



UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY



FACULTAD DE
INGENIERÍA

Climb the Cinacina

Un videojuego para la enseñanza de inglés

Informe de Proyecto de Grado presentado por

Andrés Perez, Nicolás Bermúdez, Paula Cianelli

en cumplimiento parcial de los requerimientos para la graduación de la carrera
de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de
la República

Supervisores

Luis Chiruzzo
Santiago Góngora

Montevideo, 21 de septiembre de 2025



Climb the Cinacina

Un videojuego para la enseñanza de inglés por Andrés Perez, Nicolás Bermúdez, Paula Cianelli tiene licencia [CC Atribución 4.0](#).

Agradecimientos

Este proyecto no habría sido posible sin la colaboración de los y las docentes de lengua inglesa, quienes brindaron su apoyo, evaluando el producto y aportando materiales y conocimientos fundamentales del área educativa.

También queremos agradecer especialmente a todas aquellas personas que probaron el producto o nos aconsejaron en distintos momentos a lo largo de la tesis. Un agradecimiento especial también a nuestras familias, parejas y amigos, por el apoyo brindado a lo largo del proyecto.

Por último, un especial agradecimiento a nuestros tutores, Luis y Santiago, quienes nos acompañaron en cada etapa y fueron clave para que el proyecto llegara a buen puerto.

Resumen

Este trabajo presenta *Climb the Cinacina*¹, un videojuego educativo pensado para reforzar vocabulario en inglés en el nivel inicial, priorizando la compatibilidad con las laptops del Plan Ceibal y escenarios de conectividad limitada. El proyecto se enmarca en la continuidad de la plataforma CINACINA y las políticas públicas de incorporación de tecnología educativa en Uruguay. El código del mismo se encuentra en el repositorio: <https://github.com/aperezcastagnino/tecina>

Desde el punto de vista técnico, el juego se implementa como una aplicación web con el framework Phaser, y utiliza generación procedural de niveles mediante el algoritmo de Partición Binaria del Espacio (BSP) para asegurar rejugabilidad y evitar la memorización de patrones. De esta forma, el mapa y la disposición de los elementos cambian entre cada partida.

Para introducir variación lingüística, se integra un gran modelo de lenguaje (LLM). En particular, se emplea Gemini para parafrasear consignas y diálogos, generando variantes controladas que mantienen el significado pedagógico y el nivel objetivo. La interacción con el modelo se encapsula en un servidor construido con FastAPI, preservando un diseño cliente-servidor desacoplado y facilitando la evolución del sistema.

La validación incluyó pruebas generales (34 respuestas a un cuestionario) para detectar problemas de jugabilidad y usabilidad, así como también pruebas reales en una escuela rural en Rivera. La verificación pedagógica se realizó mediante una encuesta a 25 docentes de diferentes regiones del país. Los resultados reportaron una utilidad percibida del 92 % para reforzar vocabulario, 88,6 % de intención de uso en clase y 94,3 % de recomendación a colegas, evidenciando pertinencia y potencial de adopción en aulas reales.

Finalmente, el informe discute líneas de trabajo futuro: soporte de audio en misiones, preguntas de opción múltiple, herramientas para la edición de contenido para docentes, generación de frases completas además del parafraseo, entre otras.

¹<https://aperezcastagnino.github.io/tecina/>. Última vez accedido: 20/09/25.

Palabras clave: Proyectos de Grado, Computación, Educación, Inglés, Cocina, Videojuego, Phaser, Generación procedural de niveles, IA, Modelos de lenguajes, LLM, Gemini

Glosario

Agente: Entidad virtual con comportamientos definidos que le permiten moverse, tomar decisiones y modificar el entorno de manera autónoma.

Asset: Piezas de contenido digital que se integran en un juego; incluyen el diseño visual, archivos multimedia, animaciones, sonidos y cualquier otro recurso.

Colisión: Evento que ocurre cuando dos objetos del juego chocan entre sí.

Commit: Registro de cambios en el repositorio acompañado por un mensaje descriptivo.

FPS (Frames Per Second): representa la velocidad a la que un videojuego puede generar y mostrar imágenes consecutivas.

Integrated Development Environment (IDE): o Entorno de Desarrollo Integrado es una aplicación que reúne en un mismo lugar todas las herramientas necesarias para desarrollar software.

JSON (JavaScript Object Notation): Formato que permite describir las misiones y textos de forma estructurada, facilitando la incorporación de nuevo contenido sin necesidad de modificar el código fuente.

Mapa: Escenario del videojuego que el jugador puede recorrer.

Nivel: Espacio donde ocurre el juego, que abarca no solo la disposición espacial (el mapa), sino también las mecánicas, interacciones, objetos y condiciones necesarias para cumplir con los objetivos.

NPC (Non-Playable Character): Acrónimo del término en inglés para “Personaje No Jugable”, que describe a los personajes sobre los cuales el jugador no tiene control.

Procedural: Generación automática de contenido mediante algoritmos.

Pull Request: Solicitud para que el código implementado en una rama sea revisado por otros miembros del equipo y, en caso de ser aprobado, sea fusionado con la rama principal.

Render: Un render es la imagen, animación o escena final generada a partir de un modelo digital, después de aplicar cálculos de iluminación, texturas, materiales, sombras y efectos visuales.

Roguelike: Subgénero de los videojuegos de rol caracterizado por presentar niveles generados proceduralmente, jugabilidad por turnos y muerte permanente del personaje jugador.

Scene (en Phaser): Representación de una “pantalla” dentro del juego, como el menú principal, cada nivel o la pantalla de victoria. Cada una encapsula su propia lógica.

Scrum: Marco de trabajo ágil que promueve un proceso de desarrollo iterativo e incremental, fomentando la entrega frecuente de software funcional y la adaptación continua.

Sprite: Objeto gráfico bidimensional que normalmente consiste en una imagen de mapa de bits o en una serie de imágenes que se combinan para crear una animación.

Spritesheet: Imagen que contiene varias imágenes más pequeñas (sprites) con el fin de crear una animación.

Tile: Imágenes pequeñas y de forma regular, también llamadas baldosas o mosaicos, que se utilizan para construir un mapa de manera modular.

Tilemap: Técnica popular en el desarrollo de juegos 2D que consiste en construir el mundo del juego o el mapa de niveles a partir de tiles.

WebGL: Estándar web para renderizar gráficos 3D y 2D en el navegador.

Índice general

1. Introducción	1
1.1. Objetivos del proyecto	5
1.2. Organización general del documento	5
2. Fundamentos teóricos y técnicos	7
2.1. Género Roguelike	7
2.2. Tecnologías relevadas	10
2.3. Generación del mapa	13
2.3.1. Generación estática	13
2.3.2. Generación dinámica	14
2.4. Modelo de lenguaje	19
2.5. Síntesis	20
3. Definiendo Climb the Cinacina	21
3.1. Definición conceptual	21
3.1.1. Requerimientos Funcionales	23
3.1.2. Requerimientos No Funcionales	27
3.1.3. Diseño visual	28
3.2. Diseño técnico de la solución	28
3.2.1. Arquitectura	28
3.2.2. Flujos	32
3.3. Síntesis	35
4. Implementación	37
4.1. Aplicación web	37
4.1.1. Phaser	37
4.1.2. Generación del mapa dinámico	40
4.1.3. Renderizado del mapa dinámico	46
4.1.4. Niveles	47
4.1.5. BaseLevelScene	51
4.1.6. Player	54
4.1.7. Requested Items	55
4.1.8. Health Bar	57
4.1.9. Dialog	57

4.2. Diseño	59
4.3. Servidor	64
4.4. Pruebas conceptuales	68
4.4.1. Adaptación a dispositivos móviles	68
4.4.2. Generación de textos	69
4.5. Síntesis	72
5. Validación y verificación	73
5.1. Validación	73
5.1.1. Metodología aplicada	73
5.1.2. Gestión del repositorio	74
5.1.3. Calidad del producto	76
5.1.4. Pruebas generales	79
5.1.5. Pruebas en entornos reales	81
5.2. Verificación	82
5.3. Síntesis	89
6. Conclusiones generales y Trabajo Futuro	91
6.1. Conclusiones generales	91
6.2. Trabajo futuro	93
Referencias	95
A. Respuestas al formulario de validación del juego (general)	97
A.1. ¿Entendiste rápidamente la dinámica del juego?	97
A.2. ¿Tuviste algún problema jugando?	97
A.3. Explique qué problemas/errores encontró en el juego	98
A.4. Sugerencias de mejora, teniendo en cuenta que jugarán niños	98
A.5. Ideas para la próxima versión, nuevas dinámicas o niveles	99
B. Respuestas al formulario de validación del juego (docentes)	101
B.1. ¿Te resultó fácil o difícil entender cómo se juega?	101
B.2. ¿Qué tan adecuada considera la dificultad del juego para el nivel de sus estudiantes?	101
B.3. ¿Qué tan adecuado le pareció el contenido lingüístico del juego (vocabulario, frases, gramática)?	102
B.4. ¿Qué tan útil consideras el juego para reforzar vocabulario en inglés?	102
B.5. ¿Crees que el juego motiva a los estudiantes a usar el inglés?	102
B.6. ¿Qué tan entretenido te pareció el juego?	102
B.7. ¿Recomendaría el juego a otros docentes de inglés?	103
B.8. ¿Qué cambiarías para mejorar el aprendizaje?	103
B.9. Sugerencias y comentarios de los docentes	104
C. Prueba de concepto: ChatGPT	105
D. Prueba de concepto: Claude	107

Capítulo 1

Introducción

La Agenda Uruguay Digital (AUD) es un documento donde se establecen, priorizan, articulan y difunden las iniciativas de desarrollo de la sociedad de la información y el conocimiento en la Administración pública, a través de una visión de alcance nacional, acompañadas de mecanismos de seguimiento y sustentabilidad que aseguren su continuidad y proyección.¹

Constituye una herramienta para continuar avanzando en la transformación digital de forma inclusiva y sustentable, con el objetivo de que más personas aprovechen el avance de la tecnología y puedan acceder a los beneficios de la sociedad de la información en igualdad de condiciones.

La Agenda Uruguay Digital 2025 plantea incorporar la enseñanza de competencias y habilidades digitales en la educación formal e implementar modalidades pedagógicas combinadas en todos los ciclos de la educación formal mediante el uso de la tecnología (Agesic, 2021). De modo complementario, la Política Educativa Nacional 2020-2025 formulada por el Ministerio de Educación y Cultura (MEC) remarca el fortalecimiento de las sinergias entre la ANEP (Administración Nacional de Educación Pública), MEC y Centro Ceibal con el objetivo de “promover el desarrollo de las competencias necesarias para habitar el entorno digital de modo responsable, seguro, crítico, reflexivo y creativo” (Ministerio de Educación y Cultura, 2021).

En el mismo sentido, la ANEP establece en el Plan de Desarrollo Educativo 2020-2024 una transformación curricular integral incluye “la renovación de los contenidos, enfoques y metodologías, procurando el desarrollo de competencias fundamentales para la vida en el siglo XXI” y promueve la incorporación de la ciencia, la tecnología y la ciudadanía digital en el currículo nacional (Administración Nacional de Educación Pública (ANEP), 2020).

¹<https://www.gub.uy/agencia-gobierno-electronico-sociedad-informacion-conocimiento/politicas-y-gestion/programas/agenda-digital-del-uruguay>
Última vez accedido: 19/07/2025.

Sumado a lo anterior, en el 2010 se promulgó la Ley N° 18640² que declara el Proyecto Conectividad Educativa de Informática Básica para el Aprendizaje en Línea (CEIBAL) política de Estado: “*Declárense de interés nacional los programas de carácter general que tengan como objeto la innovación educativa mediante la inclusión de tecnologías digitales, promoviendo la mejora en la educación con impacto en los procesos de aprendizaje, inclusión y crecimiento personal, en la niñez, la adolescencia y la juventud.*”. Que junto con el Plan Estratégico 2021-2025 del Centro Ceibal (Centro Ceibal, 2023) presenta en su propuesta de valor:

- Asegurar el acceso a la tecnología al servicio de los aprendizajes.
- Innovar en infraestructura tecnológica, contenidos, experiencias y entornos digitales.
- Contribuir a la mejora continua de los procesos educativos y desarrollar competencias para la ciudadanía digital.

A raíz de las leyes, estrategias y programas establecidos, se puede evidenciar que existe un consenso en torno a la importancia de innovar en las prácticas educativas mediante la tecnología y el deseo por parte de distintos entes del Estado uruguayo en avanzar en esa dirección.

Respecto a la enseñanza de lenguas extranjeras en la educación pública, la ANEP tiene una unidad especializada, la Dirección de Políticas Lingüísticas (DPL), encargada de diseñar, coordinar e implementar las políticas educativas en materia de lenguas. Su misión es “*formar plurilingües que puedan, por medio del uso de las lenguas, interactuar en ámbitos sociales, académicos y/o laborales. Para alcanzar la calidad de plurilingües, los ciudadanos conocerán una lengua extranjera de comunicación internacional (inglés), una lengua de comunicación regional (portugués) y una tercera lengua extranjera vinculada a sus intereses personales (italiano, francés, alemán u otras)*”³.

La filosofía que guía a la DPL es la de asegurar que el aprendizaje de idiomas sea un derecho universal y permanente, y no un privilegio de unos pocos. En tal sentido, sus lineamientos transforman la enseñanza de lenguas extranjeras “*de un privilegio de algunos a un derecho de todos los niños, jóvenes y docentes*”⁴, ampliando la oferta educativa en idiomas y promoviendo el plurilingüismo a lo largo de toda la trayectoria educativa. Lo que implica, por ejemplo, extender la enseñanza de lenguas a todos los niveles educativos, integrar los idiomas en planes y programas curriculares, y ofrecer oportunidades de formación continua tanto para estudiantes como para docentes en el dominio de lenguas.

²<https://www.impo.com.uy/bases/leyes/18640-2010>

Última vez accedido: 20/07/2025.

³<https://www.anep.edu.uy/codicen/politicas-linguisticas>

Última vez accedido: 20/07/2025.

⁴<https://www.anep.edu.uy/codicen/politicas-linguisticas/presentacion>

Última vez accedido: 20/07/2025

Para alcanzar una cobertura total de todo el país, la ANEP implementó una solución combinada, aprovechando tanto la modalidad presencial tradicional como nuevas modalidades de enseñanza remota apoyadas en tecnología. Por un lado, se ampliaron los programas de Segundas Lenguas con docentes presenciales de inglés en las escuelas que cuentan con maestros o profesores especializados. Por otro lado, y especialmente para aquellas escuelas donde no se dispone de un docente de inglés, se recurrió a la enseñanza mediante videoconferencia interactiva a través del Plan Ceibal en Inglés.

Sin embargo, la solución establecida para las escuelas urbanas resultó parcial para las escuelas del ámbito rural, donde la situación es un poco diferente. Más de la mitad de las escuelas rurales tienen problemas de accesibilidad, lo que hace poco posible la existencia de un profesor presencial en las mismas. Asimismo, esa falta de accesibilidad a veces se traduce en nula o mala conectividad que impide que la enseñanza por videoconferencia sea la forma de atender a dichas escuelas. Por otra parte, la realidad de la mayoría de las escuelas rurales del Uruguay es que las mismas son multigrado, lo que agrega otro nivel más de complejidad.

Geográficamente, el Uruguay cuenta con un número superior de escuelas rurales en el norte del Río Negro. Dichas escuelas no solo superan en número a las del sur, sino que además existe mayor cantidad de escuelas con problemas de accesibilidad.

Es por ello que surge la creación de un programa que no dependa de la presencia de un docente de inglés y que tampoco necesite de que exista conectividad a internet para su funcionamiento. En ese contexto surge el programa “*Inglés Sin Límites (ISL)*”.⁵

ISL tiene como objetivo general la universalización de la enseñanza del idioma inglés en todas las escuelas del país. Dos de sus objetivos son:

- Dar cumplimiento efectivo al derecho que poseen todos los niños de Uruguay a aprender segundas lenguas.
- Crear un curso de inglés que se adapte a las necesidades del ámbito rural cuando no se posee un profesor presencial o cuando no se tiene acceso a recursos como videoconferencias a través de Ceibal en inglés.

La creación de los materiales didácticos utilizados (videos, planes de clase y materiales extra), así como su distribución, está a cargo de la Dirección de PPLL, ya que se tiene un equipo integrado por maestros y profesores de inglés con experiencia tanto en el ámbito urbano como rural.

Los planes de estudio se compilan y se envían junto a los materiales audiovisuales para que el maestro los aplique. En toda clase de lenguas extranjeras es de vital importancia el rol del modelo lingüístico, es decir, la persona que modela la lengua a enseñar. Tradicionalmente, es el docente quien realiza esta

⁵<https://www.anep.edu.uy/codicen/politicas-linguisticas/focus-on-first>
Última vez accedido: 20/07/2025

labor. Pero debido al contexto en el que este programa se plantea, el docente necesita el modelo lingüístico como apoyo a su labor ([Administración Nacional de Educación Pública \(ANEP\), 2019](#)), ya que en muchas ocasiones los docentes no tienen una formación completa en el idioma y, por lo tanto, aprenden en conjunto con los alumnos.

Con la misma finalidad de brindar soporte a la enseñanza de inglés, en 2018 surge la plataforma A (Capacitación en Inglés en el Aula Con Inteligencia Artificial), llevada adelante por docentes de la Facultad de Ingeniería y de la Facultad de Información y Comunicación de la UdelAR, en colaboración con ISL⁶. La cual está pensada para brindar soporte en la enseñanza de inglés en el nivel inicial y ofrece diversos juegos educativos, como sopa de letras, crucigramas o batalla naval basada en oraciones, entre otros. Todos estos juegos emplean distintas técnicas de Procesamiento de Lenguaje Natural (PLN) para la generación automática de ejercicios y juegos didácticos de inglés ([Chiruzzo y cols., 2022](#)). Los juegos de la plataforma demandan pocos recursos, ya que la finalidad de la CINACINA es que sea utilizada en las computadoras brindadas por el Plan Ceibal.⁷

La existencia de esta plataforma, así como las distintas políticas y planes que el Estado uruguayo ha implementado en los últimos años, refuerzan el sentido de continuar desarrollando nuevos juegos para seguir extendiendo la propuesta de la CINACINA. Además, es ampliamente reconocido que los juegos, tanto físicos como virtuales, pueden constituir herramientas didácticas efectivas en el ámbito académico, ya que permiten reforzar conocimientos adquiridos y estimular el trabajo en equipo ([García-Hernández y cols., 2023](#)).

A causa de este contexto, es que se propuso el desarrollo de un nuevo juego con un enfoque innovador: un juego *roguelike* que incorpore un modelo de lenguaje. El objetivo de este proyecto es evaluar la factibilidad técnica de desarrollar un videojuego *roguelike* integrado con un modelo de lenguaje, así como analizar el valor pedagógico que puede aportar una propuesta con estas características.

⁶<https://www.fing.edu.uy/inco/grupos/pln/cinacina>

Última vez accedido: 20/07/2025

⁷<https://ceibal.edu.uy/institucional/que-es-ceibal/>

Última vez accedido: 26/07/2025

1.1. Objetivos del proyecto

- Crear un videojuego estilo roguelike con el objetivo de reforzar la enseñanza de inglés en escuelas rurales para niñas y niños de nivel inicial.
- Utilizar un modelo de lenguaje para agregar dinamismo al juego con el fin de evitar la monotonía.
- El juego tiene que poder ser fácilmente modificable con la finalidad de agregar/modificar/quitar contenido.

1.2. Organización general del documento

El resto del documento se estructura en capítulos de la siguiente forma:

El capítulo 2 (Fundamentos teóricos y técnicos) sienta las bases del proyecto, introduciendo el género de videojuegos roguelike y justificando la elección de tecnologías utilizadas para su creación. Se profundiza en los desafíos técnicos clave, como la generación dinámica de mapas y la integración del modelo de lenguaje Gemini para el parafraseo de contenido.

El capítulo 3 (Definiendo Climb the Cinacina) detalla la conceptualización del videojuego, estableciendo los requerimientos funcionales y no funcionales que guiaron su desarrollo. Además, se presenta el diseño técnico de la solución, abordando la arquitectura del sistema, los flujos de interacción y la estructura del código.

El capítulo 4 (Implementación) expone la implementación técnica del sistema. Se describe el funcionamiento de los componentes principales de la aplicación web, así como también se aborda el rol del servidor y los resultados de las pruebas conceptuales.

El capítulo 5 (Validación y verificación) presenta el proceso utilizado para asegurar la calidad. La sección de validación detalla la metodología de desarrollo, las herramientas de control de calidad y los hallazgos de las pruebas generales. En cambio, la sección de verificación analiza los resultados de las encuestas realizadas a docentes de inglés, confirmando la utilidad pedagógica del juego.

El capítulo 6 (Conclusiones generales y trabajo futuro) presenta las conclusiones del proyecto, resumiendo los objetivos cumplidos y los aprendizajes obtenidos. Finalmente, se proponen futuras líneas de trabajo para expandir y mejorar el videojuego, basadas en las sugerencias recopiladas.

El documento finaliza con las referencias bibliográficas y los anexos que contienen material complementario, como las respuestas detalladas de las encuestas de validación y verificación.

Capítulo 2

Fundamentos teóricos y técnicos

Este capítulo sienta las bases teóricas y técnicas del proyecto, explorando los conceptos fundamentales que guiaron su desarrollo. Se introduce el género de videojuegos roguelike, detallando sus características principales como la generación procedural de niveles y la muerte permanente, elementos clave para la rejugabilidad. A continuación, se presenta un análisis de las tecnologías evaluadas, justificando la elección de Phaser como el framework de desarrollo principal por su ligereza y funcionalidad. Finalmente, se abordan dos de los desafíos técnicos más importantes: la generación de mapas y la integración de un modelo de lenguaje para enriquecer la experiencia educativa del juego.

2.1. Género Roguelike

El género roguelike es un subgénero de los videojuegos de rol caracterizado por presentar niveles generados proceduralmente, jugabilidad por turnos, gráficos basados en mosaicos (tiles) y muerte permanente del personaje jugador¹. En otras palabras, cada partida de un roguelike se desarrolla a través de laberintos, creados aleatoriamente (garantizando que no haya dos experiencias iguales) y, si el personaje muere, el juego no permite reanudar desde un punto guardado, sino que obliga a comenzar de nuevo. Durante la partida, el jugador deberá explorar el laberinto, enfrentarse o evadir enemigos, recolectar objetos y completar misiones que le hayan asignado.

En la figura 2.1 se muestra el juego, *The Way Home*² a modo de ejemplo del género.

¹<https://plarium.com/es/glossary/roguelike-games>. Última vez accedido: 21/07/2025

²<https://twh.concode.co/>. Última vez accedido: 22/07/2025

El género se diferencia de otros géneros más lineales como las plataformas (por ejemplo, *Mario Bros*) o las carreras (*Need for Speed*), ya que ofrece una libertad controlada que permite al jugador decidir su camino, con una dificultad adaptativa que equilibra el reto con la progresión (Adams, 2014).

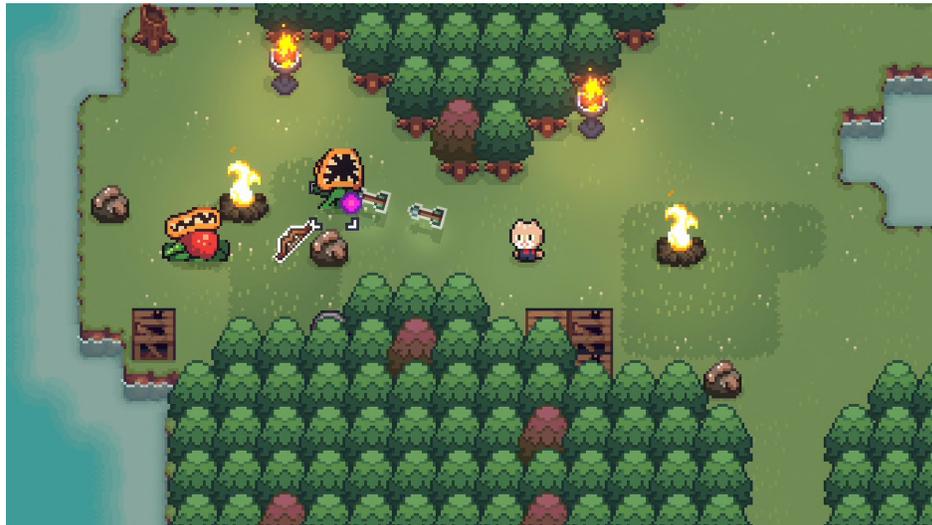


Figura 2.1: The Way Home: Pixel Roguelike

Asociado al género y a los videojuegos, existen una serie de términos que son necesarios definir para un correcto abordaje de la temática:

- **Asset:** Son las piezas de contenido digital que se integran en un juego, incluyen el diseño visual y archivos multimedia, animaciones, sonidos y cualquier otro recurso.³
- **Tile:** Se puede traducir como baldosa o mosaico. Son imágenes pequeñas, de forma regular que se utilizan para construir un mapa.⁴ En la figura 2.2 se pueden ver varios de los tiles utilizados en el proyecto.

