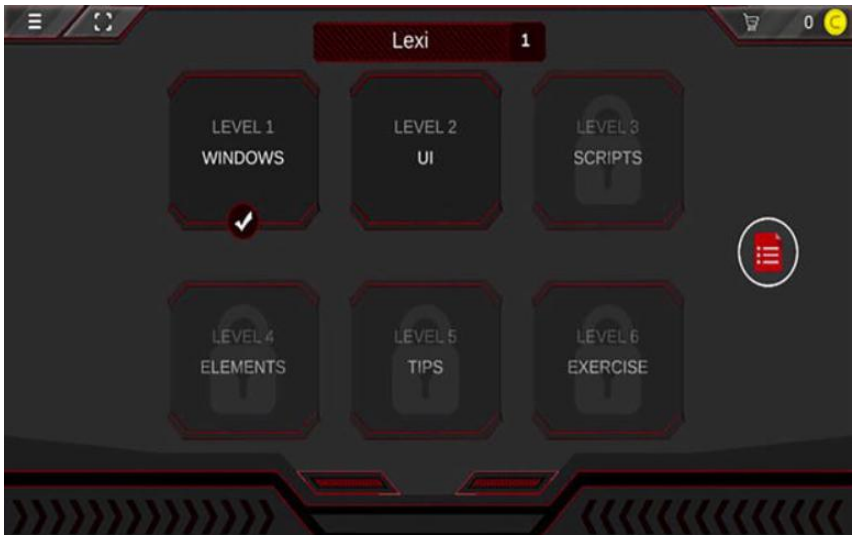
Dentro del análisis de motores, Unity fue descrito como una herramienta que facilita la creación rápida de prototipos mediante un sistema basado en componentes agregados a los objetos del juego. Los autores también destacaron la cantidad de recursos de aprendizaje disponibles, su comunidad de desarrolladores y la posibilidad de exportar proyectos a diferentes plataformas[14].

El tutorial interactivo fue organizado en seis niveles, cada uno dedicado a un área distinta de Unity. Los niveles se desbloqueaban de manera progresiva, por lo que era necesario completar uno antes de acceder al siguiente. Cada nivel incluía una explicación breve y un video con información más detallada. La mayoría también incorporaba ejercicios prácticos, pruebas e indicaciones de ayuda para orientar al estudiante durante la

actividad[14]. La Ilustración 4 permite reconocer el avance mediante la marca del nivel completado y los candados que restringen el acceso a los niveles posteriores.

Ilustración 4

Interfaz de selección de niveles del tutorial interactivo en Unity.



Nota. Sobota y Pietriková

El último nivel presentó un minijuego con errores incorporados de manera intencional. Para finalizar la actividad, los estudiantes debían localizar los problemas y corregirlos hasta conseguir que el juego funcionara. Al terminar, también debían completar un cuestionario y entregar una evidencia del nivel alcanzado[14]. De esta forma, el tutorial combinó la explicación de funciones del motor con ejercicios, resolución de errores y seguimiento del avance.

Los autores consideraron que el tutorial de Unity favorecía el aprendizaje activo, el trabajo al propio ritmo, la resolución de problemas y la retroalimentación durante las actividades[14]. La investigación muestra una forma de integrar contenido y práctica dentro de un motor de videojuegos, usando niveles progresivos y ejercicios que requieren la participación directa del estudiante. Sin embargo, la documentación revisada se concentra

principalmente en el aprendizaje de las funciones generales de Unity y en el uso del motor como herramienta educativa.

2.1.2. Using Unity to Teach Game Programming

Golob y Pagnon Eriksson desarrollaron un recurso educativo para enseñar programación de videojuegos a estudiantes de primer año de la carrera de Desarrollo de Videojuegos de Malmö University. El trabajo buscó complementar un curso en el que los estudiantes tenían su primer contacto práctico con Unity y debían crear un juego dentro de un tiempo limitado. Para atender esta necesidad, los autores construyeron una plantilla inicial de un videojuego de rol y la acompañaron con tutoriales escritos paso a paso[15].

La propuesta estuvo formada por dos componentes. El primero fue un proyecto funcional en Unity que reunía las mecánicas desarrolladas durante los tutoriales. El segundo consistió en documentos que explicaban cómo implementar cada tema desde una escena preparada. La versión final servía como referencia para comprobar el resultado, mientras que las escenas iniciales permitían comenzar cada actividad sin tener que construir todo el proyecto desde cero[15].

El proyecto de Unity representó un juego de aventura en primera persona. La escena permitía controlar un personaje, interactuar con objetos, aceptar una misión, recoger un arma y combatir contra un objetivo. Los autores organizaron los archivos en dos carpetas principales: FinalProject, que contenía las soluciones completas y los scripts comentados, y TutorialProject, que reunía los materiales iniciales necesarios para desarrollar cada actividad. La Ilustración 5 presenta una vista general del escenario en el que se integraron las actividades descritas[15].

Ilustración 5

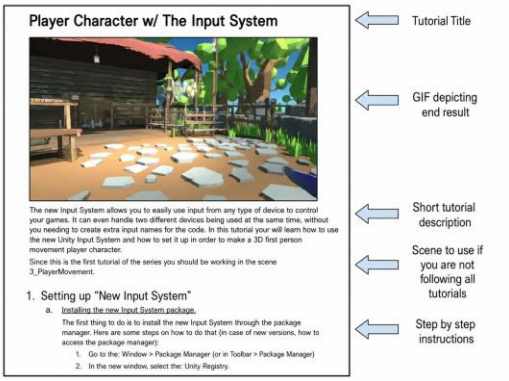
Vista general del escenario utilizado en los tutoriales de programación en Unity.

Nota. Golob y Pagnon Eriksson

La documentación siguió una estructura común para todos los tutoriales. Cada documento incluía un título, una introducción, la escena inicial que debía utilizarse, el desarrollo paso a paso y una sección denominada Challenge Yourself. La introducción mostraba, cuando era posible, una animación del resultado esperado. Después, las instrucciones explicaban el proceso con imágenes y la sección final proponía una actividad para aplicar lo aprendido de una manera diferente[15]. La Ilustración 6 presenta un ejemplo de esta plantilla aplicada a uno de los tutoriales.

Ilustración 6

Estructura general de los documentos tutoriales.



Nota. Golob y Pagnon Eriksson

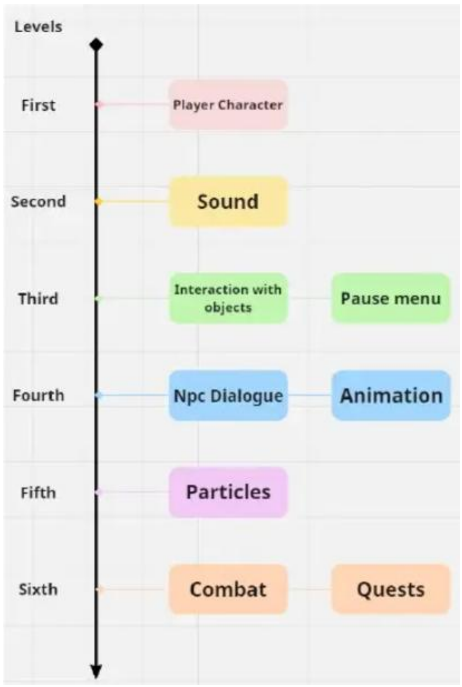
Los contenidos abordaron la configuración del sistema de entradas, el movimiento y la rotación de cámara, el sonido, la interacción con objetos mediante rayos e interfaces, la interfaz del menú de pausa, el diálogo con personajes, las animaciones, las partículas, las misiones y el combate. Algunos tutoriales introdujeron conceptos de programación como clases de instancia única[9], objetos programables, corrutinas[10], colas, acciones e interfaces. De esta manera, el recurso no se limitó al uso de las ventanas del motor, sino que también presentó sistemas y estructuras aplicadas a mecánicas de videojuego[15].

Aunque cada tutorial podía realizarse de forma independiente, los autores propusieron una secuencia para quienes deseaban completar todo el contenido. Este orden respondía a que algunos temas utilizaban objetos o scripts construidos en actividades anteriores. Por ejemplo, el sistema de diálogo retomaba la lógica de interacción desarrollada previamente. La independencia entre los documentos también permitía que los estudiantes con conocimientos anteriores omitieran ciertos temas y comenzaran desde un punto más avanzado[15]. La Ilustración 7 muestra que algunos contenidos se ubicaban en un mismo nivel, lo que permitía desarrollarlos de forma paralela dentro de la secuencia recomendada.

[9] Clase de instancia única: clase que mantiene una sola instancia durante la ejecución.
[10] Corrutina: rutina que puede pausarse y continuar su ejecución posteriormente.

Ilustración 7

Secuencia sugerida para el desarrollo de los tutoriales.



Nota. Golob y Pagnon Eriksson

El desarrollo de cada tema siguió tres acciones. Primero se implementaba la funcionalidad dentro de la plantilla; luego se redactaba el tutorial paso a paso; y, finalmente, el otro integrante del equipo realizaba la actividad y entregaba observaciones. Los autores también recibieron comentarios de docentes durante el proceso, que sirvieron para corregir tanto la implementación como las explicaciones[15].

Para evaluar el recurso participaron nueve estudiantes de primer año y cuatro docentes o asistentes del curso. Los estudiantes realizaron una actividad mientras explicaban en voz alta lo que pensaban y después participaron en una entrevista. Los asistentes del curso revisaron el proyecto desde una perspectiva educativa. Las respuestas fueron transcritas y analizadas por temas, aunque los propios autores reconocieron que el tamaño de la muestra fue una limitación[15].

La valoración general fue favorable. Los estudiantes mostraron interés por utilizar el recurso y afirmaron haber aprendido información nueva o haber ampliado conocimientos anteriores. Siete de los ocho participantes que respondieron sobre el orden de los tutoriales indicaron que preferirían seguir la secuencia sugerida de principio a fin. Los especialistas consideraron que los temas eran relevantes y que el recurso podía complementar el curso, aunque también señalaron la necesidad de mejorar algunas instrucciones, incorporar más imágenes y establecer una forma clara de mantenimiento y evaluación[15].

Este trabajo muestra una forma de combinar una plantilla funcional, escenas preparadas y documentos escritos para enseñar programación de videojuegos. Entre sus fortalezas se encuentran la variedad de mecánicas, la posibilidad de consultar una solución completa y la organización de los contenidos mediante una secuencia flexible. Sus principales limitaciones fueron la muestra reducida, la falta de implementación de todos los temas opcionales previstos y la necesidad de mantener actualizado el material ante los cambios en las versiones del motor[15].

2.1.3. GameDevDojo

Holly, Habich y Pirker presentaron GameDevDojo, una aplicación educativa desarrollada en Unity para enseñar conceptos básicos relacionados con la creación de videojuegos. La propuesta surgió ante la limitada presencia de métodos interactivos y lúdicos para explicar el propio desarrollo de juegos. En lugar de introducir a los usuarios directamente en un motor profesional con todas sus herramientas, la aplicación ofrece un entorno simplificado en el que pueden experimentar con objetos, componentes y comportamientos previamente preparados[16].

La aplicación utilizó el sistema de físicas de Unity para permitir que los usuarios modificaran objetos y observaran los efectos de los cambios durante la ejecución. El jugador podía seleccionar elementos del escenario, añadir componentes, modificar sus propiedades y comprobar inmediatamente cómo estas decisiones afectaban al comportamiento del nivel. El sistema no exigía conocimientos previos específicos y contaba con un manual inicial que incluía un video para explicar sus controles y principales funciones[16].

Los autores incorporaron elementos de gamificación como reconocimiento, elección impuesta, novedad, objetivos, progresión, repetición y elementos sensoriales. Cuando el usuario configuraba correctamente una parte de la actividad, el sistema muestra una animación de confeti. También se permitía reiniciar los niveles, repetir actividades ya desbloqueadas y consultar el estado de los componentes necesarios para alcanzar cada objetivo[16].

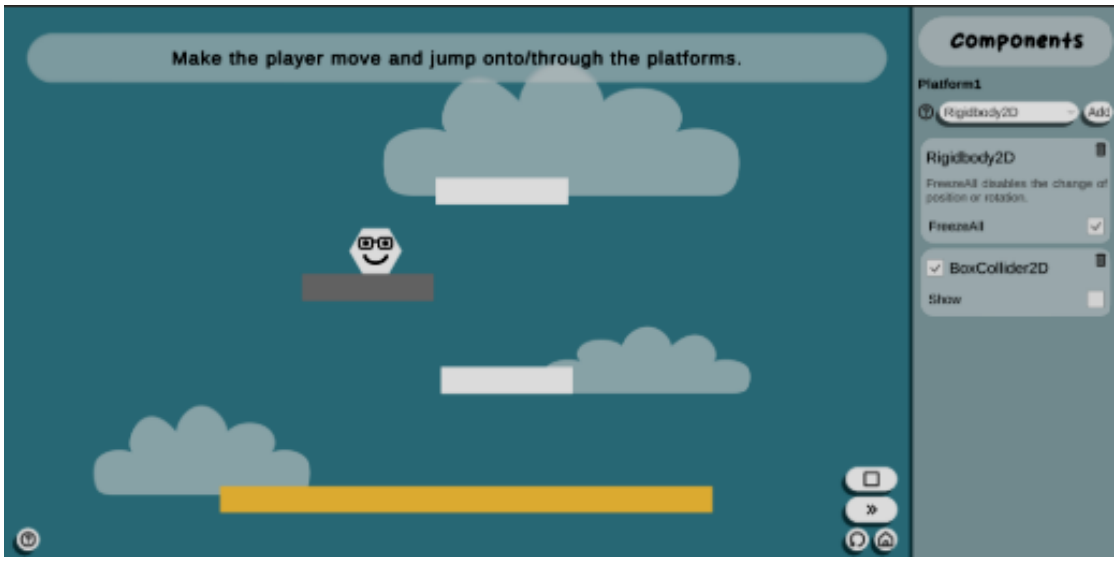
La organización del aprendizaje se realizó mediante cinco niveles progresivos. El primero abordaba el cambio de apariencia de un objeto y la aplicación de gravedad. El segundo y el tercero se concentraban en colisiones y tipos de colliders, incluyendo consideraciones sobre el uso de memoria. El cuarto nivel trabajaba el movimiento y el salto mediante scripts previamente definidos y componentes de plataformas. Finalmente, el quinto nivel introducía activadores, elementos coleccionables y componentes de interfaz[16].

Cada nivel muestra el objetivo en la parte superior de la pantalla. En el lateral derecho se ubicaba una vista para añadir o modificar los componentes de los objetos, mientras que los botones inferiores permiten iniciar, detener y validar el funcionamiento de la solución. Esta distribución ayuda a mantener en una misma interfaz el escenario de prueba, las instrucciones y las opciones necesarias para completar la actividad[16]. La Ilustración 8 permite identificar esta organización mediante la separación entre el área central de trabajo,

el panel lateral de componentes y los controles de validación situados en la parte inferior derecha.

Ilustración 8

Diseño de un nivel de GameDevDojo con escenario, objetivo, vista de componentes y controles.



Nota. Holly, Habich y Pirker

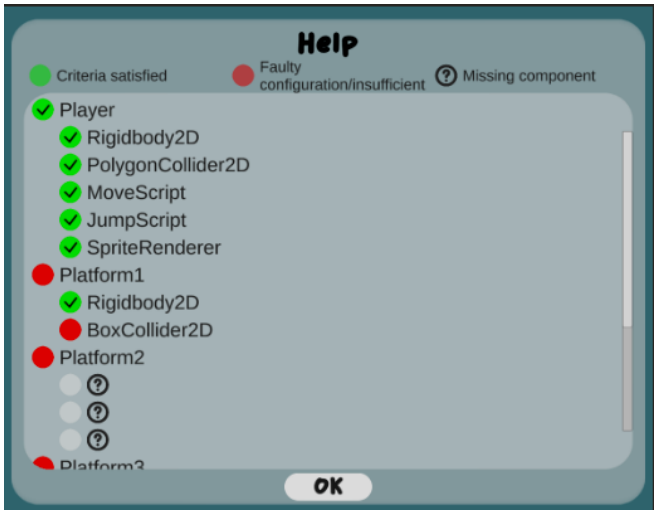
Para completar los niveles, el usuario debe seleccionar los objetos y añadir componentes desde un menú desplegable. Entre los elementos disponibles se encuentran SpriteRenderer2D, Rigidbody2D, diferentes tipos de Collider2D y scripts previamente preparados. Estos últimos permiten incorporar comportamientos como el movimiento y el salto mediante propiedades ajustables, por ejemplo, la velocidad de desplazamiento o la altura del salto[16].

La aplicación también incorporó un sistema de ayuda que muestra el estado de cada objeto y de los componentes requeridos. Los indicadores verdes señalan configuraciones correctas, mientras que los rojos identifican componentes ausentes o propiedades configuradas de forma incorrecta. Además, el usuario puede consultar descripciones sobre la función de cada componente mediante botones de ayuda integrados en la interfaz[16]. La

Ilustración 9 permite reconocer este mecanismo de verificación mediante una lista jerárquica de objetos y componentes, en la que los colores distinguen de forma inmediata las configuraciones correctas y los elementos que requieren revisión.

Ilustración 9

Vista de ayuda para identificar componentes faltantes o configurados incorrectamente.



Nota. Holly, Habich y Pirker

La evaluación se realizó mediante un estudio con 57 estudiantes de educación media, con edades entre 12 y 18 años. Los participantes fueron divididos en dos grupos: 25 estudiantes utilizaron directamente GameDevDojo, mientras que 32 recibieron primero un documento teórico de ocho páginas con los mismos conceptos. Antes y después de la actividad, ambos grupos respondieron diez preguntas de conocimiento. También se evaluaron la experiencia de usuario, el aprendizaje, el diseño y el nivel de participación mediante cuestionarios estandarizados[16].

Los resultados mostraron que los dos grupos mejoraron sus respuestas después de la actividad. El grupo que utilizó GameDevDojo presentó un incremento del 16,4%, mientras que el grupo que trabajó con el documento teórico alcanzó un incremento del 25,3%. Sin embargo, la diferencia general entre ambos grupos no fue estadísticamente significativa. Los

estudiantes que utilizaron la aplicación mostraron un ligero aumento en su interés por el desarrollo de videojuegos, mientras que en el grupo teórico el interés disminuyó después de la actividad[16].

En los niveles iniciales, los participantes que recibieron previamente la explicación teórica obtuvieron mejores porcentajes de finalización. Esta tendencia cambió en las actividades más avanzadas, donde quienes habían utilizado directamente el entorno interactivo resolvió las tareas con mayor facilidad. Los autores señalaron que este resultado podía estar relacionado con la aplicación práctica de los conocimientos, aunque también reconocieron que el tiempo limitado disponible para el segundo grupo pudo influir en su rendimiento[16].

La valoración de la aplicación fue favorable en las categorías de aprendizaje, participación y diseño. Los estudiantes consideraron que las funciones eran útiles y fáciles de utilizar, y señalaron que volverían a trabajar con este tipo de recurso. También se valoraron de manera positiva la claridad de los objetivos, la retroalimentación sobre el progreso y los elementos audiovisuales[16]. Seguidamente, la Tabla 1 sintetiza estos resultados y permite observar que la categoría de diseño y facilidad de uso obtuvo el promedio más alto, con 4,52 sobre 5.

Tabla 1

Valoración de la experiencia con GameDevDojo en una escala de 1 a 5.

Criterio evaluado	Promedio sobre 5
Aprendizaje	4,23
Participación	4,38
Diseño y facilidad de uso	4,52

Nota. Elaboración propia con base en Holly, Habich y Pirker

El trabajo muestra que los conceptos relacionados con componentes, físicas, colisiones y scripts pueden presentarse mediante actividades interactivas sin utilizar

inicialmente toda la complejidad de un motor de videojuegos. Entre sus fortalezas se encuentran la progresión por niveles, la observación inmediata de los cambios, la validación de las soluciones y el sistema de ayuda. Como limitaciones, los autores mencionaron el tamaño de la muestra, el uso de respuestas basadas en la percepción de los participantes y la duración de 100 minutos de la evaluación, que no permitió analizar los efectos del aprendizaje a largo plazo[16].

2.1.4. Cuadro comparativo de trabajos relacionados

Después de analizar los tres trabajos, se establecieron criterios comunes para comparar sus características técnicas y didácticas. Se consideraron el tipo de recurso desarrollado, la organización del aprendizaje, los contenidos abordados, la forma de presentar las actividades, los mecanismos de ayuda y validación, los materiales complementarios y el proceso de evaluación. Esta comparación permite reconocer las estrategias utilizadas para enseñar programación y desarrollo de videojuegos mediante experiencias prácticas apoyadas en Unity[14], [15], [16].

Tabla 2

Comparación de los trabajos relacionados.

Criterio	Interactive Tutorial in Unity	Using Unity to Teach Game Programming	GameDevDojo
Tipo de recurso	Tutorial interactivo integrado dentro de Unity.	Plantilla funcional de un videojuego de rol acompañada por tutoriales escritos.	Aplicación educativa desarrollada en Unity.
Público objetivo	Estudiantes universitarios de desarrollo de videojuegos.	Estudiantes de primer año de Desarrollo de Videojuegos.	Estudiantes de educación media entre 12 y 18 años.
Organización del aprendizaje	Seis niveles progresivos divididos en secciones y pasos.	Tutoriales independientes con una secuencia recomendada.	Cinco niveles que aumentan gradualmente su dificultad.
Contenidos y enfoque técnico	Funciones generales de Unity, interfaz, scripts y corrección de errores.	Entrada, movimiento, cámara, sonido, interacción, diálogos,	Componentes, gravedad, colisiones, movimiento, salto, activadores e interfaz.

		animación, partículas, misiones y combate.	
Estrategia didáctica	Integra textos, videos, ejercicios, ayudas, desbloqueo de niveles y recompensas dentro del motor.	Utiliza documentos paso a paso, escenas preparadas, ejemplos visuales y retos finales.	Propone experimentación directa, objetivos por nivel, manual inicial y retroalimentación visual.
Participación en la programación	El usuario completa tareas y trabaja con elementos y scripts dentro de Unity.	El usuario escribe y modifica scripts en C# para implementar diferentes mecánicas.	El usuario configura componentes y scripts previamente preparados, con escritura directa de código limitada.
Validación y retroalimentación	Los scripts comprueban las actividades antes de permitir el avance.	El resultado se contrasta con el proyecto final y las instrucciones del tutorial; no existe validación automática.	El sistema verifica componentes y propiedades mediante indicadores visuales, mensajes y una opción de validación.
Evaluación realizada	Aplicación en un contexto universitario y observación del tiempo empleado para completar el tutorial.	Pruebas con estudiantes y revisión de docentes o especialistas mediante actividades y entrevistas.	Estudio comparativo con 57 participantes y cuestionarios aplicados antes y después de la actividad.
Aporte principal	Integra explicación, práctica y comprobación dentro del mismo entorno.	Combina un proyecto funcional con material escrito para enseñar distintas mecánicas.	Facilita la experimentación y ofrece retroalimentación inmediata sobre la configuración realizada.
Limitación principal	Se concentra principalmente en el uso general de las funciones de Unity.	Depende de documentos externos y no verifica automáticamente el avance.	Utiliza componentes y scripts preconfigurados, por lo que la programación directa es limitada.

Nota. Elaboración propia con base en Sobota y Pietriková, Golob y Pagnon Eriksson y Holly, Habich y Pirker

Los tres trabajos integran la práctica como parte central del aprendizaje, aunque aplican estrategias diferentes. *Interactive Tutorial in Unity* incorpora las instrucciones y comprobaciones dentro del propio motor; *Using Unity to Teach Game Programming* relaciona un proyecto funcional con tutoriales escritos; y *GameDevDojo* simplifica la interacción con componentes para permitir la experimentación y la retroalimentación inmediata. En conjunto, los trabajos muestran que la progresión de contenidos, la práctica guiada, la observación de resultados y la orientación ante los errores son elementos recurrentes en la enseñanza del desarrollo de videojuegos. También se observa que ninguna de las propuestas reúne al mismo tiempo la escritura directa de código, la validación

automática de cada actividad y un material didáctico complementario centrado en la explicación de los fundamentos lógicos y matemáticos de las mecánicas[14], [15], [16].

2.2. Bases teóricas

En esta sección se revisan estudios sobre el uso de videojuegos y entornos interactivos en la enseñanza de programación. Estos trabajos permiten entender cómo la práctica dentro de un sistema visual puede apoyar la resolución de problemas y facilitar la comprensión de los efectos que produce el código. Luego se estudian las características de los entornos guiados, la organización didáctica de sus contenidos y el aporte que pueden ofrecer al relacionar la teoría con la práctica.

2.2.1. Evolución del aprendizaje de programación mediante videojuegos

Aprender programación no consiste solamente en conocer la forma correcta de escribir el código. También es necesario entender el problema, separar sus partes y ordenar los pasos necesarios para llegar a una solución. Wing relaciona estas capacidades con el pensamiento computacional, entendido como una forma de abordar problemas mediante conceptos como la descomposición, la abstracción y la construcción de procesos que puedan ser ejecutados por una persona o por una máquina[5]. Luxton-Reilly organizan la literatura sobre enseñanza inicial de programación en torno a cuatro dimensiones: los estudiantes, la enseñanza, el currículo y la evaluación[17].

Las investigaciones sobre enseñanza de programación han incorporado videojuegos para presentar algunos conceptos mediante actividades prácticas. En estos recursos, el estudiante puede realizar una acción y observar la respuesta del sistema. Esto permite relacionar una instrucción con un resultado visible, en lugar de trabajar únicamente con explicaciones o ejercicios escritos[4], [16].

Kazimoglu desarrollaron un juego serio[11] en el que conceptos iniciales de programación fueron relacionados con habilidades de pensamiento computacional y acciones realizadas dentro de la experiencia. El juego incluyó actividades vinculadas con la creación de algoritmos, la simulación, la resolución de problemas y la corrección de errores. La evaluación inicial contó con 25 estudiantes y mostró una percepción favorable sobre el uso del recurso, aunque los autores señalaron que eran necesarias pruebas más amplias[4].

Otros estudios se han ocupado de la posibilidad de adaptar y reutilizar juegos educativos. Silva y Silveira revisaron recursos abiertos creados para enseñar programación y lógica computacional. Los autores encontraron que muchos trabajos recomiendan utilizar componentes que puedan modificarse o reutilizarse, pero observaron que esta práctica todavía es limitada. También señalaron que varios juegos disponibles no cuentan con documentación académica que permita conocer su desarrollo o sus resultados[18].

Con el tiempo, también aparecieron propuestas centradas en enseñar conceptos propios del desarrollo de videojuegos. GameDevDojo fue creado para trabajar fundamentos de esta área mediante una experiencia con objetivos y niveles. El estudio comparó el recurso con una forma tradicional de enseñanza y contó con 57 participantes. Los resultados mostraron mejoras en el aprendizaje y una valoración favorable de la motivación generada por la actividad[16].

Los motores de videojuegos también se han incorporado en cursos de formación. Kuznetsov et al. describieron un curso basado en Unity que combinó explicaciones, actividades de laboratorio y el desarrollo de un proyecto. Los participantes valoraron de forma positiva las prácticas realizadas y la comprensión del proceso de creación de

[11] Juego serio: videojuego creado con una finalidad educativa o formativa.

videojuegos. Más adelante, Moiseienko presentó otro curso basado en Unity que integró programación, diseño, pruebas, planificación y trabajo colaborativo[8].

Los estudios revisados muestran que la enseñanza de programación mediante videojuegos puede adoptar varias formas. Algunos recursos presentan conceptos dentro de un juego educativo[4], [16], mientras que otros permiten construir proyectos directamente en un motor[8]. En ambos casos, la posibilidad de observar los resultados del código facilita la práctica, pero debe estar acompañada por instrucciones y actividades que ayuden a comprender lo que ocurre.

2.2.2. Estructura narrativa y didáctica del aprendizaje guiado

Un entorno guiado organiza las actividades para que el usuario reciba apoyo mientras aprende a completar una tarea. Este acompañamiento se relaciona con el andamiaje educativo, que consiste en ofrecer ayuda según las necesidades del estudiante y reducirla cuando adquiere mayor seguridad. Van de Pol identifica como elementos principales la adaptación del apoyo, su disminución progresiva y la transferencia de responsabilidad hacia quien aprende[11].

En programación, esta orientación puede aplicarse mediante instrucciones divididas en pasos. Al comienzo, el estudiante recibe explicaciones detalladas sobre los componentes y acciones que debe utilizar. Más adelante, puede resolver actividades con menos indicaciones y aplicar los conocimientos adquiridos en ejercicios anteriores. Esto evita presentar al mismo tiempo una gran cantidad de variables, condiciones, métodos y componentes[11].

Los ejemplos resueltos también pueden ayudar a explicar una mecánica antes de pedir al estudiante que la construya. Margulieux estudió ejemplos de programación que incluían

etiquetas breves para indicar el propósito de cada parte del proceso. Esta forma de presentar el contenido ayudó a los estudiantes a reconocer los pasos utilizados para resolver un problema y se relacionó con una reducción de retiros y resultados insuficientes en cursos iniciales[10].

Otro elemento importante es la respuesta que recibe el usuario al completar una actividad. En un entorno guiado, esta puede indicar si falta un componente, si una variable tiene un valor incorrecto o si una acción aún no se ha ejecutado. Su función no es solo señalar el error, sino orientar al usuario sobre la parte que debe revisar antes de continuar. Al finalizar la actividad, el sistema debe ofrecer una retroalimentación que confirme el resultado obtenido y comunique si el usuario puede avanzar al siguiente paso[12].

Toukiloglou y Xinogalos analizaron los sistemas de apoyo utilizados en juegos serios para programación. Su revisión incluyó 18 publicaciones relacionadas con diez juegos distintos. Los autores encontraron diferentes formas de presentar ayudas y retroalimentación, pero también señalaron que este campo todavía se encuentra en una etapa inicial y que no todos los sistemas han sido evaluados con el mismo nivel de profundidad[12].

La organización técnica también influye en la forma de enseñar. Cuando cada mecánica se divide en componentes con una función definida, es más sencillo trabajarla por separado y comprobar su comportamiento antes de integrarla. Silva y Silveira encontraron que la organización basada en componentes puede favorecer la adaptación y reutilización de los juegos educativos, aunque todavía existen pocas propuestas documentadas que aprovechen completamente esta posibilidad[18].

Por lo tanto, un entorno guiado para programación debe combinar instrucciones claras, actividades progresivas, ejemplos explicados y respuestas que ayuden a identificar errores. Estos recursos deben orientar el proceso sin reemplazar la participación del

estudiante, ya que el objetivo final es que pueda comprender y modificar una solución con mayor autonomía.

2.2.3. Estructura y secuencia didáctica del libro complementario

La narrativa se entiende como la forma en que se ordenan y conectan las explicaciones, las instrucciones y las actividades del libro didáctico. Aunque el recurso también incluye elementos visuales, su diseño gráfico corresponde al área visual del proyecto. Por esta razón, este apartado se concentra en la estructura del contenido, la claridad de las explicaciones y su relación con los conceptos y procesos propios del área de programación.

Los fragmentos de código no deben aparecer como soluciones aisladas. Es necesario explicar la función que cumple cada grupo de instrucciones y su relación con el comportamiento de la mecánica. Margulieux señala que dividir un procedimiento en objetivos más pequeños ayuda a reconocer los pasos utilizados para resolver un problema. Esta forma de organización permite que el estudiante comprenda la lógica del proceso antes de intentar modificarlo o aplicarlo en otra situación[10]. La investigación analiza ejemplos resueltos de programación organizados mediante subobjetivos y su utilidad para reconocer la estructura de una solución.

