



Instituto Politécnico Nacional

Escuela Superior de Cómputo

ESCOM

Trabajo Terminal

“Videojuego de Resolución de Desafíos a través de la Programación Visual para el Fomento del Desarrollo de la Lógica de Solución de Problemas (Scrap Coder)”

No. 2021-A026

Que para cumplir con la opción de titulación curricular en la carrera de **Ingeniería en Sistemas Computacionales**

Presentan

Alberto Ehad García Barradas

Joel Harim Hernández Javier

Director

Dr. Yaxkin Flores Mendoza

CDMX, México a 8 de junio de 2022

INSTITUTO POLITÉCNICO NACIONAL





Instituto Politécnico Nacional

Escuela Superior de Cómputo

Subdirección académica



No. de TT: 2021-A026

México a 8 de junio de 2022

Documento técnico

Trabajo Terminal

"Videojuego de Resolución de Desafíos a través de la Programación Visual para el Fomento del Desarrollo de la Lógica de Solución de Problemas (Scrap Coder)"

Presentan

Alberto Ehad García Barradas¹

Joel Harim Hernández Javier²

Director

Dr. Yaxkin Flores Mendoza

Resumen

En este reporte técnico se muestran los alcances y la implementación de este trabajo terminal. Este proyecto consiste en un juego donde la mecánica principal es la programación de un robot para la que se puedan resolver problemas mediante un lenguaje de programación visual, esto con el fin de fomentar el desarrollo de la lógica de solución de problemas.

Palabras clave: Videojuego, Programación Visual, Lógica de Solución de Problemas, Gamificación

1. albertogb46@gmail.com

2. joel.programador@gmail.com

CARTA RESPONSIVA

Que otorga visto bueno y avala la conclusión de documentación del Trabajo Terminal bajo los lineamientos establecidos por la Comisión Académica de Trabajos Terminales (CATT)

CDMX, a 13 de junio de 2022

**M EN C. ANDRÉS ORTIGOZA CAMPOS
PRESIDENTE DE LA COMISIÓN ACADÉMICA DE TRABAJOS TERMINALES
P R E S E N T E**

EN ATENCIÓN A:

M. EN E. ELIA TZINDEJHÉ RAMÍREZ MARTÍNEZ
SECRETARIA EJECUTIVA

Por medio de la presente, se informa que el Trabajo Terminal Núm. 2021-A026

Que lleva por Título: Videojuego de Resolución de Desafíos a través de la Programación Visual para el Fomento del Desarrollo de la Lógica de Solución de Problemas (Scrap Coder)

Fue concluido satisfactoriamente por:

García Barradas Alberto Ehad
Hernández Javier Joel Harim

Se avala que la documentación entregada mediante discos en formato DVD fue **revisada de manera precisa y exhaustiva** con el propósito de asegurar que los avances desarrollados bajo la supervisión de quien o quienes suscriben, hayan cumplido con lo planteado en el protocolo original, así como en lo establecido por el Documento Rector de Operación y Evaluación para los Trabajos Terminales de la ESCOM.

ATENTAMENTE
"LA TÉCNICA AL SERVICIO DE LA PATRIA"

(Nombre completo y Firma)
Director Dr. Yaxkin Flores Mendoza

Advertencia

"Este documento contiene información desarrollada por la Escuela Superior de Cómputo del Instituto Politécnico Nacional, a partir de datos y documentos con derecho de propiedad y por lo tanto, su uso quedará restringido a las aplicaciones que explícitamente se convengan."

La aplicación no convenida exime a la escuela de su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte técnico podrá obtenerse en la **Subdirección Académica de la Escuela Superior de Cómputo del Instituto Politécnico Nacional**, situada en Av. Juan de Dios Bátiz s/n Teléfono: **57296000**, extensión **52000**.

Agradecimientos

Agradecemos a nuestro director Dr. Yaxkin Flores Mendoza por orientarnos, ayudarnos y asesorarnos a lo largo del desarrollo de este proyecto.

A mi familia, mis amigos y mi pareja, que siempre me han apoyado y me ayudaron a llegar hasta este punto, pero sobre todo a mi director, el Dr. Yaxkin quien se acercó y me motivó cuando expresé mi interés en desarrollar un videojuego como proyecto para mi TT, y a mi compañero Joel Hernández; éste es el segundo proyecto en el que trabajo con él y ambos se convirtieron en mis dos proyectos favoritos de la carrera y en los que más me enorgullece haber trabajado, es un gran colega y amigo y no podría haber conseguido a alguien mejor para desarrollar este proyecto conmigo.

Alberto García.

A mi señor y salvador Cristo Jesús por haberme dado la sabiduría para completar este trabajo. A mi familia y amigos por su apoyo incondicional en estos tiempos difíciles. A todas las personas que en internet han compartido su conocimiento libremente.

Joel Hernández.

0

Índice



Índice general

Índice	6
Índice general	7
Introducción	14
Descripción del problema	15
Objetivo	16
Objetivo general	16
Objetivo particular	16
Productos o resultados esperados	17
Estado del arte	18
Trabajos terminales / tesis	19
Aplicaciones y servicios	19
Scratch	19
CodeMonkey	19
CodeCombat	20
Grasshopper	20
Tabla comparativa	21
Pixel Art	21
Hyper Light Drifter	22
Moonlighter	22
Terraria	23
Undertale	23
Celeste	24
Marco Teórico	25
Gamificación	26
Programación Visual	26
Aprendizaje basado en TIC	27

Desarrollo de la habilidad de resolución de problemas	28
Game Concept Document	30
Concepto	31
Tipo y plataforma	31
Público objetivo	31
Atmósfera	31
Contexto	31
Personajes	31
Ubicaciones	31
Jugabilidad	32
Mecánicas principales	32
Estructura del juego	32
Inspiración	32
Otros videojuegos / programas	32
Arte	32
Game Design Document	33
Narrativa	34
Contexto	34
Historia	35
Ubicaciones	37
Asentamiento	37
Casa de Boron, "El Loco"	37
Personajes	38
Bel (Protagonista)	38
Diseño	38
Bocetos y otros sprites	38
Boron, "El Loco"	39
Diseño	39

Bocetos y otros sprites	39
Copper	40
El pueblo	40
Ejército autómata	41
Robots	41
Bocetos y sprites	42
Marcus, El líder	42
Items	44
Puerta	44
Palanca	45
Botón	45
Caja	46
Textura de ladrillos	47
Textura de adoquines	47
Textura de suelo de mosaicos	47
Textura de suelo simple	48
Barandal conectable	48
Máquina grande	49
Bloques de evento	50
Instrucción genérica	50
Bloque de repetición	51
Bloque de condición	52
Condicionales	53
Valores numéricos y arreglos	54
Pantallas del menú principal	55
Guión	57
SCRIPT	59
Storyboard	69

Mecánicas del juego	71
Flujo de acciones dentro del juego	71
Descripción de las mecánicas del juego.	72
Caminar	72
Mover objetos	72
Tomar objetos	72
Guardar objetos	72
Programar el robot	72
Botón	73
Palanca	73
Puerta	73
Technical Design Document	74
Herramientas a utilizar	75
Motor de videojuego: Unity	75
Aseprite	76
Visual Studio	76
Visual Studio Code	77
Paint .NET	78
Plataforma de desarrollo	79
Metodología	80
Cronograma de actividades	81
Arte y gráficos	82
Resolución y relación de aspecto	82
Configuración de los sprites	83
Configuración del grid	83
Configuración de la cámara	83
Interfaz y progreso del juego	84
Interfaz	84

Carga inicial	84
Pantalla principal	85
Pantalla de selección de niveles	85
Cuadro de confirmación genérico	87
Cuadro de información genérico	87
Interfaz dentro del juego	88
Cinemática	90
Flujo de pantallas	91
Progreso del juego	92
Físicas, colisiones e interacción	93
Físicas	93
Interacción	93
Eventos de Unity relevantes	93
Awake	93
Start	93
Update	94
FixedUpdate	94
OnTriggerEnter2D	94
OnTriggerExit2D	94
OnPointerDown	94
OnBeginDrag	94
OnDrag	94
OnEndDrag	94
Plataforma objetivo	95
Implementación	96
Programación visual e intérprete	97
Jerarquía y ordenamiento	98
Nodos	100

Gráficos	100
Conexión entre nodos	101
Intérprete	103
Analizador	103
Ejecutor	103
Variables y arreglos	106
IntepreterElement	108
Mecánicas y Jugabilidad	110
Acciones del jugador	110
Movimiento	110
Inspección	111
Reiniciar nivel	113
Comportamiento del robot	113
RobotController	113
Sistema de energía	115
Cables	115
Puertas	116
Botones	117
Palancas	117
Paneles	118
Mensajes	119
Tutoriales	120
Niveles	122
Arquitectura de niveles	122
Diseño de niveles	126
Nivel 1-1, Primeros pasos.	126
Nivel 1-2, Zigzag.	127
Nivel 1-3, Marcha sin fin.	128

Nivel X-1, Espiral.	130
Nivel X-2, Ordenamiento.	132
Resultados	134
Pruebas con el lenguaje visual	135
Serie de Fibonacci	135
Ordenamiento por burbuja	137
Conclusiones y Trabajo a futuro	140
Conclusiones generales	141
Conclusiones personales	142
Alberto Ehad García Barradas	142
Joel Harim Hernández Javier	143
Trabajo a futuro	144
Mejor implementación de UI	144
Entornos más ricos y variados, mecánicas más complejas	144
Visor mejorado	144
Mecánicas de Bel más pulidas	144
Diseños de niveles más completos.	144
Referencias	145
Referencias generales	146
Anexos	151

1

Introducción

En este capítulo se describen los antecedentes y se delimita la problemática, así como los objetivos y los productos que se esperan para este proyecto.



Descripción del problema

Para el siglo XXI, la UNESCO [1] considera que la resolución de problemas y la creatividad son habilidades básicas que tienen que ser desarrolladas como formación fundamental en la educación humana, y que según Henriquez y Sotomayor [2] son un elemento que emergió como un eje prioritario en el Estudio Regional Comparativo y Explicativo.

Ambas habilidades, dicen Rogalski y Samurçay [3], son pilares fundamentales que se utilizan para diversos ámbitos de la vida cotidiana, ya sea para el seguimiento de instrucciones, reconocimiento de patrones o para saber cómo actuar ante eventualidades inesperadas; y también lo son para la adquisición de conocimiento de programación que, junto con las herramientas cognitivas, la experiencia, la práctica y el conocimiento estructurado, forman el dominio de requerimientos necesarios para poder analizar problemas y solucionarlos.

La habilidad de comprensión y resolución de problemas, según la OCDE [4], se considera dentro de lo que es el Nivel 2 en la prueba Programme for International Student Assessment (PISA por sus siglas en inglés). Dicho nivel es considerado como "... el nivel básico de conocimiento que se requiere para participar plenamente en una sociedad moderna." [4].

En México, la prueba PISA 2018 [4] reveló que los estudiantes mexicanos, en promedio, obtuvieron un puntaje bajo en las tres áreas evaluadas: lectura, matemáticas y ciencia, comparando el nivel con el promedio mundial de la OCDE, y sólo el 1% obtuvo un desempeño en los niveles de competencias más altos, a comparación del 10% a nivel global.

La OCDE define a los estudiantes que obtuvieron el Nivel 1 en la prueba PISA como de bajo rendimiento, es decir que "pueden responder y seguir cuestiones e instrucciones sencillas, pero no pueden enfrentarse a la resolución de problemas que requieran razonamientos complejos" [4]. Así mismo la prueba PISA 2018 [4] arrojó que el 35% de los estudiantes no lograron alcanzar el nivel mínimo de competencia, es decir, están en el Nivel 1, a comparación del 13% a nivel mundial.

Los datos anteriores dejan de manifiesto que es de relevancia nacional el atender y promover el desarrollo de estas habilidades anteriormente mencionadas en la población mexicana, particularmente el análisis y resolución de problemas.

Este proyecto busca crear un videojuego, llamado **Scrap Coder**, que fomente en los jugadores el desarrollo de la lógica de resolución de problemas mediante acertijos que se encontrarán a lo largo del juego a través de la programación visual.

La jugabilidad contará con dos elementos principales, el protagonista controlado directamente por el jugador y un robot programable, el cual podrá ser programado

mediante el uso de módulos de programación para definir su comportamiento; para así superar obstáculos encontrados en cada nivel del juego, usando tanto las acciones programadas en el robot como las acciones realizadas por el protagonista.

Este videojuego utilizará los supuestos teóricos de la gamificación y del aprendizaje por TICs para poder lograr su objetivo; y la programación visual, ya que es un medio propicio para introducir la programación a personas sin experiencia a través de los videojuegos, según Sáez y Cázar [5].

Objetivo

Objetivo general

Desarrollar un videojuego 2D para la plataforma de escritorio Windows de resolución de desafíos o puzzles usando la programación visual como mecánica principal para fomentar el desarrollo de la lógica de solución de problemas en los jugadores.

Objetivo particular

1. Diseñar e implementar el intérprete para la parte de programación visual.
2. Realizar el arte de los personajes, objetos, escenarios e interfaz del juego.
3. Diseñar e implementar dos niveles junto con la lógica que involucra para resolverlos.

Productos o resultados esperados

A continuación, en la Figura 1, se muestra la arquitectura del videojuego propuesto. El producto obtenido será una aplicación de escritorio para sistemas Windows con extensión .exe, la cuál será el videojuego en sí.

Los productos que se esperan desarrollar son:

- El videojuego como programa ejecutable.
- La documentación técnica.
- El manual de usuario.

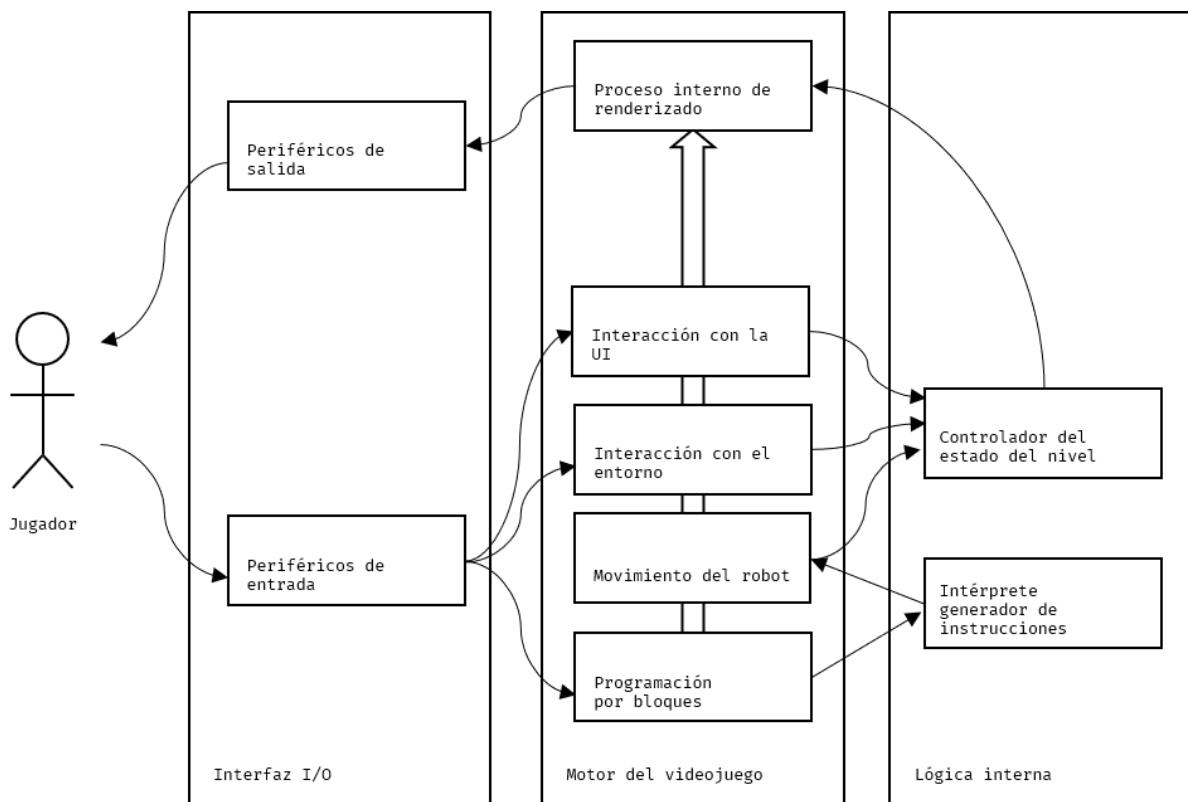


Figura 1.1. Arquitectura general del videojuego.

El videojuego tendrá las siguientes características:

1. Dos niveles que tendrán varios desafíos definidos a través de puzzles que forman un reto más grande.
2. Intérprete que generará las instrucciones para manejar al robot.

2

Estado del arte

En este capítulo se describirán brevemente trabajos terminales y tesis cuyo objetivo es similar al nuestro, así como otros proyectos que comparten un propósito parecido.



Trabajos terminales / tesis

El proyecto con número de registro **TT 2014-A070** llamado "MECANISMO PROGRAMABLE PARA NIÑOS: El desarrollo de un lenguaje de programación mediante una interfaz de arrastrar y soltar para la programación de mecanismos de hardware.", realizado en la Escuela Superior de Cómputo (ESCOM), utiliza artefactos y componentes de hardware controlados mediante una interfaz de programación visual [6].

Tiene el objetivo de enseñar conceptos lógicos y la creación de programas simples mediante íconos gráficos, esto integrando la manipulación de sensores y motores que forman parte de un circuito diseñado por el propio usuario.

La tesina 2010 **C7.1447** realizado en la Unidad Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas (UPIICSA) llamado "Desarrollo de un videojuego didáctico para la enseñanza de conjuntos y sus operaciones con JAVA 2D", la cual se centra en la enseñanza de conceptos matemáticos utilizando como medio un videojuego, profundizando en los diversos tipos de juegos que existen y la posibilidad de su uso didáctico [7].

Aplicaciones y servicios

Scratch

Scratch se define como una plataforma para programar historias interactivas, juegos y animaciones, los cuales pueden ser compartidos en la propia comunidad online. Marji [8] indica que fue desarrollado por el grupo Lifelong Kindergarten del MIT Media Lab en 2003.

Según las propias palabras de la Scratch Foundation, fundación sin ánimo de lucro que continúa desarrollando y moderando este software, Scratch ayuda a las personas a "aprender creativamente, razonar sistemáticamente y trabajar colaborativamente" [9].

Scratch permite a los usuarios insertar y manipular imágenes, música, sonido, animaciones, y dar a todo objeto dentro de cada escena de un comportamiento o interactividad. La forma en la que el usuario puede programar este comportamiento de los objetos es mediante un lenguaje de programación visual, es decir, permite arrastrar y soltar bloques y elementos que permiten la construcción de un algoritmo [9].

CodeMonkey

CodeMonkey es un entorno educativo basado en juegos que tiene el propósito de enseñar a los niños a codificar sin ninguna experiencia previa. Consiste en un simulador que permite programar ingresando código escrito, ya sea en CoffeScript o en Python, a un mono dentro de un escenario lleno de obstáculos [10].

Los diversos niveles dentro de CodeMonkey permite el uso de ciertas instrucciones para que el estudiante aprenda ciertos conceptos sobre la programación; además el propio entorno permite calificaciones y seguimiento automático y algunos controles para profesores que hagan uso de este software [10].

CodeCombat

Es una plataforma de clases en línea sobre programación que utilizan como medio un simulador en forma de juego e instrucción personalizada con profesores [11].

Tiene dos juegos disponibles, el más reciente es llamado Ozaria, que es un juego de aventura y es utilizado, acompañado de clases en línea, para enseñar Ciencias de la Computación; el otro juego, llamado CodeCombat, permite enseñar competencias conforme el estudiante va jugando [11].

Contiene un intérprete interno que permite escribir código en Python o en Javascript para que el estudiante pueda resolver retos que se encuentran en cada nivel con el que se pretende enseñar programación y promover el desarrollo de habilidades necesarias para codificar [11].

Grasshopper

Es una aplicación web y para móviles desarrollada por Google por su departamento de productos experimentales , Area 120, con el objetivo de que las personas puedan aprender a programar utilizando muy poco tiempo a lo largo del día [12].

Permite simular el movimiento de un saltamontes para realizar ciertas tareas u objetivos programando utilizando Javascript. Su interfaz para crear el código de programación presenta un código incompleto donde el usuario tiene que escribir las variables o sentencias necesarias para que el reto sea completado [12].

Tabla comparativa

A continuación, presentamos una tabla comparativa que resume las principales características que tienen las anteriores aplicaciones y servicios anteriormente mencionados.

Software	Programación Visual	Videojuego	Gamificación	Aprendizaje basado en retos
Scratch	✓		✓	
CodeMonkey		✓	✓	✓
CodeCombat		✓	✓	✓
Grasshopper		✓	✓	✓
Scrap Coder	✓	✓	✓	✓

Tabla 2.1. Comparación de características.

Pixel Art

Los píxeles (palabra que viene del acrónimo inglés *picture element, pixel*) se definen, según la Real Academia Española, como la menor unidad homogénea que forma parte de una imagen digital [13].

Se denomina Pixel Art (Arte de píxeles) al estilo o diseño artístico digital basado en píxeles, en donde es muy importante el uso del color para una correcta representación de las imágenes, de acuerdo con Juan Carlos [14]. En este tipo de arte los píxeles funcionan como bloques de construcción para la imagen generando un estilo visual muy parecido al arte de mosaicos, la técnica de bordado puntos de cruz y otros tipos de bordado [15].

De acuerdo con MuBert [16] el Pixel Art es una expresión artística que se asocia a los videojuegos, siendo su origen en este campo las limitaciones visuales que tenían los creativos y artistas durante el comienzo de este género de entretenimiento.

Al ser un estilo de arte presentes desde el inicio de los videojuegos, la lista de juegos que han utilizado el Pixel Art como su diseño artístico principal es muy extensa, pero aquí se presentan algunos juegos actuales principales que utilizan esta técnica para representar sus gráficos, algunos seleccionados por Ahmed [17]:

Hyper Light Drifter

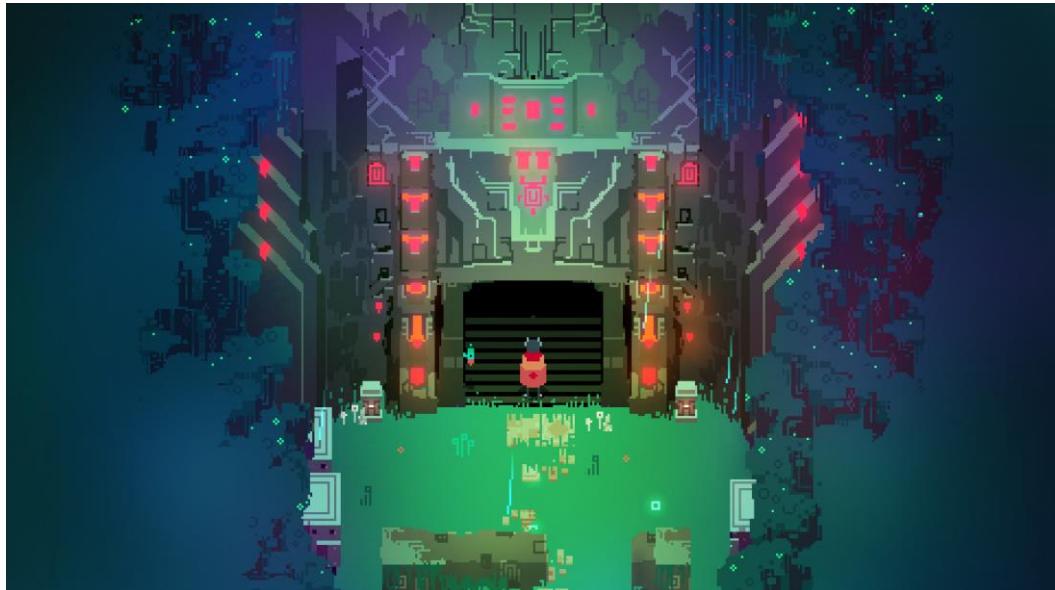


Figura 2.1. Hyper Light Drifter. [18]

Es un juego con un estilo visual de 16 bits con colores vibrantes. El personaje principal, el *Drifter*, tiene una enfermedad que lo atormenta y al buscar una forma de aliviarla, este personaje explorará las tierras olvidadas por el tiempo.

Moonlighter



Figura 2.2. Moonlighter. [19]

Es un RPG (Videojuego de rol) con una historia sobre un comerciante que durante el día se encarga de vender y comprar artefactos e insumos sencillos, y durante la noche sueña con ser un héroe que entra a mazmorras para combatir enemigos por recompensas.

Terraria



Figura 2.2. Terraria. [20]

Lanzado en 2011, Terraria es un juego tipo *SandBox* (caja de arena) de supervivencia que le presenta al jugador diversas herramientas para que pueda interactuar y modificar el mundo a su alrededor.

Undertale



Figura 2.2. Undertale. [21]

Un juego de aventuras que cuenta la historia de un niño que cae en el Inframundo, un lugar lleno de monstruos amables y con mucho sentido del humor. Durante el transcurso del juego, el protagonista podrá aprender sobre el valor de la amistad, del amor y del perdón.

Celeste

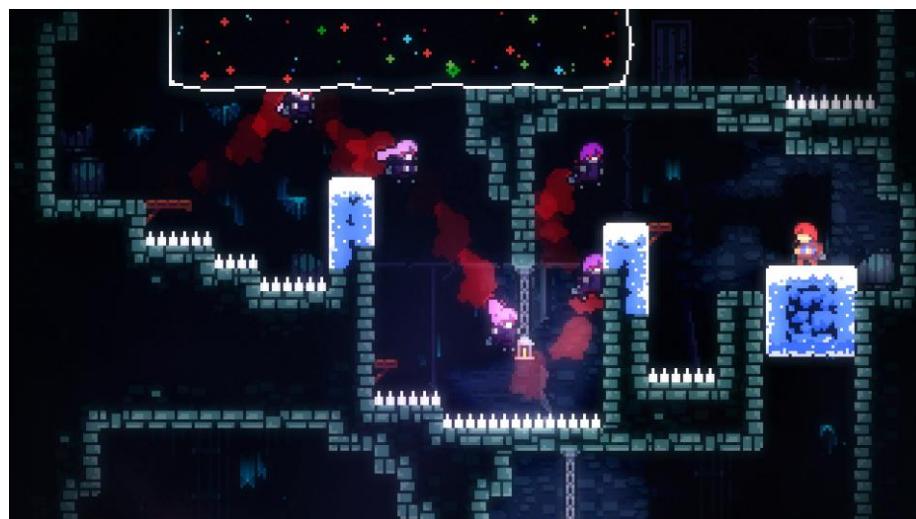


Figura 2.2. Celeste. [22]

Es un juego que presenta una historia de autodescubrimiento para la protagonista, Madeline. Es un juego de plataformas que trata sobre el viaje de Madeline a la parte más alta de la montaña Celeste.

3

Marco Teórico

En este capítulo se describen los supuestos teóricos que se utilizan para el desarrollo de este videojuego.



Gamificación

La gamificación es una técnica de aprendizaje que traslada la mecánica de los juegos hacia otro ámbito, generalmente educativo y/o profesional, con el fin de conseguir mejores resultados, de acuerdo con Gaitán [23].

Hamari [24] define la gamificación como los desarrollos culturales o tecnológicos en los cuales la realidad se vuelve más lúdica y, por lo tanto, permite la acumulación de habilidades, beneficios motivacionales, creatividad, crecimiento y felicidad en general.

Adicionalmente, Zichermann y Cunningham [25] mencionan que la gamificación permite fomentar y potenciar la motivación mediante actividades recreativas, así como reforzar la conducta mediante el uso de técnicas y dinámicas propias de los juegos.

Costa [26] nos indica que existen varios elementos que generalmente forman parte de las aplicaciones gamificadas, de los cuales algunos son las siguientes:

1. Puntos: Sirven como representación numérica del progreso del jugador.
2. Medallas: Representaciones visuales de logros o el completado de algún objetivo dentro del juego.
3. Tablas de clasificación: Listas donde cada entrada son jugadores ordenados de acuerdo con su desempeño o resultados utilizando ciertos criterios medibles de éxito, según Costa et al. [27].
4. Historias con sentido: Las historias son un elemento de los juegos que no están relacionados con el desempeño del jugador, pero proveen un contexto a las actividades que se realizan más allá de obtener puntos o medallas.

En nuestro caso, las técnicas recreativas serán la propia naturaleza de nuestro proyecto, la jugabilidad y las diversas mecánicas que se integrarán en el juego, como los puntos y las medallas por pasar los niveles, así como la historia que servirá como medio de cohesión para el resto de los elementos antes mencionados.

Programación Visual

Un lenguaje de programación visual (LPV), en computación, es cualquier lenguaje de programación que permite al usuario crear programas o algoritmos manipulando elementos visuales en lugar de expresarlos de forma textual, previniendo errores sintácticos típicos dado que la sintaxis de dichas declaraciones se encuentra implementada en las formas visuales del mismo lenguaje, como indican Budde et al. [28].

Repenning [29] nos indica que los principales obstáculos que los lenguajes visuales resuelven (o al menos, tratan de abordar) respecto a los lenguajes textuales, se pueden resumir en tres puntos importantes:

1. Sintaxis: Los LPVs utilizan íconos o bloques, formas y diagramas para construir los programas, lo cual reduce o incluso elimina los errores sintácticos.
2. Semántica: Visualmente se provee el significado de cada bloque o elemento que componen al programa.
3. Pragmática: En algunas implementaciones, se puede observar el comportamiento del programa o los estados de este conforme es ejecutado.

De acuerdo con la teoría del desarrollo cognitivo de Piaget [30], algunos de los procesos más importantes que se desarrollan en la tercera etapa llamado el estadio de las operaciones concretas (que ocurre entre los 7 y 11 años), son:

1. Clasificación: Poder identificar conjuntos de objetos según apariencia, tamaño entre otras características.
2. Descentralimiento: Se tiene en cuenta múltiples factores para resolver el problema.
3. Seriación: Poder ordenar objetos de acuerdo con ciertas características de estos.
4. Transitividad: Reconocimiento de relaciones entre varias cosas en una serie.

La programación visual, por ejemplo, la utilizada en la plataforma Scratch, es un medio propicio para introducir a las personas sin experiencia a la programación, según Sáez y Cázar [5].

Por lo tanto, podemos decir que la programación visual, debido a su naturaleza visual apoyada en el reconocimiento de formas y patrones visuales, es una vía favorable para que los niños que tengan desarrolladas las capacidades del tercer estadio descrito por Piaget sean introducidos en la programación [30].

El videojuego integrará un lenguaje de programación visual para la programación del comportamiento del robot y el uso de sus respectivas habilidades.

Aprendizaje basado en TIC

Las Tecnologías de la Información y la Comunicación (TIC) tienen diferentes definiciones atribuidas a su concepción. Belloc [31] menciona que son resultado de los avances científicos y tecnológicos en la informática y telecomunicaciones, y que, dependiendo del contexto socio-tecnológico, son el conjunto de tecnologías que permiten el acceso, producción, tratamiento y comunicación de información.

Betancourt [32] define al aprendizaje basado en TIC como la utilización de las TIC como herramientas didácticas y pedagógicas, teniendo en cuenta el factor motivacional que representan, en el proceso del aprendizaje y la enseñanza, haciendo este proceso más dinámico.

El aprendizaje basado en TIC facilita el aprendizaje a distancia, sin la presencia física de un profesor, y por su propia naturaleza ayuda a desarrollar habilidades de aprendizaje autónomo y favorece la lectura de comprensión, señala Betancourt [32].

Scratch, software que utiliza la programación visual como medio principal de comunicación como ya fue descrito anteriormente y que también es considerado una TIC, ha sido evaluado en varios contextos, por ejemplo, Meerbaum-Salant et al. [33] observaron que tras implantar un plan de formación de Scratch en dos aulas de noveno grado, se observó una mejora en la asimilación de conceptos sobre informática; además, al usarse en un centro extraescolar urbano, Malone et al [34] señala que con el paso del tiempo los participantes eran capaces de desarrollar programas dentro del software cada vez más complejos.

Un principio clave de Scratch, según Alonso [35], es que permite remendar o reparar y el videojuego desarrollado en este proyecto lo cumple al proveer la posibilidad al jugador de explorar, probar algo, ver que no funciona y arreglarlo, llegando a nuevas ideas.

Además, este proyecto también cumple con ser *meaningful*, es decir, “lleno de significado”, otro principio clave de Scratch como lo indica Alonso [35], ya que, al utilizar como medio de comunicación entre el usuario y el juego la programación visual se permite mayor comprensión de lo que se está haciendo al ser visual, como fue descrito anteriormente.

Estas dos características clave del proyecto permiten identificarlo como una TIC, al funcionar como simulador de un entorno de programación y su facilidad para comprender lo que está sucediendo. Además de tener el respaldo que tiene Scratch al ofrecer dos de sus principales características.

Desarrollo de la habilidad de resolución de problemas

La UNICEF [36] define la habilidad de resolución de problemas como “la capacidad para identificar un problema, tomar medidas lógicas para encontrar una solución deseada, y supervisar y evaluar la implementación de tal solución”.

Rogalski y Samurçay [3], hablando en un contexto de aprendizaje sobre la programación, nos indican que la habilidad de resolución de problemas es desarrollada y evoluciona en el individuo durante el proceso de adquisición de conocimiento.

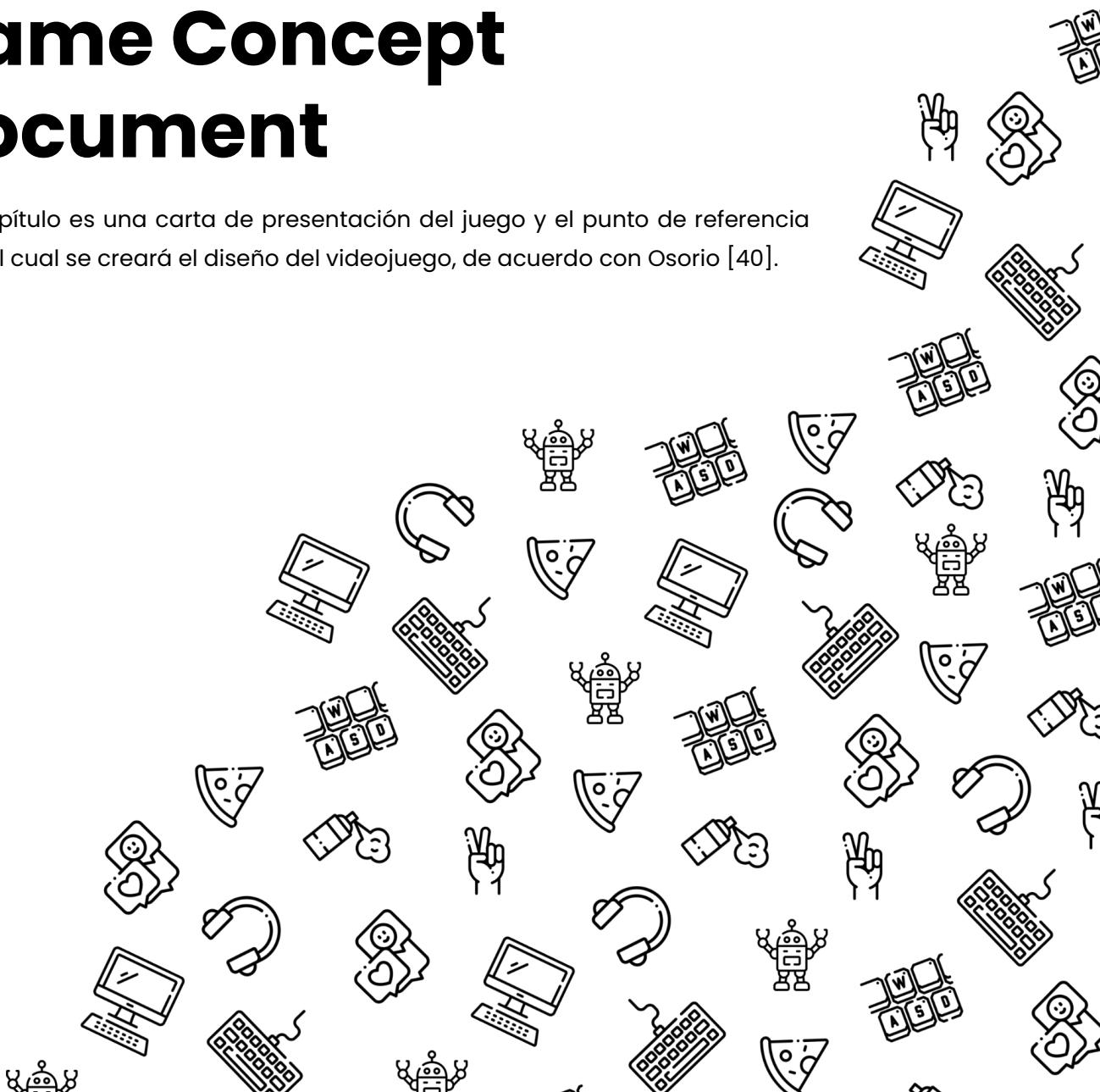
Además, esta habilidad permite construir y evaluar nuevo conocimiento, formando una relación interdependiente entre el conocimiento y la resolución de problemas, y esto a su vez, refleja la relación cercana entre desarrollar la habilidad de programar y la habilidad de resolver problemas [3].

También Tichon [37] y Ventura [38] sugieren que existe una relación clara entre jugar videojuegos y mejorar la resiliencia. Según IOM Human Resources, las personas resilientes tienen habilidades de resolución de problemas efectivas y eficientes [39] al poder identificar situaciones, ser assertivos y poder organizar estrategias al ver posibilidades donde los demás sólo ven confusión [40].

4

Game Concept Document

Este capítulo es una carta de presentación del juego y el punto de referencia sobre el cual se creará el diseño del videojuego, de acuerdo con Osorio [40].



Concepto

Eres una sobreviviente en un mundo dominado por los robots. En tu aventura para salvar a tu pueblo, aprenderás a controlar a un robot mal funcional y usar sus habilidades para poder superar los desafíos y puzzles que se encuentran en tu camino.

Tipo y plataforma

Videojuego de aventuras y resolución de desafíos de un solo jugador en tercera persona con perspectiva tipo isométrica para el sistema operativo Windows 10 de computadoras de escritorio utilizando el motor de videojuegos Unity.

Se utilizará el estilo visual Pixel Art para el arte del juego.

Público objetivo

Jóvenes de 12 años en adelante.

Atmósfera

Contexto

En un futuro cercano las naciones se enfrentaron contra una armada de robots fuera de control, dando paso a una guerra sin igual de la que las máquinas surgieron como vencedores. Ahora los sobrevivientes se ocultan y buscan una forma de poder recuperar la Tierra de las garras del enemigo.

Personajes

- Bel (Protagonista)
- Boron, El loco
- Copper
- El pueblo
- Ejército autómata.
- Robots.
- Marcus, El líder

Ubicaciones

- Asentamiento
- Casa de Boron, El Loco

Jugabilidad

Mecánicas principales

- Caminar
- Mover objetos
- Programar el robot
- Presionar botones
- Activar / desactivar palanca
- Activar / desactivar puerta

Estructura del juego

1. Reto 1:
2. Reto 2:
3. Reto 3:

Inspiración

Otros videojuegos / programas

- Scratch
- Hyper light drifter

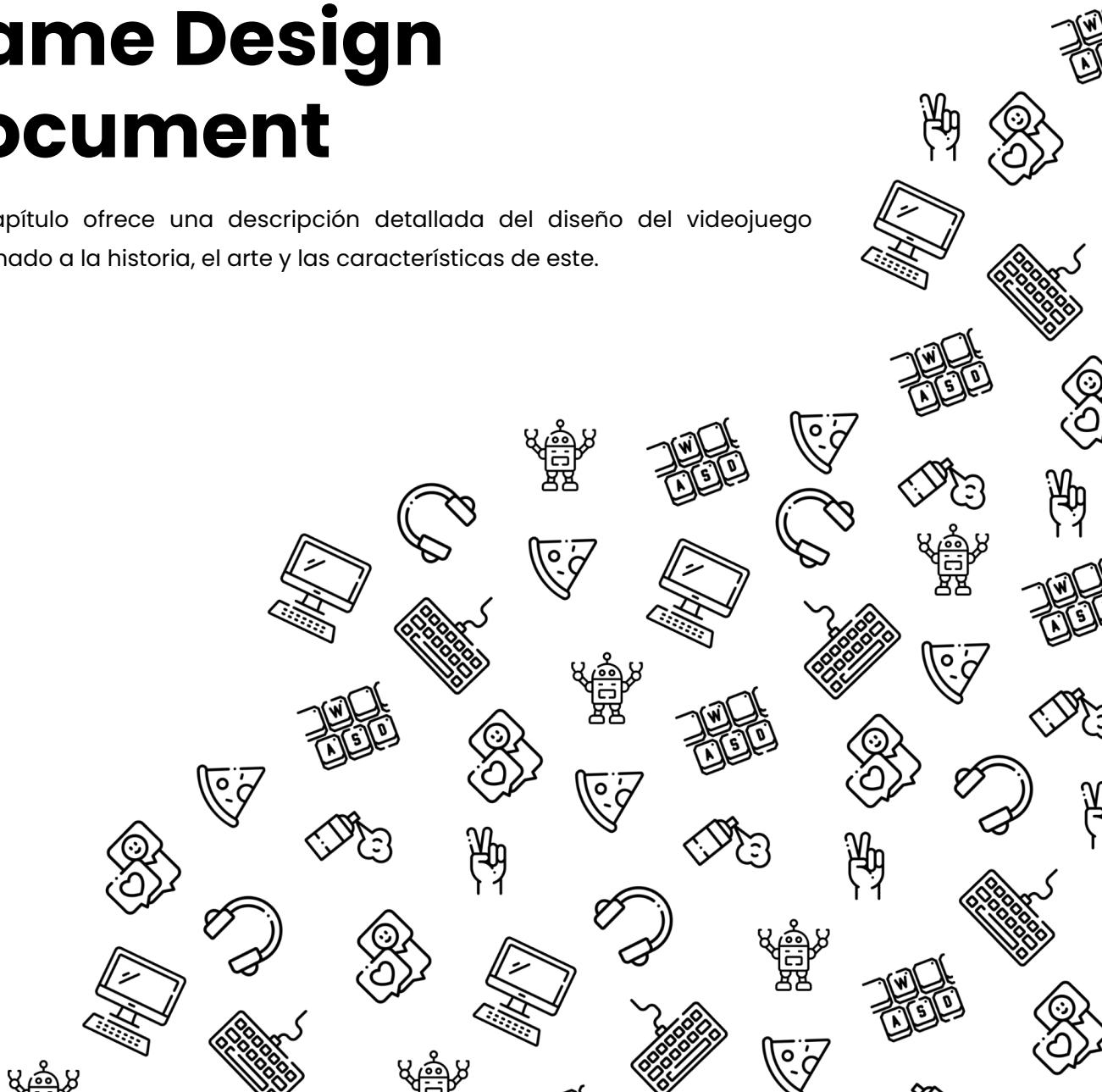
Arte

- Steampunk
- Pixel Art

5

Game Design Document

Este capítulo ofrece una descripción detallada del diseño del videojuego relacionado a la historia, el arte y las características de este.