³<https://www.autodesk.com/solutions/game-assets>

Última vez accedido: 25/07/2025

⁴<https://developer.mozilla.org/en-US/docs/Games/Techniques/Tilemaps>

Última vez accedido: 25/07/2025



Figura 2.2: Tiles utilizados en la construcción de Climb the Cinacina

- **Tilemap:** Es una técnica muy popular en el desarrollo de juegos 2D, que consiste en construir el mundo del juego o el mapa de niveles a partir de tiles. Esto conlleva a mejoras de rendimiento y uso de memoria, ya que no se necesitan archivos de imagen grandes que contengan mapas de niveles completos, sino que estos se construyen reutilizando múltiples veces pequeñas imágenes o fragmentos de imágenes.⁵
- **Sprite:** Es un objeto gráfico bidimensional (2D). Normalmente consiste en una imagen de mapa de bits o en una serie de imágenes que se combinan para crear una animación.⁶
- **Spritesheet:** Es una imagen que contiene varias imágenes más pequeñas con el fin de crear una animación.⁷ En la figura 2.3 se muestra el spritesheet de un zorro, un ítem que aparece en distintos niveles del juego. Si se observa detenidamente, y con un poco de imaginación, es posible visualizar el movimiento que realizaría el animal. Cada una de las imágenes individuales es lo que se denomina un sprite.



Figura 2.3: Ejemplo de spritesheet

⁵<https://developer.mozilla.org/en-US/docs/Games/Techniques/Tilemaps>

Última vez accedido: 25/07/2025

⁶<https://www.codeandweb.com/texturepacker/knowledgebase/what-is-a-sprite>

Última vez accedido: 25/07/2025

⁷<https://www.codeandweb.com/what-is-a-sprite-sheet>

Última vez accedido: 25/07/2025

- **Colisión:** Una colisión ocurre cuando dos objetos chocan entre sí.⁸
- **NPC:** Es un acrónimo del término “Non-Playable Character” (Personaje No Jugable), que, como su nombre indica, describe a los personajes sobre el jugador no tiene control.⁹
- **Mapa:** Es el escenario del videojuego que el jugador puede recorrer.¹⁰
- **Nivel:** Un nivel es el espacio donde ocurre el juego.¹¹ A diferencia del concepto de mapa, el nivel abarca no solo la disposición espacial, sino también las mecánicas necesarias para que el juego funcione. Esto incluye las interacciones posibles del jugador, así como los objetos, herramientas y condiciones que permiten cumplir con los objetivos del juego.

2.2. Tecnologías relevadas

Para la construcción del juego se consideraron diversas tecnologías disponibles, teniendo en cuenta las limitaciones impuestas por los requerimientos no funcionales, descritos en la sección 3.1.2. Las tecnologías fueron: Unity¹², Three.js¹³, PixiJS¹⁴ y Phaser¹⁵.

Unity es un motor gráfico multiplataforma ampliamente utilizado en la industria por su capacidad para desarrollar juegos tanto en 2D como en 3D. Sin embargo, debido a su alta demanda de recursos de hardware para el renderizado de gráficos, se consideró inadecuado para este proyecto dadas las limitaciones de las laptops del plan Ceibal. Otra biblioteca para gráficos 3D fue **Three.js**, la cual también fue desestimada por su elevado consumo de recursos gráficos.

PixiJS es una biblioteca de JavaScript¹⁶ (JS), especializada en renderizado de gráficos 2D, que destaca por su rapidez y eficiencia. Sin embargo, aunque ofrece buen rendimiento gráfico, su enfoque principal está limitado al renderizado visual, careciendo de funcionalidades adicionales como manejo de propiedades físicas (como por ejemplo, las colisiones entre objetos) o gestión de escenas, aspectos esenciales para el juego propuesto. Desarrollar estas funcionalidades desde cero habría consumido mucho tiempo, razón por la cual también fue descartada.

⁸<https://catnessage.com/wikigame/collision/>. Última vez accedido: 25/07/2025

⁹<https://www.ecigames.net/media/what-are-non-playable-characters-npcs>
Última vez accedido: 25/07/2025

¹⁰<https://catnessage.com/wikigame/map>. Última vez accedido: 25/07/2025.

¹¹<https://book.leveldesignbook.com/introduction>. Última vez accedido: 25/07/2025.

¹²<https://unity.com/es/>. Última vez accedido: 27/05/2025

¹³<https://threejs.org/>. Última vez accedido: 27/05/2025

¹⁴<https://pixijs.com/>. Última vez accedido: 27/05/2025

¹⁵<https://phaser.io/>. Última accedido: 27/05/2025

¹⁶<https://developer.mozilla.org/es/docs/Web/JavaScript>
Última vez accedido: 15/07/2025

Finalmente, se optó por **Phaser**, un framework de JS especialmente diseñado para el desarrollo de juegos en HTML5. Phaser cuenta con una amplia comunidad de usuarios, lo que lo convierte en una de las opciones más populares y garantiza soporte continuo durante varios años. Adicionalmente, se destacan varias ventajas clave que justifican esta elección:

- **Ligereza y rendimiento:** Phaser es liviano y está optimizado para funcionar eficientemente en dispositivos con bajo rendimiento, condición ideal para el contexto de las laptops del plan Ceibal.
- **Funcionalidades integradas:** A diferencia de PixiJS, Phaser ofrece múltiples funcionalidades esenciales integradas, como física básica, gestión de escenas y animaciones, lo que reduce considerablemente el tiempo de desarrollo.
- **Comunidad y documentación:** Phaser posee una extensa documentación y una comunidad activa que brinda soporte constante, lo que garantiza su utilización a largo plazo y facilita la resolución de problemas.

Además, Phaser se utilizó en conjunto con TypeScript¹⁷. Este es un lenguaje de programación que añade la verificación de tipos estáticos a JavaScript. Se puede pensar en él como un superconjunto de JavaScript, lo que significa que todo lo disponible en JavaScript también está disponible en TypeScript. Por lo tanto, cualquier programa en JavaScript es sintácticamente válido en TypeScript. La verificación de tipos se realiza en tiempo de compilación, por lo que el código de TypeScript nunca se visualizará en un navegador.¹⁸

La elección de utilizar ambas tecnologías fue con el objetivo de detectar errores tempranamente durante el desarrollo, mejorar la legibilidad del código y promover una mejor organización y estructura. Además, a través de los tipos se disminuye la curva de aprendizaje del nuevo framework.

Por último, se incorporó ViteJS¹⁹ como herramienta central para completar el proceso de desarrollo y empaquetado del código. Vite es una herramienta de construcción moderna, diseñada específicamente para optimizar tanto el entorno de desarrollo como la generación del producto final en aplicaciones web. Ofrece una experiencia de desarrollo notablemente más ágil y eficiente al aprovechar la modularidad nativa de los navegadores modernos, combinada con un sistema de compilación altamente optimizado para producción²⁰.

¹⁷<https://www.typescriptlang.org/>
Última vez accedido: 27/05/2025

¹⁸<https://developer.mozilla.org/en-US/docs/Glossary/TypeScript>
Última vez accedido: 15/07/2025

¹⁹<https://vite.dev/>
Última vez accedido: 27/05/2025

²⁰<https://vitejs.dev/guide/why.html>
Última vez accedido: 26/07/2025

Estas características resultaron fundamentales para garantizar un rendimiento adecuado en contextos de conectividad limitada e inestable, así como para asegurar una ejecución fluida en dispositivos con capacidades de hardware reducidas.

La combinación de estas tres tecnologías permitió mantener una curva de aprendizaje moderada durante el desarrollo. A su vez, garantizó que el producto final pudiera ejecutarse sin demandar recursos significativos, logrando así cumplir con los requerimientos no funcionales establecidos para el proyecto.

2.3. Generación del mapa

Como se explicó al inicio del capítulo, el género roguelike se caracteriza por presentar niveles generados de forma procedural, ya que uno de sus pilares fundamentales es la rejugabilidad, la cual se ve potenciada por la variación constante de los mapas entre partidas. Esta característica resulta clave para ofrecer experiencias únicas en cada sesión de juego y evitar que los jugadores memoricen soluciones o patrones fijos.

En este sentido, uno de los principales desafíos de desarrollo fue la generación de mapas. Para ello, se contemplaron dos enfoques posibles: una generación estática, basada en mapas predefinidos, o una generación dinámica en tiempo real, alineada con las convenciones del género.

Ambas alternativas fueron analizadas, pero se optó por la generación dinámica debido a su capacidad de mantener el interés y la motivación de los niños durante el juego y el proceso de aprendizaje. La posibilidad de crear entornos variados y cambiantes en cada partida promueve la curiosidad, evita la repetición mecánica de soluciones y asegura una experiencia educativa más rica, continua y atractiva.

2.3.1. Generación estática

Para la generación de mapas estáticos se utilizó la herramienta Tiled²¹. Tiled es un editor gráfico de mapas de código abierto que permite definir niveles mediante la creación y disposición visual de elementos como capas, objetos colisionables y zonas interactivas.

Las capas son conjuntos organizados de tiles que permiten separar visualmente distintos componentes del mapa, como el fondo, los elementos decorativos, o las zonas de colisión. Los objetos colisionables definen áreas del mapa donde el jugador no puede atravesar, como paredes, obstáculos o límites del escenario. Por su parte, las zonas interactivas son regiones del mapa que activan eventos o interacciones específicas, como la apertura de una puerta, la aparición de un objeto o el inicio de una misión.

La principal ventaja de la generación de mapas estáticos con Tiled radica en la capacidad para diseñar niveles visualmente robustos y complejos. Lo que permitió experimentar rápidamente con diferentes configuraciones y evaluar aspectos estéticos y funcionales del juego. Sin embargo, este enfoque presentaba desventajas significativas, principalmente relacionadas con el tiempo de creación requerido para diseñar y ajustar cada nivel individualmente. Además, los mapas generados de forma estática carecían de variabilidad entre partidas, lo que afectaba negativamente la rejugabilidad y el dinamismo, aspectos cruciales para mantener el interés y la motivación de los niños. En la figura 2.4 se puede visualizar la interfaz del programa Tiled y cómo se ve un mapa generado de

²¹<https://www.mapeditor.org/>. Última vez accedido: 27/05/2025

manera estática.

Debido a estas limitaciones, y buscando potenciar la variabilidad y capacidad de mantener el interés constante del jugador, se optó por adoptar una generación dinámica de mapas, dejando atrás la creación manual de mapas estáticos mediante Tiled. De todas maneras, resultó especialmente útil durante las etapas iniciales del desarrollo, ya que facilitó la comprensión del funcionamiento y la interacción entre los distintos componentes del juego, como el personaje principal, los objetos del entorno y las mecánicas básicas de interacción y colisión. Además, permitió contar con una primera versión del mapa para comenzar a experimentar con la biblioteca seleccionada.

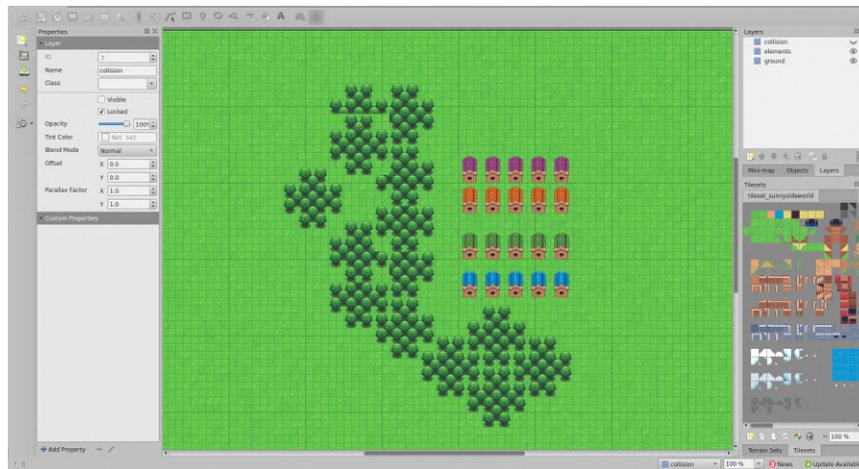


Figura 2.4: Tiled: Generación estática del mapa

2.3.2. Generación dinámica

La generación dinámica o procedural del mapa implica la creación automática de la topología, geometría y elementos relacionados con el modo de juego. En general, los métodos típicos de generación procedural constan de tres componentes fundamentales según especifican [Shaker, Liapis, Togelius, Lopes, y Bidarra \(2016\)](#):

- **Modelo representacional:** Una representación abstracta y simplificada del mapa que proporciona una vista general de la estructura final del nivel.
- **Método de construcción del modelo representacional:** Técnicas y estrategias empleadas para estructurar el mapa.
- **Generación geométrica:** Proceso mediante el cual se construye la geometría concreta del mapa a partir del modelo representacional.

Los beneficios derivados del uso de generación procedural son notorios, ya que permite crear grandes cantidades de contenido diverso y detallado sin necesidad de diseñar manualmente cada elemento individual. Esto aumenta significativamente la rejugabilidad del juego, ofreciendo experiencias únicas en cada partida y motivando a los jugadores a explorar diferentes escenarios. Además, se logra una notable eficiencia en el desarrollo, pues reduce considerablemente el tiempo y los recursos necesarios para crear niveles de forma manual, lo que permite al equipo concentrarse en otros aspectos esenciales, como la narrativa o el perfeccionamiento de mecánicas específicas. Finalmente, la generación procedural disminuye la aparición de problemas inesperados, dado que se fundamenta en algoritmos previamente probados y validados por expertos, minimizando riesgos y posibles dificultades durante el desarrollo.

Para llevar a cabo la generación, existen diversas estrategias según [Shaker y cols. \(2016\)](#):

- **Agent-based dungeon growing:** Utiliza entidades autónomas denominadas agentes, que recorren un espacio vacío creando habitaciones, corredores y elementos del nivel mediante reglas predefinidas. Un agente es una entidad virtual con comportamientos definidos que le permiten moverse, tomar decisiones y modificar el entorno de manera autónoma. Estos agentes comienzan en un punto inicial del mapa y, bajo ciertas reglas, avanzan por el espacio vacío “excavando” nuevas áreas, como si fueran mineros. Durante este proceso, deciden cuándo construir habitaciones, conectar corredores o colocar obstáculos. Estas decisiones suelen depender de parámetros aleatorios, heurísticas o condiciones específicas, generando diseños variados.
- **Cellular Automata:** Esta técnica emplea una matriz de celdas, donde cada celda posee un estado (por ejemplo, “vacío” o “pared”) que se actualiza según reglas simples aplicadas iterativa y localmente. Cada celda determina su nuevo estado basándose en el estado de las celdas vecinas. Se inicia con una grilla de valores asignados aleatoriamente, sobre la cual se aplican reglas del estilo: *“Si una celda tiene más de 4 vecinas ocupadas, se convierte en pared; si tiene menos de 2 vecinas ocupadas, se convierte en espacio abierto”*. Después de varias iteraciones, la matriz evoluciona hasta mostrar patrones naturales, como habitaciones, pasillos y áreas conectadas, todo a partir de reglas simples.
- **Grammar-based dungeon generation:** Una gramática es un conjunto de reglas formales que especifican cómo construir un nivel paso a paso. Se define un conjunto de instrucciones (gramática) para guiar la construcción del nivel. Por ejemplo, partiendo de una sala grande y aplicando un conjunto de reglas, como “dividirla en dos habitaciones” o “añadir un pasillo”, se puede lograr que el nivel crezca de forma estructurada y con cierta variedad.

- **Space Partitioning:** Un algoritmo de partición espacial genera una partición del espacio, es decir, una subdivisión de un espacio 2D o 3D en subconjuntos disjuntos, de forma que cualquier punto dentro del espacio pertenezca exactamente a uno de esos subconjuntos. Suelen operar de forma jerárquica, es decir, cada celda en la partición puede subdividirse aplicando recursivamente el mismo algoritmo. Esto permite que las particiones se organicen en lo que se conoce como árbol de partición espacial. El método más popular de partición espacial es la partición binaria del espacio (BSP, Binary Space Partition), que divide recursivamente un espacio en dos subconjuntos. A través de este método, el espacio puede representarse como un árbol binario, denominado árbol BSP.

Cabe agregar que diversos juegos exitosos emplean técnicas más sofisticadas de algoritmos procedurales para generar mundos completos, como es el caso de No Man's Sky²², conocido por su vasto universo generado proceduralmente, que permite explorar planetas únicos con ecosistemas diversos. Otro ejemplo destacado es X-COM 2²³, que utiliza generación semi-procedural en sus mapas tácticos, combinando elementos predefinidos generados en tiempo real, a diferencia del enfoque completamente aleatorio de No Man's Sky.

La implementación elegida, que se profundiza en el capítulo 4, se basa en el algoritmo de **Spacing Partitioning**. Este algoritmo se destaca por su capacidad para generar niveles mediante la división iterativa del espacio, asegurando una distribución eficiente y diversa del contenido dentro del mapa. También se destacan su facilidad de implementación, rapidez y eficiencia, características que permiten generar niveles dinámicos incluso en tiempo real con un bajo costo computacional, lo cual es ideal para el contexto del proyecto.

No obstante, es importante señalar que, al igual que la mayoría de los métodos constructivos, este algoritmo ofrece un control limitado sobre las características específicas del nivel generado. En otras palabras, el algoritmo no provee mecanismos para detectar soluciones inválidas, por lo que no garantiza la consistencia y jugabilidad del contenido generado. En consecuencia, dichos controles deben ser gestionados en la implementación del mismo.

Este enfoque proporcionó un equilibrio entre variabilidad, eficiencia y jugabilidad, permitiendo abordar los objetivos propuestos en este proyecto respecto a la generación dinámica de mapas. En el algoritmo 1 se presenta una versión del mismo.

²²<https://www.nomanssky.com/>. Última vez accedido: 15/05/2025

²³<https://xcom.com/>. Última vez accedido: 15/05/2025

Algoritmo 1 Generación del nivel mediante BSP

```
1: Comenzar con toda el área del nivel (nodo raíz del árbol BSP)
2: procedure PARTICIONAR(área)
3:   if área es mayor que el tamaño mínimo then
4:     Dividir él área en dos subáreas
5:     Llamar recursivamente a PARTICIONAR en cada subárea
6:   else
7:     Designar área como hoja del árbol BSP
8:   end if
9: end procedure
10: for all hojas del árbol BSP do
11:   Crear una habitación dentro de los límites de la subárea
12: end for
13: for all pares de habitaciones adyacentes do
14:   Conectar las habitaciones mediante corredores
15: end for
```

El proceso se basa, como ya se dijo anteriormente, en dividir recursivamente el espacio disponible en subáreas más pequeñas, y luego conectar las “habitaciones” generadas dentro de estas regiones. Se denomina “habitación” a una zona indivisible del mapa, que posteriormente se utilizará para asegurar la accesibilidad interna. Esto permite colocar distintos elementos en su interior con la garantía de que el jugador pueda llegar a ellos sin restricciones.

Los pasos del algoritmo 1 se pueden pensar como los siguientes pasos:

1. **Inicio con el área completa:** Se comienza con el área total destinada al nivel, que se considera el nodo raíz de un árbol BSP.
2. **División del área:** El área actual se divide en dos subáreas utilizando una línea de división, que puede ser horizontal o vertical. La elección de la dirección y la posición de la división puede ser aleatoria o seguir criterios específicos para lograr variedad en el diseño.
3. **Recursión en subáreas:**
 - a) Si una subárea resultante es más grande que un tamaño mínimo predefinido, se repite el proceso de división en esa subárea.
 - b) Si la subárea es igual o menor al tamaño mínimo, se considera una hoja del árbol BSP y no se divide más.
4. **Creación de habitaciones:** Para cada hoja del árbol BSP (subárea indivisible), se crea una habitación. Estas habitaciones se colocan dentro de los límites de la subárea correspondiente y pueden tener dimensiones variables para añadir diversidad.

5. **Conexión de habitaciones:** Finalmente, se conectan las habitaciones entre sí mediante corredores. Estos corredores se trazan entre las habitaciones de subáreas adyacentes, asegurando que todo el mapa sea navegable y que el jugador pueda acceder a todas las áreas.

Este pseudocódigo presentado genera un nivel con una estructura jerárquica y organizada, facilitando el control sobre el diseño y asegurando la conectividad entre sus diferentes partes.

2.4. Modelo de lenguaje

Otro aspecto clave del proyecto a resolver fue la incorporación de un modelo de lenguaje en el juego con el fin de mejorar la experiencia del mismo o del aprendizaje. Un modelo de lenguaje es un sistema computacional que aprende patrones del lenguaje natural a partir de grandes volúmenes de texto, con el objetivo de predecir la probabilidad de ocurrencia de una secuencia de palabras o de generar texto coherente a partir de una entrada dada. Según definen [Jurafsky y Martin \(2023\)](#), es un modelo de aprendizaje automático que predice las próximas palabras.

Dentro de la familia de los modelos de lenguaje, los Large Language Models (LLM) son ampliamente utilizados hoy en día. Estos sistemas llevan la idea de “predecir la próxima palabra” a una escala masiva, basándose en arquitecturas de redes neuronales profundas ([Goodfellow, Bengio, y Courville, 2016](#)).

La innovación que impulsa a los LLM modernos es la arquitectura “Transformer”, introducida por [Vaswani y cols. \(2017\)](#). Su mecanismo de autoatención (self-attention) permite al modelo ponderar la relevancia de cada palabra en una secuencia respecto a todas las demás. Esta capacidad es crucial para capturar dependencias a largo plazo y contextos complejos de manera eficiente, superando las limitaciones de arquitecturas previas. Es este entrenamiento a gran escala sobre la tarea de predicción lo que les otorga a los LLM capacidades para generar, resumir, traducir y responder preguntas con un alto grado de coherencia ([Kasneci y cols., 2024](#)).

El objetivo principal de integrar un LLM en el proyecto fue con la finalidad de introducir variabilidad en las frases y consignas presentadas al jugador. Dado que de esta manera se logra incrementar el dinamismo y la rejugabilidad del juego educativo, evitando la repetición de contenido y reforzando el aprendizaje del idioma.

Con este propósito, se empleó la técnica del parafraseo, que consiste en reformular oraciones manteniendo su significado original, pero variando su estructura y vocabulario. En entornos educativos, el parafraseo es una herramienta valiosa, ya que favorece la exposición a formas lingüísticas diversas y ayuda a evitar la memorización mecánica, promoviendo así una comprensión más profunda del contenido ([Kasneci y cols., 2024](#)).

Para llevar a cabo la implementación del parafraseo, se seleccionó Gemini²⁴, una familia de modelos de lenguaje desarrollada por Google. La elección se basó en su alto rendimiento en tareas de comprensión y generación de lenguaje natural, así como en la facilidad de integración a través de su SDK (Software Development Kit). La versión de Gemini utilizada fue *gemini-2.0-flash* debido a su

²⁴<https://gemini.google.com/?hl=es>. Última vez accedido: 11/08/2025

licencia gratuita.

Además de sus capacidades lingüísticas, los LLM modernos como Gemini, Claude²⁵, ChatGPT²⁶ son multimodales. Lo que significa que pueden procesar y razonar de manera íntegra sobre información en múltiples formatos, como texto, imágenes, audio y vídeo ([The Gemini Team, 2023](#)). Si bien para la generación de variantes de texto se utilizó solamente Gemini, esta capacidad multimodal de los LLM también fue aprovechada de manera secundaria en el proyecto, por ejemplo, al utilizar ChatGPT para la generación de imágenes de assets o fondos de pantalla, como se verá en la sección [4.2](#).

La interacción con el modelo de lenguaje se implementó en Python²⁷, un lenguaje ampliamente reconocido por su robusto ecosistema de bibliotecas orientadas a la inteligencia artificial. Para exponer esta funcionalidad como un servicio web, se empleó el framework FastAPI²⁸. Este framework destaca por su simplicidad y productividad: aprovecha el tipado estándar de Python, genera automáticamente documentación interactiva (Swagger UI) y ofrece soporte integrado para validación, autenticación, inyección de dependencias y pruebas²⁹. En cuanto al tipado estático, FastAPI se apoya en la biblioteca Pydantic³⁰, la cual no solo habilita sugerencias y autocompletado en el IDE (Integrated Development Environment), facilitando así el proceso de desarrollo, sino que también contribuye a reducir la introducción de errores, además de mejorar la mantenibilidad y la legibilidad del código.

2.5. Síntesis

En este capítulo se establecen los pilares técnicos del proyecto. Se define el género roguelike, cuya generación procedural de niveles es clave para la rejugabilidad. Se justifica la elección de Phaser como framework principal por su ligereza y eficiencia en equipos de bajos recursos. Se detalla el método de generación dinámica de mapas usando el algoritmo de Partición Binaria del Espacio (BSP) para crear escenarios únicos. Finalmente, se explica el uso del modelo de lenguaje Gemini para parafrasear consignas, introduciendo variabilidad y evitando la repetición en el contenido educativo.