Los diagramas, capturas y ejemplos pueden apoyar las explicaciones, pero en este apartado se consideran por su función didáctica y no por su composición visual. Mayer explica que las palabras y las imágenes pueden facilitar la comprensión cuando se complementan y se presentan cerca del contenido con el que se relacionan. En un libro sobre programación, estos recursos ayudan a representar procesos, mostrar el funcionamiento de

una mecánica y relacionar el código con el comportamiento obtenido en un entorno interactivo[13].

La dificultad del contenido también debe avanzar de forma progresiva. Los primeros capítulos pueden presentar explicaciones más detalladas, mientras que los siguientes retoman conocimientos anteriores y los integran en mecánicas más complejas. Esta disminución gradual del apoyo se relaciona con la transferencia de responsabilidad hacia el estudiante conforme adquiere mayor experiencia[11].

Los libros y materiales didácticos también pueden apoyar el aprendizaje guiado al ofrecer una referencia que el estudiante puede consultar durante la práctica. Los ejemplos explicados y organizados en pasos ayudan a reconocer el proceso seguido para resolver un problema de programación[10]. En el campo del desarrollo de videojuegos, Golob y Pagnon Eriksson elaboraron una plantilla de juego acompañada por tutoriales escritos para enseñar temas básicos de programación. La propuesta fue valorada de forma favorable por estudiantes y especialistas como un material de apoyo para el aprendizaje práctico[15]. De esta manera, un recurso escrito puede acompañar la creación de mecánicas al explicar el propósito del código y orientar la revisión de cada paso, sin reemplazar la práctica dentro del entorno de desarrollo.

2.2.4. Aporte educativo y tecnológico

Los videojuegos educativos permiten practicar dentro de un entorno en el que las acciones producen una respuesta visible. En el trabajo de Kazimoglu et al., las actividades del juego se relacionaron con procesos como la construcción de algoritmos, la simulación y la corrección de errores[4]. GameDevDojo también mostró que una experiencia basada en juegos podía utilizarse para enseñar fundamentos del desarrollo de videojuegos. El grupo que

utilizó la aplicación mostró mayor interés y resolvió con mayor rapidez algunas tareas avanzadas; sin embargo, la diferencia global en el incremento del conocimiento frente al grupo que recibió material teórico no fue estadísticamente significativa[16].

En la enseñanza de mecánicas, esta posibilidad resulta útil porque el estudiante puede modificar una variable o una condición y observar el cambio dentro de la escena[16]. El resultado permite comprobar si la solución funciona, pero también sirve para analizar por qué el sistema responde de una forma determinada.

Desde el punto de vista tecnológico, Unity reúne escenas, objetos, componentes y scripts dentro de un mismo entorno[19]. Los cursos estudiados por Kuznetsov y Moiseienko muestran que el motor puede utilizarse para trabajar programación, resolución de problemas y creación de proyectos relacionados con videojuegos. También permite que los estudiantes practiquen con una herramienta utilizada fuera del contexto académico[7], [8].

El uso de una herramienta de desarrollo no garantiza por sí mismo que el aprendizaje sea claro. Silva y Silveira encontraron que todavía existen pocos juegos educativos abiertos para programación con documentación académica suficiente[18]. Toukiloglou y Xinogalos también indican que los apoyos incorporados en juegos de programación no siempre han sido evaluados de manera amplia[12]. Estos resultados muestran la importancia de describir cómo funciona el recurso, explicar la relación entre las actividades y los contenidos, y recoger información de los usuarios.

Los antecedentes revisados muestran que los videojuegos y motores de desarrollo pueden apoyar la enseñanza de programación cuando las actividades, las explicaciones y las respuestas del sistema se encuentran relacionadas. Esta base sustenta la creación de una propuesta que no se limite a mostrar código, sino que ayude a comprender cómo cada instrucción influye en el comportamiento de una mecánica.

2.3. Bases metodológicas

Las metodologías de desarrollo establecen una forma organizada de planificar, ejecutar y controlar las actividades necesarias para construir un producto de software. Su elección influye en la distribución de las etapas, la documentación, la participación del equipo, la gestión de cambios y la manera en que se realizan las entregas. En este apartado se analizan las metodologías Cascada, Scrum y Extreme Programming, debido a que representan enfoques diferentes para la administración de proyectos: uno secuencial y dos de carácter iterativo. Posteriormente, se comparan sus principales características, ventajas y limitaciones para determinar cuál se ajusta mejor a la organización y las necesidades del proyecto.

2.3.1. Cascada

La metodología en cascada organiza el desarrollo de software mediante una secuencia de etapas definidas previamente. Cada fase produce información o resultados que sirven como entrada para la siguiente, por lo que el avance se realiza de manera ordenada desde la identificación de los requerimientos hasta la entrega y el mantenimiento del sistema. Este enfoque prioriza la planificación inicial, la documentación y el seguimiento del proceso antes y durante la implementación[20].

El origen de este modelo suele relacionarse con el trabajo presentado por Winston W. Royce en 1970 sobre la administración de sistemas de software de gran tamaño. En dicho estudio se representó un proceso compuesto por requerimientos del sistema, requerimientos del software, análisis, diseño, codificación, pruebas y operación. Sin embargo, Royce también señaló los riesgos de aplicar estas etapas como una secuencia completamente rígida

y recomendó incorporar revisiones, documentación, participación del cliente y retroalimentación hacia fases anteriores[21].

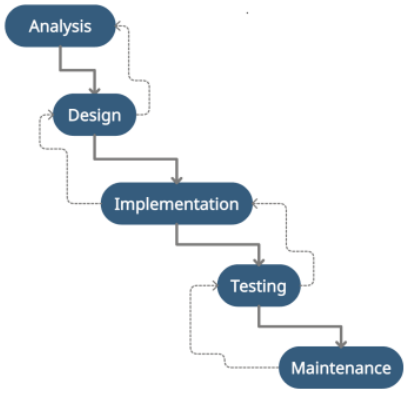
Aunque existen distintas adaptaciones del modelo, su estructura suele organizarse en las siguientes fases:

Análisis de requerimientos	Se identifican las necesidades que deberá satisfacer el sistema, sus funciones, restricciones, usuarios y condiciones de operación. El resultado de esta etapa debe permitir comprender qué se desarrollará antes de definir su estructura técnica.
Diseño	Se establece cómo estará organizado el software. En esta fase se determinan la arquitectura, los componentes, las relaciones entre los elementos, las estructuras de datos, las interfaces y los recursos requeridos para responder a los requerimientos definidos.
Implementación	Los diseños se convierten en código y se construyen los componentes que conforman el sistema. Las actividades de esta fase se apoyan en las especificaciones preparadas anteriormente para mantener correspondencia entre lo planificado y el producto desarrollado.
Pruebas	Se comprueba el funcionamiento de los componentes y su integración. Las pruebas permiten identificar errores, verificar el cumplimiento de los requerimientos y revisar que las funciones produzcan los resultados esperados.
Despliegue y mantenimiento	El producto se prepara para su utilización y se atienden errores o ajustes identificados después de la entrega. El mantenimiento también puede incluir modificaciones requeridas para conservar la operatividad del sistema.

Estas fases representan una adaptación de la estructura original, debido a que la cantidad y el nombre de las etapas pueden variar entre diferentes autores y organizaciones. A pesar de estas variaciones, se conserva el principio de completar y revisar una fase antes de continuar con la siguiente[20]. La Ilustración 25 representa esta secuencia mediante flechas continuas entre las etapas, mientras que las líneas discontinuas indican la posibilidad de regresar a una fase anterior para realizar ajustes.

Ilustración 10

Fases del modelo de desarrollo en cascada.



Nota. Saravanos y Curinga

Entre las ventajas de esta metodología se encuentra la posibilidad de establecer desde el inicio las actividades, los entregables y la documentación correspondiente a cada fase. Esta organización facilita el seguimiento del progreso, la distribución de responsabilidades y la revisión de los resultados obtenidos antes de avanzar. Puede resultar apropiada cuando los requerimientos se encuentran suficientemente definidos, el alcance presenta poca variación y se necesita mantener un registro claro del proceso[22].

Su principal limitación aparece cuando se producen cambios importantes después de haber completado las etapas iniciales. Una modificación tardía puede obligar a revisar los requerimientos, el diseño y parte de la implementación. Asimismo, cuando las pruebas generales se concentran al final, algunos problemas de integración pueden descubrirse después de haber invertido tiempo y recursos en las fases anteriores[22]. Royce advirtió que los inconvenientes encontrados durante las pruebas podían ocasionar modificaciones considerables en el diseño y en los requerimientos[21].

Por esta razón, la metodología en cascada requiere requerimientos claros, documentación consistente y revisiones antes de cerrar cada etapa. Su estructura permite

mantener una ruta de trabajo definida, pero ofrece menor flexibilidad que los enfoques iterativos cuando el alcance cambia constantemente[22].

2.3.2. Scrum

Scrum es un marco de trabajo ágil utilizado para abordar problemas complejos mediante entregas parciales y ciclos de trabajo repetitivos. Aunque suele mencionarse como una metodología, la Guía Scrum lo define como un marco ligero que orienta a las personas, los equipos y las organizaciones en la generación de valor mediante soluciones adaptables. Su estructura establece responsabilidades, eventos y artefactos, pero permite incorporar técnicas y prácticas adicionales según las características del proyecto[23].

Este marco se fundamenta en el empirismo, según el cual las decisiones se toman a partir de la experiencia y de la observación de los resultados. Para sostener este enfoque se utilizan tres pilares: transparencia, inspección y adaptación. La transparencia permite que el trabajo y su estado sean comprensibles; la inspección ayuda a detectar problemas o desviaciones; y la adaptación permite realizar ajustes cuando los resultados obtenidos no conducen al objetivo previsto[23].

Scrum también establece cinco valores que orientan el comportamiento del equipo: compromiso, enfoque, apertura, respeto y valentía. Estos valores favorecen la colaboración, la comunicación de los problemas y la concentración en los objetivos definidos para el producto y para cada ciclo de trabajo[23].

El equipo Scrum es una unidad autogestionada y multidisciplinaria que trabaja sobre un mismo objetivo de producto. Generalmente está compuesto por diez personas o menos y reúne las capacidades necesarias para generar un incremento útil durante cada Sprint. Dentro del equipo se reconocen tres responsabilidades principales[23].

Tabla 3

Responsabilidades del Equipo Scrum.

Responsabilidad	Función principal
Product Owner	Maximiza el valor del producto, define el objetivo y organiza los elementos del Product Backlog.
Scrum Master	Orienta al equipo en la aplicación de Scrum, facilita el proceso y ayuda a eliminar impedimentos.
Desarrolladores	Planifican y ejecutan el trabajo necesario para producir un incremento durante cada Sprint.

Nota. Elaboración propia con base en Schwaber y Sutherland

Tabla 4

Eventos del marco de trabajo Scrum.

Evento	Momento o duración	Propósito
Sprint	Ciclo de hasta un mes.	Generar un incremento útil y alcanzar el objetivo establecido.
Sprint Planning	Inicio del Sprint.	Definir el objetivo, el trabajo seleccionado y la forma de realizarlo.
Daily Scrum	Reunión diaria de 15 minutos.	Revisar el avance y ajustar el plan de trabajo.
Sprint Review	Final del Sprint.	Inspeccionar el resultado y recibir retroalimentación.
Sprint Retrospective	Después de la revisión.	Analizar el proceso e identificar mejoras para el siguiente ciclo.

Nota. Elaboración propia con base en Schwaber y Sutherland

Tabla 5

Artefactos y compromisos de Scrum.

Artefacto	Descripción	Compromiso asociado
Product Backlog	Lista ordenada y actualizable de las necesidades del producto.	Objetivo del producto.
Sprint Backlog	Trabajo seleccionado para el Sprint y plan para desarrollarlo.	Objetivo del Sprint.
Incremento	Resultado funcional acumulado durante el ciclo.	Definición de terminado.

Nota. Elaboración propia con base en Schwaber y Sutherland

Entre las ventajas de Scrum se encuentra la posibilidad de revisar el producto con frecuencia, recibir retroalimentación al terminar cada ciclo y modificar las prioridades conforme se obtiene nueva información. La división del trabajo en incrementos también

permite detectar dificultades antes de completar la totalidad del producto y mantener una comunicación periódica entre el equipo y las partes interesadas[23].

Su aplicación requiere participación constante, objetivos claros y disponibilidad para realizar los eventos establecidos. Cuando los requerimientos presentan poca variación y el trabajo depende de etapas que deben completarse en un orden previamente definido, puede resultar más apropiado un enfoque planificado que uno basado en ciclos continuos de inspección y adaptación[22]. Además, cuando se eliminan responsabilidades, eventos o artefactos esenciales, no se aplica Scrum en su totalidad, sino únicamente algunas de sus prácticas[23].

Por estas características, Scrum resulta adecuado para productos complejos cuyo alcance puede evolucionar durante el desarrollo y que requieren entregas frecuentes, revisión continua y adaptación de prioridades[23].

2.3.3. Extreme Programming (XP)

Extreme Programming, conocida por sus siglas XP, es una metodología ágil orientada específicamente al desarrollo de software. Fue impulsada principalmente por Kent Beck como una respuesta a proyectos en los que los requerimientos pueden cambiar y se necesita obtener retroalimentación continua. A diferencia de los modelos que concentran el análisis, el diseño, la programación y las pruebas en etapas separadas, XP propone realizar estas actividades de manera frecuente durante todo el desarrollo[24], [25].

Su funcionamiento se basa en ciclos cortos, entregas incrementales, comunicación constante y aplicación disciplinada de prácticas técnicas. El sistema se construye de forma progresiva, comenzando por las funciones de mayor prioridad y manteniendo la posibilidad de ajustar el alcance conforme aparecen nuevas necesidades. Las pruebas automatizadas, la

integración continua y la mejora permanente del código permiten comprobar los cambios y mantener una versión funcional durante el desarrollo[25].

XP se apoya en cinco valores que orientan las decisiones y la colaboración del equipo: comunicación, simplicidad, retroalimentación, valentía y respeto. Estos valores no representan actividades independientes, sino principios que se expresan mediante las prácticas utilizadas durante el desarrollo[24].

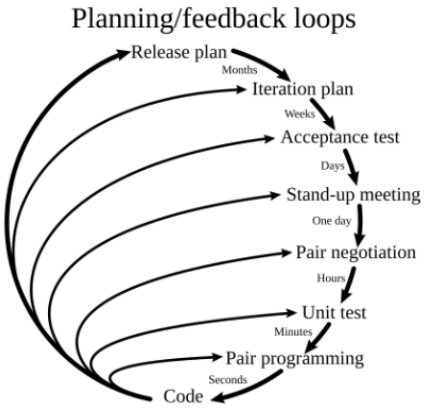
Organización del trabajo en XP

Las necesidades del producto suelen expresarse mediante historias de usuario breves que describen las funciones esperadas desde la perspectiva del cliente. Estas historias se priorizan y se seleccionan para formar entregas pequeñas e iteraciones de corta duración. Durante cada iteración se desarrollan, prueban e integran las funciones elegidas, y el resultado obtenido permite recibir retroalimentación antes de continuar con el siguiente conjunto de actividades[24].

La planificación no se considera definitiva desde el inicio. Se establece una dirección general, pero el contenido de las siguientes iteraciones puede modificarse según el avance, las prioridades y los resultados observados. Esta organización permite que el diseño evolucione junto con el software, en lugar de definir anticipadamente todas sus estructuras y comportamientos[25]. La Ilustración 26 representa estos ciclos de retroalimentación en distintas escalas de tiempo.

Ilustración 11

Ciclos de planificación y retroalimentación en Extreme Programming.



Nota. Wells, CC BY-SA 3.0.

Prácticas de Extreme Programming

La formulación clásica de XP reúne prácticas de planificación, programación, pruebas e integración que se complementan entre sí. Su aplicación conjunta busca mantener el código funcional, facilitar la colaboración y responder con rapidez a los cambios.

Tabla 6

Prácticas principales de Extreme Programming.

Práctica	Función
Juego de planificación	Permite priorizar las historias de usuario y determinar el trabajo que se incluirá en cada entrega.
Entregas pequeñas	Produce versiones funcionales en periodos breves para obtener retroalimentación temprana.
Diseño simple	Mantiene una estructura suficiente para resolver las necesidades actuales sin añadir complejidad innecesaria.
Pruebas frecuentes	Comprueba el comportamiento del código mediante pruebas automatizadas y criterios de aceptación.
Desarrollo guiado por pruebas	Plantea las pruebas antes o durante la programación para orientar la implementación de cada comportamiento.
Refactorización	Mejora la estructura interna del código sin modificar su comportamiento observable.
Programación en parejas	Dos desarrolladores trabajan conjuntamente sobre el mismo código, revisando y tomando decisiones de manera continua.
Propiedad colectiva del código	Permite que los integrantes autorizados modifiquen cualquier parte del código cuando sea necesario.
Integración continua	Incorpora los cambios al repositorio compartido con frecuencia y comprueba que el sistema continúe funcionando.
Ritmo sostenible	Busca mantener una carga de trabajo constante y evitar jornadas prolongadas que afecten la calidad.

Cliente disponible	Mantiene una comunicación cercana con una persona capaz de aclarar necesidades y prioridades.
Estándares de codificación	Establece criterios comunes para conservar un código uniforme, comprensible y mantenible.

Nota. Elaboración propia con base en Beck y Andrés

Las prácticas no funcionan únicamente como actividades aisladas. Las pruebas respaldan la refactorización, la integración continua permite detectar incompatibilidades y la programación en parejas favorece la revisión constante. De esta manera, cada práctica contribuye al funcionamiento de las demás y ayuda a conservar una versión operativa del software durante el desarrollo[24], [25].

Entre sus ventajas se encuentran la detección temprana de errores, la posibilidad de adaptar las prioridades y la obtención frecuente de versiones funcionales. La refactorización y las pruebas automatizadas también contribuyen a conservar la calidad interna del código, mientras que la programación en parejas y la propiedad colectiva favorecen la distribución del conocimiento entre los integrantes[25].

Su aplicación requiere una participación constante del equipo, disponibilidad para revisar las necesidades y disciplina para mantener las pruebas, la integración y la refactorización. La dependencia de una comunicación frecuente y de una documentación menos extensa puede presentar dificultades cuando el equipo se encuentra distribuido, cambia con frecuencia o necesita conservar registros formales detallados[22]. Además, prácticas como la programación en parejas, la integración continua y las entregas frecuentes pueden ser difíciles de sostener cuando no existe suficiente coordinación entre los participantes[24].

XP resulta especialmente útil en proyectos que necesitan aceptar cambios frecuentes y mantener una validación técnica continua. Sin embargo, su conveniencia depende de la

participación del equipo, de la posibilidad de automatizar las pruebas y de la disposición para aplicar sus prácticas de manera constante.

2.3.4. Cuadro comparativo de metodologías de desarrollo

Para contrastar los enfoques estudiados, se consideraron aspectos relacionados con la planificación, la organización del trabajo, la participación del equipo, la gestión de cambios, las entregas, las pruebas y la documentación. Esta comparación permite identificar las condiciones en las que cada alternativa resulta más conveniente y fundamentar la selección realizada para el proyecto.

Tabla 7

Comparación de enfoques para el desarrollo de software.

Criterio	Cascada	Scrum	XP
Enfoque de desarrollo	Secuencial, organizado mediante fases previamente definidas.	Iterativo e incremental, organizado mediante Sprints.	Iterativo e incremental, con énfasis en prácticas técnicas de programación.
Planificación	Se establece principalmente al inicio del proyecto.	Se actualiza al comienzo de cada Sprint según las prioridades del producto.	Se ajusta mediante historias de usuario, iteraciones cortas y entregas pequeñas.
Organización del trabajo	Las actividades avanzan desde los requerimientos hasta el diseño, implementación, pruebas y entrega.	El trabajo se distribuye mediante el Product Backlog y el Sprint Backlog.	El desarrollo combina planificación, programación, pruebas, integración y retroalimentación continua.
Gestión de cambios	Los cambios tardíos pueden requerir la revisión de fases ya completadas.	Las prioridades pueden modificarse entre Sprints según la retroalimentación recibida.	Los cambios se incorporan continuamente mediante iteraciones breves y mejora del código.
Entregas	El producto completo se obtiene después de finalizar las etapas planificadas.	Se genera un incremento funcional al finalizar cada Sprint.	Se producen versiones pequeñas y funcionales con frecuencia.
Responsabilidades	Se asignan según las etapas, actividades y funciones del equipo.	Establece las responsabilidades de Product Owner, Scrum Master y desarrolladores.	Promueve responsabilidad compartida, cliente disponible y propiedad colectiva del código.
Participación de las partes interesadas	Se concentra principalmente en la definición inicial, las	Participan periódicamente en la	Mantienen comunicación frecuente para aclarar historias,

	revisiones y la aceptación del producto.	revisión de los incrementos.	prioridades y criterios de aceptación.
Pruebas	Se desarrollan principalmente después de la implementación, aunque pueden realizarse verificaciones en cada fase.	Las pruebas forman parte del trabajo necesario para completar cada incremento.	Constituyen una práctica central y se realizan continuamente junto con la programación.
Prácticas técnicas	No establece prácticas de programación específicas.	No prescribe técnicas concretas de codificación o diseño.	Incluye pruebas automatizadas, refactorización, programación en parejas e integración continua.
Documentación	Mantiene documentación definida para los requerimientos, el diseño, la implementación y las pruebas.	Prioriza artefactos suficientes para transparentar y administrar el trabajo.	Favorece la comunicación directa y el código funcional sobre la documentación extensa.
Adaptabilidad	Adecuada cuando el alcance y los requerimientos presentan estabilidad.	Adecuada cuando las prioridades pueden cambiar durante el desarrollo.	Adecuada cuando existen cambios frecuentes y se requiere validación técnica continua.
Fortaleza principal	Facilita la planificación, la documentación y el seguimiento ordenado de las fases.	Permite inspeccionar el producto regularmente y adaptar las prioridades.	Mantiene una revisión constante de la calidad interna y del funcionamiento del código.
Aspecto por considerar	Los cambios realizados en etapas avanzadas pueden afectar el trabajo previamente completado.	Requiere participación constante y aplicación formal de responsabilidades, eventos y artefactos.	Exige disciplina para mantener pruebas, integración, refactorización y colaboración continua.

Nota. Elaboración propia con base en Royce, Schwaber y Sutherland, Beck y Andrés

En virtud de la revisión y comparación de las metodologías analizadas, se determinó que el enfoque en cascada se ajustaba mejor a las características del proyecto. Su estructura secuencial resultó adecuada para un trabajo académico con un alcance previamente delimitado y entregables definidos en cada etapa[20], [22]. La propuesta requería analizar las necesidades educativas, establecer la organización del sistema, preparar el contenido didáctico, implementar las funcionalidades, comprobar su funcionamiento y realizar la validación correspondiente.

2.4. Bases tecnológicas

En este apartado se analizan las herramientas y tecnologías relacionadas con la programación y la organización técnica del proyecto. La revisión se divide en tres grupos: motores de desarrollo de videojuegos, lenguajes de programación y sistemas de control de versiones con plataformas de repositorios. En cada grupo se estudian distintas alternativas y se comparan sus características, funcionamiento, compatibilidad, ventajas y limitaciones, con el propósito de determinar cuáles se ajustan mejor a las necesidades del sistema educativo interactivo.

2.4.1. Motores de desarrollo de videojuegos

Los motores de desarrollo proporcionan las funciones necesarias para crear escenas, integrar recursos, programar comportamientos y controlar la ejecución de un videojuego o aplicación interactiva. En esta sección se investigan Unity, Unreal Engine y Godot Engine, considerando aspectos como su arquitectura de trabajo, lenguajes compatibles, capacidades para proyectos 2D y 3D, requisitos técnicos, licencias, documentación y facilidad de aprendizaje. Finalmente, se presenta un cuadro comparativo que permite justificar la selección del motor utilizado en el proyecto.

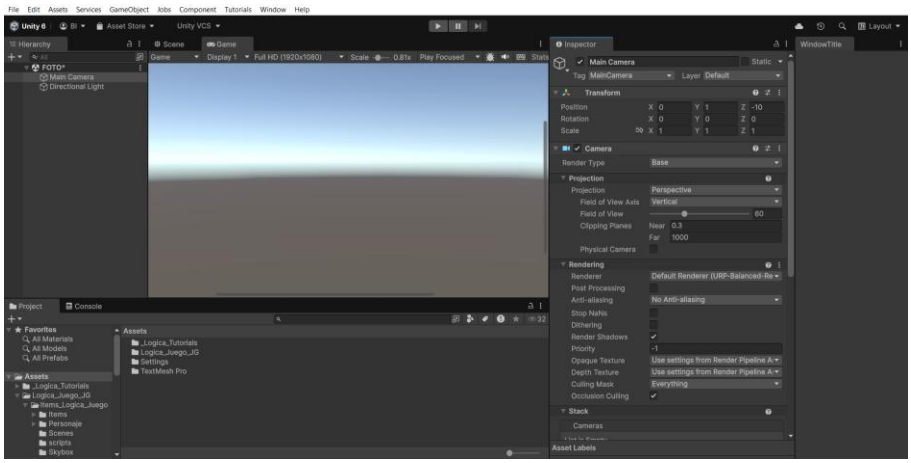
2.4.1.1. Unity

Unity es un motor de desarrollo utilizado para crear videojuegos y aplicaciones interactivas en dos y tres dimensiones. Su entorno reúne herramientas para construir escenas, integrar recursos gráficos y sonoros, programar comportamientos, configurar sistemas físicos y desarrollar interfaces de usuario. También permite generar compilaciones para distintas plataformas, como Windows, Linux, dispositivos móviles, navegadores, consolas y equipos de realidad extendida[26].

El Editor de Unity organiza el proceso de desarrollo mediante varias ventanas. Scene permite construir y modificar el entorno; Game muestra el resultado durante la ejecución; Hierarchy contiene los objetos presentes en la escena; Project organiza los archivos y recursos; e Inspector permite consultar y cambiar las propiedades del elemento seleccionado. Estas ventanas pueden distribuirse de acuerdo con las necesidades del desarrollador[27]. La Ilustración 10 presenta esta distribución aplicada a una escena del proyecto, con la jerarquía a la izquierda, el área de trabajo al centro, los recursos en la parte inferior y el Inspector a la derecha.

Ilustración 12

Ventanas principales del Editor de Unity.



Nota. Elaboración propia con base de Unity

La estructura de trabajo del motor se basa en escenas, objetos y componentes. Los elementos contenidos en una escena se representan mediante GameObjects, los cuales funcionan como contenedores a los que se agregan componentes para definir su apariencia y comportamiento. Por ejemplo, un objeto puede incorporar componentes de representación gráfica, sonido, colisión, física y programación. Todos los GameObjects poseen un componente Transform, que almacena su posición, rotación y escala[28].

Esta organización permite separar las responsabilidades de un objeto entre varios componentes. Un personaje puede contener un elemento visual, un colisionador, un cuerpo físico y uno o varios scripts encargados de controlar sus acciones. El Inspector muestra los componentes vinculados y permite modificar sus propiedades sin alterar directamente el código, lo que facilita la configuración y las pruebas durante el desarrollo[28].

Para la programación de comportamientos, Unity ofrece soporte nativo para el lenguaje C#. Los scripts permiten recibir entradas, realizar cálculos, controlar objetos, establecer reglas y desarrollar mecánicas. Una clase que hereda de MonoBehaviour puede incorporarse a un GameObject como un componente personalizado, conectando el código con los elementos y propiedades gestionados desde el Editor[29].

Unity dispone de herramientas específicas para proyectos 2D y 3D. En el desarrollo bidimensional incluye funciones para trabajar con sprites, animaciones, escenarios, iluminación y físicas. El motor mantiene sistemas físicos diferenciados para objetos 2D y 3D, lo que permite configurar cuerpos, colisiones, gravedad y movimiento según las características de cada proyecto[30].

Otro aspecto relevante es la disponibilidad de documentación y recursos educativos. La plataforma Unity Learn ofrece tutoriales, proyectos y recorridos guiados para distintos niveles de experiencia. Sus contenidos abarcan el uso del Editor, programación, creación de videojuegos y desarrollo de experiencias 2D y 3D, lo que facilita el aprendizaje progresivo de las herramientas del motor[31].

Unity cuenta con diferentes planes de licencia. El plan Unity Personal puede utilizarse de manera gratuita por personas y organizaciones que no superen el límite financiero establecido en sus condiciones vigentes, actualmente fijado en 200.000 dólares estadounidenses durante los doce meses anteriores[32]. Debido a que las condiciones

comerciales pueden cambiar, deben revisarse en la documentación oficial antes de iniciar o publicar un proyecto.

En cuanto a los requisitos técnicos, la documentación de Unity 6 recomienda un mínimo de 8 GB de memoria RAM para ejecutar el Editor, aunque el consumo final depende de la complejidad del proyecto y de los paquetes incorporados[33].

En conjunto, Unity ofrece una arquitectura basada en componentes, programación mediante C#, herramientas para proyectos 2D y 3D, publicación multiplataforma y una amplia disponibilidad de recursos de aprendizaje. Entre sus limitaciones se encuentran el consumo de recursos del Editor y la necesidad de revisar la compatibilidad de los paquetes y las condiciones de licencia entre versiones[33], [34].

2.4.1.2. Godot Engine

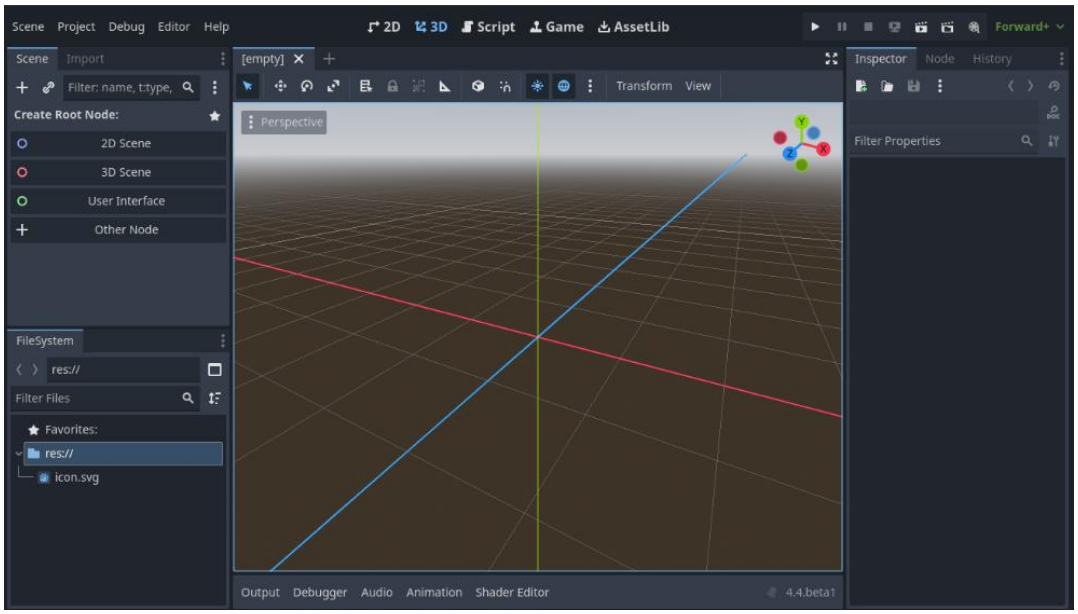
Godot Engine es un motor de desarrollo gratuito y de código abierto orientado a la creación de videojuegos y aplicaciones interactivas en dos y tres dimensiones. El proyecto es mantenido por una comunidad de desarrolladores y proporciona herramientas para construir escenas, programar comportamientos, trabajar con gráficos, sonido, animaciones, físicas e interfaces de usuario desde un entorno unificado[35].