Narrativa

Contexto

Los seres humanos desarrollaron su tecnología más allá de lo que eran capaces de controlar, tecnología capaz de fabricar una armada incontenible de máquinas capaces de arrasar con todo a su paso; eventualmente, esta tecnología cayó en las manos equivocadas, y de este error, nació el ejército más poderoso en la Tierra, y con éste, la guerra más grande que el planeta haya visto desde su creación.

Tristemente, aún con el esfuerzo colectivo de las naciones del mundo en combatir esta atemorizante amenaza, no tuvieron oportunidad alguna contra el ejército autómata.

En un último desesperado intento de sobrevivir, los pocos humanos sobrevivientes permanecieron ocultos, temerosos al mundo y al poder que alguna vez gobernaron. Generaciones pasaron y las máquinas convirtieron la tierra en una mega fábrica de ensamblaje de robots, mientras tanto, los humanos hacían su mayor esfuerzo en mantener un perfil bajo y no tentar a su suerte contra las máquinas.

Un grupo de sobrevivientes encontró un refugio entre los escombros de una instalación de comunicaciones, un lugar que hacía que los robots se alejaran de éste por alguna misteriosa razón.

Descubrieron que la antena principal del complejo, la cual previo a la guerra era usada para comunicaciones, de alguna manera interfiere con las comunicaciones entre los robots y su central, las cuales son esenciales para hacer seguimiento de la ubicación y estado de todos los robots y, en caso de encontrar rastros de vida, mandar una señal de alerta; los robots están programados para evitar zonas que interfieran con la comunicación con la central, ya que esto podría resultar en robots perdidos, por lo cual, en cuanto entran al rango de influencia de la torre los robots regresan por donde vinieron.

Historia

En el poblado protegido por la antena, vive una pequeña y cautelosa comunidad, la cual se ve ahora en riesgo ya que la antena que los esconde del ejército de robots comenzó a presentar fallas en dos de sus componentes principales, haciendo que la antena deje de funcionar de forma correcta temporalmente, lo cual permite que los robots entren mientras que la antena no esté operando.

Al principio no tardaba mucho en volver a operar de manera correcta, pero con el tiempo los períodos de fallo duraban más, y cuando la antena eventualmente volvía a funcionar, los robots que lograron entrar al territorio de los humanos retrocedían al perder contacto con su base. El temor de que la antena un día se apagara y no volviera a encenderse era cada vez mayor en los miembros del asentamiento por lo que tenían que hacer algo al respecto antes de que se les acabara el tiempo.

El líder de la comunidad convocó a un grupo de exploradores voluntarios para salir en una expedición con el fin de conseguir las piezas necesarias para reparar la antena y así poder evitar una inminente catástrofe; Bel, una pequeña maquinista del pueblo, ofrece también su ayuda en la misión, pero debido a su edad y baja estatura, fue rechazada para participar.

Bel, ignorando la orden de su líder, sabiendo que los exploradores no serían suficientes para conseguir lo que necesitan, sale por su cuenta a escondidas de los demás, esperando encontrar algo que pudiera ser útil. Bel salió del área de protección de la antena y en su camino aún cerca del asentamiento, un robot explorador la detecta y la comienza a perseguir, ella corre de vuelta a la zona segura y el robot poco después de entrar al campo detecta que sus comunicaciones están fallando y comienza a dar marcha atrás, pero queda atorada entre cables, quedando inmóvil.

Bel, corriendo sin mirar atrás, temiendo que el robot aún le siga el paso, llega a la casa de su amigo Boron, el loco del pueblo, una de las pocas personas a las que le tiene confianza, le cuenta sobre su escape y del robot que encontró. Ambos van a donde el robot y El Loco se acerca, después de analizarlo se percata que el módulo que da la instrucción al robot de atacar se ha estropeado, por lo que el robot a pesar de ejecutar su programación para exterminar humanos, nunca atacaría realmente a su objetivo; Boron, estando seguro de que el robot no atacaría, logra quitar la desconectar la condición que hace al robot volver por donde vino al detectar interferencia e inmediatamente comienza a seguir a la niña sin tomar acciones hostiles.

Después de confirmar que era seguro estar cerca del robot, lo llevaron a casa de Boron a escondidas de los demás del pueblo para realizar pruebas. Dentro de su casa, Boron

explica a Bel que él ha estado interesado en los robots y que tiene algunos módulos que ha usado en otras máquinas que le pueden ser de utilidad para poder reprogramar al robot.

Con los pocos módulos que Boron le obsequió, la pequeña Bel aprende a programar el robot y hacerlo interactuar con su entorno, y se le ocurre llevarlo con ella para que ella le ayude a conseguir las piezas que necesitan para reparar la antena.

Bel vuelve a salir a escondidas del asentamiento, pero en esta ocasión con el robot como acompañante, y en su búsqueda encuentra otros componentes para programar al robot, los cuales los aprende a usar y los utiliza para finalmente lograr encontrar una de las piezas que se necesitaba.

De regreso al asentamiento, Bel es reprendida por su desobediencia, pero de igual manera es elogiada por haber encontrado una de las piezas ella sola, cambiando la perspectiva que tiene el asentamiento hacia los robots y El Líder le permite continuar con su búsqueda de las piezas restantes.

Ubicaciones

Asentamiento

El asentamiento es un poblado humano con casas amontonadas alrededor de una gran antena de comunicaciones que los protege de los robots del exterior; Aquí los humanos traen piezas para desmantelar y construir casas y herramientas. El asentamiento está conformado por zonas destinadas a la agricultura, al comercio, viviendas, desmantelado de máquinas, entre otras. Rodeando al pueblo dentro del área de la antena no hay mucho, casi todas las partes de robots han sido recuperadas por los habitantes para ser usadas, las partes de edificios que estaban ahí fueron arrancadas de sus cimientos para construir casas cerca de la antena, por lo que se puede ver cómo solían haber estructuras ahí; y apartado del resto de las casas, se encuentra la casa de Boron, "El Loco", quien vive en exilio lejos de los demás, pero aún dentro de los confines de la zona segura.

Casa de Boron, "El Loco"

Su casa es la más alejada de la antena debido a su exilio; la ubicación de su hogar no sólo refleja su exilio, también refleja su posición dentro de su comunidad, siendo la persona que está por debajo de todos. Es una casa hecha de partes de metal oxidadas y viejas rodeada de basura y partes de robots. Aquí Boron realiza experimentos con las piezas de los robots que los demás del pueblo no se atreven a manipular.

Personajes

Bel (Protagonista)

Una joven y brillante maquinista, habitante del asentamiento. A pesar de su baja estatura, es una chica valiente y decidida.

El único amigo que tiene es Boron, "El Loco". Ella era su estudiante más joven y la más destacada de los aprendices de Boron antes de que fuera exiliado; también es la única del asentamiento que mantiene una relación con él desde su destierro; debido a esto, muchos en el pueblo la hacen menos y demeritan sus conocimientos como maquinista, sin mencionar que por su estatura y edad no le permitían participar expediciones o tareas pesadas.

Más adelante conoce al robot que se convierte en su compañero en la misión de salvar al pueblo. El robot ayuda a Bel a hacer todo lo que ella no puede, se convierte en una extensión de ella que la ayuda a ser más fuerte, más grande y dura, permitiéndole superar desafíos que nadie más sería capaz de hacer sin ayuda.

Diseño



Figura 5.1. Diseño principal de Bel.

Bocetos y otros sprites



Figura 5.2. Bocetos de Bel.

Boron, "El Loco"

Un hábil maquinista de edad avanzada, quien enseñó a muchos la ciencia detrás de las máquinas, él mostró gran interés en los robots y su funcionamiento, él asegura que si comprendieran con mayor profundidad el funcionamiento de los robots serían capaces de controlarlos y usarlos a su favor; una idea no muy popular entre los sobrevivientes, sobretodo los extremistas anti tecnología.

Las invenciones de Boron se volvieron cada vez más avanzadas y los extremistas no permitirían que "jugara con fuego" dentro del pueblo, por lo que convencieron al líder de lo peligroso que es y que debía ser exiliado. El líder, Marcus, tuvo que tomar una decisión y finalmente lo exilió, obligándolo a vivir alejado del pueblo para así no amenazar la seguridad de los demás con sus experimentos.

Es el mentor y más cercano amigo de Bel, quien, a pesar de su destierro, aún lo visita de manera constante y no lo trata de manera distinta a pesar de su exilio.

Diseño



Figura 5.3. Diseño principal de Boron.

Bocetos y otros sprites



Figura 5.4. Bocetos de Boron 1.



Figura 5.5. Bocetos de Boron 2.



Figura 5.6. Bocetos de Boron 3.

Copper

Es el robot que persigue a la protagonista en el principio de su aventura. Tras sufrir un accidente intentando abandonar el territorio humano, algunos de sus componentes fueron dañados, entre ellos el componente que hace a los robots atacar a los humanos. Tras ser levemente modificado por Boron, pasa a convertirse en el acompañante de Bel en su misión para salvar a su pueblo de la amenaza robot.

Copper tiene una compuerta que le da a Bel acceso a su interior, permitiéndole reconfigurarla para realizar diferentes tareas; de alguna manera, Copper es una extensión de Bel, dándole toda la fuerza que ella no tiene, haciendo que no necesite de nadie más que su mecánico acompañante.

El pueblo

La mayoría de los humanos que intentaron ocultarse de las máquinas fracasaron, ya que eventualmente fueron localizados y eliminados, sin embargo, existe un pequeño poblado de humanos que sobrevivió debido a que se establecieron cerca de las instalaciones de una estación de comunicaciones que tiene una gran antena que emite constantemente una señal, un componente del circuito se estropeó, haciendo que la señal no viaje tan lejos, pero la distancia que consigue abarcar tiene un intenso flujo de señales, interfiriendo con las comunicaciones de los robots con su base al estar dentro de su rango de influencia. Los habitantes del asentamiento se dieron cuenta de esto y probaron los límites de la antena

para tener claro qué tanto pueden apartarse de su base sin perder la protección de la antena.

La mayoría de los sobrevivientes, aterrados por la amenaza robot y por la ambición de la humanidad en el pasado los arrastró a esta situación, abandonaron sus conocimientos conectados con la investigación y fabricación de robots, convirtiéndose en un tabú, sin embargo, era aceptada la fabricación de otros artefactos eléctricos mientras se mantuvieran alejados de la concepción de los robots, por lo que aún unos pocos eran capaces de fabricar herramientas que les facilitara enfrentar y defenderse de los robots, así como explorar tierras desconocidas. A pesar de que la tecnología en cierto grado sea aceptada en el pueblo, hay algunos extremistas anti tecnología que se rehúsan a usar herramientas avanzadas o estar cerca de quienes las usan.

Debido al miedo a los robots, la cercanía de las casas a la antena refleja el estatus de sus dueños en la sociedad, en donde mientras más lejos estás del centro, menos valor tiene tu propiedad y además de reflejar directamente tu posición en la sociedad, siendo la persona con la casa más cerca a la antena, el líder del pueblo y la más alejada Boron, el loco exiliado.

Ejército autómata

Un gran ejército que domina el mundo formado por millones de robots. Existen robots de todas formas y tamaños, diseñados para diversas tareas, desde rastrear y exterminar humanos, hasta recolección de robots averiados o perdidos.

Este ejército cuenta con cuartel general, su ubicación se desconoce, pero se cree que todos los robots se comunican directamente con la central de comunicaciones de dicho cuartel. Además de esto, cuentan con múltiples cuarteles esparcidos por sectores en todo el mundo, cada uno se encarga de cubrir su correspondiente sector, así como fabricar y desplegar los robots que sean necesarios para cubrir las necesidades de dicho sector; estas necesidades incluyen extracción de materiales para construcción y reparación, búsqueda y caza de humanos, mantenimiento de unidades, entre otras.

Robots

Los robots son la fuerza principal del ejército autómata. Son máquinas programadas para cumplir objetivos específicos; son capaces de ejecutar múltiples tareas al mismo tiempo y mantener comunicación con el cuartel general en todo momento.

Los robots son programados usando módulos físicos que representan instrucciones, estructuras de control, espacios de memoria, etcétera. Casi todos los robots tienen instalado un código ofensivo en caso de que encuentren humanos u otras criaturas durante la ejecución de su tarea; esto no significa que cualquier robot que lo tenga tomará acciones siempre que encuentre un humano, en caso de ser un robot no diseñado para el

combate, sólo atacará en caso de que el humano no represente una amenaza para la integridad del robot, por lo que niños y ancianos peligran más en el exterior ya que casi cualquier robot con el que se crucen intentará exterminarlos.

Se recomienda salir en grupos ya que hay pocos robots que estén programados para enfrentar incluso grupos grandes de humanos. Una estrategia común pero poco ortodoxa entre exploradores es que el miembro más débil del grupo guíe al grupo sin ninguna clase de camuflaje, ya que así incluso los robots débiles podrían acercarse creyendo que va sólo y sorprenderlo con un ataque en equipo. Esta estrategia es peligrosa en más de un sentido, ya que, si alguno de los robots se da cuenta que no es un solo humano, sino un grupo, podrían alertar a base central para que ellos manden a un o varios robots especializados en combate en masas.

Bocetos y sprites



Figura 5.7. Diseño final de los robots.

Marcus, El líder

El firme y duro líder del asentamiento humano. Su función dentro de su comunidad es la de supervisar y guiar a su pueblo en múltiples asuntos, tales como la supervisión de la seguridad del asentamiento, la resolución de conflictos dentro del pueblo, entre otros. Además de ser el cazador y explorador más experimentado de la tribu, convirtiéndose también en la cabeza del grupo de caza y exploración.

Desde pequeño se preparó para tomar la posición de líder. Fue criado por su padre junto con su hermano mayor, en un principio su padre quería que su primogénito se convirtiera en su sucesor, pero éste estaba desinteresado en el puesto ya que estaba mucho más

interesado en la investigación y la tecnología. Marcus, para demostrar a su padre su valía, entrenó sin parar con el fin de convertirse en lo que su padre consideraba un líder digno de su pueblo. Su padre sin embargo, siempre demeritó los esfuerzos y logros de Marcus, siempre calificandolo como insuficiente; haciendo que Marcus desarrollara resentimiento hacia su hermano, ya que lo culpaba por el rechazo de su padre.

Eventualmente, la muerte llegó a su padre, dejando al pueblo momentáneamente sin un líder, en su lecho de muerte, el moribundo líder aún pedía a su primogénito tomar su lugar, cosa que rechazó una vez más. Tras la muerte de la cabeza del pueblo, al no haber un líder, Marcus se autoproclamó el nuevo líder del asentamiento y se propuso demostrar a todos lo equivocado que estaba su padre sobre sus capacidades.

Items

Puerta

Puerta bloqueada que es abierta mediante la pulsación simultánea de dos botones.

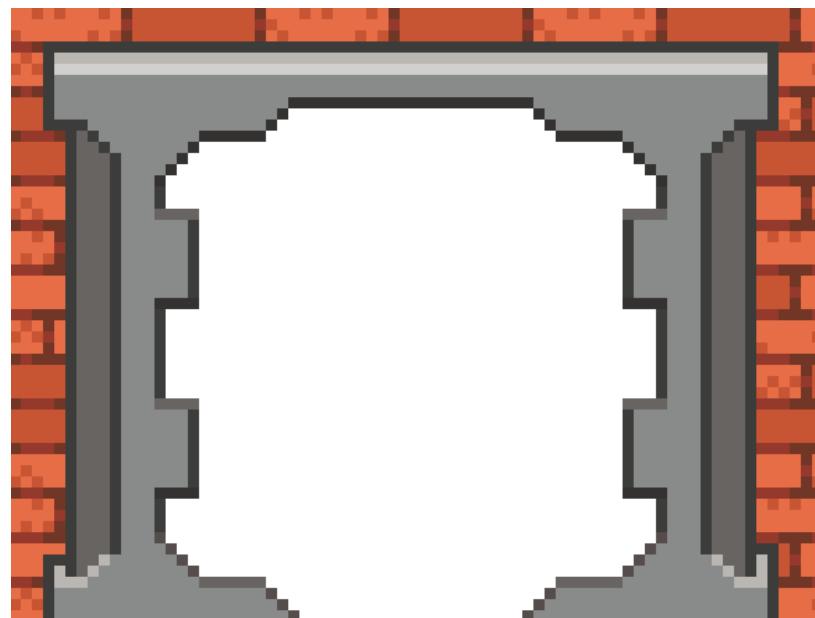


Figura 5.8. Marco exterior.

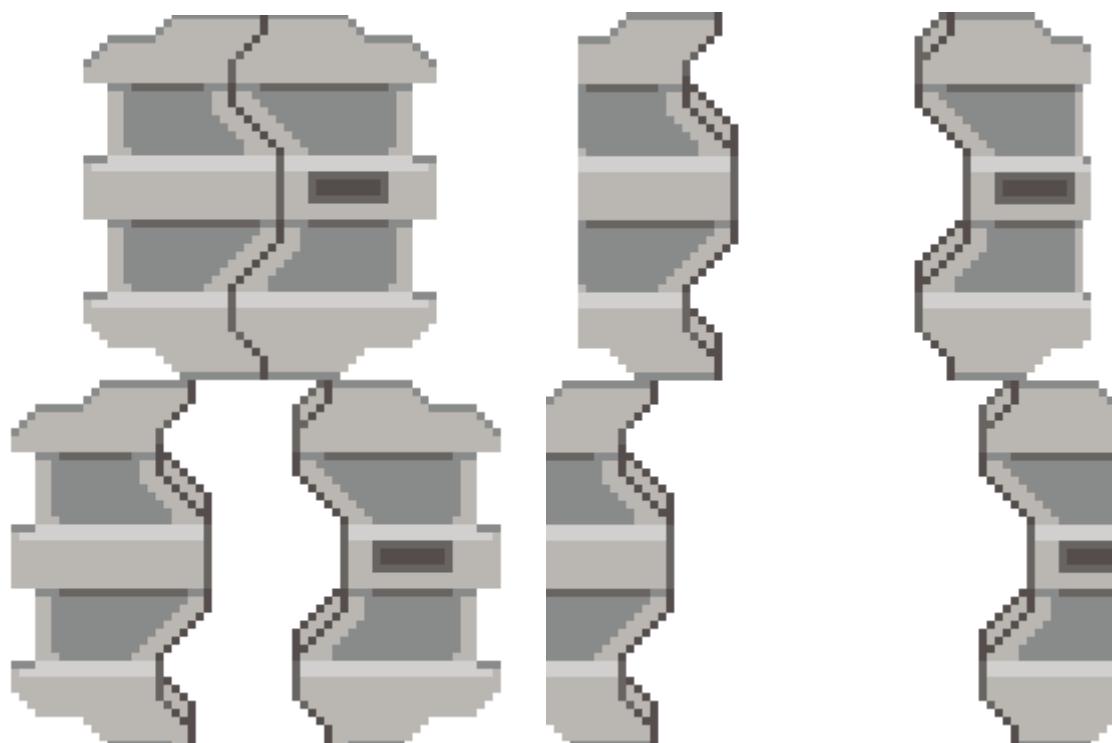


Figura 5.9. Fases de apertura.

Palanca

Permite encender o apagar otros objetos, o cambiar su estado de forma continua.

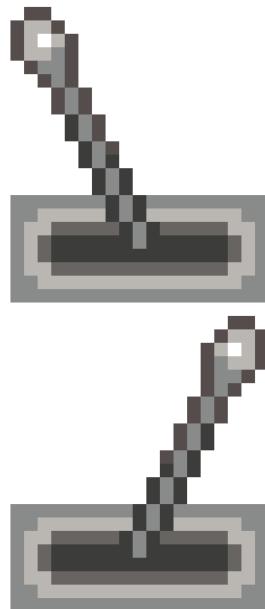


Figura 5.10. Fases de palanca.

Botón

Permite encender o apagar otros objetos, o cambiar su estado de forma temporal.

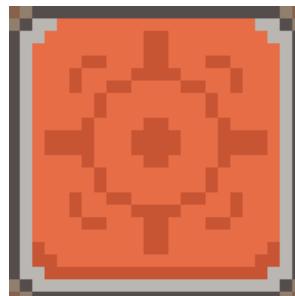


Figura 5.11. Botón presionado.

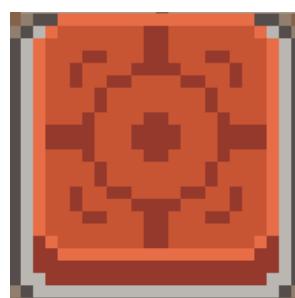


Figura 5.12. Botón no presionado.

Caja

Elemento móvil que permite presionar otros objetos, como un botón.

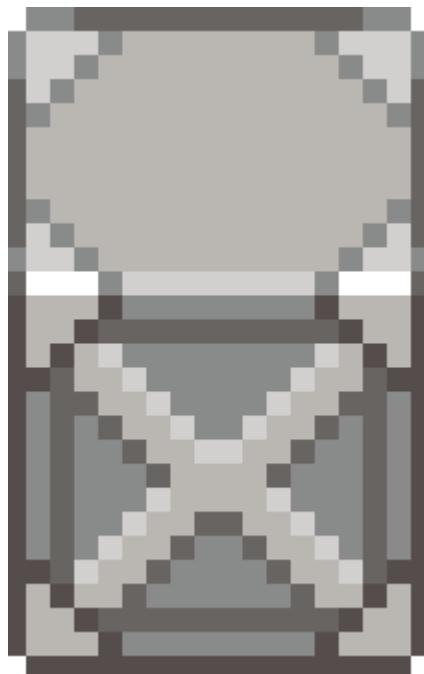


Figura 5.13. Diseño de caja genérica.

Textura de ladrillos

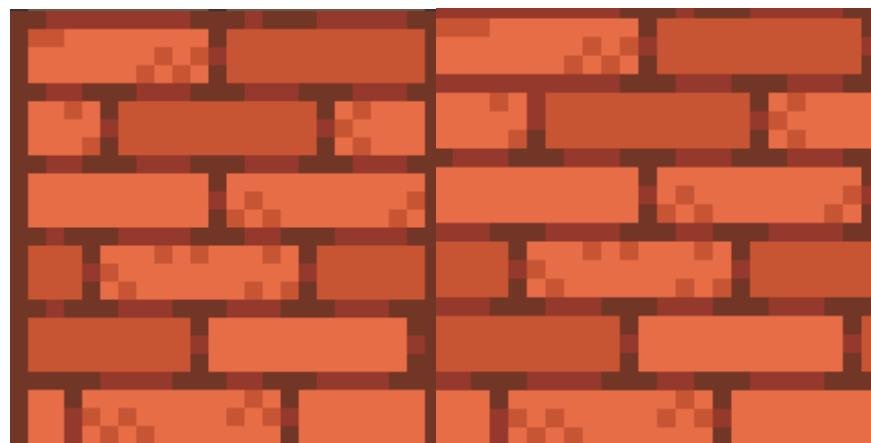


Figura 5.14. Texturas de ladrillo.

Textura de adoquines

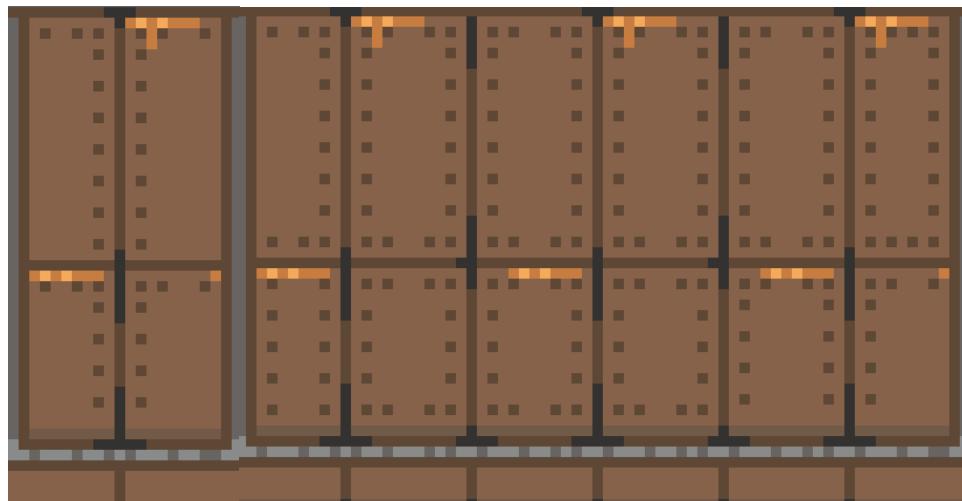


Figura 5.15. Texturas de adoquines.

Textura de suelo de mosaicos



Figura 5.16. Texturas de mosaicos.

Textura de suelo simple

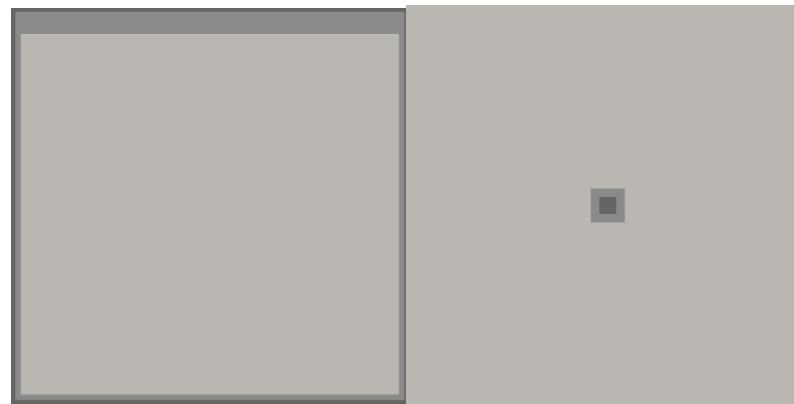


Figura 5.17. Texturas de suelo simple.

Barandal conectable

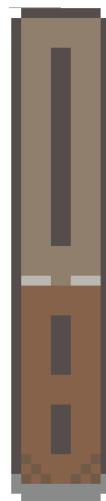


Figura 5.18. Barandal vertical.

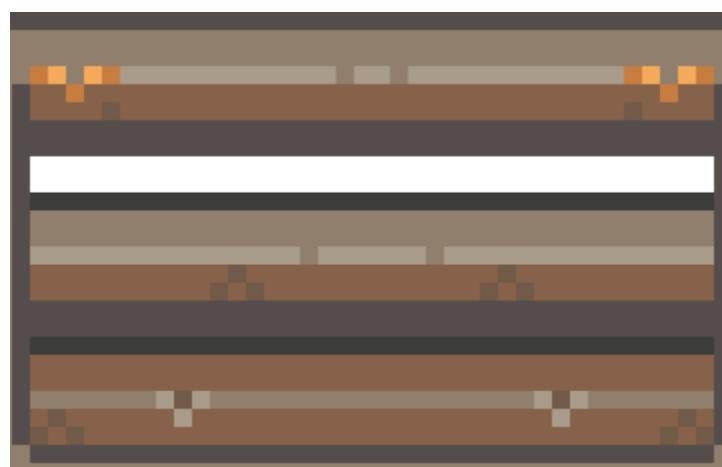


Figura 5.19. Barandal horizontal.

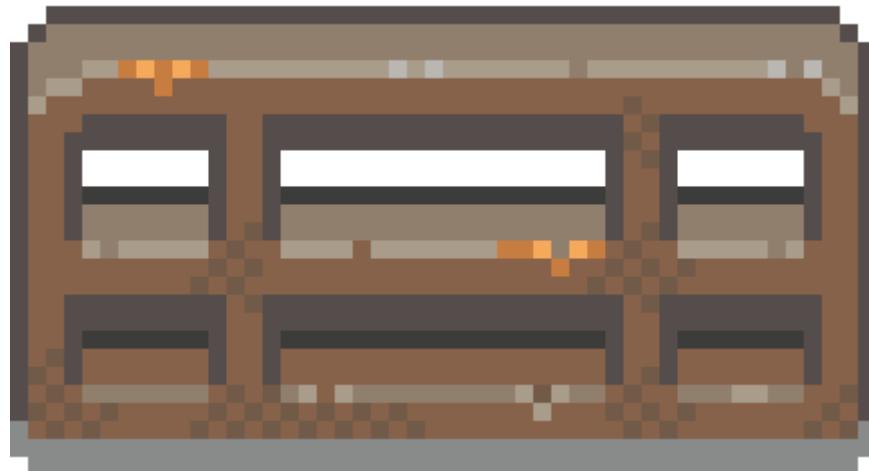


Figura 5.20. Barandal conectado.

Máquina grande

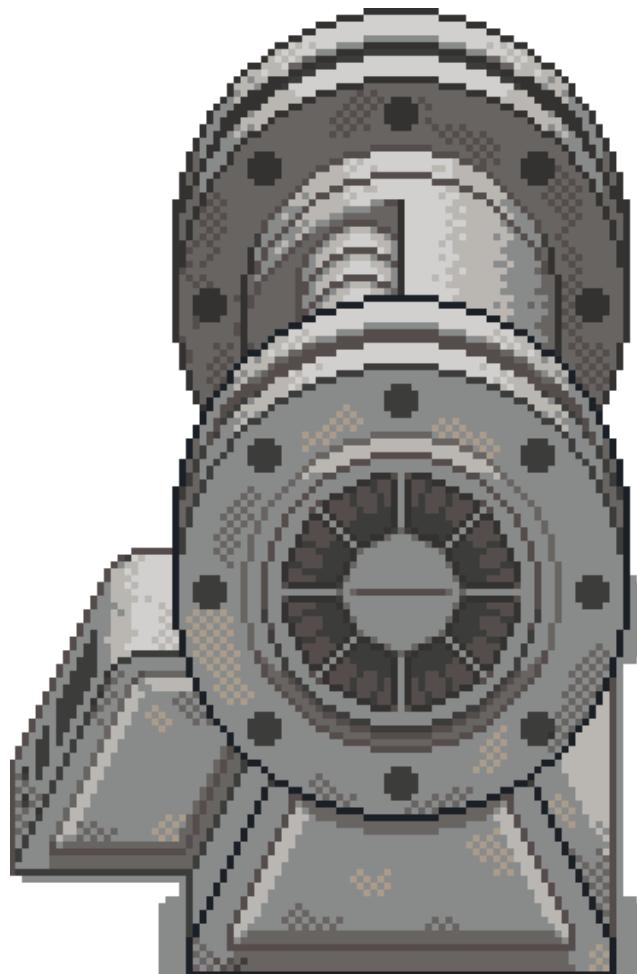


Figura 5.21. Diseño de máquina grande.

Bloques de evento



Figura 5.22. Bloque de inicio.



Figura 5.23. Bloque final.

Instrucción genérica



Figura 5.24. Bloque de instrucción genérica.



Figura 5.25. Bloque de instrucción genérica.

Bloque de repetición



Figura 5.26. Repetición condicional.



Figura 5.27. Repetición con valor numérico.

Bloque de condición



Figura 5.28. Condición simple.

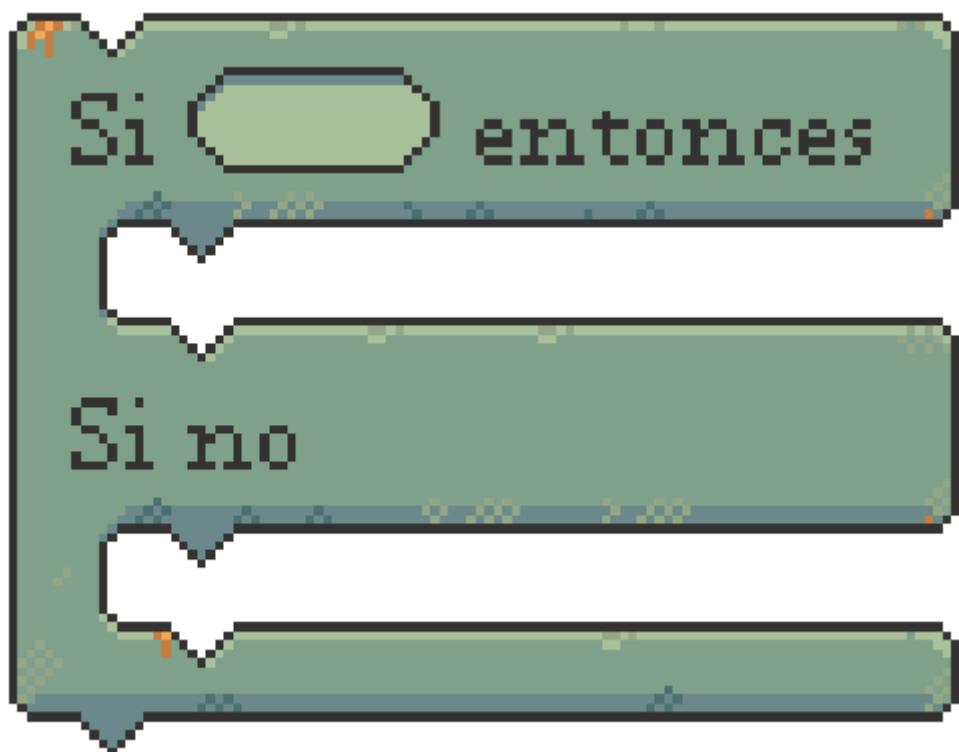


Figura 5.29. Condición compuesta.

Condicionales



Figura 5.30. Constantes de verdadero y falso.



Figura 5.31. Condicional Y



Figura 5.32. Condicional O.



Figura 5.33. Condicional de negación.

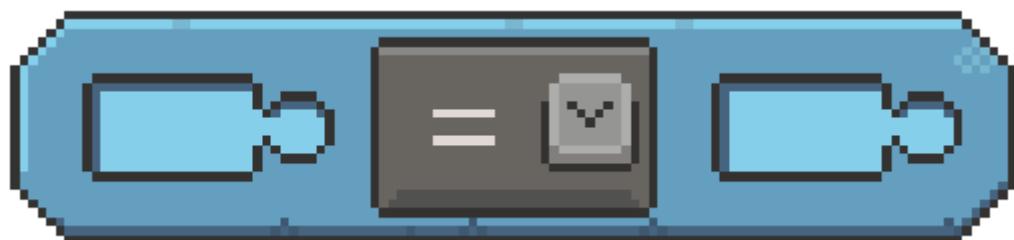


Figura 5.34. Condicional de comparación de valores numéricos.

Valores numéricos y arreglos



Figura 5.35. Valor de entrada numérico.



Figura 5.36. Variable.



Figura 5.37. Operación numérica.



Figura 5.38. Arreglo.

Pantallas del menú principal

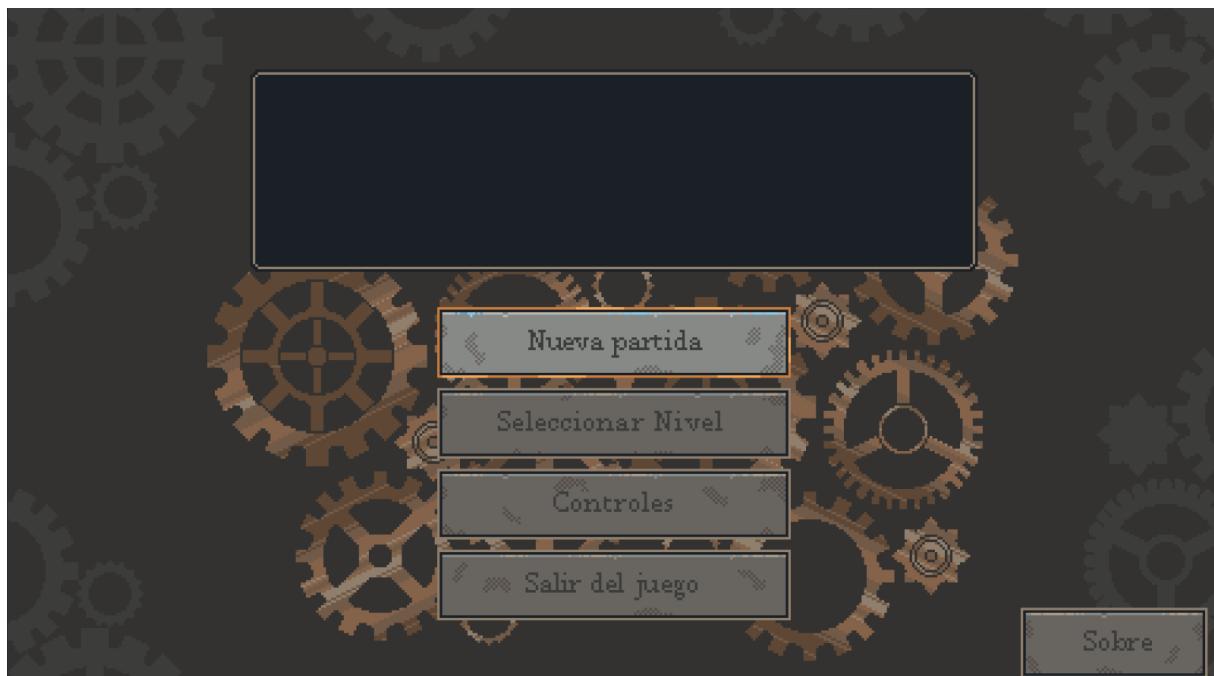


Figura 5.39. Pantalla de inicio.



Figura 5.40. Ventana emergente para preguntar si cerrar el juego.



Figura 5.41. Pantalla de información de desarrolladores.

Guión

Escenas

- Cinemática inicio
 - Fábrica interior – día. Maquinistas y científicos fabricando robots
 - Interior. Filas de robots malvados se ven entre la oscuridad
 - Exterior, día. Guerra entre humanos y robots
 - Mundo bajo el poder de los robots
 - Exterior, instalación de comunicaciones. Se ve cómo cambia con el tiempo convirtiéndose en el asentamiento. Toma cerca de la antena mostrando cómo falla
- Diálogo inicial
 - Exterior Asentamiento, noche. La tribu reunida alrededor de algo como una hoguera. Se discute el tema de las expediciones para reparar la antena, Bel se acerca al jefe y pide la dejen acompañarlos y es rechazada
- Al día siguiente
 - Exterior, Asentamiento, día. El grupo ya se fue y Bel se prepara para salir a escondidas
 - Exterior, Frontera de seguridad, día. Bel sale de los confines del área segura , comienza a apartarse y escucha un ruido, seguido de esto ve a un robot corriendo hacia ella. Bel corre de vuelta a la zona segura pero antes de llegar, por la prisa de huir del robot, tropieza y cae por una pendiente empinada, terminando en el piso, detrás de ella, el robot, el cual también cayó, dañando su programación. Bel se levanta, ve al robot junto a ella y grita, cierra los ojos unos segundos, los vuelve a abrir y se da cuenta de que el robot no está haciendo nada “se descompuso?” Comienza a alejarse lentamente de él y el robot la sigue. En este momento podríamos mostrar los controles básicos del jugador, indicando que se mueva a la derecha, abajo, izquierda y arriba de manera consecutiva, para así ver que el robot la sigue sin herirla.
 - Exterior, Casa de Boron. Bel lleva al robot con Boron, él la ve acercarse en la distancia y se espanta, toma una herramienta y corre hacia ella para atacar al robot y ayudarla. Conforme corre hacia ella se da cuenta de que el robot no le está haciendo nada y se calma. Bel le cuenta lo sucedido y Boron comienza a revisar al robot, explica a Bel que ocurrió dentro del robot. Boron

para probar, le da a Bel unos pocos módulos y le pide que los inserte en el robot de una forma en específico. Así Boron le enseña a Bel a programar al robot. Bel decide que con la ayuda de su robot seguiría con su misión. Boron le da herramientas que la ayudarán en su aventura y la acompaña a la frontera y se despiden.