²⁵<https://claude.ai/>. Última vez accedido: 11/08/2025

²⁶<https://openai.com/index/chatgpt/>. Última vez accedido: 11/08/2025

²⁷<https://www.python.org/>. Última vez accedido: 10/08/2025

²⁸<https://fastapi.tiangolo.com/>. Última vez accedido: 10/08/2025

²⁹<https://fastapi.tiangolo.com/features/>. Última vez accedido: 12/08/2025

³⁰<https://docs.pydantic.dev/latest/>. Última vez accedido: 12/08/2025

Capítulo 3

Definiendo Climb the Cinacina

El principal reto para definir el videojuego consistió en crear un juego educativo que resultara atractivo, fácil de usar y que pudiera ejecutarse correctamente en las laptops distribuidas por el Plan Ceibal.

Este capítulo presenta un recorrido completo desde la definición del juego hasta el diseño técnico de la solución implementada. Comenzando por la definición conceptual del videojuego, en donde se describe de manera clara y concisa de qué trata, junto con sus requerimientos funcionales y no funcionales. Luego, se abordan el diseño arquitectónico, la estructura del sistema, los flujos principales de interacción y la organización del código fuente.

3.1. Definición conceptual

A partir de las necesidades identificadas en el capítulo 1 y considerando las características propias del género roguelike mencionadas en 2.1, Climb the Cinacina se define como una aventura educativa en la que el jugador asume el rol de un personaje que debe completar misiones en inglés para avanzar a través de distintos niveles.

El juego se compone de nueve niveles, organizados en tres bloques temáticos. Cada bloque contiene tres niveles que comparten una ambientación y desafíos específicos, lo que permite introducir progresivamente nuevo vocabulario y mantener el interés del jugador. A lo largo de toda la experiencia, las instrucciones, diálogos y objetivos están presentados exclusivamente en inglés, fomentando así la inmersión en el idioma.

El jugador asume el control de un personaje que puede desplazarse libremente por los escenarios, interactuar con distintos objetos y entablar conversaciones con personajes no jugables (NPCs). Estos NPCs asignan misiones que consisten en localizar y recolectar determinados objetos distribuidos en el entorno. Una vez obtenido el objeto correcto, el jugador debe entregarlo al NPC correspondiente para completar la misión y avanzar hacia la siguiente, o bien desbloquear el acceso al próximo nivel en caso de haber cumplido con todas las misiones disponibles. Si el jugador entrega un objeto equivocado, se le aplicará una penalización que consiste en la reducción de un porcentaje de su barra de vida. Al acumular un número determinado de errores, deberá reiniciar el nivel desde el comienzo, lo que incrementa la dificultad y fomenta la atención y precisión durante la partida.

Cada nivel tiene un diseño único, generado proceduralmente, de manera que la disposición de elementos varía en cada partida. Lo que asegura que el jugador se enfrente a escenarios distintos en cada intento, aumentando la rejugabilidad y evitando la memorización de patrones fijos. La dificultad del juego progresa de forma gradual. A medida que el jugador avanza, se incrementa la complejidad de las misiones, tanto en la cantidad de elementos a recolectar como en las estructuras lingüísticas utilizadas.

El diseño del videojuego priorizó la simplicidad, por lo que se eligió una estética 2D. Además, de esta manera se requieren menos recursos de hardware para su funcionamiento, ideal para las laptops distribuidas por el Plan Ceibal. Asimismo, la interfaz buscó ser intuitiva, permitiendo que el jugador pueda concentrarse en la comprensión de las misiones y en la resolución de los desafíos propuestos, dado que la edad del público objetivo son niños entre 8 y 11 años.

En resumen, el juego combina mecánicas clásicas del género roguelike con elementos educativos, ofreciendo una experiencia lúdica que promueve el aprendizaje del inglés mediante la exploración, la resolución de misiones contextuales y la interacción con personajes dentro de un entorno virtual dinámico.

3.1.1. Requerimientos Funcionales

Los requerimientos funcionales son enunciados acerca de los servicios que el sistema debe proveer, de cómo debería reaccionar ante entradas particulares y de cómo debería comportarse en situaciones específicas (Sommerville, 2011). En nuestro caso, esto implica definir qué acciones, comportamientos y características debe ofrecer el juego para cumplir su propósito educativo. Los requerimientos funcionales se encuentran divididos en las siguientes categorías:

Organización del juego

El juego deberá estar estructurado en niveles, organizados en subconjuntos según su temática. Cada nivel debe presentar un conjunto de objetivos definidos a través de misiones que implican la recolección de objetos. Al completar dichas misiones, el sistema debe marcar el nivel como finalizado y habilitar los siguientes niveles, los cuales podrán ser uno o varios simultáneamente.

La dificultad de los niveles será progresiva y se reflejará en: 1) un mayor número de objetos solicitados; 2) misiones compuestas por múltiples pasos; y 3) consignas redactadas mediante frases enunciativas de mayor complejidad.

En la figura 3.1 se presenta la pantalla de selección de niveles implementada, la cual muestra el grafo de progresión de los mismos. El espacio vacío entre las copas de los árboles fue utilizado para representar el camino que conecta los distintos niveles.

Las temáticas abordadas en los nueve niveles se distribuyen en tres grupos: los primeros tres niveles están enfocados en frutas, los siguientes tres en animales y los últimos tres en objetos del hogar.



Figura 3.1: Pantalla de selección de niveles

El juego también deberá contar con una pantalla de carga, un menú de inicio y pantallas específicas para los estados de nivel completado, juego completado y derrota, con el objetivo de ofrecer una experiencia de usuario completa y coherente.

Mecánicas principales

El jugador deberá poder controlar el desplazamiento del personaje utilizando las flechas del teclado, permitiéndole moverse en las cuatro direcciones dentro del entorno. Se deberá restringir el acceso a zonas inaccesibles, respetando los límites y colisiones definidos en el mapa. Todos los movimientos deben reflejarse en tiempo real en pantalla.

Además, podrá interactuar con objetos y NPCs mediante la tecla **espacio**. Esta tecla se considera la tecla de “acción” ya que se utiliza para recolectar o soltar elementos, así como para iniciar o avanzar en los diálogos con los NPCs. La respuesta del sistema debe ser inmediata, mostrando visualmente las acciones realizadas, ya sea mediante la recolección visible de objetos o la apertura de cuadros de diálogo al interactuar con un NPC.

Durante la partida, el sistema debe mostrar una barra de vida visible de forma permanente. Esta barra se reduce en tiempo real cada vez que el jugador entrega un objeto incorrecto a un NPC. La penalización debe ser acumulativa y, al alcanzar el umbral máximo de errores, el jugador pierde el nivel y deberá comenzar desde el principio. A modo ilustrativo, en la figura 3.2 se puede visualizar en la esquina superior izquierda la barra de vida implementada.

El sistema debe registrar el progreso del jugador, guardando el nivel más alto

alcanzado y el estado del mapa actual en caso de no completarlo. Al reanudar la partida, el jugador debe continuar desde el punto exacto donde se detuvo, sin que el mapa sea regenerado.

Entorno de exploración

El jugador debe poder desplazarse libremente dentro de un entorno compuesto por distintas áreas, tales como caminos, zonas bloqueadas, objetos interactivos y personajes no jugables. La navegación debe ser continua y fluida, sin interrupciones ni tiempos de carga entre las distintas secciones del mapa.

En la figura 3.2 se muestra un ejemplo de estas características implementadas. El camino, visible en la parte superior del personaje, representa una zona transitable donde el jugador puede desplazarse libremente. Como ejemplo de zona bloqueante se pueden mencionar los árboles, los cuales delimitan áreas inaccesibles para el personaje, al igual que ocurre con el espacio ocupado por los NPCs. Finalmente, también se pueden ver objetos interactivos, siendo en este caso las manzanas distribuidas en el entorno, que el jugador podrá recolectar como parte de las misiones asignadas.

Cabe destacar que, en el camino, no se pueden ubicar objetos bloqueantes, ya que estos impedirían el paso y podrían hacer inaccesibles determinadas zonas del mapa. Por este motivo, se distingue conceptualmente el “camino” del resto del espacio libre, asegurando así la transitabilidad del jugador. En la sección 4.1.2 se detallan los distintos tipos de objetos que se colocan en el mapa y las restricciones asociadas a su ubicación.



Figura 3.2: Imagen de los distintos elementos de un mapa

Al inicio de cada partida, el sistema deberá generar automáticamente el mapa

mediante parámetros predefinidos combinados con elementos aleatorios. Este procedimiento asegura la variabilidad de escenarios, evitando la repetición de patrones y ofreciendo al jugador experiencias únicas en cada sesión.

Para que un mapa sea considerado válido, debe cumplir con las siguientes condiciones:

- Incluir un punto de inicio accesible para el jugador.
- Contener al menos un NPC alcanzable.
- Tener una ruta libre de obstáculos que conecte los objetivos de la misión con el jugador.

En la sección [4.1.2](#) se profundiza al respecto de la implementación y cómo se determina que un mapa generado es válido.

Misiones

Cada nivel está compuesto por una o varias misiones asignadas por personajes no jugables. Estas misiones se presentan mediante frases en inglés que describen el objeto que el jugador debe buscar y recolectar. Al interactuar con un NPC, el sistema debe mostrar la consigna de forma clara y legible en pantalla, utilizando un cuadro de diálogo que facilite su lectura.

Además, es necesario contar con un mecanismo que permita al jugador consultar en cualquier momento el objeto solicitado, sin necesidad de repetir la interacción con el NPC. Este recordatorio debe mantenerse visible de forma permanente durante la partida, garantizando así que el jugador pueda verificar su objetivo en cualquier momento sin interrumpir la fluidez del juego.

La resolución de una misión requiere que el jugador interprete correctamente la solicitud y entregue el objeto correspondiente al NPC emisor. Una vez cumplido el objetivo, el sistema debe registrar la misión como completada y habilitar la siguiente misión o nivel, según corresponda. En aquellos niveles donde existan varios NPCs, los mismos podrán ofrecer pistas adicionales para ayudar al jugador a comprender mejor el objetivo de la misión. Estas pistas fomentan la interacción y la exploración del entorno, promoviendo un aprendizaje contextualizado del vocabulario.

En la figura [3.3](#) se puede observar un ejemplo de diálogo donde se presenta la consigna de una misión, indicando claramente el objeto a recolectar. Además, se puede visualizar el recordatorio de qué ítem debe recolectar el usuario, siendo en este caso tres bananas.



Figura 3.3: Ejemplo de una misión

3.1.2. Requerimientos No Funcionales

Los requerimientos no funcionales establecen restricciones y condiciones que afectan al sistema en su conjunto, asegurando la accesibilidad, rendimiento y usabilidad del videojuego en el contexto educativo propuesto (Sommerville, 2011).

Consumo del ancho de banda

El juego debe ser funcional en entornos con conectividad limitada o intermitente. Para ello, el diseño debe garantizar que, tras la primera carga en el navegador, todos los archivos necesarios se descarguen localmente, permitiendo que el juego funcione de manera autónoma sin requerir una conexión permanente a Internet durante la partida.

Compatibilidad con laptops escolares

El videojuego debe ser compatible con las laptops distribuidas por el Plan Ceibal, considerando sus limitaciones de hardware y software. El desarrollo deberá utilizar tecnologías web estándar como JavaScript, asegurando la correcta visualización y ejecución en navegadores actualizados (Firefox y Chrome).

No se contempla el desarrollo de versiones específicas para dispositivos móviles o tabletas, dado que el proyecto está orientado a su uso en el ámbito escolar a través de computadoras portátiles.

3.1.3. Diseño visual

Respecto al apartado visual, es fundamental considerar diversos aspectos relacionados con el diseño y la accesibilidad de la interfaz. El estilo gráfico, la paleta de colores, la tipografía y la disposición de los elementos en pantalla son factores clave que inciden directamente en la experiencia del usuario, especialmente tratándose de un público infantil.

En particular, la tipografía debe seleccionarse teniendo en cuenta criterios de contraste, tamaño y legibilidad. Se recomienda utilizar letras en mayúsculas, con un tamaño adecuado y alto contraste respecto al fondo, para facilitar la lectura de los textos. Esto resulta especialmente importante, dado que muchos de los niños y niñas destinatarios del juego pueden no contar aún con habilidades lectoras plenamente desarrolladas.

Esto conlleva a que la interfaz del sistema debe ser intuitiva, es decir, permitir que los jugadores comprendan su funcionamiento sin necesidad de instrucciones extensas. Si bien los niños y niñas de hoy suelen tener experiencia previa en videojuegos, el diseño debe garantizar accesibilidad y claridad desde los primeros momentos de interacción.

Además, el juego debe tener coherencia visual, lo que conlleva a que es necesario definir una temática general que guíe el diseño gráfico y narrativo, de manera que todos los elementos visuales, los personajes, el mapa, los mensajes y pantallas secundarias mantengan una misma identidad estética.

Los gráficos 2D son una buena opción para llevar a cabo estos requerimientos debido a su baja demanda de recursos y su potencial atractivo visual. Este estilo es adecuado para dispositivos de hardware limitado, como las Ceibalitas, y al mismo tiempo ofrece una estética llamativa y fácilmente reconocible.

Cabe agregar que todo el contenido incluido debe respetar los derechos de autor, utilizando materiales con licencia apropiada o desarrollados específicamente para el proyecto.

3.2. Diseño técnico de la solución

3.2.1. Arquitectura

La arquitectura del sistema sigue el modelo clásico cliente-servidor, como se muestra en la figura 3.4, ampliamente utilizado en el desarrollo de aplicaciones modernas debido a su escalabilidad y flexibilidad. En este proyecto, como se dijo anteriormente en la sección 2.2 el cliente es una aplicación web construida con TypeScript utilizando la librería Phaser, mientras que el servidor está implementado en Python con el framework FastAPI. Ambos componentes están

desacoplados, lo que permite su desarrollo, prueba y despliegue de manera independiente.

El rol del servidor es el de proveer una forma de utilizar el modelo de Gemini para la generación de paráfrasis que se mencionó en la sección 2.4. Esta decisión responde a buenas prácticas de seguridad, ya que evita exponer directamente desde el cliente la clave privada del modelo, lo cual podría derivar en usos indebidos o vulnerabilidades. En el capítulo 4.3 se exponen las capacidades que brinda el servidor.

En una primera instancia, el cliente se comunicaba en tiempo real con el servidor para obtener paráfrasis de frases predefinidas al momento de cargar el nivel. Sin embargo, esta solución no resultó eficiente, ya que incrementaba considerablemente los tiempos de carga dentro del juego y, además, aumentaba la dependencia de la conexión a Internet.

Debido a estas limitaciones, se decidió eliminar dicha comunicación en tiempo de ejecución y utilizar el servidor únicamente como una herramienta para generar las paráfrasis de manera independiente al juego. La implementación del servidor no fue modificada, ya que, al tratarse de una API, conserva la posibilidad de ser consumida por terceros en el futuro si así se desea.

También se utilizó Docker para simplificar el desarrollo y despliegue de la aplicación. Docker¹ es una plataforma de software que permite empaquetar aplicaciones junto con todas sus dependencias en unidades llamadas contenedores. Estos contenedores son entornos aislados y portables que pueden ejecutarse de manera consistente en diferentes sistemas operativos y máquinas, garantizando que la aplicación funcione de forma idéntica en cualquier entorno. Esto facilita enormemente el despliegue, la escalabilidad y el desarrollo, minimizando conflictos entre desarrolladores.

¹<https://www.docker.com/> Última vez accedido: 03/08/2025

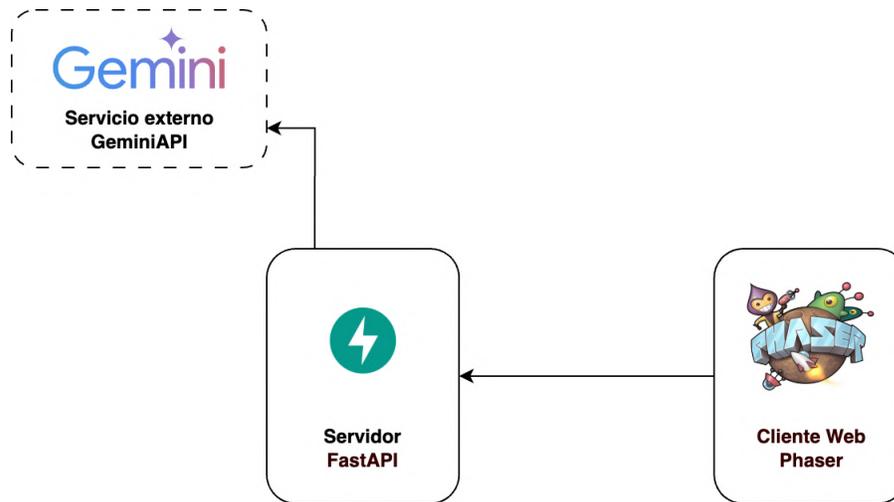


Figura 3.4: Arquitectura del sistema

Uno de los principales objetivos en el diseño técnico del juego era que fuera fácilmente extensible y modificable, incluso por personas sin grandes conocimientos técnicos. Si en el futuro se desea agregar o modificar niveles, el proceso debería ser simple.

Para lograr esto, se diseñó un sistema de misiones y textos basado en archivos JSON (JavaScript Object Notation). Este formato permite describir las misiones de forma estructurada, facilitando la incorporación de nuevo contenido sin necesidad de modificar el código fuente. El fragmento de código 1 muestra un ejemplo del JSON utilizado.

Para la persistencia de datos no se utilizó una base de datos centralizada. Toda la información relacionada con el progreso del jugador se almacena localmente en el navegador, mediante *Local Storage*. El Local Storage se puede pensar como una “pequeña base de datos” que tienen los navegadores para guardar información relacionada con un sitio web. Es una solución suficiente y eficiente para los requerimientos del proyecto, evitando complejidades innecesarias.

De esta manera, si el usuario cierra o recarga la página, puede retomar la partida desde el punto exacto donde la dejó. En particular, el mapa cargado será el mismo que el generado anteriormente, ya que, de lo contrario, se generaría uno distinto al tratarse de un sistema de generación dinámica.

Estructura del código

El código del proyecto fue estructurado siguiendo el paradigma de programación orientada a objetos (POO), seleccionado por su capacidad de promover la reutilización y facilitar la organización lógica del sistema.

Durante las etapas de análisis y diseño se identificó que la lógica entre los distintos niveles del juego era, en su mayor parte, común, con algunas variaciones específicas. Estas diferencias podían ser encapsuladas dentro de la configuración propia de cada nivel, lo que permitió definir una clase base que concentrara toda la lógica compartida, mientras que las particularidades de cada nivel quedaban delegadas a sus respectivas configuraciones.

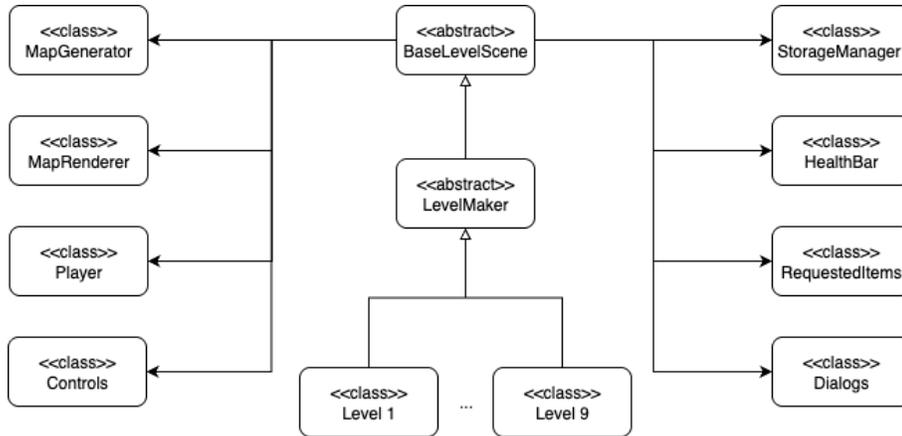


Figura 3.5: Clases principales del nivel

Como se puede observar en el esquema de la figura 3.5, los niveles heredan de *BaseLevelScene*, clase central del juego que se relaciona directamente con todas las clases clave del sistema, tales como la barra de vida, los controles del jugador, entre otras. A su vez, esta clase hereda de *Phaser.Scene*, la clase proporcionada por Phaser para la gestión de escenas. Estas clases serán abordadas en profundidad en el siguiente capítulo dedicado a la implementación.

3.2.2. Flujos

Flujo general

El juego presenta un flujo general simple y lineal, sin mayores ramificaciones.

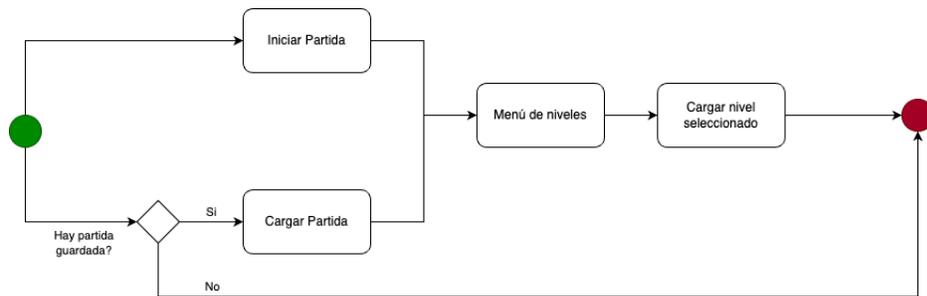


Figura 3.6: Flujo general del juego

Como se observa en la figura 3.6, cuando el jugador comienza a jugar, tiene la posibilidad de cargar una partida guardada (si existe) o iniciar una nueva. Luego accede a la pantalla de selección de niveles, donde puede visualizar todos los niveles disponibles, así como su estado (bloqueado o desbloqueado). Al seleccionar un nivel, este se carga efectivamente.

Debido a la simplicidad del juego, se decidió que no era necesario contar con una pantalla de configuración adicional.

Flujo de misiones

Las misiones están definidas mediante un archivo JSON que especifica los estados posibles junto con sus textos asociados, la cantidad de ítems a recolectar y el asset que será el objeto buscado.

Como se puede ver en el fragmento de código 1, dentro de la clave `dialog` hay dos misiones diferentes. La primera de ellas comienza con *"My pigeon flew away"*, mientras que la segunda *"Hi! I'm a friendly scientist!"*. A su vez, se pueden apreciar sus distintas etapas, el `assetKey` que es la clave (nombre) del objeto que se va a renderizar y que el jugador tiene que recolectar, y la `quantityToCollect` que indica la cantidad de ítems.

Cabe agregar que la clave "dialogs" hace referencia al componente visual que se encarga de gestionar las misiones, es decir, por un lado abarca la parte visual del renderizado y, por otro, la gestión de los distintos estados.

Fragmento de código 1 Estructura de una misión en formato JSON

```
{
  "title": "Level 5",
  "dialogs": [
    {
      "questStart": [
        "My two pigeons flew away!",
        "Can you help me find them?"
      ],
      "questInProgress": ["Still looking for my pigeons!"],
      "questPartiallyCompleted": [
        "Great, you found one pigeon! Just one more to go!"
      ],
      "questWrongItem": ["That's not a pigeon!"],
      "questFinished": ["Thank you! The pigeons are safe now!"],
      "questHints": ["Look for the white flying birds"],
      "assetKey": "PIGEON",
      "quantityToCollect": 2
    },
    {
      "questStart": [
        "Hi! I'm a friendly scientist!",
        "Can you help me find three tiny frogs?",
        "I want to learn about how they jump!"
      ],
      "questInProgress": ["Have you seen any jumping frogs?"],
      "questPartiallyCompleted": [
        "Yay! You're doing great! Let's find more frogs!"
      ],
      "questWrongItem": [
        "Oops! That's not a frog - I'm looking for green jumping
        friends!"
      ],
      "questFinished": [
        "Wow! Thank you! Now I can watch these cute frogs jump!"
      ],
      "questHints": ["Search for tiny green creatures that hop
        in the grass!"],
      "assetKey": "FROG",
      "quantityToCollect": 3
    }
  ]
}
```

Los diálogos también tienen la capacidad de manejar distintas variantes de un mismo enunciado para generar dinamismo, aspecto que será profundizado en la sección 4.1.9. Esta característica permite agregar distintos textos, incluyendo aquellos generados por Gemini en el servidor.

Respecto a las misiones, las mismas van siendo tomadas en orden y se considera que un nivel está completo cuando todas ellas han sido completadas. En el esquema 3.7 se presenta el flujo de estados de una misión.