El Editor de Godot se organiza mediante áreas de trabajo para contenidos 2D, 3D, programación, ejecución del juego y acceso a recursos. La interfaz también incorpora el panel Scene, donde se muestra la estructura de la escena activa; FileSystem, que contiene los archivos del proyecto; e Inspector, desde el cual se modifican las propiedades del elemento seleccionado. En la parte inferior se encuentran herramientas como la consola, el depurador, el editor de animaciones y el mezclador de audio[36]. La Ilustración 11 presenta esta

disposición con el panel de escena a la izquierda, el espacio de trabajo al centro, el sistema de archivos en la parte inferior izquierda y el Inspector a la derecha.

Ilustración 13

Áreas principales del Editor de Godot Engine.



Nota. Godot Engine

La organización interna del motor se basa en nodos y escenas. Los nodos son los elementos fundamentales del proyecto y pueden cumplir funciones específicas, como mostrar una imagen, reproducir sonido, representar una cámara, detectar colisiones o controlar un personaje. Cada nodo posee un nombre, propiedades editables y la posibilidad de recibir funciones o comportamientos mediante programación[37].

Los nodos pueden agruparse en forma de árbol para construir elementos más complejos. Por ejemplo, un personaje puede combinar un nodo encargado del movimiento, otro para mostrar su imagen, uno para la cámara y otro para representar su área de colisión. Cuando este conjunto se guarda, se convierte en una escena que puede reutilizarse e incorporarse varias veces dentro de otras escenas. Esta estructura favorece la composición y reutilización de elementos durante el desarrollo[37].

Godot admite distintos lenguajes para programar la lógica de los videojuegos. Entre las opciones oficiales se encuentran GDScript, C# y, mediante la tecnología GDExtension, C y C++. GDScript fue creado específicamente para el motor y utiliza una sintaxis ligera integrada directamente con su estructura de nodos y escenas. También es posible combinar más de un lenguaje dentro de un mismo proyecto, según las necesidades técnicas de cada componente[38].

La documentación recomienda GDScript como punto de inicio para usuarios principiantes, debido a su integración con el motor y a su sintaxis directa. El soporte para C# requiere utilizar la versión del Editor preparada para .NET[12] y normalmente un editor externo, como Visual Studio o Visual Studio Code. Aunque este soporte se considera estable, existe una cantidad menor de recursos educativos para C# en comparación con GDScript[38].

Para el desarrollo visual, Godot dispone de herramientas y nodos diferenciados para proyectos 2D y 3D. El área 2D también se utiliza para construir interfaces, mientras que el espacio 3D permite trabajar con modelos, iluminación, cámaras y escenarios tridimensionales. El Editor puede ejecutarse en Windows, macOS y Linux, mientras que los proyectos pueden publicarse para plataformas de escritorio, dispositivos móviles y navegadores. La publicación en consolas requiere procesos y conocimientos adicionales[35], [36].

Godot se distribuye bajo la licencia permisiva MIT. Esta licencia permite usar, estudiar, modificar y distribuir el motor, incluso en proyectos comerciales. Los proyectos creados con Godot pueden publicarse bajo una licencia diferente; sin embargo, cuando se

[12] .Net: plataforma utilizada para desarrollar y ejecutar aplicaciones con C#.

distribuye el motor o una parte sustancial de este junto con el proyecto, debe incluirse el texto correspondiente a la licencia MIT[39].

En cuanto a los requisitos técnicos mínimos para computadoras, la documentación establece 4 GB de memoria RAM para el Editor nativo y aproximadamente 200 MB de almacenamiento para el ejecutable, los archivos del proyecto y la caché. Las plantillas necesarias para exportar proyectos requieren espacio adicional. Los requisitos gráficos varían según el sistema de renderizado seleccionado y la complejidad de la aplicación[40].

En conjunto, Godot Engine ofrece una estructura basada en nodos y escenas, compatibilidad con varios lenguajes de programación, herramientas para proyectos 2D y 3D y una licencia abierta que permite modificar el propio motor. Entre los aspectos que deben considerarse se encuentran las diferencias de soporte y documentación entre sus lenguajes, así como la necesidad de seleccionar correctamente la versión del Editor y el sistema de renderizado[38], [40].

2.4.1.3. Unreal Engine

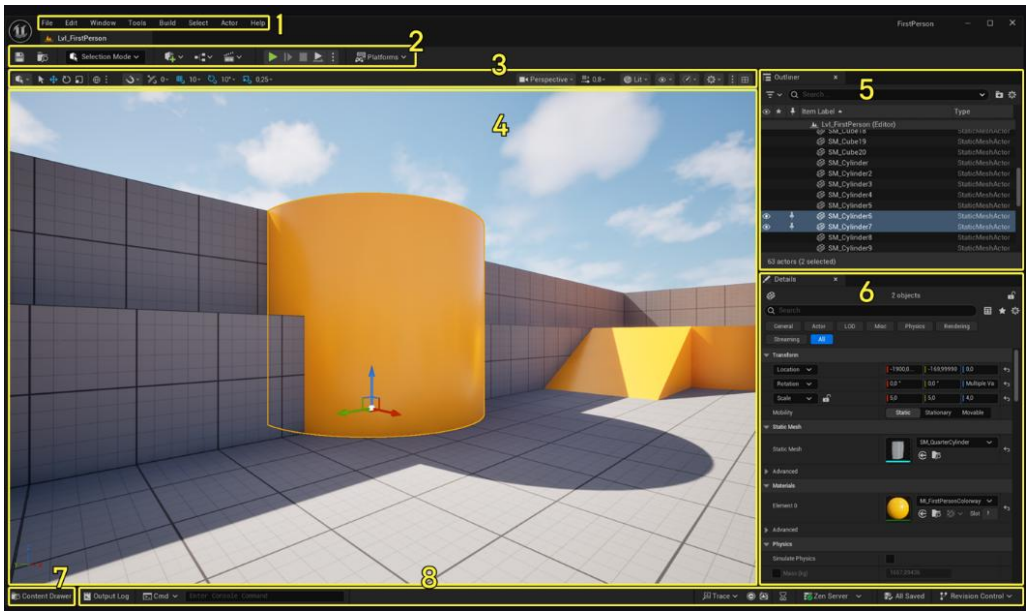
Unreal Engine es un motor de desarrollo creado por Epic Games para la producción de videojuegos y experiencias interactivas en tiempo real. Su entorno integra herramientas para construir niveles, programar comportamientos, trabajar con animaciones, físicas, interfaces, efectos visuales y contenido audiovisual. Aunque destaca por sus capacidades para proyectos tridimensionales de alta calidad gráfica, también dispone de herramientas para el desarrollo de experiencias en distintas plataformas[41].

El Editor de Unreal Engine organiza el trabajo mediante diferentes paneles. El Viewport permite visualizar y modificar el nivel; el Outliner presenta los objetos incluidos en la escena; el Details Panel muestra las propiedades del elemento seleccionado; el Content

Drawer organiza los archivos y recursos del proyecto; y la barra de herramientas reúne opciones para guardar, compilar, ejecutar y configurar la aplicación. La distribución de estas áreas puede modificarse según las necesidades del desarrollador[41]. La Ilustración 12 identifica cada zona mediante una numeración, lo que facilita reconocer la ubicación de los menús, la barra de herramientas, el Viewport, el Outliner, el panel de detalles y los accesos inferiores del Editor.

Ilustración 14

Áreas principales del Editor de Unreal Engine.



Nota. Unreal Engine

La estructura de Unreal Engine se basa en niveles, Actors y componentes. Un Actor es cualquier objeto que puede colocarse o generarse dentro de un nivel, como una cámara, una fuente de iluminación, un personaje o un elemento del escenario. Los Actors pueden incorporar componentes que agregan funciones concretas, por ejemplo, una representación gráfica, colisiones, sonido, movimiento o lógica relacionada con la interacción[42].

Los componentes permiten dividir el comportamiento de un objeto en responsabilidades más específicas. Un personaje puede estar formado por un componente

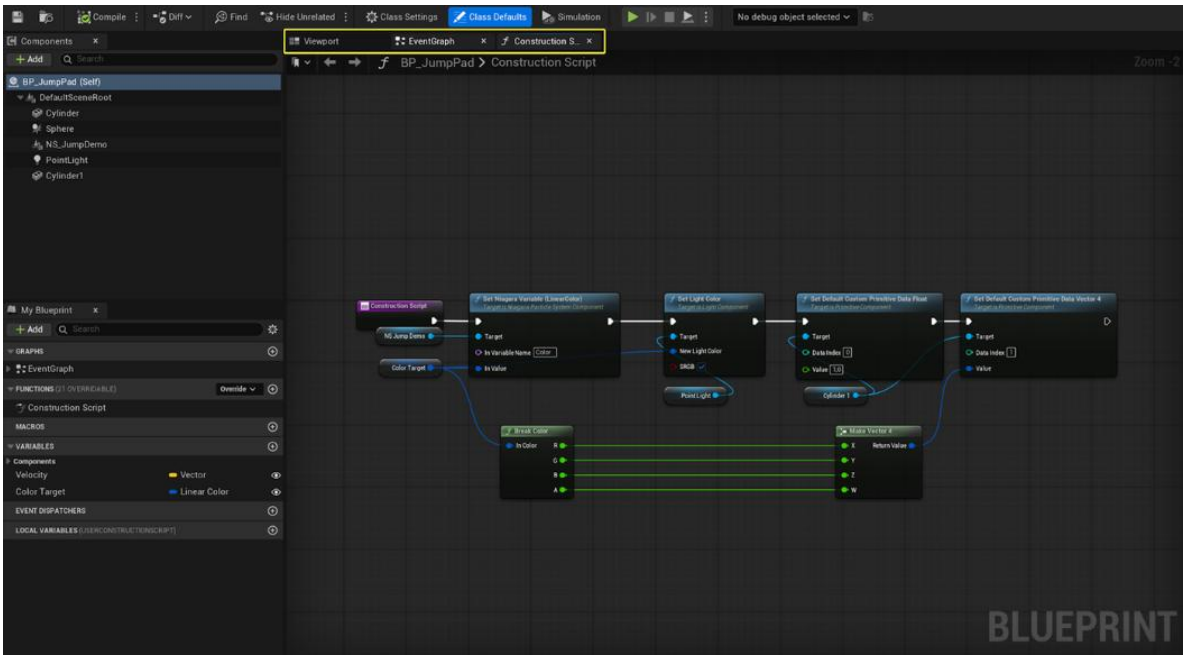
visual, una cápsula de colisión, una cámara y un componente encargado del movimiento. Los Actors pueden incorporar componentes que agregan funciones concretas, por ejemplo, una representación gráfica, colisiones, sonido, movimiento o lógica relacionada con la interacción[42].

Unreal Engine permite desarrollar la lógica mediante C++ y mediante su sistema de programación visual Blueprints. C++ se utiliza para crear clases, sistemas y comportamientos con un mayor control sobre el código. Blueprints, en cambio, permite construir la lógica mediante nodos conectados dentro de un editor gráfico. Este sistema puede manejar variables, eventos, condiciones, funciones y flujo de ejecución sin exigir que toda la funcionalidad sea escrita directamente en código[43].

Ambas formas de programación pueden utilizarse de manera conjunta. Los programadores pueden crear sistemas base mediante C++ y exponer determinadas propiedades y funciones para que sean ampliadas mediante Blueprints. Esta integración permite relacionar la arquitectura desarrollada mediante código con la configuración visual de los comportamientos dentro del Editor[43]. La Ilustración 13 presenta un ejemplo de Blueprint, donde los nodos y sus conexiones representan la secuencia de operaciones configuradas visualmente.

Ilustración 15

Programación visual mediante Blueprints en Unreal Engine.



Nota. Unreal Engine

Para proyectos bidimensionales, Unreal Engine incluye Paper 2D, un sistema basado en sprites que permite desarrollar juegos 2D y experiencias híbridas que combinan elementos bidimensionales y tridimensionales. Este sistema incorpora sprites, animaciones mediante Flipbooks, conjuntos de mosaicos y mapas de mosaicos para construir personajes, objetos y escenarios[44].

En cuanto a los requisitos técnicos, la documentación oficial recomienda para Unreal Engine 5 un procesador Intel o AMD de cuatro núcleos y 2,5 GHz o superior, 32 GB de memoria RAM y al menos 8 GB de memoria gráfica. Estos valores corresponden a la configuración recomendada para el Editor y pueden variar según las funciones de renderizado utilizadas, la compilación del código y la complejidad de los recursos incorporados[45].

Unreal Engine puede utilizarse gratuitamente durante el desarrollo de videojuegos. De acuerdo con el modelo de licencia vigente, los productos que incorporan el motor durante

su ejecución deben pagar una regalía del 5% sobre los ingresos brutos acumulados que superen el primer millón de dólares estadounidenses. Las condiciones pueden variar según el tipo de producto, la forma de distribución y los acuerdos establecidos por Epic Games, por lo que deben revisarse antes de publicar una aplicación comercial[46].

En conjunto, Unreal Engine ofrece herramientas para la creación de entornos, programación mediante C++ y Blueprints, organización basada en Actors y componentes, y desarrollo de experiencias 2D y 3D. Entre los aspectos que deben considerarse se encuentran sus requisitos técnicos y la necesidad de comprender la relación entre el código, la programación visual y los componentes utilizados dentro del nivel[42].

2.4.1.4. Cuadro comparativo de motores de desarrollo

Después de revisar las características de Unity, Godot Engine y Unreal Engine, se establecieron criterios comunes para comparar sus posibilidades de desarrollo. Se consideraron la arquitectura de trabajo, los lenguajes compatibles, las herramientas para proyectos 2D y 3D, la publicación en diferentes plataformas, el modelo de licencia, los requisitos técnicos y los recursos disponibles para el aprendizaje[31], [35], [41]. Estos parámetros permiten reconocer las diferencias entre los motores y determinar cuál responde mejor a las necesidades técnicas y educativas del proyecto.

Tabla 8

Comparación de motores de desarrollo de videojuegos.

Criterio	Unity	Godot Engine	Unreal Engine
Tipo de licencia	Software propietario. El plan Personal es gratuito cuando no se supera el límite financiero establecido por Unity.	Software gratuito y de código abierto distribuido bajo licencia MIT.	Software propietario. Su uso para videojuegos se basa en un modelo de regalías cuando el producto supera el límite de ingresos establecido.
Arquitectura principal	Escenas formadas por GameObjects y componentes.	Escenas formadas por árboles de nodos.	Niveles compuestos por Actors y componentes.

Lenguajes principales	C#.	GDScript y C#; permite extensiones mediante C y C++.	C++ y programación visual mediante Blueprints.
Programación visual	Puede incorporarse mediante paquetes como Visual Scripting, pero no constituye su forma principal de programación.	Dispone de herramientas visuales para sistemas específicos, aunque la lógica se desarrolla principalmente mediante scripts.	Blueprints forma parte central del motor y permite programar comportamientos mediante nodos.
Desarrollo 2D	Incluye herramientas específicas para sprites, animaciones, físicas y construcción de escenarios 2D.	Posee un entorno 2D independiente con nodos, físicas y herramientas orientadas a proyectos bidimensionales.	Utiliza el complemento Paper 2D para proyectos 2D y experiencias híbridas 2D/3D.
Desarrollo 3D	Integra herramientas para escenas, iluminación, físicas, animación y renderizado tridimensional.	Permite crear proyectos 3D mediante nodos, cámaras, iluminación, físicas y distintos sistemas de renderizado.	Presenta herramientas avanzadas de renderizado, iluminación, animación y creación de entornos tridimensionales.
Publicación multiplataforma	Permite generar aplicaciones para computadoras, dispositivos móviles, web, consolas y realidad extendida.	Exporta a sistemas de escritorio, dispositivos móviles y web; algunas plataformas pueden requerir configuraciones o herramientas adicionales.	Permite publicar en computadoras, dispositivos móviles, consolas y sistemas de realidad extendida.
Recursos de aprendizaje	Cuenta con documentación oficial, Unity Learn, tutoriales y una comunidad amplia.	Dispone de documentación oficial, proyectos de ejemplo y recursos creados por su comunidad.	Ofrece documentación, cursos, ejemplos y recursos educativos mediante Epic Developer Community.
Requisitos técnicos	Unity 6 recomienda un mínimo de 8 GB de memoria RAM; el consumo aumenta según la complejidad del proyecto.	Puede ejecutarse con requisitos más reducidos; la documentación establece especificaciones mínimas para proyectos sencillos 2D y 3D.	Recomienda 32 GB de memoria RAM y 8 GB o más de memoria gráfica para el Editor.
Fortaleza principal	Equilibrio entre programación con C#, herramientas 2D y 3D, organización por componentes y disponibilidad de recursos educativos.	Código abierto, licencia permisiva, tamaño reducido y estructura flexible basada en nodos.	Alta capacidad gráfica e integración entre C++ y programación visual mediante Blueprints.
Aspecto por considerar	El rendimiento depende de la configuración, los paquetes utilizados y la complejidad del proyecto.	La disponibilidad de materiales, extensiones y soporte puede variar según el lenguaje o la plataforma utilizada.	Presenta requisitos de hardware elevados y una mayor complejidad inicial por la cantidad de herramientas incorporadas.

Nota. Unity, Godot y Unreal

2.4.2. Lenguajes de programación

Los lenguajes de programación permiten definir la lógica, las reglas y los comportamientos que intervienen en el funcionamiento de un videojuego o sistema interactivo. Su elección depende del motor de desarrollo, del tipo de proyecto y del nivel de control requerido durante la programación. En esta sección se analizan C++, C# y GDScript, por ser los lenguajes asociados principalmente con Unreal Engine, Unity y Godot Engine. La revisión considera su forma de uso, integración con cada motor, facilidad de aprendizaje, rendimiento y aplicación en la creación de mecánicas. Finalmente, se presenta un cuadro comparativo para determinar cuál se ajusta mejor a las necesidades del proyecto.

2.4.2.1. C++

C++ es un lenguaje de programación de propósito general y tipado estático, utilizado para desarrollar sistemas que requieren control sobre la estructura del programa y los recursos empleados durante su ejecución. No se limita a un único paradigma, ya que admite programación imperativa, orientada a objetos y genérica. Mediante clases, herencia y funciones virtuales puede organizar sistemas orientados a objetos, mientras que las plantillas permiten definir estructuras y algoritmos genéricos que el compilador instancia para los tipos utilizados[47].

La programación orientada a objetos en C++ permite reunir datos y funciones dentro de clases. Estas pueden representar elementos de un videojuego, como personajes, objetos, controladores o sistemas de interacción. La herencia facilita la creación de clases especializadas a partir de comportamientos comunes, mientras que el polimorfismo permite ejecutar diferentes implementaciones mediante una misma interfaz. Sin embargo, el propio lenguaje no obliga a organizar todos los programas de esta manera, por lo que también

pueden emplearse funciones, estructuras simples y programación genérica según las necesidades del sistema[47].

C++ también proporciona mecanismos para controlar la duración de los objetos y la administración de los recursos. Los objetos con duración automática se destruyen cuando finaliza el bloque en el que fueron creados, mientras que la memoria dinámica puede reservarse durante la ejecución. Los constructores y destructores permiten relacionar la creación y destrucción de un objeto con la adquisición y liberación de recursos. Este principio, conocido como RAII, ofrece un control amplio sobre el ciclo de vida de los recursos, pero requiere mantener una organización cuidadosa para evitar errores relacionados con la memoria y las referencias a objetos[47].

En Unreal Engine, C++ se utiliza para definir clases de jugabilidad que heredan de clases base proporcionadas por el motor, como AActor. Estas clases pueden contener componentes, variables y funciones encargadas de establecer el comportamiento de los objetos durante la ejecución. Una vez compiladas, quedan disponibles dentro del Editor y pueden incorporarse a los niveles del proyecto[48]. La Ilustración 14 presenta la declaración de la clase AFloatingActor, en la que se observan su herencia de AActor, el componente visual y los métodos utilizados para inicializar y actualizar su comportamiento.

Ilustración 16

Estructura de una clase programada en C++ para Unreal Engine.

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "FloatingActor.generated.h"

UCLASS()
class QUICKSTART_API AFloatingActor : public AActor
{
  GENERATED_BODY()

public:
  // Sets default values for this actor's properties
  AFloatingActor();

  UPROPERTY(VisibleAnywhere)
  UStaticMeshComponent* VisualMesh;