SCRIPT

FADE IN:

MONTAJE DE CINEMÁTICA:

- Toma de una ciudad con edificios altos, vehículos voladores entre otras cosas.

NARRADOR

La raza humana estaba en la época dorada del desarrollo tecnológico. Una era en la que la tecnología era desarrollada a una velocidad exponencial

- El interior de una fábrica muestra maquinistas y científicos fabricando robots. En el centro hay un robot a medio armar, con piezas colgando y sueltas. Tiene un ojo/faro en el centro que conforme progresó la escena va encendiéndose de un color rojo brillante.

NARRADOR

Pronto un laboratorio de maquinistas comenzó a trabajar en un invento revolucionario. Máquinas reprogramables capaz de realizar varias funciones, tareas complejas, manipular objetos, los robots fueron creados para servir a la humanidad haciendo todo tipo de tareas. Tristemente...

- Escena oscura en la que sólo se ven luces rojas alineadas, conforme sigue la escena, la iluminación comienza a permitir que se vean más los robots. La cámara sube y permite ver cientos de luces de más robots, todos alineados como soldados.

NARRADOR

... Esta tecnología fue robada y cayó en manos equivocadas, de este error nació el ejército más poderoso en la Tierra y con éste...

- Toma de un campo de batalla entre humanos y máquinas.

NARRADOR

... La guerra más grande que el planeta haya visto desde su creación. Tristemente la humanidad no tuvo oportunidad.

- Misma toma mostrada de la ciudad ahora en ruinas, con humo y un cielo más oscuro.

NARRADOR

Algunos humanos sobrevivieron ocultándose, aunque no todos compartieron la misma suerte, ya que la mayoría de grupos de sobrevivientes terminaba siendo rastreado y eliminado..

- Instalación de comunicaciones con gran antena en el centro. Se muestra su apariencia cuando aún estaba en buenas condiciones.

NARRADOR

Un grupo de afortunados sobrevivientes se ocultaron en unas instalaciones de comunicaciones, se quedaron esperando el inminente ataque de los robots pero pronto se dieron cuenta de que algo raro pasaba.

- Las mismas instalaciones, ahora mostrando cómo de la antena sale un campo semitransparente, como si fuese un escudo.

NARRADOR

Parece que la antena principal del complejo por alguna razón interfiere fuertemente con las comunicaciones de los robots, cuando un robot se acerca lo suficiente a la antena pierde comunicación con su cuartel general, haciendo que regrese por donde vino, dando prioridad a recobrar comunicación con el cuartel.

- Se sigue mostrando la antena, pero se muestra cómo cambió su apariencia, mostrando su estado años más tarde, siendo ahora el refugio de los sobrevivientes.

NARRADOR

Las instalaciones se convirtieron en el hogar de múltiples generaciones que se adaptaron al orden impuesto por el ejército autómata, manteniendo un perfil bajo y no tentando su suerte contra las máquinas. Los supervivientes pudieron vivir de manera

relativamente tranquila dentro de los confines de su territorio. Pero esta tranquilidad no duraría para siempre.

- Toma cerca del suelo, donde se permite ver residentes del asentamiento presenciando cómo robots cruzan la frontera de seguridad, algunos son vistos huyendo.

NARRADOR

La antena que los protegió por generaciones comenzó a fallar. La antena dejaba de funcionar apropiadamente, vulnerando la seguridad de los refugiados momentáneamente;

- Una toma similar, mostrando a las mismas personas de la toma pasada, ahora viendo a los robots retirarse.

NARRADOR

Después de un breve periodo, la antena volvió a funcionar correctamente, ahuyentando a los robots fuera de su alcance. Esto podría parecer un problema no tan grande a primeras, pero con el paso del tiempo los periodos de fallo se extendieron cada vez más.

El temor de que un día la antena no vuelva a funcionar correctamente crecía en el corazón de los residentes cada vez más.

- Vista en picada de una reunión entre los hombres del asentamiento, todos reunidos alrededor del fuego de una fogata, dispuestos a discutir la situación.

NARRADOR

Los sobrevivientes, desesperados, tuvieron una reunión con su líder, Marcus, para discutir cómo solucionarían esta terrible situación.

FIN DE MONTAJE

EXT. ASENTAMIENTO - NOCHE

La tribu de sobrevivientes se reúne en el centro del pueblo para discutir sobre el problema del mal funcionamiento de la antena. Múltiples miembros del pueblo se encuentran reunidos alrededor de su líder, Marcus.

MARCUS, un firme y estricto líder; es un tipo duro y temperamental, es el miembro más fuerte y experimentado del pueblo.

MARCUS

(serio)

Me temo que nos estamos quedando sin opciones y tiempo. Hemos realizado múltiples búsquedas en las proximidades del asentamiento y no encontramos las piezas que necesitamos para reparar la antena; me queda claro que si queremos encontrarlas tendremos que adentrarnos alguna de las fábricas más cercanas, son los únicos lugares donde encontraremos lo que necesitamos.

No obligaré a nadie a acompañarme tan lejos del asentamiento sin saber qué máquinas podríamos encontrarnos ahí. Si alguno se ofrece a acompañarme en esta expedición, que lo diga ahora.

Un grupo de adultos con armas se ofreció a apoyar en la expedición. Algunas otras personas dispersas se ofrecieron también

Entre todos los adultos, una pequeña niña, BEL, una brillante maquinista, es una niña con una actitud fuerte y rebelde pero que quiere lo mejor para su pueblo, aún cuando no es tan bien recibida por sus propios miembros. Bel hace un gesto indicando que también ofrecía su participación; en ese momento Marcus, el líder, la detiene.

MARCUS

Bel, no. Esta no es una excursión en la que una niña como tu pueda participar, sólo te pondrías en riesgo y al resto del grupo por tener que cuidarte además de que nos retrarasarías.

BEL

¡Juro que no los atrasaré! Sé que puedo ayudarlos, si sólo me das una oportuni...

Marcus interrumpe a Bel a media oración.

Marcus

(molesto)

¡Bel, no me hagas repetirlo, soy el líder y soy responsable de la seguridad de todos aquí! No voy a llevar a una niña a una expedición tan lejos del refugio sólo por un capricho. Tú te quedarás aquí, fin de la discusión.

Marcus le da la espalda a Bel y se retira. Bel se queda de pie en su lugar, molesta e impotente.

EXT. FRONTERA DEL ASENTAMIENTO – DÍA

Marcus marcha fuera del asentamiento con sus hombres, Bel espera escondida detrás de un objeto a que se vayan para que así ella pueda salir después. Pasa un rato y Bel cruza la frontera del asentamiento, comienza a apartarse.

Después de un rato caminando, escucha un ruido, comienza a voltear en todas las direcciones hasta que ve un robot, es un robot con un cuerpo esférico, dos ojos, piernas y brazos. Apenas se da cuenta de la situación en la que se encuentra, Bel comienza a correr de vuelta a la frontera con el robot corriendo detrás de ella. Bel corre desesperadamente y sin darse cuenta tropieza y cae por una pendiente; el robot que la persigue también cae por la pendiente recibiendo un fuerte golpe contra el piso, terminando en el piso junto a Bel.

Bel aún aturdida por el golpe, levanta la mirada y ve que el robot ya estaba de vuelta sobre sus pies, justo a un lado de ella. Bel suelta un grito, se levanta y da unos pasos retrocediendo hasta chocar de espaldas con una pared cerrando sus ojos esperando recibir un ataque. El robot siguió sus pasos y se detuvo justo frente a ella..

Pasan unos segundos y no ocurre nada, Bel abre nuevamente sus ojos y ve al robot frente a ella sin hacer nada.

BEL (DIALOGO INTERNO)

¿Qué está pasando? ¿Se habrá averiado?

Bel da unos pasos en diferentes direcciones y ve cómo el robot la sigue siempre a donde vaya sin atacar.

BEL (DIALOGO INTERNO)

No sé qué está pasando... pero de alguna forma este robot parece no ser hostil hacia mí. Será mejor que lo lleve con Boron.

EXT. CASA DE BORON – DÍA

Bel, acompañada por el robot, camina a casa de Boron.

BORON, el exiliado, es un maestro maquinista y mentor de Bel, fue expulsado por orden del líder después de que los extremistas lo acusaran de ser una amenaza contra la seguridad del pueblo por sus ideas sobre investigar a los robots e investigar a los robots.

Boron estaba trabajando fuera de su casa cuando llegó Bel junto al robot. Boron se exaltó, pero rápidamente se dio cuenta de que el robot no sólo estaba dentro de la zona segura, sino que también siguió a Bel hasta ahí sin herirla.

Boron se acerca al robot y comienza a examinarlo mientras pregunta a Bel.

BORON

Esto es interesante, parece que algunos de los circuitos dentro de este robot se descompusieron tras un fuerte impacto, el robot está funcionando sólo con algunos de los componentes que siguen funcionando en él. Es por eso que a pesar de cruzar la frontera no volvió, lo mismo explica el por qué te sigue sin atacarte, los módulos encargados de ese comportamiento dejaron de funcionar...

Boron se calla por un momento.

Bel... ¿Dónde encontraste este robot?

BEL

Verás... Quise salir para ayudar a conseguir las piezas para reparar la antena por mi cuenta y...

Boron la interrumpe sin dejar de examinar al robot.

BORON

¿Saliste tú sola fuera de nuestras fronteras? No suena a algo que Marcus permitiría, saliste a escondidas, ¿verdad?

BEL

(molesta)

¡Sé lo que dirás, lo mismo que Marcus siempre me dice pero ya estoy harta, sólo quiero demostrar que puedo ayudar!

Bel intenta calmarse y continúa.

BEL

Pero no vengo a que me sermonees, vengo porque algo pasa con este robot, nunca habíamos podido estar cerca de uno que aún esté funcionando, todos los robots que se acercan al asentamiento son destruidos por los perros de Marcus.

(emocionada)

¡Esta es la oportunidad perfecta para averiguar más sobre robots y cómo funcionan! Vamos Boron, sé que este robot te intriga, ayúdame a usarlo. Estoy segura de que con su ayuda seré capaz de volver con las piezas para reparar la antena.

BORON

No lo sé, Bel... Aún con la ayuda del robot sería peligroso salir tú sola...

BEL

Boron, no hay nadie que pueda acompañarme, los demás del pueblo me delatarían si se enteran que metí un robot al asentamiento, y Marcus y sus hombres no dudarían en destruirlo. La única persona que estaría dispuesta a acompañarme eres tú, pero ya eres muy viejo y la vida en el exilio no te ha sentado nada bien.

Volveré a salir Boron, no puedo quedarme cruzada de brazos mientras la antena falla más y más, lo haré ya sea con tu ayuda o sin ella.

Boron lo piensa unos momentos y suspira.

BORON

De acuerdo, pero si vas a salir no puedo permitir que salgas así sin más, necesitas estar más preparada.

Boron le entrega a Bel unos goggles como los que él siempre tiene puestos.

BORON

Estos goggles son similares a los que yo uso. Te permiten ver el flujo de la electricidad, incluso a través de paredes y objetos. Si vas a la fábrica fuera del asentamiento puede que te encuentres con maquinaria que necesites usar o puertas que quieras abrir. Usar los goggles te permitirá ver a qué componentes están conectados a ellos para poder activarlos.

Otro importante uso de los goggles es que te permiten ver cómo fluye la electricidad dentro de los robots. Con esto puedes no sólo ver en donde hay robots hostiles, también puedes analizar el patrón con el que la electricidad fluye en ellos y determinar cómo están programados, permitiéndote predecir su comportamiento en lugar de arriesgarte sin saber qué es lo que podrían hacer. Es indispensable que uses estos goggles siempre que llegues a un nuevo lugar o estés en exteriores fuera del asentamiento para así ubicar a los robots cercanos y actuar con cautela.

Ahora bien, sobre el robot. Te enseñaré lo básico de cómo funciona la programación de estas máquinas y así puedas reprogramar al tuyo.

Boron toma un puñado de componentes y se los da a Bel.

BEL

¿Qué son estos?

BORON

Son algunos componentes que he podido rescatar de robots destruidos por los cazadores, los robots usan funcionan con circuitos armados con estas piezas.

Boron abre la compuerta frontal del robot y le enseña el circuito a Bel

[Lo que sigue es la explicación del tutorial, una vez tengamos un prototipo programable podremos comenzar a redactar el tutorial]

BORON

¡Excelente! Estoy orgulloso, Bel. Eso es todo lo que puedo enseñarte de momento con los módulos que te pude proporcionar.

BEL

(emocionada)

¡Hurra! Somos el mejor equipo e...

BORON

¿Qué sucede Bel?

BEL

Es sólo que siento que debería darle un nombre.

BORON

¿Cómo te gustaría ponerle?

Bel piensa por un momento el nombre que le gustaría dar a su robot. (Podríamos permitir al jugador escoger el nombre del robot, dando como sugerencia Copper).

BEL

(entusiasmada)

¡Copper! sí, su nombre es Copper.

BORON

Jajaja, excelente nombre. Bel y Copper.

Boron guarda silencio un momento y suspira.

BORON

Supongo que ya estás lista... Quisiera que no lo hicieras pero sé que no puedo detenerte. Recuerda todo lo que te enseñé y ten mucho cuidado ahí afuera. Ahora ve, y no dejes que te vean a ti o a Copper los demás del pueblo.

Bel abraza a Boron y se va corriendo junto con Copper detrás de ella.

Ext. Frontera del Asentamiento - día

Bel y Copper llegan una vez más a la frontera, Bel se pone sus goggles y salen juntos en su aventura para encontrar las piezas para la antena.

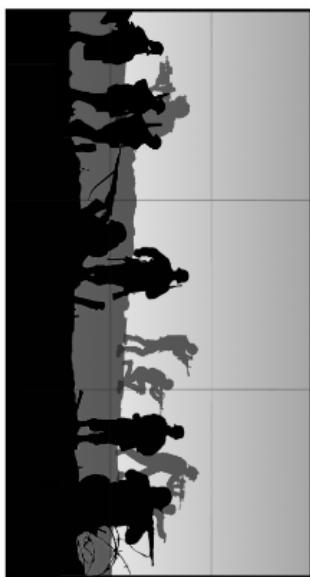
PRODUCTION: Scrap Coder

ASPECT RATIO:

DATE: 04/11/2021

PAGE: 1

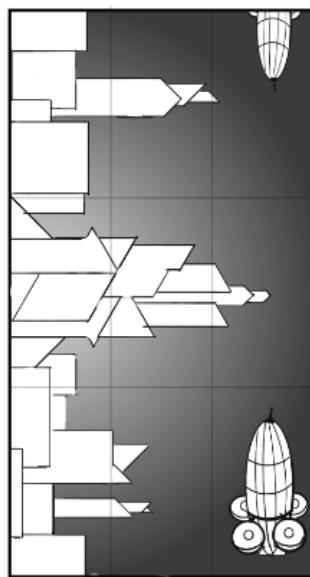
Storyboard



Scene: 1

Shot: 4

Ciudad futurista con aero-naves sobrevolando.



Scene: 1

Shot: 1

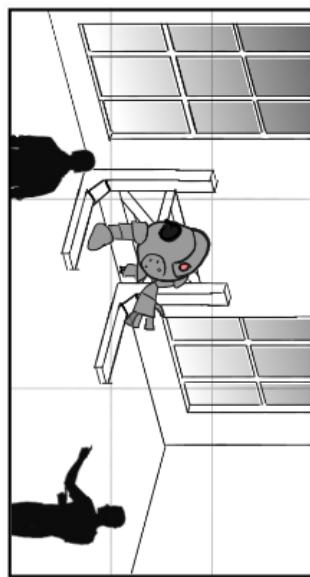
Ciudad futurista con aero-naves sobrevolando



Scene: 1

Shot: 5

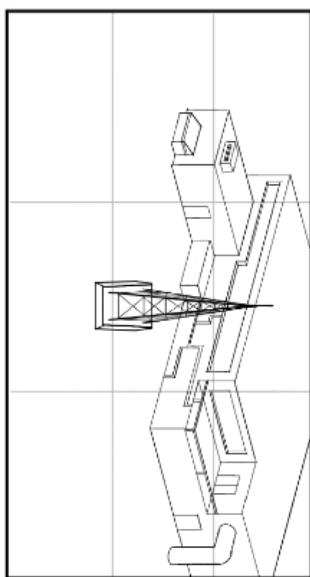
De un robot. La luz roja del robot se va encendiendo con el tiempo.



Scene: 1

Shot: 2

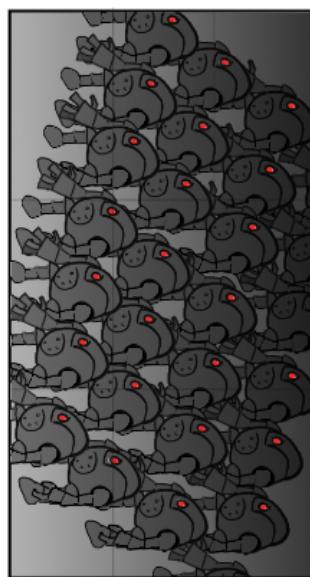
Laboratorio con científicos trabajando en la construcción de un robot. La luz roja del robot se va encendiendo con el tiempo.



Scene: 1

Shot: 6

Ciudad destruida soltando humo. Las aero-naves pierden altura y se desploman contra los edificios.



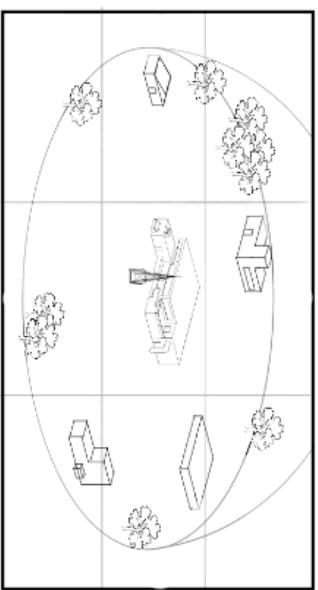
Scene: 1

Shot: 3

Filas de ejército de robots en la oscuridad con sus rojas luces mostrando sus números.

Scene: 1

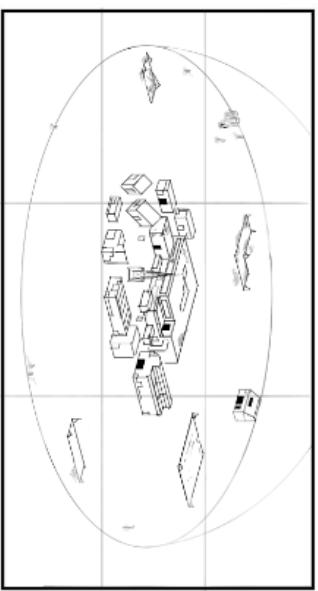
Shot: 7



Campo invisible que emana la antena que los proteje de la amenaza robot.

Scene: 1

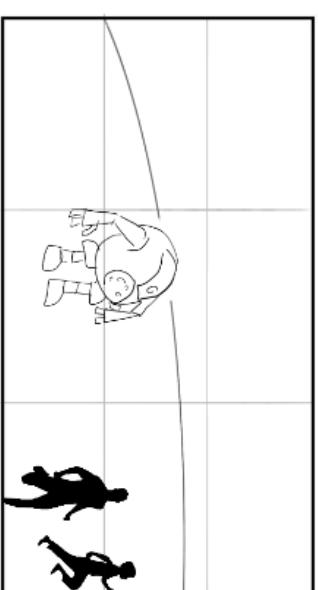
Shot: 8



Mismas instalaciones muchos años en el futuro, los De los alrededores fueron destruidos para construir casas más cerca del edificio principal.

Scene: 1

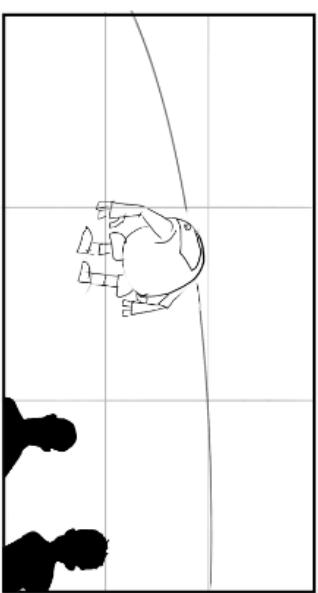
Shot: 9



La antena falla, vulnerando sus defensas y dejando entrar a los robots al asentamiento. Humanos huyen de un robot intruso.

Scene: 1

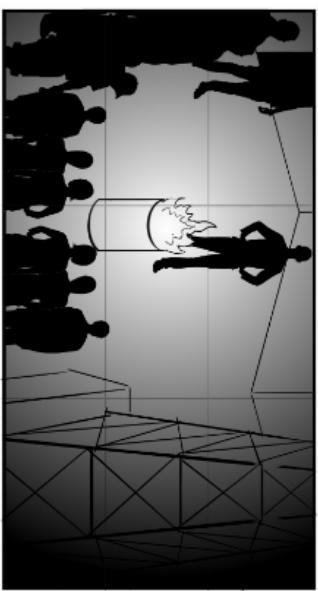
Shot: 10



La antena vuelve a operar correctamente, reactivando el escudo, haciendo que el robot se retire. Los humanos sólo presencian al robot alejarse.

Scene: 1

Shot: 11



Humanos reunidos en el centro del asentamiento con su líder al rededor del fuego.

Mecánicas del juego

Flujo de acciones dentro del juego

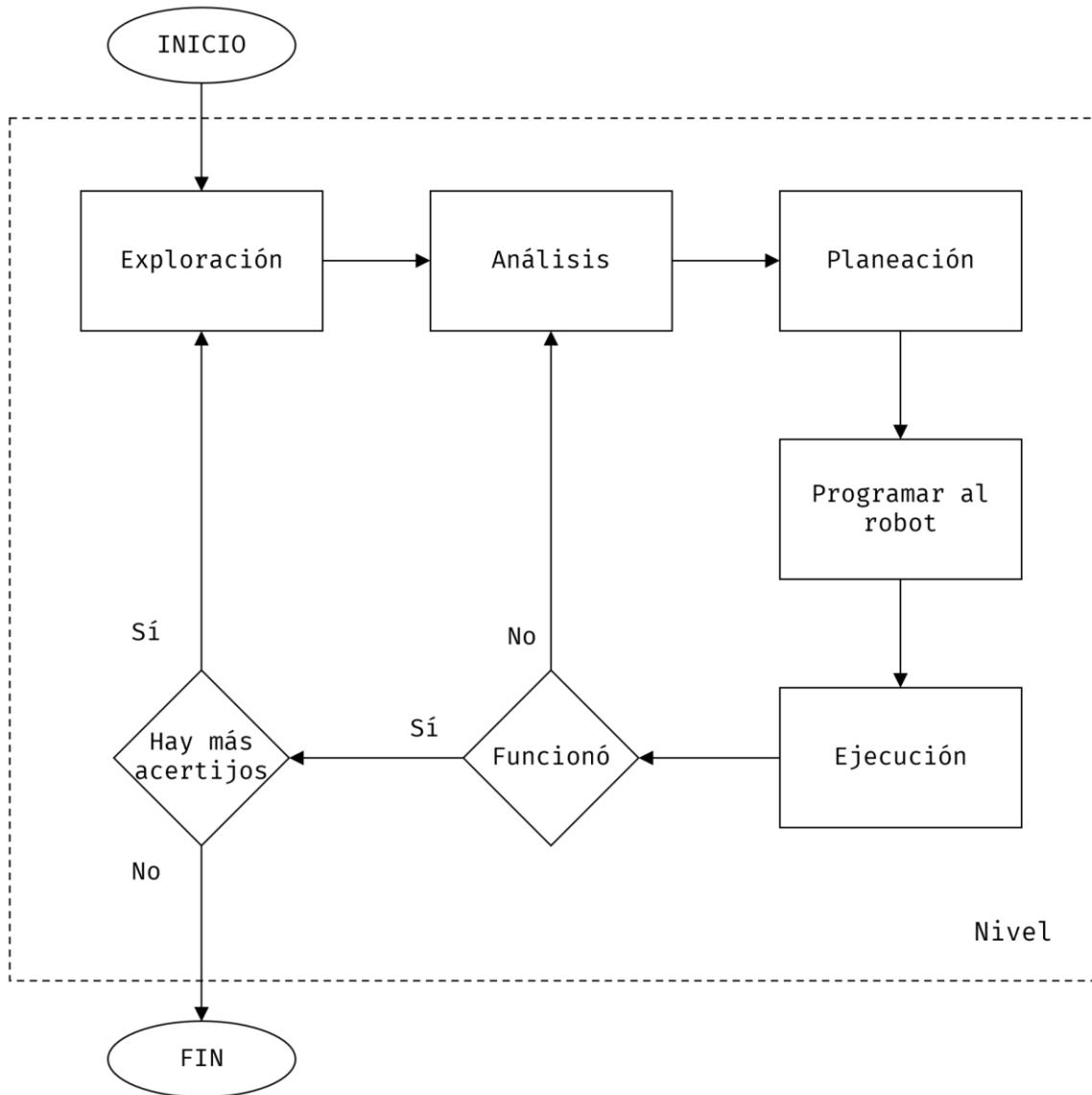


Figura 5.42. Diagrama de flujo de acciones a lo largo del juego.

Descripción de las mecánicas del juego.

Caminar

- El usuario podrá moverse en las cuatro direcciones del espacio 2D. Es decir, arriba, abajo, a la derecha y a la izquierda.
- El usuario también podrá realizar la combinación de dos movimientos en dos ejes distintos a la vez, es decir, podrá moverse en diagonal como sigue: arriba-derecha, arriba-izquierda, abajo-derecha, abajo-izquierda.
- La acción de caminar debe de ser detenida al entrar a un área de colisión o al interactuar con ciertos objetos, así como al entrar al modo Programar el robot.

Mover objetos

- El usuario podrá mover objetos utilizando la mecánica de Caminar.
- Se podrán mover objetos en una dirección a lo largo de los 2 ejes, es decir, arriba, abajo, a la derecha y a la izquierda.
- Se podrán mover objetos en diagonal utilizando el movimiento en diagonal.
- Los objetos serán empujados, por lo que el jugador tendrá que colocarse en el lado opuesto del objeto hacia respecto a donde lo quiere mover.

Tomar objetos

- El jugador podrá cargar objetos y dejarlos de cargar.
- No todos los objetos que puede cargar se podrán mover, así como no todos los objetos que podrá mover se podrán cargar.

Guardar objetos

- El jugador guardará objetos en su inventario.
- El jugador tendrá que acercarse al objeto y presionar alguna tecla para poder guardarlo.

Programar el robot

- Para acceder a la programación del robot, el jugador tendrá que acercarse físicamente al robot.
- Mientras esté en el modo de Programar al robot, el jugador no podrá moverse.
- El jugador tendrá acceso a una interfaz de usuario que se sobrepondrá a la escena actual y mostrará tres secciones, una donde estarán las categorías de los bloques actualmente desbloqueados, los bloques en sí y un área de trabajo donde se podrán arrastrar los bloques para crear las rutinas de programación del robot.

Botón

- Los botones se encontrarán en el suelo y serán plataformas que pueden ser presionadas por otros objetos móviles, incluso por el jugador.
- Activarán una señal que puede ser utilizada para activar otros mecanismos.

Palanca

- Las palancas tendrán dos estados, uno activado y otro desactivado.
- Mientras las palancas estén en modo activado, activarán una señal para activar otros mecanismos.

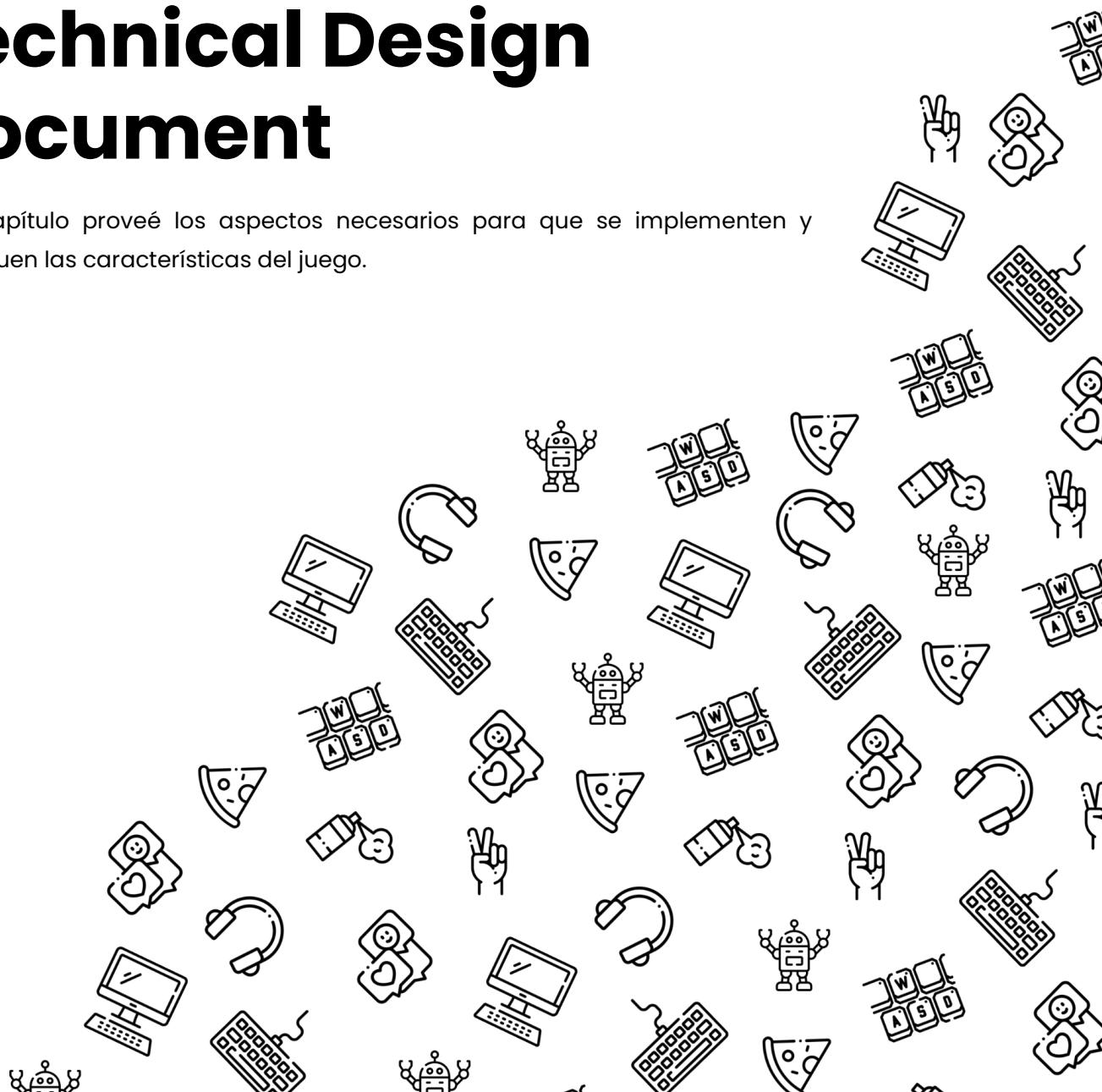
Puerta

- Las puertas son barreras físicas que impiden al jugador y al robot pasar hacia otra sección.
- Pueden ser abiertas utilizando una señal de activación.

6

Technical Design Document

Este capítulo proveé los aspectos necesarios para que se implementen y codifiquen las características del juego.



Herramientas a utilizar

Motor de videojuego: Unity

Unity se define como una plataforma de desarrollo en tiempo real para desarrollar videojuegos, producción de animaciones y filmes, y creación de experiencias inmersivas en 3D para la industria y otros campos profesionales [42].

La aplicación principal de esta plataforma, es decir, el llamado **editor**, es un motor multiplataforma creado por Unity Technologies disponible para sistemas operativos móviles, sistemas de escritorio, consolas, web, y otros [43].

Unity ofrece varios planes gratuitos para la utilización del editor, un **plan para estudiantes de institutos asociados** que permite el uso de todas las características profesionales, y un **plan personal**, que es el que se ha seleccionado para el desarrollo de este proyecto [44].

El único requisito para ser apto para el uso del plan personal es que los ingresos sean menos de 100 mil dólares estadounidenses en los últimos 12 meses previos al registro del plan, y ofrece las siguientes características:

- La última versión núcleo de la plataforma de desarrollo Unity.
- Recursos como introducción y aprendizaje al editor de Unity.

Se decidió utilizar Unity como plataforma de desarrollo ya que es una herramienta gratuita pero con muchas características profesionales que ha sido utilizada en varios videojuegos populares, como Rust, My Friend Pedro, Ori And The blind Forest y Cuphead, como nos indica Drake [45].

Además, su curva de aprendizaje es menor y existen más recursos de aprendizaje disponibles, a comparación de otras alternativas, como Unreal Engine.

Requerimientos mínimos	Windows
Versión del sistema operativo	Windows 7 (SP1+), 10, 11
CPU	Arquitectura x64 con soporte para conjunto de instrucciones SSE2
API gráfica	Gráficas con soporte para DX10, DX11 y DX12

Tabla 6.1. Requisitos mínimos del editor de Unity.

También escogimos Unity debido a que no requiere grandes recursos computacionales para funcionar, como se puede observar en la **Tabla 6.1**, aunque esto va a depender al final de la complejidad de este proyecto.

El editor de Unity utiliza por defecto el IDE Visual Studio, un entorno de programación profesional y empresarial desarrollado por Microsoft, para la edición del código del proyecto.

Aseprite

Aseprite es un programa para crear sprites (gráfico, mapa de bits) animados. Sus características principales, indicadas por su desarrollador Capello [46], son las siguientes:

- Los sprites están compuestos por capas y marcos como conceptos separados.
- Soporta la configuración de diferentes perfiles y modos de color.
- Animación con previsualización en tiempo real y en modo *capas de cebolla*.
- Importación y guardado de imágenes / sprites de y en diferentes formatos.
- Herramientas específicas para la creación de Pixel Art.
- Modo baldosas (tiles) para el dibujo de patrones y texturas.
- Automatización de tareas con una CLI integrada.
- Personalización en el entorno de trabajo.
- Entre otras características.

Además Aseprite es Open Source, cuyo código fuente está bajo la licencia EULA, y el ejecutable es distribuido con tres tipos de licencia, una **licencia educativa** enfocada en profesores e instituciones educativas, una **licencia de prueba** que no permite guardar los archivos producidos y una **licencia de pago** publicada en Steam, con un precio de \$139.49 pesos mexicanos [47].

Este programa fue escogido para crear la mayoría del arte del juego al ser este estilo Pixel Art, además de todas las características que se han mencionado antes y que son útiles para la producción de este tipo de gráficos. También sus requisitos mínimos son básicos, como se puede observar en la **Tabla 6.2**.

Requerimientos mínimos	Windows
Versión del sistema operativo	Windows Vista, 7, 8, 10, 11
RAM	128 MB
Almacenamiento	40 MB

Tabla 6.2. Requisitos mínimos de Aseprite.

Visual Studio

Visual Studio es un entorno de desarrollo (IDE) de Microsoft para programar, depurar, probar e implementar soluciones para muchas plataformas. Está disponible para Windows y sistemas operativos Mac.

Ofrece muchos conjuntos de herramientas enfocados a diversos flujos de trabajo y plataformas profesionales y empresariales, como lo son la Web y la nube, el escritorio y sistemas móviles, juegos, procesamiento y análisis de datos, entre otras [48].

Comparándolo con Visual Studio Code, "Visual Studio no es tan rápido en su ejecución ni mucho menos tan liviano. Esto debido a la cantidad de elementos y funcionalidades con las que cuenta, que lo hacen mucho más pesado, según las necesidades de cada usuario puede llegar a requerir de 2.3 hasta poco más de 60 GB de espacio en disco", según nos indica Mozko [49].

Aún así, utilizaremos a Visual Studio como editor opcional ya que es el predeterminado por Unity para programar los proyectos desarrollados en este.

Microsoft ofrece tres licencias para el uso de este software, la primera de ellas es la **licencia empresarial**, la cual ofrece más características relacionadas a la depuración, diagnóstico avanzado y pruebas [50].

La **licencia profesional**, que es de costo y la **licencia comunitaria**, que es gratuita, ofrecen lo mismo exceptuando sus escenarios de uso admitido, ya que la licencia gratuita solo puede ser utilizada por organizaciones no empresariales con un máximo de 5 usuarios, requisito que es cubierto por nuestro proyecto [50].

Visual Studio Code

Visual Studio Code (VS Code) es un editor de código gratuito, de código libre y multiplataforma desarrollado por Microsoft [51] basado en Electron JS.

Ofrece varias características integradas como IntelliSense, el cual es un servidor que ofrece coloreado de sintaxis y autocompletado inteligente basado en los tipos de variables, funciones, definiciones y módulos importados; además los comandos del sistema de control de versiones GIT están integrados en la interfaz de VS Code, permitiendo revisar diferencias, archivos en espera, y realizar *commits* desde el editor.

También ofrece un debugger (depurador) de código integrado en el propio editor con breakpoints (puntos de ruptura), llamadas a la pila y una consola interactiva.

Es altamente personalizable al tener soporte para extensiones que agregan más características al editor como soporte para nuevos lenguajes, temas, depuradores y conexión con servicios adicionales, todo esto en diferentes procesos, permitiendo así al programador adaptarlo a sus necesidades.

Fue escogido también para la edición del código por ser una herramienta muy ligera a comparación de Visual Studio, consumiendo menos recursos y permitiendo una edición más rápida.

Paint .NET

Es un software de edición de imágenes y fotografías para sistemas operativos Windows. Contiene características que se encuentran en otros programas profesionales de pago de edición de fotografía, algunas de las cuales son:

- Soporte para capas.
- Historial ilimitado.
- Efectos especiales.
- Variedad de herramientas de edición.

Comenzó su desarrollo como un proyecto de diseño universitario supervisado por Microsoft, y actualmente es mantenido por Rick Brewster [52]. Originalmente planteado como el reemplazo para Microsoft Paint, el proyecto ha crecido para ser una herramienta poderosa comparable con otros softwares comerciales.

Este programa se ha escogido como opción para la producción de arte en pixel art, ya que, a pesar de no estar enfocado en la producción de este tipo de arte, este programa permite producir sprites estáticos gracias a sus características.

Plataforma de desarrollo

La plataforma de desarrollo que se sugiere para el desarrollo del proyecto debe de cumplir con requisitos de hardware más altos en comparación con los requisitos para el producto final.

Respecto al software, el sistema operativo donde se va a desarrollar el videojuego será Windows 10, por lo que se utilizará la versión del editor de Unity para este sistema operativo. La versión escogida del editor es la **versión 2020.3.20f1**.

Respecto al hardware, el equipo de trabajo ocupará sus equipos personales ya que cumplen y sobrepasan los requisitos mínimos que se esperan para el videojuego.

En específico se contará con dos equipos de escritorio, en las tablas **6.3** y **6.4** se describen sus características importantes.

Hardware	Descripción
Unidad gráfica de procesamiento (GPU)	Nvidia GTX 1060 con 3GB GDDR5
Unidad central de procesamiento (CPU)	AMD Ryzen 2200 G @3.5 GHz con gráficos integrados Radeon Vega Graphics
Memoria de acceso aleatorio (RAM)	16 GB DDR4

Tabla 6.3. Hardware del equipo 1.

Hardware	Descripción
Unidad gráfica de procesamiento (GPU)	Nvidia GTX 1660 Super de Gigabyte con 6GB GDDR6
Unidad central de procesamiento (CPU)	AMD Ryzen 6 3600 @3.9 GHz con 6 núcleos / 12 hilos
Memoria de acceso aleatorio (RAM)	32 GB DDR4 @3000 MHz

Tabla 6.4. Hardware del equipo 2.

Metodología

La metodología de desarrollo de software *Xtreme Programming* (XP) ha demostrado ser exitosa para el desarrollo de videojuegos a nivel mundial [53]; por lo que utilizaremos una adaptación de esta misma para el desarrollo del videojuego, ya que permitirá tener funciones y módulos en menor tiempo al mismo tiempo que se realizan modificaciones en el código ya existente.

Esta metodología nos permitirá diseñar y desarrollar el videojuego en pequeños incrementos modificables cuando ciertos requisitos cambien o se observe que las soluciones implementadas no sean las mejores.