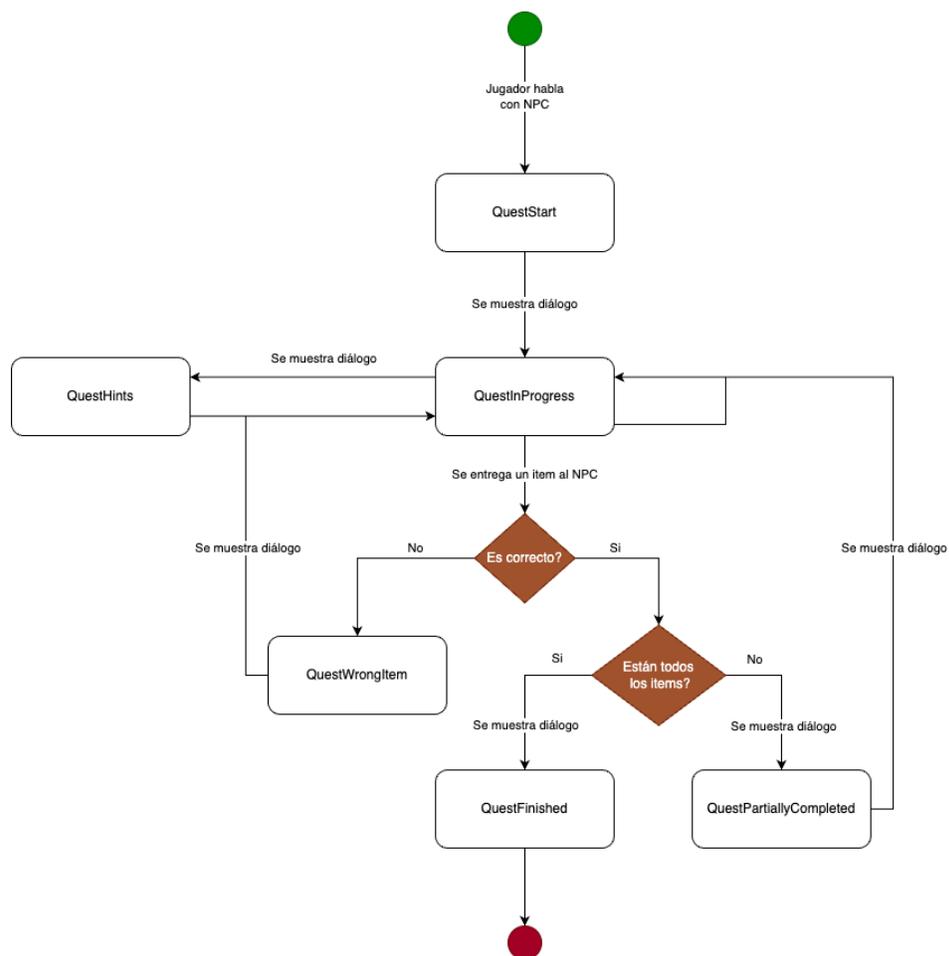


Figura 3.7: Flujo de una misión

Cuando un jugador interactúa con un NPC sin haber comenzado ninguna misión, se inicia el estado `QuestStart`, mostrando el texto inicial de la misión. Una vez desplegado este diálogo, el estado pasa automáticamente a `QuestInProgress`.

En este estado se presentan varias posibilidades:

- Si el jugador entrega un ítem correcto pero aún falta recolectar otros, se despliega el estado `QuestPartiallyCompleted`.
- Si entrega un ítem incorrecto, se muestra `QuestWrongItem`.
- Puede interactuar nuevamente con el NPC original para refrescar la información, mostrándose `QuestInProgress`.
- Puede interactuar con otro NPC para obtener pistas adicionales (`QuestHints`).

En todos estos casos, tras mostrar el diálogo, el sistema vuelve al estado `QuestInProgress`.

Cuando el jugador completa todos los ítems solicitados, se transiciona al estado `QuestFinished`, donde se despliega un mensaje final y la misión se marca como completada. Si es la última misión, se desbloquea el siguiente nivel; en caso contrario, se mantiene en espera hasta que el jugador solicite la siguiente misión.

3.3. Síntesis

En este capítulo se ha detallado el proceso de definición y diseño del videojuego educativo propuesto, *Climb the Cinacina*, estableciendo las bases conceptuales y técnicas que guiaron su desarrollo. El principal desafío fue concebir una experiencia lúdica que, además de ser educativa, fuese adaptable ya que es uno de los objetivos del proyecto, como se planteó en [1.1](#).

La solución propuesta combina la rejugabilidad del género roguelike con misiones de aprendizaje de inglés, estructuradas en nueve niveles con dificultad progresiva y generación procedural de escenarios para mantener el interés del jugador. Se establecieron requerimientos funcionales claros que definen las mecánicas de juego, como el control del personaje, la interacción con NPCs y la gestión de misiones, y requerimientos no funcionales que priorizan el bajo consumo de ancho de banda y la compatibilidad con el hardware objetivo.

Arquitectónicamente, se optó por un diseño que, si bien inicialmente contemplaba una comunicación en tiempo real entre el videojuego y el servidor, fue descartado para garantizar la jugabilidad offline. El uso de archivos JSON para la configuración de misiones y el almacenamiento local para el progreso del jugador refuerzan la modularidad y la simplicidad, facilitando futuras expansiones del juego.

En síntesis, este capítulo ha establecido las bases para la implementación de “Climb the Cinacina”, definiendo un juego roguelike con un enfoque educativo para niños.

Capítulo 4

Implementación

En este capítulo se presentan los principales aspectos de la implementación, describiendo su funcionamiento y las decisiones técnicas tomadas durante el desarrollo. Se abordan tanto la aplicación web, es decir, el juego en sí, como el servidor, que, si bien cumple un rol secundario dentro del sistema, resulta fundamental.

En lo que respecta al juego, se explican las funcionalidades principales que provee el framework, así como el diseño y la lógica detrás de las clases `BaseLevelScene` y `LevelMaker`, que cumplen un rol central en la creación de niveles. Además, se describen otros componentes que, si bien son secundarios, tienen un impacto directo en la experiencia de juego, como `HealthBar`, `RequestedItems`, `Player`, el sistema de controles, entre otros.

4.1. Aplicación web

4.1.1. Phaser

Uno de los conceptos principales que posee el framework es el de `Scene`, el cual constituye una de las clases fundamentales del mismo. Puede pensarse como la representación de una “pantalla” dentro del juego. Por ejemplo, el menú principal, cada nivel o la pantalla de victoria. Todas estas son distintas `Scenes` y cada una encapsula su propia lógica. Tiene un ciclo de vida en donde se destacan tres métodos principales:

- `preload()`: encargado de cargar todos los recursos necesarios antes de que la escena comience. Esto incluye imágenes, spritesheets, archivos de audio, fuentes, entre otros recursos.
- `create()`: se ejecuta una vez que todos los recursos han sido cargados. En este método se crean y posicionan los distintos objetos que compondrán la escena: sprites, textos, personajes, físicas, etc.

- **update()**: se ejecuta en cada frame (cuadro) del juego. En él, se controla la lógica principal del juego, como el movimiento de los personajes, la detección de colisiones, las actualizaciones de estado y demás comportamientos dinámicos. Este método se ejecuta de forma continua, pudiéndose asimilar a un ciclo `while(true)`, hasta que ocurre algún evento que determine el fin de la escena.

En resumen, `preload()` permite cargar los recursos necesarios; `create()` inicializa y muestra los elementos de la escena; y `update()` gestiona la lógica del juego en tiempo real. Como cada Scene representa una pantalla distinta, con su propia lógica embebida para transicionar de una a otra, se utiliza el método `this.scene.start(scene_id)`. Cuando se transiciona, se ejecuta el ciclo de vida de la siguiente Scene y la actual se destruye, aunque existen mecanismos para preservarlas, pero no se utilizan en el proyecto.

Otro componente fundamental de Phaser es el objeto **Game**, el cual constituye el núcleo del juego. Esto repercute en que todo el juego ocurra dentro de dicho objeto, y desde el punto de vista del HTML sea simplemente una etiqueta en vez de un extenso y complejo HTML, como se puede observar en el fragmento de código 2.

Fragmento de código 2 Código HTML del juego

```
<html lang="en" style="height: 100%;">
  <head> ... </head>
  <body style="height: 100%;">
    <div id="app">
      <div id="game-container"
        width: 100%;
        height: 100%;">
        <canvas
          width="1920"
          height="1040"
          style="image-rendering: pixelate">
        </canvas>
      </div>
    </div>
  </body>
</html>
```

En este fragmento, además de los elementos estándar de HTML, se destaca el elemento `<canvas>`, el cual está directamente asociado al objeto **Game**. Esta asociación se realiza mediante la propiedad `parent`, que define el contenedor del `canvas` (en este caso, el div con id `game-container`). La etiqueta `canvas` permite renderizar gráficos y animaciones de forma dinámica dentro del navegador.

Como se observa en el fragmento de código 3, al crear el objeto `Game` se definen varias propiedades, siendo las más relevantes: `type`, `pixelArt`, `scene`.

La propiedad `type` define el sistema de renderizado a utilizar, pudiendo tomar los valores: `canvas`, `webgl` o `auto`. Si se define la propiedad como `webgl`, se fuerza el uso de WebGL. Si es `canvas`, se utilizará el renderizado Canvas; mientras que `auto` permite que el navegador elija la mejor opción soportada, siendo prioritario WebGL. WebGL soporta renderizado 3D y generalmente ofrece mejor rendimiento, mientras que Canvas es más limitado pero más ampliamente soportado, especialmente para gráficos 2D.

En cuanto al parámetro `pixelArt` permite habilitar o deshabilitar el suavizado de imágenes y sprites al ser escalados o transformados. Su activación es clave en este proyecto, ya que permite mantener los bordes nítidos y los píxeles bien definidos, logrando una estética de píxeles definidos, propia de los videojuegos clásicos. Por defecto, los navegadores aplican suavizado, lo que hace que los bordes se vean borrosos o difusos.

Por último, la propiedad `scene` es un array que contiene todas las posibles pantallas que tiene el videojuego. Por defecto, el primer elemento del array es la primera pantalla que se muestra al iniciar el juego, siendo en este caso la pantalla de **Preloader**. En ella, se visualiza una barra de carga mientras se cargan todos los assets utilizados en el juego.

Fragmento de código 3 Configuración empleada en el objeto Game

```
const config: Types.Core.GameConfig = {
  type: Phaser.WEBGL,
  pixelArt: true,
  backgroundColor: "#000F00",
  scale: {
    parent: "game-container",
    width: GAME_DIMENSIONS.WIDTH,
    height: GAME_DIMENSIONS.HEIGHT,
    mode: Phaser.Scale.FIT,
    autoCenter: Phaser.Scale.CENTER_BOTH,
  },
  scene: [
    Preloader,
    MainMenu,
    LevelsMenu,
    Level1,
    Level2,
    ...,
    Level9,
    GameOver,
    WinScene,
    EndScene,
  ],
};
```

4.1.2. Generación del mapa dinámico

Como se presentó en la sección 2.3.2, el algoritmo implementado para la generación dinámica de mapas se basa en el enfoque conocido como **Space Partitioning**. No obstante, su implementación requirió varias adaptaciones para ajustarse a las necesidades específicas del proyecto.

El proceso de generación del mapa se divide en cinco etapas principales, que se describen a continuación:

1. Generación de particiones.
2. Garantizar interconexión.
3. Generación de habitaciones.
4. Conexión de habitaciones.
5. Asignación de assets.

Representación de la información

Para inicializar un nivel se utiliza una configuración que define tanto las dimensiones del mapa como los distintos tipos de elementos que lo poblarán. Esta configuración permite al sistema de generación procedural crear entornos variados y controlados. A continuación se describen sus componentes principales:

- **Nombre del mapa (*name*):** Identificador único del nivel, útil para su registro o carga en distintas escenas del juego. Ejemplo: `SceneKeys.LEVEL_1`.
- **Dimensiones del mapa (*dimensions*):** Define el tamaño total del mapa, expresado en cantidad de tiles a lo ancho y alto. Ejemplo: `width: 80, height: 60`
- **minPartitionSize:** tamaño mínimo que puede tener una partición del mapa..
- **minRoomSize:** tamaño mínimo permitido para una habitación. Estas restricciones aseguran que las habitaciones generadas sean transitables y útiles.

Cada tile, o elemento gráfico del mapa, se clasifica según su función. Esta clasificación se define mediante el enumerado `TileType` y determina tanto su comportamiento como la forma en que interactúa con el jugador. En el fragmento de código 4 se presenta el enumerado implementado.

Fragmento de código 4 Enumerado `TileType`

```
export enum TileType {
  WALKABLE_SPACE = "WALKABLE_SPACE",
  // Espacios transitables.
  INTERACTIVE_STATIC_OBJECT = "INTERACTIVE_STATIC_OBJECT",
  // Objetos con los que se puede
  // interactuar, pero no se pueden atravesar como NPCs.
  INTERACTIVE_OBJECT = "INTERACTIVE_OBJECT",
  // Objetos interactivos atravesables bajo ciertas condiciones,
  // por ejemplo, recoger un ítem.
  OBSTACLE = "OBSTACLE",
  // Objetos completamente intransitables.
}
```

Una vez generado, el mapa se representa mediante una matriz de dimensiones $m \times n$, donde cada celda contiene un par de valores: el tipo de tile (`TileType`) y el identificador del asset correspondiente. El tipo de tile se utiliza posteriormente durante la etapa de renderizado para definir el comportamiento o la interacción permitida sobre esa celda.

Generación de particiones

Como se explicó en la Sección 2.3.2, el algoritmo comienza creando una partición raíz que abarca todo el mapa. A partir de allí, se divide recursivamente en subparticiones, primero intentando dividir horizontalmente y luego verticalmente.

Una partición puede dividirse siempre que su tamaño sea al menos el doble de `minPartitionSize`, asegurando que las subparticiones cumplan con el mínimo requerido. La ubicación exacta de la división se determina aleatoriamente dentro del rango válido. En la imagen 4.1 se muestra un ejemplo de la selección de un corte sobre una partición de tamaño 12. El tamaño mínimo permitido para la partición es 4, por lo tanto, la región factible para realizar el corte debe ser mayor que 4 para garantizar que la partición a la izquierda sea mayor al mínimo y menor que 8 para garantizar que la partición a la derecha cumpla con la misma condición.

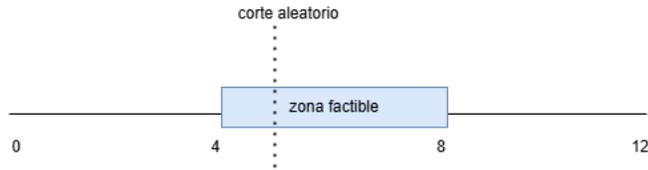


Figura 4.1: Selección aleatoria del punto de corte

Una representación visual luego de varias iteraciones de este proceso se puede ver en la figura 4.2.

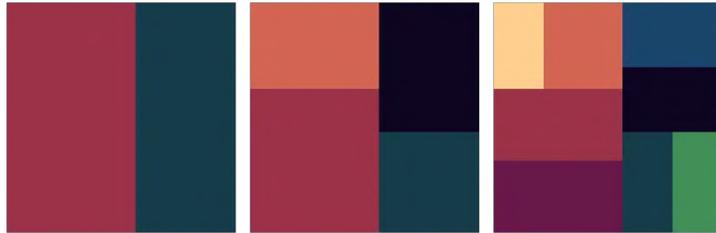


Figura 4.2: Proceso de particionado

Garantizar interconexión y generación de habitaciones

Una vez generada la estructura de particiones, el siguiente paso es asignar una habitación a cada hoja del árbol de particiones. Cada habitación es un espacio rectangular cuya posición y dimensiones se determinan aleatoriamente, respetando el tamaño mínimo `minRoomSize`. Los límites de la habitación definen las paredes, y el interior es considerado espacio transitable (`WALKABLE_SPACE`).

Para evitar regiones inaccesibles, es esencial garantizar la interconexión entre todas las habitaciones. Esto se logra mediante un recorrido en profundidad (Depth

First Search (Kleinberg y Tardos, 2006)) sobre el árbol de particiones, conectando las habitaciones de los nodos hijos en cada paso a través de pasillos. En el código 5 se muestra el fragmento principal de esta lógica. De esta forma, cada habitación está asociada exclusivamente a un nodo hoja del árbol, como se muestra en la figura 4.3

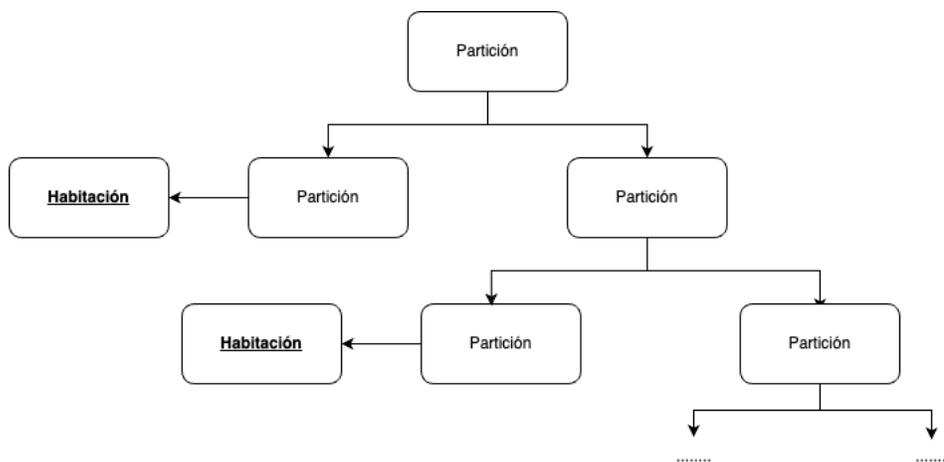


Figura 4.3: Relación entre particiones y habitaciones

Fragmento de código 5 fillAndConnectRooms: conexión de habitaciones

```
private static fillAndConnectRooms(  
    matrix: number[][],  
    partition: Partition,  
): void {  
    if (leftRoom && rightRoom) {  
        this.connectRooms(matrix, leftRoom, rightRoom);  
    }  
  
    this.fillAndConnectRooms(matrix, partition.left);  
    this.fillAndConnectRooms(matrix, partition.right);  
}
```

La conexión de habitaciones se realiza alineando primero las coordenadas X y luego las Y, “dibujando” el pasillo sobre la matriz, como puede verse en el pseudocódigo 2.

Algoritmo 2 Conectar habitaciones mediante pasillos

```
1: procedure CONECTARHABITACIONES(puntoA, puntoB, matriz)
2:   while puntoA.x ≠ puntoB.x do
3:     matriz[puntoA.y][puntoA.x] ← CELDA_OCUPADA
4:     if puntoA.x < puntoB.x then
5:       puntoA.x ← puntoA.x + 1
6:     else
7:       puntoA.x ← puntoA.x - 1
8:     end if
9:   end while
10:  while puntoA.y ≠ puntoB.y do
11:    matriz[puntoA.y][puntoA.x] ← CELDA_OCUPADA
12:    if puntoA.y < puntoB.y then
13:      puntoA.y ← puntoA.y + 1
14:    else
15:      puntoA.y ← puntoA.y - 1
16:    end if
17:  end while
18: end procedure
```

Asignación de assets

Una vez definida la estructura del mapa (habitaciones y pasillos), se procede a la asignación de assets a cada celda de la matriz. Esto puede realizarse de dos maneras: por frecuencia o por cantidad fija.

- **Assets por frecuencia:** Se define la probabilidad de aparición de cada tipo de asset, y se asigna a cada celda de forma aleatoria de acuerdo a estas probabilidades. Por ejemplo, si se asigna 50 % de probabilidad a **GRASS** y 50 % a **FLOWER_GRASS**, la distribución resultante tenderá a ser equitativa.
- **Assets por cantidad fija:** En este caso, se especifica la cantidad exacta de un asset a colocar, y se distribuye aleatoriamente entre las habitaciones, procurando lograr una distribución uniforme en todo el mapa.

Esta flexibilidad permite no solo lograr una apariencia variada y natural, sino también garantizar la presencia de elementos clave en ubicaciones específicas, aspecto fundamental en el diseño de niveles para videojuegos. En la figura 4.4 se muestran dos configuraciones distintas:

En 4.4a, la configuración asigna un 75 % del espacio al asset *flower_grass* y el 25 % restante al asset *grass*. En cambio, en 4.4b, la asignación es equitativa, con un 50 % para cada asset.



(a) Configuración 75 % y 25 % para 2 assets distintos.



(b) Configuración 50 % para ambos assets.

Figura 4.4: Renderizado de un espacio del tipo `WALKABLE_SPACE` con configuraciones distintas para ambos assets.

Validaciones y robustez

El algoritmo incorpora diversas validaciones para garantizar que la generación sea válida y coherente:

- El mapa debe ser al menos dos veces más grande que el tamaño mínimo de partición.
- Las configuraciones de los tiles no pueden estar vacías.
- El tamaño mínimo de la habitación debe ser menor o igual al de partición

Si alguna de estas condiciones no se cumple, el sistema arroja un error descriptivo, evitando inconsistencias en la generación. En caso de fallar alguna validación, el mapa vuelve a generarse, aunque en condiciones normales esto no debería ocurrir, dado que los parámetros están predefinidos para cumplir los requisitos.

Resumen del proceso

En resumen, el proceso de generación dinámica del mapa se puede resumir en los siguientes pasos:

1. Inicialización de la estructura de datos y validación de parámetros.
2. Generación recursiva de particiones hasta alcanzar el tamaño mínimo.
3. Asignación aleatoria de habitaciones dentro de cada partición hoja.
4. Conexión de todas las habitaciones mediante pasillos para garantizar accesibilidad global.

5. Asignación de assets a cada celda, según criterios de frecuencia o cantidad fija.
6. Cálculo de la posición inicial del jugador y de la ubicación de objetos interactivos o estáticos.

De esta forma, el algoritmo implementado asegura la creación de mapas jugables, variados y coherentes, con todos los elementos necesarios distribuidos adecuadamente y evitando la existencia de áreas inaccesibles.

4.1.3. Renderizado del mapa dinámico

Una vez que se genera la representación lógica del mapa, se procede al renderizado.

Fragmento de código 6 Método principal del renderizado

```
static render(scene: Scene, map: MapStructure): void {
    const {
        dimensions: { width: numberOfColumns, height: numberOfRows },
    } = map;
    for (let row = 0; row < numberOfRows; row += 1) {
        for (let column = 0; column < numberOfColumns; column += 1) {
            const x = TILE_SIZE / 2 + column * TILE_SIZE;
            const y = TILE_SIZE / 2 + row * TILE_SIZE;

            const tile: TileConfig = map.tiles[row]![column]!;

            this.ensureAssetGroup(scene, map, tile.type, tile.assetKey);
            this.renderTile(
                scene,
                map,
                x,
                y,
                tile.type,
                tile.assetKey,
                tile.frame
            );
        }
    }
}
```

Como se puede ver en el fragmento 6, se recorre celda por celda de la matriz en donde se calcula su posición en la pantalla y se selecciona el asset correspondiente mediante el método `renderTile()`.

Los assets se organizan en grupos según su tipo mediante un método que utiliza internamente la propiedad `staticGroup` proporcionada por Phaser. Un `staticGroup` es un tipo de datos pensado para objetos estáticos y que no responden a las fuerzas físicas del entorno, como la gravedad o colisiones dinámicas. Este tipo de grupos es ideal para representar elementos del escenario como paredes, suelos o decoraciones fijas.

La agrupación en `staticGroup` resultó fundamental ya que permitió definir comportamientos específicos y optimizar el rendimiento del juego, evitando cálculos innecesarios en objetos que no requieren físicas activas.

Un ejemplo de agrupación son los tiles del tipo `INTERACTIVE_STATIC_OBJECT` y `OBSTACLE` debido a que son colisionables pero sin necesidad de una física dinámica, ya que no se desplazan luego de ser colisionados.

Por último, para estandarizar el renderizado, todos los assets tienen el mismo tamaño y su posición se calcula como:

$$\text{posición} = \frac{\text{TILE_SIZE}}{2} + \text{númeroDeFilaOColumna} \times \text{TILE_SIZE} \quad (4.1)$$

4.1.4. Niveles

Como se mencionó en la sección 3.2.1, se definió una arquitectura que distingue dos componentes clave: por un lado, la clase `BaseLevelScene`, encargada de contener la lógica común a todos los niveles y, por otro lado, la clase `LevelMaker`, responsable de crear y agregar comportamiento específico de cada nivel. Esta separación responde al principio de responsabilidad única, el cual es explicado por [Martin \(2009\)](#) y busca favorecer la encapsulación y la reutilización de código.

Cada nivel en concreto se implementa como una instancia de `LevelMaker`, a la cual se le asignan ciertos atributos que definen su comportamiento. En el diagrama de clases 3.5 puede observarse la relación entre las distintas clases y cómo se integran con el resto del sistema. A continuación, se abordarán ambas clases.

Clase `LevelMaker`

Esta clase tiene las siguientes responsabilidades principales:

1. Invocar la escena base: `BaseLevelScene`.
2. Crear las animaciones específicas del nivel.
3. Configurar las colisiones correspondientes.
4. Agregar comportamiento al ciclo de vida de la `Scene`

Para crear un nivel, la clase `LevelMaker` requiere recibir un objeto de tipo `LevelConfig`, tal como se muestra en el fragmento de código 7.

Fragmento de código 7 Tipos de entrada el método `LevelMaker`

```
export type MapConfiguration = {
  name: string;
  tilesConfig: TileConfig[];
  defaultFloorAsset: TileConfig;
  dimensions: {
    width: number;
    height: number;
  };
  minPartitionSize: number;
  minRoomSize: number;
};

export type MinimalMapConfiguration = Pick<
  MapConfiguration,
  "name" | "tilesConfig" | "defaultFloorAsset"
> &
  Partial<Omit<MapConfiguration, "name"
    | "tilesConfig" | "defaultFloorAsset">>;

export type LevelConfig = MinimalMapConfiguration & {
  onPreload?: (scene: Phaser.Scene) => Promise<void>;
  onCreate?: (scene: Phaser.Scene) => Promise<void>;
};
```

Este objeto permite personalizar el comportamiento de las funciones del ciclo de vida de la `Scene`, a través de los métodos `onPreload()` y `onCreate()`. Esta flexibilidad resulta especialmente útil en casos donde un nivel necesita un comportamiento particular. Por ejemplo, el nivel 1 se diferencia del resto porque, ya que al ser el primer nivel que juega el usuario, muestra las instrucciones de cómo jugar. Dicho mensaje se puede ver en la figura 4.10e.