protected:
  // Called when the game starts or when spawned
  virtual void BeginPlay() override;
```

Nota. Epic Games

La integración de C++ con Unreal Engine incorpora macros como UCLASS(), UFUNCTION() y UPROPERTY(). Estas permiten que el motor reconozca las clases, funciones y variables programadas, y que determinados elementos puedan mostrarse o modificarse desde el Editor. De esta manera, la programación textual puede conectarse con actores, componentes, propiedades y otras funciones administradas por el entorno de desarrollo[49].

C++ también puede combinarse con Blueprints. Los sistemas principales pueden programarse mediante clases de C++, mientras que ciertas propiedades y funciones pueden exponerse para su utilización en gráficos visuales. Esta combinación permite conservar una base programada y, al mismo tiempo, configurar comportamientos o valores desde el Editor sin modificar continuamente el código fuente[49].

Entre las características de C++ se encuentran el control sobre los recursos, la programación mediante clases, el uso de plantillas y su integración con sistemas de bajo nivel. Estas posibilidades permiten desarrollar estructuras complejas y componentes

reutilizables[47]. Como aspectos por considerar, el lenguaje exige comprender la compilación y la administración del ciclo de vida de los objetos, elementos que pueden representar una dificultad inicial para usuarios con poca experiencia en programación.

2.4.2.2. C#

C# es un lenguaje de programación de propósito general, multiplataforma y de tipado estático, desarrollado para la plataforma .NET. Su estructura se basa principalmente en la programación orientada a objetos, aunque también incorpora características de otros enfoques, como la programación funcional y genérica. El tipado estático permite que el compilador compruebe la compatibilidad entre variables, parámetros y valores antes de ejecutar el programa, lo que ayuda a identificar determinados errores durante la etapa de desarrollo[50].

El lenguaje permite organizar la información mediante clases, estructuras, registros e interfaces. Las clases pueden reunir datos y comportamientos, aplicar encapsulación y derivarse de otras clases mediante herencia. Las interfaces definen contratos que distintas clases o estructuras pueden implementar, mientras que los tipos genéricos permiten reutilizar algoritmos y colecciones con diferentes tipos de datos. Estas características facilitan la separación de responsabilidades y la construcción de sistemas formados por componentes relacionados[50].

C# utiliza administración automática de memoria mediante un recolector de basura. Este sistema identifica los objetos que ya no poseen referencias y recupera el espacio que ocupaban, por lo que el desarrollador no necesita liberar manualmente cada objeto creado. Sin embargo, en aplicaciones que se ejecutan en tiempo real es necesario controlar la creación

frecuente de objetos temporales, debido a que un aumento de asignaciones puede incrementar el trabajo del recolector y afectar el rendimiento durante la ejecución[51].

Dentro de Unity, los archivos escritos en C# se utilizan para crear componentes personalizados, controlar objetos y programar la lógica de las mecánicas. Una clase que hereda de MonoBehaviour puede agregarse a un GameObject, de forma similar a los componentes incluidos por el propio motor. Desde estos scripts es posible responder a entradas del usuario, modificar propiedades, ejecutar cálculos, activar eventos y controlar cambios durante la ejecución. La Ilustración 15 presenta la estructura inicial de un script PlayerMovement, donde la herencia de MonoBehaviour permite utilizar métodos del ciclo de ejecución de Unity, como Start y Update[29].

Ilustración 17

Creación de un script de movimiento mediante C# en Unity.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

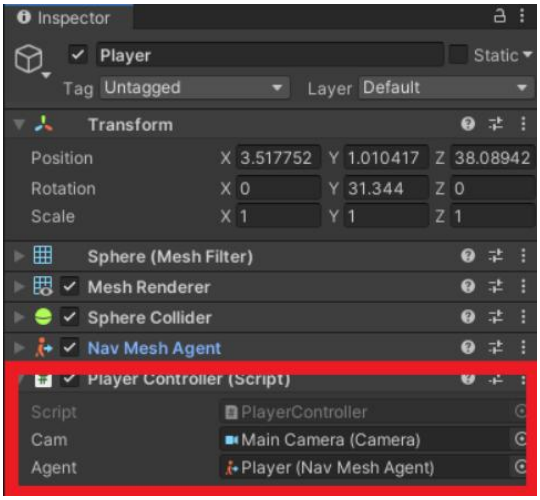
Nota. Unity Learn

Los campos públicos y los campos privados marcados con el atributo [SerializeField] pueden mostrarse en el Inspector. Esto permite modificar parámetros, asignar referencias y probar diferentes valores sin editar directamente el archivo de código. De esta manera, el

script mantiene la lógica programada mientras determinadas propiedades permanecen disponibles para su configuración desde el Editor. La Ilustración 16 ejemplifica esta configuración mediante el componente Player Controller, cuyas referencias a la cámara y al agente de navegación se asignan desde el Inspector[52].

Ilustración 18

Configuración de las variables de un script C# desde el Inspector de Unity.



Nota. Unity Learn

Unity compila los scripts C# incluidos en el proyecto y utiliza un backend de scripting para convertirlos en instrucciones ejecutables. Dependiendo de la plataforma de destino y de la configuración del proyecto, el motor puede utilizar Mono o IL2CPP. Mono utiliza compilación en tiempo de ejecución, mientras que IL2CPP convierte el código y los ensamblados de C# en código C++, que posteriormente se compila mediante las herramientas nativas de la plataforma para producir el archivo ejecutable[43].

Entre las características relevantes de C# se encuentran su tipado estático, la organización mediante clases e interfaces, la administración automática de memoria y su integración directa con Unity. Estas propiedades facilitan la creación de comportamientos reutilizables y la conexión entre el código y los objetos del Editor[50]. Como aspecto que

debe considerarse, la ejecución en tiempo real requiere controlar las asignaciones frecuentes de memoria para reducir el trabajo del recolector de basura[51].

2.4.2.3. GDScript

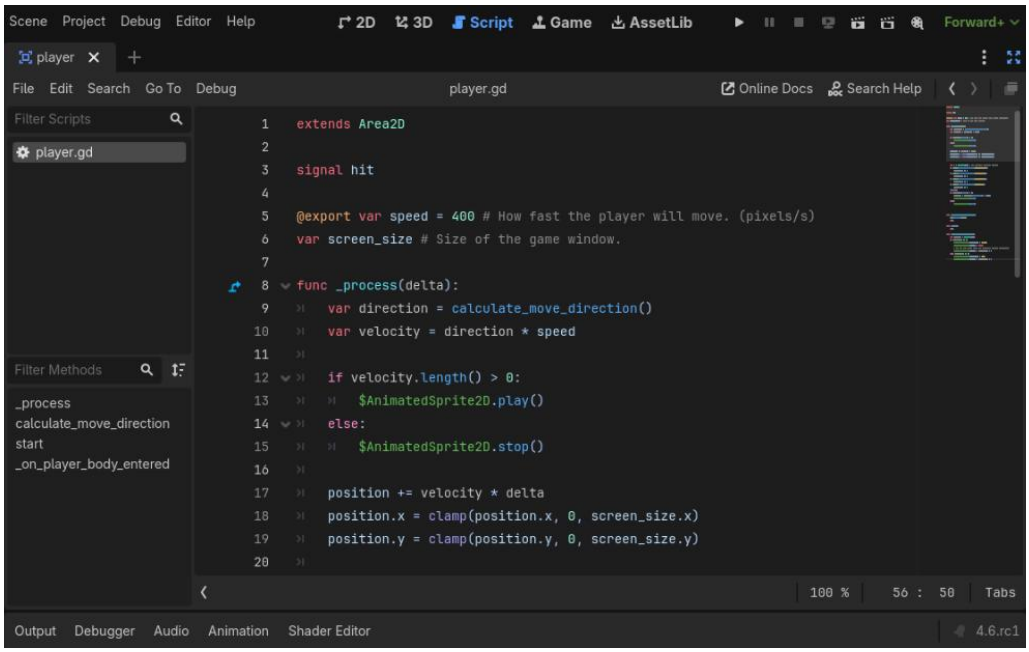
GDScript es un lenguaje de programación de alto nivel creado específicamente para Godot Engine. Se caracteriza por seguir un enfoque orientado a objetos e imperativo, utilizar una sintaxis basada en sangrías y permitir tipado gradual. Esto significa que el desarrollador puede escribir variables sin declarar un tipo fijo o añadir tipos explícitos cuando requiere una estructura más controlada. Su diseño busca mantener una integración directa con los nodos, las escenas y las funciones propias del motor[53].

Los archivos desarrollados en este lenguaje utilizan la extensión .gd. Al asociar un script con un nodo, este amplía el comportamiento del elemento y puede acceder a sus propiedades y métodos. Por ejemplo, un script conectado a un personaje puede procesar las entradas del usuario, modificar su posición, controlar animaciones o responder a colisiones. Esta relación directa entre el código y los nodos forma parte de la estructura de programación utilizada por Godot[38], [53].

Godot incorpora un editor de scripts integrado que permite escribir y depurar GDScript sin utilizar obligatoriamente un programa externo. Entre sus funciones se encuentran el resaltado de sintaxis, la indentación automática, la comprobación de errores, el autocompletado, los puntos de interrupción y el acceso a la referencia de clases. Estas herramientas permiten mantener la programación y la configuración de las escenas dentro del mismo entorno de desarrollo[54]. La Ilustración 17 presenta este espacio de programación con el archivo player.gd abierto y los métodos del script organizados en el panel lateral.

Ilustración 19

Ejemplo de programación mediante GDScript en Godot Engine.



Nota. Godot Engine

GDScript permite utilizar tipado estático en variables, constantes, parámetros, funciones y valores de retorno. Por ejemplo, una variable puede declararse mediante var speed: float, mientras que una función puede especificar el tipo de dato que recibe y el que devuelve. El uso de tipos ayuda a detectar errores antes de ejecutar el proyecto, mejora el autocompletado del Editor y facilita la lectura del código. La documentación también indica que el código tipado puede utilizar operaciones optimizadas cuando los tipos se conocen durante la compilación[55].

El lenguaje dispone de clases, funciones, arreglos, diccionarios, enumeraciones y señales. La palabra clave extends permite establecer el nodo o la clase que amplía el script, mientras que class_name registra una clase personalizada para que pueda utilizarse desde otras partes del proyecto. De esta forma, pueden crearse comportamientos reutilizables y organizar el código de acuerdo con las responsabilidades de los objetos[53].

Otra característica es la anotación @export, utilizada para mostrar variables del script en el Inspector. Estas propiedades pueden modificarse desde el Editor sin intervenir directamente en el código. Godot también dispone de anotaciones para establecer rangos, listas de opciones, categorías y grupos de propiedades, lo que facilita la configuración de parámetros relacionados con el movimiento, la velocidad o la duración de una acción[56].

GDScript fue diseñado principalmente para programar la lógica de los videojuegos dentro de Godot. Su sintaxis ligera y su integración con el motor favorecen la creación y modificación de comportamientos asociados con los nodos. La documentación oficial lo recomienda como lenguaje inicial para quienes comienzan a trabajar con Godot, aunque el motor también permite utilizar C# y extensiones desarrolladas mediante C o C++[38].

Entre sus características se encuentran la integración directa con el Editor, el uso opcional de tipado estático, la exposición de propiedades en el Inspector y una sintaxis con pocos elementos adicionales[53]. Como aspecto por considerar, su uso se encuentra estrechamente vinculado con Godot Engine y dispone de un ecosistema más específico que lenguajes de propósito general como C# o C++[38].

2.4.2.4. Cuadro comparativo de lenguajes de programación

Después de revisar C++, C# y GDScript, se establecieron criterios comunes para comparar sus características y su relación con los motores investigados. Se consideraron el sistema de tipado, la organización del código, la administración de memoria, la integración con el motor, la facilidad de aprendizaje y el nivel de control que ofrecen durante la programación de mecánicas[47]. La comparación permite identificar las diferencias entre los lenguajes y justificar cuál responde mejor a las necesidades del proyecto.

Tabla 9

Comparación de lenguajes de programación utilizados en motores de videojuegos.

Criterio	C++	C#	GDScript
Motor asociado	Unreal Engine.	Unity.	Godot Engine.
Propósito general	Lenguaje de propósito general orientado a sistemas y aplicaciones que requieren control sobre los recursos.	Lenguaje de propósito general desarrollado para la plataforma .NET y utilizado en distintos tipos de aplicaciones.	Lenguaje creado específicamente para programar dentro de Godot Engine.
Paradigmas admitidos	Admite programación imperativa, orientada a objetos, genérica y funcional.	Se basa principalmente en programación orientada a objetos y también incorpora programación genérica y funcional.	Lenguaje orientado a objetos e imperativo, integrado con la estructura de nodos y escenas de Godot.
Sistema de tipado	Tipado estático. Los tipos se comprueban durante la compilación.	Tipado estático y fuerte. El compilador comprueba la compatibilidad de los tipos antes de ejecutar el programa.	Tipado gradual. Permite utilizar tipado dinámico o declarar tipos de manera explícita.
Organización del código	Utiliza clases, estructuras, funciones, plantillas, archivos de cabecera y archivos de implementación.	Utiliza clases, estructuras, interfaces, propiedades, métodos y tipos genéricos.	Cada archivo .gd representa un script o clase que puede extender un nodo u otra clase del motor.
Administración de memoria	Permite controlar directamente los recursos y utilizar mecanismos como RAII[13] para asociar su liberación con el ciclo de vida de los objetos.	Utiliza administración automática de memoria mediante el recolector de basura de .NET.	Godot utiliza conteo de referencias para determinados objetos; los nodos y otros objetos deben liberarse mediante las funciones proporcionadas por el motor.
Integración con el motor	Unreal Engine permite crear Actors, componentes y sistemas mediante C++. Las macros de reflexión conectan las clases y propiedades con el Editor.	Unity permite convertir clases derivadas de MonoBehaviour en componentes que pueden agregarse a los GameObjects.	Los scripts se asocian directamente con nodos y pueden acceder a sus propiedades, señales y funciones.
Configuración desde el Editor	Las macros UCLASS, UFUNCTION y UPROPERTY permiten exponer elementos al Editor y a Blueprints.	Los campos públicos o marcados con [SerializeField] pueden modificarse desde el Inspector.	La anotación @export permite mostrar y modificar variables desde el Inspector.
Programación visual complementaria	Puede combinarse con Blueprints para ampliar o configurar sistemas creados en código.	Puede complementarse con herramientas de programación visual, aunque el uso de scripts	La lógica se desarrolla principalmente mediante scripts; el motor incluye

[13] RAII: técnica que vincula la gestión de recursos con la vida de un objeto.

		C# es su forma principal de programación.	recursos visuales para funciones específicas.
Proceso de ejecución	El código se compila como parte del proyecto nativo de Unreal Engine.	Unity compila los scripts y puede ejecutarlos mediante los sistemas compatibles con la plataforma seleccionada, como Mono o IL2CPP.	El lenguaje es interpretado por Godot; el tipado estático permite detectar errores adicionales y utilizar operaciones optimizadas.
Facilidad inicial de aprendizaje	Requiere comprender compilación, clases, punteros, referencias y ciclo de vida de los recursos, por lo que puede presentar una curva inicial más amplia.	Su sintaxis y administración automática de memoria reducen parte de la complejidad relacionada con los recursos.	Presenta una sintaxis breve basada en sangrías y está diseñado para trabajar directamente con los elementos de Godot.
Nivel de control	Ofrece un control amplio sobre la organización del programa, la memoria y los sistemas de bajo nivel.	Ofrece un equilibrio entre estructura, seguridad de tipos y administración automática de memoria.	Prioriza la integración y rapidez de programación dentro de Godot sobre el control de bajo nivel.
Fortaleza principal	Control técnico, flexibilidad y conexión directa con la arquitectura de Unreal Engine.	Claridad estructural, integración con Unity y disponibilidad de herramientas del ecosistema .NET.	Sintaxis directa e integración estrecha con nodos, escenas y funciones de Godot.
Aspecto por considerar	Su complejidad, compilación y gestión de recursos pueden dificultar el aprendizaje inicial.	Las asignaciones frecuentes pueden aumentar el trabajo del recolector de basura en aplicaciones ejecutadas en tiempo real.	Su uso se encuentra principalmente vinculado con Godot y dispone de un ecosistema más específico que C++ o C#.

Nota. C++, Microsoft, Unity, Godot Engine y Unreal Engine

2.4.3. Plataformas de repositorios para el control de versiones

Las plataformas de repositorios permiten almacenar el código fuente, conservar el historial de cambios y coordinar el trabajo entre los integrantes de un proyecto. Además de integrar funciones propias de Git, estas herramientas incorporan mecanismos para administrar ramas, revisar modificaciones, registrar tareas y automatizar procesos relacionados con el desarrollo. En este apartado se analizan GitHub, GitLab y Bitbucket, considerando sus características de colaboración, control de acceso, integración continua y manejo de archivos, con el propósito de determinar cuál se ajusta mejor a las necesidades del proyecto.

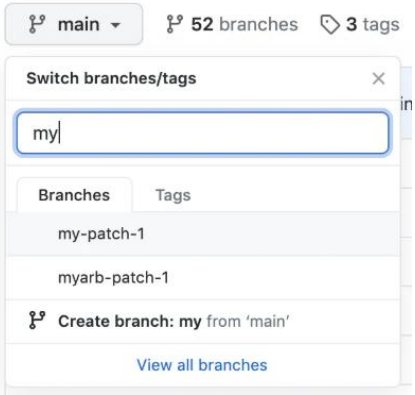
2.4.3.1. GitHub

GitHub es una plataforma que permite alojar repositorios administrados mediante Git y acceder a ellos desde un entorno web. Un repositorio puede almacenar el código fuente, archivos de configuración, documentación y el historial de modificaciones de un proyecto. La plataforma permite crear repositorios públicos, accesibles desde Internet, y repositorios privados, limitados a las personas autorizadas[57].

El trabajo colaborativo se organiza mediante ramas. Una rama mantiene una versión paralela del repositorio y permite realizar cambios sin modificar de manera inmediata la rama principal. De esta forma, cada integrante puede desarrollar o corregir una parte del proyecto y conservar sus modificaciones separadas hasta que estén preparadas para integrarse[58]. La Ilustración 18 muestra el selector de ramas de GitHub, desde el cual se puede localizar una rama existente o crear una nueva tomando como base la rama main.

Ilustración 20

Creación y selección de ramas dentro de un repositorio de GitHub.



Nota. GitHub

Las modificaciones realizadas en una rama pueden proponerse mediante una solicitud de incorporación denominada pull request. Esta función permite comparar los archivos

modificados, revisar los commits[14], mantener conversaciones sobre los cambios y ejecutar comprobaciones antes de integrar el contenido. Durante la revisión, los colaboradores pueden aprobar la propuesta, añadir comentarios o solicitar modificaciones. Una vez cumplidos los requisitos establecidos, el *pull request* puede fusionarse con la rama de destino o cerrarse sin integrar los cambios[59]. GitHub organiza en un mismo espacio la conversación, los *commits*, las comprobaciones automatizadas y las diferencias entre archivos.

GitHub también incorpora un sistema de incidencias denominado Issues, utilizado para registrar errores, tareas, solicitudes de funciones y actividades pendientes. Cada incidencia puede incluir una descripción, comentarios, etiquetas, responsables y fechas de seguimiento. Además, puede vincularse con una rama o un pull request, de modo que la tarea se cierre automáticamente cuando se incorpore la solución correspondiente[60].

La administración del repositorio puede reforzarse mediante reglas de protección de ramas. Estas reglas permiten exigir revisiones de aprobación o comprobaciones satisfactorias antes de fusionar una modificación. También pueden limitar determinadas acciones sobre ramas importantes, ayudando a reducir cambios directos o integraciones que no hayan sido revisadas[58].

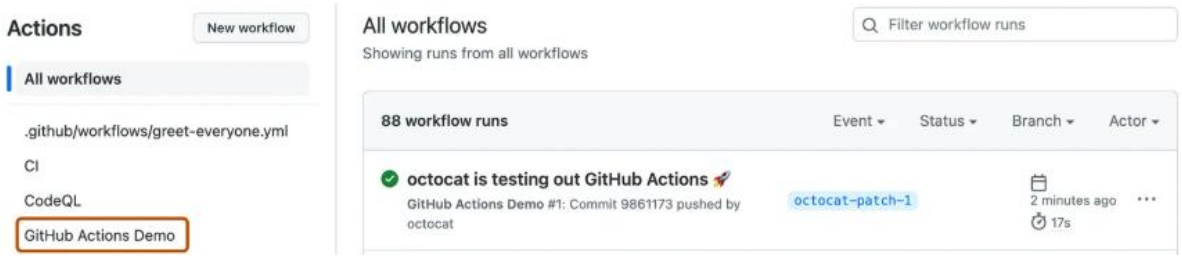
Para la automatización, la plataforma integra GitHub Actions, un sistema de integración y entrega continua. Los flujos de trabajo se definen mediante archivos YAML[15] ubicados dentro del repositorio y pueden ejecutarse cuando ocurre un evento, como el envío de nuevos cambios o la creación de un pull request. Estos flujos permiten automatizar tareas como compilar el proyecto, ejecutar pruebas y realizar despliegues[61]. La Ilustración 19

[14] Commit: registro de un conjunto de cambios en un repositorio.
[15] YAML: formato de texto utilizado para organizar datos de configuración.

presenta el historial de ejecución de estos flujos y permite identificar su estado, duración y evento de activación.

Ilustración 21

Ejecución de un flujo de trabajo automatizado mediante GitHub Actions.



Nota. GitHub

Debido a que algunos proyectos incluyen archivos de gran tamaño, GitHub es compatible con Git Large File Storage, conocido como Git LFS. Esta extensión sustituye los archivos grandes almacenados directamente en el repositorio por archivos de puntero, mientras que el contenido original se conserva en un sistema de almacenamiento separado. Los límites máximos dependen del plan contratado; actualmente, GitHub Free y GitHub Pro permiten archivos individuales de hasta 2 GB mediante Git LFS[62].

GitHub Free permite trabajar con repositorios públicos y privados, aunque algunas herramientas avanzadas para repositorios privados dependen de planes superiores. La plataforma también permite administrar colaboradores y permisos para controlar quién puede visualizar, modificar o gestionar un repositorio[57].

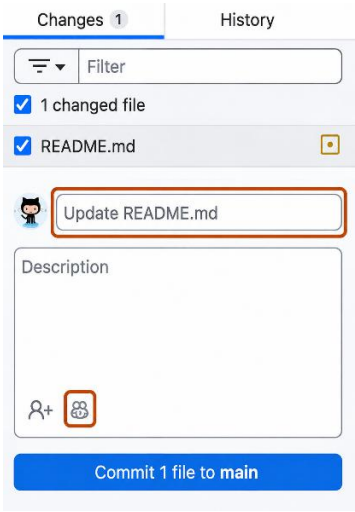
Para administrar el repositorio desde el equipo local, GitHub dispone de GitHub Desktop, una aplicación gráfica que permite revisar archivos modificados, registrar commits, gestionar ramas y sincronizar los cambios con el repositorio remoto mediante operaciones como push y pull[16]. Su interfaz facilita la ejecución de tareas habituales de Git sin depender

[16] Push y pull: operaciones para enviar y recibir cambios de un repositorio remoto.

exclusivamente de comandos escritos en una terminal[63]. La Ilustración 20 muestra la vista de cambios de la aplicación, donde se identifican los archivos modificados y los campos utilizados para registrar un nuevo commit.

Ilustración 22

Revisión y registro de cambios mediante GitHub Desktop.



Nota. GitHub

Entre las características principales de GitHub se encuentran la administración de ramas, las solicitudes de incorporación, el seguimiento de tareas, la revisión de código y la automatización mediante GitHub Actions. Como aspectos por considerar, el manejo de recursos grandes puede requerir Git LFS y algunas funciones avanzadas dependen del plan utilizado[62].

2.4.3.2. GitLab

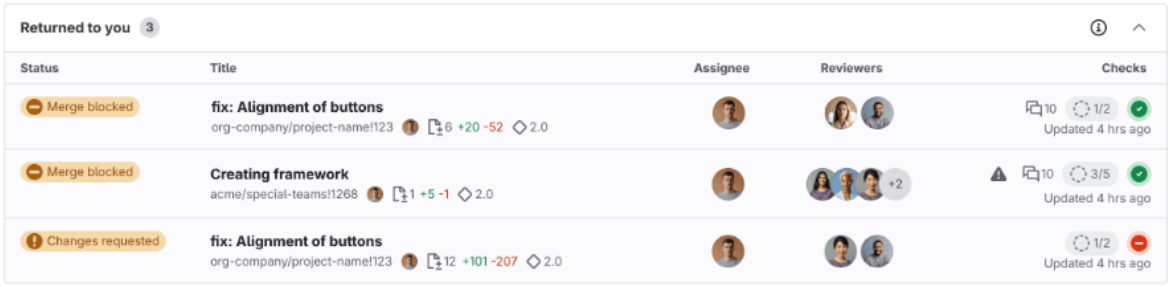
GitLab es una plataforma para alojar repositorios administrados mediante Git y organizar el trabajo relacionado con un proyecto de software. Dentro de la plataforma, el repositorio forma parte de un proyecto que reúne el código, el historial de cambios, la configuración de acceso y otras funciones de colaboración. Los archivos pueden añadirse desde la interfaz web o mediante Git, y el repositorio puede clonarse utilizando HTTPS o SSH[64].