La siguiente figura muestra el proceso que se va a seguir para desarrollar el sistema:

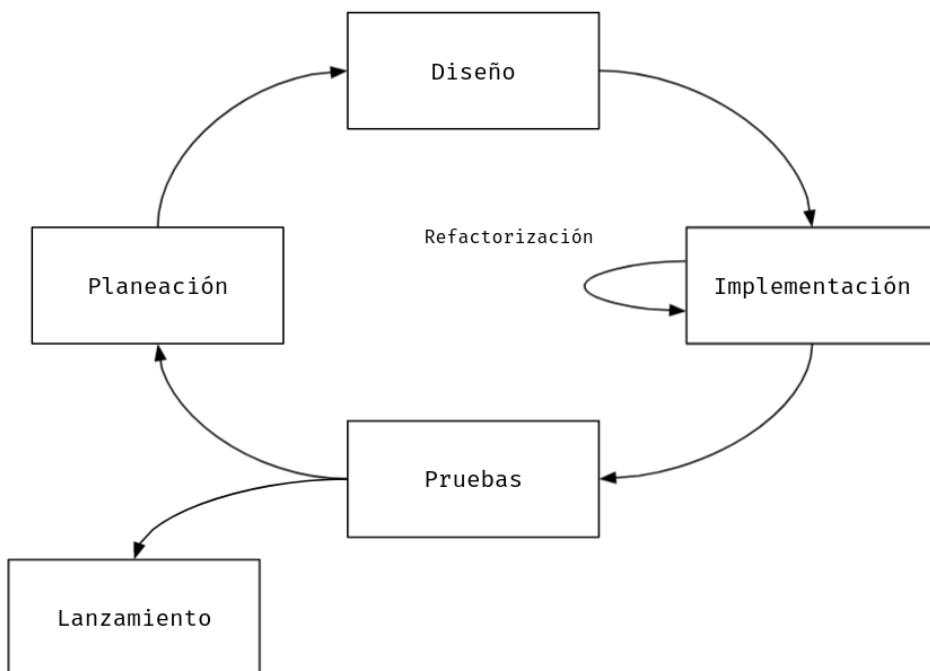


Figura 6.1. Diagrama general de la metodología.

Cronograma de actividades

A continuación se muestra un cronograma de actividades general para el desarrollo del videojuego respecto al periodo de desarrollo 2021 - 2022.

Cada integrante del equipo de desarrollo contribuirá a cada una de las actividades descritas en el cronograma general.

Actividad	AGO	SEP	OCT	NOV	DIC	ENE	FEB	MAR	ABR	MAY	JUN
Documentación inicial											
Personajes e ítems											
Mecánicas											
Interfaz											
Entorno											
Niveles											
Intérprete											
Documentación final											
Evaluación											

Tabla 6.5. Cronograma general de actividades.

Arte y gráficos

El juego va a utilizar la técnica Pixel Art como su estilo artístico principal, por lo que los gráficos y la configuración de los diversos sprites e imágenes dentro el editor Unity tienen que ser configuradas de forma especial utilizando algunas herramientas especializadas que son ofrecidas por el mismo Unity.

- Los escenarios del juego estarán construidos a base de un sistema en cuadrícula (*grid*).
- Cada celda de la cuadrícula tendrá un tamaño de 24 x 24 píxeles.
- El juego tendrá una interfaz gráfica de usuario (GUI) también con Pixel Art, por lo que esta necesita ser renderizada *en línea* con el resto de gráficos presentados en el juego.

Resolución y relación de aspecto

Se estableció que la resolución objetivo para el juego será de **480 x 270** píxeles, esto con el objetivo de conservar una baja resolución para monitores 1080p (**1920 x 1080** píxeles) y manteniendo su relación de aspecto de **16:9**.

Esto significa que cada pixel en el juego será representado con bloques de 4 x 4 píxeles en monitores de resolución 1080p o Full HD y con bloques 8 x 8 píxeles en monitores 4k o Ultra HD. Con esta configuración, habrá 20 celdas de 24x24 a lo ancho y 11.25 a lo alto, como se puede observar en la **Figura 6.2**.

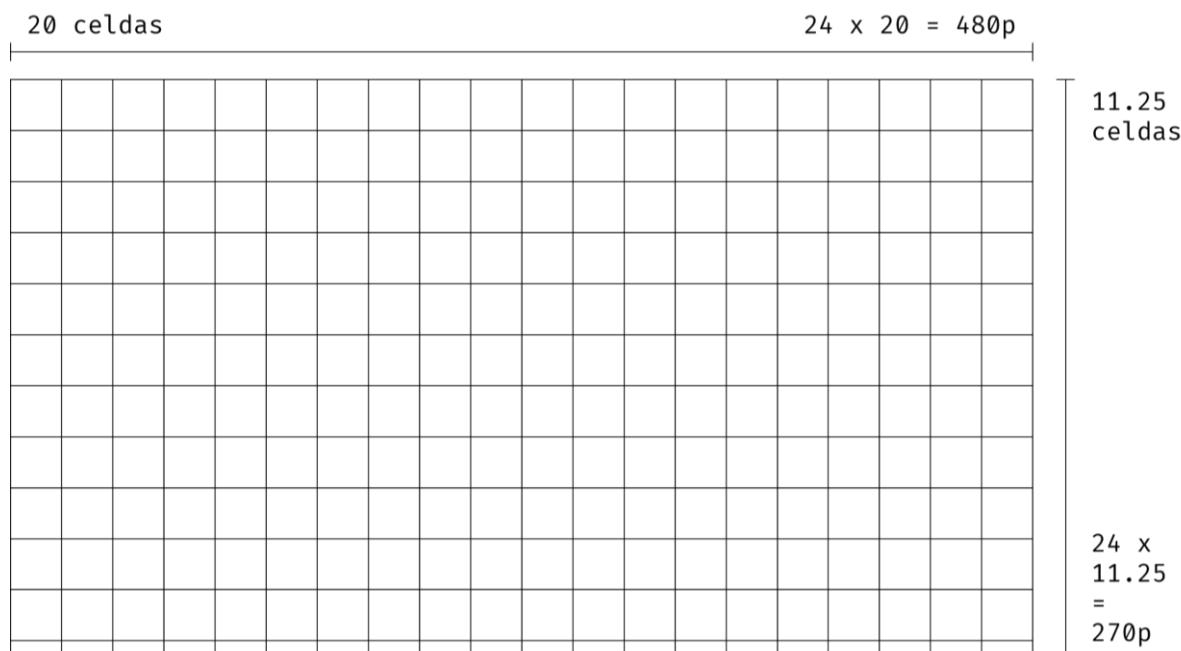


Figura 6.2. Configuración de celdas

Con esta configuración establecida, podemos asegurar que el proceso en el proceso de exportación de los sprites y otros assets no habrá pérdida de calidad.

Configuración de los sprites

Cada sprite utilizado en el proyecto tendrá la siguiente configuración:

- Píxeles por unidad: 24
- Modo de filtro: Point
- Compresión: None
- Configurar el tamaño máximo mayor a la resolución del atlas del sprite.
- Configurar un pivote en píxeles.

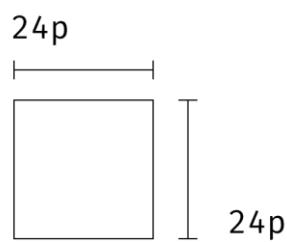
Configuración del grid

En la configuración de *Increment Snap*, en los campos de *Move*, configurar el valor a 1 / 24. Además, en el *Tilemap* configurar el tamaño de la celda a 24.

Configuración de la cámara

La cámara dentro de Unity es el objeto que se encarga de renderizar los gráficos del juego, por lo que es importante configurarla para que se integre bien con el estilo de arte escogido para el juego y con las configuraciones previamente establecidas.

- Los Píxeles Por Unidad serán de 24.
- *Reference Resolution* será nuestra resolución objetivo, es decir, 480 x 270.
- *Pixel Snapping* como no configurado.



Una celda

Una unidad cuadrada

Figura 6.3. Relación entre celda y unidad.

Interfaz y progreso del juego

Interfaz

Descripción de las diversas pantallas que tendrá el juego en diagramas de alto nivel.

Carga inicial

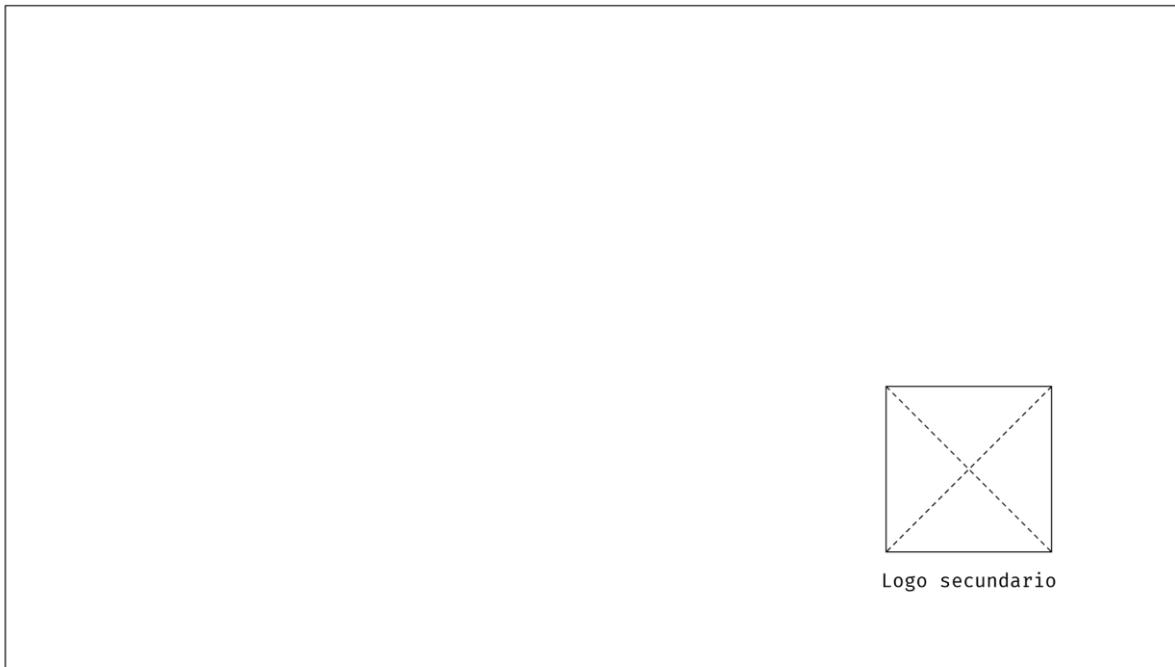


Figura 6.4. Pantalla de carga inicial.

- Carga en memoria de los recursos principales.
- Presenta al jugador un periodo de espera.

Pantalla principal

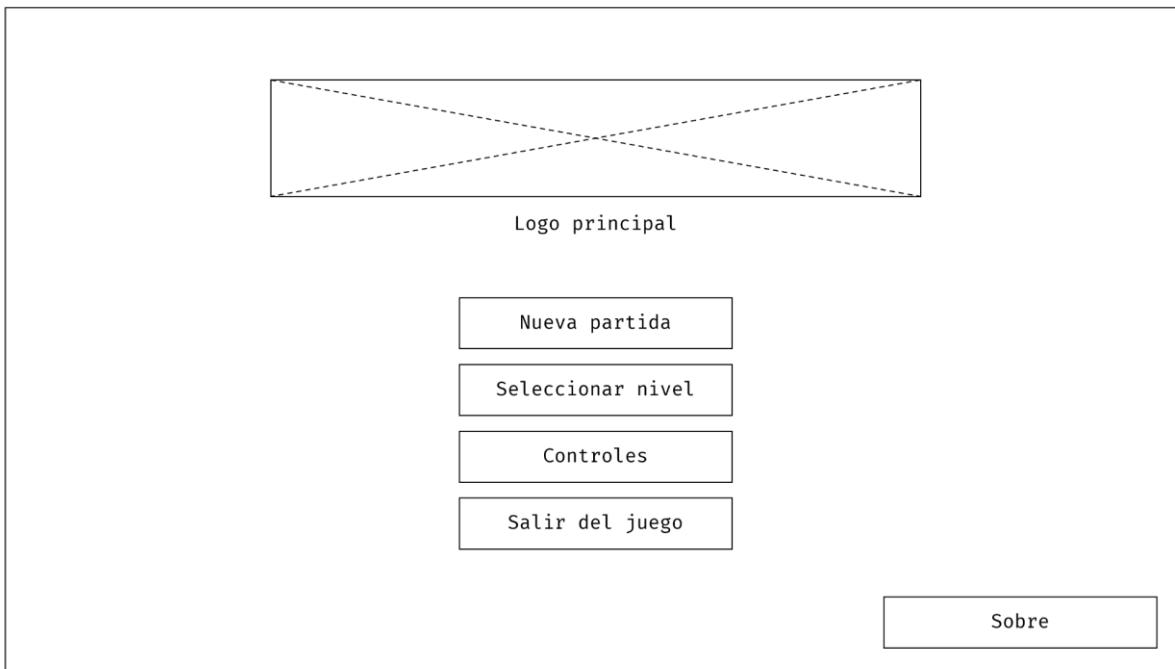


Figura 6.5. Pantalla principal.

- Permite comenzar una nueva partida.
- Permite acceder a la sección de selección de nivel.
- Permite acceder a la sección de Controles.
- Permite salir del juego.
- Permite acceder a la sección de Sobre o Créditos.

Pantalla de selección de niveles

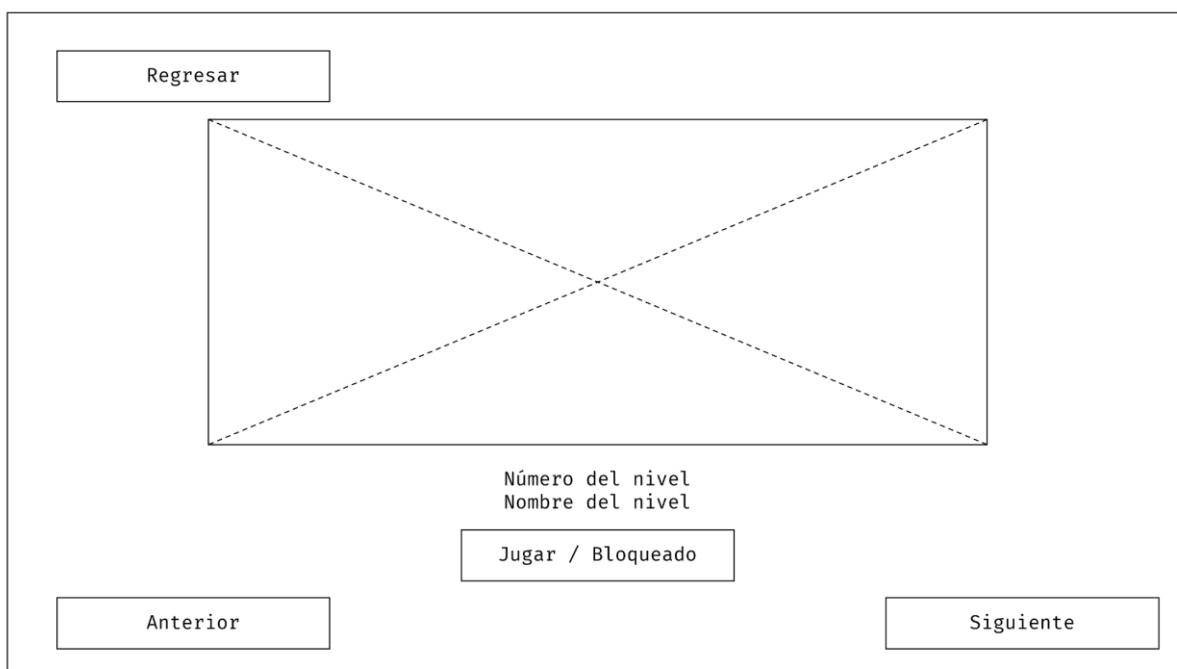


Figura 6.6. Pantalla de selección de niveles.

- Permite seleccionar un nivel desbloqueado.
- Permite observar todos los niveles desbloqueados y bloqueados.
- Permite regresar al menú principal.

Cuadro de confirmación genérico

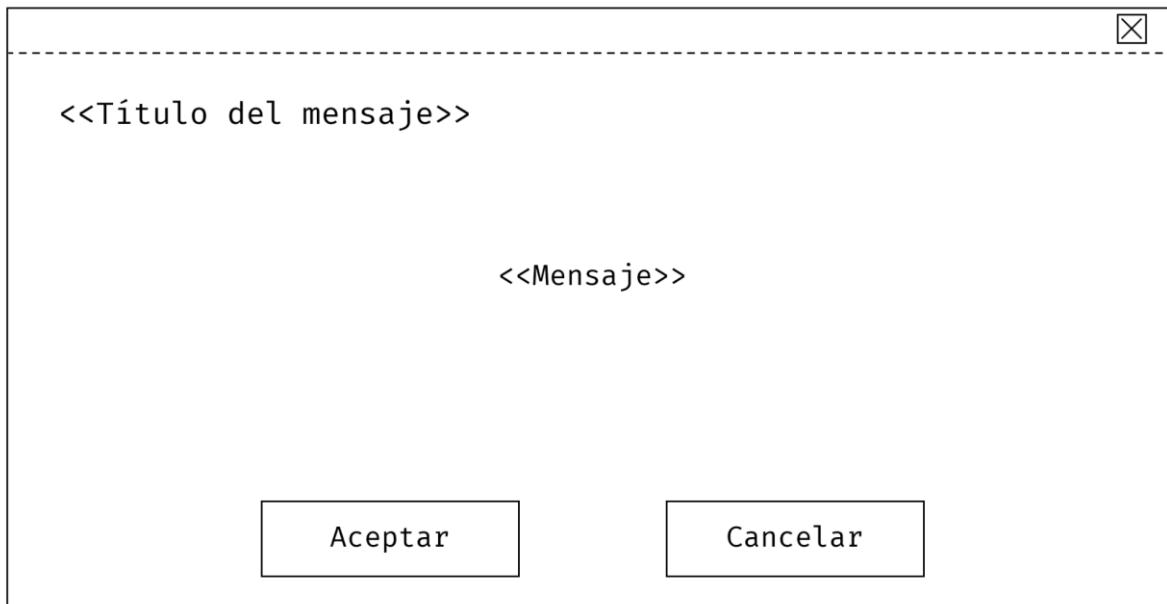


Figura 6.7. Cuadro de confirmación genérico

- Presentan una confirmación a una acción a realizar.
- Devuelven un valor booleano respecto a lo que el usuario decidió.

Cuadro de información genérico

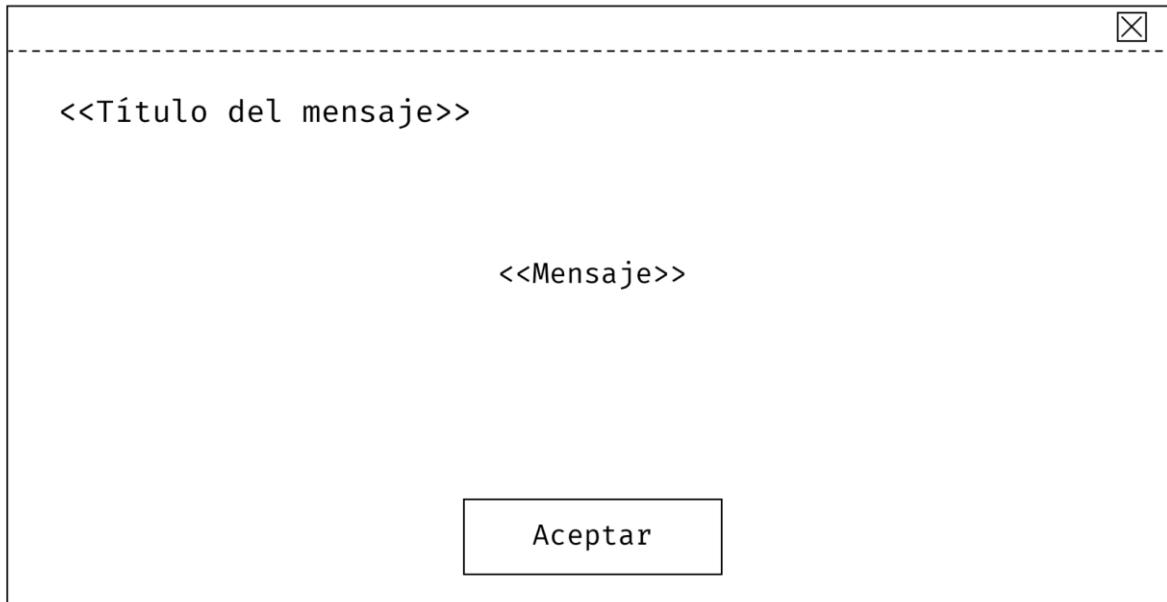


Figura 6.8. Cuadro de información genérico

- Presentan al usuario un mensaje con información.

Interfaz dentro del juego

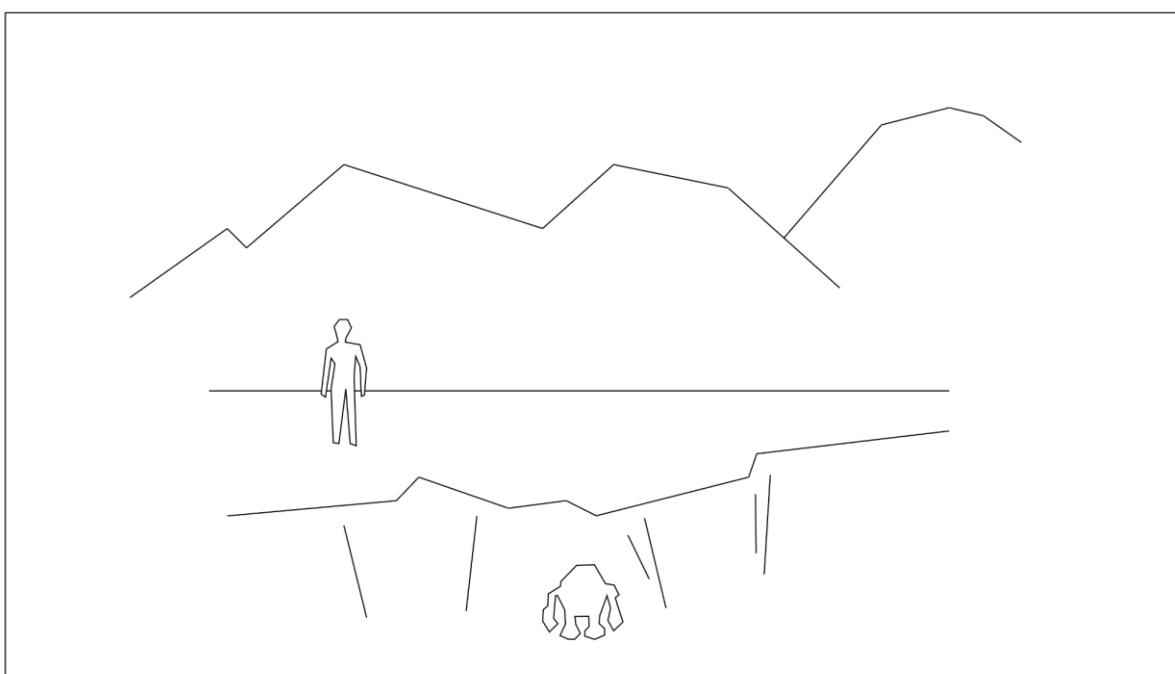


Figura 6.9. Interfaz principal.

- Permite observar el entorno del juego.

Interfaz de programación visual

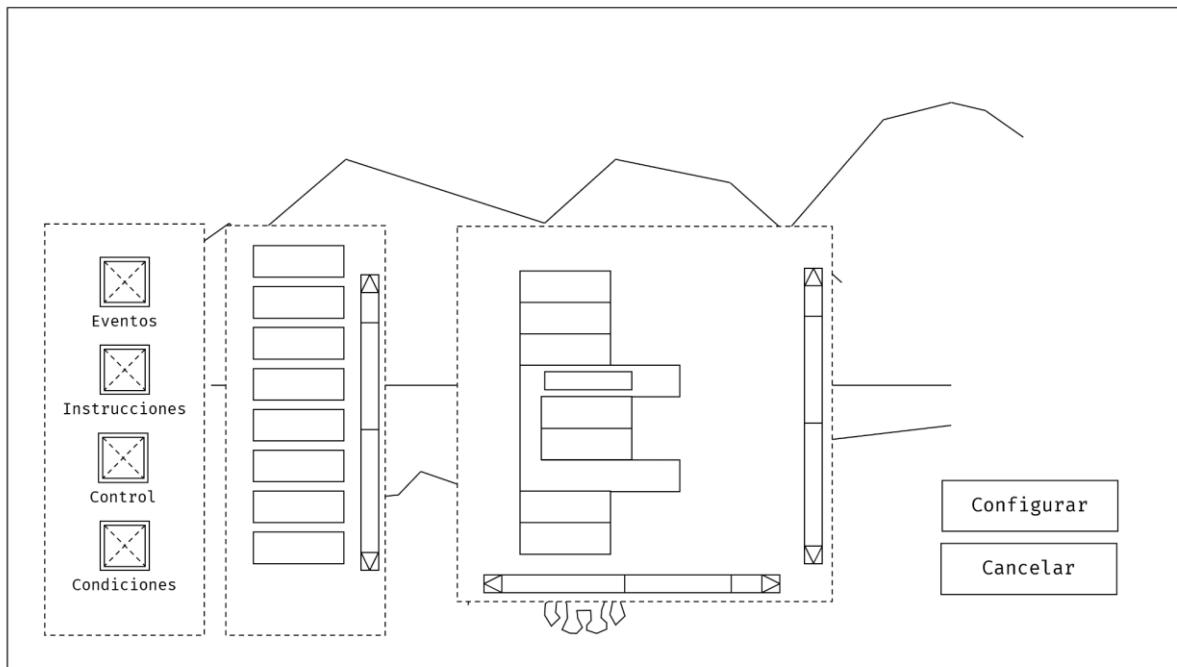


Figura 6.10. Interfaz de programación visual.

- Presenta una sección para selección de bloques de programación.
- Presenta una sección que lista los bloques de cierto tipo.
- Presenta una sección de trabajo para arrastrar los bloques.
- Presenta un botón para configurar lo programado.
- Presenta un botón para cancelar.

Cinemática

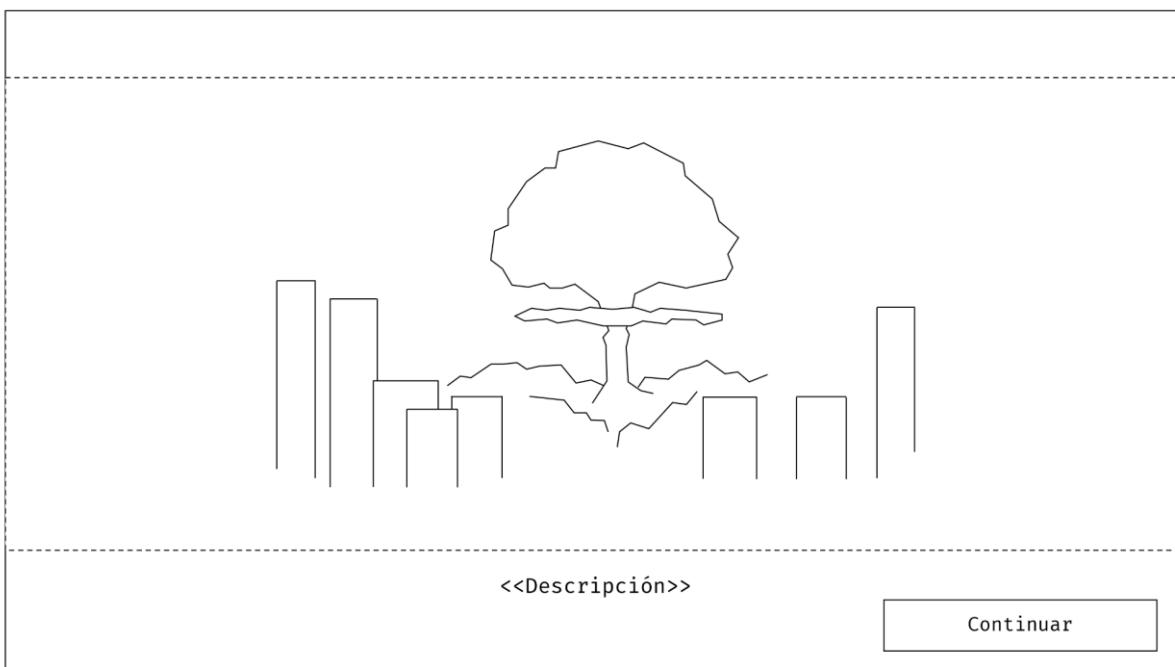


Figura 6.11. Interfaz de cinemática.

- Muestra la cinemática.
- Muestra la descripción de lo que sucede.
- Muestra un botón de continuar.

Flujo de pantallas

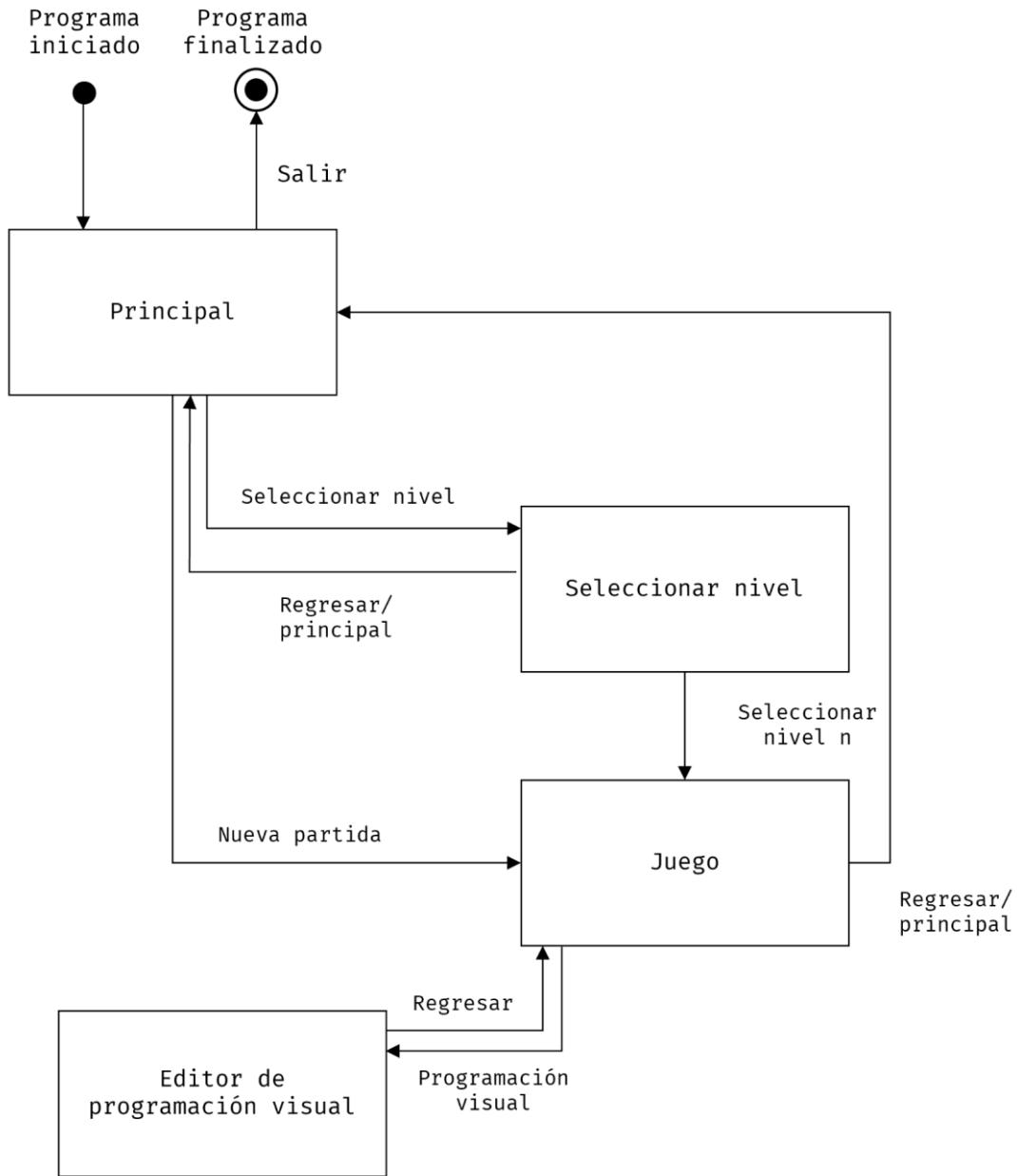


Figura 6.12. Máquina de estados del flujo de pantallas.

Progreso del juego

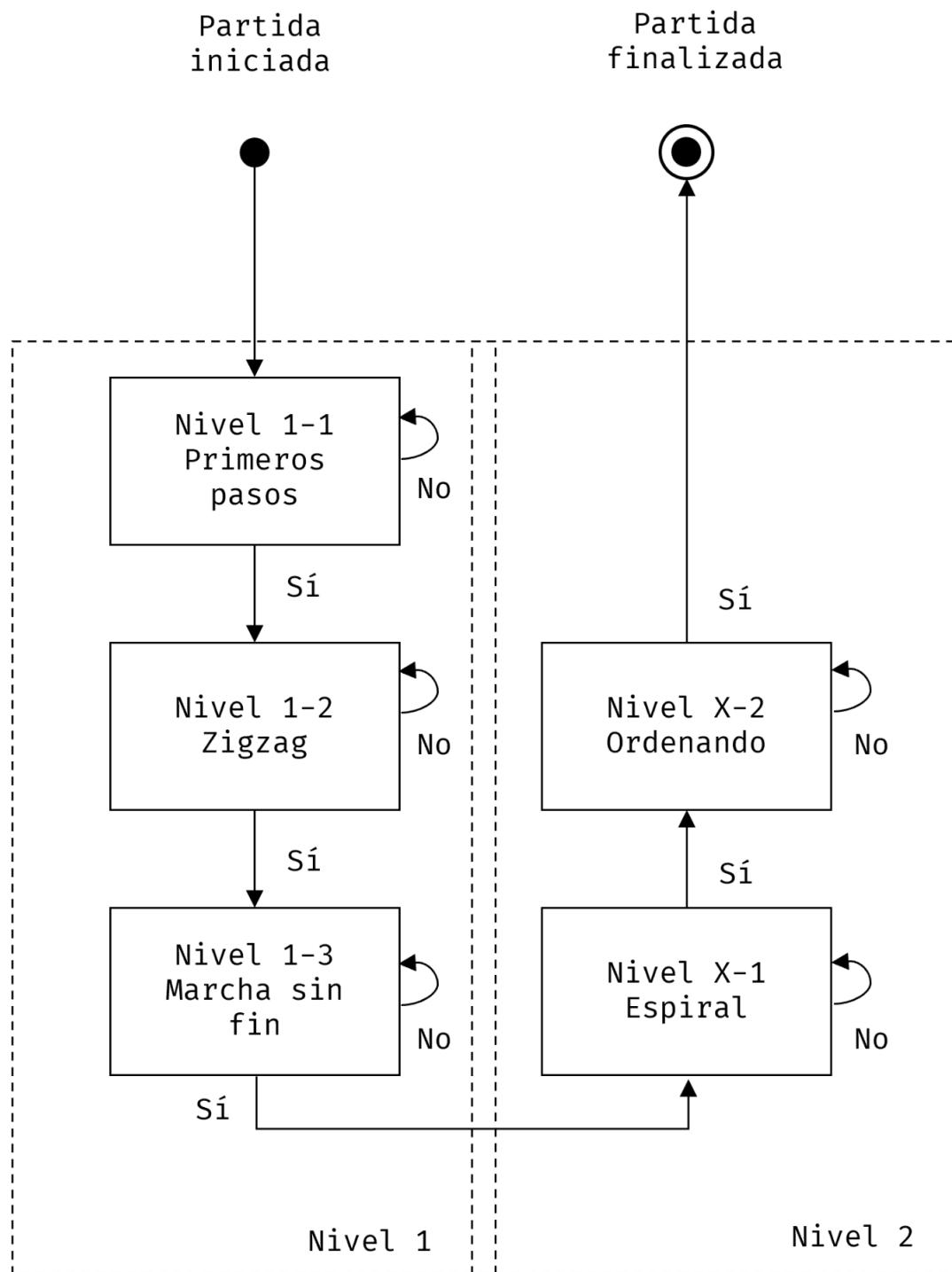


Figura 6.13. Máquina de estados del flujo de niveles.

Físicas, colisiones e interacción

Físicas

- Permite que varias mecánicas del juego sean implementadas.
- Algunos objetos dentro del juego deben de poder ser empujados por el jugador utilizando una fuerza que los impulse.
- Debe ser realista, es decir, los objetos deben comportarse como en la vida real.
- Las físicas serán necesarias para: tomar ítems, soltar items, empujar objetos, objetos arrastrados.
- Se usará el motor de físicas de Unity.

Interacción

- Esto es como el jugador y el robot podrán interactuar con el mundo.
- La interacción es con los siguientes objetos:
 - Puertas.
 - Ítems que se pueden cargar.
 - Objetos empujables.
 - Banda transportadora.
 - Interacción con el robot.

Eventos de Unity relevantes

Unity ofrece una amplia variedad de eventos que son llamados por el propio motor cuando ocurren ciertas condiciones, lo que ofrece una plataforma unificada para la programación de la interacción del jugador con el juego.

Awake

Es el primer método del **ciclo de ejecución** llamado por Unity al cargar una instancia dentro de una escena. Es heredado por la clase principal **MonoBehaviour** y sirve principalmente para inicializar variables o estados antes del resto de comportamientos del objeto.

Start

Es el método del **ciclo de ejecución** que es llamado en el primer *frame* en que un objeto es activado, justo antes de cualquier otro método. Es heredado por la clase principal **MonoBehaviour** y este método es propicio para hacer inicializaciones respecto al objeto cuando el resto de componentes ya se encuentran cargados.

Update

Cada *frame* dentro del juego este método es llamado. Es heredado por la clase principal **MonoBehaviour** y sirve para implementar el comportamiento general que sea necesario en cada *frame*.

FixedUpdate

Es un método que se llama con la frecuencia de actualización de las físicas del motor de Unity. Si se tiene lógica que se quiere ejecutar un determinado número de veces cada segundo, o no se desea que se ejecute con una frecuencia variable, entonces **FixedUpdate** es el método a utilizar.

OnTriggerEnter2D

Es ejecutado cuando otro objeto (entrante) entra dentro de un *collider* que se comporta como *trigger* que compone a otro objeto (receptor). Recibe como parámetro el *collider* del objeto entrante.

OnTriggerExit2D

Es ejecutado cuando otro objeto (saliente) sale del *collider* que se comporta como *trigger* que compone a otro objeto (emisor). Recibe como parámetro el *collider* del objeto que sale.

OnPointerDown

Método descrito en la interfaz **IPointerDownHandler** que es llamado cada vez que el objeto UI es presionado por el usuario.

OnBeginDrag

Método descrito en la interfaz **IBeginDragHandler** que es llamado cuando el objeto UI comienza a ser arrastrado por el usuario.

OnDrag

Método descrito en la interfaz **IDragHandler** que es llamado mientras el objeto UI es arrastrado por el usuario.

OnEndDrag

Método descrito en la interfaz **IEndDragHandler** que es llamado al finalizar el arrastre del objeto UI por el usuario.

Plataforma objetivo

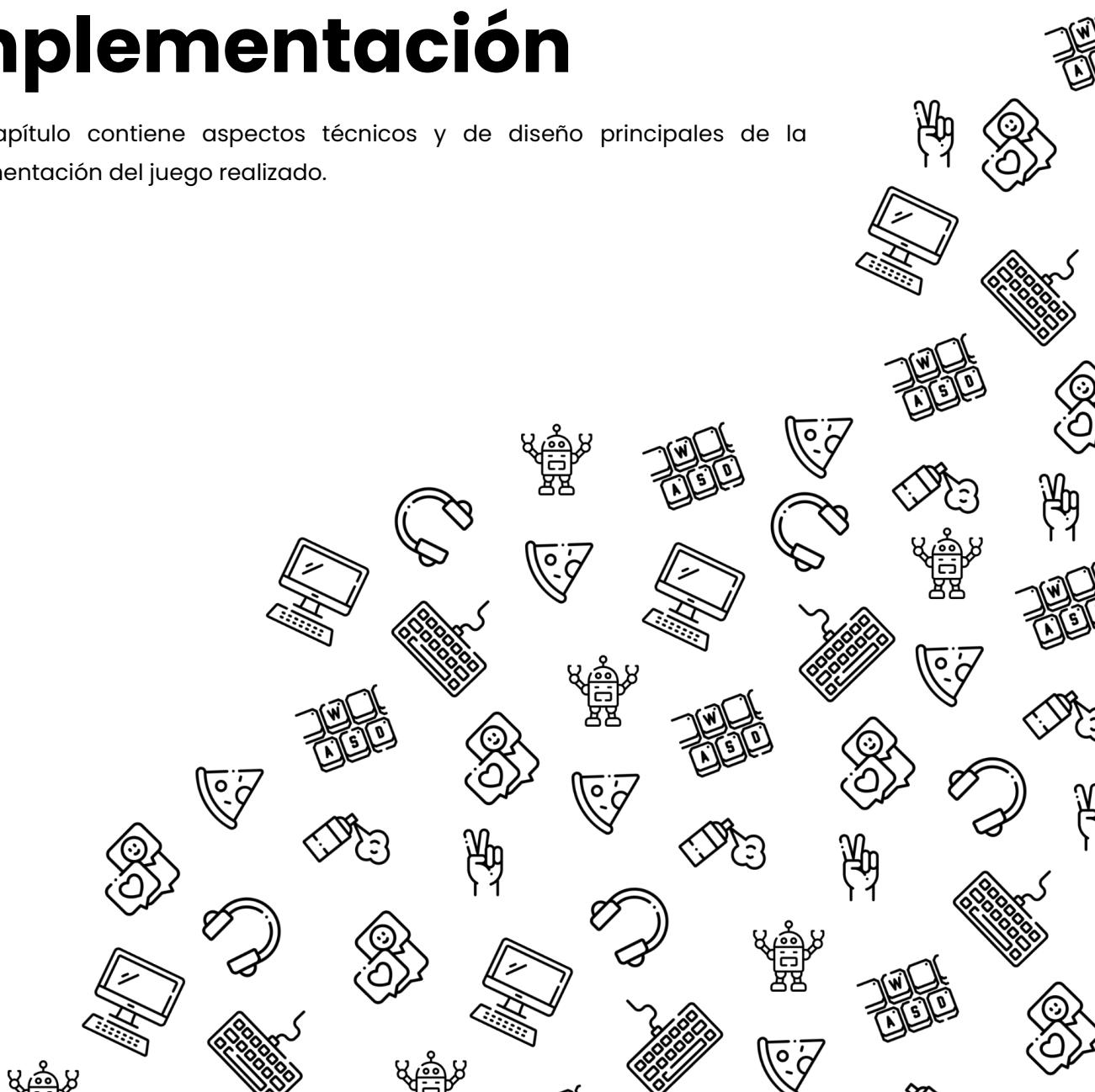
Componente	Descripción
Sistema operativo	Windows 10
Memoria de acceso aleatorio (RAM)	4 GB
Almacenamiento	N/A

Tabla 6.6. Requerimientos de la plataforma objetivo.