Además de las propiedades ya descritas en la sección 4.1.2, se destaca especialmente la propiedad `tilesConfig`, la cual contiene la definición de cómo será el mapa que se generará. En el fragmento 8 se puede observar la configuración utilizada para el nivel 1 y el resultado de ella se puede ver en la figura 4.5.

Fragmento de código 8 Objeto TileConfig del nivel 1

```
tilesConfig: [  
  {  
    type: TileType.INTERACTIVE_STATIC_OBJECT,  
    assetKey: CharacterAssets.GUY.ASSET_KEY,  
    frame: CharacterAssets.GUY.FRAME,  
    quantity: 1,  
  },  
  {  
    type: TileType.INTERACTIVE_STATIC_OBJECT,  
    assetKey: CharacterAssets.GIRL.ASSET_KEY,  
    frame: CharacterAssets.GIRL.FRAME,  
    quantity: 1,  
  },  
  {  
    type: TileType.WALKABLE_SPACE,  
    assetKey: TileKeys.GRASS,  
    frequency: 50,  
  },  
  {  
    type: TileType.WALKABLE_SPACE,  
    assetKey: TileKeys.FLOWER_GRASS,  
    frequency: 50,  
  },  
  {  
    type: TileType.OBSTACLE,  
    assetKey: TileKeys.TREE,  
    frequency: 3,  
  },  
  {  
    type: TileType.INTERACTIVE_OBJECT,  
    assetKey: ItemAssets.APPLE.assetKey,  
    quantity: 1,  
    initialState: ItemState.HIDDEN,  
    isAnimated: true,  
  },  
],
```

Se puede ver que hay dos NPCs definidos, *Guy* y *Girl*, así como dos tipos de tiles del tipo `WALKABLE_SPACE`: *Grass* y *Flower*, y uno del tipo `OBSTACLE`: *Tree*, cuya cantidad se determina mediante el parámetro de frecuencia. Además, se define un objetivo interactivo *Apple*, que representa el ítem que debe ser recolectado en la misión. Este objeto está configurado para que comience oculto y cuente con una animación.



Figura 4.5: Mapa renderizado basado en la configuración del fragmento 8

Cada uno de los elementos del arreglo es de tipo `TileConfig`, cuya definición se muestra en el fragmento 9. En dicha estructura se especifica el `TileType`, tal como se describió en 4, junto con el `assetKey`, la frecuencia o cantidad del asset, su estado inicial (es decir, si se muestra oculto o visible al comenzar el nivel) y si posee animación o no. De esta manera es que se logra personalizar los distintos tipos de tiles de manera general.

Fragmento de código 9 Estructura del `TileConfig`

```
export type TileConfig = {
  type: TileType;
  assetKey: string;
  frequency?: number;
  quantity?: number;
  initialState?: ItemState;
  isAnimated?: boolean;
};
```

Dado que cada nivel puede tener sus propios recursos gráficos, se definió que la clase `LevelMaker` sea la encargada de crear las animaciones. Esta decisión permite una gestión más flexible y coherente de los assets visuales, además de centralizar la lógica en una única clase, favoreciendo la mantenibilidad del código.

Además, si bien actualmente la lógica de colisiones es común a todos los niveles, se prevé la posibilidad de redefinirla o extenderla en el futuro, de forma similar a lo que sucede con los métodos `onPreload()` y `onCreate()`. En el pseudocódigo 3 se muestra cómo se utiliza el método `setupCollisions()` para indicar que los elementos definidos como `obstacles` deben ser colisionables, mientras que los `interactiveItems` pueden ser recogidos por el jugador.

Algoritmo 3 Manejo de colisiones

```
1: procedure SETUPCOLLISIONS(LevelMaker)
2:   obstaculos ← elementosEstáticos
3:   Llamar a setupCollisions() de la clase base
4:   for all item en elementosInteractivos do
5:     Hacer que el item sea arrastrable
6:   end for
7: end procedure

8: procedure SETUPCOLLISIONS(BaseLevelScene)
9:   if la lista de obstáculos está vacía then
10:    return
11:  end if
12:  gruposDeColisión ← lista de grupos asociados a los obstáculos
13:  for all grupo en gruposDeColisión do
14:    Agregar colisión entre el jugador y el grupo
15:  end for
16: end procedure
```

4.1.5. BaseLevelScene

Como se aprecia en el diagrama de clases (ver figura 3.5), `BaseLevelScene` es la clase núcleo del juego y es la que orquesta el funcionamiento de un nivel.

El método `preload()` se encarga de generar un mapa únicamente si no existe uno generado previamente, debido a que guarda la partida cada vez que se comienza un nuevo nivel. Luego, carga los recursos necesarios para la escena y crea las animaciones correspondientes.

Luego, en el método `create()` se renderiza el mapa previamente generado, se inicializan los componentes visuales del videojuego, incluyendo el personaje del jugador y sus controles. También se configura la cámara, responsable de definir cómo el jugador percibe el entorno del juego. En el fragmento 10 se puede visualizar este método.

Fragmento de código 10 Método create del BaseLevelScene

```
protected async create(): Promise<void> {
  try {
    await this.initializeScene();
  } catch (error) {
    console.error("Failed to create scene:", error);
  }
}

private async initializeScene(): Promise<void> {
  MapRenderer.render(this, this.map);

  await Promise.all([
    this.initializePlayer(),
    this.initializeDialogs();
    this.initializeHealthBar();
    this.initializeCamera(),
  ]);

  this.setupCollisions();
}
```

Finalmente, se configura la lógica de colisiones. Esto implica tomar los grupos de assets antes mencionados en la sección 4.1.3 y verificar si deben ser agregados a la capa de físicas o no. Los grupos de assets que colisionan con el jugador son: `OBSTACLE`, `INTERACTIVE_OBJECT` e `INTERACTIVE_STATIC_OBJECT`.

Como no todos los elementos deben necesariamente comportarse de la misma manera ante una colisión, en esta etapa se definen los escenarios y acciones que desencadena cada tipo de colisión.

El método `update()` se encarga de gestionar las acciones del jugador en tiempo real, así como de actualizar distintos elementos visuales, como por ejemplo la barra de vida. Tal como se explicó en la Sección 4.1.1, este método se ejecuta de forma continua durante cada ciclo del juego (similar a un loop infinito). En el fragmento de código 11 se puede ver su implementación.

Fragmento de código 11 Método update del BaseLevelScene

```
update(): void {
    if (!this.controls) return;

    if (this.controls.wasSpaceKeyPressed()) {
        if (this.dialog?.isVisible) {
            this.dialog.showNextMessage();
            return;
        }

        this.tryToInteractWithNearNPC();

        if (this.heldItem && !this.dialog?.isVisible) {
            this.tryDropHeldItem();
        }

        return;
    }

    if (this.dialog?.isVisible) return;

    this.player.move(this.controls.getDirectionKeyPressed());

    if (this.heldItem) {
        this.heldItem.setPosition(this.player.x, this.player.y);
    }

    if (this.dialog?.areAllDialogsCompleted
        && !this.closingSceneStarting) {
        this.closingSceneStarting = true;
        this.closingScene();
    }
}
```

Durante cada iteración, se verifica si el jugador presionó la tecla de acción espacio mediante el método `wasSpaceKeyPressed()`. Si hay un diálogo abierto, se muestra el siguiente mensaje utilizando `dialog.showNextMessage()`. En caso contrario, se intenta interactuar con un NPC cercano mediante el método `tryToInteractWithNearNPC()`.

En cambio, si el jugador tiene un objeto en su poder y no hay un diálogo visible, se intenta dejar el ítem utilizando el método `tryDropHeldItem()`.

A continuación, se actualiza la posición del jugador según la tecla de dirección

presionada, mediante el método `player.move()`. Si el jugador sostiene un ítem, este se mantiene sincronizado con su posición.

Finalmente, si todos los diálogos han sido completados y aún no se ha iniciado la escena de cierre, se activa la transición llamando al método `closingScene()`.

4.1.6. Player

La clase `Player` representa al personaje principal controlado por el usuario y extiende de la clase `Phaser.Physics.Arcade.Sprite`. Esta clase base, provista por Phaser, encapsula las funcionalidades de un objeto visual `sprite` que puede interactuar con el sistema de físicas del juego.

Heredar de `sprite` permite al personaje ser dibujado en pantalla, tener animaciones, colisionar con otros objetos y desplazarse dentro del entorno mediante un sistema de físicas.

El jugador puede desplazarse libremente por el entorno utilizando las teclas de dirección del teclado. La dirección y velocidad del desplazamiento se controlan a través del método `move()`, que también actualiza la animación correspondiente.

Las animaciones del jugador están basadas en un `spritesheet` definido previamente en el `preloader` del juego. Cada dirección de movimiento (arriba, abajo, izquierda, derecha) tiene su propia animación, que se activa cuando el jugador se mueve en esa dirección. Al detenerse, se muestra un cuadro estático correspondiente a la última dirección registrada. Esto se gestiona mediante el método `setAnimation()` y la función `anim.play()`, lo que garantiza una transición fluida entre estados de movimiento e inactividad. En el fragmento de código [12](#) se puede ver dicha implementación.

Fragmento de código 12 Método `move()` de la clase `Player`

```
move(direction: Direction) {
    if (direction === DIRECTION.NONE) {
        this.isMoving = false;
        this.anims.stop();
        this.setVelocity(0);
    } else {
        switch (direction) {
            case DIRECTION.UP:
                this.setVelocity(0, -this.velocity);
                break;
            case DIRECTION.DOWN:
                this.setVelocity(0, this.velocity);
                break;
            case DIRECTION.LEFT:
                this.setVelocity(-this.velocity, 0);
                break;
            case DIRECTION.RIGHT:
                this.setVelocity(this.velocity, 0);
                break;
            default:
                this.setVelocity(0, 0);
        }

        this.isMoving = true;
        this._direction = direction;
        this.anims.play(`PLAYER_${this._direction}_ANIMATION`, true);
    }
}
```

4.1.7. Requested Items

Con el objetivo de guiar al jugador en cada misión, se diseñó el componente visual `Requested Items` que muestra, de manera clara, los objetos requeridos para completar la misión. Esta funcionalidad permite al jugador mantener en todo momento una referencia visual indicando qué ítems deben ser recolectados y cuántos faltan entregar.

Este componente se encarga de mostrar en la parte superior derecha de la pantalla, los assets correspondientes a los objetos solicitados por los NPCs, como se muestra en la figura 4.6. A medida que el jugador entrega correctamente cada objeto, uno de los íconos desaparece, permitiendo visualizar el progreso dentro de la misión.



Figura 4.6: Componente Requested Items

Este componente tiene un fundamento educativo particular. Al presentar visualmente los ítems requeridos en paralelo con sus nombres en inglés durante el diálogo con el NPC, se busca reforzar el vínculo entre imagen y palabra. Esta repetición visual y contextual favorece la asociación semántica y mejora la retención de vocabulario nuevo. Al mantener visible el objetivo de la misión en todo momento, se favorece la autonomía del jugador y se reduce la necesidad de instrucciones o recordatorios constantes. Además, es muy útil para nuestro contexto, donde muchos estudiantes están aún aprendiendo frases completas.

4.1.8. Health Bar

La barra de vida es un componente visual encargado de representar el estado de salud actual del jugador, como se muestra en la Figura 4.7. Su presencia constante en pantalla permite reflejar, en tiempo real, las consecuencias negativas de las acciones del jugador.



Figura 4.7: Componente Health bar

Cada vez que el jugador entrega un objeto incorrecto, se reduce el valor actual de la barra de vida. Esta disminución se representa visualmente mediante una animación que reduce la barra roja, permitiendo al jugador identificar fácilmente su estado de vida.

Cuando la cantidad de errores alcanza el límite definido en la configuración del sistema, la barra de vida se vacía por completo y el jugador pierde la partida, debiendo reiniciar el nivel. En ese momento, el sistema informa al jugador de su derrota y activa automáticamente la transición a la escena **GAME OVER**, que señala el final del nivel y ofrece la opción de volver a intentarlo.

4.1.9. Dialog

El sistema de diálogos permite manejar las interacciones textuales entre el jugador y los NPCs del juego. Se diseñó con un enfoque modular y extensible, facilitando la separación entre la lógica de presentación, la gestión del estado del diálogo y el contenido narrativo.

Para ello se implementó la clase abstracta `BaseDialog`, responsable de la visualización y actualización del contenido del diálogo, y una función auxiliar que permite cargar y transformar los datos crudos de los JSONs que conforman las misiones (ver sección 3.2.2) en estructuras listas para ser utilizadas por el sistema. `BaseDialog` provee una estructura común para la visualización y actualización del contenido, mediante la definición de métodos abstractos como `show()`, `hide()` y `setMessageComplete()`.

Estos métodos pueden ser implementados por las clases hijas, permitiendo flexibilidad según las necesidades específicas de cada escena o NPC.

En cuanto a la clase `Dialog`, que hereda de `BaseDialog`, gestiona la lógica específica de selección y presentación de diálogos. Los textos pueden ser generados previamente en el servidor mediante el modelo de lenguaje Gemini (véase [2.4](#)) o redactados manualmente. Para incorporar variabilidad en las respuestas de los NPC, se emplea el método auxiliar `selectRandomText()`, el cual selecciona aleatoriamente un enunciado de entre varios posibles para un mismo estado de la misión. De este modo, se logra mayor diversidad en los textos presentados, lo que fomenta la rejugabilidad.

Además, la clase es responsable de comunicar a los componentes relacionados el asset relacionado con la misión, es decir, el objeto que será recolectado.

En resumen, la clase es responsable por:

- Inicializar la interfaz gráfica del diálogo.
- Controlar la visibilidad y el estado del diálogo.
- Seleccionar mensajes adecuados según el progreso del jugador.
- Gestionar la interacción visual con el usuario.
- Informar a las clases interesadas el objeto solicitado.

Otra función relevante que tiene la clase es la de obtener y transformar los datos de diálogo definidos para una escena específica. Para esto, se accede al caché de Phaser con el fin de recuperar los datos asociados a la escena actual, generando así un arreglo de objetos `DialogData`. Esta estructura se puede ver en el fragmento [13](#).

Fragmento de código 13 Estructura lógica de un diálogo

```
export type DialogData = {
  id?: string;
  description?: string;
  questStart: string[] [];
  questInProgress: string[] [];
  questPartiallyCompleted: string[] [];
  questFinished: string[] [];
  questWrongItem: string[] [];
  hints?: string[] [];
  options?: string[];
  correctOption?: string;
  asset?: AssetConfig;
  quantityToCollect?: number;
  completed?: boolean;
  disable?: boolean;
}
```

Es importante aclarar que dicho fragmento representa el tipo genérico de los diálogos, en donde cada nivel tiene asociado un archivo con extensión `.json` que define el contenido de los diálogos que serán utilizados en la escena correspondiente. En el fragmento [1](#) se puede ver un ejemplo de una instancia específica de `DialogData`.

Se pueden ver también varios campos secundarios pero necesarios con el fin de gestionar correctamente los mismos, como por ejemplo `completed`, `disable`.

4.2. Diseño

El diseño visual del juego fue pensado con el objetivo de generar una experiencia clara, amigable y estéticamente atractiva para un público infantil. Se procuró mantener la coherencia estética a lo largo de las distintas pantallas, así como también la simplicidad en las interfaces para facilitar su comprensión y navegación.

Se optó por una estética de estilo pixel, usando gráficos con una resolución 32x32 píxeles. Esta elección fue útil tanto por motivos técnicos (bajo costo de renderizado y compatibilidad con dispositivos de bajo rendimiento como las laptops del Plan Ceibal) como por criterios visuales: los gráficos pixelados y de colores vibrantes suelen resultar especialmente atractivos para niños y niñas, además de evocar un estilo retro que genera cercanía y simplicidad.

Para lograr homogeneidad entre los distintos elementos gráficos, se estableció una paleta de colores unificada (ver figura 4.8), la cual fue definida mediante un proceso iterativo de ajustes visuales.

La elección se apoyó en la exploración de combinaciones cromáticas generadas con la herramienta digital Colors.co¹ y en el análisis de paletas utilizadas en otros videojuegos y dibujos animados con estética infantil como referencia inicial estética.

La paleta combina colores claros y oscuros para generar contraste, facilitando la legibilidad de elementos en pantalla. Además, se basa en una armonía análoga de tonos cálidos que transmiten sensaciones de cercanía y naturalidad, atributos especialmente relevantes para un público infantil.

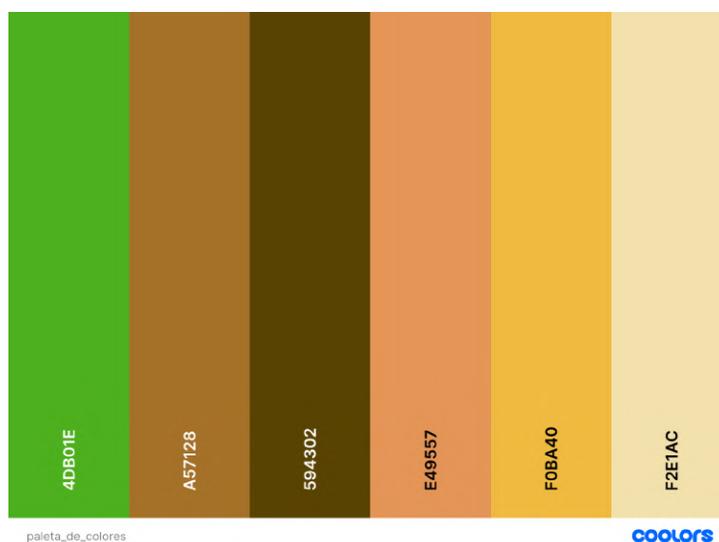


Figura 4.8: Paleta de colores utilizada.

Los recursos visuales empleados en el proyecto provienen de dos fuentes principales. En primer lugar, se recurrió a bancos de imágenes y animaciones con licencias libres de copyright, especialmente para personajes y objetos. En segundo lugar, se generaron recursos personalizados mediante inteligencia artificial, utilizando la herramienta de generación de imágenes integrada en ChatGPT (DALL·E 3²). A partir de descripciones textuales (*prompts*), se obtuvieron tiles, sprites y fondos adaptados a las necesidades del juego.

¹<https://coolors.co/>. Última vez accedido: 14/08/2025

²<https://openai.com/es-419/index/dall-e-3/>. Última vez accedida: 17/08/2025

En todos los casos, los *prompts* incluyeron lineamientos visuales consistentes, como la estética pixelada, el uso de gráficos de baja resolución (32×32 píxeles) y la aplicación de la paleta de colores definida previamente, con sus respectivos códigos hexadecimales para asegurar coherencia con el resto del sistema.

Si bien se intentó la creación manual de fondos y tiles, los resultados no alcanzaban la uniformidad visual deseada. En contraste, la generación con IA permitió explorar variaciones rápidas sobre una misma idea y obtener una base editable que se ajustaba mejor a los requerimientos estéticos y funcionales del proyecto. Esta decisión permitió garantizar originalidad y respeto por los derechos de autor en todos los recursos empleados.

Por ejemplo, para generar el asset del árbol otoñal, se empleó el siguiente prompt:

“A pixel art tree with a round green top and a brown trunk, in the style of retro 16-bit video games. Bright colors, flat background, simple and cartoonish design. Use this color palette: #F2E1AC, #F0BA40, #E49557, #594302, #A57128, #ADB01E.”

Si bien los resultados obtenidos mediante IA fueron útiles como punto de partida, fue necesario realizar ediciones manuales posteriores para lograr uniformidad estética. Por ejemplo, los tiles de pasto generados en diferentes imágenes presentaban variaciones en textura, patrón y color, por lo que se corrigieron manualmente para mantener una apariencia coherente en todos los escenarios.

Todos los tiles fueron generados y editados, los personajes fueron descargados de fuentes con licencia libre. Por otro lado, el 70 % de los items recolectables fueron descargados y el otro 30 % fueron generados. La figura 4.9 muestra algunos tiles generados mediante DALLE-E.

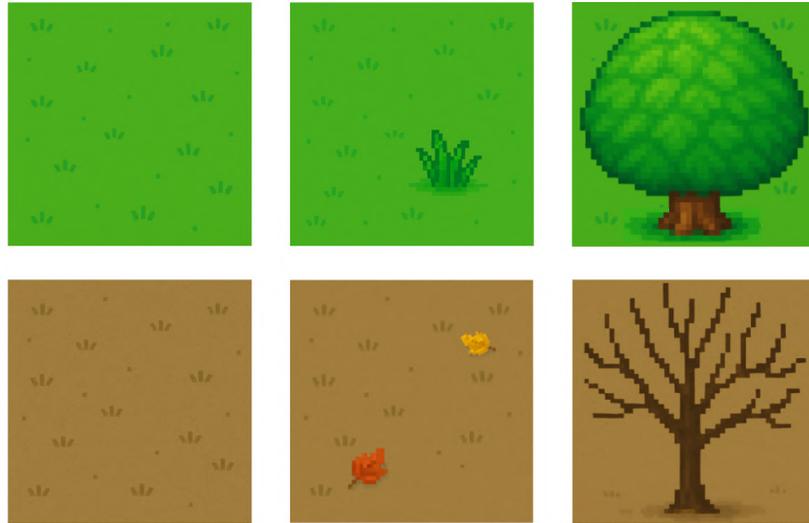


Figura 4.9: Tiles generados por DALLE-E

Por último, las pantallas fueron generadas con el objetivo de optimizar el tiempo de desarrollo y lograr un diseño más alineado con la temática del juego. La búsqueda de imágenes específicas con licencias adecuadas puede ser un proceso largo y tedioso, por lo que la generación de recursos visuales permitió mayor control y agilidad en el proceso creativo.

Para aportar dinamismo en aquellas pantallas que incluyen botones, se optó por utilizar múltiples imágenes en lugar de una imagen estática. Esto permitió construir botones interactivos a partir de dichas imágenes, facilitando su transformación y adaptación durante la ejecución del juego.

En resumen, se diseñaron pantallas con funcionalidades y estilos visuales diferenciados, manteniendo siempre una coherencia estética general. En la figura [4.10](#) muestra distintas pantallas representativas.



(a) Menú principal



(b) Selección de niveles



(c) Pantalla de nivel otoñal



(d) Pantalla de victoria



(e) Cartel de instrucciones

Figura 4.10: Distintas pantallas del juego desarrolladas para facilitar la navegación y reforzar la estética visual.

Cabe destacar que, en todos los casos, se priorizó la legibilidad del texto, el contraste adecuado entre fondo y elementos interactivos. En particular, los cuadros de diálogo se diseñaron con fondos oscuros, bordes bien definidos y tipografía en mayúsculas de alto contraste, lo cual favorece la lectura rápida. Esta elección busca mejorar la accesibilidad y asegurar una lectura fluida, especialmente en usuarios que aún están desarrollando habilidades de lectura.

4.3. Servidor

El servidor implementado tiene una única responsabilidad fundamental: interactuar con el modelo de lenguaje Gemini para la generación de parafraseos, tal como se describió en la sección 2.4. Para su desarrollo se adoptó el patrón de diseño Modelo–Vista–Controlador (MVC), ampliamente utilizado en el ámbito del software por su capacidad de organizar y estructurar aplicaciones de forma modular. Esta separación favorece la mantenibilidad, la reutilización de componentes y la escalabilidad del sistema (Gamma, Helm, Johnson, y Vlissides, 1995).

Este patrón se compone de tres elementos interrelacionados:

- **Modelo:** gestiona los datos y la lógica de negocio.
- **Vista:** presenta la información al usuario o cliente.
- **Controlador:** recibe las solicitudes, coordina la interacción entre el modelo y la vista y devuelve la respuesta correspondiente.

En esta implementación, el **Modelo** encapsula la lógica de interacción con el modelo de lenguaje Gemini, la **Vista** corresponde al objeto de respuesta enviado al cliente y el **Controlador** procesa el *request* que se recibe al solicitar un parafraseo. La adopción del patrón MVC ofrece una base sólida para futuras extensiones. Esta organización permite incorporar nuevas funcionalidades sin comprometer la claridad ni la mantenibilidad del código.

El servidor expone un único *endpoint*, que recibe el objeto mostrado en el fragmento 14. Dicho objeto contiene los distintos estados de una misión, dado que resulta de interés poder parafrasear cualquiera de ellos. Además, puede incluir un prompt personalizado en caso de que se desee guiar la generación o explorar diferentes posibilidades.

Fragmento de código 14 Modelo de Dialogo

```
class DialogContentInput(BaseModel):
    questStart: Optional[str] = None
    questInProgress: Optional[str] = None
    questPartiallyCompleted: Optional[str] = None
    questWrongItem: Optional[str] = None
    questFinished: Optional[str] = None
    questHints: Optional[str] = None
    prompt: Optional[str] = None
```

El controlador se encarga de aplicar el prompt a cada uno de los distintos estados y luego devuelve un objeto con una estructura similar a la de 14, pero con la diferencia de que las claves poseen el sufijo “IA” (por ejemplo, *questStartIA*).