El desarrollo colaborativo se organiza mediante ramas, que permiten modificar una versión separada del proyecto sin alterar inmediatamente la rama principal. Posteriormente, los cambios pueden integrarse mediante una merge request, en la que se comparan la rama de origen y la rama de destino. Este espacio reúne la descripción del cambio, los archivos modificados, los commits, los comentarios, las revisiones y el estado de las comprobaciones automatizadas[65].

Una merge request también permite asignar responsables y revisores. Los revisores pueden comentar líneas específicas, solicitar modificaciones o aprobar la incorporación de los cambios. Dependiendo de la configuración del proyecto, una solicitud puede permanecer bloqueada mientras existan conflictos, conversaciones sin resolver, comprobaciones fallidas o aprobaciones obligatorias pendientes[65]. La Ilustración 21 permite identificar el estado de cada solicitud, las personas asignadas para su revisión y las comprobaciones que todavía impiden completar la integración.

Ilustración 23

Seguimiento de solicitudes de integración y revisiones en GitLab.



Nota. GitLab

GitLab incorpora un sistema de Issues para registrar errores, tareas, solicitudes y actividades pendientes. Las incidencias pueden contener responsables, etiquetas, fechas, comentarios y otros datos utilizados para organizar el trabajo. También pueden mostrarse

dentro de tableros visuales, donde cada incidencia aparece como una tarjeta que puede desplazarse entre listas para representar diferentes etapas del proceso[66].

Las acciones sobre ramas importantes pueden regularse mediante reglas de protección asociadas con los roles del proyecto. Estas reglas permiten determinar quién puede enviar o fusionar cambios, impedir eliminaciones accidentales y exigir procesos de revisión antes de integrar una modificación. De esta manera, se reducen los cambios directos o no autorizados sobre las ramas protegidas[67].

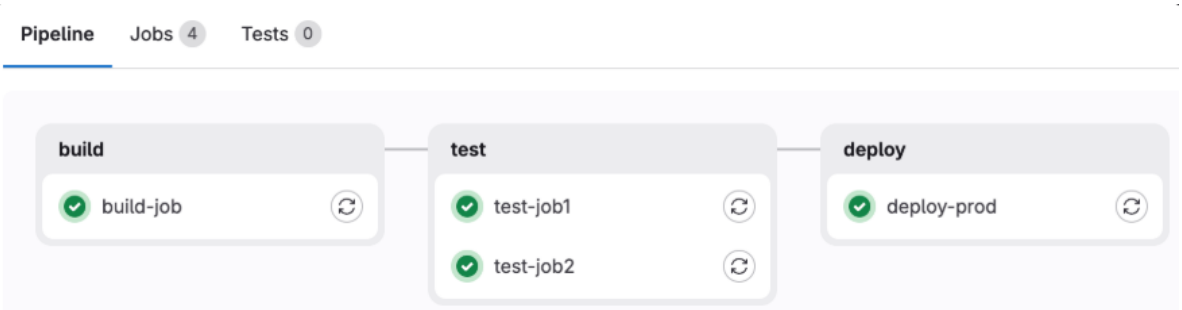
Para automatizar tareas, la plataforma incorpora GitLab CI/CD[17]. Su configuración se define mediante un archivo .gitlab-ci.yml, ubicado en la raíz del repositorio. En este archivo se establecen los trabajos, las etapas y las condiciones de ejecución. Los trabajos son procesados por agentes denominados runners y pueden utilizarse para compilar aplicaciones, ejecutar pruebas o realizar procesos de despliegue[68].

Las etapas de una canalización se ejecutan de manera secuencial de acuerdo con el orden definido, mientras que los trabajos pertenecientes a una misma etapa pueden procesarse en paralelo cuando existen runners disponibles. GitLab muestra el resultado de cada trabajo, sus registros de ejecución, su duración y una representación visual del estado de la canalización[68]. La Ilustración 22 muestra esta organización mediante las etapas build, test y deploy, y permite observar que dentro de la etapa de prueba se ejecutan dos trabajos separados.

Ilustración 24

Representación de trabajos y etapas en una canalización de GitLab CI/CD.

[17] CI/CD: automatización de la integración, pruebas y entrega del software.



Nota. GitLab

Para los proyectos que contienen recursos binarios de gran tamaño, GitLab ofrece compatibilidad con Git Large File Storage. Esta extensión almacena el archivo original fuera del repositorio y conserva en Git un pequeño archivo de referencia. Su utilización evita que cada modificación de una imagen, audio, video o archivo binario aumente directamente el historial del repositorio y afecte operaciones como la clonación o actualización del proyecto[69].

Entre las modalidades disponibles se encuentran GitLab.com y GitLab Self-Managed. GitLab.com funciona como un servicio alojado que no requiere instalar ni mantener una instancia propia, mientras que Self-Managed permite instalar, administrar y mantener GitLab en infraestructura controlada por la organización. Esta segunda alternativa ofrece mayor control sobre los datos y la infraestructura, pero también requiere encargarse de la configuración, actualización y mantenimiento del sistema[70].

Entre las características principales de GitLab se encuentran la administración de repositorios, las solicitudes de integración, los permisos por roles, el seguimiento de tareas y la integración de CI/CD dentro de una misma plataforma. También ofrece soporte para Git LFS, aspecto relevante en proyectos que contienen recursos audiovisuales[69]. Como aspectos por considerar, algunas funciones avanzadas dependen del plan contratado y la

configuración de canalizaciones o de una instalación propia puede requerir conocimientos adicionales de administración[70].

2.4.3.3. Bitbucket

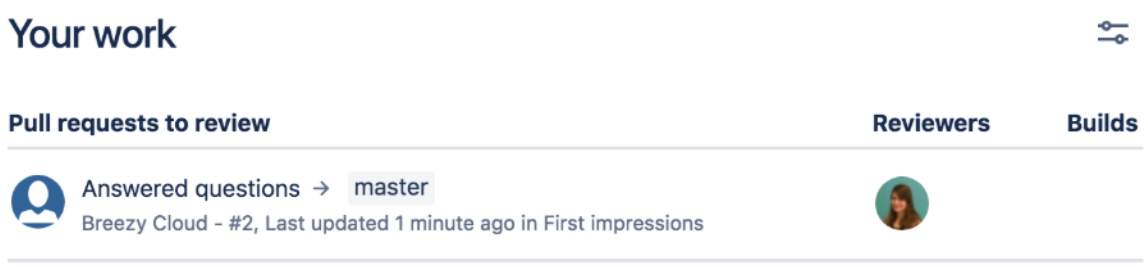
Bitbucket es una plataforma de Atlassian destinada al alojamiento y la administración de repositorios basados en Git. Los repositorios se crean dentro de espacios de trabajo y pueden asociarse con proyectos, lo que permite organizar el código y los archivos relacionados con el desarrollo. También pueden configurarse como públicos o privados, de acuerdo con las necesidades de acceso del proyecto[71].

El desarrollo colaborativo puede organizarse mediante ramas. Estas permiten trabajar en nuevas funciones, correcciones o versiones sin modificar directamente la rama principal. Bitbucket también dispone de un modelo de ramificación que permite definir ramas de desarrollo, producción, funciones, versiones, correcciones y soluciones urgentes, junto con prefijos que ayudan a mantener una nomenclatura uniforme[72].

Cuando los cambios realizados en una rama están preparados para revisión, pueden proponerse mediante una pull request. Esta función permite comparar la rama de origen con la de destino, consultar los archivos y commits modificados, asignar revisores y mantener conversaciones antes de fusionar el contenido. También pueden crearse tareas dentro de la revisión para identificar correcciones que deben completarse antes de aprobar la integración[73]. La Ilustración 23 muestra una pull request dirigida hacia la rama master, junto con la persona asignada como revisora y el estado general de la revisión dentro del panel de trabajo.

Ilustración 25

Seguimiento de una pull request asignada para revisión en Bitbucket.



Nota. Atlassian

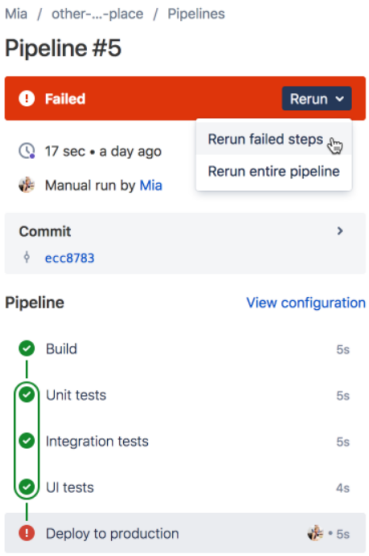
Los administradores pueden aplicar restricciones sobre ramas importantes y establecer condiciones para aceptar una *pull request*. Estas reglas permiten controlar qué usuarios o grupos pueden enviar o fusionar cambios y configurar comprobaciones antes de aceptar una modificación. Algunas comprobaciones obligatorias dependen del plan utilizado[74].

Bitbucket incorpora un servicio de automatización denominado Bitbucket Pipelines. Las canalizaciones se configuran mediante el archivo bitbucket-pipelines.yml, ubicado en la raíz del repositorio. En este archivo se definen los pasos utilizados para compilar, comprobar o desplegar una aplicación. Las tareas se ejecutan dentro de contenedores creados por la plataforma y pueden configurarse de acuerdo con el flujo de trabajo del proyecto[75].

La plataforma permite consultar el estado de cada canalización, sus pasos, registros y resultados. Esto facilita la identificación de errores relacionados con la configuración, las pruebas o la generación del proyecto antes de incorporar los cambios a una rama principal[75]. La Ilustración 24 muestra una canalización en la que las etapas de compilación y pruebas finalizaron correctamente, mientras que el despliegue presentó un error; además, la interfaz permite repetir únicamente los pasos fallidos o ejecutar nuevamente todo el proceso.

Ilustración 26

Seguimiento de las etapas y resultados de una canalización en Bitbucket Pipelines.



Nota. Atlassian

Una característica vinculada con el ecosistema de Atlassian es su integración con Jira. Al conectar ambas plataformas, los repositorios pueden mostrar tareas relacionadas con el código y vincular ramas, commits, pull requests, compilaciones y despliegues con elementos de trabajo. También es posible actualizar el estado de una tarea de Jira durante la integración de una solicitud[76].

Para gestionar archivos de mayor tamaño, Bitbucket Cloud admite Git Large File Storage. Esta extensión conserva los archivos grandes en un sistema de almacenamiento separado y mantiene referencias dentro del repositorio. Atlassian recomienda conservar los repositorios por debajo de 2 GB y establece un límite máximo de 4 GB; cuando se alcanza este límite, no es posible incorporar nuevos cambios que aumenten su tamaño[77], [78].

Bitbucket Cloud ofrece un plan gratuito para espacios de trabajo de hasta cinco usuarios, además de planes comerciales con más almacenamiento, minutos de ejecución para

Pipelines y funciones adicionales. Debido a que estas condiciones pueden cambiar, deben verificarse en la página oficial cuando se realice la comparación entre las plataformas[78].

Entre las características principales de Bitbucket se encuentran la organización de repositorios mediante espacios de trabajo y proyectos, la revisión de código mediante pull requests, la automatización con Pipelines y su integración con Jira[76]. Como aspectos por considerar, el plan gratuito establece un número reducido de usuarios y el almacenamiento de proyectos con numerosos recursos audiovisuales puede requerir Git LFS[77].

2.4.3.4. Cuadro comparativo de plataformas de repositorios

Después de revisar GitHub, GitLab y Bitbucket, se establecieron criterios comunes para comparar sus funciones de almacenamiento, colaboración y seguimiento de proyectos. Se consideraron la gestión de ramas, la revisión de cambios, la automatización, el control de tareas, el manejo de archivos grandes, las integraciones disponibles y las condiciones de sus planes gratuitos. La comparación permite reconocer las diferencias entre las plataformas y justificar la alternativa utilizada para alojar y administrar el repositorio del proyecto.

Tabla 10

Comparación de plataformas de repositorios para el control de versiones.

Criterio	GitHub	GitLab	Bitbucket
Repositorios	Permite crear repositorios públicos y privados.	Permite administrar repositorios dentro de proyectos públicos, internos o privados.	Permite organizar repositorios públicos y privados mediante espacios de trabajo y proyectos.
Gestión de ramas	Permite crear ramas, comparar cambios y establecer reglas de protección.	Incorpora ramas y reglas de protección según los roles y permisos del proyecto.	Permite definir ramas y modelos de ramificación para funciones, versiones y correcciones.
Revisión e integración	Utiliza pull requests para revisar, discutir y fusionar modificaciones.	Emplea merge requests para revisar cambios, comentarios, comprobaciones y aprobaciones.	Utiliza pull requests con revisores, comentarios y tareas asociadas.

Seguimiento de tareas	GitHub Issues permite registrar errores, actividades y solicitudes.	GitLab Issues permite organizar tareas, responsables, etiquetas y fechas.	Puede utilizar incidencias propias y vincular el desarrollo con tareas de Jira.
Automatización	GitHub Actions permite configurar flujos para compilar, probar y desplegar proyectos.	GitLab CI/CD organiza trabajos y etapas mediante el archivo .gitlab-ci.yml.	Bitbucket Pipelines define procesos de compilación, pruebas y despliegue mediante bitbucket-pipelines.yml.
Revisión automatizada	Las comprobaciones pueden mostrarse dentro de los pull requests.	Las canalizaciones pueden ejecutarse específicamente sobre las merge requests.	Las canalizaciones pueden activarse al crear o actualizar una pull request.
Archivos grandes	Admite Git LFS para almacenar archivos binarios fuera del historial principal.	Integra compatibilidad con Git LFS en proyectos alojados y administrados.	Admite Git LFS y asigna almacenamiento según el plan utilizado.
Aplicación de escritorio	Dispone de GitHub Desktop para revisar cambios, crear commits, trabajar con ramas y sincronizar el repositorio.	Puede administrarse mediante Git, la interfaz web y clientes externos compatibles.	Puede utilizarse mediante Git, la interfaz web y clientes externos compatibles.
Integraciones destacadas	Se integra con herramientas del ecosistema de GitHub y servicios externos.	Reúne repositorios, planificación, revisión, CI/CD y funciones de seguridad dentro de una misma plataforma.	Mantiene una integración directa con Jira y otras herramientas de Atlassian.
Plan gratuito	Incluye repositorios públicos y privados sin límite de cantidad.	Dispone de un nivel gratuito con funciones básicas de repositorios, colaboración y CI/CD.	Es gratuito para equipos de hasta cinco usuarios e incluye repositorios públicos y privados.
Fortaleza principal	Facilidad de colaboración, disponibilidad de recursos y aplicación oficial de escritorio.	Integración amplia de desarrollo, seguimiento y automatización dentro de la misma plataforma.	Integración estrecha con Jira y organización orientada al ecosistema de Atlassian.
Aspecto por considerar	Los proyectos con numerosos recursos binarios pueden requerir configurar Git LFS y almacenamiento adicional.	La cantidad de funciones y la configuración de CI/CD pueden aumentar la complejidad inicial.	El plan gratuito limita el número de usuarios y los recursos disponibles para Pipelines y Git LFS.

Nota. GitHub, GitLab y Atlassian

2.5. Arquitecturas de software

La arquitectura de una aplicación interactiva debe establecer una separación comprensible entre los datos del sistema, la representación visual y las acciones que responden a la interacción del usuario. Esta organización permite evitar que la lógica funcional quede mezclada con los elementos de presentación, lo que facilita la comprensión

de las responsabilidades asignadas a cada componente. Dentro de las arquitecturas orientadas a interfaces se encuentran distintas variantes de la familia Model-View-, entre las que destacan Modelo–Vista–Controlador y Modelo–Vista–Presentador [79].

2.5.1. Modelo–Vista–Controlador

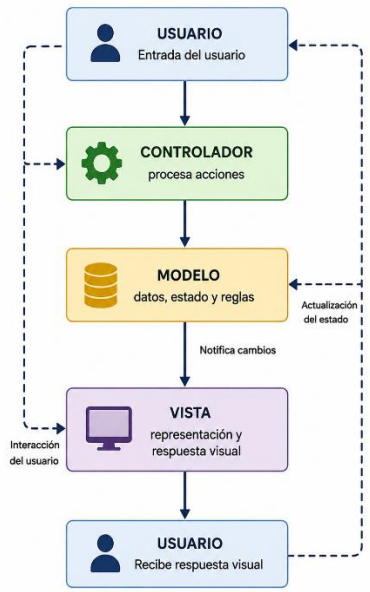
El patrón Modelo–Vista–Controlador, conocido por sus siglas MVC, surgió como una forma de organizar aplicaciones interactivas mediante la división de sus responsabilidades en tres elementos. El modelo representa la información y el estado del dominio; la vista presenta esa información al usuario; y el controlador interpreta las acciones realizadas sobre la interfaz y coordina las modificaciones necesarias dentro del sistema. Esta separación permite que la presentación pueda modificarse sin concentrar toda la lógica y los datos en un mismo componente[80].

Aunque existen distintas interpretaciones de MVC, su principio central consiste en separar el dominio de la aplicación de su presentación y del procesamiento de las entradas. Syromiatnikov y Weyns explican que la familia MVC distingue la representación del dominio, la visualización del estado de la aplicación y el control de la interacción del usuario. Sin embargo, la forma exacta en que estos elementos se comunican puede cambiar según el lenguaje, el framework o el tipo de aplicación[80].

En la adaptación utilizada en el proyecto, el modelo almacena los datos utilizados por el sistema, la vista representa esos datos dentro de la interfaz o de la escena y el controlador procesa las entradas, ejecuta decisiones y actualiza el modelo. La vista puede reflejar posteriormente el nuevo estado mediante eventos o mecanismos de observación. Esta organización permite separar, por ejemplo, los valores que representan el progreso de una actividad, los elementos visuales que muestran ese avance y los scripts que comprueban si el usuario cumplió una condición.

Ilustración 27

Estructura general del patrón Modelo–Vista–Controlador.



Nota. Elaboración propia con base en Syromiatnikov y Weyns

En el desarrollo de videojuegos, la separación entre la lógica de juego y la representación visual puede facilitar el mantenimiento y la adaptación del software. Olsson, Toll, Wingkvist y Ericsson analizaron la aplicación de MVC en cinco proyectos de videojuegos y encontraron diferentes formas de implementar el patrón. Los autores observaron que la calidad obtenida dependía de la organización concreta de sus componentes y que una reestructuración arquitectónica podía mejorar aspectos relacionados con la independencia del motor gráfico y la portabilidad. Por tanto, el uso de MVC no garantiza por sí solo una arquitectura adecuada, sino que requiere definir con claridad las responsabilidades y dependencias de cada elemento[81].

Entre las ventajas de MVC se encuentra la separación de responsabilidades, ya que los datos, la presentación y el procesamiento de las acciones se distribuyen entre componentes distintos. Esta organización puede favorecer la modificación de la interfaz, la reutilización de la lógica y la identificación de los elementos responsables de cada

comportamiento. No obstante, si el controlador concentra demasiadas funciones o la vista accede de forma desorganizada al modelo, pueden volver a generarse dependencias que dificulten el mantenimiento. Por esta razón, la aplicación del patrón debe adaptarse a las necesidades y al tamaño del proyecto.

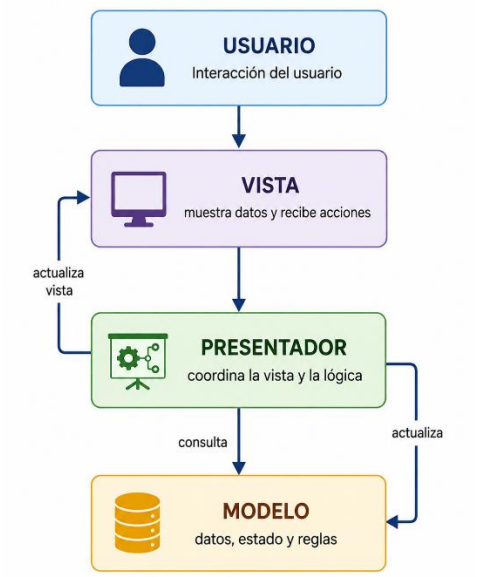
2.5.2. Modelo–Vista–Controlador

El patrón Modelo–Vista–Presentador, identificado mediante las siglas MVP, fue desarrollado como una generalización de MVC para organizar aplicaciones orientadas a objetos. Potel explicó este enfoque como un modelo de programación en el que el presentador coordina los elementos necesarios para relacionar los datos, la presentación y la interacción. Su propósito consiste en mantener una separación clara entre el dominio del sistema y la interfaz utilizada por el usuario[79].

MVP también se compone de tres elementos. El modelo conserva los datos y el estado de la aplicación; la vista muestra la información y recibe las acciones del usuario; y el presentador actúa como intermediario entre ambos. A diferencia de la adaptación habitual de MVC, la vista comunica las entradas al presentador, quien consulta o modifica el modelo y prepara la información que debe mostrarse posteriormente. De esta manera, el presentador mantiene la coordinación principal de la interacción.

En una variante de MVP denominada Passive View, la vista no accede directamente al modelo. Su responsabilidad se limita a representar la información proporcionada por el presentador y a comunicarle las acciones realizadas por el usuario. Esta separación puede facilitar la comprobación independiente de la lógica, debido a que el presentador puede evaluarse sin depender completamente de los elementos visuales. Sin embargo, también requiere establecer interfaces, eventos y relaciones adicionales entre los componentes[82].

Ilustración 28
Estructura general del patrón Modelo–Vista–Presentador.



Nota. Elaboración propia con base en Potel

En Unity, MVP puede aplicarse cuando los componentes visuales reciben las entradas mediante botones, interruptores, controles deslizantes u otros eventos de interfaz. La vista transmite estas acciones al presentador; el presentador modifica el modelo; y, después, actualiza la información mostrada. Esta estructura resulta útil cuando se requiere mantener la lógica de presentación fuera de los componentes visuales y realizar pruebas sobre ella de manera independiente. Como aspecto favorable, MVP establece una comunicación más explícita entre sus componentes y reduce el acceso directo de la vista al modelo. Esto puede favorecer las pruebas y la sustitución de la interfaz.

Como aspecto por considerar, el patrón puede incrementar la cantidad de interfaces, eventos y clases necesarias, especialmente cuando existen numerosas pantallas o elementos interactivos. Por ello, su aplicación resulta más justificable cuando la complejidad de la interfaz o los requisitos de comprobación independiente compensan ese incremento estructural.

2.5.3. Comparación de arquitecturas de software

Tabla 11

Comparación entre MVC y MVP

Criterio	Modelo–Vista–Controlador	Modelo–Vista–Presentador
Flujo de interacción	El controlador recibe las acciones del usuario, modifica el modelo y permite que la vista represente el nuevo estado.	La vista recibe las acciones del usuario y las comunica al presentador, que coordina la consulta o modificación del modelo y actualiza la vista.
Relación entre vista y modelo	Puede existir comunicación entre la vista y el modelo, dependiendo de la variante implementada.	La vista no accede directamente al modelo; la comunicación se realiza mediante el presentador.
Distribución de la lógica	La lógica de control se concentra en uno o varios controladores.	La lógica de presentación e interacción se concentra en el presentador.
Dependencia de la interfaz	El nivel de dependencia depende de cómo se implementen la vista y el controlador.	La vista puede mantenerse más pasiva, lo que reduce su participación en la lógica del sistema.
Facilidad para realizar pruebas	Los controladores pueden comprobarse de manera independiente cuando se encuentran correctamente separados de la vista.	El presentador puede probarse sin ejecutar completamente la interfaz, especialmente en la variante Passive View.
Complejidad estructural	Presenta una organización relativamente directa, adecuada para sistemas con una relación clara entre entrada, estado y respuesta visual.	Puede requerir más interfaces, eventos y clases para mantener la comunicación entre la vista, el presentador y el modelo.