7

Implementación

Este capítulo contiene aspectos técnicos y de diseño principales de la implementación del juego realizado.



Programación visual e intérprete

Los nodos, como se ha visto en el diseño, tienen formas irregulares diferentes que necesitan ser dinámicas para poder crecer, ensancharse o achicarse dependiendo de su contenido. Para su renderizado, se optó por utilizar **SpritesShapes**, un paquete de herramientas flexible y poderoso que Unity ofrece para la construcción de formas libres y expandibles [54].

Unity ofrece un conjunto de herramientas llamado **Unity UI** para el desarrollo de interfaces gráficas en el propio motor basado en *GameObjects* que utiliza componentes para el ordenamiento, posicionamiento y estilizado de las interfaces [55] pero que tiene una desventaja respecto a la naturaleza de nuestro proyecto: no tiene soporte *out-the-box* para los **SpritesShapes**.

En lugar de implementar una solución personalizada para la construcción de estas figuras libres requeridas para los nodos y así poder utilizar **Unity UI** (lo cual probablemente hubiera producido un mejor resultado, pero a costa de mayor tiempo de desarrollo), se optó por construir un sistema de interfaz y posicionamiento de elementos personalizado algo rudimentario pero con las suficientes herramientas para las necesidades del proyecto.

Jerarquía y ordenamiento

Se implementó un sistema personalizado de ordenamiento en el eje z para que cada collider, dependiendo de su prioridad, pueda recibir los eventos por parte del usuario, que al final son RayCast que son lanzados por la cámara a través del EventSystem. Además esta jerarquía en el eje z se usó para el ordenamiento gráfico de cada nodo.

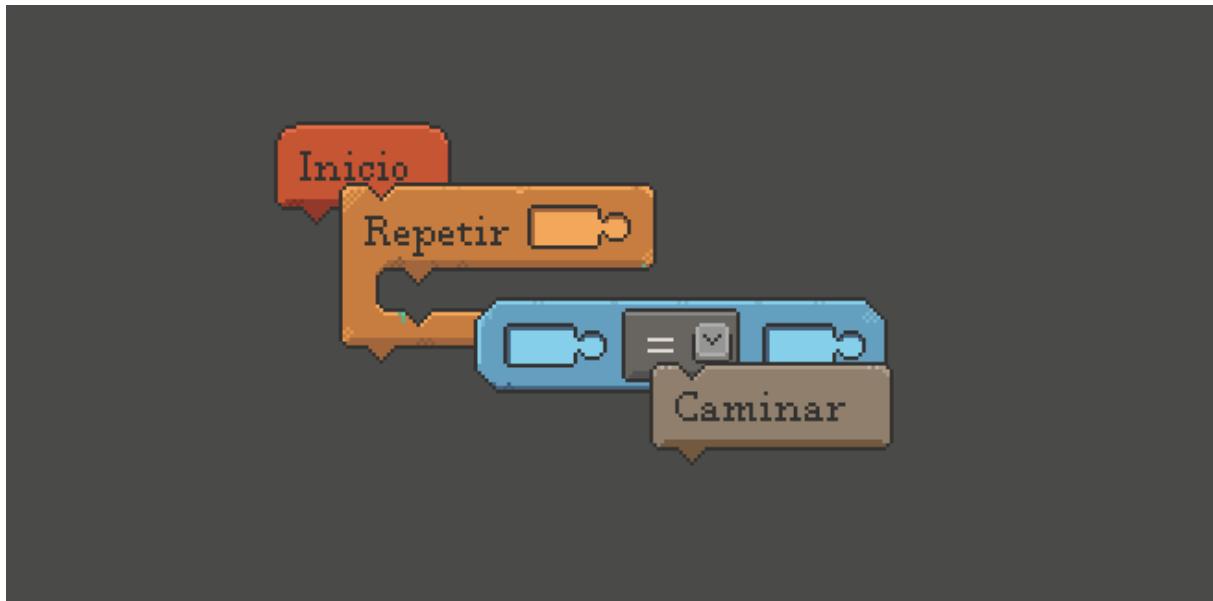


Figura 7.1. Ordenamiento de cada nodo.

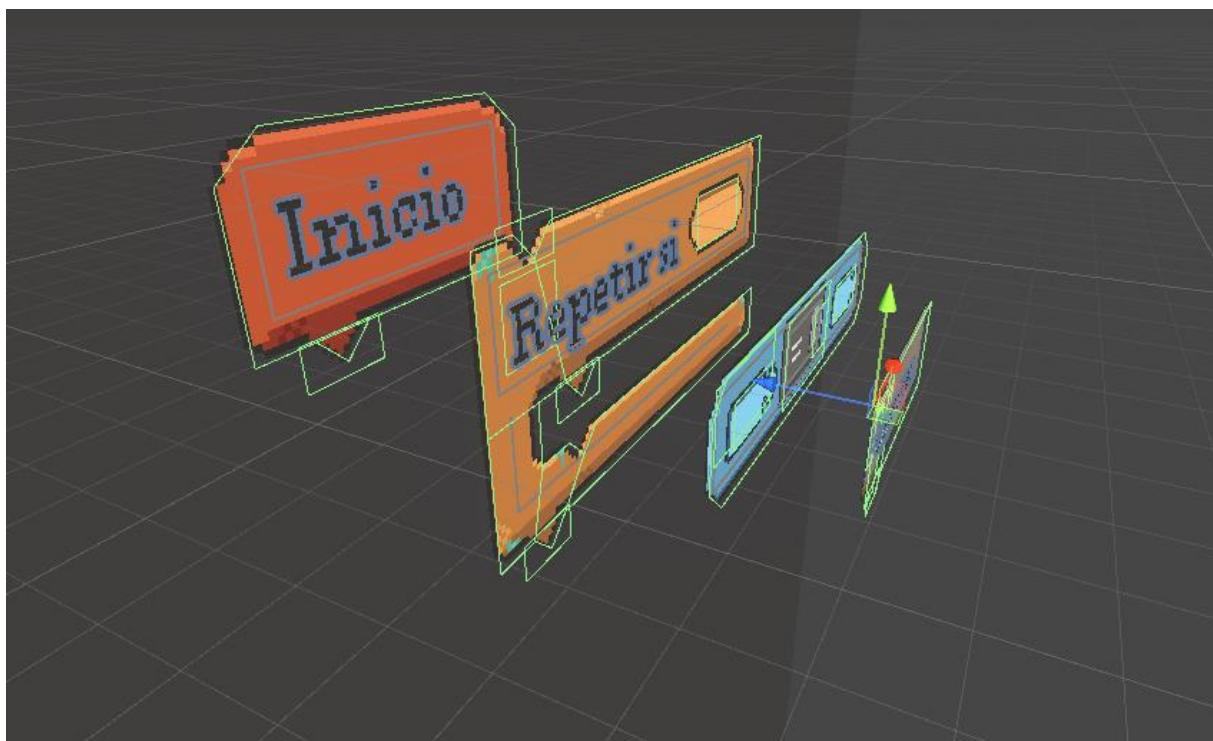


Figura 7.2. Vista en perspectiva del acomodamiento de cada nodo.

El resto de los elementos gráficos de cada interfaz siguen la misma lógica, acomodados por el mismo controlador que define la jerarquía y el orden de cada nodo. Estas clases controladoras son *HierarchyController* que se comunica con las clase *InterfaceCanvas*, siendo cada una de ellas un singleton.

Además, se configuró la influencia de cada collider para que fuera local a la capa de la interfaz UI, y así no existiera interferencia con las demás capas del juego.

	Default	TransparentFX	Ignore Raycast	Water	UI	Entities	DistortionEffect
Default	✓	✓	✓	✓	✓	✓	✓
TransparentFX	✓	✓	✓	✓	✓	✓	✓
Ignore Raycast	✓	✓	✓	✓	✓	✓	✓
Water	✓	✓	✓	✓			
UI				✓			
Entities	✓	✓					
DistortionEffect	✓						

Figura 7.3. Configuración de la matriz de colisión del proyecto.

Nodos

Gráficos

Los gráficos de los nodos están formados por varios conjuntos de **SpritesShapes** junto con varios Sprites simples, que componen los diversos estados que sirven para diversos efectos gráficos, como el evento OnCLick, o cuando se muestra el estado de ejecución.

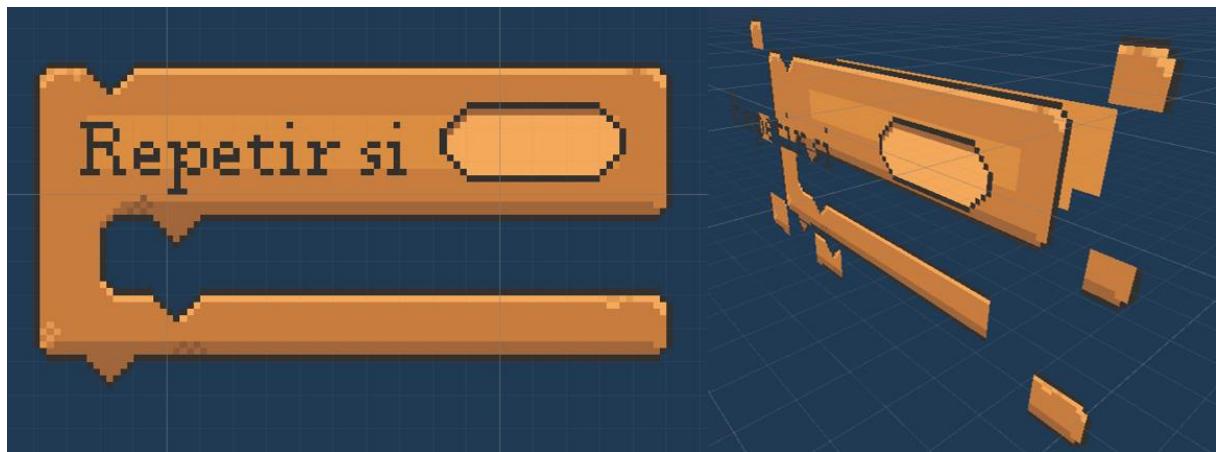


Figura 7.4. Arquitectura de los componentes visuales de un nodo.

Además, para que el EventSystem de Unity hiciera que los nodos ejecuten eventos, se utilizaron múltiples PolygonColliders (mostrados con bordes verdes) adaptados a la forma del nodo y también se utilizaron colliders para las zonas de anclaje.

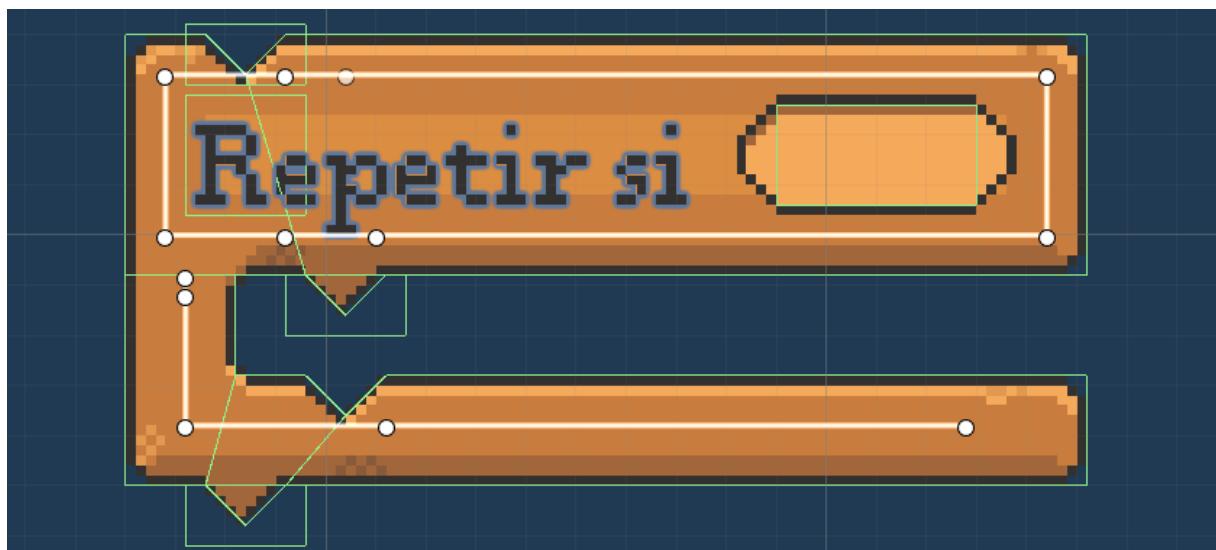


Figura 7.5. Colliders y zonas de anclaje de un nodo.

Cada una de las **SpritesShapes** tienen segmentos programados que insertan variaciones sutiles para dotar de cierta aleatoriedad a las figuras, así como los sprites de cada arista que igualmente tienen variaciones que son aleatoriamente escogidas para cada instancia de nodo.



Figura 7.6. Variaciones de nodos.

Conexión entre nodos

La mecánica principal del proyecto es la programación visual, y los nodos que permiten programar al robot tienen una forma en específico que, visualmente, sólo los permite conectarse en ciertos “huecos” o uniones.

La arquitectura de conexión del nodo es simple:

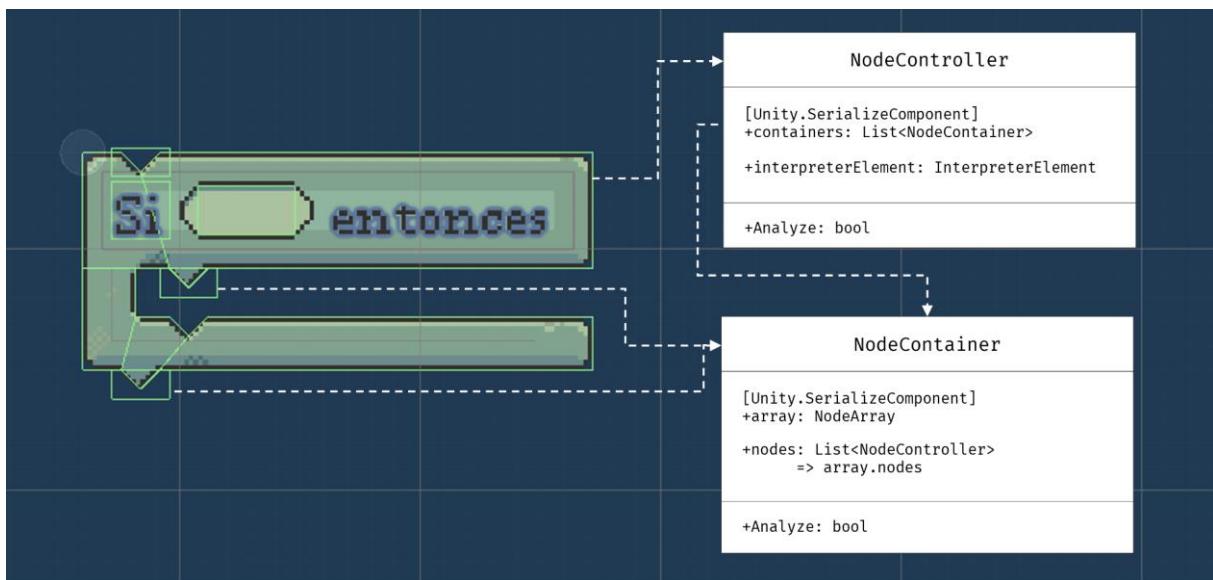


Figura 7.7. Arquitectura de conexión de los nodos y sus clases relacionadas.

Cada nodo tiene una clase principal del tipo *NodeController* como componente, la cual es la encargada de repartir los nodos que se arrastran a las zonas de anclaje a sus respectivos contenedores.

Los nodos tienen múltiples zonas de anclaje, que son objetos contenedores denotados visualmente por un Sprite que representa un hueco o una unión, y contienen jerárquicamente a otros nodos de cierta categoría, lo que forma al árbol de jerarquía.

La conexión entre nodos es de acuerdo a la categoría de nodo, las cuales son las siguientes:

- Condiciones.
- Instrucciones.
- Valores, que también acepta variables.
- Variables.
- Arreglos.

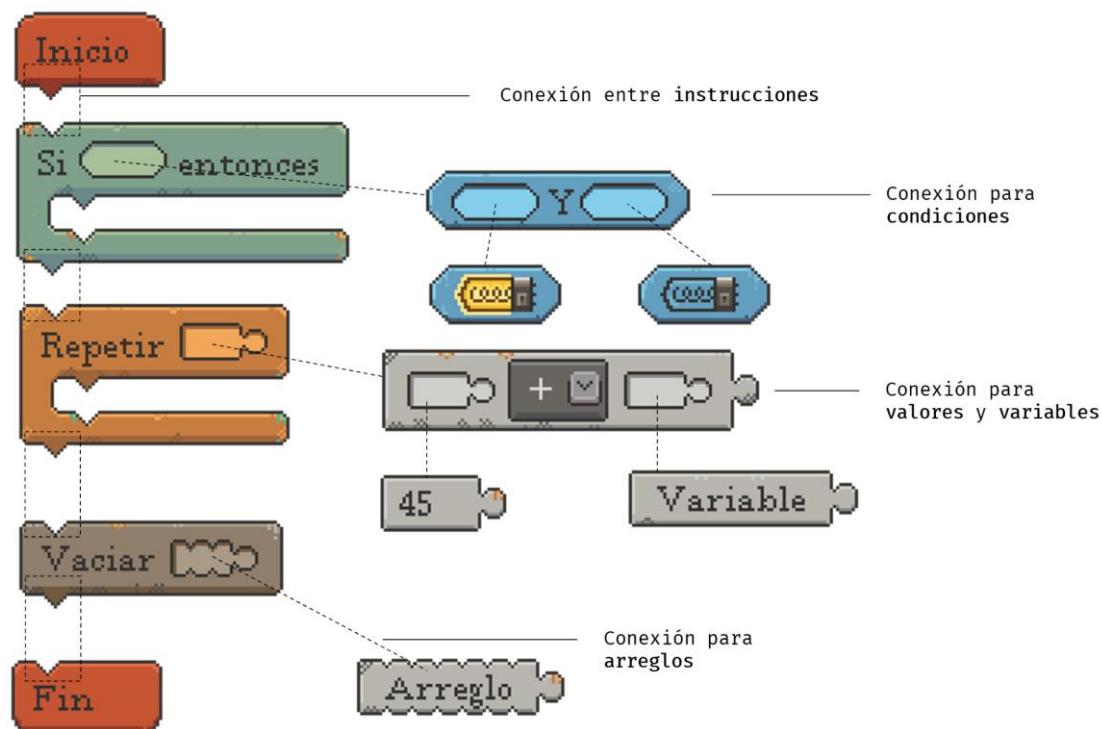


Figura 7.8. Conexión entre categorías de nodos.

Intérprete

El intérprete utiliza la jerarquía creada por el mismo usuario al arrastrar y soltar nodos dentro de los mismos. Los nodos, al poder conectarse solo en lugares específicos de acuerdo a su categoría, permite que la jerarquía se pueda utilizar como un árbol sintáctico con una tokenización implícita y con un análisis léxico muy ligero, dejando la generación de código intermedio como una simple llamada a métodos ya predefinidos.

Cada uno de los nodos tiene un componente llamado *interpreterElement*, que es el comportamiento individual de cada uno de los nodos, y este es llamado cuando el intérprete es ejecutado, y también un método llamado *Analyze()*, el cual se ejecuta en la fase de análisis léxico..

Analizador

Antes de que la clase *Executer*, que se encarga de ejecutar cada una de las instrucciones para que realicen su trabajo, se realiza un análisis muy breve al árbol de nodos que consiste en verificar lo siguiente:

- Existe un inicio.
- Existe un fin.
- El inicio está conectado con el fin.
- Cada contenedor, de cada nodo, está lleno con al menos un nodo.
- Los campos de textos tengan un input no vacío y válido.

Ejecutor

Después de que el analizador comprobó la validez de los nodos, la clase *Executer* procede a la ejecución del primer nodo, y de forma recursiva, recorre cada uno de los contenedores de cada nodo y ejecuta el comportamiento individual de cada uno de ellos.

El comportamiento individual de cada nodo, que son derivaciones de la superclase *InterpreterElement*, define la naturaleza y las reglas de la gramática del lenguaje visual, indicando la sintaxis, que partes de la instrucción se ejecuta primero, cómo se evalúan las expresiones y dónde se asignan, entre otras cosas.

Estos comportamientos individuales también son los encargados de comunicarse con la clase encargada de guardar y gestionar los valores de las variables y los arreglos, como se verá más adelante.

A continuación, se describe el pseudocódigo simplificado del ejecutor, que utiliza las siguientes variables de instancia:

```
nextAnswer := ""
stack := []
```

Función inicial del ejecutor:

```
fn execute(beginning): -> void

    if (analyzer.analyze(beginning) is false):
        return

    stack.push(beginning.interpreterElement)
    executeNext()

end
```

Función que se llama cada ciclo de actualización o cada vez que el robot termina una instrucción:

```
fn executeNext(): -> void

    current := stack.peek()
    state := "robot"

    if (current.type is end):
        stack.pop()
        stop()
        return

    if (current.isFinished):
        stack.pop()

    if (current.isExpression):
        nextAnswer := current.answer
    else:
        pushNext(current.next)

    else:
        state := current.executeNextStep(nextAnswer)

    if (state is "immediately"):
        executeNext()

end
```

Función ejecutada por el robot al terminar una instrucción para informar al Ejecutor de que finalizó:

```
fn receiveAnswerFromRobot(answer): -> void

    nextAnswer := answer
    executeNext()

end
```

Función para añadir un nuevo nodo a la pila:

```
fn pushNext(next): -> void

    if (next exists):
        next.reset()
        stack.push(current)

end
```

Para demostrar mejor la interacción entre el ejecutor, los nodos y el controlador del robot, se presenta el diagrama de secuencia simplificado del proceso de comunicación cuando el ejecutor está funcionando:

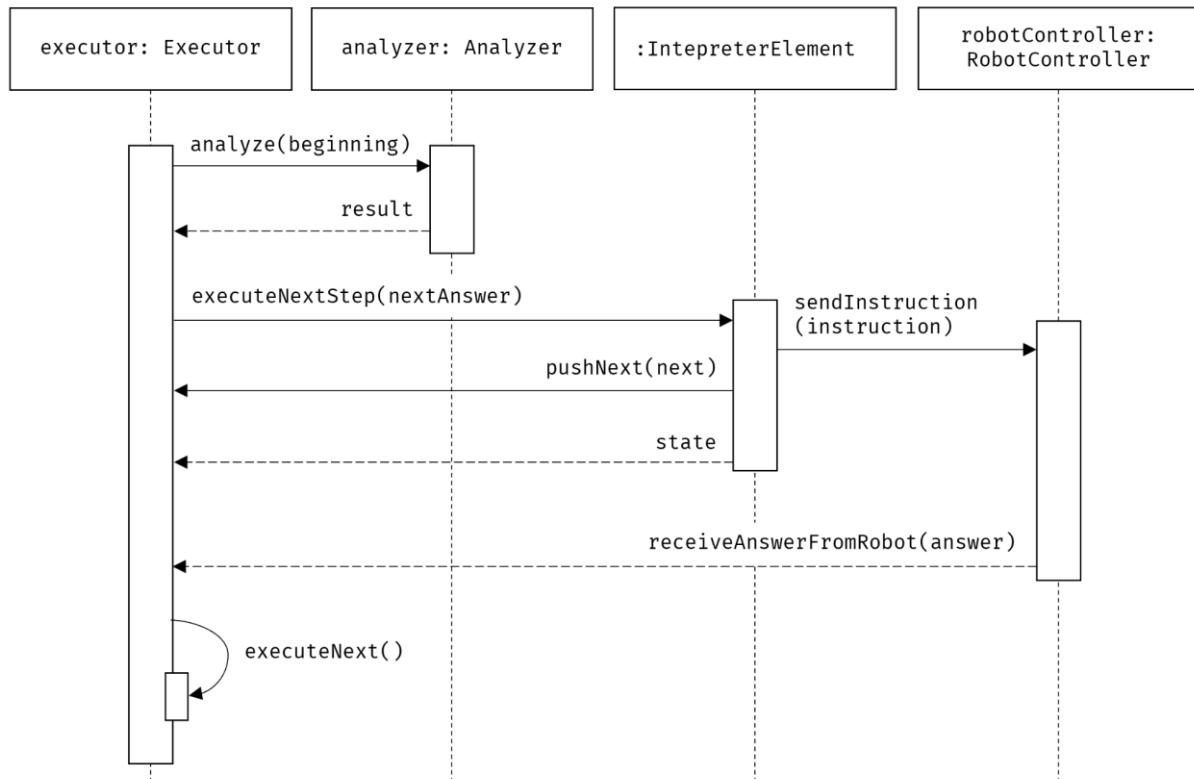


Figura 7.9. Secuencia de comunicación entre el ejecutor y el controlador del robot.

Variables y arreglos

Las variables y los arreglos son casos especiales de nodos, ya que cada instancia de nodo debe hacer referencia a una instancia única compartida que es la encargada de guardar el valor o los valores de cada uno de estos *símbolos*.

El encargado de guardar estos valores es la clase *SymbolTable*, que además de guardar cada una de las declaraciones de cada variable y/o arreglo, también se encarga de actualizar la tabla de valores que se encuentra en la interfaz si el valor o los valores son cambiados en el código y también lleva la cuenta y gestión de los nodos que se encuentran presentes en la zona de trabajo.

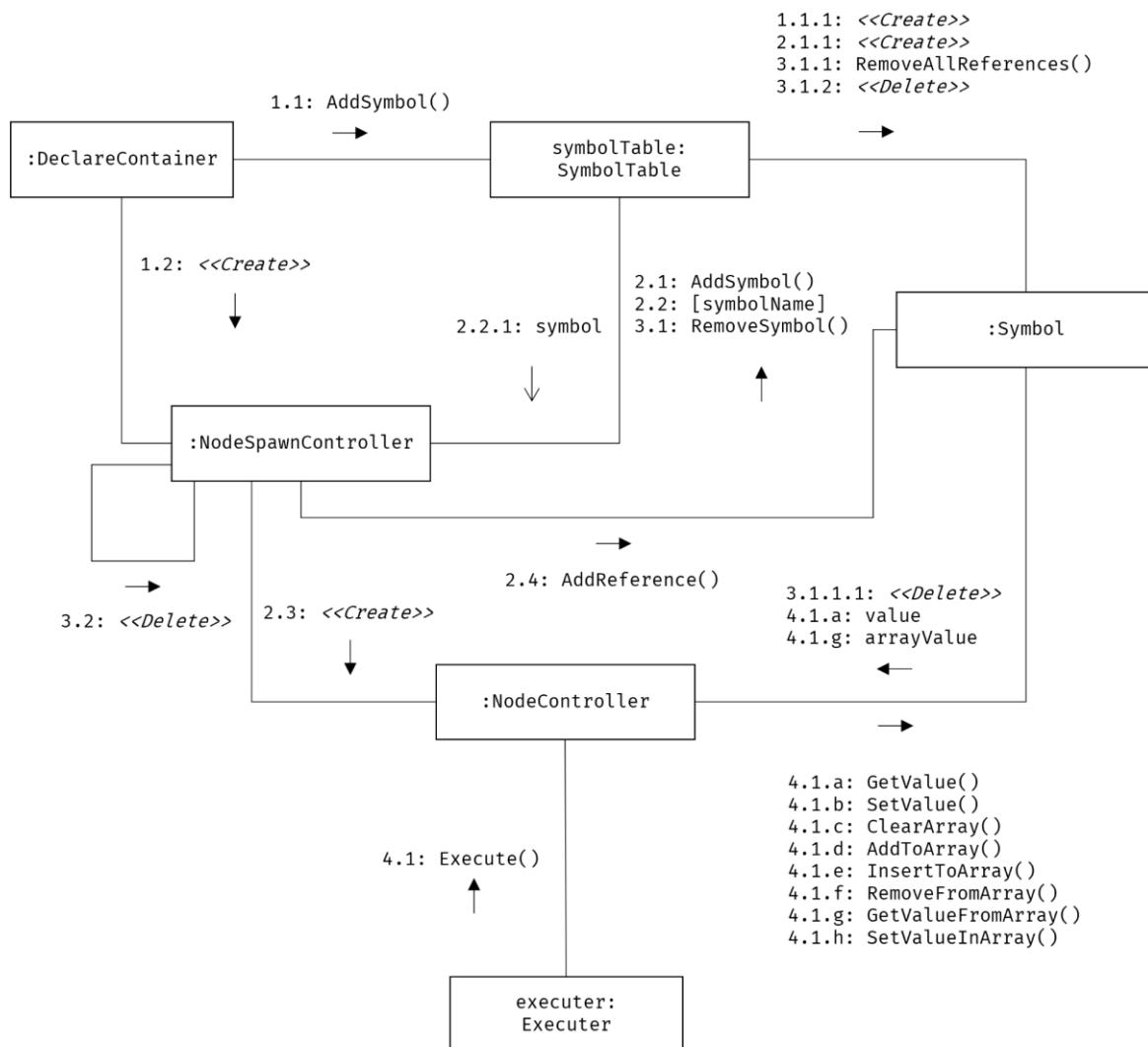


Figura 7.10. Interacción simplificada entre la declaración de símbolos, la tabla de símbolos, los nodos y el ejecutor.

Cada declaración de variable y arreglo es representada bajo una instancia de la clase *Symbol*, instancia que es guardada en una tabla hash que es miembro de *SymbolTable*, que utiliza el nombre del símbolo como llave para acceder a la instancia en concreto.

Para simplificar el diseño, cada tipo de nodo, como Inicio, Repetir, etc., son representados como símbolos que no guardan ningún valor, para así poder gestionarlos desde la misma clase controladora *SymbolTable*.

Entonces cada vez que es necesario acceder a una variable o un valor de un arreglo en concreto o que estos cambien, las instrucciones que son llamadas por el *Executer* se comunican con la *SymbolTable* para realizar esas operaciones, como se puede ver en la **Figura 7.10**.

InterpreterElement

Como ya se describió anteriormente, el comportamiento individual de cada nodo define el comportamiento de la gramática del lenguaje visual, indicando la sintaxis, el orden de las partes de cada instrucción, así como la evaluación y uso de las expresiones si es que las hubiera.

El componente *interpreterElement* de un nodo es una clase derivada de la superclase abstracta *InterpreterElement*, la cual define el comportamiento para los diversos estados visuales que sufre el nodo cuando se está en ejecución, define el estado inicial y cómo obtener el siguiente *interpreterElement* a ejecutar; dejando la implementación de la gestión y cambio de estado interno, así como la ejecución, a las derivaciones.

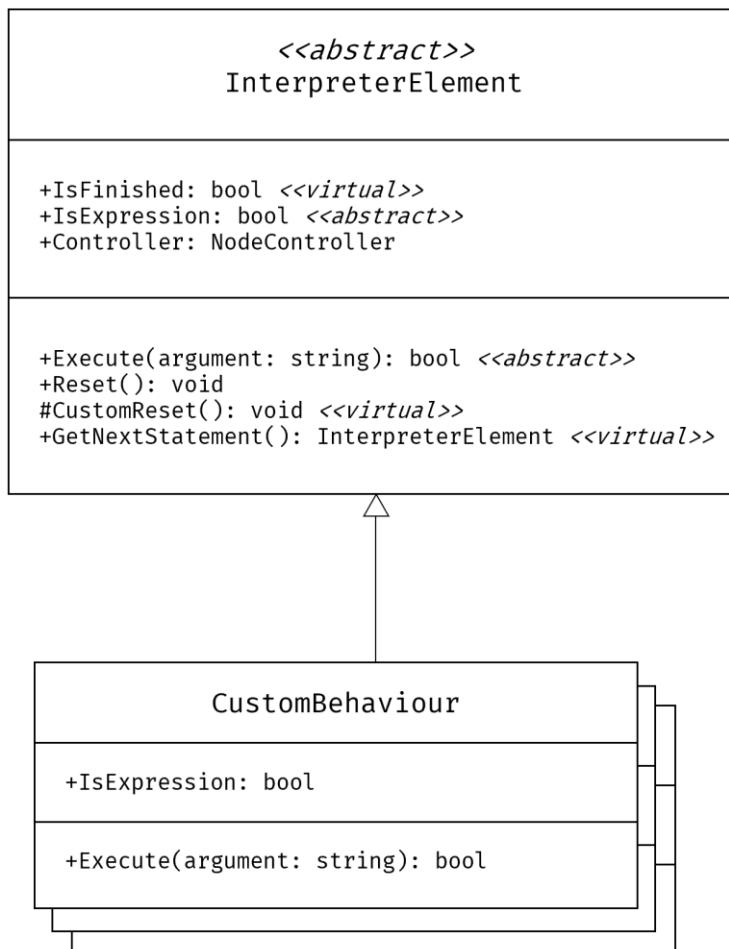


Figura 7.11. Relación entre la superclase *InterpreterElement* y el comportamiento individual de cada nodo.

Para exemplificar el comportamiento individual de los nodos, vamos a tomar como ejemplo el nodo Repetir (archivo fuente: **RepeatInterpreter.cs**), cuya ejecución toma 3 estados distintos:

→ **PushingCondition:**

En este estado, el nodo añade a la pila del ejecutor el *InterpreterElement* de la condición que se encuentra añadido al nodo para su evaluación.

→ **EvaluatingCondition:**

El nodo recibe el resultado de la expresión evaluada, al ser un booleano, decide si continuar o no con la condición. Si sí, entonces cambia su estado interno a ExecutingInstructions, de lo contrario, marca IsFinished como verdadero.

→ **ExecutingInstructions:**

El nodo añade el primer *InterpreterElement* de los nodos que tenga como instrucciones a la pila del ejecutor y cambia su estado a PushingCondition, para evaluar de nuevo la condición cuando todas sus instrucciones hayan acabado.

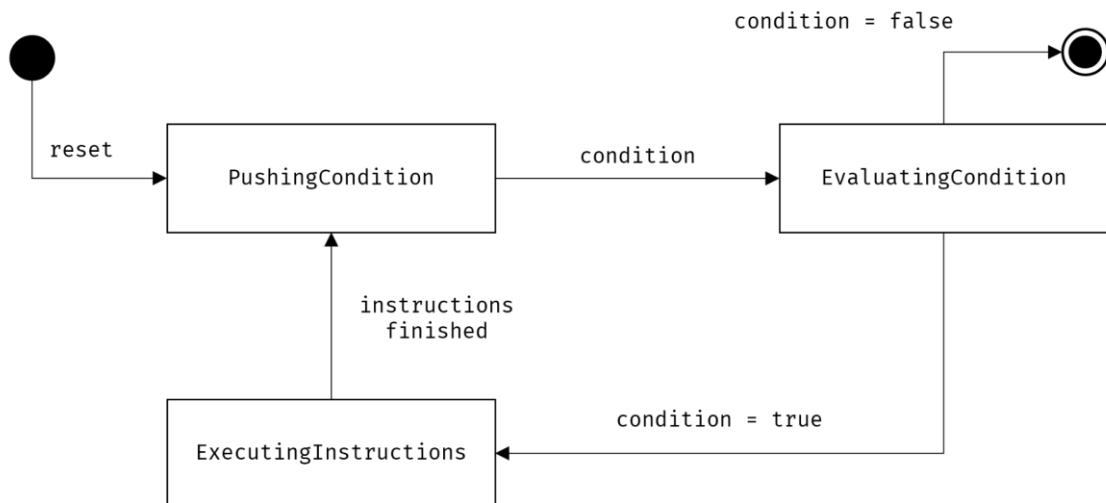


Figura 7.12. Flujo de estados del nodo Repetir.

Mecánicas y Jugabilidad

Acciones del jugador

Movimiento

El jugador es capaz de moverse dentro del mapa tanto de forma horizontal como vertical usando las flechas o las teclas WASD. El movimiento del jugador en el mapa está delimitado por los colliders en el mismo. Debido a la perspectiva decidida para el arte del juego, el collider integrado en el jugador, lo que realmente colisiona con los otros colliders se encuentra únicamente a la altura de los pies.

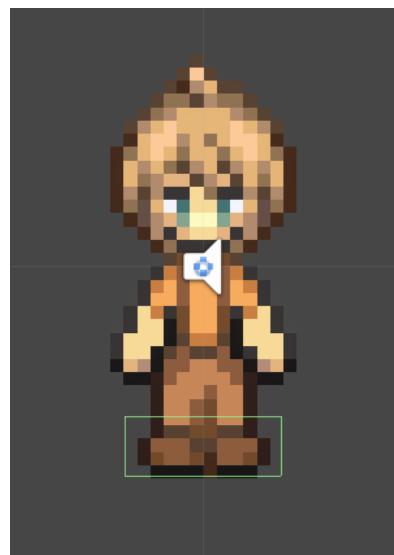


Figura 7.13. Sprite del personaje con BoxCollider a la altura de los pies.

Interactuar

La interacción es una mecánica que le permite al jugador accionar palancas y acceder al código de Copper. Para determinar si el jugador puede interactuar con algo, tiene que colisionar con el collider de un objeto interactuable y ver en dirección al mismo. Para que el usuario vea que puede interactuar con algo al entrar en contacto con él, se muestra un ícono de "E" (la tecla asignada para interactuar) sobre la cabeza de Bel.



Figura 7.13. Bel junto con el sprite usado para indicar al jugador que puede interactuar con un objeto.

Inspección

La función de Inspección es una mecánica que le permite al jugador ver cómo elementos como botones, palancas y puertas están conectados entre sí, así dando pistas al jugador sobre qué tiene que activar para poder avanzar al siguiente nivel. Para esto, se tiene en escena un GameObject “cables” el cuál contiene otras instancias de cables en la escena los cuales se ocultan al iniciar el juego y son mostrados mientras el modo inspección está activo.



Figura 7.14. Un cable conectando un botón y una puerta vistos en el modo de Inspección.

Otra característica del modo Inspección, es que muestra agrega un efecto de distorsión de color verde con líneas verticales que se desplazan hacia abajo.



Figura 7.15. GameObject con el Shader de “Inspector”.

Este efecto se obtiene gracias al Shader “Inspector”, el cual fue configurado usando el entorno visual de edición de Shaders, “Shader Graph”, en donde el resultado es producto de una serie de filtros, los cuales en conjunto con el uso de una función tiempo y de un nodo de Gradient Noise. Este shader es usado como material en un Sprite cuadrado que cubre la pantalla y es mostrado mientras el jugador está en modo Inspección.

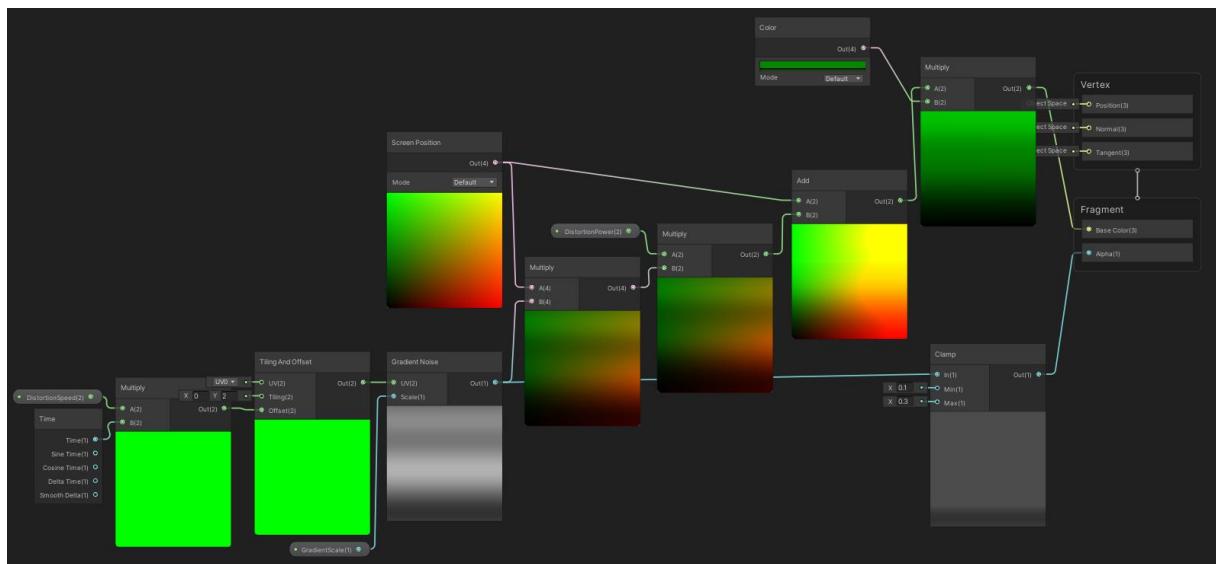


Figura 7.16. Shader Graph del Shader “Inspector”.