De esta manera, es posible identificar cuáles fueron definidos manualmente y cuáles fueron generados por Gemini. En el fragmento 15 se muestra un extracto del controlador, donde se observa el *prompt* utilizado en caso de que no se proporcione uno nuevo al realizar la petición.

Fragmento de código 15 Extracto de GeminiController.py

```
async def get_paraphrase(dialogData: DialogContentInput):
    PROMPT_TEMPLATE = """
    For each of the sentences I will list below, separated by
    semicolon, give me a paraphrase of each one.
    The result must have the same format.
    Make sure to keep the same language as the original sentences.
    The sentences are
    """

    DIALOG_TYPES = [
        "questStart",
        "questInProgress",
        "questPartiallyCompleted",
        "questWrongItem",
        "questFinished",
        "questHints",
    ]

    generated_results = {}
    for dialog_type in DIALOG_TYPES:
        original_texts = getattr(dialogData, dialog_type, [])
        prompt_to_use = dialogData.prompt or PROMPT_TEMPLATE

        await result =
        ↪ GeminiService.process_paraphrase(original_texts,
        ↪ prompt_to_use)

        generated_key = f"{dialog_type}IA"
        generated_results[generated_key] = result

    return generated_results
```

El servicio `GeminiService` es el encargado de realizar efectivamente la llamada al LLM. Además, se ocupa de completar la construcción del prompt y de seleccionar el modelo que será utilizado. Parte de esta lógica puede observarse en el fragmento 16.

Fragmento de código 16 Extracto de GeminiService.py

```
class GeminiService:
    MODEL_NAME = "gemini-2.0-flash"

    @staticmethod
    def process_paraphrase(
        text_list: list[str],
        prompt: str
    ) -> list[str]:
        full_prompt = f"{prompt};{' '.join(text_list)}"
        model_response =
            ↪ GeminiService._get_model_paraphrase(full_prompt)
        return [x.strip() for x in model_response.split(";")]
```

En cuanto al prompt utilizado, fue el resultado de varias iteraciones sucesivas, en las que se fue refinando para incluir tanto el formato de salida esperado como la preservación del idioma original de las frases.

En las figuras 4.11 y 4.12 se muestra un ejemplo de una request al servidor y su respectiva respuesta. La interfaz mostrada en las figuras corresponde a SwaggerUI ³. Esta herramienta proporciona documentación interactiva para APIs y permite, además, realizar solicitudes directamente desde la propia interfaz. Esta herramienta se encuentra integrada de forma nativa en FastAPI, lo cual constituyó una de las razones en la elección de dicho framework.

³<https://swagger.io/tools/swagger-ui/>. Última vez accedido: 17/08/2025.

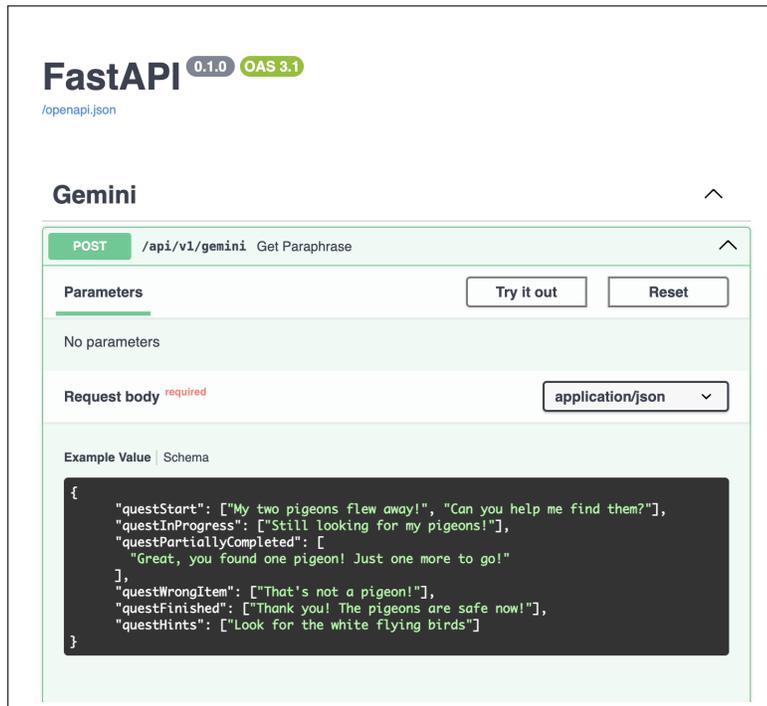


Figura 4.11: Ejemplo de un request al servidor

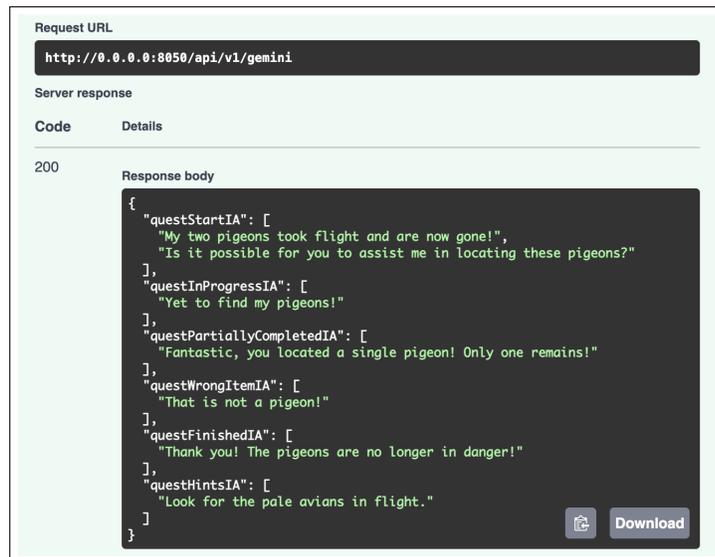


Figura 4.12: Respuesta al request de la figura 4.11

4.4. Pruebas conceptuales

Además de las implementaciones principales, se llevaron a cabo dos pruebas conceptuales para evaluar la viabilidad de futuras extensiones del proyecto. La primera de ellas consistió en adaptar el juego a dispositivos móviles, lo que implicaba la detección de gestos táctiles en la pantalla. La segunda exploró el uso de LLMs para la generación dinámica de los textos de las misiones.

4.4.1. Adaptación a dispositivos móviles

En la versión actual, los eventos táctiles no son reconocidos, lo que imposibilita jugar en pantallas táctiles. Sin embargo, la adaptación no representó un desafío significativo: fue suficiente con escuchar los eventos táctiles en lugar de los clics del ratón, ya que Phaser se encarga de gestionar automáticamente la resolución y el escalado de la interfaz.

La Figura 4.13 ilustra la prueba conceptual realizada. Para ello, se utilizó el simulador de dispositivos móviles de Chrome, emulando un Pixel 7 en orientación horizontal.

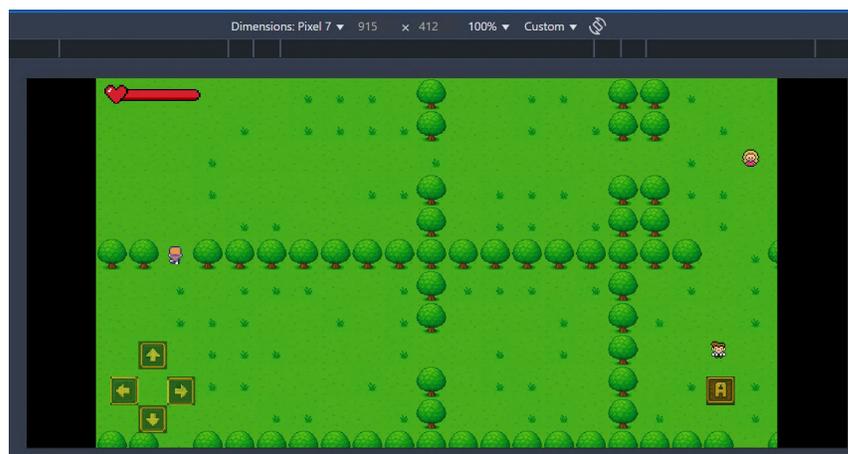


Figura 4.13: Prueba conceptual: Juego adaptado a dispositivos móviles

La única modificación necesaria fue la incorporación de botones virtuales para el movimiento y la acción, permitiendo así la interacción mediante toques. Dado que Phaser abstrae gran parte de la gestión de entrada y visualización, la migración completa a dispositivos móviles no supondría un riesgo técnico elevado. Cabe destacar que esta adaptación no se completó en su totalidad, ya que no formaba parte de los objetivos prioritarios del proyecto, por lo que se limitó únicamente a una validación inicial de factibilidad.

4.4.2. Generación de textos

Luego de haber utilizado Gemini para la generación de paráfrasis a partir de textos existentes, surgió la inquietud de explorar si sería posible generar nuevos textos y, a partir de ellos, obtener sus respectivas paráfrasis de manera automática. Es decir, sin proporcionar un texto inicial como referencia. De ser esto factible, se abriría un abanico de posibilidades en cuanto a la automatización de la creación de contenido educativo.

Con el objetivo de validar esta hipótesis, se realizó una prueba conceptual utilizando modelos de lenguaje disponibles en línea, específicamente ChatGPT⁴ y Claude⁵. La prueba consistió en solicitar a estos modelos la generación de frases relacionadas con el contexto educativo del proyecto, y posteriormente pedirles la creación de paráfrasis de dichas frases. El propósito fue evaluar la viabilidad de automatizar la generación de contenido educativo a partir de modelos de lenguaje, sin necesidad de partir de textos predefinidos. Para ello se definió el siguiente prompt y se ejecutó con el modelo o3 de ChatGPT

```
Estoy desarrollando un juego educativo para niños uruguayos de entre 9 y 11 años. El juego consiste en completar misiones en inglés relacionadas con frutas.
```

```
Quiero que generes misiones completas en un formato estructurado. Cada misión debe incluir exactamente una frase para cada una de las siguientes categorías:
```

```
questStart  
questInProgress  
questPartiallyCompleted  
questWrongItem  
questFinished  
questHints
```

```
Por cada frase de cada categoría, generará 4 paráfrasis diferentes que mantengan el mismo significado y nivel de comprensión, usando siempre un inglés claro, amigable y accesible para niños de 9 a 11 años.
```

```
Cada misión está asociada a un asset (tipo de fruta) y una cantidad (entre 1 y 7).  
Las frutas disponibles son:
```

```
apple  
bananas
```

⁴<https://chatgpt.com/>. Última vez accedido: 09/07/2025

⁵<https://claude.ai/>. Última vez accedido: 09/07/2025

cherries
kiwi
melon
orange
pineapple
strawberry

El resultado debe estar en formato de tabla, con las siguientes columnas:

Categoría
Texto
Asset
Cantidad

Asegurate de que cada grupo de frases (original + sus 3 paráfrasis) esté claramente agrupado como una única misión.

Ejemplo de frase original:

questStart: "Hi! I'm Sam. Today I need one apple.
Can you help me find it?"

Ejemplos de paráfrasis:

"Hello! My name's Sam. I really need an apple today.
Could you get it for me?"
"Hey friend! Sam here. Can you help me pick one apple today?"
"Good day! I'm Sam. I'd love it if you could find one apple for me."

Generaré 3 misiones completas, variando los assets y las cantidades en cada una. Mantené siempre frases simples, naturales y amigables, en inglés claro y adaptado a niños.

El resultado obtenido fue muy satisfactorio, ya que, sin muchas iteraciones sobre el prompt, se obtuvieron buenos resultados. En los anexos [C](#) y [D](#) se pueden encontrar los textos de la segunda misión generada y los generados por Claude.

El modelo eligió para la primera misión el asset `pineapple` (nunca antes utilizado en el juego) y una cantidad de 4 elementos. Para el estado inicial de la misión, `questStart`, se obtuvieron las siguientes frases:

- "Hey there! I'm Sara the chef. I need 4 pineapples for a big fruit salad. Can you help?"
- "Hello, friend! Sara here. Please gather 4 pineapples for my salad."

- “Hi! I’m Sara. Could you pick four pineapples so I can make a salad?”
- “Good day! Sara speaking. I’m counting on you to collect 4 pineapples.”

Como se observa, las cuatro frases son similares, producto de la paráfrasis. Sin embargo, presentan la desventaja de que el modelo eligió el nombre “Sara”. Esto introduce una limitación no deseada, ya que obliga a asignar un género específico al NPC que entrega la misión, cuando la intención del diseño es que estos personajes mantengan una identidad neutra en términos de género. Este inconveniente podría evitarse fácilmente indicando en el prompt que no se utilicen nombres propios o que se empleen nombres unisex.

Para el estado `questInProgress` se generaron las frases:

- “Great start! Keep finding the pineapples.”
- “Nice work! Stay on the lookout for pineapples.”
- “Good job! Keep searching until you have the pineapples.”
- “Super! Continue gathering pineapples.”

Al igual que antes, las frases son muy similares entre sí. Una mejora al prompt sería incluir el vocabulario que deberían manejar los alumnos en determinado año, con el fin de controlar la dificultad de las frases.

Para el caso de `questPartiallyCompleted`, las cuatro frases cumplen el propósito de la paráfrasis, sin embargo, todas indican que faltan dos elementos por recolectar, lo cual solamente es correcto cuando efectivamente restan dos. Dado que `questPartiallyCompleted` se muestra cada vez que el jugador entrega un ítem correcto (véase la sección 3.2.2), bastaría con ajustar el prompt para que no emplee números fijos y así mostrar un mensaje de ánimo independiente de la cantidad recolectada, por ejemplo: *“Good job! There’s not much left to finish!”*. Las frases generadas fueron:

- “Awesome! You grabbed 2 pineapples. Just 2 more to go!”
- “Nice! Two pineapples collected—two remain.”
- “Great job! Halfway there—two pineapples left.”
- “Well done! You have 2 pineapples; find the last 2.”

En cambio, los textos obtenidos para el caso de error, `questWrongItem`, cuando el jugador entrega un ítem incorrecto, fueron adecuados. No obstante, podría añadirse un texto explicativo que ayude al jugador a entender mejor su error; esta es otra posible mejora al prompt.

- “Oops! That’s not a pineapple. We need big spiky pineapples.”

- “Uh oh! Wrong fruit. Look for pineapples, please.”
- “Whoops! That isn’t a pineapple. Try again.”
- “Sorry, that fruit isn’t right. Bring pineapples.”

Finalmente, para el texto de cierre de la misión se obtuvieron:

- “Fantastic! You collected 4 pineapples. Salad time, thanks!”
- “Hooray! Four pineapples gathered. You’re incredible!”
- “Great job! All 4 pineapples are here. Thank you!”
- “Awesome! You brought every pineapple. Much appreciated!”

Las frases finales también fueron correctas y no presentan grandes diferencias entre sí.

Esta prueba de concepto demuestra que es viable implementar un juego de estilo roguelike, u otros géneros, con un mayor protagonismo de los modelos de lenguaje, ya que estos pueden generar todo el material educativo necesario. Sin embargo, dicho contenido debe ser revisado por docentes de inglés para asegurar que resulte adecuado y comprensible para niñas y niños.

4.5. Síntesis

En este capítulo se detallaron los principales aspectos técnicos de la implementación del sistema, abarcando tanto la aplicación web como el servidor. Se describió el uso del framework Phaser para la gestión de escenas y renderizado, así como la estructura modular basada en clases como `BaseLevelScene` y `LevelMaker`, que facilitan la organización y reutilización del código en la creación de niveles.

Se explicó en profundidad el algoritmo de generación dinámica de mapas, destacando la utilización del enfoque de *Space Partitioning* y la adaptación de este método para asegurar variedad, robustez y accesibilidad en los niveles generados. También se abordó la lógica de renderizado y el manejo de colisiones, junto con la integración de componentes clave como la barra de vida, el sistema de objetos solicitados y el sistema de diálogos.

Las pruebas conceptuales realizadas confirmaron la viabilidad de adaptar el juego a dispositivos móviles sin mayores desafíos y de aprovechar modelos de lenguaje para la generación dinámica de textos, aunque también requiere la necesidad de ajustes y supervisión pedagógica.

En síntesis, la implementación alcanzada sienta las bases para futuras ampliaciones del proyecto, permitiendo explorar nuevas funcionalidades y escenarios con un riesgo técnico reducido.

Capítulo 5

Validación y verificación

La validación y verificación son etapas del proceso de construcción del software que tratan de responder dos preguntas (Sommerville, 2011):

- *¿Estamos construyendo el producto correctamente?*
- *¿Estamos construyendo el producto correcto?*

En otras palabras, la validación trata de que el producto haya seguido buenas prácticas y se haya comprobado su correcto funcionamiento. En cambio, la verificación consiste en que el producto realmente satisfaga la necesidad detectada inicialmente.

5.1. Validación

Para garantizar que el producto desarrollado sea correcto, resulta fundamental revisar la metodología empleada, la gestión del repositorio realizada, el control de calidad aplicado durante el desarrollo y las pruebas realizadas.

5.1.1. Metodología aplicada

La metodología utilizada para el desarrollo del proyecto fue Scrum¹, un marco de trabajo ágil que promueve un proceso iterativo e incremental. Este enfoque fomenta la entrega frecuente de incrementos funcionales de software y la adaptación continua a los cambios, lo que resulta especialmente adecuado en contextos dinámicos o cuando no se cuenta con requisitos completamente definidos desde el inicio.

Debido a las particularidades del proyecto, fue necesario adaptar el marco tradicional de Scrum, manteniendo siempre sus principios ágiles fundamentales.

¹<https://www.scrum.org/resources/what-scrum-module>
Última vez accedido: 23/05/2025

Estas adaptaciones incluyeron una selección reducida y una frecuencia ajustada de las ceremonias: se mantuvieron únicamente la planificación (planning), la revisión (review) y la reunión diaria (daily), que se llevó a cabo entre dos y tres veces por semana. Los sprints se establecieron con una duración fija de dos semanas. Además, se coordinó una reunión quincenal con los tutores, con el objetivo de revisar el progreso alcanzado, anticipar posibles desvíos y repriorizar tareas para el siguiente ciclo de trabajo.

Para la planificación y seguimiento de tareas se utilizó un tablero de GitHub Projects², lo que permitió contar con una visión clara del estado del proyecto. Las tareas se etiquetaban para identificar rápidamente su naturaleza:

- backend
- frontend
- feature
- bug
- technical task
- informe

A su vez, las tareas eran priorizadas según el siguiente esquema:

- **P0:** Errores críticos que impedían el funcionamiento del sistema.
- **P1:** Errores que afectaban significativamente la experiencia del usuario.
- **P2:** Funcionalidades o errores de menor impacto, pero relevantes para la calidad del producto.

Esta clasificación permitió priorizar el trabajo sprint a sprint y identificar tempranamente cuáles eran los puntos a resolver en primer lugar.

5.1.2. Gestión del repositorio

Para la gestión del repositorio se siguió la convención de Conventional Commits³ con la finalidad de mantener una metodología estructurada. Esta convención establece reglas claras para escribir mensajes descriptivos en cada commit, facilitando así la generación automatizada de versionado semántico y una comprensión rápida de los cambios realizados. Un “commit” es un registro de cambios en el repositorio acompañado por un mensaje descriptivo.

²<https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects>

Última vez accedido: 23/05/2025

³<https://www.conventionalcommits.org/en/v1.0.0/>

Última vez accedido: 29/05/2025

Cada commit se estructuró con un prefijo que categorizaba el tipo de cambio efectuado:

- feat: para nuevas funcionalidades.
- fix: para corrección de errores.
- improvement: para mejoras en funcionalidades existentes.
- refactor: para cambios que no modifican funcionalidad, pero mejoran la estructura del código.

En la figura 5.1 se expone un ejemplo de varios commits que siguen esta nomenclatura.

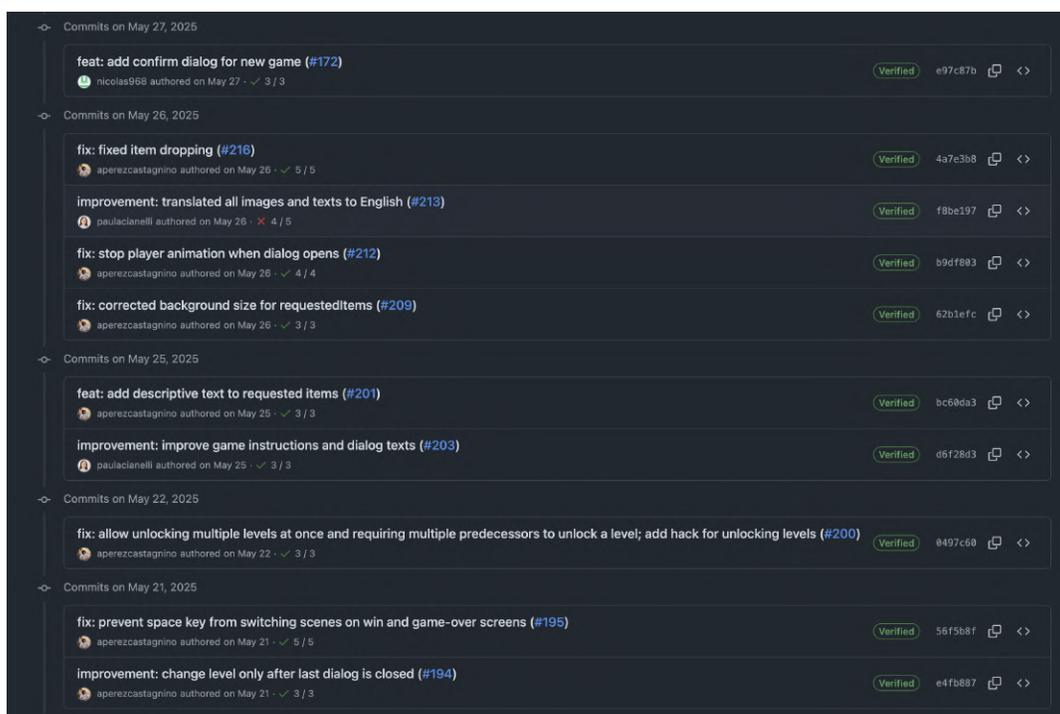


Figura 5.1: Conventional commits aplicado en la rama de *development*

5.1.3. Calidad del producto

La calidad del producto fue garantizada mediante el uso de diversas herramientas orientadas a asegurar buenas prácticas de programación, mantener un código limpio y organizar el manejo del repositorio.

Se utilizó Git⁴ como sistema de control de versiones y GitHub⁵ como plataforma de alojamiento del código, donde se definieron flujos de trabajo (workflows) para automatizar tanto el chequeo de la calidad del código como el despliegue de la aplicación.

Además, se promovió la revisión entre pares mediante la obligatoriedad de generar *pull requests* (*PR*) para todo cambio que se quisiera incorporar al proyecto. Un pull request es una solicitud para que el código implementado en una rama sea revisado por otros miembros del equipo y, en caso de ser aprobado, fusionado (merged) con otra rama, en nuestro caso, la rama principal (como main o develop). Esta práctica tiene como objetivo evitar la introducción de errores y facilitar la incorporación de mejoras.

Gracias a este mecanismo, fue posible no solo asegurar el cumplimiento de criterios funcionales y de estilo, sino también fomentar el intercambio de conocimiento técnico dentro del equipo, detectar errores de forma temprana y proponer mejoras estructurales al código.

En el proyecto de frontend, se hizo un uso intensivo de *linters*, herramientas que analizan el código fuente de forma estática con el objetivo de detectar errores comunes, inconsistencias y malas prácticas antes de la ejecución. En particular, se utilizó ESLint⁶, ampliamente adoptado en el ecosistema JavaScript/TypeScript. También se aprovechó el sistema de tipado estático proporcionado por TypeScript, lo cual permitió detectar errores antes de la ejecución, mejorar la documentación implícita del código y facilitar la navegación y comprensión de estructuras complejas. El tipado estricto contribuyó significativamente a la robustez y mantenibilidad del sistema.

Complementariamente, se utilizó Prettier⁷ para formatear automáticamente el código y mantener una estructura visual consistente en todos los archivos del proyecto, independientemente del editor utilizado por cada integrante del equipo. Esto redujo conflictos en los commits y facilitó la lectura del código.

⁴<https://git-scm.com/>. Última vez accedido: 29/05/2025

⁵<https://github.com/>. Última vez accedido: 29/05/2025

⁶<https://eslint.org/>. Última vez accedido: 29/05/2025

⁷<https://prettier.io/>. Última vez accedido: 29/05/2025

En el backend, basado en Python, se emplearon Ruff⁸ y Pylance⁹, que ayudaron a reforzar la calidad del código mediante chequeos sintácticos, de estilo y de tipos. Además, se utilizó la biblioteca Pydantic¹⁰, que permitió definir modelos de datos con validaciones automáticas y tipado fuerte. Pydantic facilitó la validación de estructuras de entrada y salida en las APIs, reduciendo errores y mejorando la confiabilidad del sistema. Su integración con el framework FastAPI también permitió una documentación automática y precisa de los métodos, reforzando la coherencia entre la lógica de negocio y los contratos de datos.