Aplicación en Unity	Se adapta a proyectos donde los scripts controlan acciones, modifican el estado y actualizan elementos de la escena o interfaz.	Resulta útil cuando se busca separar con mayor rigor la interfaz de Unity de la lógica encargada de presentarla.
Principal riesgo	El controlador puede acumular demasiadas responsabilidades si no se divide correctamente.	El incremento de clases y conexiones puede resultar innecesario para proyectos de alcance reducido.

Nota. Elaboración propia con base en Syromiatnikov y Weyns, y Potel

Para el desarrollo de Lógica en Juego se aplicó el patrón Modelo–Vista–Controlador, cuya separación entre modelo, vista y controlador permite distribuir los datos del sistema, la representación visual y el procesamiento de las interacciones entre componentes con responsabilidades diferenciadas. Esta organización se ajustó al funcionamiento del entorno guiado, en el que el usuario interactuó con los elementos de una escena, realizó las

configuraciones solicitadas y recibió una respuesta visual después de cada proceso de validación.

Dentro de esta arquitectura, el modelo reunió la información relacionada con el estado de las mecánicas, los valores configurados, el progreso del usuario y los criterios necesarios para completar cada actividad. La vista estuvo conformada por los elementos visibles de la escena, las páginas del tutorial, los mensajes, los indicadores y la retroalimentación presentada durante el recorrido. Por su parte, el controlador estuvo constituido por los scripts encargados de procesar las acciones del usuario, comprobar las configuraciones realizadas, actualizar el estado del sistema y determinar cuándo se cumplían las condiciones necesarias para continuar.

La aplicación de MVC permitió mantener diferenciadas las responsabilidades de los componentes que participaron en la experiencia educativa. Los datos y las condiciones de las actividades se administraron desde el modelo; la información y las respuestas del sistema se presentaron mediante la vista; y las acciones, comprobaciones y cambios de progreso fueron gestionados por los controladores. Esta distribución es coherente con el uso de MVC en proyectos de videojuegos, donde la separación entre la lógica, el estado y la presentación puede favorecer la modularidad y el mantenimiento del sistema[81].

2.6. Herramientas de desarrollo

A partir de la comparación realizada de los motores de desarrollo, se seleccionó Unity como motor de desarrollo porque sus características se ajustaron mejor a las necesidades del sistema educativo interactivo. El uso de C# permitió programar las mecánicas y las validaciones mediante scripts, mientras que la organización por GameObjects y componentes facilitó relacionar el código con los elementos configurados en cada escena[19]. La elección

también estuvo respaldada por los trabajos relacionados analizados, ya que Interactive Tutorial in Unity, Using Unity to Teach Game Programming y GameDevDojo emplearon este motor para integrar contenidos formativos, actividades prácticas y sistemas de retroalimentación. Además, se consideraron la disponibilidad de documentación, los recursos de aprendizaje y la experiencia previa con la herramienta[31].

A partir de la comparación realizada de los lenguajes de programación, se seleccionó C# debido a su compatibilidad con Unity y a la forma en que sus scripts pueden incorporarse como componentes dentro de los GameObjects[29]. Su sistema de clases, interfaces y campos serializados facilita la organización de comportamientos y la configuración de parámetros desde el Editor[50]. También se consideraron la experiencia previa con el lenguaje, la disponibilidad de documentación y su capacidad para mantener una separación clara entre las distintas responsabilidades del código.

A partir de la comparación realizada de plataformas de repositorio, se seleccionó GitHub para alojar el repositorio del proyecto, debido a la posibilidad de trabajar de manera privada, conservar el historial de modificaciones y compartir los archivos entre los integrantes[57]. Para la administración local GitHub Desktop, cuya interfaz permite revisar los cambios antes de registrarlos, crear commits, trabajar con ramas y sincronizar el contenido mediante operaciones de envío y actualización[63]. Estas funciones facilitan la organización del trabajo sin depender únicamente de comandos escritos en una terminal. GitHub Desktop permite crear ramas, revisar cambios, registrar commits y enviar las modificaciones al repositorio remoto.

2.7. Marco conceptual

El marco conceptual reúne los términos técnicos necesarios para comprender el funcionamiento y la finalidad del sistema educativo interactivo desarrollado. Los conceptos seleccionados se relacionan directamente con la programación de mecánicas, la organización del aprendizaje guiado, los fundamentos matemáticos aplicados al movimiento y el procedimiento utilizado para recoger la valoración de los participantes.

Tabla 12

Conceptos técnicos aplicados al proyecto.

Concepto	Definición aplicada al proyecto
Programación de mecánicas de videojuego	Proceso de desarrollar comportamientos interactivos mediante código, entradas, variables, condiciones y eventos. En el proyecto se aplicó a la implementación del movimiento, sprint, dash, zonas y máquina de estados [1], [2].
Entorno guiado de aprendizaje	Sistema que organiza actividades mediante instrucciones, pasos progresivos y validaciones. En el proyecto permitió orientar al usuario durante la implementación de cada mecánica dentro de Unity [10], [11].
Pensamiento computacional	Capacidad para analizar un problema, dividirlo en partes y estructurar una solución lógica. En el proyecto facilitó la comprensión de la relación entre código, condiciones y comportamiento del sistema [5], [6].
Script en Unity	Archivo de código desarrollado en C# que define funciones y comportamientos dentro de la escena. En el proyecto se utilizó para controlar acciones, validaciones y mecánicas del entorno guiado[29].
Vector	Estructura matemática que representa dirección y magnitud. En el proyecto se empleó para calcular el desplazamiento del jugador dentro de la escena.
Normalización	Proceso mediante el cual un vector conserva su dirección y ajusta su magnitud a uno. En el proyecto se utilizó para mantener una velocidad uniforme en el movimiento del personaje[83].
Máquina de estados	Estructura que organiza el comportamiento de un sistema mediante estados y transiciones. En el proyecto permitió controlar condiciones como reposo, movimiento, sprint, dash o permanencia en el aire [2].
Sprint	Mecánica que incrementa temporalmente la velocidad de desplazamiento del personaje. En el proyecto se integró como una ampliación del movimiento base del jugador.

Dash	Mecánica de desplazamiento rápido y breve en una dirección determinada. En el proyecto se programó considerando impulso, duración y tiempo de reutilización.
Escala de Likert	Instrumento de medición que registra niveles de acuerdo o valoración mediante categorías ordenadas. En el proyecto se utilizó para evaluar la percepción de los participantes sobre claridad, utilidad y facilidad de uso del sistema[84].

Nota. Elaboración propia

Capítulo III

3. Manual De Usuario

3.1. Presentación y perfil del usuario

Lógica en Juego es un entorno formativo guiado que funciona dentro del Editor de Unity. Su propósito es acompañar al usuario en la realización progresiva de actividades relacionadas con el movimiento del personaje, la ampliación de sus controles, la interacción con zonas del entorno y la gestión de estados. Las instrucciones se presentan mediante la ventana integrada de tutoriales, mientras que las actividades se desarrollan directamente sobre los objetos, scripts y componentes del proyecto.

La experiencia articula el contenido del libro didáctico complementario con actividades prácticas dentro de Unity. El libro introduce y refuerza los conceptos correspondientes a cada tema y, posteriormente, el entorno guiado permite aplicarlos mediante instrucciones secuenciales, elementos resaltados, configuraciones y pruebas en Play Mode. De esta manera, se establece una relación ordenada entre la consulta del contenido, su implementación práctica y la comprobación del resultado.

Las actividades se encuentran distribuidas en cuatro módulos que deben completarse progresivamente: movimiento del personaje, ampliación del control del jugador, sistema de zonas del entorno y sistema de estados. En cada módulo, el usuario debe identificar el elemento señalado, realizar la actividad indicada, ejecutar la escena y revisar el resultado de la validación. El progreso se registra automáticamente y permite acceder al contenido siguiente, por lo que se recomienda respetar el orden presentado en la ventana Tutorials.

El entorno está dirigido principalmente a estudiantes o desarrolladores en formación interesados en comprender la programación de mecánicas de videojuego mediante Unity.

También puede ser utilizado por docentes o evaluadores que necesiten revisar el desarrollo de las actividades y comprobar el funcionamiento del recorrido. Debido a que el usuario trabaja directamente dentro del Editor, se recomienda contar con conocimientos básicos sobre la interfaz de Unity y nociones iniciales de programación en C#.

El proyecto no dispone de una aplicación ejecutable independiente, ya que la experiencia se desarrolla completamente dentro del Editor de Unity. Para utilizarla, el usuario debe disponer de la carpeta completa del proyecto, abrirla mediante Unity Hub y seguir las instrucciones integradas en cada módulo.

Tabla 13

Recursos y requisitos para descargar e instalar el proyecto.

Recurso	Requisito
Equipo	Computador de escritorio o portátil compatible con Unity 6
Procesador	Procesador de 64 bits, Intel Core i3 o equivalente como mínimo.
Memoria RAM	8 GB de RAM como mínimo.
Sistema operativo	Windows 10 o superior, sistema en el que se comprobó el funcionamiento del recorrido
Software	Unity Hub y Unity 6.0, versión 6000.0.49f1
Proyecto	Carpeta completa Logica-en-Juego, con todos sus archivos y subcarpetas
Almacenamiento	Aproximadamente 3 GB de espacio libre para almacenar el proyecto e importar sus recursos, además del espacio necesario para la instalación de Unity.
Controles	Teclado y ratón
Conexión a Internet	Necesaria para descargar Unity o recibir el proyecto; el recorrido puede utilizarse posteriormente desde el Editor
Dispositivos adicionales	No requiere visor de realidad virtual o aumentada ni mando de juego

Nota. Elaboración propia

3.2. Acceso e inicio

El acceso comienza con la carpeta completa del proyecto. Deben conservarse juntos sus archivos y subcarpetas, ya que Unity los utiliza durante la apertura y la preparación del entorno. Si el proyecto se recibe en un archivo comprimido, primero debe extraerse en una ubicación con permisos de lectura y escritura.

Ilustración 29

Procedimiento de inicio del proyecto Lógica en Juego en Unity.

Nota. Elaboración propia

Cuando la apertura es correcta, el Editor muestra la escena Niveles, la vista Game y la ventana del recorrido guiado. La pantalla inicial presenta el nombre LÓGICA EN JUEGO, una explicación breve y el botón Comenzar, como se observa en la Ilustración 30.

Ilustración 30

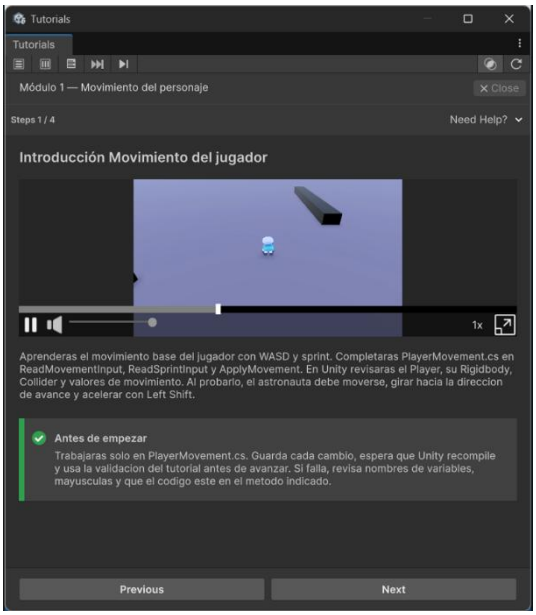
Pantalla inicial del entorno guiado.

Nota. Elaboración propia

Después de las indicaciones generales, el sistema abre el primer módulo. La cabecera permite reconocer el nombre Módulo 1 — Movimiento del personaje, el número de paso y los botones de navegación. La Ilustración 31 muestra la primera página del módulo y confirma que el recorrido se encuentra listo para comenzar.

Ilustración 31

Acceso al primer módulo del recorrido.



Nota. Elaboración propia

Durante la apertura inicial, Unity puede dedicar varios minutos a preparar los recursos. El usuario debe esperar hasta que el Editor vuelva a responder con normalidad antes de seleccionar elementos o iniciar Play Mode. Una vez disponible la ventana Tutorials, el recorrido se realiza desde sus páginas y no es necesario buscar instrucciones fuera del proyecto.

3.3. Navegación y controles

La interacción combina la ventana Tutorials con las áreas habituales del Editor que intervienen en la actividad. Cada una cumple una función concreta dentro del recorrido:

Tabla 14

Ventas de Unity con su uso en el recorrido.

Elemento	Uso dentro del recorrido
Ventana Tutorials	Presenta las instrucciones, los pasos y el avance
Hierarchy	Permite seleccionar objetos de la escena
Project	Permite localizar scripts y recursos
Inspector	Permite revisar o modificar los componentes del elemento seleccionado
Game	Muestra la actividad mientras la escena se encuentra en Play Mode
Console	Presenta el resultado de la validación al finalizar la prueba
Botón Play	Inicia y detiene la ejecución de la escena
Next / Previous	Permiten avanzar o regresar entre instrucciones
Complete / Done	Finalizan el módulo cuando corresponde

Nota. Elaboración propia.

Los botones Next y Previous se utilizan para recorrer las páginas del módulo. Next permanece inactivo cuando todavía falta una acción obligatoria. Al completar la acción solicitada, la señal de la instrucción cambia y el botón se habilita. En la última página se utiliza Complete o Done, según el módulo, para cerrarlo y registrar su finalización.

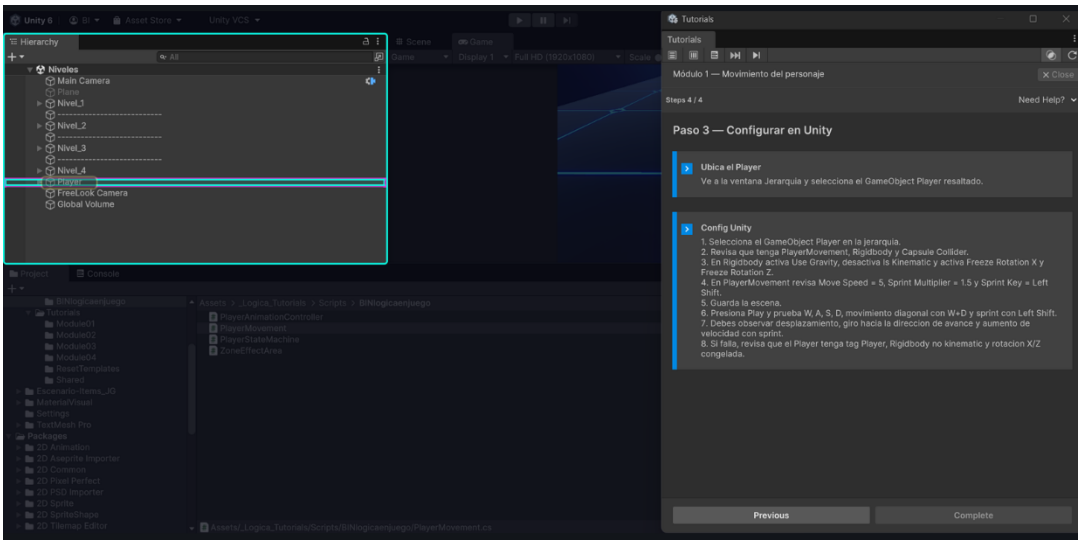
El entorno utiliza señales visuales para dirigir la atención. Un borde cian con una línea magenta identifica el área del Editor que debe utilizarse, como Hierarchy o Project. Dentro de la instrucción, una franja azul y una flecha indican una acción activa; cuando la

condición se cumple, la franja cambia a verde y aparece una marca de verificación. Estas señales no reemplazan la lectura del paso, sino que permiten relacionar la indicación con el lugar exacto donde debe trabajarse.

La Ilustración 32 presenta una instrucción activa del Módulo 1 junto con el objeto Player resaltado en Hierarchy. El usuario debe seleccionar el elemento señalado antes de continuar con su configuración.

Ilustración 32

Elemento solicitado y resaltado dentro del Editor.

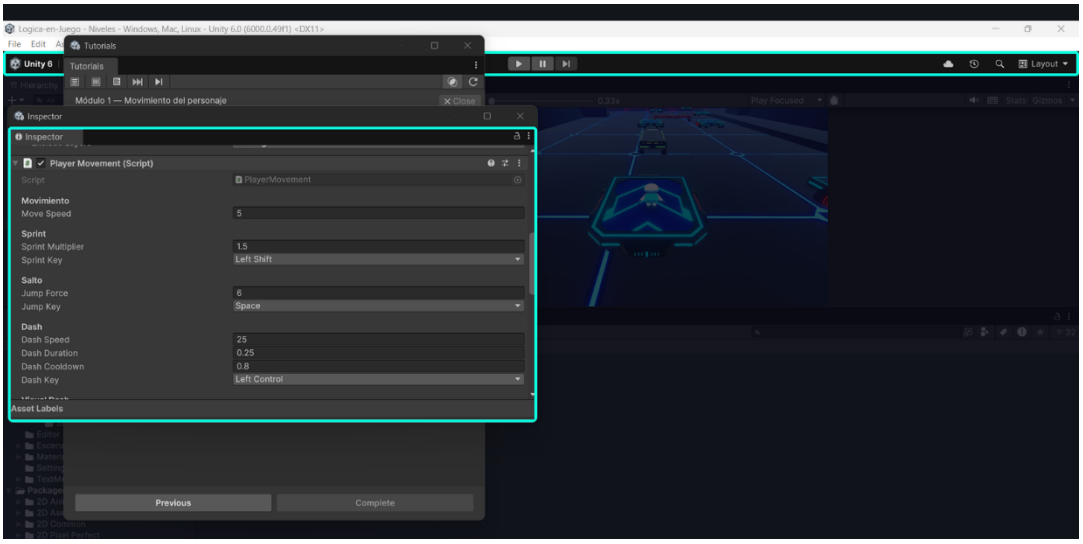


Nota. Elaboración propia

Una vez seleccionado el objeto, el Inspector muestra la información necesaria para la actividad. La Ilustración 33 permite reconocer el componente de movimiento y los controles asociados al recorrido, sin necesidad de abandonar la ventana de Unity.

Ilustración 33

Configuración de la actividad en el Inspector.

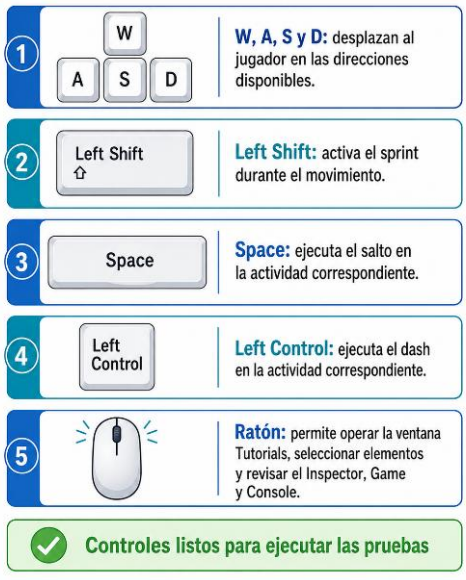


Nota. Elaboración propia

Para realizar las pruebas se emplean los siguientes controles:

Ilustración 34

Controles utilizados durante las pruebas del proyecto.



Nota. Elaboración propia

Las teclas de la actividad se utilizan únicamente después de iniciar Play Mode y de colocar el foco en la vista Game. Fuera de Play Mode, el teclado y el ratón se destinan a la navegación del Editor.

3.4. Funciones e interacciones disponibles

El recorrido está compuesto por cuatro módulos consecutivos. La tabla siguiente resume el propósito observable de cada uno y la comprobación general que se realiza al finalizar su prueba.

Tabla 15

Cómo se observan los módulos y su comprobación.

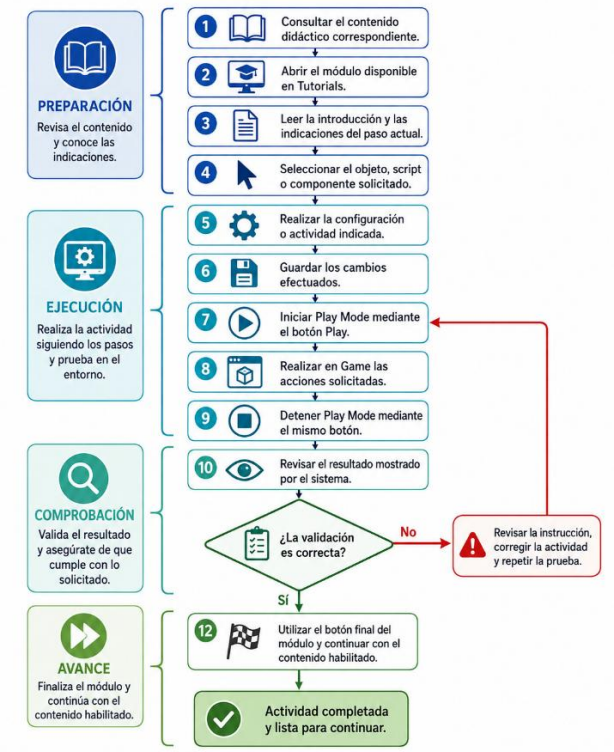
Módulo	Propósito de la actividad	Comprobación general
Movimiento del Personaje	Aplicar movimiento direccional y sprint al jugador	El sistema comprueba que existan desplazamiento y sprint durante Play Mode
Ampliación del control del jugador	Incorporar salto, dash y su señal visual	El sistema comprueba las acciones de salto, dash y estela
Sistema de zonas del entorno	Probar tres efectos de zona y la recuperación posterior	El sistema comprueba la activación de las zonas y la restauración del comportamiento
Sistema de estados	Observar cambios de estado asociados al movimiento y a las zonas	El sistema comprueba la transición, el efecto activo y el retorno al estado normal

Nota. Elaboración propia

Cada módulo contiene sus propias instrucciones específicas. Por esta razón, el manual se concentra en la secuencia común de uso y no reproduce todos los textos ni valores mostrados dentro de Tutorials. La actividad general se desarrolla de la siguiente manera:

Ilustración 35

Desarrollo general de la actividad.



Nota. Elaboración propia

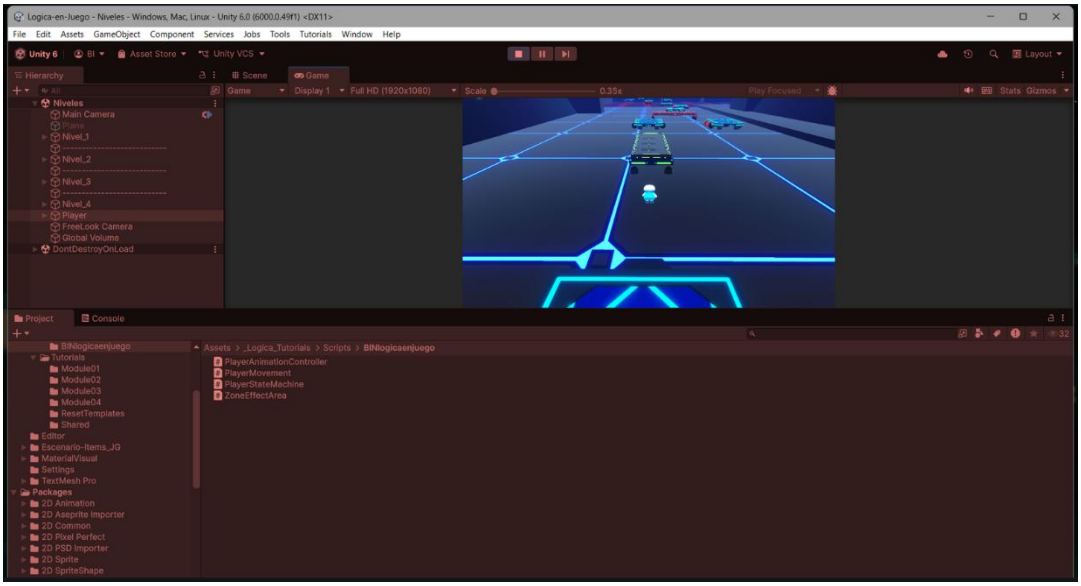
3.4.1. Validación y progreso

La comprobación práctica se produce después de ejecutar la escena. El usuario debe iniciar Play Mode, realizar todas las acciones indicadas y detener la ejecución. Al salir de Play Mode, el sistema evalúa lo observado y muestra el resultado en Console. La prueba no debe considerarse terminada mientras la escena continúe en ejecución.

La Ilustración 36 muestra el botón de ejecución activo y la vista Game durante la prueba de movimiento. El cambio de color del Editor y el icono cuadrado en el control central permiten reconocer que Play Mode se encuentra activo.

Ilustración 36

Actividad ejecutada en Play Mode.

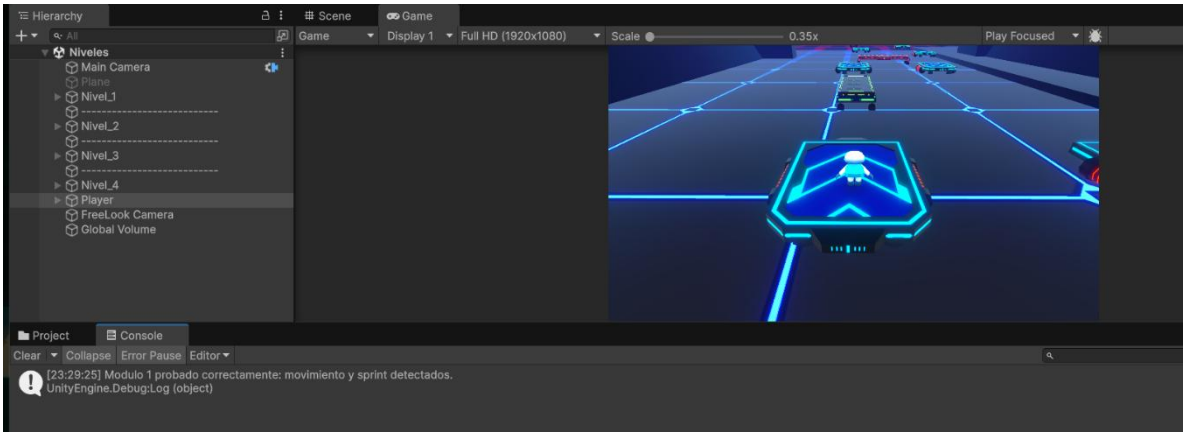


Nota. Elaboración propia

Cuando la prueba es correcta, Console presenta el mensaje de confirmación correspondiente. En el caso representativo del Módulo 1, el resultado informa que se detectaron movimiento y sprint. La Ilustración 37 muestra esta validación después de detener Play Mode.

Ilustración 37

Validación correcta de la actividad.



Nota. Elaboración propia

Una validación correcta registra el avance automáticamente, pero el usuario todavía debe volver a Tutorials y pulsar Complete o Done para cerrar el módulo. Si el resultado no es correcto, puede revisar la instrucción, ajustar la actividad y repetir la secuencia. El contenido siguiente queda disponible de acuerdo con el progreso alcanzado.

3.4.2. Solución de problemas frecuentes

Tabla 16

Posibles problemas y sus acciones a tomar.

Problema	Posible causa	Acción recomendada
La bienvenida no aparece	Ya fue cerrada o mostrada anteriormente	Abrirla desde el menú Tutorials > Welcome Dialog
La lista de módulos no está visible	La ventana Tutorials fue cerrada	Abrir nuevamente Tutorials > Show Tutorials
El tutorial no permite avanzar	Falta completar la acción solicitada	Revisar la instrucción y esperar la marca de cumplimiento
El Inspector no muestra el elemento solicitado	Se seleccionó otro objeto o recurso	Seleccionar exactamente el elemento indicado
La actividad no se valida	No se realizaron todas las acciones o no se detuvo Play Mode	Repetir la prueba, detener la ejecución y revisar Console

Nota. Elaboración propia

3.5. Evidencia de uso

El recorrido lineal comienza con la recepción y descompresión de la carpeta Logica-en-Juego, la cual se agrega y abre desde Unity Hub utilizando Unity 6.0, versión 6000.0.49f1. Una vez cargada la escena Niveles, el usuario accede a la ventana de bienvenida y pulsa Comenzar, como se observa en la Ilustración 27. Posteriormente, revisa las indicaciones generales e ingresa al Módulo 1: Movimiento del personaje, presentado en la Ilustración 28.

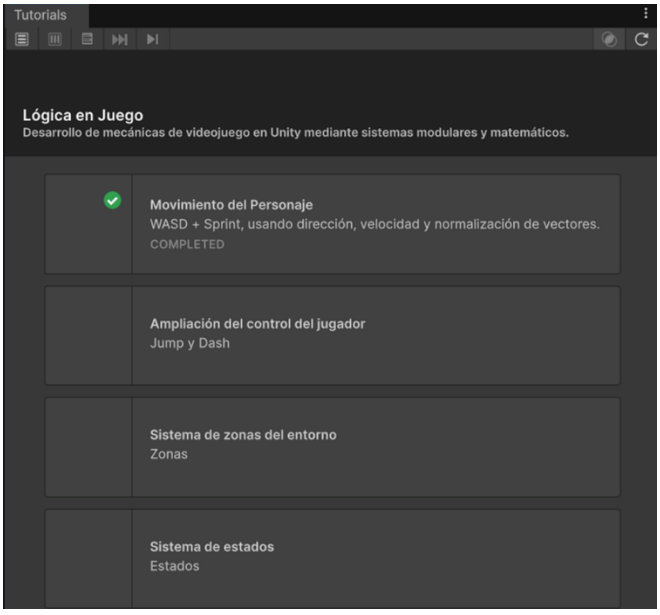
Durante la actividad, el usuario sigue las instrucciones de la ventana Tutorials y localiza el objeto o recurso señalado mediante las ayudas visuales del entorno. La Ilustración 29 muestra la selección del objeto Player, mientras que la Ilustración 30 presenta la configuración correspondiente en el Inspector. Después de realizar y guardar los cambios solicitados, se inicia la ejecución de la escena.

En Play Mode, el usuario controla al personaje mediante las teclas W, A, S y D y activa el sprint con Left Shift, tal como se evidencia en la Ilustración 31. Una vez realizadas las acciones requeridas, se detiene la ejecución y el sistema comprueba automáticamente el resultado. La Ilustración 32 presenta el mensaje de validación correcta mostrado en la ventana Console.

Finalmente, el usuario regresa a Tutorials y pulsa Complete para cerrar la actividad. El progreso queda registrado y el contenido siguiente se habilita. Como se muestra en la Ilustración 38, el módulo Movimiento del Personaje aparece con el estado Completed, mientras que Ampliación del control del jugador queda disponible para continuar el recorrido.

Ilustración 38

Registro del progreso y continuidad del recorrido.



Nota. Elaboración propia

La secuencia representativa concluye con la actividad validada, el módulo cerrado y el avance registrado. El mismo criterio de uso se mantiene en los módulos posteriores:

consultar el contenido, seguir la instrucción, seleccionar el elemento señalado, realizar la actividad, probarla en Play Mode, revisar el resultado y continuar según el progreso.

Capítulo IV

4. Manual Del Programador

El presente manual documenta la arquitectura, la organización técnica, la programación de las mecánicas y la integración educativa de Lógica en Juego. El sistema fue desarrollado dentro de Unity como una experiencia de aprendizaje compuesta por un entorno guiado, una escena de práctica y un libro didáctico asociado con cuatro módulos progresivos.

La documentación se concentra en los elementos implementados en el proyecto: scripts de ejecución, componentes de Unity, herramientas de Editor, validaciones, control de progreso y relación entre el contenido conceptual y las actividades prácticas. La explicación del código se mantiene a nivel funcional y arquitectónico, mientras que el desarrollo detallado de los fundamentos matemáticos corresponde al libro didáctico.