Reiniciar nivel

En caso de que el jugador cometa un error, la mejor opción es usar la función de Reiniciar, la cual es activada al presionar la tecla “R”, esto revertirá todos los cambios en la sala en actual.

Comportamiento del robot

RobotController

El Script RobotController recibe las instrucciones mandadas por el intérprete para que se traduzcan en dentro del juego, tales como caminar, girar, escanear, entre otras. La forma en la que se sabe qué es lo que está realizando Copper es haciendo uso de un enum que indica la acción realizada en el momento, cuando Copper termine de ejecutar la acción en cuestión, su estado pasará a None, cuando esto ocurre, notifica al Intérprete para continuar con el resto de la ejecución del algoritmo.

El estado de Copper, en conjunto con la dirección en la que está viendo es usado también para determinar la animación que debe de ejecutarse actualmente. Para controlar esto se hizo uso de un árbol de estados.

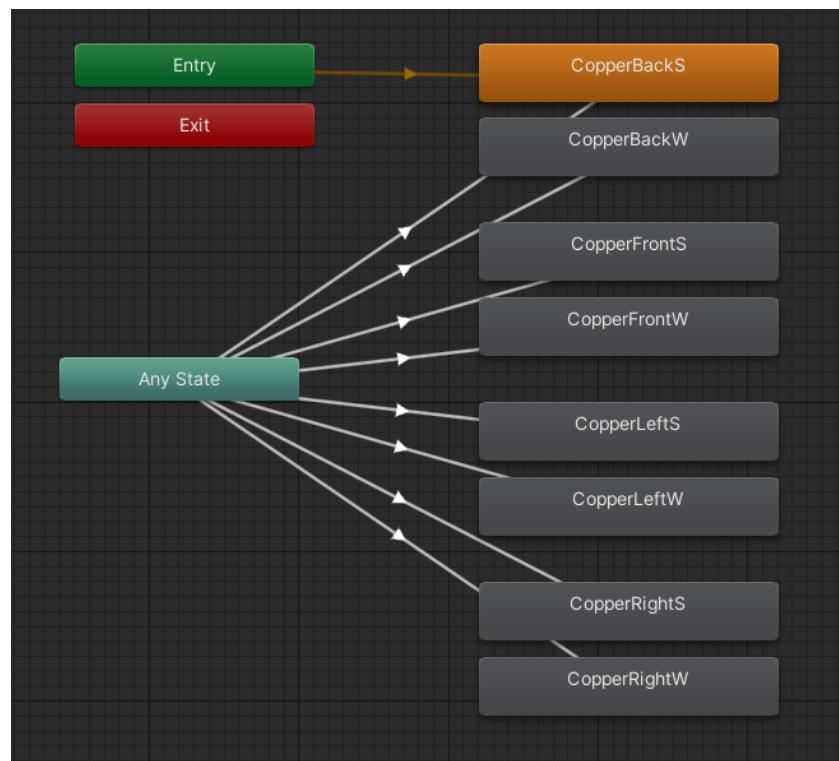


Figura 7.17. Árbol de estados de las animaciones de Copper

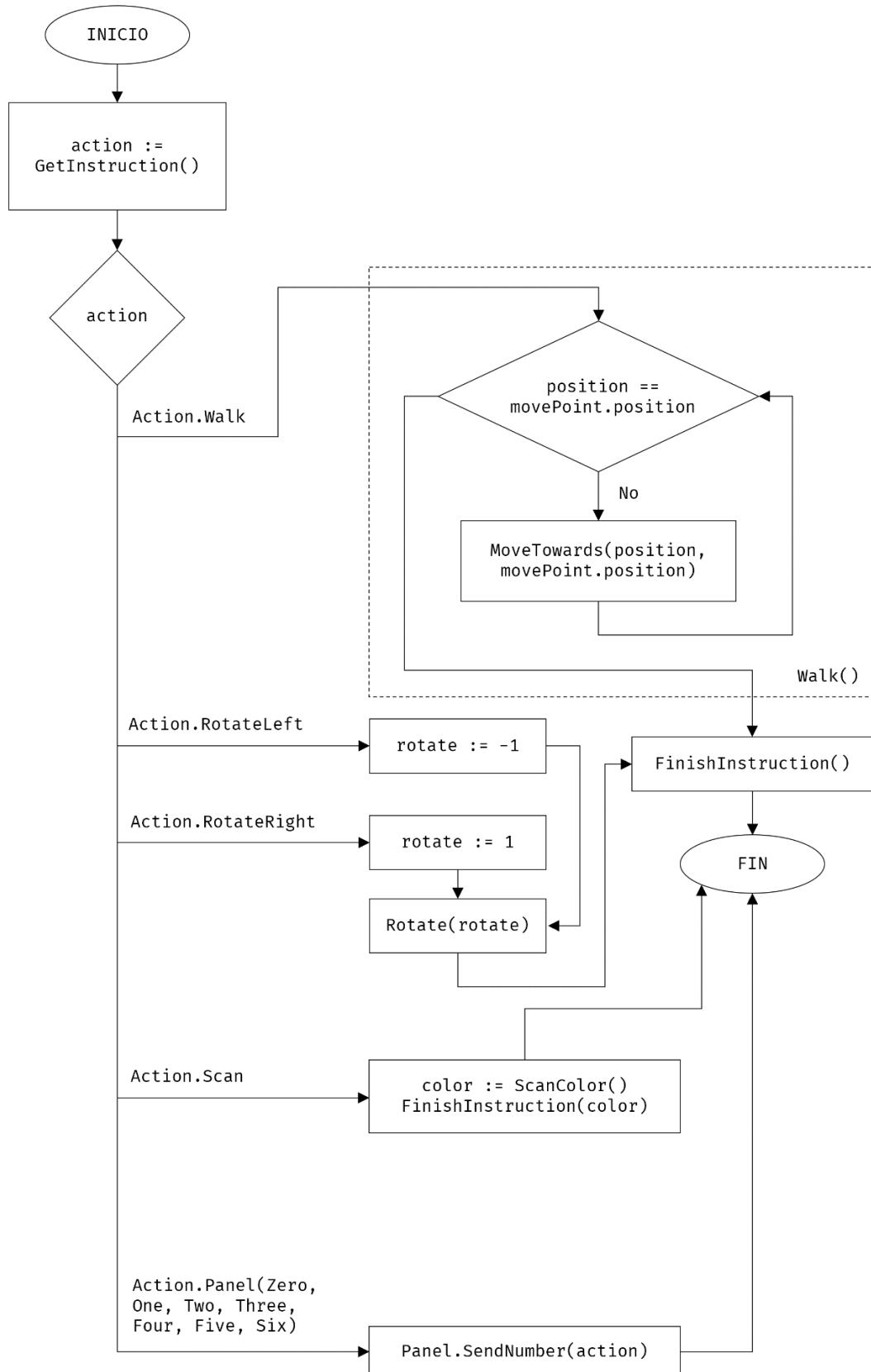


Figura 7.18 Diagrama de flujo simplificado de las decisiones del RobotController

Sistema de energía

Una característica importante en el juego es el sistema de energía, éste es usado para determinar qué elementos están conectados entre sí, por ejemplo, qué botón abre la puerta, cuántos botones se tienen que presionar de manera simultánea para que permanezca abierta. Para todas las interacciones entre los elementos se hace uso del evento **evento_Energia**; cada elemento que emite una señal posee un id y cuando hay un cambio en su estado (encendido / apagado), emite una señal a través del evento en donde envía su id, de esta manera, sólo los elementos que están conectados al mismo reaccionan a esto.

Cables

Las conexiones entre elementos dentro del sistema son representadas haciendo uso de los cables revelados con el modo inspección; para saber a qué elemento está conectado un cable, se almacena el id del mismo. Los cables se suscriben **evento_Energia**, así, cuando se mande un mensaje a través de éste, los cables comparan el id recibido con el almacenado y sólo reaccionan en caso de que estos concuerden, cambiando de estado (encendido / apagado).

Los cables reflejan su estado haciendo uso de dos diferentes shaders, uno para cuando no está pasando corriente y otro para el caso contrario.



Figura 7.19. Un cable encendido con el Shader “CableOn”.

Los Shaders fueron creados usando el entorno visual de Shader Graph, lo que se hace para que se vea más brillante cuando está encendido que apagado, es que se multiplica el color del sprite original por un color más claro y eso es usado como el color final del Shader.

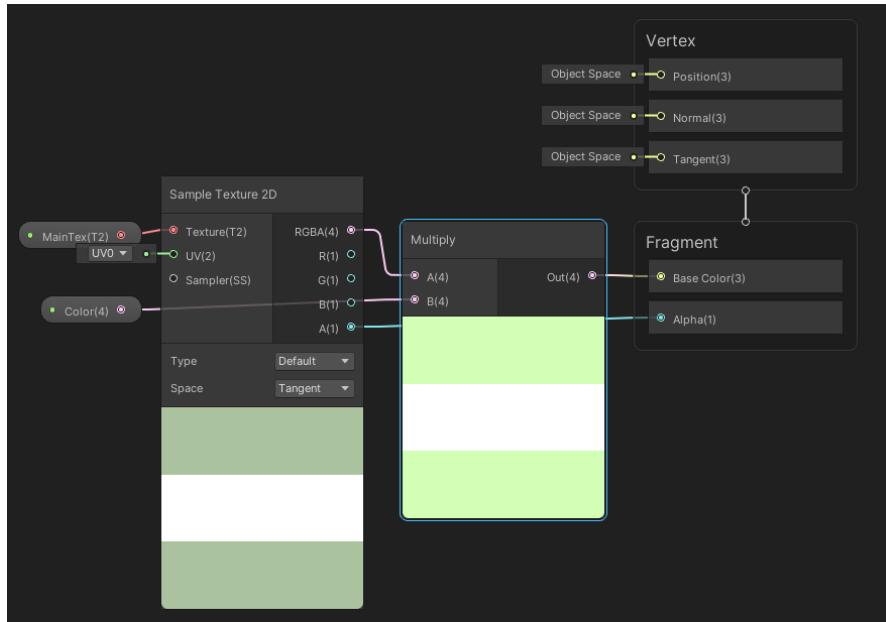


Figura 7.20. Shader Graph del Shader “CableOn”.

Puertas

Las puertas se encuentran presentes en todos los niveles. Las puertas, al igual que los cables, están suscritas al método _Energia. Las puertas almacenan un arreglo de enteros, los cuales representan los ids de los elementos conectados a la misma, así, cuando reciben un mensaje a través de método _Energia, monitorean los cambios presentados en todos los elementos a los cuales está conectada. La forma en la que las puertas determinan si se deben de abrir o cerrarse es con un ciclo en donde a un TRUE, se le hacen una serie de operaciones AND con los estados de los elementos conectados a la puerta, una vez terminado el ciclo, si el resultado es TRUE, la puerta se abre, en caso contrario se cierra.

Cuando una puerta da a una sala contenida en el nivel, al abrirse se “corta” junto con el muro en el cual se encuentra para así no obstruir visualmente el contenido de la habitación.

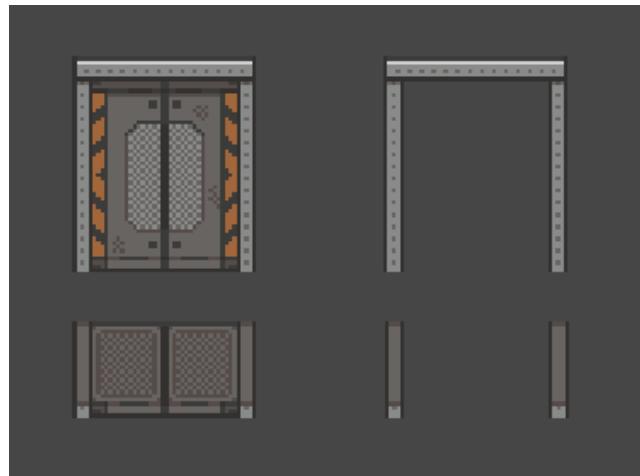


Figura 7.21. Sprites de Puertas cerradas y abiertas, tanto en su formato completo como “cortado”.

Botones

Los botones almacenan su ID, el cual es enviado a través de metodo_Energia cada que detectan que han sido ya sea presionados o si han dejado de ser apretados. Los botones tienen un collider en modo Trigger, lo que permite saber si algún objeto se pone o se quita de sobre el botón. Para lidiar con la posibilidad de que más de un objeto entre en éste trigger, el botón lleva cuenta de los objetos que se encuentran sobre él; esta cuenta aumenta cuando un objeto entra y disminuye cuando uno sale, así, el botón siempre se encuentra presionado mientras que la cuenta sea mayor a 0.

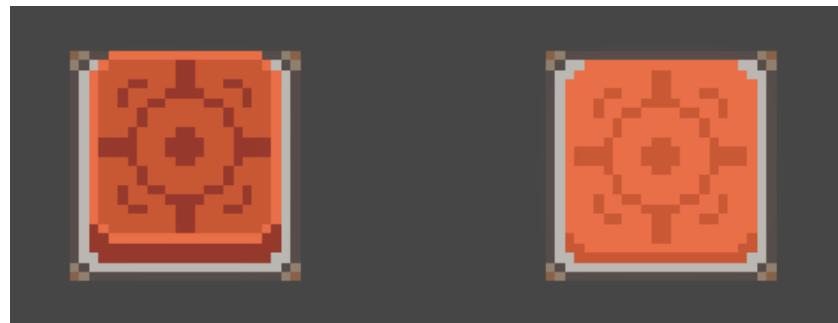


Figura 7.22. Sprites de un botón sin presionar y presionado.

Palancas

Las palancas son objetos interactuables, cuando el jugador entra en contacto con ellas y mira en dirección de éstas, sale el icono de interacción avisando al jugador que puede activarla.

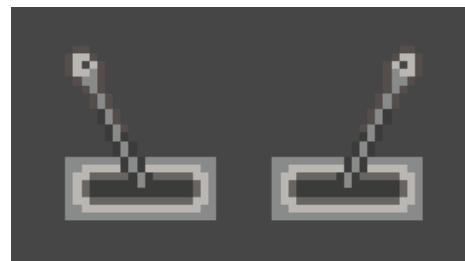


Figura 7.23. Sprites de una palanca desactivada y activada.

Paneles

Los Paneles son teclados en los cuales Copper puede ingresar una serie de números haciendo uso del nodo “Ingresar número al panel” mientras se encuentra frente a él. Ingresar la secuencia de números correcta manda una señal de encendido a la puerta a la cual se encuentra conectada. Los paneles tienen un display sobre ellos donde muestra el número ingresado.

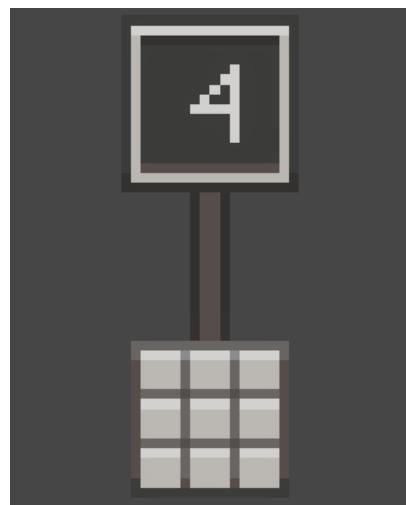


Figura 7.24. Sprite de Panel mostrando en el display que el último número introducido fue el 4.

Mensajes

Se implementó un sistema de encolamiento de mensajes para informar al jugador sobre aspectos o eventos dentro del juego que podrían ser de su interés; o para informar de errores, principalmente del editor de código y la ejecución de este, mensajes enviados principalmente por el Ejecutor.

El API de mensajes que se implementó soporta los siguientes parámetros / opciones que ofrecen más flexibilidad al programar:

- *Message: string.* El mensaje que se quiere mostrar.
- *Type: MessageType.* El ícono predeterminado que se usará para desplegar el mensaje. Soporta *Normal, Warning* y *Error*.
- *Seconds: int.* El número de segundos que el mensaje será desplegado. Si este parámetro es negativo, el mensaje se desplegará de forma indefinida hasta que el usuario lo descarte.
- *IsFinite: bool.* Indica si el mensaje se desplegará en un tiempo finito o no. Sobreescribe el parámetro *Seconds*.
- *CustomSprite: Unity.Sprite.* Imagen a mostrar en lugar del ícono.
- *HideInNewMessage: bool.* Indica si el mensaje se ocultará si un nuevo mensaje es añadido, independientemente de su duración.
- *OnFullShowCallBack: System.Action.* Función callback que se ejecuta cuando el mensaje completa su animación de entrada.

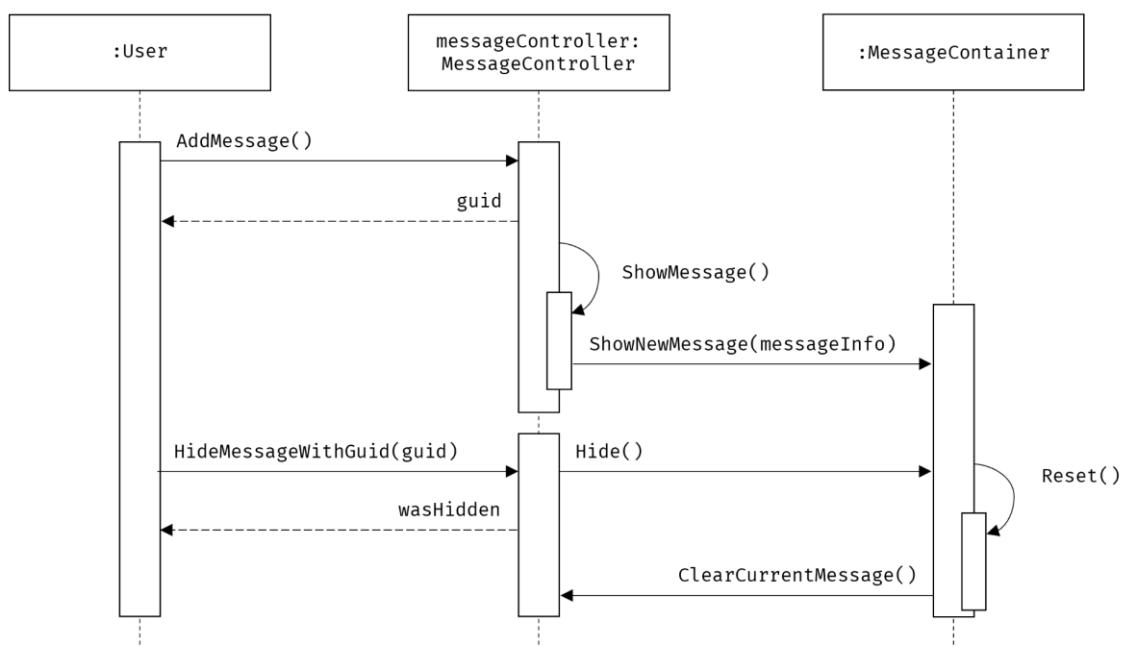


Figura 7.25. Secuencia de comunicación para mostrar / ocultar mensajes.

Además, se genera un *System.Guid* que es retornado para su posterior utilización que permite identificar el mensaje independientemente de su contenido.

Tutoriales

Se utilizó el API de mensajes para la creación de tutoriales sencillos que guían al usuario en los diferentes niveles del juego. La mayoría de los mensajes no desaparecen por tiempo a menos que aparezcan otros mensajes no relacionados al tutorial, y desaparecen cuando se realizan las acciones que son indicadas en los mensajes.

Por ejemplo, el mensaje que le indica al jugador cómo moverse utilizando ciertas teclas desaparece hasta que las 4 teclas son utilizadas por el jugador.



Figura 7.26. Mensaje tutorial indicando al jugador los controles de movimiento.

Esto se logró utilizando un sistema de señales que son enviadas cuando suceden diversos eventos a un controlador que son recibidas por *TutorialController*, un singleton encargado de servir como mediador entre el juego y la instancia del tutorial actual que se encarga de indicar qué mensajes mostrar y bajo qué eventos estos cambian para mostrar los siguientes mensajes.

Cada nivel tiene una instancia de una clase derivada de la superclase *Tutorial*, donde se definen los mensajes específicos que cada nivel va a tener y los eventos a los que va a responder el tutorial. Esta instancia es también la encargada de mandar los mensajes hacia el *MessageController*.

Esta instancia guarda el *guid* del mensaje que agregó para que cuando el evento correcto suceda, este pueda ocultar el mensaje específico, no un mensaje diferente.

El tutorial comienza cuando el *SplashScreenController* lanza el evento *OnDisappear*, para que los mensajes se logren apreciar sin que la Splash Screen inicial interrumpa los mensajes.

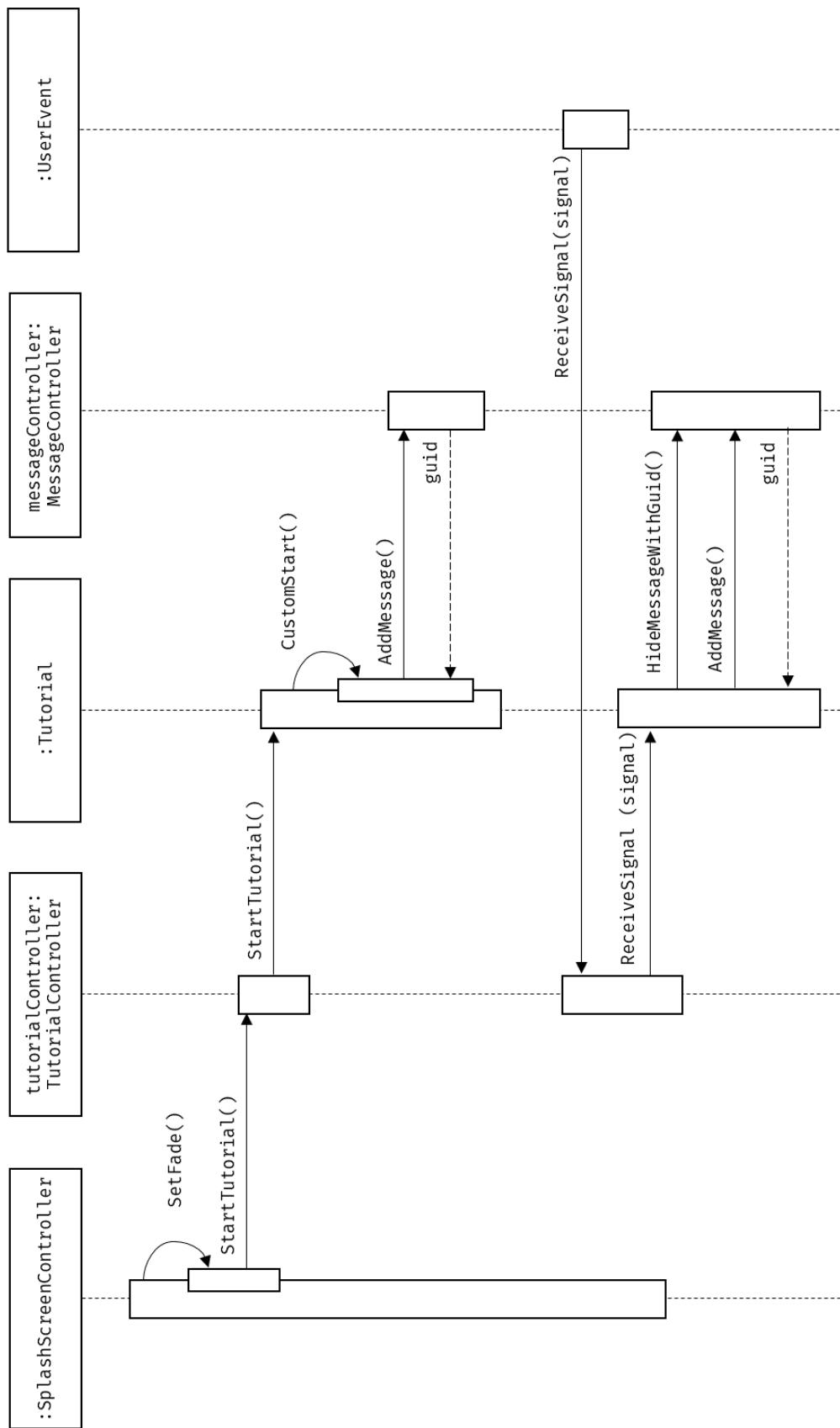


Figura 7.27. Secuencia de comunicación entre el *TutorialController*, los eventos y el *MessageController*.

Niveles

Arquitectura de niveles

Los niveles están conformados por múltiples capas de Tilemaps. Los Tilemaps son cuadrículas donde se ponen los sprites generados para crear diferentes partes del mapa; estos Sprites se agrupan en algo llamado Tile Palettes, su función es contener los mosaicos que se pondrán en la cuadrícula para fácil acceso.

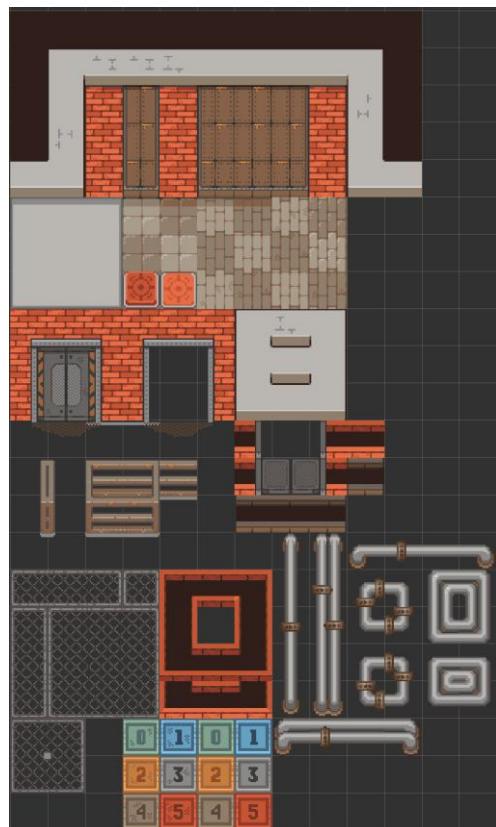


Figura 7.28. Captura del Tile Palette producido para crear los niveles, conteniendo muros, placas, tubos, puertas, etc.

En cada nivel se usaron al menos 6 Tilemaps (más 1 extra que se puede duplicar en caso de que la situación lo amerite), cada uno con su propósito en el diseño de los niveles, la importancia de la existencia de algunos de estos tilemaps y el por qué no se hizo en menos capas radica en el orden de renderizado, ya que en ocasiones se necesita que algo se renderice siempre debajo o encima de otros elementos. Los Tilemaps necesarios para la creación de los niveles son:

1. **Ground:** Este tilemap es usado para dibujar la textura que tendrá el piso; el suelo necesita su propia capa ya que así se define qué es lo que se encuentra siempre hasta abajo y que nunca se renderiza sobre otros objetos.

2. **GroundMisc:** Este tilemap tiene como función permitir agregar detalles sobre el suelo como suciedad o desgaste. La razón por la cual estos detalles no son incluidos en la capa del piso es porque eso requeriría que cada tile del piso tenga una versión alternativa con suciedad o desgaste para tener la libertad de poner estos detalles donde sea.
3. **Foreground:** En este tilemap se dibujan los elementos que siempre se renderizan sobre cualquier otro elemento en la pantalla (a excepción del UI).
4. **WallCollide:** Aquí se dibujan los muros con los cuales el jugador podrá colisionar. Para que esto ocurra, se usa un componente llamado Tilemap Collider 2D, esto hace que cada tile que contenga un sprite en la cuadrícula, tendrá un collider.

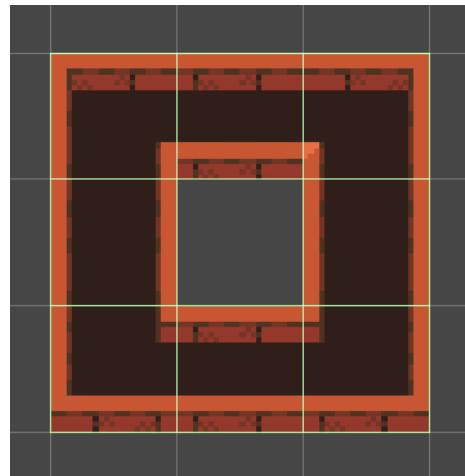


Figura 7.29. Captura del Tilemap con el componente de Tilemap Collider 2D.

En adición de esto, se incluyeron dos componentes más, Rigidbody 2D y Composite Collider 2D, esto para unir los colliders generados por el Tilemap Collider 2D y se vuelva un sólo collider de una sola pieza. Esto es importante ya que evita que los colliders de elementos que se mueven en el mapa tengan interacciones inesperadas al colisionar con las paredes.

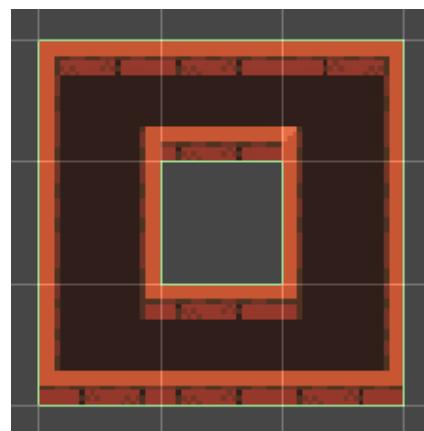


Figura 7.30. Captura del collider compuesto gracias al uso de Composite Collider 2D.

5. **Railing:** Esta capa es para elementos sobreuestos en el piso con los cuales el jugador colisiona, entre ellos barandales y tubos. Aunque los elementos en ésta capa sean obstáculos, éste tilemap no viene acompañado de un Tilemap Collider 2D, esto ya que los tiles de barandal y tubos tienen formas irregulares, lo cual al general colliders con éstos, pueden llevar a interacciones inesperadas con otros colliders, para evitar esto, se creó la capa **InvisibleCollider**.

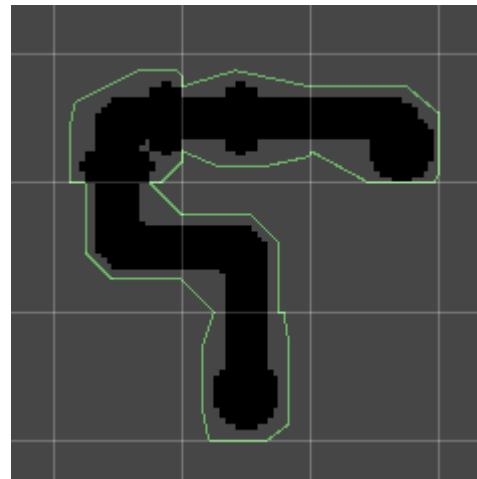


Figura 7.31. Collider irregular que se generaría en caso de que la capa Railing tuviera un componente de collider.

6. **InvisibleCollider:** Esta capa sirve para poner colliders independientemente de la apariencia de los elementos del nivel, para que los tiles rellenos en esta capa no alteren la imagen del nivel, se deshabilita el Tilemap Rendererer. Para asegurar un colliders regulares, se usan tiles cuadrados. Para esta capa se usaron los mismos componentes que en WallCollide para generar un sólo collider compuesto por cuerpo.

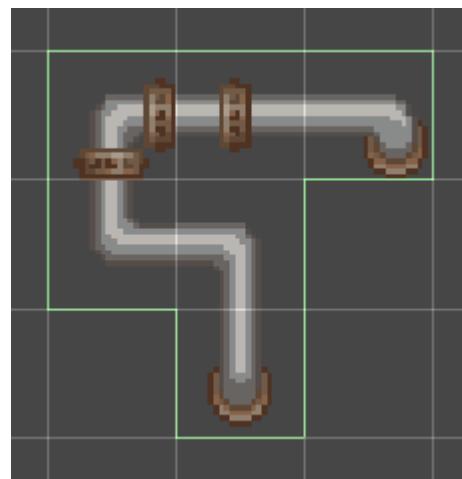


Figura 7.32. Collider regular generado en la capa InvisibleCollider.

7. HideWall (Opcional): Esta capa no es indispensable para todo tipo de nivel, pero en algunas ocasiones se puede necesitar una o más de éstas para el diseño de un nivel. En casos en los que haya una habitación bloqueada por una puerta, mientras que el jugador no tenga acceso a ésta, el muro que separa la habitación actual del jugador con la siguiente obstruye parcialmente la visión del jugador de lo que se encuentra al otro lado. La función de éste tilemap es mostrarse y ocultarse dependiendo de si el jugador ha abierto la puerta a la siguiente sección o no, para así obstruir o mostrar lo que se oculta detrás de la pared.

Ésta capa funciona como una extensión de la puerta, monitoreando el estado de la misma para saber si es necesario ocultar éste muro o no.



Figura 7.33. Ejemplo del uso de un tilemap de tipo HideWall.

Diseño de niveles

Nivel 1-1, Primeros pasos.

Este nivel está compuesto por una sola habitación, un botón y una puerta. Además de esto, hay unas placas numeradas cuyo propósito es el volver más explícito para un jugador sin experiencia en programación cuántos pasos tiene que dar Copper para alcanzar el botón. La solución de este nivel es hacer que Copper de 4 pasos en línea recta para que el jugador cruce la puerta al siguiente nivel. El objetivo de este nivel es familiarizar al jugador con el movimiento de Copper y el cómo se cuenta la cantidad de pasos que éste tendrá que dar para desplazarse de un punto a otro.

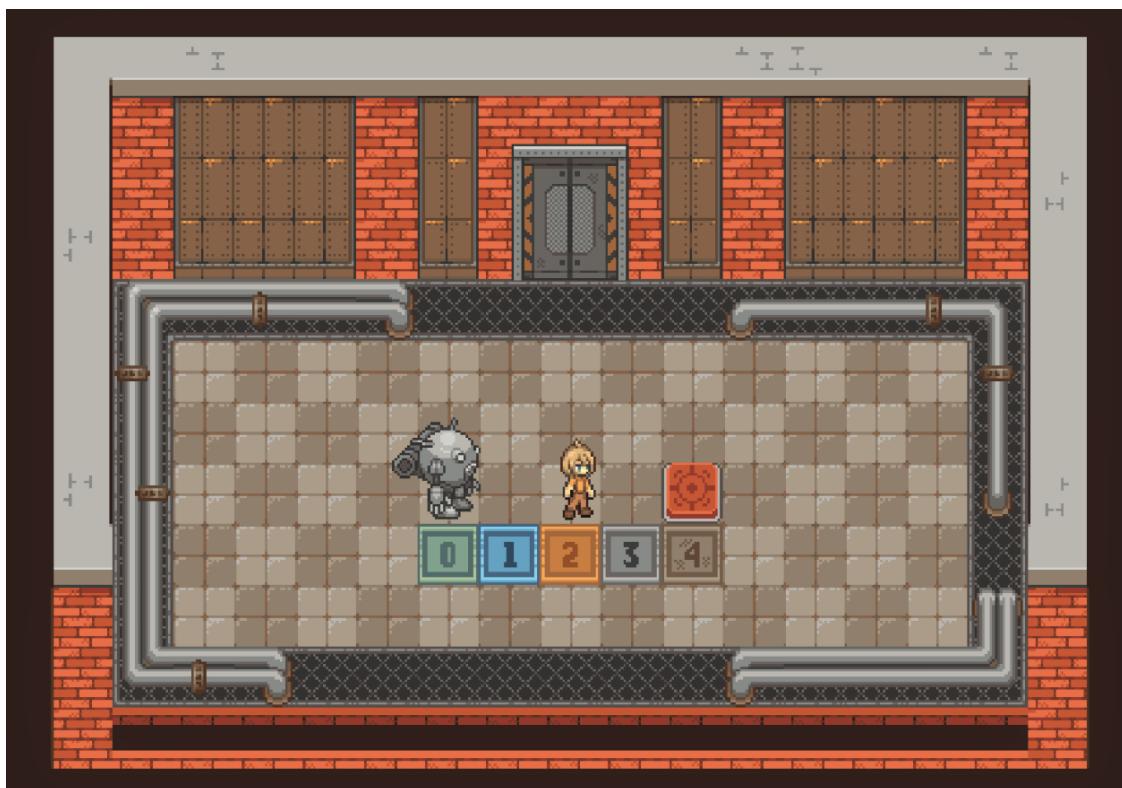


Figura 7.34. Captura pantalla del nivel 1-1.

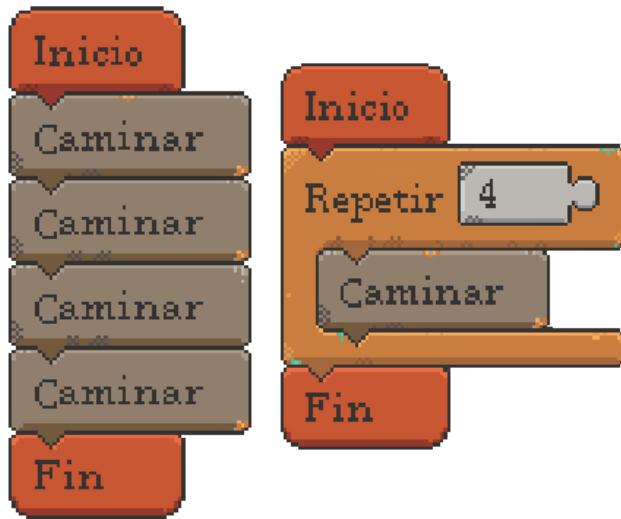


Figura 7.35. Dos alternativas soluciones al nivel 1-1.

Nivel 1-2, Zigzag.

Este nivel está compuesto por una sola habitación, un botón y una puerta y un pasillo. El objetivo del jugador es hacer que Copper llegue nuevamente al botón para abrir la puerta; para esto Copper tendrá que girar a la izquierda, avanzar cuatro espacios, girar a la derecha, avanzar dos espacios más, girar una vez más a la izquierda y finalmente avanzar dos espacios más. El objetivo de este nivel es apoyar más al jugador a familiarizarse con las acciones básicas que puede realizar Copper.

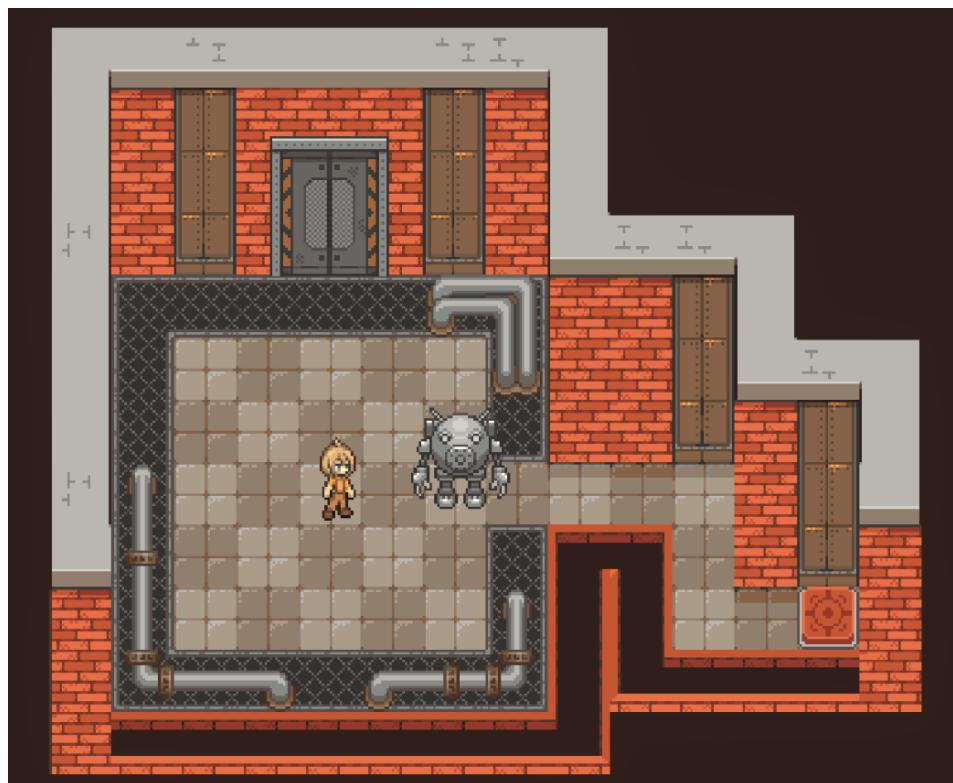


Figura 7.36. Captura pantalla del nivel 1-2.