Gracias al uso de GitHub Actions (GA)¹¹, se automatizó la ejecución de estas herramientas en cada PR o push hacia cualquier rama, lo cual permitió detectar de forma temprana si el código no cumplía con las reglas definidas por Prettier o por los linters, previniendo así errores antes de ser integrados a la rama principal. Esta automatización se implementó mediante la creación de dos flujos en GA, los cuales se ejecutan automáticamente bajo ciertas condiciones predefinidas.

Para mantener las dependencias seguras y actualizadas, se integró Dependabot, una herramienta que analiza periódicamente los archivos de dependencias del proyecto y propone actualizaciones automáticas a través de la creación de PRs, en donde el usuario puede ver el contenido de la actualización y decidir si aplicarla o no. En la figura 5.2 se expone cómo es un PR creado por esta herramienta. Esta incorporación permitió prevenir vulnerabilidades de seguridad y mantener el entorno actualizado.

⁸<https://docs.astral.sh/ruff/>. Última vez accedido: 29/05/2025

⁹<https://marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance>
Última vez accedido: 29/05/2025

¹⁰<https://docs.pydantic.dev/latest/>. Última vez accedido: 29/05/2025

¹¹<https://github.com/features/actions/>. Última vez accedido: 29/05/2025

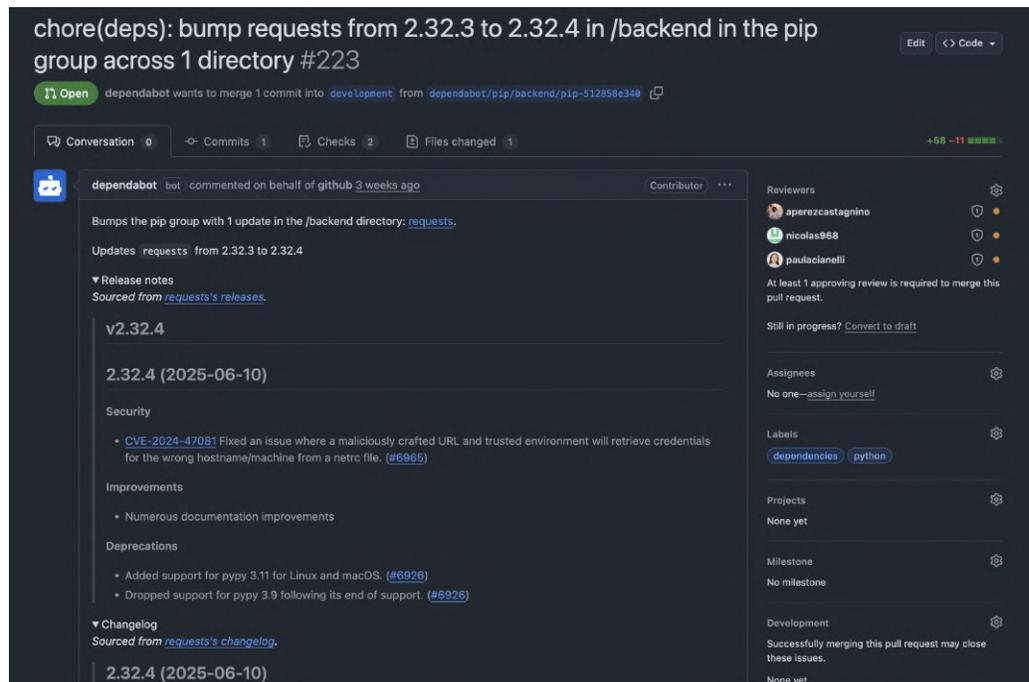


Figura 5.2: Pull Request creado por la herramienta Dependabot

En conjunto, todas estas prácticas y herramientas ayudaron a garantizar que el código desarrollado fuera no solo funcional, sino también legible, coherente y fácil de mantener en el tiempo.

5.1.4. Pruebas generales

Con el objetivo de identificar la mayor cantidad posible de errores de jugabilidad y problemas de comprensión del juego, se optó por realizar una prueba a través de una encuesta en línea. Dicha encuesta fue difundida entre personas del entorno cercano, familiares, amistades y compañeros de trabajo, por lo que no estuvo dirigida a un público objetivo. La muestra incluyó tanto a adolescentes como a adultos mayores. En total, se obtuvieron 34 respuestas, las cuales fueron analizadas con el fin de categorizarlas y priorizarlas.

Los errores reportados se clasificaron en tres categorías:

- **Errores graves:** errores que afectan directamente la jugabilidad o la progresión en los niveles.
- **Errores medianos:** errores que no impiden jugar, pero pueden confundir o entorpecer la experiencia.
- **Errores menores:** fallos de baja incidencia.

De las 34 respuestas obtenidas, se hallaron 19 errores, los cuales se clasificaron como: 5 graves, 9 medianos y 5 menores.

Los errores graves descubiertos fueron:

- Niveles que no se desbloquean o el progreso no se guarda correctamente.
- Pulsar la tecla **Espacio** reinicia inesperadamente el nivel bajo ciertas circunstancias.
- Bloqueo total al entregar un ítem antes de iniciar el diálogo.
- Objetos arrojados fuera de los límites del mapa quedan inaccesibles y bloquean la finalización de la partida.
- La habilidad de soltar objetos se encuentra imposibilitada en algunos casos.

Los errores medianos descubiertos fueron:

- Pantalla de selección de niveles no indica claramente el orden y número de los niveles.
- Textos mezclan inglés y español generando inconsistencias idiomáticas.
- Animales estáticos impiden el paso a través de pasillos estrechos.
- Tras abrir un diálogo, el personaje continúa desplazándose sin control.
- Ausencia de menú de pausa o botón para abandonar el nivel.
- Objetos y cuadros de diálogo se superponen dificultando la lectura.

- Barra de carga no representa el avance real.
- Pistas mostradas no aportan información nueva al jugador.
- Velocidad del jugador demasiada alta, complica el desplazamiento.

Los errores menores descubiertos fueron:

- Exceso de ítems sin propósito claro satura el mapa.
- Mensaje de victoria se cierra antes de que el jugador pueda leerlo.
- Texto del diálogo final es tapado por el aviso de “ganaste”.
- Algunos diálogos de misión se repiten innecesariamente.
- Frutilla y manzana muy similares visualmente confunden al jugador.

De todos estos errores reportados, muchos fueron solucionados a lo largo de varios sprints, con el objetivo de no frenar el avance del desarrollo ni la redacción del informe. Los cinco errores graves fueron resueltos, junto con la mayoría de los errores medianos y menores. Otros fueron descartados, ya que contaban con una justificación, como en el caso de “Frutilla y manzana muy similares visualmente confunden al jugador”, dado que se trataba de un efecto buscado.

Respecto a las sugerencias recibidas, se destacaron 5, de las cuales se implementó solamente la primera.

- Incluir un breve tutorial o guía de controles al inicio y accesible durante la partida.
- Ampliar el tamaño de los ítems y mejorar el contraste de colores para facilitar su identificación.
- Añadir indicadores visuales (flechas, mini-mapa o iconos) que orienten al jugador hacia su objetivo.
- Incorporar música de fondo y efectos de sonido que enriquezcan la experiencia.
- Permitir recoger y transportar varios objetos a la vez, o gestionar un pequeño inventario.

En el anexo [A](#) se encuentran más sugerencias textuales obtenidas de las pruebas generales. Además, se encuentran los resultados de todas las preguntas del cuestionario.

5.1.5. Pruebas en entornos reales

Además de las pruebas con el público general, se realizaron pruebas en las laptops provistas por el Plan Ceibal. Se utilizó una computadora facilitada por el Instituto de Computación (InCo) y otra en una escuela rural de Rivera, en una situación real de uso. En ambas instancias, se comprobó el correcto funcionamiento del juego, tanto en términos de rendimiento como de jugabilidad, validando así su compatibilidad con el hardware objetivo.

En la figura 5.3 se pueden ver dos fotos tomadas durante una clase. Esta instancia fue posible gracias a una docente que se ofreció voluntariamente a realizar la prueba con sus alumnos.

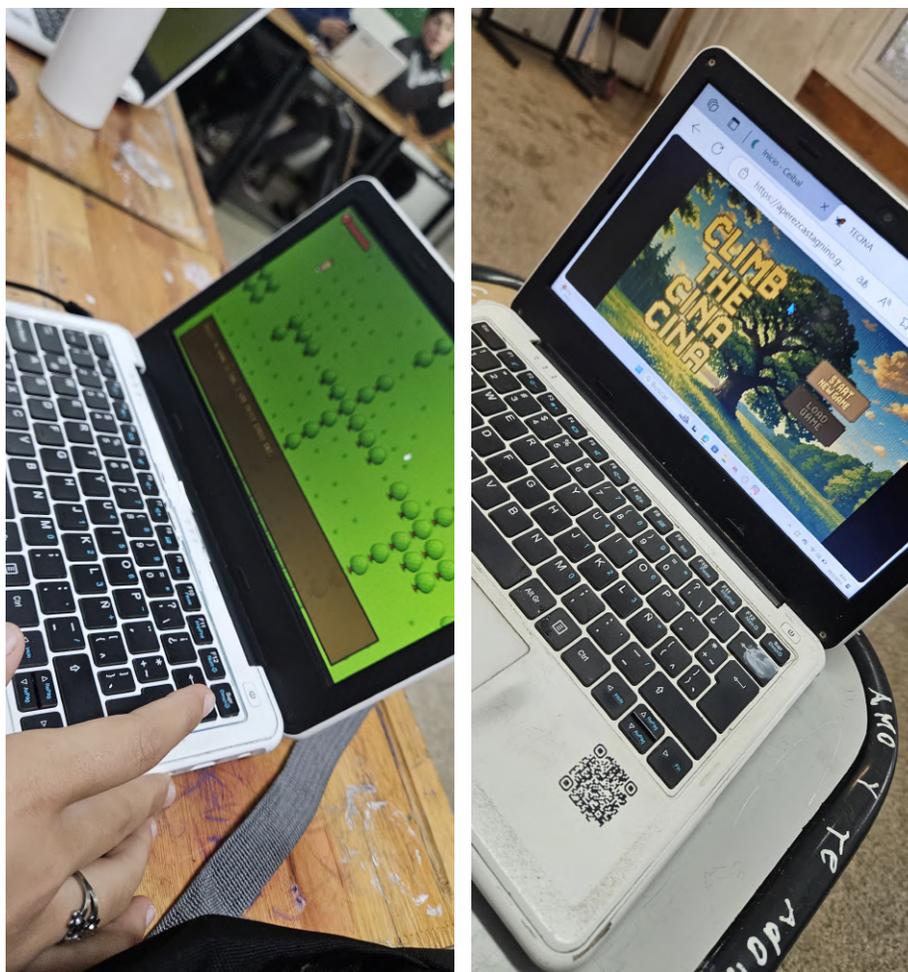


Figura 5.3: Prueba real en una escuela rural de Rivera.

5.2. Verificación

Para verificar que el producto construido cumple con las hipótesis planteadas inicialmente, se definieron tres objetivos:

1. Evaluar el producto con docentes de inglés, tanto en términos generales como desde una perspectiva pedagógica.
2. Determinar si el juego resulta útil para reforzar el aprendizaje del idioma.
3. Comprobar que el juego funciona correctamente en las laptops del Plan Ceibal.

Evaluación con docentes

Para alcanzar los primeros dos objetivos se diseñó un formulario con una serie de preguntas, y se buscó contactar docentes de diferentes regiones del país, no solamente de Montevideo, con el propósito de obtener una muestra representativa a nivel nacional.

El cuestionario, dirigido a docentes de inglés, incluyó preguntas orientadas a relevar información sobre el departamento, el tipo de institución, los años de experiencia del docente y las edades de los estudiantes. Asimismo, se indagó en diversos aspectos vinculados con la usabilidad y la pertinencia del juego, en particular la facilidad de uso, la adecuación del contenido lingüístico y del nivel de dificultad, su utilidad para reforzar vocabulario, así como el grado de motivación y el nivel de entretenimiento percibido. Finalmente, se incorporaron preguntas abiertas con el propósito de recoger sugerencias de mejora tanto desde una perspectiva pedagógica como desde la dimensión lúdica. Algunas de las preguntas formuladas, junto con sus resultados, se presentan en el anexo [B](#).

En total, se recopilaron respuestas de 25 docentes provenientes de distintos puntos del país, con experiencia en escuelas privadas, públicas y rurales. En conjunto, sus aportes representaron 53 experiencias de enseñanza desarrolladas en 53 instituciones. Tal como se muestra en la figura [5.4](#), estas instituciones se encuentran distribuidas en los departamentos de Montevideo, Salto, Rivera, Canelones, Colonia, San José, Paysandú, Florida y Maldonado. Obteniendo así información de 9 de los 19 departamentos del país, lo que garantiza una muestra geográficamente diversa.

Además, las instituciones se distribuyeron en 30 privadas, 18 públicas y 5 rurales, lo que también contribuye a dicha diversidad.

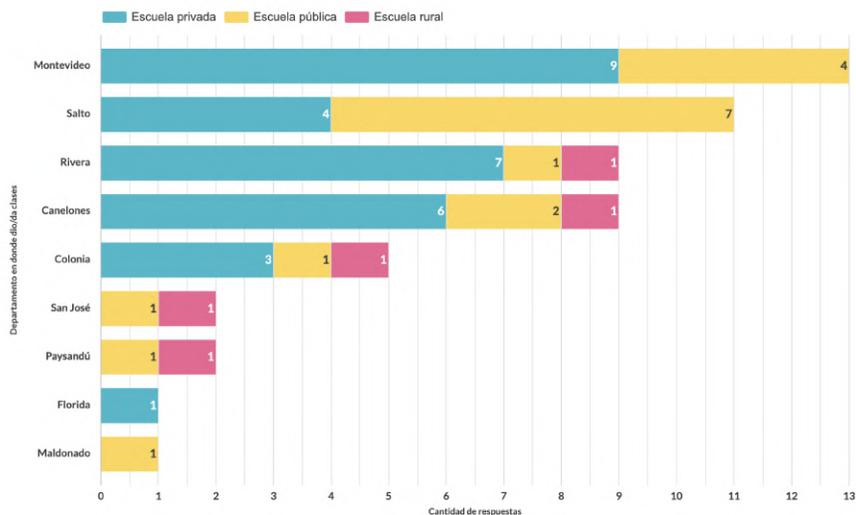


Figura 5.4: Instituciones privadas, públicas y rurales por departamento.

Respecto a la pregunta “¿Qué tan adecuado le pareció el contenido lingüístico del juego (vocabulario, frases, gramática)?”, cuyo propósito fue evaluar si el contenido resultaba apropiado para estudiantes de distintas edades, se obtuvieron los siguientes resultados: 21 docentes lo calificaron como “Muy adecuado”, 20 como “Adecuado” y 14 como “Poco adecuado”. La figura 5.5 muestra la distribución de estas respuestas en función de los distintos rangos etarios.

Las observaciones negativas se centraron principalmente en los siguientes aspectos:

- “Las frases y vocabulario no están acordes a un mismo nivel del público objetivo del juego. Sugiero enfocar el contenido hacia un área en particular o incluso presentar varias versiones del juego, por ejemplo: CLIMB THE CINA CINA Fruit Edition / CLIMB THE CINA CINA Animals Edition”.
- “Que el vocabulario utilizado sea el que se presenta en clases”.
- “Incluir instrucciones más claras respecto a encontrar al personaje para seguir la consigna”.
- “Que todo el contenido que se visualiza en el juego esté en inglés”.
- “Que exista la posibilidad de reproducir el texto de la misión en audio”.

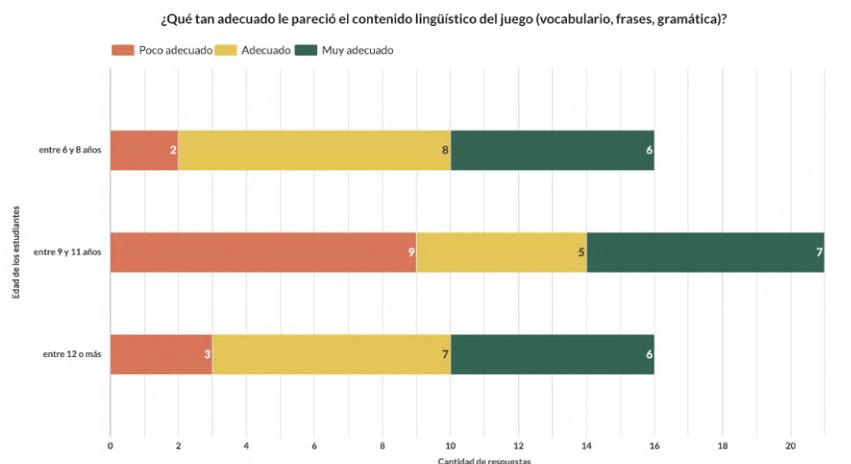


Figura 5.5: Respuestas a la pregunta “¿Qué tan adecuado le pareció el contenido lingüístico del juego (vocabulario, frases, gramática)?” distribuidas por edades.

En relación con la pregunta anterior, se planteó la consigna “¿Qué tan adecuada considera la dificultad del juego para el nivel de sus estudiantes?” con el objetivo de evaluar si el nivel de dificultad de las consignas planteadas era el apropiado. En caso contrario, una posible corrección podría consistir en utilizar un vocabulario más sencillo o frecuente, frases más cortas o con menor carga contextual.

De los resultados, visibles en la figura 5.6, se concluye que la dificultad fue considerada adecuada. De un total de 53 respuestas, solo una indicó lo contrario. Este resultado positivo puede atribuirse al hecho de que los textos fueron definidos o corregidos con el apoyo de docentes de inglés. En particular, los textos de los tres últimos niveles fueron directamente elaborados por docentes y luego parafraseados mediante el sistema, como se explicó previamente.

La única respuesta negativa se debió a que el docente consideró necesario que los textos fueran más largos, con una tipografía más grande y un vocabulario más variado, ya que los encontraba demasiado fáciles.

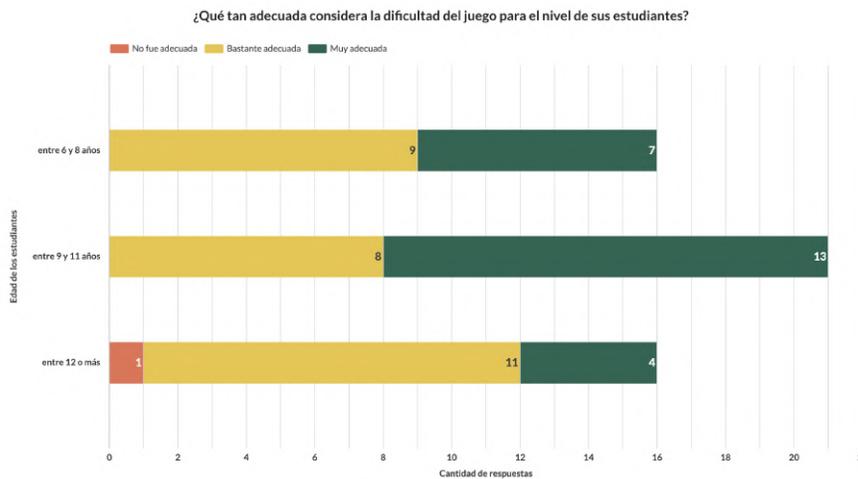


Figura 5.6: Respuestas a la pregunta “¿Qué tan adecuada considera la dificultad del juego para el nivel de sus estudiantes?” distribuidas por edades.

Una pregunta clave en relación con los objetivos planteados fue la de la motivación. Para que el juego contribuya al aprendizaje, debe ser intrínsecamente motivador; de lo contrario, los estudiantes se aburren y no desean jugar. Por lo tanto, se incluyó la pregunta “¿Crees que el juego motiva a los estudiantes a usar el inglés?”, la cual la gran mayoría de los docentes respondió afirmativamente. De las 53 respuestas, solo una indicó que no resultaba motivante, y, al igual que en la pregunta anterior, fue la misma persona. El motivo fue que, a su entender, el juego debería tener un mayor contenido lingüístico y un nivel de dificultad más alto para resultar verdaderamente motivador. En la figura 5.7 se presentan los resultados de esta pregunta.

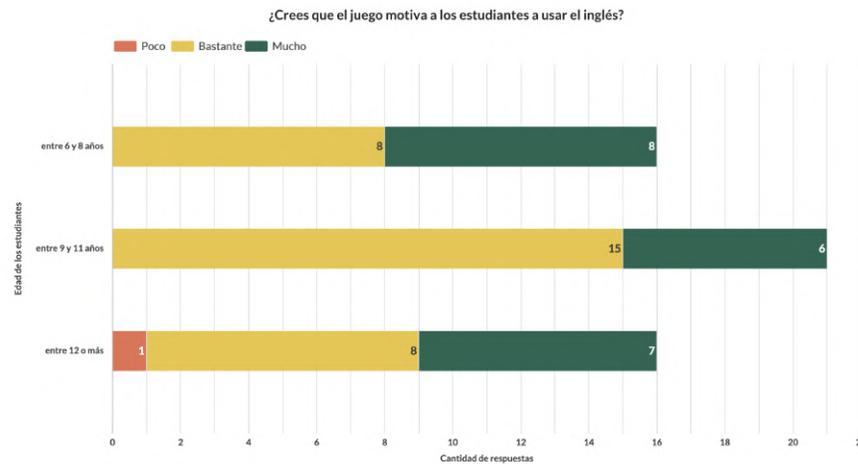


Figura 5.7: Respuestas a la pregunta “¿Crees que el juego motiva a los estudiantes a usar el inglés?” distribuidas por edades.

También se incluyó la pregunta “¿Qué tan útil consideras el juego para reforzar vocabulario en inglés?” con el objetivo de obtener una valoración directa sobre uno de los ejes centrales del trabajo. Como se muestra en la figura 5.8, la gran mayoría (un 92 %) consideró que el juego es efectivamente útil para reforzar el vocabulario. Un comentario destacado fue:

“Generalmente con los alumnos, lo que encontramos como material para jugar son solamente sopas de letras, crucigramas y matching games. Es difícil encontrar material realmente interactivo. Y peor aún con los alumnos de niveles más altos”.

Esto sugiere que el juego encuentra un lugar real dentro del público objetivo, aportando una herramienta valiosa para el reforzamiento del aprendizaje.

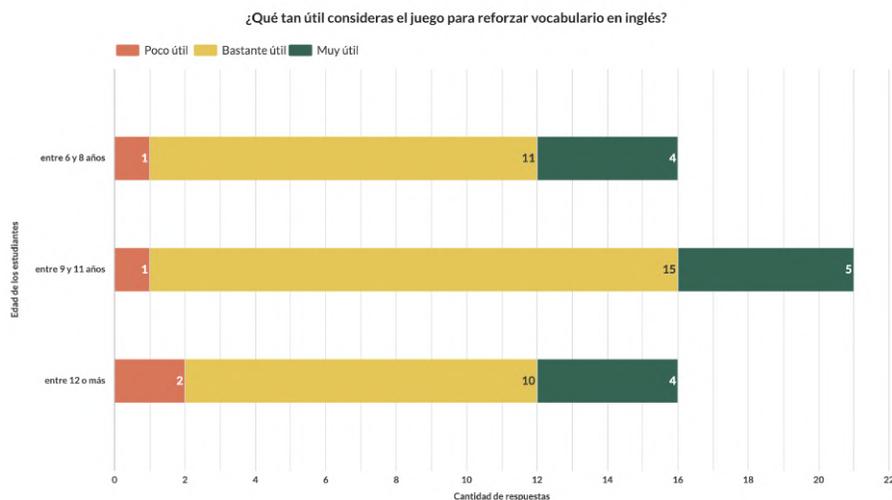


Figura 5.8: Respuestas a la pregunta ‘¿Qué tan útil consideras el juego para reforzar vocabulario en inglés?’ distribuidas por edades.

Finalmente, se realizaron dos preguntas a modo de cierre: “¿Qué tan útil consideras el juego para reforzar vocabulario en inglés?” y “¿Qué tan probable es que uses el juego en tus clases?”.

Los resultados, visibles en la figura 5.9, muestran que la mayoría de los docentes lo utilizaría en sus clases y lo recomendaría a otros colegas. En particular, se obtuvieron 47 respuestas afirmativas (88,6%) respecto a su uso en clase, y 50 respuestas afirmativas (94,3%) en cuanto a recomendarlo a otros docentes.

Estos resultados permiten concluir que el juego cumple con su objetivo de apoyar el aprendizaje del idioma, siendo percibido como útil, motivador y aplicable en contextos reales de enseñanza.