4.1. Arquitectura general del sistema

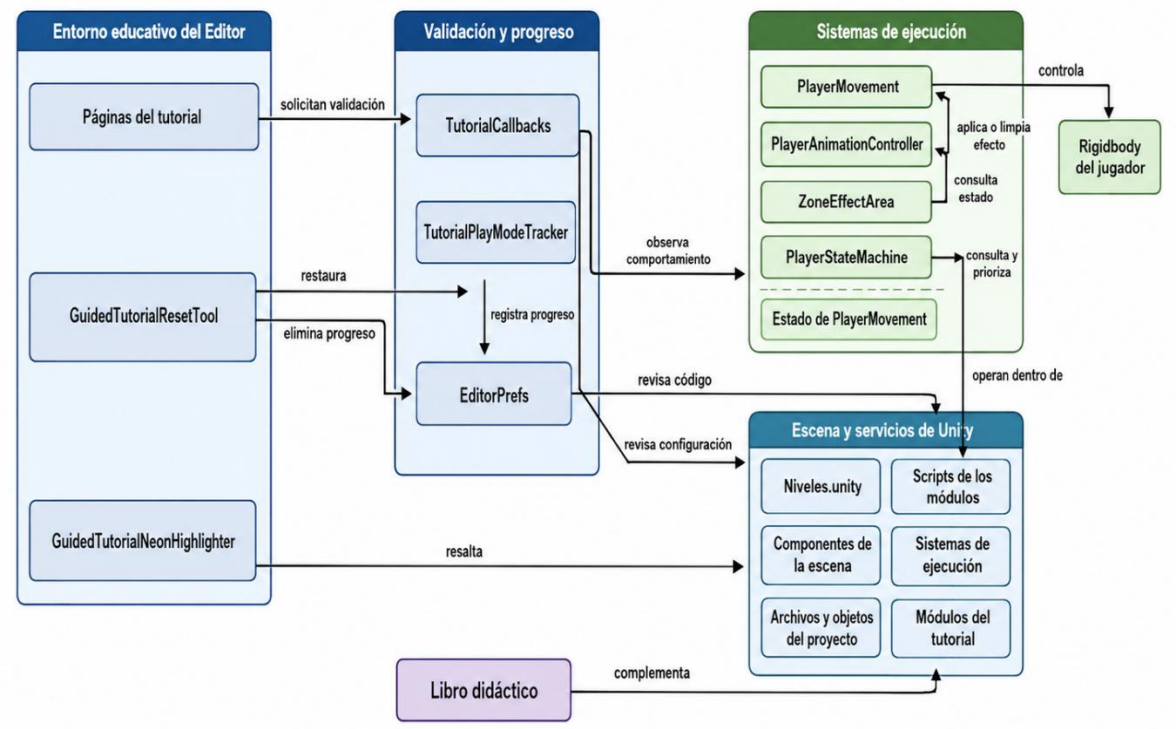
Lógica en Juego se organizó como un sistema educativo híbrido en el que la ejecución de las mecánicas, la guía dentro del Editor y el contenido didáctico cumplen funciones complementarias. La escena de práctica contiene al jugador, los niveles y los elementos interactivos; los scripts runtime controlan las mecánicas; y las herramientas de Editor administran las instrucciones, validaciones y avance del usuario.

La arquitectura se distribuye en tres capas. La capa de ejecución reúne la escena Niveles.unity y los componentes que controlan movimiento, animación, zonas, estados, plataformas y reaparición. La capa educativa contiene las páginas del tutorial, los criterios de comprobación, el seguimiento en Play Mode y el registro del progreso. La capa didáctica corresponde al libro, que explica los fundamentos lógicos y matemáticos aplicados en cada módulo.

La relación entre estas capas permite que una misma mecánica sea explicada, implementada, ejecutada y validada dentro de un flujo continuo. La arquitectura general se presenta en la Ilustración 39.

Ilustración 39

Arquitectura general de Lógica en Juego.



Nota. Elaboración propia

4.2. Entorno tecnológico y organización técnica

El desarrollo del sistema requirió herramientas para programar las mecánicas, construir el recorrido tridimensional, crear las actividades educativas y administrar los archivos del proyecto. Además de describir las tecnologías utilizadas, este apartado presenta la ubicación de los recursos, la organización de la escena y la separación del código fuente.

4.2.1. Tecnologías y herramientas utilizadas

El proyecto se desarrolló en Unity 6000.0.49f1 mediante scripts en C#. La representación visual se configuró con Universal Render Pipeline, el seguimiento de cámara

se apoyó en Cinemachine y el recorrido educativo se implementó con In-Editor Tutorials. Git y GitHub Desktop se utilizaron para conservar el historial de cambios y coordinar el trabajo sobre el repositorio.

Tecnologías y herramientas utilizadas en Lógica en Juego

Tabla 17

Tecnologías principales del proyecto.

Tecnología o herramienta	Versión o configuración	Aplicación en el proyecto
Unity	6000.0.49f1	Motor y Editor utilizados para la escena, los componentes, la física, la animación y las herramientas educativas.
C#	Integrado con Unity	Lenguaje empleado en los scripts runtime y en las herramientas del Editor.
Universal Render Pipeline	17.0.4	Pipeline gráfico utilizado por materiales, iluminación, cámaras y volumen global.
Cinemachine	3.1.6	Control de la cámara de seguimiento del jugador dentro del recorrido.
In-Editor Tutorials	4.1.1	Páginas guiadas, criterios, progreso y navegación del entorno educativo.
Git y GitHub Desktop	Control de versiones	Registro del desarrollo y coordinación del repositorio.

Nota. Elaboración propia

4.2.2. Estructura de carpetas y recursos

La organización de Assets separa los recursos del escenario, la lógica educativa, los materiales visuales y la configuración general. Esta distribución facilita localizar los elementos asociados con la ejecución y evita mezclar las herramientas de tutorial con los recursos utilizados directamente por la escena.

Tabla 18

Estructura principal de carpetas.

Carpeta	Contenido y función
Assets/Escenario-Items_JG/Items_Logica_Juego	Escena, personaje, plataformas, modelos, materiales, animaciones y scripts del recorrido.

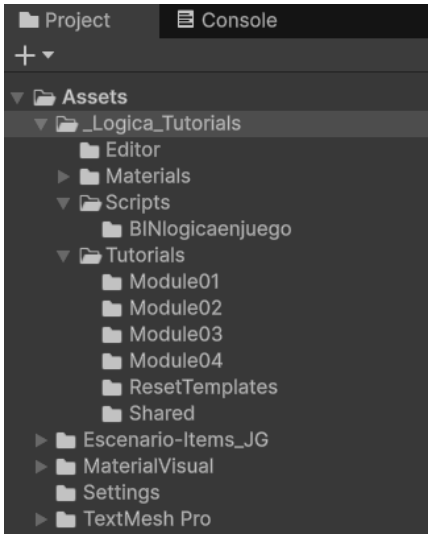
Assets/_Logica_Tutorials	Páginas, scripts educativos, plantillas de reinicio, materiales y recursos del tutorial.
Assets/MaterialVisual	Imágenes y videos utilizados como apoyo dentro de las actividades.
Assets/Settings	Configuraciones de URP y recursos generales del proyecto.
Packages	Manifiesto y versiones de los paquetes instalados.
ProjectSettings	Configuración del Editor, entrada, calidad, gráficos y escena de ejecución.

Nota. Elaboración propia

La estructura visible en la ventana Project se presenta en la Ilustración 40.

Ilustración 40

Organización de carpetas y recursos del proyecto.



Nota. Elaboración propia

4.2.3. Organización de la escena y elementos reutilizables

La ejecución se concentra en Niveles.unity. Dentro de la jerarquía se encuentran las raíces Nivel_1, Nivel_2, Nivel_3 y Nivel_4, además del objeto Player, las cámaras y el volumen global. Esta disposición mantiene los cuatro módulos dentro de un mismo recorrido y permite conservar las referencias entre el jugador, las zonas y las herramientas educativas durante toda la sesión.

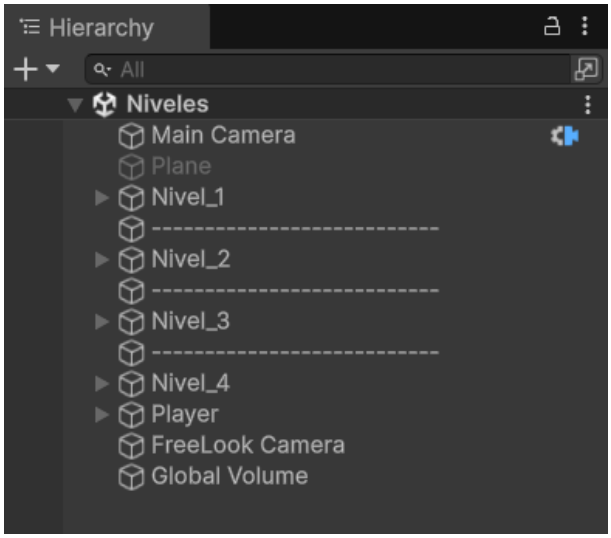
Los elementos repetitivos del escenario, como plataformas, rampas, bordes y piezas del recorrido, se organizaron mediante prefabs. El personaje también dispone de una

configuración reutilizable que reúne el modelo y los componentes necesarios para las mecánicas.

La relación entre la jerarquía, la escena y los componentes principales se muestra en la Ilustración 41.

Ilustración 41

Organización de Niveles.unity y sus elementos principales.



Nota. Elaboración propia

4.2.4. Organización del código fuente

Los scripts se distribuyeron según su función. La lógica runtime contiene los comportamientos que actúan durante la ejecución, mientras que las herramientas del tutorial se alojan en carpetas de Editor o Shared según el alcance requerido. Esta separación evita que las utilidades exclusivas del Editor intervengan en la lógica de las mecánicas.

Las clases educativas relacionadas con las mecánicas utilizan el namespace BINlogicaenjuego. Los campos configurables se exponen principalmente mediante SerializeField y los nombres de clases y métodos mantienen una convención consistente. Los marcadores M1, M2, M3 y M4 enlazan los scripts base con las páginas, los validadores y las plantillas de reinicio.

4.3. Arquitectura de clases y componentes

La arquitectura de clases separa el control del jugador, la interacción con el entorno y las herramientas educativas. PlayerMovement funciona como punto principal de las mecánicas, mientras que las demás clases consultan su información o aplican modificaciones sobre su comportamiento.

Tabla 19

Responsabilidad de las clases principales.

Clase	Ámbito	Responsabilidad
PlayerMovement	Runtime	Gestiona movimiento, sprint, salto, dash, detección del suelo y efectos externos.
PlayerAnimationController	Runtime	Actualiza los parámetros del Animator a partir del estado del jugador.
ZoneEffectArea	Runtime	Detecta la interacción con las zonas y aplica o retira sus efectos.
PlayerStateMachine	Runtime	Interpreta las condiciones del jugador y establece el estado principal.
Platform	Runtime	Controla el desplazamiento de las plataformas móviles.
PlayerRespawnController	Runtime	Administra el punto de reaparición y devuelve al jugador después de una caída o contacto peligroso.
ElectricDeathZone	Runtime	Detecta el contacto con zonas de daño y activa la reaparición.
RespawnCheckpoint	Runtime	Actualiza el punto desde el que reaparece el jugador.
TutorialCallbacks	Editor	Comprueba scripts, componentes y configuraciones vinculadas con las actividades.
TutorialPlayModeTracker	Editor	Observa el comportamiento de las mecánicas durante Play Mode.
GuidedTutorialResetTool	Editor	Restablece el progreso y las plantillas educativas.
GuidedTutorialNeonHighlighter	Editor	Resalta archivos y objetos relacionados con la página activa.

Nota. Elaboración propia

4.3.1. Relación entre clases y componentes

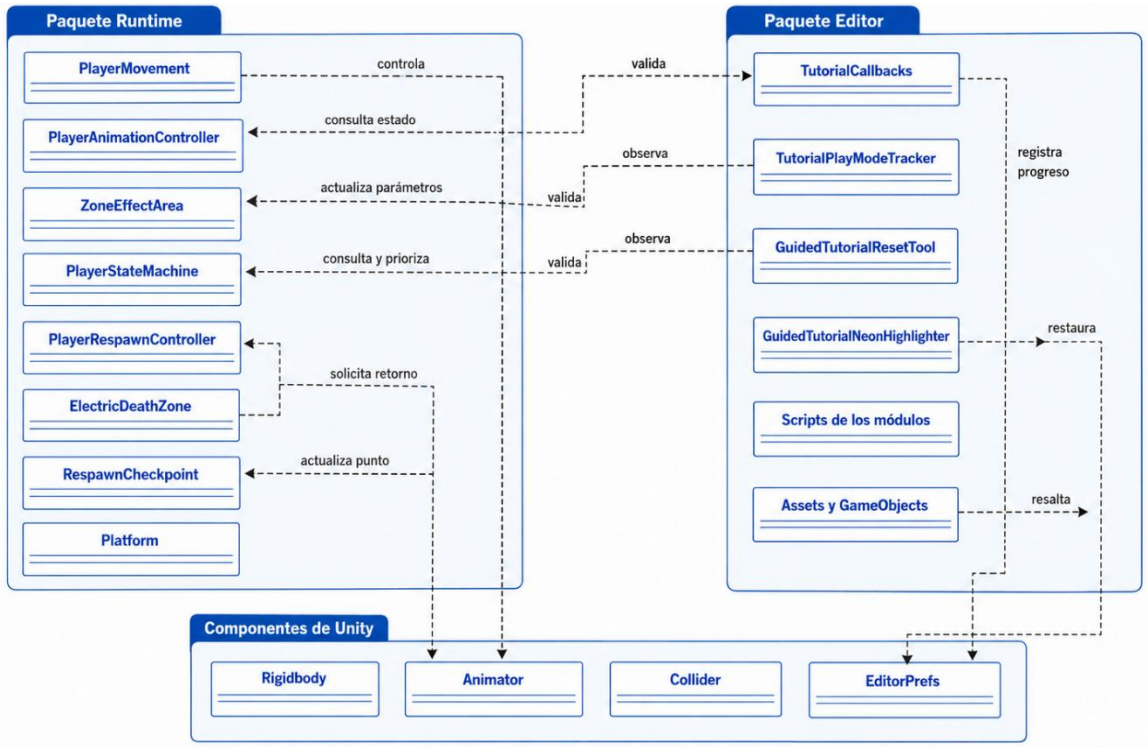
PlayerMovement se encuentra asociado con el objeto Player y utiliza Rigidbody para aplicar el desplazamiento físico. PlayerAnimationController y PlayerStateMachine consultan las propiedades expuestas por este controlador, mientras que ZoneEffectArea se comunica con él cuando el jugador entra, permanece o sale de una zona.

Las herramientas educativas observan la estructura y el comportamiento sin sustituir la ejecución de las mecánicas. TutorialCallbacks valida la implementación y la configuración; TutorialPlayModeTracker reconoce acciones durante la ejecución; GuidedTutorialNeonHighlighter orienta visualmente; y GuidedTutorialResetTool devuelve el proyecto a su estado inicial.

Las dependencias principales se representan en la Ilustración 42.

Ilustración 42

Relación entre las clases de ejecución y las herramientas educativas.



Nota. Elaboración propia

4.4. Proceso de desarrollo de los módulos educativos

El proyecto se desarrolló mediante una metodología en cascada adaptada. El trabajo avanzó desde la definición de cada mecánica hasta su implementación, comprobación y

transformación en una actividad educativa. La elaboración del capítulo correspondiente del libro se realizó después de consolidar el funcionamiento del módulo en Unity.

La validación permitió regresar a las etapas técnicas cuando se detectaron diferencias entre el comportamiento esperado y el resultado obtenido, sin modificar el orden general del proceso. La secuencia aplicada se presenta en la Ilustración 43.

Ilustración 43

Proceso de desarrollo de los módulos mediante la metodología en cascada.

Nota. Elaboración propia

4.4.1. Desarrollo técnico de las mecánicas

Para cada módulo se definieron el comportamiento esperado, las entradas, las variables y los componentes involucrados. Posteriormente, la mecánica se programó dentro de Niveles.unity y se relacionó con el objeto Player o con los elementos del escenario que correspondían a su funcionamiento.

Las pruebas en Play Mode se realizaron antes de construir el contenido educativo. Esta comprobación permitió corregir la respuesta física, las condiciones de activación y la comunicación entre componentes sobre una versión funcional de la mecánica.

4.4.2. Adaptación al entorno guiado y validación

Después de comprobar cada mecánica, los scripts se adaptaron mediante marcadores y bloques de trabajo asociados con los módulos M1, M2, M3 y M4. Sobre esta base se elaboraron las páginas, las instrucciones, el resaltado contextual y los criterios de validación.

La adaptación mantuvo la infraestructura necesaria para ejecutar el proyecto y convirtió determinadas operaciones en actividades progresivas. Las comprobaciones revisaron tanto la configuración de Unity como el comportamiento producido durante Play Mode.

4.4.3. Elaboración del libro didáctico

El contenido del libro se redactó a partir de las mecánicas implementadas y validadas. Esta secuencia permitió explicar los fundamentos lógicos y matemáticos sobre comportamientos ya comprobados y mantener correspondencia entre el capítulo, la actividad guiada y el resultado visible en Unity. El proceso de elaboración y la relación establecida entre el libro y los módulos del entorno se explican con mayor detalle en el apartado correspondiente.

4.5. Programación de las mecánicas

Las mecánicas se organizaron en cuatro módulos progresivos. Cada módulo amplía el comportamiento construido previamente y se relaciona con una sección específica del recorrido, una actividad del entorno guiado y un capítulo del libro didáctico.

4.5.1. Módulo 1: movimiento y sprint

El primer módulo implementa el desplazamiento básico del jugador dentro del recorrido tridimensional. La mecánica permite controlar el movimiento sobre los ejes X y Z y aumentar temporalmente la velocidad mediante la activación del sprint. Su funcionamiento

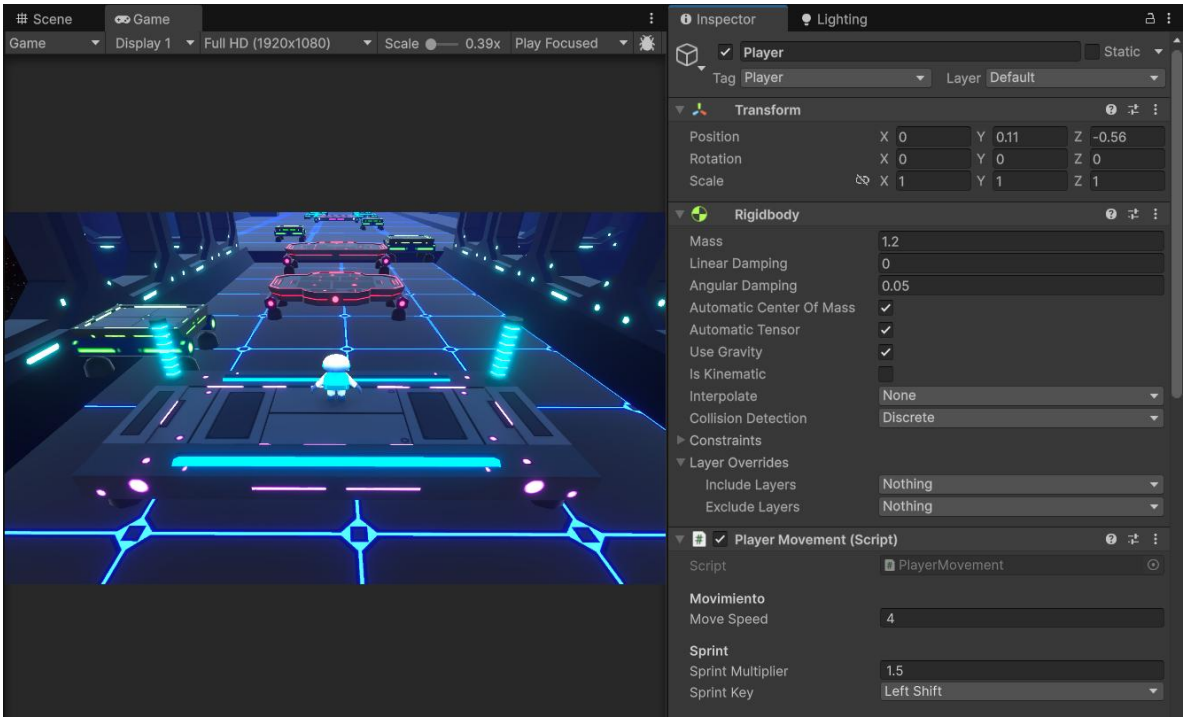
se concentra en la clase PlayerMovement, que utiliza el componente Rigidbody para aplicar el desplazamiento físico.

Las entradas horizontal y vertical se combinan en un vector de dirección. Este vector se normaliza antes de multiplicarlo por la velocidad, con el propósito de mantener un desplazamiento uniforme cuando el jugador se mueve de forma diagonal. La velocidad aplicada depende del estado del sprint: se utiliza el valor base durante el movimiento normal y un valor superior mientras se mantiene activa la tecla correspondiente.

La relación entre la configuración del objeto Player y el comportamiento observado dentro del primer nivel se presenta en la Ilustración 44.

Ilustración 44

Relación entre la configuración del objeto Player y las mecánicas de movimiento y sprint.



Nota. Elaboración propia

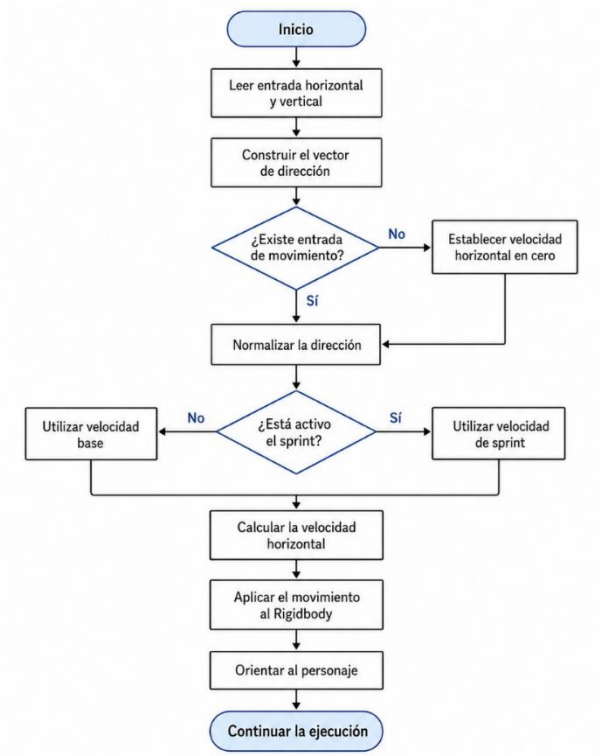
En esta configuración, PlayerMovement procesa la dirección y selecciona la velocidad, mientras que Rigidbody transforma esos valores en el desplazamiento del personaje dentro de la escena.

La lectura de la entrada se realiza en Update, mientras que el movimiento físico se aplica en FixedUpdate. Esta separación permite procesar continuamente las acciones del usuario y actualizar el Rigidbody de acuerdo con el ciclo de física de Unity. Al modificar la velocidad se conserva la componente vertical, evitando que el movimiento horizontal interfiera con la gravedad o con la mecánica de salto incorporada posteriormente.

El personaje también se orienta hacia la dirección del desplazamiento mediante Quaternion.LookRotation. La rotación solo se actualiza cuando existe una entrada válida, lo que impide calcular una orientación a partir de un vector sin dirección. La secuencia de funcionamiento se sintetiza en la Ilustración 45.

Ilustración 45

Flujo general de las mecánicas de movimiento y sprint.



Nota. Elaboración propia

4.5.2. Módulo 2: salto y dash

El segundo módulo amplía el control del personaje mediante las mecánicas de salto y dash. Ambas se incorporan en PlayerMovement y utilizan el Rigidbody configurado previamente, por lo que mantienen una relación directa con el sistema de movimiento desarrollado en el primer módulo.

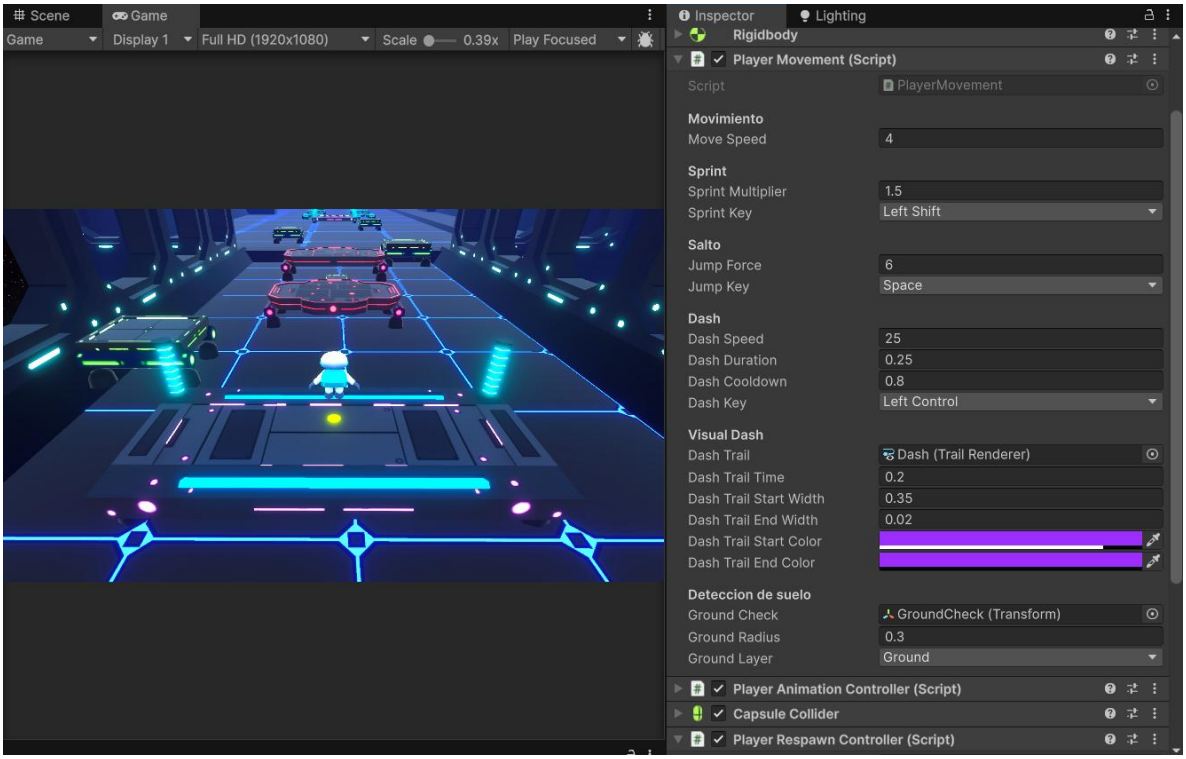
El salto modifica la velocidad vertical cuando se detecta contacto con una superficie válida. La comprobación se realiza mediante GroundCheck y evita nuevas activaciones mientras el jugador permanece en el aire. Durante la acción se conserva el control horizontal, permitiendo ajustar la trayectoria sin interrumpir el desplazamiento vertical.

El dash aplica una velocidad elevada durante un intervalo breve. Su ejecución se controla mediante una duración y un tiempo de reutilización, mientras que una estela visual permite reconocer el desplazamiento dentro de la escena.

La relación entre GroundCheck, los parámetros configurados en PlayerMovement y la ejecución de ambas mecánicas se presenta en la Ilustración 46.

Ilustración 46

Relación entre la configuración del objeto Player y las mecánicas de salto y dash.

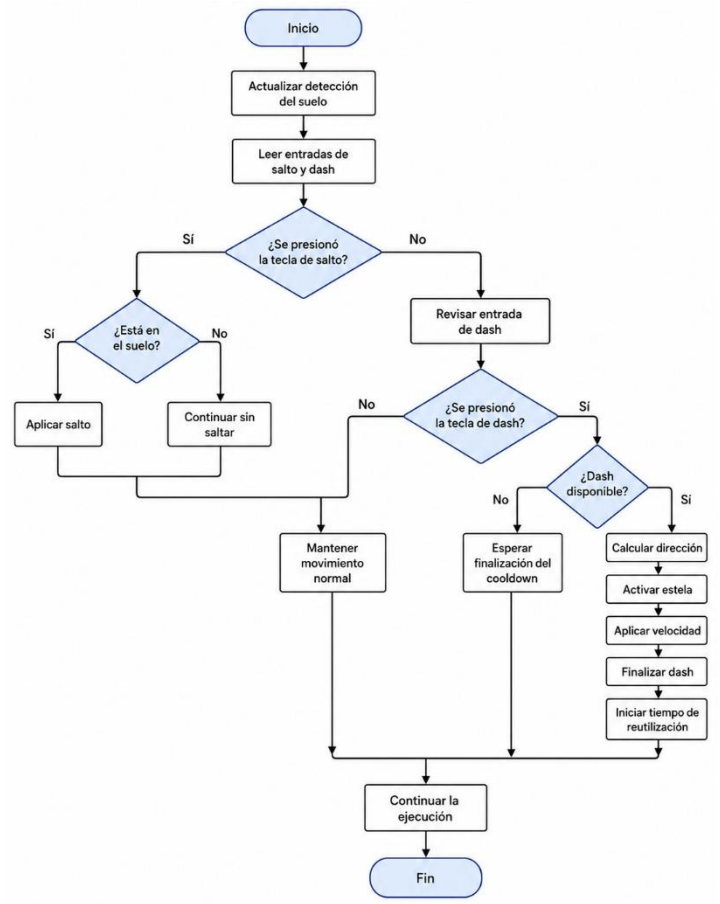


Nota. Elaboración propia

GroundCheck condiciona la disponibilidad del salto, mientras que los valores del dash determinan su rapidez, duración y reutilización. La secuencia lógica del módulo se representa en la Ilustración 47.

Ilustración 47

Flujo general de las mecánicas de salto y dash.



Nota. Elaboración propia

4.5.3. Módulo 3: zonas del entorno

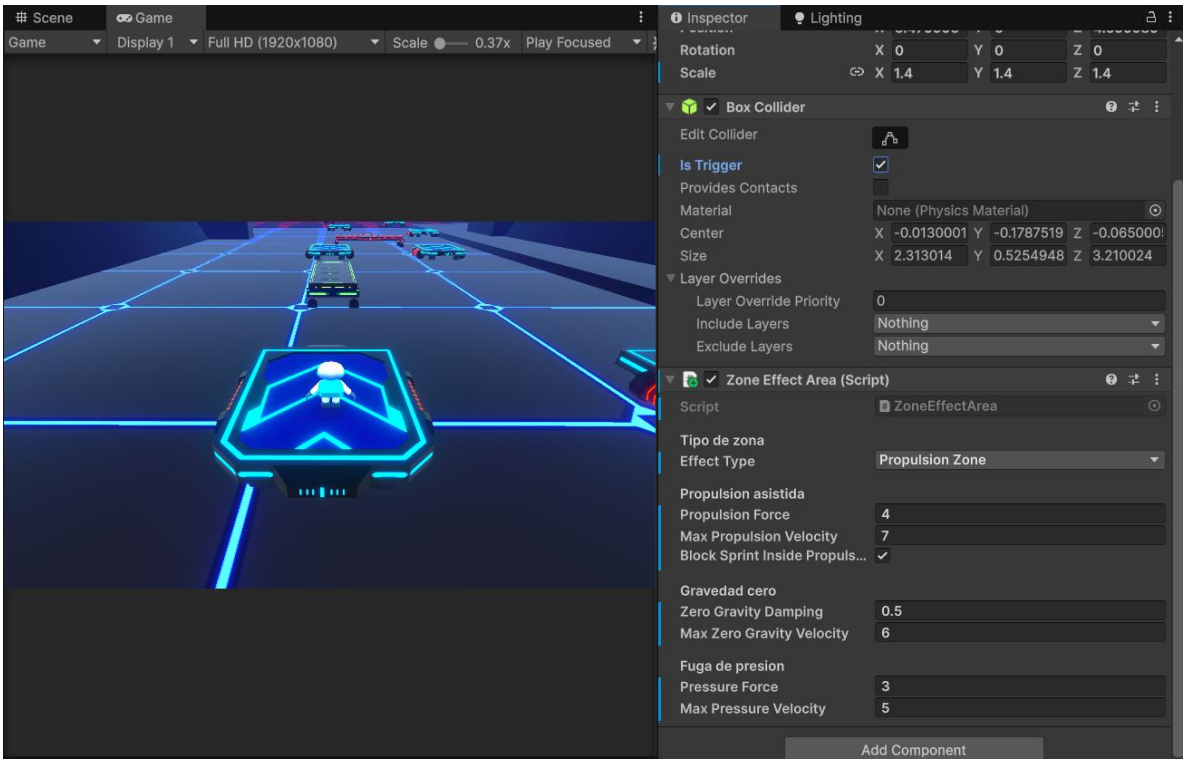
El tercer módulo incorpora áreas capaces de modificar temporalmente el comportamiento del jugador. Su funcionamiento se concentra en ZoneEffectArea, componente encargado de identificar el efecto configurado y comunicarse con PlayerMovement durante la interacción.

Las zonas se configuraron como áreas de impulso, gravedad cero y presión. Cada objeto utiliza un Collider marcado como trigger para delimitar el espacio de interacción y ejecutar la respuesta correspondiente cuando el jugador entra, permanece o sale del área.

La relación entre el volumen de la zona, su configuración en el Inspector y la trayectoria del jugador se presenta en la Ilustración 48.

Ilustración 48

Relación entre la configuración de ZoneEffectArea y la interacción del jugador con el entorno.

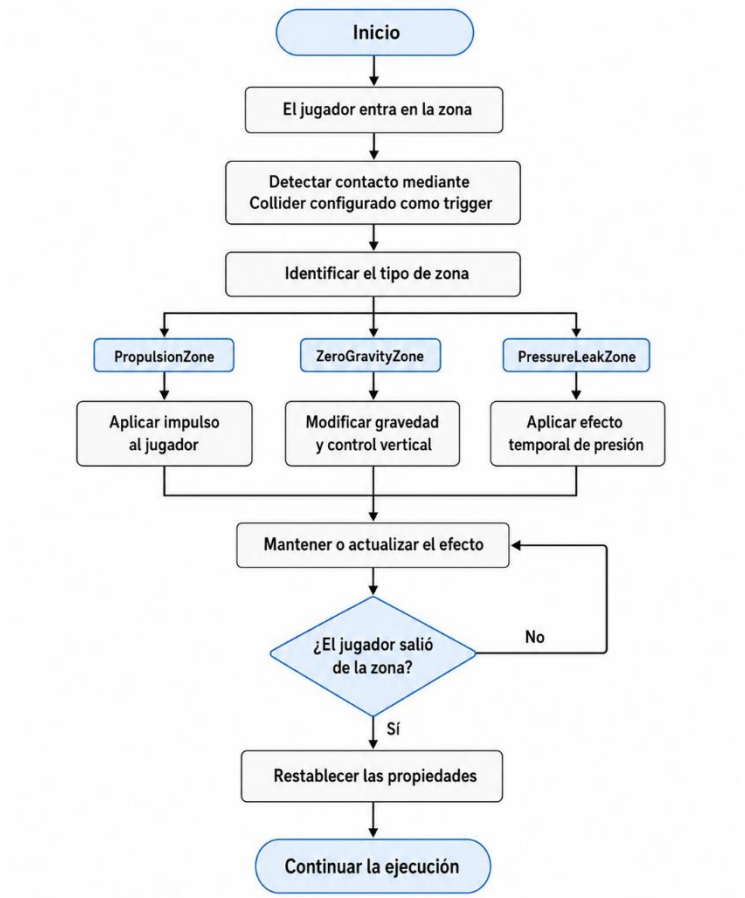


Nota. Elaboración propia

El trigger realiza la detección y ZoneEffectArea determina la modificación que debe aplicarse. Al abandonar la zona, las propiedades afectadas regresan a su condición normal. El flujo completo se muestra en la Ilustración 49.

Ilustración 49

Flujo general de interacción con las zonas del entorno.



Nota. Elaboración propia

4.5.4. Módulo 4: máquina de estados

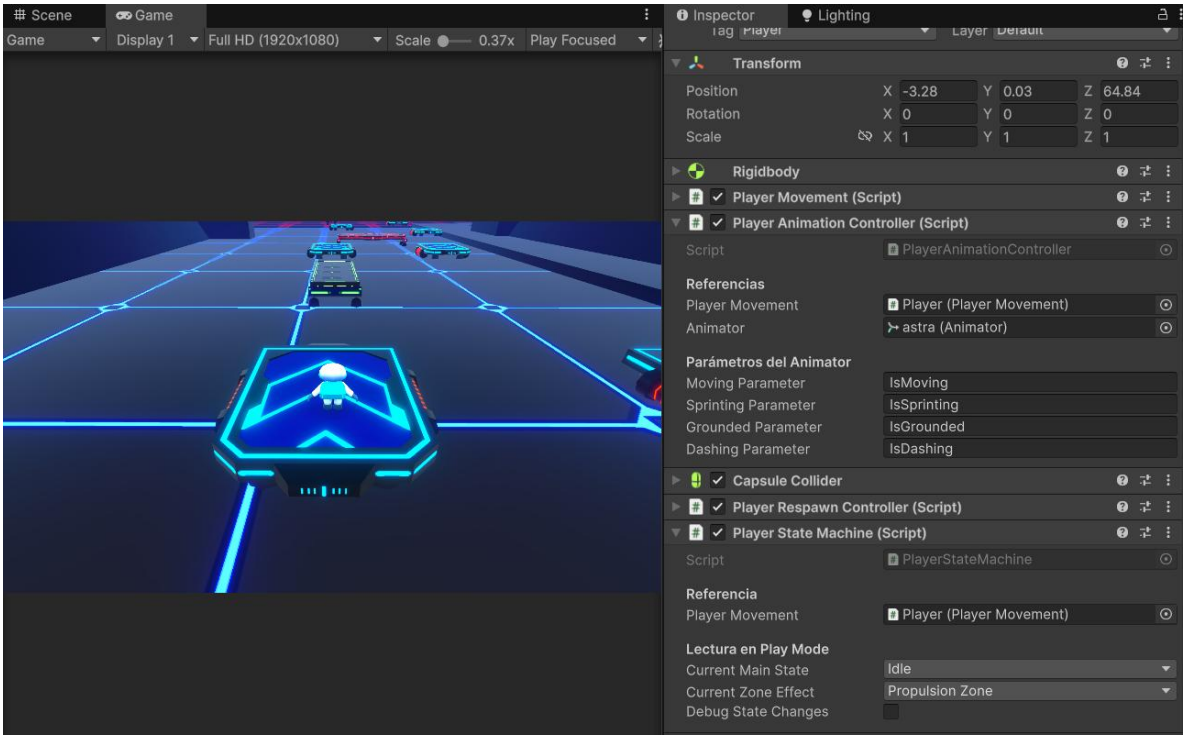
El cuarto módulo integra los comportamientos anteriores mediante PlayerStateMachine. Este componente consulta la información generada por PlayerMovement y establece el estado principal del personaje de acuerdo con las condiciones presentes durante la ejecución.

La máquina contempla los estados Idle, Moving, Sprinting, Airborne, Dashing y ZeroGravityDrift. Cuando varias condiciones coinciden, se aplica un orden de prioridad para conservar una representación única y coherente del comportamiento actual.

PlayerStateMachine no ejecuta nuevamente las mecánicas, sino que interpreta la información producida por ellas. La relación entre ambos componentes y el estado visible en la escena se presenta en la Ilustración 50.

Ilustración 50

Relación entre PlayerMovement, PlayerStateMachine y el estado del jugador.

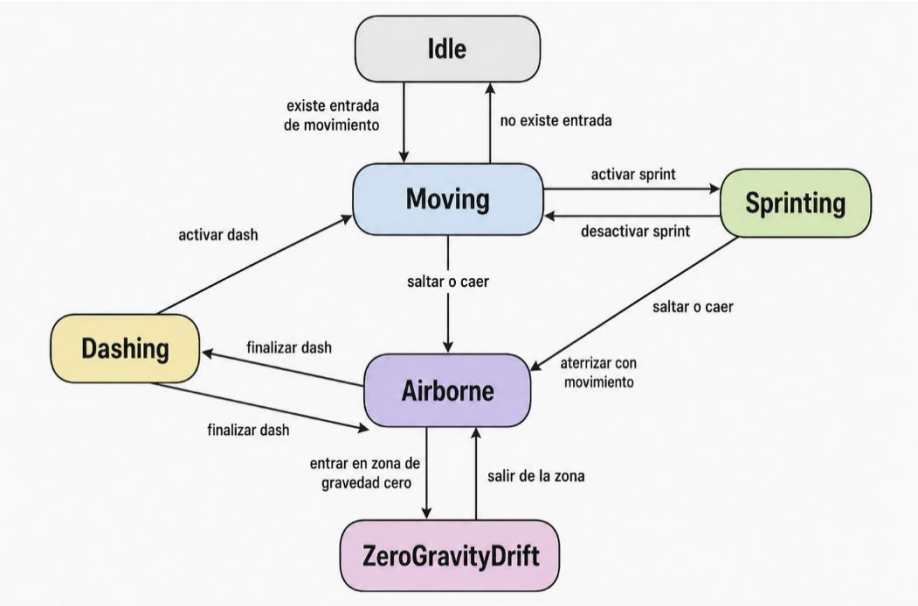


Nota. Elaboración propia

La representación de los estados facilita que la animación y las herramientas educativas consulten el comportamiento del jugador sin intervenir directamente en el control físico. Las transiciones principales se muestran en la Ilustración 51.

Ilustración 51

Estados y transiciones principales del jugador.



Nota. Elaboración propia

4.6. Sistema de tutoriales, validación y progreso

Una vez implementadas las mecánicas, estas se integraron en un recorrido guiado mediante In-Editor Tutorials. El sistema presenta las actividades dentro del propio Editor, dirige al usuario hacia los archivos y objetos relacionados y comprueba el resultado antes de habilitar el avance.

4.6.1. Organización de las páginas y orientación contextual

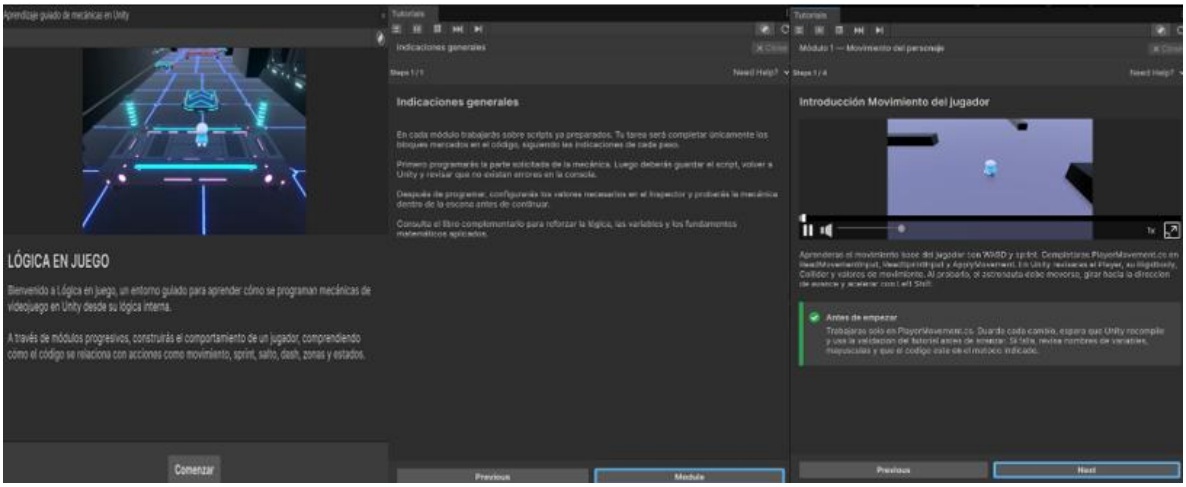
El recorrido comienza con la página de bienvenida, continúa con las indicaciones generales y conduce a la selección de módulos. Cada conjunto contiene páginas introductorias, actividades prácticas y comprobaciones asociadas con la mecánica correspondiente.

GuidedTutorialNeonHighlighter identifica la página activa y resalta el elemento relacionado en las ventanas Project o Hierarchy. Esta guía visual conecta la instrucción con el lugar exacto del proyecto donde se realiza la actividad.

La organización de las páginas y la orientación contextual se presentan en la Ilustración 52.

Ilustración 52

Organización de las páginas y orientación contextual del entorno guiado.



Nota. Elaboración propia

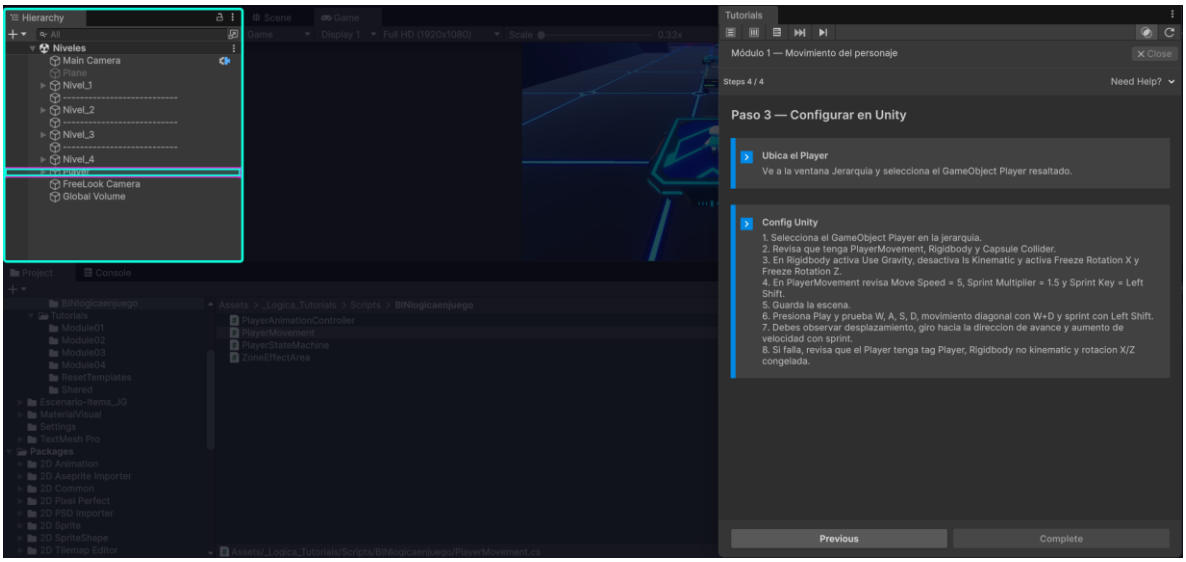
4.6.2. Validación de las actividades

TutorialCallbacks reúne las comprobaciones utilizadas por las páginas. El sistema revisa la estructura esperada de los scripts, la presencia de componentes y los valores configurados en el Inspector. Las validaciones de código reconocen los fragmentos requeridos, aunque existan variaciones de espacios o saltos de línea.

Las comprobaciones de configuración abarcan Rigidbody, GroundCheck, los parámetros del dash, los colliders utilizados como trigger, el tipo de zona y la referencia entre PlayerStateMachine y PlayerMovement. La relación entre la instrucción, la configuración y el criterio completado se muestra en la Ilustración 53.

Ilustración 53

Relación entre la actividad guiada, la configuración y su validación.



Nota. Elaboración propia

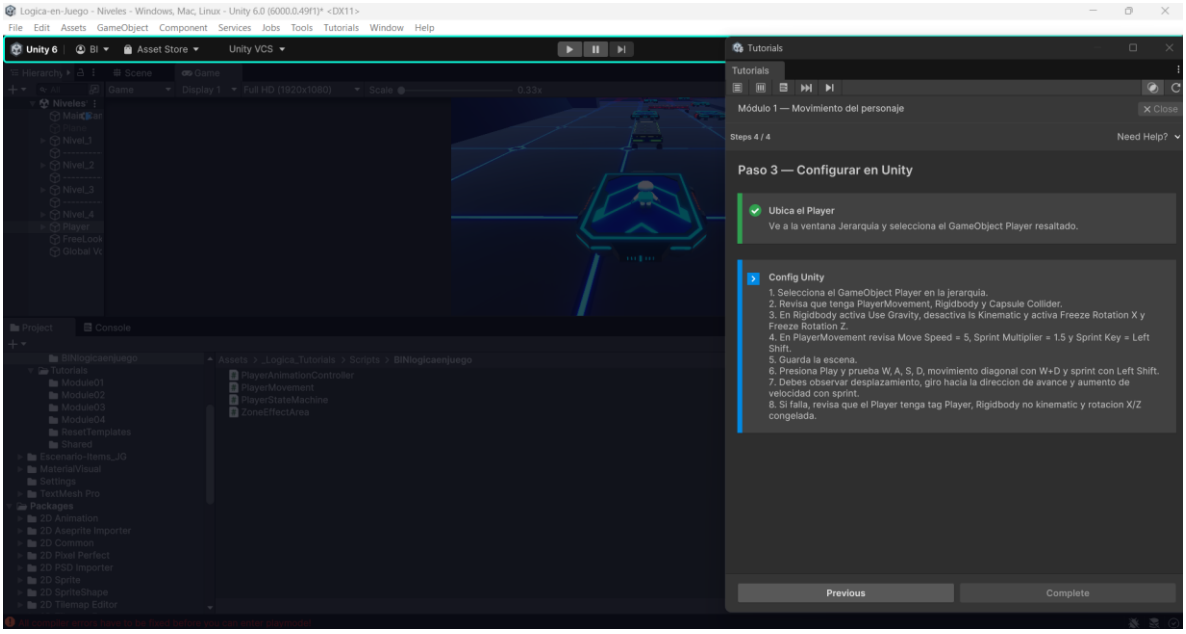
4.6.3. Seguimiento durante Play Mode

TutorialPlayModeTracker observa los comportamientos que solo pueden comprobarse durante la ejecución. El componente localiza los scripts relacionados y consulta sus estados para reconocer el movimiento, el sprint, el salto, el dash, los efectos de zona y las transiciones de la máquina de estados.

La actividad se completa cuando la acción esperada ocurre dentro de la escena. Esta relación entre el comportamiento y el criterio de la página se presenta en la Ilustración 54.

Ilustración 54

Seguimiento y validación de una mecánica durante Play Mode.



Nota. Elaboración propia

4.6.4. Progreso, desbloqueo y reinicio

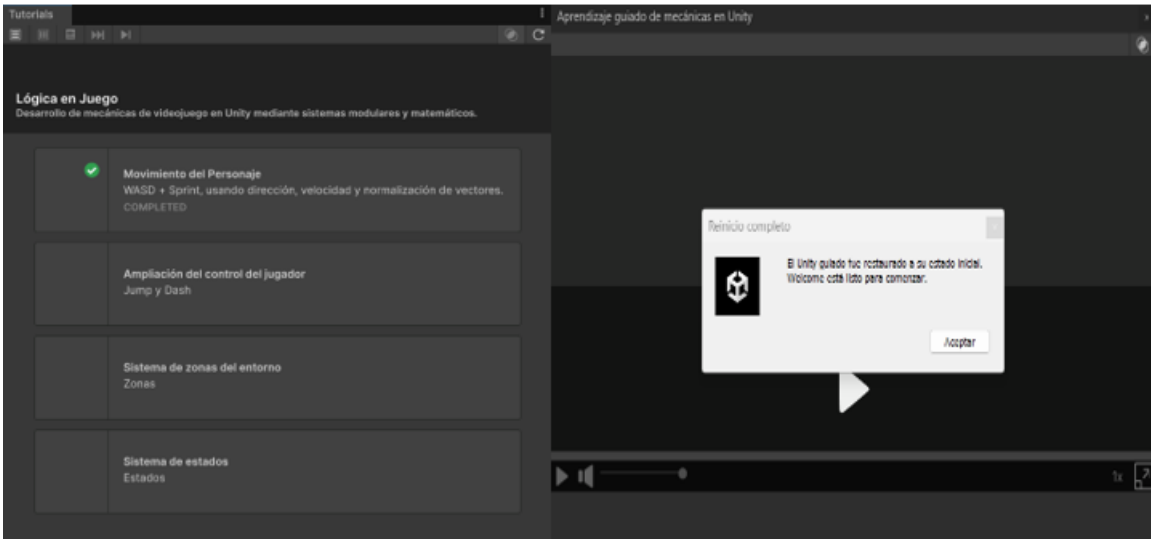
El progreso se almacena mediante EditorPrefs. Al completar las actividades de un módulo, el sistema registra el resultado y habilita el siguiente, manteniendo la secuencia de aprendizaje definida para el recorrido. La información se conserva entre sesiones del Editor.

GuidedTutorialResetTool elimina las claves de avance y restaura las plantillas educativas de PlayerMovement, ZoneEffectArea y PlayerStateMachine. El reinicio devuelve el proyecto a su estado inicial y permite repetir el proceso completo.

La relación entre el bloqueo, el desbloqueo y el reinicio se muestra en la Ilustración 55.

Ilustración 55

Registro del progreso, desbloqueo de módulos y reinicio del entorno guiado.



Nota. Elaboración propia

4.7. Integración del libro didáctico con los módulos

El libro didáctico constituye el componente conceptual del sistema y mantiene correspondencia con los cuatro módulos desarrollados en Unity. Cada capítulo explica la lógica y los fundamentos matemáticos de una mecánica que posteriormente se implementa y valida dentro del entorno guiado.

El libro y Unity cumplen funciones complementarias. El recurso escrito desarrolla los conceptos y relaciona las decisiones de programación con el comportamiento esperado; el entorno guiado conduce la actividad, señala los elementos del proyecto y comprueba el resultado.