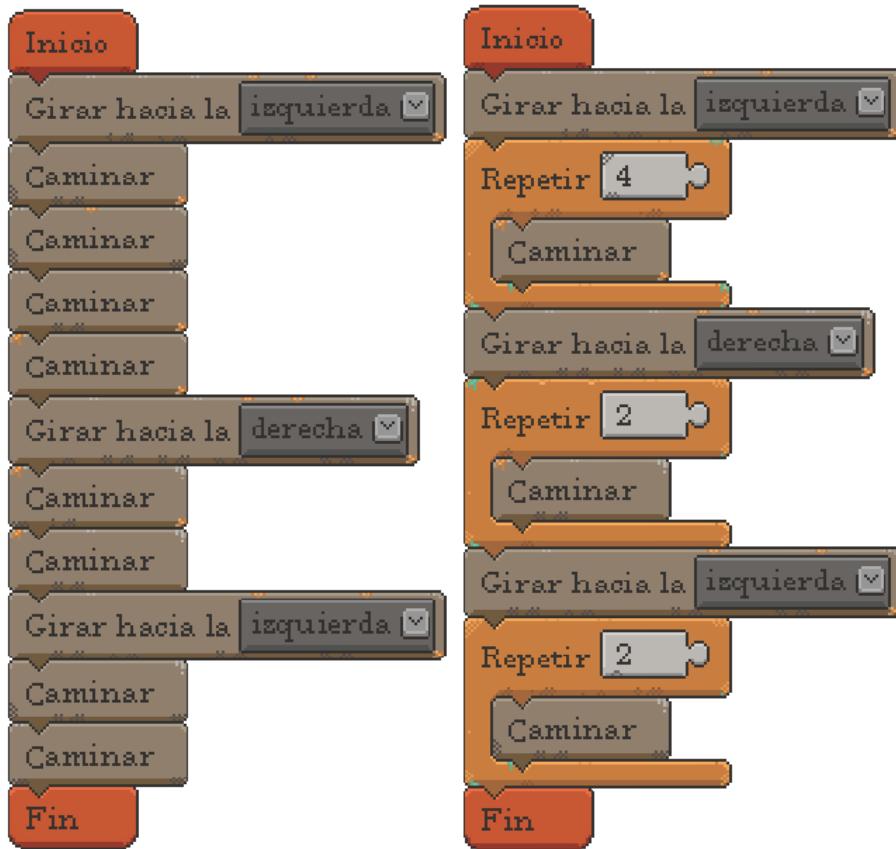


Figura 7.37. Dos alternativas soluciones al nivel 1-2.

Nivel 1-3, Marcha sin fin.

Este nivel está compuesto por una habitación, tres botones y tres puertas y dos pasillos. En este nivel, el jugador debe de hacer que Copper camine en línea recta hacia arriba, en el momento en el que Copper presione el primer botón, el jugador deberá pasar a través de la puerta a la derecha para entonces él apretar el segundo botón, el cual le abre la puerta de la izquierda a Copper, permitiéndole el paso y así Copper pueda seguir hasta llegar al último botón, el cual abre la puerta final para que el jugador la cruce. Debido a que el jugador no puede contar cuántos pasos tiene que dar Copper para llegar al final, la solución más sencilla es hacer un algoritmo con un ciclo infinito que contenga la instrucción caminar para que así no importe el largo, Copper avance hasta que un muro se lo impida.



Figura 7.38. Captura pantalla del nivel 1-3.



Figura 7.39. Solución al nivel 1-3.

Nivel X-1, Espiral.

Este nivel está compuesto por una sola habitación, un botón, una puerta y un pasillo en forma de espiral. El objetivo del jugador es hacer que Copper recorra toda la espiral hasta llegar al botón. Las cosas que el jugador debe considerar para dar con la solución de éste nivel son las siguientes. El número de pasos que Copper tiene que dar para llegar a la siguiente esquina y girar va disminuyendo de uno en uno desde diez pasos hasta un paso al final, de igual manera, el número de veces que Copper avanza por cada recta es diez, así que para solucionar esto, se debe de hacer que Copper avance y gire a la izquierda $10-n$ veces y por cada vez que se repita esto, se darán $10-n$ pasos, donde n va incrementando en uno cada que se vuelve a girar. De esta forma, el algoritmo se ejecuta hasta que n alcanza el valor de 10 y Copper deja de avanzar; para cuando esto ocurra ya estará sobre el botón y el jugador podrá pasar al siguiente nivel.

Este es un ejemplo del tipo de niveles que se pueden hacer para el juego con una dificultad más elevada comparada con los anteriores.

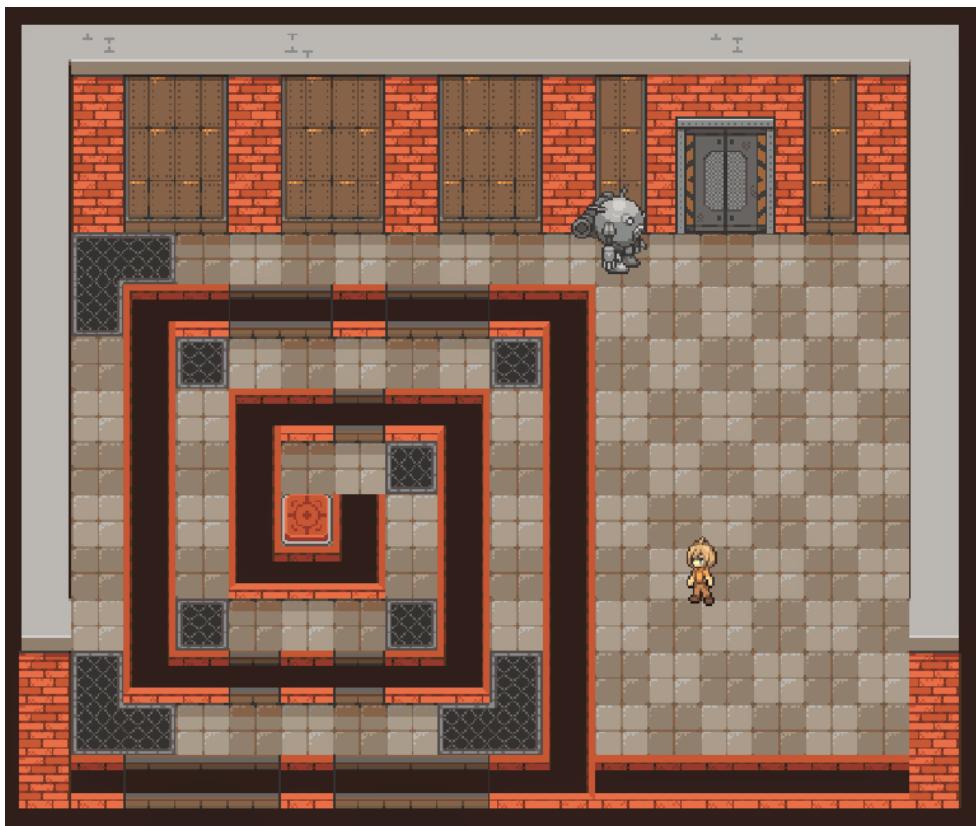


Figura 7.40. Captura pantalla del nivel X-1.

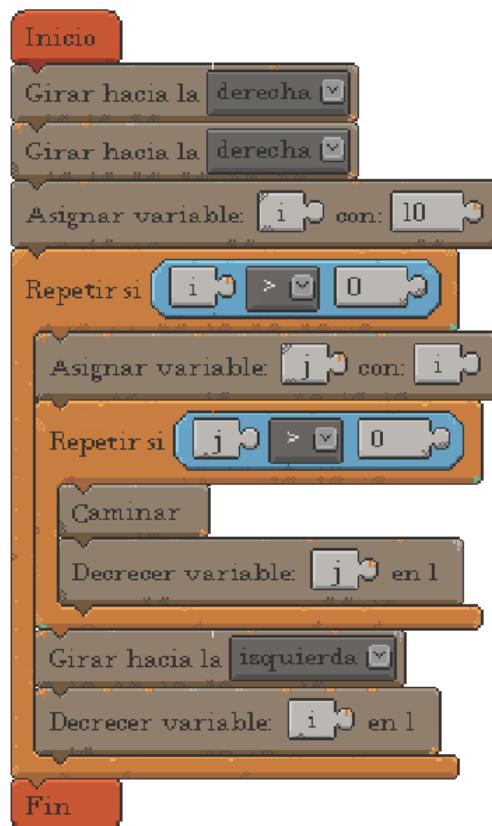


Figura 7.41. Captura pantalla del nivel X-1.

Nivel X-2, Ordenamiento.

Este nivel está compuesto por dos habitaciones, un botón, dos puertas y un panel. Para superar este nivel, el jugador debe de mantener presionado el primer botón para que Copper pueda avanzar hasta las placas, una vez Copper cruce la puerta, presionará un botón que asignará números aleatorios a las placas; lo que debe de hacer Copper es escanear los valores de cada una de ellas y guardarlas en un arreglo, para posteriormente ejecutar un algoritmo de ordenamiento y acomodar los valores del arreglo de menor a mayor e ingresarlos en ese orden en el panel, esto abrirá la puerta de la derecha permitiendo al jugador terminar el nivel.

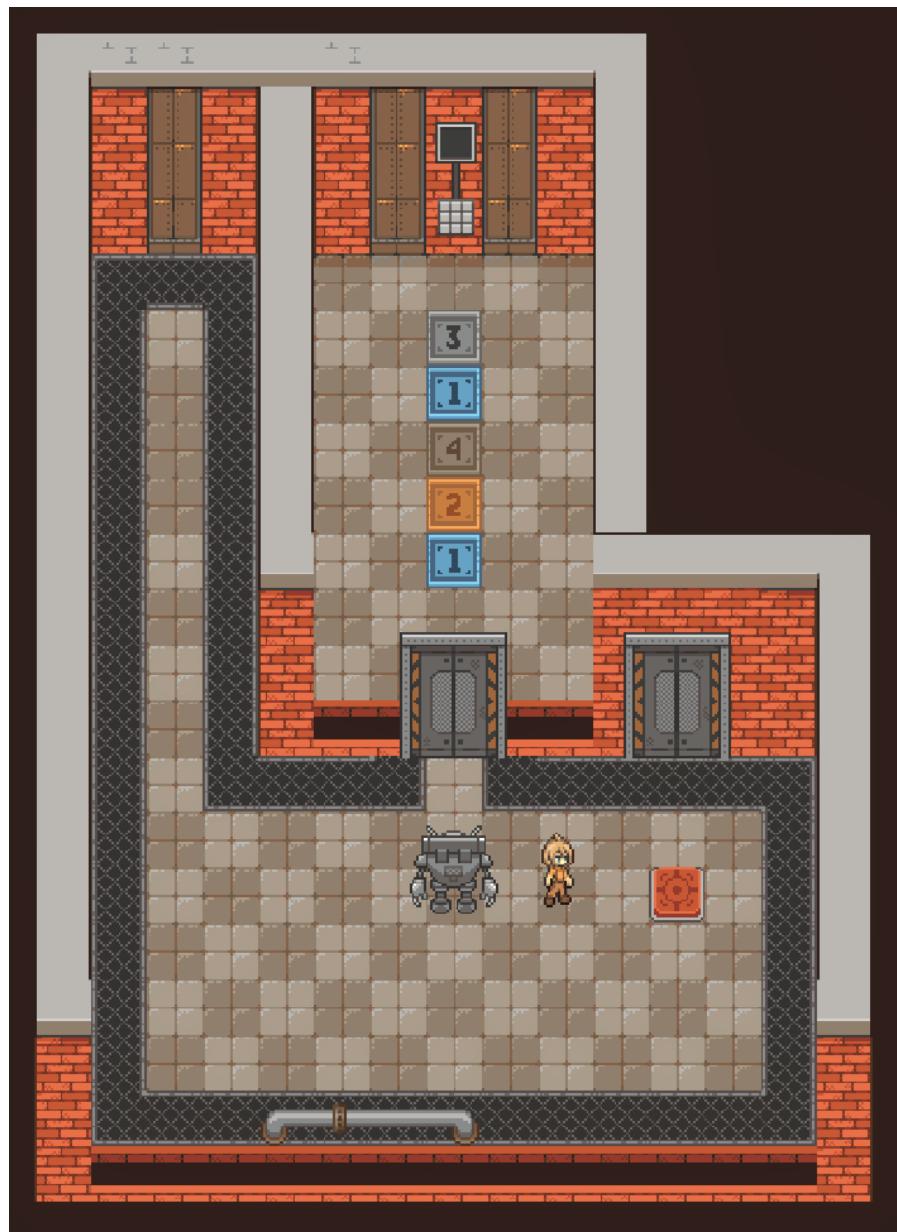


Figura 7.42. Captura pantalla del nivel X-2.

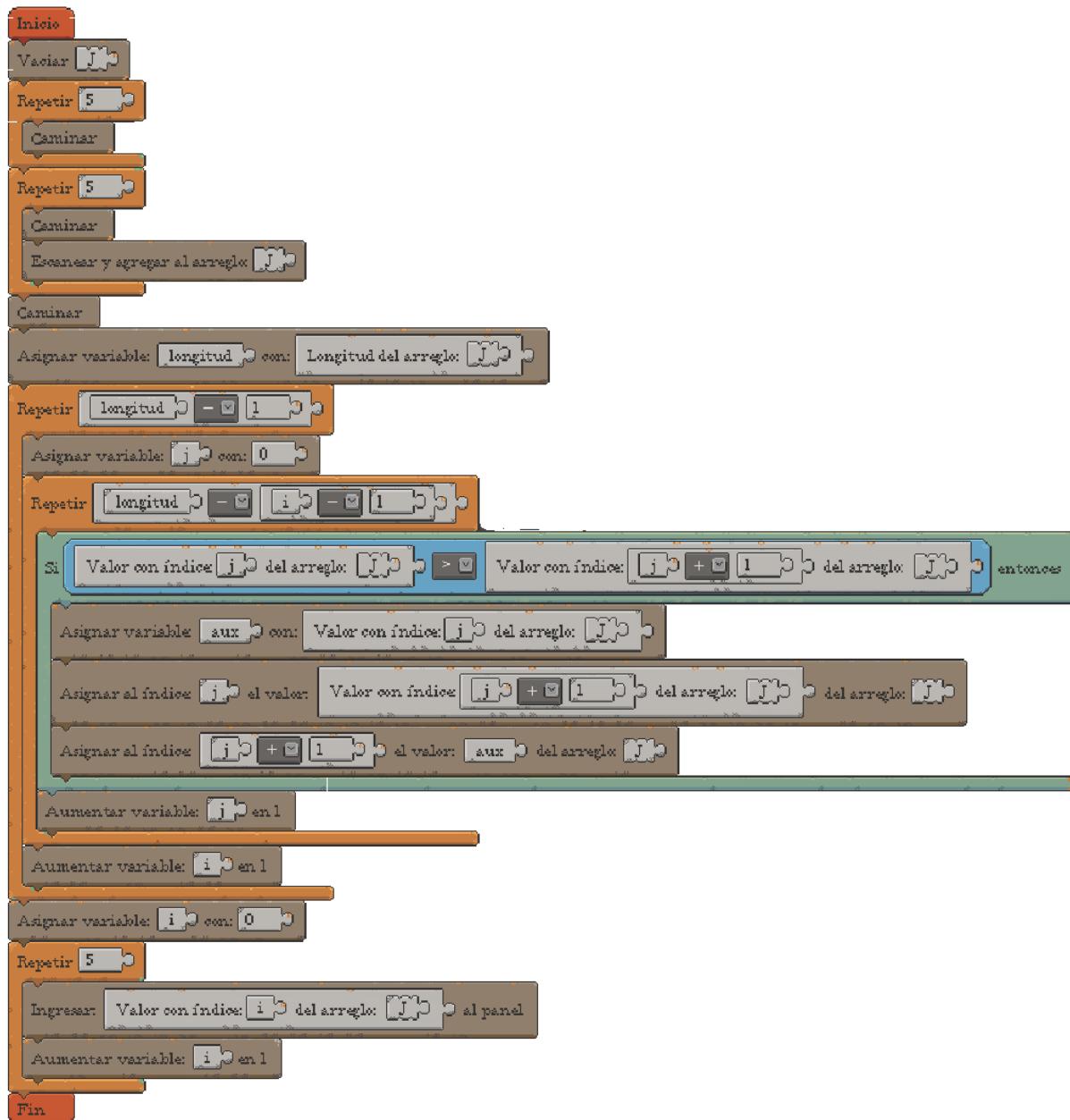


Figura 7.43. Solución al nivel X-2.

8

Resultados

Este capítulo contiene algunos resultados obtenidos con el lenguaje visual que se implementó en el videojuego.



Pruebas con el lenguaje visual

Para observar el potencial y las capacidades del lenguaje implementado en este videojuego, se realizaron diversos algoritmos utilizando la capacidad para crear y manejar variables y arreglos.

Serie de Fibonacci

En matemáticas, la sucesión o serie de Fibonacci es una sucesión infinita de números naturales que comienza con los números 0 y 1, y a partir de estos dos, cada uno de los siguientes términos de la sucesión es definido: [55]

$$\begin{aligned} F_0 &= 0; \quad F_1 = 1 \\ F_n &= F_{n-1} + F_{n-2} \quad n > 1 \end{aligned}$$

Esta sucesión, a pesar de que fue descrita varios siglos antes por matemáticos Indios [56], debe su nombre al matemático italiano Leonardo Pisano, el cual lo dió a conocer como la solución a la cría de conejos. [57]

El pseudocódigo para generar los primeros n elementos de la serie de fibonacci y guardarlos en un arreglo es el siguiente:

```
fn fibonacci(): -> array

    result[0] := 0
    result[1] := 1

    for (i in 2..n):

        a := result[i - 1]
        b := result[i - 2]
        result[i] := a + b

    return result

end
```

Expresado en el lenguaje visual implementado en el proyecto:

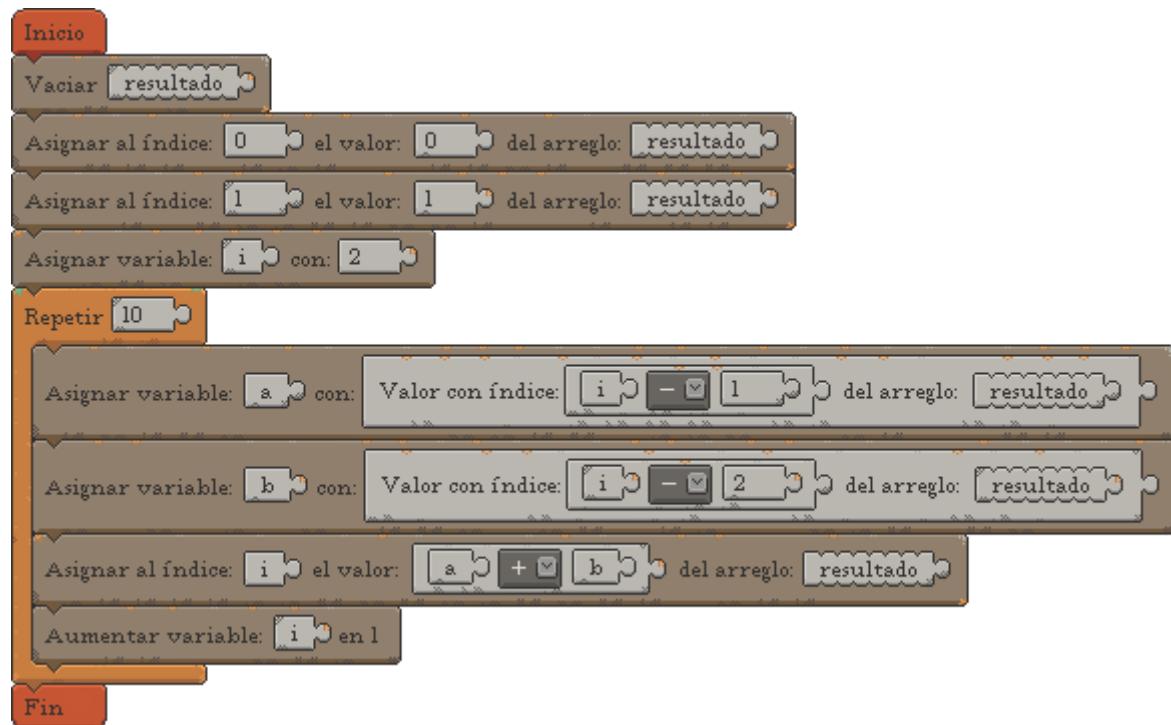


Figura 8.1. Código visual de fibonacci.

Arreglo de salida:

Arreglos	
array_resultado	
0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55
11	89

Figura 8.2. Arreglo resultado.

Ordenamiento por burbuja

Un algoritmo de ordenamiento permite poner elementos en una lista o un vector en una secuencia determinada o colocarlos por una relación de orden [58].

Un algoritmo sencillo para ordenar un arreglo dado es el algoritmo de **ordenamiento de burbuja** [59].

El pseudocódigo de un tipo de algoritmo de ordenamiento burbuja es el siguiente:

```
fn bubbleSort(array): -> array
    longitud := array.length
    for (i in 0..(longitud - 1)):
        for (j in 0..(n - i - 1)):
            if (array[j] > array[j + 1]):
                aux := array[j]
                array[j] := array[j + 1]
                array[j + 1] := aux
    return array
end
```

Arreglo de entrada:

Arreglos	
array_arreglo	
0	1
1	3
2	23
3	32
4	56
5	86

Figura 8.3. Arreglo de entrada.

El pseudocódigo del ordenamiento burbuja expresado en el lenguaje visual implementado del proyecto:

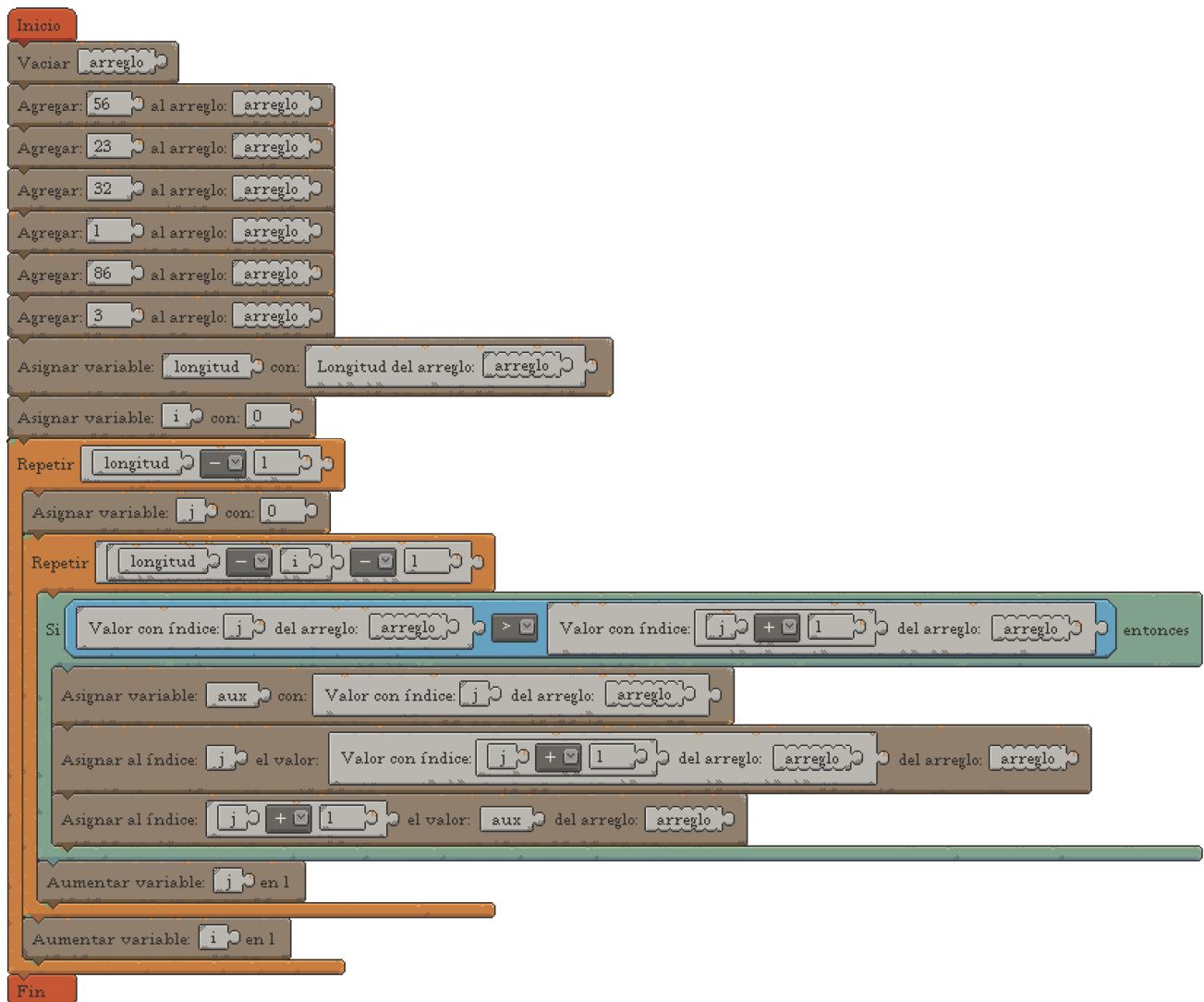


Figura 8.4. Algoritmo de ordenamiento BubbleSort.

Arreglo de salida:

Arreglos	
array_arreglo	
0	1
1	3
2	23
3	32
4	56
5	86

Figura 8.5. Arreglo resultado.

9

Conclusiones y Trabajo a futuro

Las conclusiones finales por parte del equipo de desarrollo. Así como el potencial del proyecto.



Conclusiones generales

Según las expectativas iniciales del proyecto, este sufrió muchas modificaciones a lo largo de su desarrollo, sin perder el objetivo inicial que se planteó y el cual se logró. Además los productos esperados que son la documentación junto con el videojuego con dos niveles y el intérprete fueron completados con éxito aunque no sin dificultades.

El intérprete junto con el conjunto de instrucciones, bloques y expresiones, cumplieron y excedieron las características iniciales planteadas, con la inclusión de variables y arreglos, lo cual proporciona una mayor capacidad al lenguaje visual elaborado en este proyecto.

Respecto a la elaboración de niveles e inclusión de mecánicas en el entorno, dado el tiempo y los recursos disponibles, se vió más limitado su desarrollo; quedando muchas ideas sin implementar, pero con retos y puzzles bastante robustos y con una creciente dificultad que hace uso de las características más interesantes del lenguaje visual.

Finalmente, con relación al fomento de la lógica de solución de problemas en niños mexicanos, confiamos en las bases teóricas que definimos al principio y en el trabajo de múltiples autores, los cuales nos sirvieron de guía para la inclusión y diseño de las diversas mecánicas implementadas; aunque una prueba de campo es necesaria para tener un mayor nivel de certeza.

Conclusiones personales

Alberto Ehad García Barradas

Este ha sido hasta la fecha el tercer proyecto en el que he trabajado usando la herramienta de Unity para el desarrollo de un videojuego y diría también que ha sido el más complejo de todos; la implementación de las mecánicas para el videojuego en muchas ocasiones era tan complicada que requería de bastante planeación previa a siquiera pensar en meter mano en el código, sobretodo porque requería que nos dividiríamos el trabajo, así que tanto mi compañero Joel, como yo, debíamos de saber exactamente qué haríamos y qué haría el otro.

A pesar de no haber conseguido incluir todas las características previamente planeadas debido al tiempo que tuvimos, sí considero que las mecánicas que se alcanzaron a implementar superaron mis expectativas en cuanto a cómo esperaba que se vieran y funcionaran, lo cual me enorgullece de lo que conseguimos.

Esta fue la primera vez en que trabajé con alguien para el desarrollo de un videojuego y diría que estoy bastante satisfecho con la forma en la que dividimos las tareas y en la forma en la que planeamos la implementación de las mecánicas en las que cada quien trabajaría, lo cual llevó a una muy sencilla y ágil conjunción del trabajo que hizo cada uno y se convirtiera en un sistema compuesto que funciona en armonía.

Uno de los desafíos con los que me enfrenté durante el desarrollo del proyecto fue el diseño de los niveles, ya que parecía complicado crear diversos modelos de problemas que lleven al jugador a diferentes tipos de soluciones para cada uno de estos, el llegar a los modelos usados en la creación de los niveles fue un proceso creativo bastante agradable y enriquecedor.

Aprendí muchas lecciones en cuanto a expectativas en un proyecto, administración de tiempo, desarrollo y herramientas para el trabajo en equipo, lo cual aprecio mucho y confío en que me ayudará en proyectos a futuro.

Joel Harim Hernández Javier

El desarrollo de este proyecto fue un reto importante para mí, ya que no tenía experiencia previa con el desarrollo de videojuegos utilizando la herramienta Unity, y personalmente subestimé el tiempo y los recursos necesarios para la creación de ciertos elementos dentro del juego.

Aunque el proyecto al final quedó más pequeño y con menos características que nuestras primeras expectativas, las características que sí incluimos fueron complejas de implementar, aunque parecían sencillas de hacer. Ese fue el caso con la interfaz y el sistema de nodos, que prácticamente quedó en varios meses cuando se había planeado su finalización en algunas semanas.

Cuando llegó el punto en que me sentía cómodo y tenía un cierto dominio trabajando con Unity y las demás herramientas que utilizamos para el proyecto, el tiempo se nos había acabado; y me siento frustrado al respecto por todas las cosas que ahora sé y me hubiera gustado saber desde que comencé el proyecto, lo que me hubiera permitido hacer más y mejores cosas.

Pero el conocimiento obtenido es muy valioso para mí, lo que me permitirá planear mejor los tiempos y recursos requeridos para ciertas tareas en muchas otras áreas fuera del desarrollo de videojuegos.

Trabajo a futuro

A lo largo del desarrollo de este proyecto surgieron ideas y perspectivas nuevas que por limitaciones de tiempo y de recursos no pudieron ser implementadas. Algunas de estas ideas se describen aquí.

Mejor implementación de UI

Como se mencionó antes, al no soportar SpriteShapes el paquete para crear interfaces de Unity, se optó por crear un conjunto de herramientas rudimentario para el control del orden de renderizado y de jerarquía para los nodos y para los elementos de la interfaz de usuario.

Se podría implementar un sistema personalizado de renderizado de formas libres que sustituya a SpriteShapes para obtener mejor rendimiento y mejores características de cara al desarrollo y mantenimiento de los nodos, además de orientar este proceso de dibujo de formas para que sea compatible con Unity UI y todo su conjunto de herramientas.

Entornos más ricos y variados, mecánicas más complejas

Al final los niveles y los elementos dentro de cada escena se vieron bastante limitados por el tiempo necesario para su implementación. Se desarrollarían nuevas mecánicas para aprovechar nuevos elementos en el entorno, como rayos láseres, robots enemigos, paredes móviles, etc.

Visor mejorado

Al principio se pensó en usar el visor para ver los nodos de robots enemigos y poder ver su ejecución. Se agregaría esta mecánica al mismo tiempo de que se le daría más usos, como el descubrimiento de secretos o secciones ocultas.

Mecánicas de Bel más pulidas

Se podrían incorporar mecánicas nuevas al jugador, como la posibilidad de saltar, agacharse, recoger objetos, guardarlos.

Diseños de niveles más completos.

Se tendrían niveles con los puzzles conectados en la misma escena, sin la necesidad de hacer transición entre escenas, para que el nivel se sienta más conectado, grande y diverso.

10

Referencias



Referencias generales

- [1] UNESCO, *E2030: Education and Skills for the 21st Century*, 2017. [Online] Recuperado el 26 de marzo de 2021, de
<http://www.unesco.org/new/fileadmin/MULTIMEDIA/FIELD/Santiago/pdf/Habilidades-SXXI-Buenos-Aires-Eng.pdf>
- [2] Henríquez, C., Sotomayor, C., *Avanzar en las habilidades básicas del siglo XXI*, 2020. [Online] Recuperado el 26 de marzo de 2021, de <https://es.unesco.org/news/avanzar-habilidades-basicas-del-siglo-xxi>
- [3] Rogalski, J., Samurçay, R., *Acquisition of Programming Knowledge and Skills*, 2019. [Online] Recuperado el 26 de abril de 2021 en <https://www.cl.cam.ac.uk/teaching/1011/R201/ppig-book/ch2-4.pdf>
- [4] OCDE, *Programa para la evaluación internacional de alumnos (PISA)*, PISA 2018 - Resultados, 2018. [Online] Recuperado el 2 de abril de 2021 en
https://www.oecd.org/pisa/publications/PISA2018_CN_MEX_Spanish.pdf
- [5] Sáez, J., Cózar, R., *Programación visual por bloques en Educación Primaria: Aprendiendo y creando contenidos*, Revista Complutense de Educación, 2015.
- [6] Aldama, C., Zavala, M., *MECANISMO PROGRAMABLE PARA NIÑOS*, Tesis (Ingeniería en Sistemas Computacionales), Instituto Politécnico Nacional, ESCOM, 2016.
- [7] Aguirre, J., Dzul, Román., *Desarrollo de un videojuego didáctico para la enseñanza de conjuntos y sus operaciones con JAVA 2D*, Tesina, Instituto Politécnico Nacional, UPIICSA, 2010.
- [8] Marji, Majed, *Learn to Program with Scratch*. San Francisco, California: No Starch Press, 2014.
- [9] Scratch, *About Scratch*. [Online] Recuperado el 2 de abril de 2021 en <https://scratch.mit.edu/>
- [10] CodeMonkey, *Plan options*. [Online] Recuperado el 2 de abril de 2021 en
<https://app.codemonkey.com/plans>
- [11] Codecombat, *Legal, Copyrights and Licenses*. [Online] Recuperado el 3 de abril de 2021 en
<https://codecombat.com/legal#:~:text=Cost,100%25%20money%2Dback%20guarantee.>
- [12] Grasshopper, *Preguntas frecuentes*. [Online] Recuperado el 2 de abril de 2021 en
https://grasshopper.app/es_419/faq/
- [13] Real Academia Española, *Pixel*, Diccionario panhispánico de dudas. [Online] Recuperado el 30 de octubre de 2021 de <https://www.rae.es/dpd/pixel>
- [14] Juan Carlos, *¿Qué es el Pixel Art?*, Tokio School, 2021. [Online] Recuperado el 30 de octubre de 2021 de <https://www.tokioschool.com/noticias/que-es-el-pixel-art/>
- [15] Technopedia, *What Does Pixel Art Mean?* [Online] Recuperado el 30 de octubre de 2021 de
<https://www.techopedia.com/definition/8884/pixel-art>

- [16] MuBert, F., *La belleza del Pixel Art: 31 juegazos pixelados que son un festín para los ojos*. Vida Extra, 2020. [Online] Recuperado el 30 de octubre de 2021 de
<https://www.vidaextra.com/listas/belleza-pixel-art-juegazos-pixelados-que-festin-para-ojos>
- [17] Ahmed, S., *The Best Games To Play If You Love Pixel Art*. GameRant, 2021. [Online] Recuperado el 30 de octubre de 2021 de <https://gamerant.com/games-play-love-pixel-art/>
- [18] Steam, *Hyper Light Drifter*, 2016. [JPG] [Online] Recuperado el 30 de octubre de 2021 de
https://store.steampowered.com/app/257850/Hyper_Light_Drifter/
- [19] Steam, *Moonlighter*, 2018. [JPG] [Online] Recuperado el 30 de octubre de 2021 de
<https://store.steampowered.com/app/606150/Moonlighter/>
- [20] Steam, *Terraria*, 2018. [JPG] [Online] Recuperado el 30 de octubre de 2021 de
<https://store.steampowered.com/app/105600/Terraria/>
- [21] Steam, *Undertale*, 2018. [JPG] [Online] Recuperado el 30 de octubre de 2021 de
<https://store.steampowered.com/app/391540/Undertale/>
- [22] Steam, *Celeste*, 2018. [JPG] [Online] Recuperado el 30 de octubre de 2021 de
<https://store.steampowered.com/app/504230/Celeste/>
- [23] Gaitán, V., *Gamificación: el aprendizaje divertido*, 2013. [Online] Recuperado el 12 de septiembre de 2021 en <https://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/>.
- [24] Hamari, J., *Gamification*, 2019. [Online] Recuperado el 2 de abril de 2021 en
<https://onlinelibrary.wiley.com/doi/abs/10.1002/9781405165518.wbeos1321>
- [25] Zichermann, G., Cunningham, C., *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps* Sebastopol. California. O'Reilly Media, 2011.
- [26] Costa. C., *Gamification*. Organizational Architect and Engineer Journal, 2019. [Online] Recuperado el 21 de septiembre de 2021 en <https://oae.pubpub.org/pub/e1mxx7kh/release/1>
- [27] Costa. J, Wehbe, R., Robb, J., Nacke, L., *Time's Up: Studying Leaderboards For Engaging Punctual Behaviour*. ResearchGate, Gamification 2013 Conference, 2013. [Online] Recuperado el 20 de septiembre de 2021 en
https://www.researchgate.net/publication/257519212_Time's_Up_Studying_Leaderboards_For_Engaging_Punctual_Behaviour.
- [28] Budde, R., Jost, B., Ketterl, M., Leimbach, T., *Graphical Programming Environments for Educational Robots: Open Roberta - Yet another One?*. IEEE International Symposium on Multimedia 2014, 2014. [Online] Recuperado el 15 de septiembre de 2021 en
https://www.researchgate.net/publication/269397166_Graphical_Programming_Environment_s_for_Educational_Robots_Open_Roberta_-_Yet_Another_One

- [29] Repenning, A., *Moving Beyond Syntax: Lessons from 20 Years of Blocks Programming in AgentSheets*. University of Colorado, 2017. [Online] Recuperado el 14 de septiembre de 2021 en http://ksiresearchorg.ipage.com/vlss/journal/vlss2017/vlss17paper_10.pdf.
- [30] Piaget, *Teoría del desarrollo cognitivo de Piaget*. [Online] Recuperado el 22 de septiembre de 2021 en <https://www.terapia-cognitiva.mx/wp-content/uploads/2015/11/Teoria-Del-Desarrollo-Cognitivo-de-Piaget.pdf>.
- [31] Belloch, O., *LAS TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN (T.I.C.)*. Universidad de Valencia. [Online] Recuperado el 4 de octubre de 2021 en <https://www.uv.es/~bellochc/pdf/pwtic1.pdf>
- [32] Betancourt, J., *Estrategias didácticas innovadoras: Recursos para maestros y alumnos del siglo 21*. [Online] Recuperado el 4 de abril de 2021 en <https://estrategiasdidacticassite.files.wordpress.com/2017/03/libro.pdf>
- [33] Meerbaum-Salant, O., Armoni, M. y Ben-Ari, M., *Learning computer science concepts with scratch*. Computer Science Education, 23(3), 239–264, 2013.
- [34] Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M. y Rusk, N., *Programming by choice: urban youth learning programming with scratch (Vol. 40)*. New York, NY: ACM, 2008.
- [35] Alonso, D., *Scratch como herramienta para la enseñanza de la programación en la Educación Primaria*. Tesis Doctoral, Universidad Camilo José Cela, Madrid, 2017.
- [36] UNICEF, *Mission #4 - Problem Solving*. [Online] Recuperado el 10 de noviembre de 2021 en <https://www.unicef.org/lac/en/mission-4-problem-solving>
- [37] Tichon, J.G., T. Mavin, *Experiencing Resilience via Video Games*. Social Science Computer Review, 2016, p. 1–10.
- [38] Ventura, M., V. Shute, W. Zhao, *The Relationship between Video Game Use and a Performance-Based Measure of Persistence*. Computers & Education, 2013, p. 52–58.
- [39] IOM Human Resources, What Is Resilience?, 2018. [Online] Recuperado el 10 de noviembre de 2021 en <https://www.iom.int/sites/g/files/tmzbdl486/files/staff-welfare/resilience.pdf>
- [40] Linares, R., *Resiliencia o la adversidad como oportunidad*, Ediciones Espuela de Plata, ISBN 978-8-416-03495-6.
- [41] Osorio, D., *¿Cómo se hace un concepto de juego?*. Asociación de Estudiantes de Videojuegos, 2014. [Online] Recuperado el 6 de octubre de 2021 en <https://aev.org.es/como-se-hace-un-concepto-de-juego>
- [42] Unity, *Say hello to Unity Gaming Services*. [Online] Recuperado el 29 de octubre de 2021 en <https://unity.com/>
- [43] Unity, *Build once, deploy anywhere*. [Online] Recuperado el 29 de octubre de 2021 en <https://unity.com/features/multiplatform>