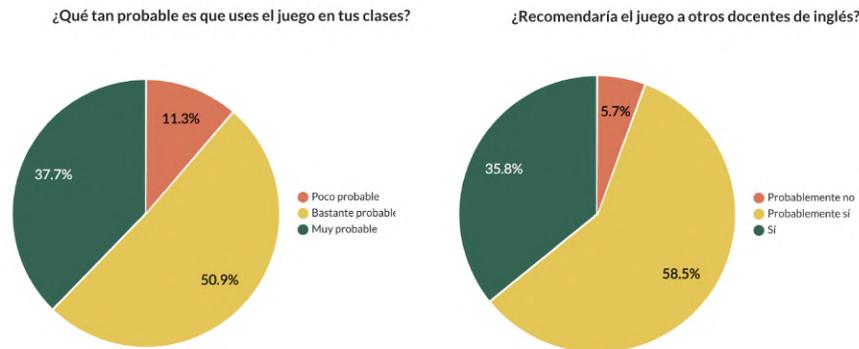


Figura 5.9: Respuestas a las preguntas “¿Qué tan útil consideras el juego para reforzar vocabulario en inglés?” y “¿Qué tan probable es que uses el juego en tus clases?” respectivamente.

Evaluación del rendimiento

Para la evaluación del juego se utilizó la medida de FPS (Frames Per Second o fotogramas por segundo), la cual representa la velocidad a la que un videojuego puede generar y mostrar imágenes consecutivas. Cuanto mayor es el número de FPS, más fluida e inmersiva resulta la experiencia del jugador¹². Cabe destacar que un fotograma es, en términos simples, una imagen fija dentro de una secuencia que, al reproducirse de forma continua y rápida, genera la ilusión de movimiento. A modo de ejemplo, si el juego corre a 60 FPS, significa que se renderizan 60 imágenes diferentes por segundo.

En general, un juego con 60 FPS se considera que posee un buen rendimiento; 120 FPS se consideran excelentes, mientras que 30 FPS reflejan un desempeño deficiente. No obstante, también influye la tasa de refresco de la pantalla; un monitor de 60 Hz solo podrá mostrar un máximo de 60 cuadros por segundo, mientras que uno de 120 Hz permitirá desplegar hasta 120 cuadros en el mismo intervalo.

En Phaser, el motor de desarrollo utilizado, existe una propiedad que indica en tiempo real los FPS, lo cual resulta útil para obtener esta medida. Además, los navegadores web también ofrecen mecanismos para monitorear los FPS.

En nuestro caso, utilizando una laptop del Plan Ceibal provista por el InCo (con 1.2 GB de memoria RAM, un procesador Celeron 1.60 GHz x4 y 24 GB de almacenamiento) se obtuvo un valor constante de 60 FPS. Este resultado representa una calificación positiva, ya que asegura una jugabilidad fluida. Cabe resaltar que las pantallas de las Ceibalitas son de 60 Hz, por lo que dicho rendimiento se encuentra alineado con la capacidad máxima de visualización del dispositivo.

¹²<https://www.lenovo.com/us/en/glossary/fps/> . Última vez accedido: 21/09/2025.

5.3. Síntesis

El proceso de validación y verificación permitió asegurar que el producto desarrollado no solo cumple con estándares de calidad técnica, sino que también responde a los objetivos pedagógicos planteados al inicio del proyecto.

Desde el punto de vista de la **validación**, se aplicaron diversas herramientas para garantizar buenas prácticas de desarrollo, calidad del código, mantenimiento del repositorio y control de versiones. El uso de linters, formateadores automáticos, tipado estricto y validaciones en los modelos de datos contribuyó a mantener un código limpio, coherente y fácil de extender. Además, la automatización mediante GitHub Actions y la integración de Dependabot reforzaron el monitoreo continuo de la calidad y la seguridad.

Las pruebas generales permitieron detectar una variedad de errores funcionales y de usabilidad. La clasificación sistemática de los errores permitió priorizar su corrección, logrando resolver todos los errores graves y la mayoría de los medianos y menores. Asimismo, se recibieron sugerencias valiosas de los usuarios, algunas de las cuales fueron implementadas y otras consideradas para futuras mejoras.

Desde el punto de vista de la **verificación**, se comprobó que el juego efectivamente puede cumplir con su propósito educativo. Las encuestas realizadas a docentes de inglés de distintas regiones del país reflejaron una evaluación ampliamente positiva en cuanto a la adecuación del contenido, la dificultad de las consignas, la motivación que genera en los estudiantes y su utilidad como recurso para reforzar el vocabulario en inglés. La diversidad geográfica e institucional de las respuestas obtenidas permitió asegurar que la solución es aplicable en distintos contextos educativos reales.

Por otro lado, la verificación del rendimiento del juego mediante los FPS corroboró la pertinencia de las tecnologías y los diseños aplicados, dado que se obtuvo un desempeño óptimo de 60 FPS en una laptop con pantalla de 60 Hz.

En síntesis, tanto la validación técnica como la verificación pedagógica demostraron que el producto desarrollado es de calidad, cumple con los requerimientos funcionales y tiene un potencial real de impacto en el aprendizaje del idioma inglés en el aula. Esto valida la hipótesis original del proyecto y justifica su aplicación como recurso didáctico.

Capítulo 6

Conclusiones generales y Trabajo Futuro

Este capítulo concluye el trabajo de la tesis de grado, presentando los objetivos cumplidos, los aprendizajes obtenidos y las futuras líneas de trabajo.

6.1. Conclusiones generales

Este proyecto representó un gran aprendizaje para el equipo. La creación de un videojuego orientado a colaborar en la educación de niños y niñas de escuelas rurales de todo el país, en un contexto con recursos limitados, permitió valorar el trabajo realizado desde una perspectiva diferente.

Desarrollar un videojuego educativo presentó diversos desafíos a los cuales nunca nos habíamos enfrentado hasta el momento. El hecho de que el proyecto se desarrollara en 2D, se basara en generación procedural y se integrara con un modelo de lenguaje, demostró ser tanto técnicamente viable como pedagógicamente valioso para reforzar el vocabulario de inglés en las laptops del Plan Ceibal. La combinación de Phaser y TypeScript brindó un buen equilibrio entre rendimiento y funcionalidad, lo que permitió que el juego se ejecutara con fluidez en equipos de recursos limitados. A su vez, la generación de mapas mediante partición espacial binaria garantizó la rejugabilidad y redujo los costos de desarrollo, al ofrecer una variedad de desafíos capaces de mantener la motivación del estudiantado.

La integración de Gemini para la paráfrasis de los textos de las misiones enriqueció la experiencia lingüística sin sobrecargar al equipo con redacción manual, aportando diversidad de expresiones y contexto dentro del juego. Además, se evidencia que los juegos educativos potenciados por modelos de lenguaje constituyen un campo lleno de posibilidades, dado que en este proyecto solo se exploró una pequeña parte de ellas.

Desde el punto de vista técnico, la arquitectura cliente-servidor desacoplada estableció bases sólidas para futuras integraciones y mejoras, protegiendo credenciales y garantizando modularidad. El uso de tipado estricto en Python y TypeScript, junto con linters, CI/CD y Dependabot, aseguró un código limpio, mantenible y seguro. La estructura de clases y la lógica de colisiones demostraron una base robusta y extensible para evoluciones futuras.

Desde el punto de vista pedagógico, las pruebas piloto realizadas con 25 docentes de inglés mostraron un 88,6 % de intención de uso en clase y un 94,3 % de recomendación a colegas, cifras que confirman que la propuesta responde a necesidades reales del aula y se percibe como un recurso innovador y motivador. Estas validaciones evidencian que el videojuego aporta un valor didáctico tangible, potenciando la enseñanza del idioma a través del juego.

Por último, la metodología ágil adoptada, una versión modificada de Scrum con sprints quincenales y ceremonias esenciales, resultó clave para entregar valor de forma iterativa y mantener la trazabilidad del proyecto.

En conjunto, los resultados obtenidos demuestran que el enfoque lúdico asistido por inteligencia artificial no solo es técnicamente factible, sino que también se alinea con el compromiso de Uruguay por continuar innovando en el ámbito educativo. Aprovechar las oportunidades que brindan las tecnologías emergentes para potenciar los procesos de enseñanza y aprendizaje fortalece el ecosistema educativo nacional y reafirma el rol del país como referente en la incorporación temprana y estratégica de herramientas digitales en las aulas. Contribuir al desarrollo de soluciones que integren pedagogía, tecnología y creatividad constituye, así, una vía prometedora para enriquecer la enseñanza de idiomas y abrir nuevas posibilidades para el aprendizaje.

Este trabajo abre la puerta a investigaciones futuras que profundicen en la analítica del aprendizaje, incorporen niveles temáticos adicionales e integren voz e imágenes, consolidando así el camino hacia experiencias educativas cada vez más innovadoras.

6.2. Trabajo futuro

A partir de las evaluaciones realizadas con los docentes, se identificaron varias líneas de mejora, tanto pedagógicas como de gamificación.

En primer lugar, se propone incorporar soporte de audio para las misiones. En muchas escuelas rurales conviven estudiantes que dominan la lectura con otros que todavía no. Al añadir locución, se facilita la asociación entre la palabra y el objeto, lo que posibilita que el juego pueda ser usado por todos los estudiantes, independientemente de su habilidad de lectura. Además, se cumple con la exigencia de la ANEP de adaptar el material didáctico a la realidad de cada estudiante.

Otra línea de trabajo sugerida por los participantes de las evaluaciones es la incorporación de preguntas múltiple opción como complemento a las misiones actuales. Esta funcionalidad permitiría reforzar el vocabulario aprendido, verificar la comprensión de forma inmediata y diversificar las interacciones, atendiendo a distintos estilos de aprendizaje.

También se propuso añadir ítems que se desplacen por el mapa, incrementando el nivel de dificultad y fomentando la exploración activa. Esta mecánica podría elevar la rejugabilidad, mantener la motivación del estudiantado y estimular la atención sostenida durante partidas más extensas.

Asimismo, varios docentes coincidieron en la conveniencia de desarrollar una narrativa general que sirva como hilo conductor de todos los niveles. Una historia unificada aportaría coherencia y continuidad, incrementando la inmersión y el compromiso del jugador, además de permitir integrar elementos culturales y lingüísticos propios del contexto uruguayo.

En cuanto a mejoras de alcance más amplio, se propone integrar el juego dentro de la plataforma Cinacina, con el fin de disponibilizarlo al público.

Otra mejora significativa sería el desarrollo de una interfaz gráfica que permita a usuarios no técnicos, como docentes, crear niveles o redefinir los textos del juego sin necesidad de conocimientos de programación. Esto abriría la posibilidad de adaptar el contenido a distintos contextos y niveles de dificultad, fomentando la personalización y la escalabilidad del recurso.

Finalmente, se plantea la implementación de las pruebas de concepto realizadas, entre ellas la versión móvil del juego, para ampliar su alcance y facilitar el acceso en diferentes dispositivos, y la generación automática de frases completas, más allá del parafraseo, potenciando así la riqueza y diversidad del contenido lingüístico presentado al jugador.

Referencias

- Adams, E. (2014). *Fundamentals of game design* (3rd ed. ed.). New Riders.
- Administración Nacional de Educación Pública (ANEP). (2019, November). *Inglés sin límites: Un proyecto democrático de enseñanza de inglés en educación primaria rural* (Inf. Téc.). Administración Nacional de Educación Pública (ANEP), Uruguay. Informe digital (PDF). Descargado de <https://www.anep.edu.uy/sites/default/files/images/Archivos/publicaciones-direcciones/Politicasylinguisticas/publicaciones/ingles-sin-limites/ISL.pdf>
- Administración Nacional de Educación Pública (ANEP). (2020). *Proyecto de presupuesto y plan de desarrollo educativo 2020-2024*. Informe digital (PDF). Descargado de https://siteal.iiop.unesco.org/sites/default/files/sit_accion_files/tomo_1_motivos_presupuesto_2020-2024_v12_web.pdf
- Agesic. (2021). *Agenda uruguay digital 2025*. Informe digital (PDF). Descargado de <https://www.gub.uy/agesic/comunicacion/publicaciones/agenda-uruguay-digital-2025>
- Centro Ceibal. (2023, June). *Plan estratégico*. Informe digital (PDF). Descargado de <https://documentos.ceibal.edu.uy/portal/2023/02/plan-estrategico-Ceibal-digital.pdf>
- Chiruzzo, L., Musto, L., Gongora, S., Carpenter, B., Filevich, J., y Rosa, A. (2022, December). Using NLP to support english teaching in rural schools. En *Proceedings of the second workshop on nlp for positive impact (nlp4pi)*. Descargado de <https://aclanthology.org/2022.nlp4pi-1.13/>
- Gamma, E., Helm, R., Johnson, R., y Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.
- García-Hernández, E., Sandoval-Escudero, A., Hernández-Rodríguez, J., Ayala-Carrillo, L., Camacho-Córdova, M. R., y Guerrero-Chávez, J. F. (2023). Gamificación en ambientes virtuales de aprendizaje: Experiencia didáctica con estudiantes de ingeniería. *Educación Química*, 34(1), 128-138. Descargado de https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1607-40412023000100128
- Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*. MIT Press. Descargado de <https://www.deeplearningbook.org/>
- Jurafsky, D., y Martin, J. H. (2023). *Speech and language processing* (3rd ed. (draft) ed.). Descargado de <https://web.stanford.edu/~jurafsky/>

[slp3/3.pdf](#) (Capítulo 3)

- Kasneci, E., Sessler, K., Bedenlier, S., Sesagiri Raamkumar, A., Niehaus, S., Schmidt, A., y Weitz, K. (2024). Large language models for education: A survey. *arXiv preprint arXiv:2403.18105*. Descargado de <https://arxiv.org/abs/2403.18105>
- Kleinberg, J., y Tardos, E. (2006). *Algorithm design*. Pearson.
- Martin, R. C. (2009). *Clean code*. Prentice Hall.
- Ministerio de Educación y Cultura. (2021, August). *Plan de política educativa nacional 2020–2025*. Informe digital (PDF). Descargado de <https://www.gub.uy/ministerio-educacion-cultura/sites/ministerio-educacion-cultura/files/documentos/noticias/Plan%20de%20Pol%C3%ADtica%20Educativa%20Nacional%202020–2025.pdf>
- Shaker, N., Liapis, A., Togelius, J., Lopes, R., y Bidarra, R. (2016). Constructive generation methods for dungeons and levels. En *Procedural content generation in games: A textbook and an overview of current research* (pp. 55–78). Descargado de <https://www.pcgbook.com/chapter03.pdf>
- Sommerville, I. (2011). *Ingeniería del software* (9th ed. ed.). Madrid, España: Pearson Educación.
- The Gemini Team. (2023). Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*. Descargado de <https://arxiv.org/abs/2312.11805>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. En *Advances in neural information processing systems* (Vol. 30). Descargado de <https://arxiv.org/abs/1706.03762>

Anexo A

Respuestas al formulario de validación del juego (general)

En este anexo se incluyen las respuestas al formulario utilizado en la validación del juego, vista en la sección [5.1](#).

Las preguntas de carácter cerradas se muestran en porcentajes, mientras que las preguntas de carácter abiertas se muestran transcritas, respetando su sentido original, con correcciones mínimas de ortografía y gramática para mayor claridad. Cabe agregar que se muestra una selección de las mismas para no extender demasiado el anexo.

A.1. ¿Entendiste rápidamente la dinámica del juego?

- Sí: 76.5 %
- Me costó algunos minutos: 17.6 %
- No entendí qué hacer: 5.9 %

A.2. ¿Tuviste algún problema jugando?

- Sí: 34.4 %
- No: 67.6 %

A.3. Explique qué problemas/errores encontró en el juego

- Me costó soltar a los animales cuando los agarraba sin querer; necesitaba mucho espacio para soltarlos.
- Se puede tirar un objeto fuera de los límites de la pantalla y no se puede recuperar.
- Los objetos se superponen con las cajas de texto.
- Tiré la manzana fuera del mapa y no pude continuar.
- Es raro que las frutas sean una limitante y bloqueen el paso.
- No se puede leer lo último que dicen porque enseguida aparece el mensaje de ganaste. En algunos niveles, los objetos que hay que buscar no aparecen en la barra de progreso.
- Después de entregar las palomas, si hablas nuevamente con la niña, ella te da la misión del científico como si fuera él.
- El mensaje al final del primer nivel desaparece muy rápido. A veces no se desbloquea el nivel 5 aunque completé el 4.
- Cuando estás caminando y hablas con un personaje mientras presionas las flechas, el personaje sigue caminando solo.

A.4. Sugerencias de mejora, teniendo en cuenta que jugarán niños

- Faltan instrucciones, me costó entender cómo empezar; hay muchas cosas que no sirven, como las frutillas y los conejos.
- Indicar con un caminito el orden de los niveles; agregar dinámicas tipo Kahoot con ejercicios sencillos en inglés.
- Letras más legibles, ritmo más lento, que los objetos no tapen los textos.
- Mostrar los controles dentro del juego; diálogos con “presiona espacio para continuar”.
- Agregar un camino para ver el orden de los niveles, contar una historia para dar un hilo conductor.
- Agregar un mapa en la pantalla para que los niños no se desorienten.
- Ilustraciones más claras para entender el objetivo y crear incentivos para avanzar.

A.5. Ideas para la próxima versión, nuevas dinámicas o niveles

- Niveles más complejos de noche con rango de visión reducido, desafíos adicionales.
- Agregar obstáculos como desafíos a superar.
- Misiones encadenadas que requieran hablar con varios NPC para conseguir lo que se busca.
- Agregar enemigos u otras dinámicas desafiantes.
- Un sistema de ayuda con pistas en inglés y español, perder puntos al usarlo.
- Que el juego cuente una historia o un cuento con hilo conductor.
- Que los niveles se conecten con una historia, agregar enemigos, permitir movimientos diagonales.

Anexo B

Respuestas al formulario de validación del juego (docentes)

En esta sección se presentan los resultados de la encuesta realizada en la etapa de verificación, en la sección 5.2. Dicha encuesta fue realizada a docentes de inglés, los cuales probaron el videojuego con sus estudiantes o de forma individual. Las preguntas de carácter cerrada se resumen en porcentajes, a diferencia de las respuestas de carácter abierta las cuales fueron transcritas. Al igual que el anexo anterior, solamente se muestra una porción de las mismas.

B.1. ¿Te resultó fácil o difícil entender cómo se juega?

- Muy fácil: 32 %
- Fácil: 64 %
- Difícil: 4 %

B.2. ¿Qué tan adecuada considera la dificultad del juego para el nivel de sus estudiantes?

- Muy adecuada: 36 %
- Bastante adecuada: 60 %
- No fue adecuada: 4 %

B.3. ¿Qué tan adecuado le pareció el contenido lingüístico del juego (vocabulario, frases, gramática)?

- Muy adecuado: 40 %
- Adecuado: 44 %
- Poco adecuado: 16 %

B.4. ¿Qué tan útil consideras el juego para reforzar vocabulario en inglés?

- Muy útil: 28 %
- Bastante útil: 64 %
- Poco útil: 8 %

B.5. ¿Crees que el juego motiva a los estudiantes a usar el inglés?

- Mucho: 44 %
- Bastante: 52 %
- Poco: 4 %

B.6. ¿Qué tan entretenido te pareció el juego?

- Muy entretenido: 32 %
- Bastante entretenido: 56 %
- Poco entretenido: 12 %

¿Qué tan probable es que uses el juego en tus clases?

- Muy probable: 40 %
- Bastante probable: 48 %
- Poco probable: 12 %

B.7. ¿Recomendaría el juego a otros docentes de inglés?

- Sí: 48 %
- Probablemente sí: 44 %
- Probablemente no: 8 %

B.8. ¿Qué cambiarías para mejorar el aprendizaje?

- Que el vocabulario utilizado sea el presentado en clases.
- Incorporar audio a las interacciones.
- Sugerencias en el primer nivel, no queda claro adonde hay que ir a buscar los elementos solicitados.
- Incluir instrucciones más claras respecto a encontrar el personaje para seguir la consigna. Solo palabras (números escritos).
- Incluir audio para acompañar la lectura de las instrucciones desplegadas en los cuadros de diálogo con el personaje principal.
- Para estudiantes más avanzados o a quienes les gusten los desafíos, consideraría incorporar algún nivel en el que deban reconocer el alimento o animal que deben encontrar a través de pistas como adjetivos, dónde está ubicado en el mapa, etc. Sin proporcionarles el nombre este directamente, promoviendo de esta forma que los/as estudiantes tengan una lectura más atenta e (idealmente) puedan conectar conceptos/vocabulario aprendidos en clase previamente y aplicarlo a la experiencia del juego.
- Pondría más texto/diálogo. Más grande la letra. Más chico el mapa. Más variado el vocabulario.

B.9. Sugerencias y comentarios de los docentes

A continuación se listan varias de las respuestas abiertas proporcionadas por los docentes encuestados, conservando su contenido original con leves ajustes para mejorar la legibilidad.

- Es atractiva la propuesta para que los alumnos refuercen el vocabulario aprendido en las clases y me parece fácil de incorporar en una planificación. Da pie para que surja otro tipo de contenido pedagógico (por ejemplo, replicar algunas de las preguntas del NPC en la clase, copiar en el cuaderno, practicar diálogos).
- Me gustó mucho y me pareció muy entretenido.
- Muy bueno para evaluar vocabulario y frases descriptivas sobre frutas. Estaría bueno que el alumno sumara puntos. Quizás por cantidad de objetivos logrados, tiempo de logro o por cantidad de vida restante. Si el alumno pudiera “rankear” su nombre en el juego al finalizarlo, esto podría actuar como un aspecto motivador (competencia) para finalizar el juego y comparar puntos obtenidos entre compañeros de clase.
- En definitiva, creo que el juego muchísimo potencial a ser explorado por los estudiantes y docentes, partiendo de la base que es un juego didáctico, un poco entretenido y te lleva conocer el inglés.
- Las frases y vocabulario no están acordes a un mismo nivel del público objetivo del juego. Sugiero enfocar el contenido hacia un área en particular, o incluso presentar varias versiones del juego, por ejemplo: CLIMB THE CINA CINA Fruit Edition/ CLIMB THE CINA CINA Animals Edition.
- Me gustó mucho la idea, considero que poseían explotar mucho más el aprendizaje de los niños.

Anexo C

Prueba de concepto: ChatGPT

Segunda misión generada en [4.1.2](#) en donde el objeto solicitado eran 5 bananas.

questStart

- “Hi! I’m Tom the monkey trainer. I need 5 bananas for snack time. Can you find them?”
- “Hello buddy! Tom here. Please gather 5 bananas for snack time.”
- “Hey friend! I’m Tom. Could you bring me five bananas to eat?”
- “Good day! Tom speaking. I’m counting on you to collect 5 bananas.”

questInProgress

- “You’re doing well! Keep looking for the bananas.”
- “Nice work! Keep your eyes open for those bananas.”
- “Great going! Keep hunting until you have the bananas.”
- “Super! Stay focused and find all the bananas.”

questPartiallyCompleted

- “Awesome! You have 3 bananas. Just 2 more left.”
- “Nice! Three bananas found two bananas remain.”
- “Good job! You got 3 bananas. Grab the last 2.”
- “Well done! Only two more bananas needed.”

questWrongItem

- “Oops! That isn’t a banana. Look for yellow bananas.”
- “Uh-oh! Wrong fruit. We need bananas, remember?”
- “Whoops! That’s not a banana. Let’s keep searching for bananas.”
- “Sorry, that fruit isn’t right. Please bring bananas.”

questFinished

- “Fantastic! You gathered 5 bananas. You’re banana tastic!”
- “Hooray! Five bananas collected—thank you so much!”
- “Great job! All 5 bananas are here. Snack time is saved!”
- “Awesome! You brought every banana I needed. Thanks!”

questHints

- “Bananas like to hide near the jungle bridge.”
- “Tip: Check by the wooden bridge; bananas often grow nearby.”
- “Hint: Look around the jungle bridge to spot bananas.”
- “Clue: Bananas can be found close to the tall vines by the bridge.”

Anexo D

Prueba de concepto: Claude

Otra variante de la misión generada en [4.1.2](#) En esta ocasión el objeto utilizado para la misión fue una manzana.

questStart

- “Hi! I’m Sam. Today I need one apple. Can you help me find it?”
- “Hello! My name’s Sam. I really need an apple today. Could you get it for me?”
- “Hey friend! Sam here. Can you help me pick one apple today?”
- “Good day! I’m Sam. I’d love it if you could find one apple for me.”

questInProgress

- “Great! You’re looking for my apple. Keep going!”
- “Awesome! You’re searching for the apple I need. Don’t give up!”
- “Perfect! You’re on the right track finding my apple.”
- “Nice work! You’re helping me look for that apple.”

questPartiallyCompleted

- “You found it! That’s exactly the apple I wanted.”
- “Amazing! You got the apple I needed. Perfect!”
- “Wonderful! That’s the right apple. You did it!”
- “Excellent! You found my apple. Thank you so much!”

questWrongItem

- “Oops! That’s not an apple. I need a red or green apple, please.”
- “Oh no! That’s not what I’m looking for. I need an apple.”
- “Not quite! I’m looking for an apple, not that fruit.”
- “Almost! But I need an apple. Can you try again?”

questFinished

- “Perfect! You found my apple. Mission complete! Thanks for helping me!”
- “Fantastic! You got my apple. The mission is done! You’re amazing!”
- “Excellent work! You found the apple I needed. Mission accomplished!”
- “Great job! You helped me get my apple. The mission is finished!”

questHints

- “Look for something red or green that grows on trees.”
- “Find a round fruit that can be red, green, or yellow.”
- “Search for a crunchy fruit that’s great for snacks.”
- “Look for a fruit that teachers love and keeps doctors away!”