Tabla 20

Complementación entre el libro y el Unity guiado.

Módulo	Contenido del libro	Aplicación en Unity
Módulo 1: movimiento y sprint	Dirección, magnitud, normalización y velocidad.	Desplazamiento sobre X y Z, orientación y aumento temporal de velocidad.

Módulo 2: salto y dash	Impulso vertical, detección del suelo, duración y reutilización.	Salto condicionado por GroundCheck y desplazamiento rápido con estela.
Módulo 3: zonas del entorno	Condiciones de entrada, permanencia y salida; modificación temporal de propiedades.	Zonas configuradas mediante triggers que alteran el comportamiento del jugador.
Módulo 4: máquina de estados	Estados, condiciones, transiciones y prioridades.	Interpretación conjunta del movimiento, salto, dash y efectos del entorno.

Nota. Elaboración propia

La integración se observa cuando un mismo contenido aparece como explicación en el libro, actividad dentro del tutorial y resultado visible en la escena. Esta relación se presenta en la Ilustración 56.

Ilustración 56

Integración entre el contenido del libro, la actividad guiada y el resultado en Unity.

Nota. Elaboración propia

4.7.1. Flujo integrado de aprendizaje

El flujo comienza con la consulta del capítulo correspondiente del libro didáctico. Posteriormente, el usuario accede a la actividad del módulo, identifica el script o componente señalado, realiza la configuración requerida y ejecuta la escena en Play Mode. Los componentes TutorialCallbacks y TutorialPlayModeTracker verifican el cumplimiento de la actividad, mientras que EditorPrefs almacena el progreso alcanzado y habilita el contenido siguiente.

4.8. Pruebas técnicas y verificación del funcionamiento

4.8.1. Pruebas funcionales de las mecánicas

Las mecánicas se comprobaron dentro de Niveles.unity mediante Play Mode. En el primer módulo se revisaron el desplazamiento, la orientación y el cambio de velocidad producido por el sprint. En el segundo se verificaron la detección del suelo, el salto, el dash, su duración y su tiempo de reutilización.

En el tercer módulo se comprobó la detección del jugador dentro de las zonas, la aplicación del efecto configurado y la restauración de las propiedades al abandonar el área. En el cuarto se verificó que PlayerStateMachine identificara el estado correspondiente según la acción y las condiciones activas.

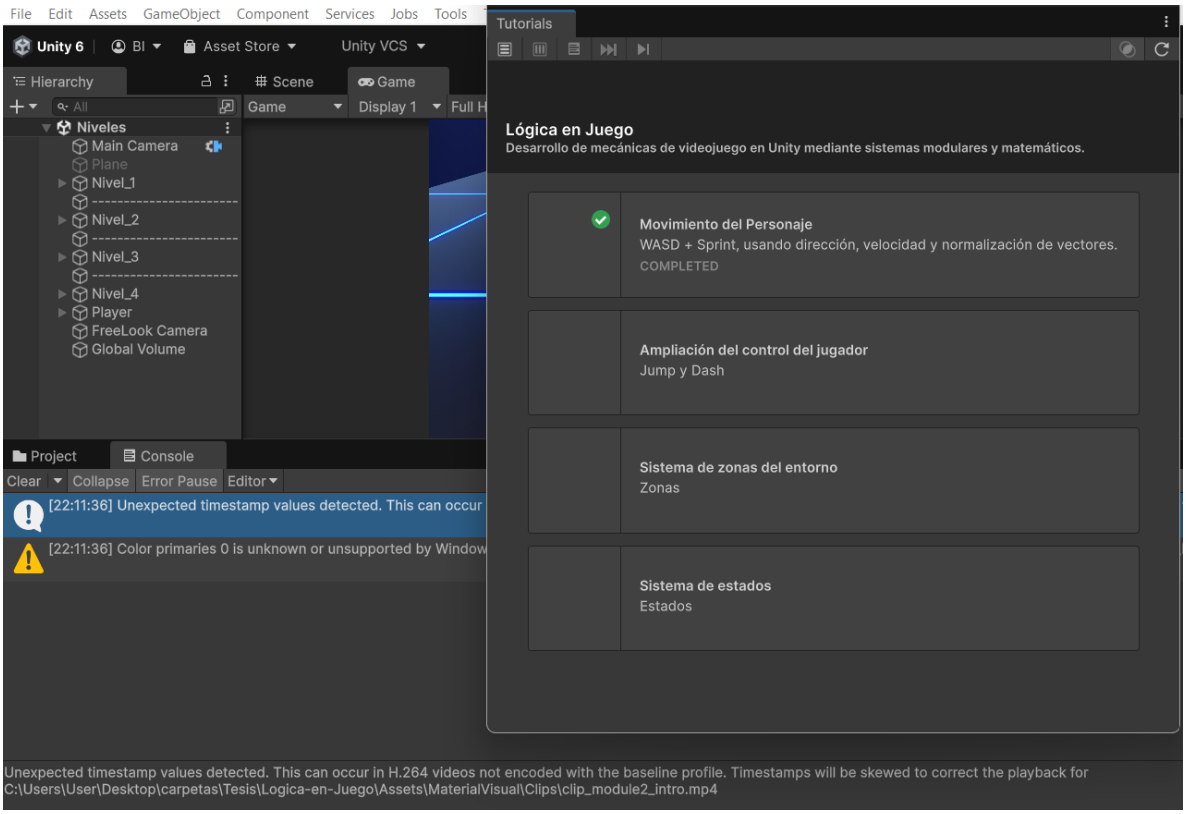
4.8.2. Verificación del entorno guiado

El recorrido se revisó desde la bienvenida hasta la finalización de los módulos. Las páginas se comprobaron junto con sus instrucciones, elementos resaltados y criterios. Las actividades de Play Mode reconocieron las acciones esperadas y actualizaron el estado de la página.

También se verificaron el registro del progreso, el desbloqueo secuencial, la persistencia entre sesiones y la restauración mediante GuidedTutorialResetTool. La prueba conjunta del sistema se presenta en la Ilustración 57.

Ilustración 57

Verificación conjunta de las mecánicas y del entorno guiado.



Nota. Elaboración propia

4.8.3. Estabilidad técnica y compilación

Durante la implementación se aplicaron medidas orientadas a conservar una ejecución estable. La lectura de entradas se mantuvo en Update y las operaciones físicas en FixedUpdate. Las direcciones se normalizaron antes de aplicar la velocidad y las referencias principales se obtuvieron durante la inicialización de los componentes.

PlayerAnimationController utiliza identificadores generados mediante Animator.StringToHash, mientras que el resaltado contextual actualiza sus comprobaciones mediante intervalos controlados. La concentración de los cuatro módulos en una sola escena mantuvo disponibles las referencias necesarias durante todo el recorrido.

La compilación de los scripts y la ejecución en Play Mode permitieron comprobar el funcionamiento conjunto del jugador, las plataformas, las zonas, la máquina de estados y las herramientas educativas sin errores que impidieran iniciar la escena.

4.9. Conclusiones y recomendaciones

4.9.1. Conclusiones

La organización del sistema mediante componentes con responsabilidades diferenciadas permitió integrar las mecánicas, las validaciones y el control de avance sin concentrar todas las funciones en un único script.

El desarrollo progresivo de los módulos facilitó la comprobación individual de las mecánicas antes de integrarlas en una experiencia continua, lo que permitió corregir errores y mantener correspondencia con las actividades guiadas.

La elaboración del libro a partir de mecánicas previamente implementadas permitió mantener coherencia entre la explicación conceptual, la actividad realizada y el resultado observado en Unity.

4.9.2. Recomendaciones

Comprobar la compatibilidad de escenas, paquetes, scripts y herramientas de tutoriales antes de migrar el proyecto a una versión diferente de Unity.

Mantener la separación de responsabilidades al incorporar nuevas funcionalidades, evitando reunir la lógica de las mecánicas, las validaciones y la interfaz en un mismo componente.

Actualizar de manera conjunta el entorno guiado y el libro didáctico cuando se modifiquen variables, instrucciones, componentes o criterios de validación.

Referencia Bibliográfica

[1] Unity, «Player movement», Unity Learn. [En línea]. Disponible en: https://learn.unity.com/tutorial/player-movement

[2] R. Nystrom, *Game programming patterns*. s.l.: genever benning, 2014.

[3] R. Mathew, S. I. Malik, y R. M. Tawafak, «Teaching Problem Solving Skills using an Educational Game in a Computer Programming Course», *Inform. Educ.*, vol. 18, n.º 2, pp. 359-373, oct. 2019, doi: 10.15388/infedu.2019.17.

[4] C. Kazimoglu, M. Kiernan, L. Bacon, y L. Mackinnon, «A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming», *Procedia - Soc. Behav. Sci.*, vol. 47, pp. 1991-1999, 2012, doi: 10.1016/j.sbspro.2012.06.938.

[5] J. M. Wing, «Computational thinking», *Commun. ACM*, vol. 49, n.º 3, pp. 33-35, mar. 2006, doi: 10.1145/1118178.1118215.

[6] D. Weintrop *et al.*, «Defining Computational Thinking for Mathematics and Science Classrooms», *J. Sci. Educ. Technol.*, vol. 25, n.º 1, pp. 127-147, feb. 2016, doi: 10.1007/s10956-015-9581-5.

[7] V. Kuznetsov, M. Moiseienko, N. Moiseienko, B. Rostalny, y A. Kiv, «Using Unity to Teach Game Development»:, en *Proceedings of the 1st Symposium on Advances in Educational Technology*, Kyiv, Ukraine: SCITEPRESS - Science and Technology Publications, 2020, pp. 506-515. doi: 10.5220/0010933400003364.

[8] Н. В. Моісеєнко, М. В. Моісеєнко, В. С. Кузнецов, Б. А. Ростальний, у А. Ю. Ків, «Teaching computer game development with Unity engine: a case study», CEUR Workshop Proceedings, sep. 2023. doi: 10.31812/123456789/8486.

[9] C. D. Hundhausen, S. A. Douglas, y J. T. Stasko, «A Meta-Study of Algorithm Visualization Effectiveness», *J. Vis. Lang. Comput.*, vol. 13, n.º 3, pp. 259-290, jun. 2002, doi: 10.1006/jvlc.2002.0237.

[10] L. E. Margulieux, B. B. Morrison, y A. Decker, «Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples», *Int. J. STEM Educ.*, vol. 7, n.º 1, p. 19, dic. 2020, doi: 10.1186/s40594-020-00222-7.

[11] J. Van De Pol, M. Volman, y J. Beishuizen, «Scaffolding in Teacher–Student Interaction: A Decade of Research», *Educ. Psychol. Rev.*, vol. 22, n.º 3, pp. 271-296, sep. 2010, doi: 10.1007/s10648-010-9127-6.

[12] P. Toukiloglou y S. Xinogalos, «A Systematic Literature Review on Adaptive Supports in Serious Games for Programming», *Information*, vol. 14, n.º 5, p. 277, may 2023, doi: 10.3390/info14050277.

[13] R. Mayer, *Multimedia Learning*, 3.ª ed. Cambridge University Press, 2020. doi: 10.1017/9781316941355.

[14] B. Sobota y E. Pietriková, «The Role of Game Engines in Game Development and Teaching», en *Computer Science for Game Development and Game Development for Computer Science*, B. Sobota y E. Pietriková, Eds., IntechOpen, 2023. doi: 10.5772/intechopen.1002257.

[15] S. Golob y N. Pagnon, «Unity to Teach Game Programming», Bachelor's thesis, Malmö University, Malmö, Sweden, 2024. [En línea]. Disponible en: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1867700

[16] M. Holly, L. Habich, y J. Pirker, «GameDevDojo - An Educational Game for Teaching Game Development Concepts», en *Proceedings of the 19th International Conference on the Foundations of Digital Games*, Worcester MA USA: ACM, may 2024, pp. 1-9. doi: 10.1145/3649921.3649992.

[17] A. Luxton-Reilly *et al.*, «Introductory programming: a systematic literature review», en *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, Larnaca Cyprus: ACM, jul. 2018, pp. 55-106. doi: 10.1145/3293881.3295779.

[18] J. P. D. Silva y I. F. Silveira, «A Systematic Review on Open Educational Games for Programming Learning and Teaching», *Int. J. Emerg. Technol. Learn. IJET*, vol. 15, n.º 09, p. 156, may 2020, doi: 10.3991/ijet.v15i09.12437.

[19] Unity, «Introduction to GameObjects», Unity Manual. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/Manual/GameObjects.html

[20] A. Saravanos y M. X. Curinga, «Simulating the Software Development Lifecycle: The Waterfall Model», *Appl. Syst. Innov.*, vol. 6, n.º 6, p. 108, nov. 2023, doi: 10.3390/asi6060108.

[21] W. Royce, «Managing the Development of Large Software Systems», en *Proceedings of IEEE WESCON*, Los Angeles, CA, USA, 1970. [En línea]. Disponible en: https://www.praxisframework.org/files/royce1970.pdf
[22] I. Sommerville, *Software engineering*, 10. ed. Boston, Munich: Pearson, 2016.
[23] K. Schwaber y J. Sutherland, «La Guía de Scrum: La Guía Definitiva de Scrum: Las Reglas del Juego», Guía, nov. 2020. [En línea]. Disponible en: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf
[24] K. Beck y C. Andres, *Extreme programming explained: embrace change*, 2. ed., 11. print. en The XP Series. Boston: Addison-Wesley, 2012.
[25] K. Beck, «Embracing change with extreme programming», *Computer*, vol. 32, n.º 10, pp. 70-77, oct. 1999, doi: 10.1109/2.796139.
[26] Unity, «Get started with Unity», Unity Manual. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/Manual/get-started-with-unity.html
[27] Unity, «Unity Editor windows and views reference», Unity Manual. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/Manual/editor-windows-views-reference.html
[28] Unity, «Introduction to components», Unity Manual. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/Manual/Components.html
[29] Unity, «MonoBehaviour», Unity Manual. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/Manual/class-MonoBehaviour.html
[30] Unity, «2D game development», Unity Manual. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/Manual/Unity2D.html
[31] Unity, «Unity Learn», Unity Learn. [En línea]. Disponible en: https://learn.unity.com/
[32] Unity, «Unity Editor Software Terms». 2026. [En línea]. Disponible en: https://unity.com/legal/editor-terms-of-service/software
[33] Unity, «System requirements for Unity 6.0», Unity Manual. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/Manual/system-requirements.html
[34] Unity, «Finding package documentation», Unity Manual. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/Manual/upm-docs.html
[35] E. Godot, «Introduction to Godot», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/getting_started/introduction/introduction_to_godot.html
[36] E. Godot, «First look at Godot's interface», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/getting_started/introduction/first_look_at_the_editor.html
[37] E. Godot, «Nodes and Scenes», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/getting_started/step_by_step/nodes_and_scenes.html
[38] E. Godot, «Scripting languages», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/getting_started/step_by_step/scripting_languages.html
[39] E. Godot, «Complying with licenses», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/about/complying_with_licenses.html
[40] E. Godot, «System requirements», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/about/system_requirements.html
[41] Epic Games, «Intro to UE», Epic Developer Community. [En línea]. Disponible en: https://dev.epicgames.com/documentation/unreal-engine/intro-to-ue?lang=en-US
[42] Epic Games, «Game Objects in Unreal Engine», Epic Developer Community. [En línea]. Disponible en: https://dev.epicgames.com/documentation/en-us/unreal-engine/game-objects-in-unreal-engine
[43] Epic Games, «Introduction to Blueprints Visual Scripting in Unreal Engine», Epic Developer Community. [En línea]. Disponible en: https://dev.epicgames.com/documentation/en-us/unreal-engine/introduction-to-blueprints-visual-scripting-in-unreal-engine
[44] Epic Games, «Paper 2D», Epic Developer Community. [En línea]. Disponible en: https://dev.epicgames.com/documentation/unreal-engine/paper-2d-overview-in-unreal-engine?lang=en-US
[45] Epic Games, «Hardware and Software Specifications for Unreal Engine», Epic Developer Community. [En línea]. Disponible en: https://dev.epicgames.com/documentation/unreal-engine/hardware-and-software-specifications-for-unreal-engine
[46] Epic Games, «Unreal Engine (UE5) licensing options», Unreal Engine. [En línea]. Disponible en: https://www.unrealengine.com/license
[47] B. Stroustrup, *The C++ programming language: C++ 11*, 4. ed., 4. print. Upper Saddle River, NJ: Addison-Wesley, 2015.

[48] Epic Games, «Unreal Engine C++ Quick Start», Epic Developer Community. [En línea]. Disponible en: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-cpp-quick-start
[49] Epic Games, «Programming with C++ in Unreal Engine», Epic Developer Community. [En línea]. Disponible en: https://dev.epicgames.com/documentation/en-us/unreal-engine/programming-with-cpp-in-unreal-engine
[50] Microsoft, «A tour of the C# language», Microsoft Learn. [En línea]. Disponible en: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview
[51] Unity, «Garbage collector overview», Unity Manual. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/Manual/performance-garbage-collector.html
[52] Unity, «SerializeField», Unity Scripting API. [En línea]. Disponible en: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/SerializeField.html
[53] E. Godot, «GDScript reference», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/tutorials/scripting/gdscript/gdscript_basics.html
[54] E. Godot, «Script Editor», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/tutorials/editor/script_editor.html
[55] E. Godot, «Static typing in GDScript», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/tutorials/scripting/gdscript/static_typing.html
[56] E. Godot, «GDScript exported properties», Godot Engine documentation. [En línea]. Disponible en: https://docs.godotengine.org/en/4.7/tutorials/scripting/gdscript/gdscript_exports.html
[57] HitHub, «About repositories», GitHub Docs. [En línea]. Disponible en: https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories
[58] GitHub, «Branches», GitHub Docs. [En línea]. Disponible en: https://docs.github.com/en/pull-requests/reference/branches
[59] GitHub, «About pull requests», GitHub Docs. [En línea]. Disponible en: https://docs.github.com/en/pull-requests/get-started/about-pull-requests
[60] GitHub, «Using issues», GitHub Docs. [En línea]. Disponible en: https://docs.github.com/en/issues/tracking-your-work-with-issues/using-issues
[61] GitHub, «Understanding GitHub Actions», GitHub Docs. [En línea]. Disponible en: https://docs.github.com/en/actions/get-started/understand-github-actions
[62] GitHub, «About Git Large File Storage», GitHub Docs. [En línea]. Disponible en: https://docs.github.com/en/repositories/working-with-files/managing-large-files/about-git-large-file-storage
[63] GitHub, «GitHub Desktop documentation», GitHub Docs. [En línea]. Disponible en: https://docs.github.com/en/desktop
[64] GitLab, «Repository», GitLab Docs. [En línea]. Disponible en: https://docs.gitlab.com/user/project/repository/
[65] GitLab, «Merge requests», GitLab Docs. [En línea]. Disponible en: https://docs.gitlab.com/user/project/merge_requests/
[66] GitLab, «Issue boards», GitLab Docs. [En línea]. Disponible en: https://docs.gitlab.com/user/project/issue_board/
[67] GitLab, «Protected branches», GitLab Docs. [En línea]. Disponible en: https://docs.gitlab.com/user/project/repository/branches/protected/
[68] GitLab, «Get started with GitLab CI/CD», GitLab Docs. [En línea]. Disponible en: https://docs.gitlab.com/ci/
[69] GitLab, «Git Large File Storage (LFS)», GitLab Docs. [En línea]. Disponible en: https://docs.gitlab.com/topics/git/lfs/
[70] GitLab, «GitLab plans», GitLab Docs. [En línea]. Disponible en: https://docs.gitlab.com/subscriptions/choosing_subscription/
[71] Atlassian, «Create a repository in Bitbucket Cloud», Atlassian Support. [En línea]. Disponible en: https://support.atlassian.com/bitbucket-cloud/docs/create-a-repository-in-bitbucket-cloud/
[72] Atlassian, «Configure a project’s branching model», Atlassian Support. [En línea]. Disponible en: https://support.atlassian.com/bitbucket-cloud/docs/configure-a-projects-branching-model/
[73] Atlassian, «Review code in a pull request», Atlassian Support. [En línea]. Disponible en: https://support.atlassian.com/bitbucket-cloud/docs/review-code-in-a-pull-request/
[74] Atlassian, «Configure a project’s branch restrictions», Atlassian Support. [En línea]. Disponible en: https://support.atlassian.com/bitbucket-cloud/docs/configure-a-projects-branch-restrictions/

[75] Atlassian, «Get started with Bitbucket Pipelines», Atlassian Support. [En línea]. Disponible en: https://support.atlassian.com/bitbucket-cloud/docs/get-started-with-bitbucket-pipelines/
[76] Atlassian, «Integrate Bitbucket and Jira», Atlassian Support. [En línea]. Disponible en: https://support.atlassian.com/bitbucket-cloud/docs/use-bitbucket-cloud-and-jira-together/
[77] Atlassian, «Manage large files with Git Large File Storage (LFS)», Atlassian Support. [En línea]. Disponible en: https://support.atlassian.com/bitbucket-cloud/docs/manage-large-files-with-git-large-file-storage-lfs/
[78] Atlassian, «Manage your plan and billing», Atlassian Support. [En línea]. Disponible en: https://support.atlassian.com/bitbucket-cloud/docs/manage-your-plan-and-billing/
[79] M. Potel, «MVP: Model-View-Presenter: The Taligent Programming Model for C++ and Java». 1996. [En línea]. Disponible en: https://www.researchgate.net/publication/255616200_MVP_Model-View-Presenter_The_Taligent_Programming_Model_for_C_and_Java_Taligent_Inc
[80] A. Syromiatnikov y D. Weyns, «A Journey through the Land of Model-View-Design Patterns», en *2014 IEEE/IFIP Conference on Software Architecture*, Sydney, Australia: IEEE, abr. 2014, pp. 21-30. doi: 10.1109/WICSA.2014.13.
[81] T. Ollsson, D. Toll, A. Wingkvist, y M. Ericsson, «Evolution and Evaluation of the Model-View-Controller Architecture in Games», en *2015 IEEE/ACM 4th International Workshop on Games and Software Engineering*, Florence, Italy: IEEE, may 2015, pp. 8-14. doi: 10.1109/GAS.2015.10.
[82] M. Fowler, «Passive View», Martin Fowler. [En línea]. Disponible en: https://martinfowler.com/eaaDev/PassiveScreen.html?utm_source=chatgpt.com
[83] T. Unity, «Vector3.normalized», Unity Scripting API. [En línea]. Disponible en: https://docs.unity3d.com/jp/current/ScriptReference/Vector3-normalized.html?utm_source=chatgpt.com
[84] R. Likert, *A technique for the measurement of attitudes*, vol. 22, 140 vols. New York, 1932. [En línea]. Disponible en: https://archive.org/details/b32829395/page/n3/mode/2up?utm_source=chatgpt.com

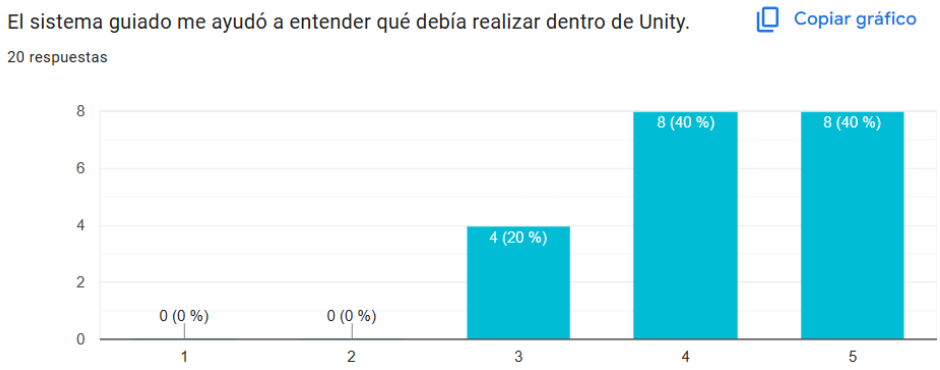
Anexos

Como parte del proceso de validación del sistema educativo interactivo, se aplicó una encuesta a 20 participantes después de utilizar el prototipo desarrollado. El instrumento estuvo compuesto por 10 preguntas orientadas a conocer la percepción de los usuarios sobre la claridad de las instrucciones, la facilidad de navegación, la relación entre el contenido teórico y práctico, la comprensión de las mecánicas y la experiencia general de uso.

Para registrar las respuestas se utilizó una escala de Likert de cinco niveles, mediante la cual los participantes indicaron su grado de acuerdo o desacuerdo con cada afirmación. Los resultados obtenidos se presentan a continuación mediante capturas de los gráficos generados por la herramienta utilizada para aplicar la encuesta. Estas evidencias permiten observar la distribución de las respuestas correspondientes a cada pregunta.

Ilustración 58

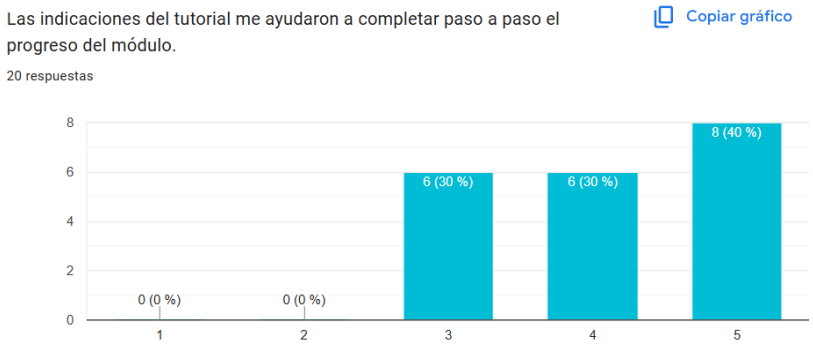
Pregunta y resultado 1.



Nota. Elaboración propia

Ilustración 59

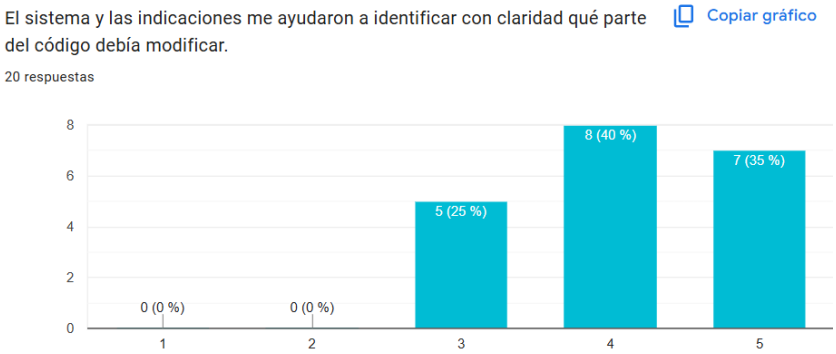
Pregunta y resultado 2.



Nota. Elaboración propia

Ilustración 60

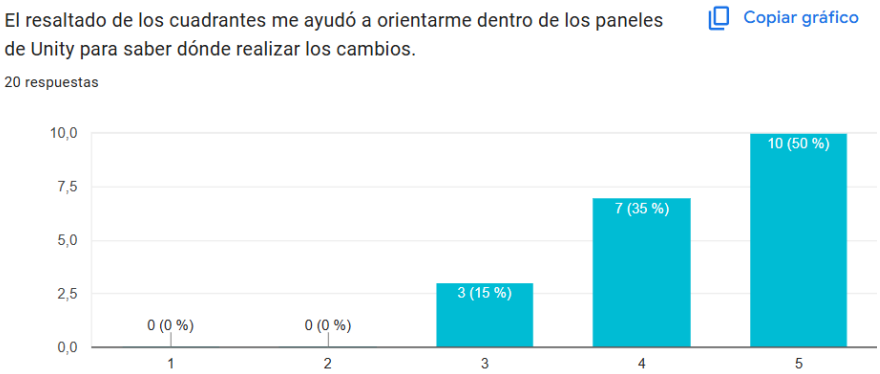
Pregunta y resultado 3.



Nota. Elaboración propia

Ilustración 61

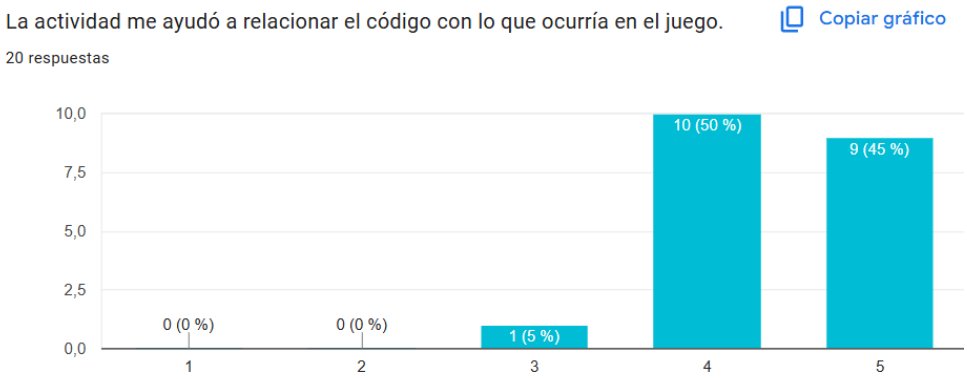
Pregunta y resultado 4.



Nota. Elaboración propia

Ilustración 62

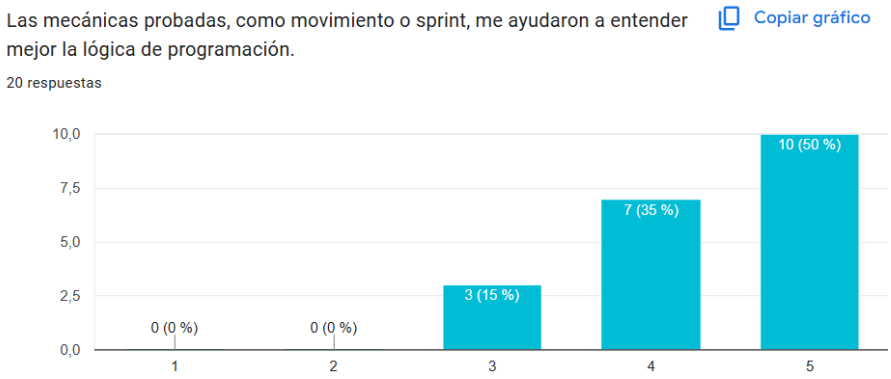
Pregunta y resultado 5.



Nota. Elaboración propia

Ilustración 63

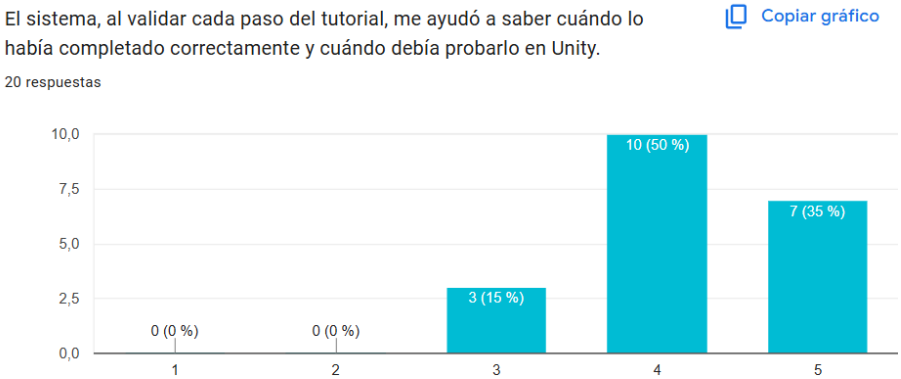
Pregunta y resultado 6.



Nota. Elaboración propia

Ilustración 64

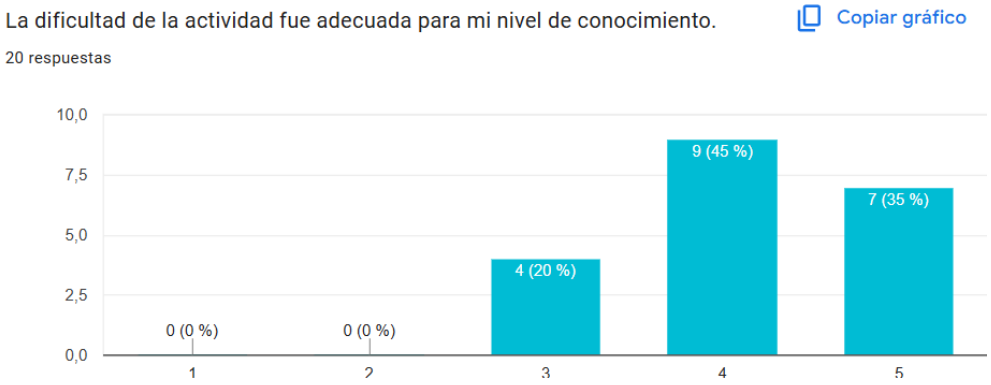
Pregunta y resultado 7.



Nota. Elaboración propia

Ilustración 65

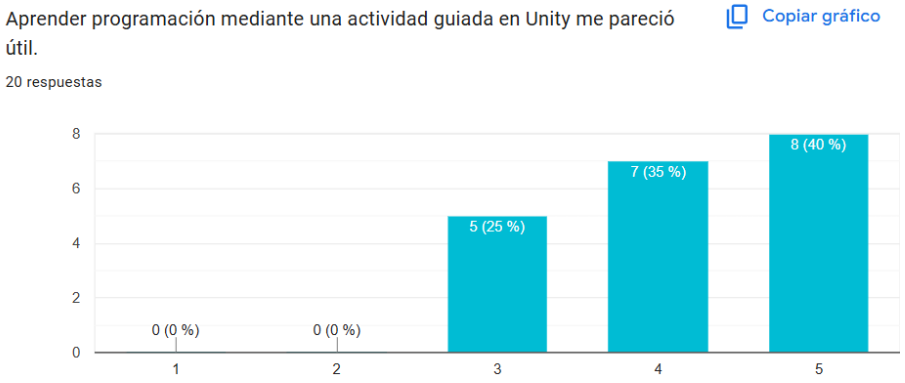
Pregunta y resultado 8.



Nota. Elaboración propia

Ilustración 66

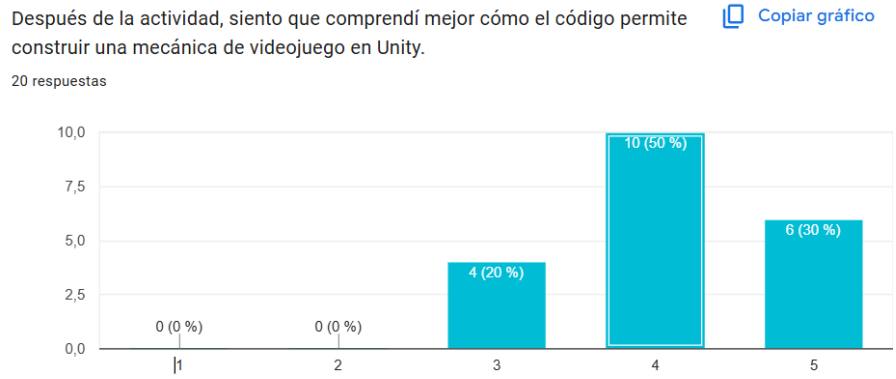
Pregunta y resultado 9.



Nota. Elaboración propia

Ilustración 67

Pregunta y resultado 10.



Nota. Elaboración propia