- [44] Unity, *Plans and pricing*. [Online] Recuperado el 29 de octubre de 2021 en <https://store.unity.com/#plans-individual>
- [45] Drake, J., *10 Great Games That Use The Unity Game Engine*. TheGamer, 2020. [Online] Recuperado el 29 de octubre de 2021 en <https://www.thegamer.com/unity-game-engine-great-games/>
- [46] Capello, D., *Aseprite README.md*. Repositorio en Github, 2021. [Online] Recuperado el 29 de octubre de 2021 en <https://github.com/aseprite/aseprite>
- [47] Steam, *Aseprite on Steam*. [Online] Recuperado el 29 de octubre de 2021 en <https://store.steampowered.com/app/431730/Aseprite/>
- [48] Visual Studio, *Visual Studio 2019*. [Online] Recuperado el 29 de octubre de 2021 en <https://visualstudio.microsoft.com/es/vs/>
- [49] Mozko, H., *¿Cuál es la diferencia entre Visual Studio y Visual Studio Code?*. Alfonso Mozko H. | Un estudiante de informática, 2018. [Online] Recuperado el 29 de octubre de 2021 en <https://alfonsomozkoh.github.io/2018/08/31/cual-es-la-diferencia-entre-visual-studio-y-visual-studio-code.html>
- [50] Visual Studio, *Comparar las ediciones de Visual Studio 2019*. [Online] Recuperado el 29 de octubre de 2021 en <https://visualstudio.microsoft.com/es/vs/compare/>
- [51] Visual Studio Code, *Code editing. Redefined*. [Online] Recuperado el 29 de octubre de 2021 en <https://code.visualstudio.com/>
- [52] Paint.NET, *About Paint.NET*. [Online] Recuperado el 30 de octubre de 2021 en <https://www.getpaint.net/>
- [53] Acerenza, N., Coppes, A., Mesa, G., Viera A., Fernández, E., Laurenzo, T., Vallespir D., *Una Metodología para el Desarrollo de Videojuegos*, 2009. [Online] Recuperado el 6 de abril de 2021 en https://www.fing.edu.uy/sites/default/files/biblio/22811/asse_2009_16.pdf
- [55] Unity, *Unity UI: Unity User Interface*, s.f. [Online] Recuperado el 9 de mayo de 2022 en <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html>
- [54] Unity, *2D Sprite Shape Overview*, s.f. [Online] Recuperado el 9 de mayo de 2022 en <https://docs.unity3d.com/Packages/com.unity.2d.spriteshape@3.0/manual/index.html>
- [55] Sloane, N. J. A., *Sequence A000045*, The On-Line Encyclopedia of Integer Sequences. [Online] Recuperado el 27 de mayo de 2022 en <https://oeis.org/A000045>
- [56] Singh, Parmanand, *The So-called Fibonacci numbers in ancient and medieval India*, Historia Mathematica 12, 1985. [Online] Recuperado el 27 de mayo de 2022 en <https://www.sciencedirect.com/science/article/pii/0315086085900217?via%3Dihub>

[57] BBC News Mundo, *Fibonacci, el matemático que se puso a contar conejos y descubrió la secuencia divina.* [Online] Recuperado el 27 de mayo de 2022 en
<https://www.bbc.com/mundo/noticias-46926506>

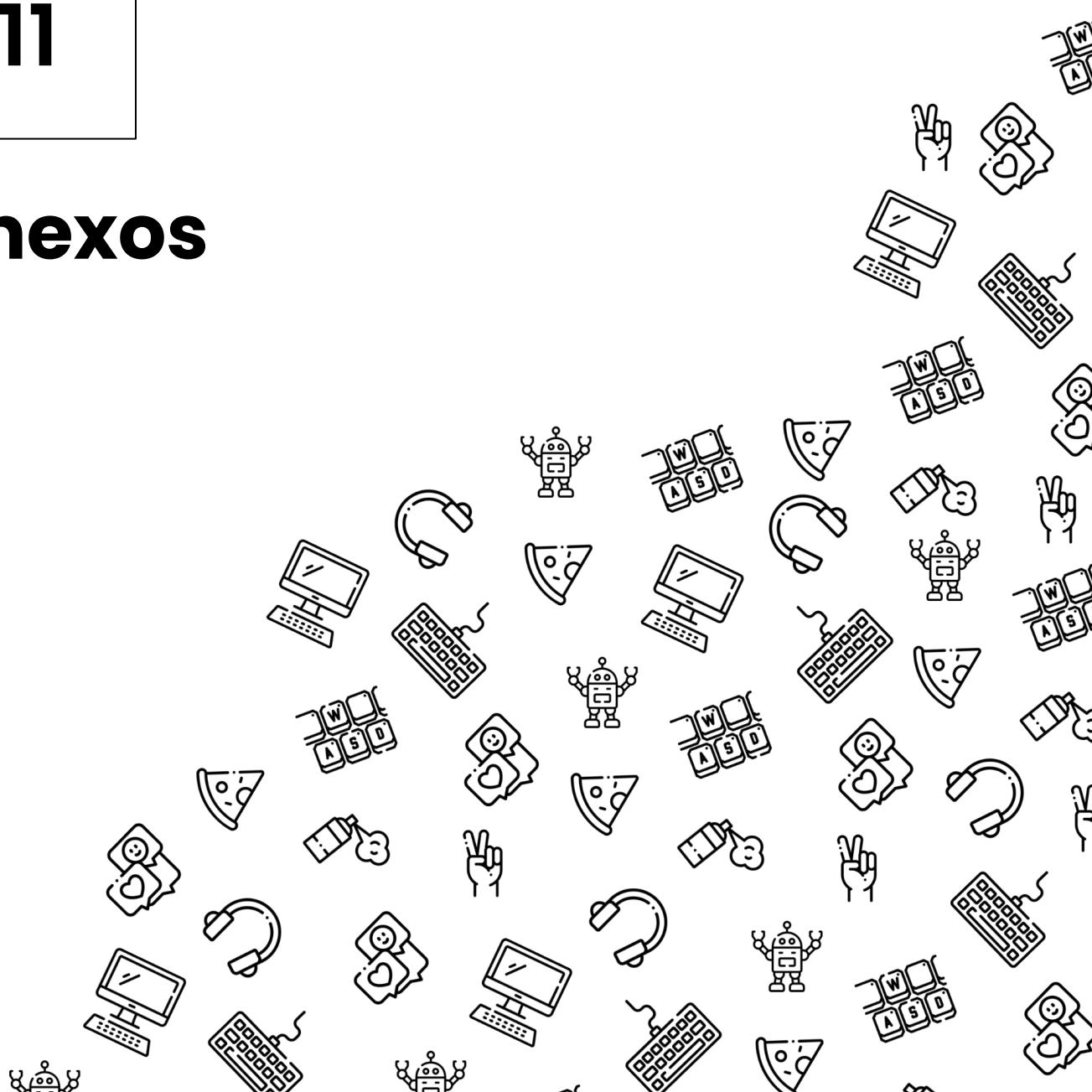
[58] Knuth, Donald E., *Sorting and Searching, The Art of Computer Programming, vol. 3* (2nd ed.), Boston: Addison-Wesley, 1998, ISBN 0-201-89685-0

[59] Astrachan, Owen. *Bubble Sort: An Archaeological Algorithmic Analysis.* SIGCSE, 2003. [Online] recuperado el 27 de mayo de 2022 en <http://www.cs.duke.edu/~ola/papers/bubble.pdf>

Íconos de portada diseñados por **Freepik**.

11

Anexos





Instituto Politécnico Nacional

Escuela Superior de Cómputo



ESCOM

Trabajo Terminal

“Videojuego de Resolución de Desafíos a través de la Programación Visual para el Fomento del Desarrollo de la Lógica de Solución de Problemas (Scrap Coder)”

No. 2021-A026

Manual de Usuario

Presentan

Alberto Ehad García Barradas

Joel Harim Hernández Javier

Director

Dr. Yaxkin Flores Mendoza

INSTITUTO POLITÉCNICO NACIONAL



Índice

Índice	2
Historia	3
Personajes	3
Cinemática	3
Controles	4
Movimiento	4
Interactuar	4
Inspección	4
Volver al menú principal	4
Reiniciar nivel	4
Menú Principal	4
Nueva Partida	5
Seleccionar Nivel	5
Controles	5
Salir del Juego	7
Sobre	7
Nivel	8
Pantalla de título	8
Objetivo	8
Elementos	8
Programación Visual	10
¿Cómo funciona?	10
¿Qué piezas tenemos disponibles?	11

Historia

Personajes

Bel (Jugador) : Una joven y brillante maquinista, habitante del asentamiento. A pesar de su baja estatura, es una chica valiente y decidida. El jugador controla a Bel dentro del juego para navegar y solucionar problemas.

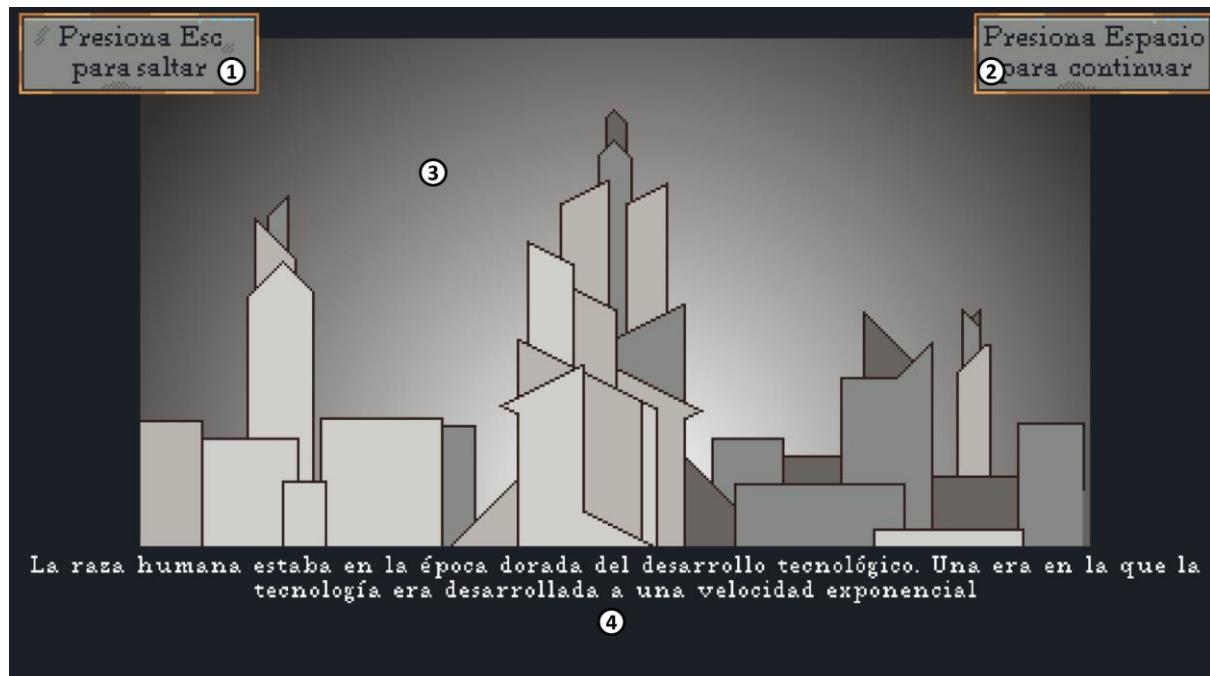


Copper (Robot) : Un robot programable que por eventos del destino terminó a un lado de Bel la ingeniera convirtiéndose en su leal acompañante. El jugador es capaz de programar a Copper para hacerlo ejecutar tareas que lo ayuden a superar los niveles.

Copper puede realizar acciones dentro del mundo como caminar, girar, leer valores e ingresar valores en un panel. Además de esto es capaz de usar variables y arreglos en solución de problemas.

Cinemática

Hay una cinemática al principio del juego, la cual es posible ver al seleccionar "Nueva Partida" en el menú principal. La cinemática narra la historia previa a los eventos dentro del juego, ésta está conformada por 11 imágenes junto con sus respectivas líneas que describen los sucesos.

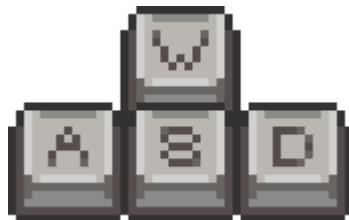


1. Indicación para saltar cinemática.
2. Indicación para avanzar.
3. Imagen de los eventos.
4. Narración de los eventos.

Controles

Movimiento

El jugador es capaz de desplazarse en el Menú Principal y en los niveles haciendo uso de las flechas, así como las teclas **W, A, S, D**; El movimiento dentro de los niveles está limitado dentro del área designada del nivel, para evitar que el jugador acceda a zonas fuera de la intención del flujo de los niveles original.



Interactuar

El jugador puede interactuar tanto con palancas como con Copper usando la tecla **E**. Además de esto, también funciona para seleccionar opciones en el menú principal.



Inspección

El jugador puede inspeccionar los niveles usando la tecla **Q**, esto le permite ver los cables que conectan componentes para ayudarlo a entender qué es lo que tiene que hacer para solucionar el nivel.



Volver al menú principal

El jugador puede volver al menú principal en cualquier momento presionando la tecla **Esc**. De esta manera puede consultar los controles de manera rápida, cargar un nivel en particular, consultar los controles o cerrar el juego de manera rápida.



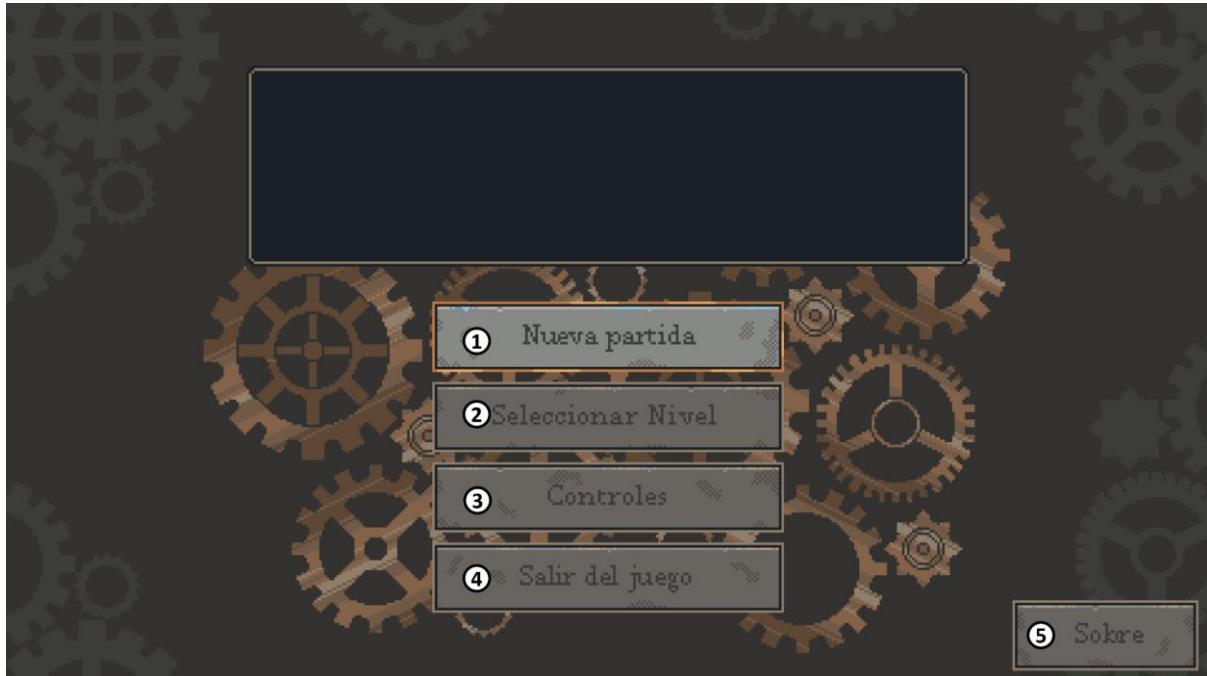
Reiniciar nivel

El jugador puede reiniciar el nivel en el que se encuentra en cualquier momento usando la tecla **R**, así si comete un error puede reiniciar rápidamente y no tener que regresar a Copper a su estado inicial.



Menú Principal

El menú principal da al jugador acceso a 5 diferentes opciones. El jugador es capaz de navegar entre estas opciones usando las flechas y seleccionando una de éstas presionando la tecla **E**.



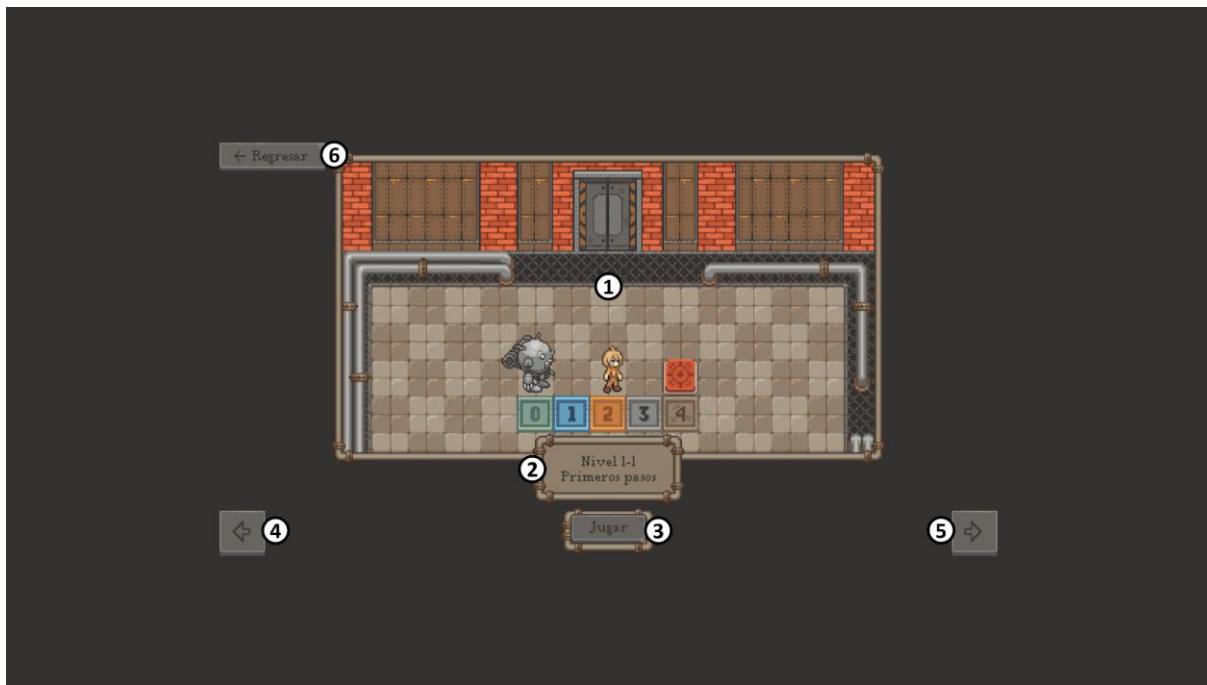
1. Nueva partida.
2. Seleccionar Nivel.
3. Controles.
4. Salir del juego.
5. Sobre.

Nueva Partida

Seleccionar la opción de “Nueva Partida” llevará al jugador a la escena de la cinemática donde se narran los eventos previos a lo que ocurre en el juego. Una vez terminado esto, se inicia el Nivel 1-1.

Seleccionar Nivel

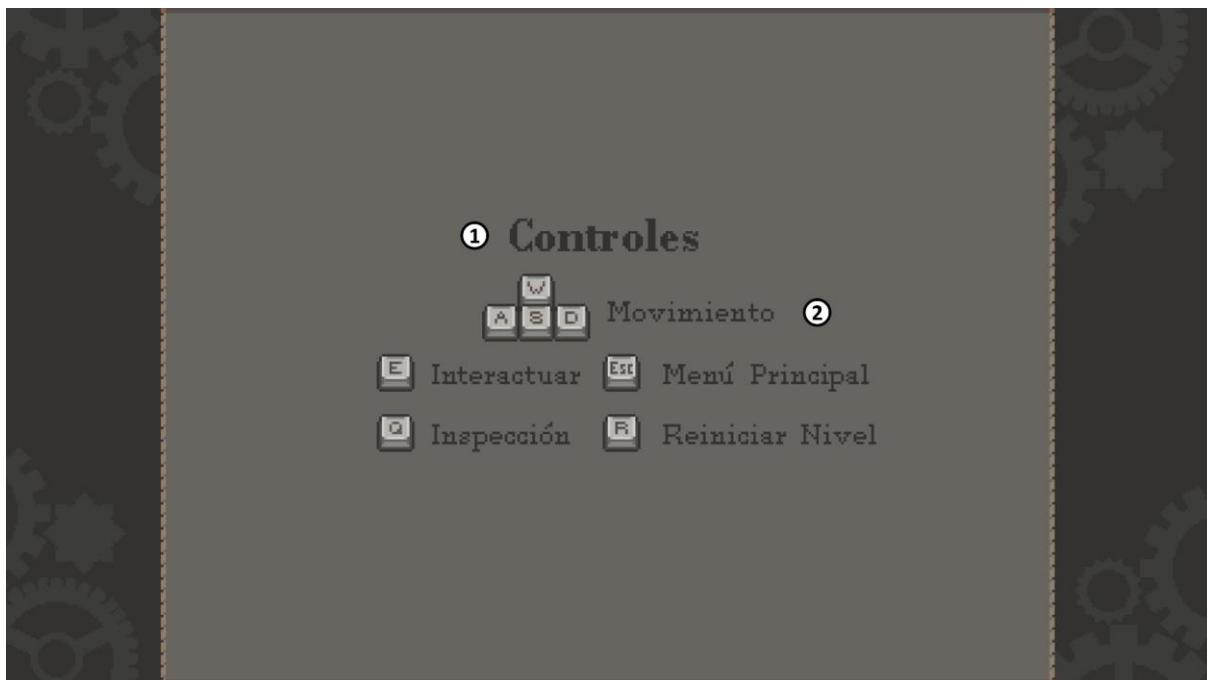
Seleccionar la opción de “Seleccionar Nivel” lleva al jugador a una pantalla en la que puede seleccionar qué nivel quiere jugar. Los niveles a los cuales el jugador no ha llegado permanecerán bloqueados hasta entonces. Aquí el jugador deberá de hacer uso del mouse para seleccionar la opción que quiera realizar.



1. Captura del nivel.
2. Título del nivel.
3. Botón para Jugar.
4. Botón para moverse al nivel anterior.
5. Botón para moverse al siguiente nivel.
6. Botón para regresar al menú principal.

Controles

La pantalla de controles muestra los controles predeterminados del juego. Aquí la única acción que el jugador puede hacer es volver al menú principal presionando **Esc**.



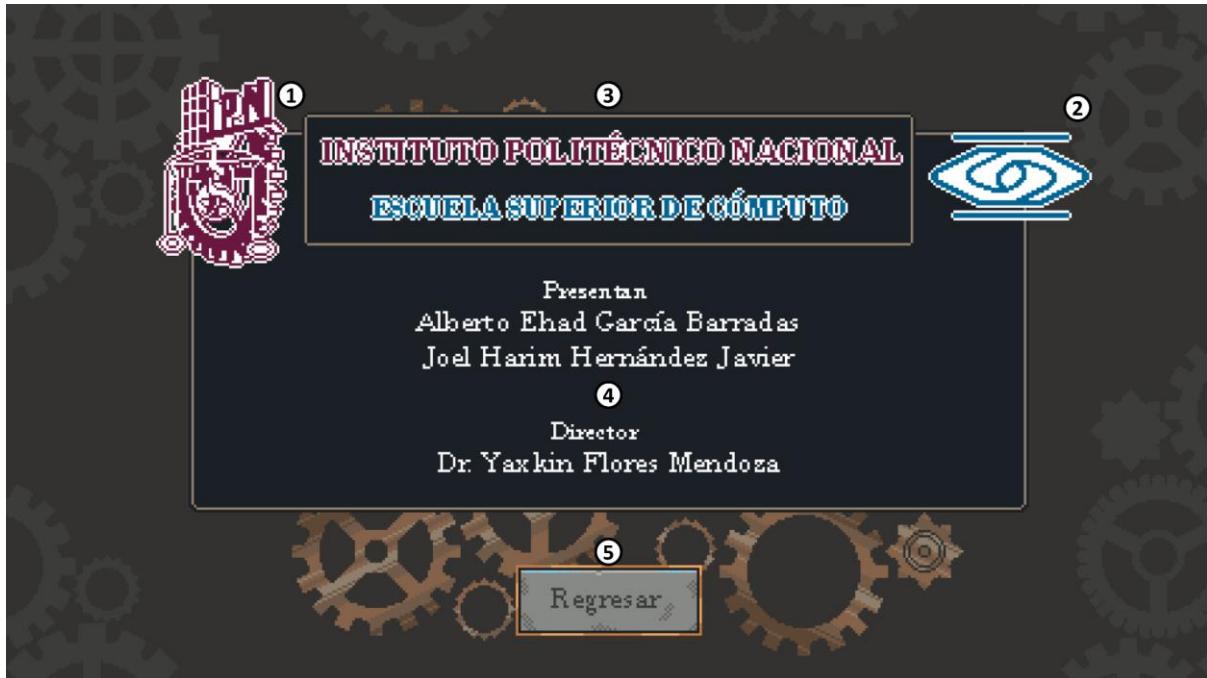
1. Título
2. Controles

Salir del Juego

Seleccionar esta opción finaliza la ejecución del juego.

Sobre

Esta opción muestra una pantalla con información sobre los creadores del juego y la institución para la cual se hizo.

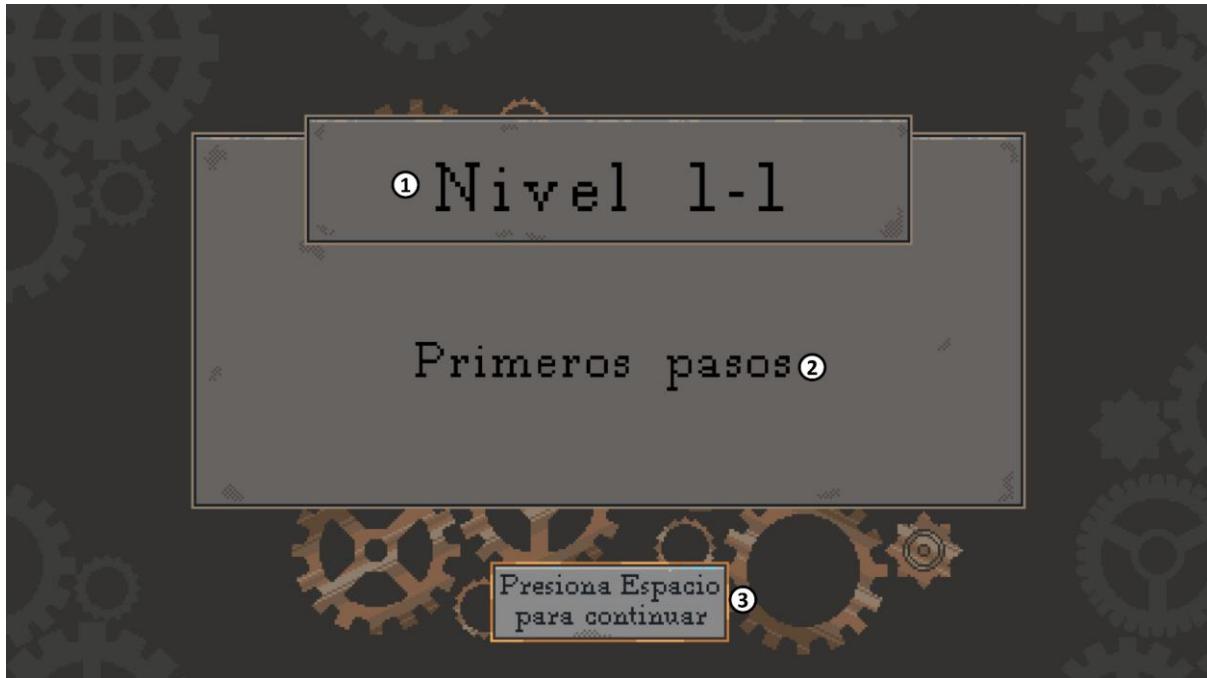


1. Logo del Instituto Politécnico Nacional en pixelart.
2. Logo de la Escuela Superior de Cómputo en pixelart.
3. Nombres de la institución y la escuela.
4. Nombres de presentadores y el director.
5. Botón para regresar al menú principal.

Nivel

Pantalla de título

Al iniciar un nivel lo primero que el jugador verá es la pantalla de título, la cual incluye una breve descripción del nivel, así como su numeración.



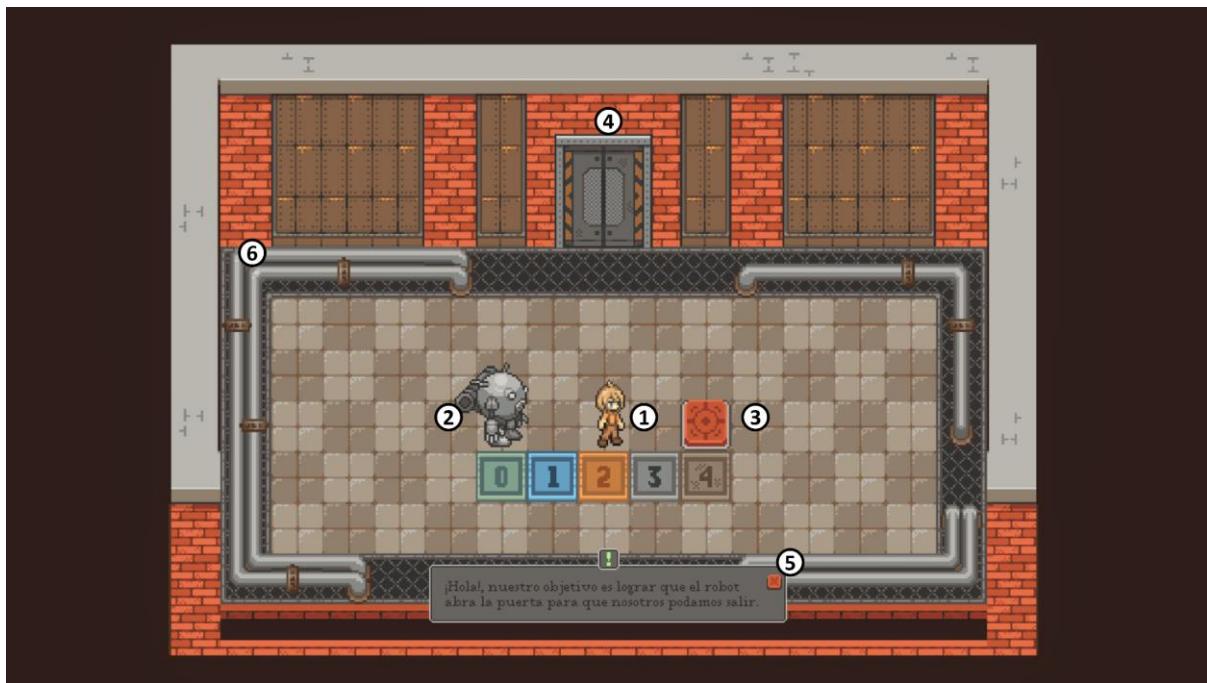
1. Título.
2. Descripción / Pista.
3. Indicación para continuar.

Objetivo

El objetivo en cada nivel es el mismo, abrir la puerta que lleva al siguiente, para esto el jugador deberá de analizar el escenario, ya sea de manera normal o con el modo de inspección y encontrar cuál es la fuente de energía que abre la puerta final y elaborar un plan que lo lleve a que ésta se encuentre abierta al final del nivel. Para esto necesitará en la mayoría de los casos la ayuda de su compañero Copper, quien puede recorrer una ruta mientras que al mismo tiempo el jugador recorre otra o presionando botones para permitir al jugador continuar.

Elementos

Cada nivel está conformado por múltiples elementos, entre ellos el jugador, Copper, los límites del mapa, interruptores o botones, el objetivo del jugador y tutoriales.



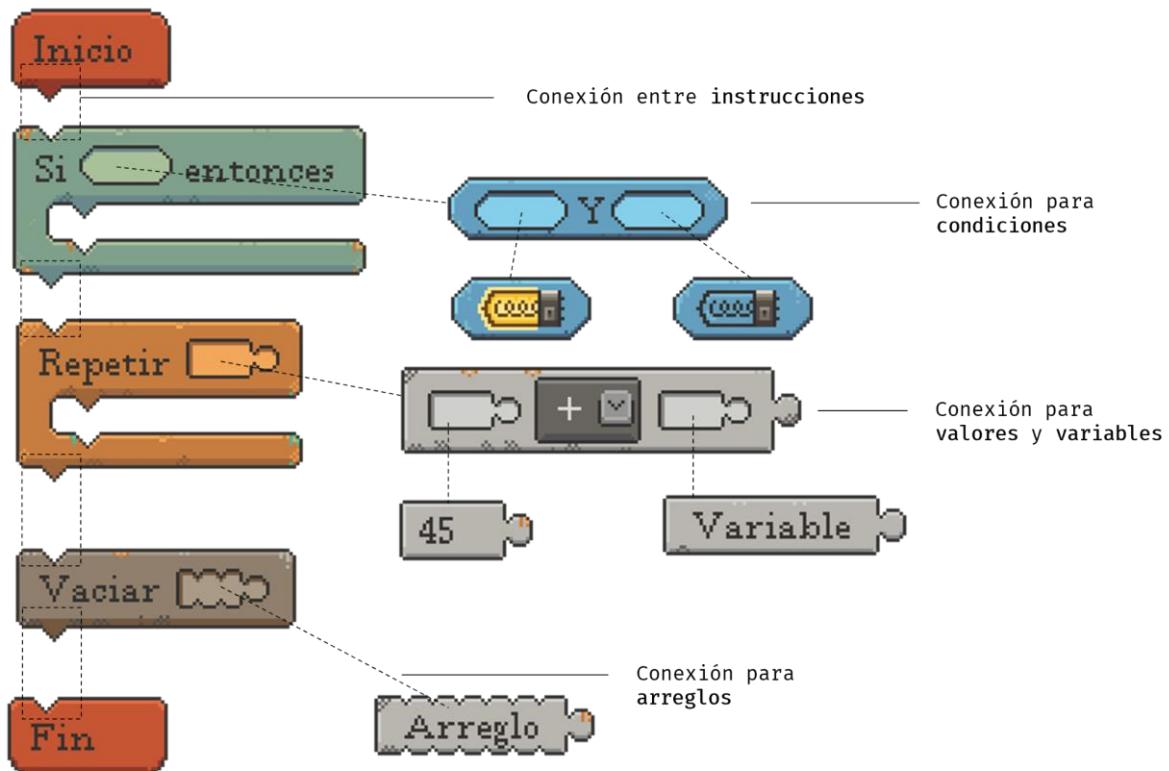
1. Bel (Jugador).
2. Copper.
3. Botón para abrir la puerta.
4. Puerta objetivo del jugador.
5. Tutorial pop-up.
6. Paredes y tubos que delimitan el espacio del escenario.

Programación Visual

¿Cómo funciona?

Los nodos son como piezas lego que se conectan entre sí en lugares específicos. Existen 5 tipos de conexiones que aceptan solamente un tipo de nodo en específico (o hasta dos).

Tenemos los **nodos normales o de instrucción**, los cuales se conectan entre sí de arriba hacia abajo, uniendo la parte posterior de un nodo con la parte superior de otro a través de una unión magnética.



Tenemos los **nodos de condición**, los cuáles siempre son evaluados a Verdadero o Falso, y que te permiten hacer preguntas lógicas para saber si algo es cierto o no.

Tenemos los **nodos de valor**, los cuáles siempre son evaluados a valores numéricos y que te permiten acceder a valores numéricos, realizar operaciones y comparaciones numéricas.

Tenemos las **variables**, las cuales también se pueden conectar a los valores, las cuáles son contenedores de información a los que podemos acceder siempre que queramos.

También tenemos los **arreglos**, los cuales son contenedores con información organizada, son listas de números que se pueden acceder mediante un índice o contador de posición, la cual comienza en 0.

Todas estas piezas te permitirán programar al robot controlando el flujo de información.

¿Qué piezas tenemos disponibles?



Inicio: Permite indicar el comienzo de tu algoritmo.



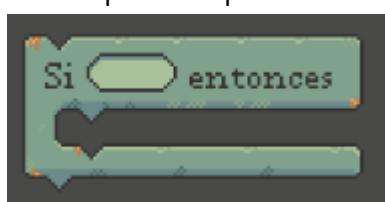
Fin: Permite indicar el fin de tu algoritmo.



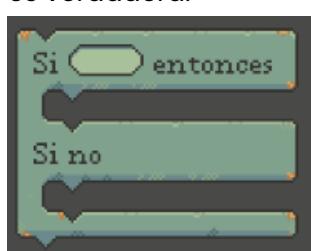
Repetir si: Repite las instrucciones que se encuentren dentro mientras la condición sea verdadera.



Repetir N veces: Repite las instrucciones que se encuentran dentro el número de veces que se indique. Si se pone 0 o un número negativo, entonces no se hará ninguna repetición.



Si, entonces: Ejecuta las instrucciones que se encuentran dentro una sola vez si la condición es verdadera.



Si, entonces, si no: Ejecuta las instrucciones que se encuentran en la primera parte solo una vez si la condición es verdadera, si no, ejecuta solo una vez las instrucciones que se encuentran en la segunda parte.



Constante Verdadera: Condición que siempre evalúa a Verdadero.



Constante Falsa: Condición que siempre evalúa a Falso.



Negación: Permite negar el valor de una condición.



Operación condicional Y: Pregunta por ambas condiciones dadas. Si ambas son verdaderas, devuelve Verdadero, de lo contrario devuelve Falso.



Operación condicional O: Pregunta por ambas condiciones dadas. Si alguna de ellas es verdadero o ambas, devuelve Verdadero, de lo contrario devuelve Falso.



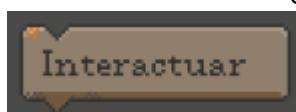
Comparación numérica: Realiza una comparación numérica y devuelve Falso o Verdadero dependiendo de los valores dados.



Caminar: Indica al robot caminar una unidad.



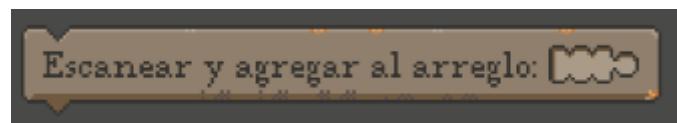
Girar: Indica al robot girar hacia alguna dirección.



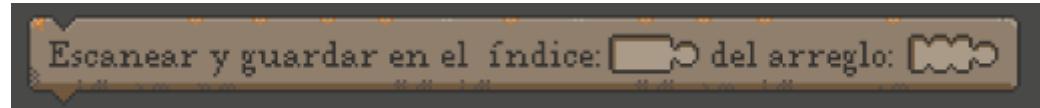
Interactuar: Indica al robot interactuar con el entorno.



Escanear y guardar en variable: Indica al robot escanear un panel y que guarde su valor en una variable.



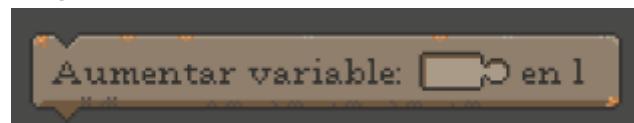
Escanear y agregar al arreglo: Indica al robot escanear un panel y que añada su valor en un arreglo.



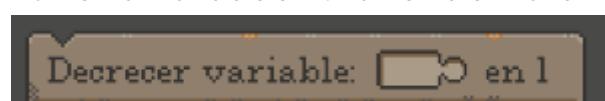
Escanear y guardar en el índice del arreglo: Indica al robot escanear un panel y que guarde en un índice en concreto de un arreglo ese valor.



Asignar variable: Establece una variable a un valor en concreto.



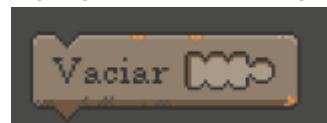
Aumentar variable en 1: Aumenta en 1 una variable dada.



Decrecer variable en 1: Decrece en 1 una variable dada.



Agregar valor al arreglo: Agrega un valor dado a un arreglo dado.



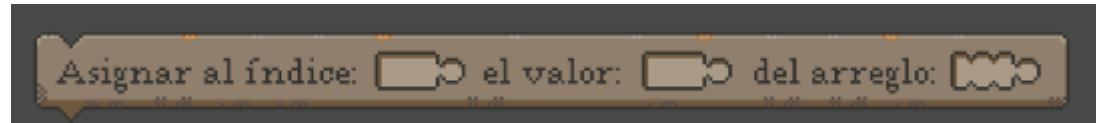
Vaciar arreglo: Elimina todos los valores de un arreglo.



Ordenar arreglo: Ordena de menor a mayor los valores de un arreglo.



Eliminar valor con índice del arreglo: Permite eliminar valor del arreglo.



Asignar al índice de un arreglo un cierto valor: Asigna al índice concreto de un arreglo dado un cierto valor. Si ese índice no existe, los elementos entre el último elemento y el índice son creados y establecidos en 0.



Insertar valor en un índice de un arreglo: Inserta un valor en el índice indicado del arreglo dado.



Ingresar valor a panel numérico: Ingresa un valor numérico a un panel numérico.



Campo numérico: Permite ingresar un número escrito por el jugador.



Operación numérica: Permite realizar una operación numérica entre dos valores y devuelve el resultado.



Valor del índice del arreglo: Devuelve el valor guardado del arreglo dado en el índice indicado.



Longitud del arreglo: Obtiene el número de elementos guardados en el arreglo dado.