UNIVERSIDAD DE GUAYAQUIL

FACULTAD:

CIENCIAS MATEMÁTICAS Y FÍSICAS

CARRERA:

SOTFWARE

ASIGNATURA:

PROCESO DE SOFTWARE

CURSO:

3-1

ESTUDIANTE:
GAME GONZÁLEZ MANUEL ENRIQUE JEREMY
LLAMBO WILLIAMS GONZÁLES
JEAN CARLOS RAMOS ESPINOZA BYRON
LÓPEZ JOSEPH LEÓN
JUSTIN MARTINEZ JOSEPH ELLIS

DOCENTE:

ING. REYES WAGNIO MANUEL FABRICIO

1. Análisis de requisitos

1.1. Descripción del propósito educativo

El propósito del juego es reforzar y evaluar conocimientos sobre datos culturales, históricos y generales del Ecuador mediante una dinámica interactiva. El juego integra mecánicas de plataforma 2D con una actividad educativa final: responder una pregunta relacionada con el país. Con esto se busca:

- Incentivar el aprendizaje lúdico.
- Motivar a los estudiantes a repasar conocimientos mientras juegan.
- Evaluar de manera divertida el aprendizaje adquirido.

El juego combina exploración, recolección de objetos y resolución de preguntas, fomentando la concentración, la memoria y la agilidad cognitiva.

1.2. Público objetivo

- Nivel educativo: Estudiantes de educación básica media o bachillerato.
- Edad recomendada: Entre 10 y 17 años.
- Conocimientos previos: Habilidades básicas para manejar un teclado y nociones generales sobre temáticas del Ecuador (historia, geografía, cultura, etc.).

1.3. Requisitos funcionales

Los requisitos funcionales describen lo que el sistema debe hacer:

Requisitos Funcionales	
RF01	El sistema debe permitir al jugador controlar un personaje en un entorno de plataforma 2D.
RF02	El personaje debe poder moverse, saltar y recorrer el nivel.
RF03	El juego debe generar 10 monedas distribuidas por el nivel.
RF04	El jugador debe poder recoger las monedas.
RF05	Una vez recogidas las 10 monedas, debe aparecer una llave en el nivel.
RF06	El jugador debe poder recoger la llave.
RF07	El juego debe incluir una puerta que solo pueda abrirse si el jugador tiene la llave.
RF08	Al abrir la puerta, debe mostrarse una pregunta relacionada con el Ecuador.
RF09	El jugador debe seleccionar una respuesta entre las opciones disponibles.
RF10	El juego debe verificar si la respuesta es correcta.
RF11	Si la respuesta es correcta, el sistema debe mostrar un mensaje de éxito y permitir avanzar al siguiente nivel.
RF12	Si la respuesta es incorrecta, el nivel debe reiniciarse desde el inicio.

RF13	El juego debe registrar si se completó el nivel (opcional si lo piden).

Requisitos No Funcionales	
RF01	El juego debe ser fácil de entender para niños y adolescentes. Los controles deben ser simples e intuitivos.
RF02	Debe ejecutarse de forma fluida en computadores de bajos recursos (por ejemplo, 4GB RAM, CPU básica).
RF03	El sistema debe reiniciar correctamente el nivel sin errores al fallar la pregunta.
RF04	El código debe permitir agregar nuevas preguntas o niveles en el futuro sin reescribir la lógica principal.
RF05	El juego debe poder compilarse y ejecutarse en Windows o navegadores (si fue hecho en Unity, Godot, GameMaker, o en este caso GDevelop).
RF06	Los textos deben ser claros, legibles y con buen contraste para los estudiantes.
RF07	Poder enseñar con las preguntas datos sobre el Ecuador.

1.5. Identificación de actores y escenarios

Actor	Descripción	Rol en el sistema
Jugador	Usuario o cliente que controla al personaje.	Interactúa con el juego, recoge objetos, responde la pregunta.
Sistema / Juego	Encargado de gestionar los niveles.	Entidad que controla la lógica interna, muestra preguntas, valida respuestas y administra niveles.
Desarrollador	Personas que se encargan de los niveles, preguntas y código.	Mantienen y mejoran el juego.

Escenarios principales Escenario 1:

Recolección de monedas

1. El jugador entra al nivel.
2. Recorre el mapa recogiendo las 10 monedas.
3. Cuando obtiene todas, se habilita la llave.

Escenario 2: Obtención de la llave

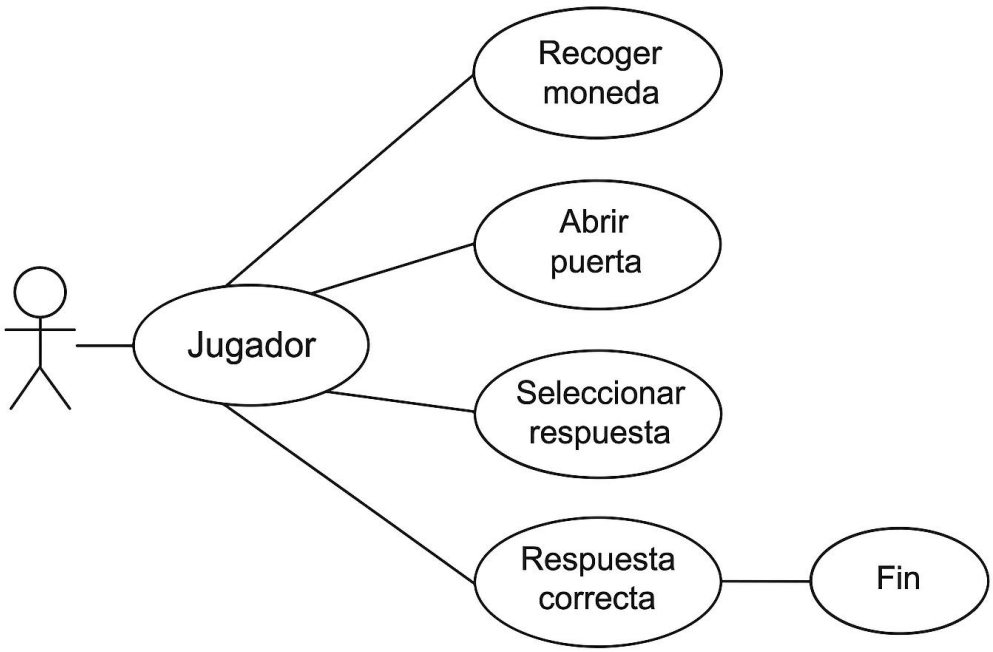
1. La llave aparece.

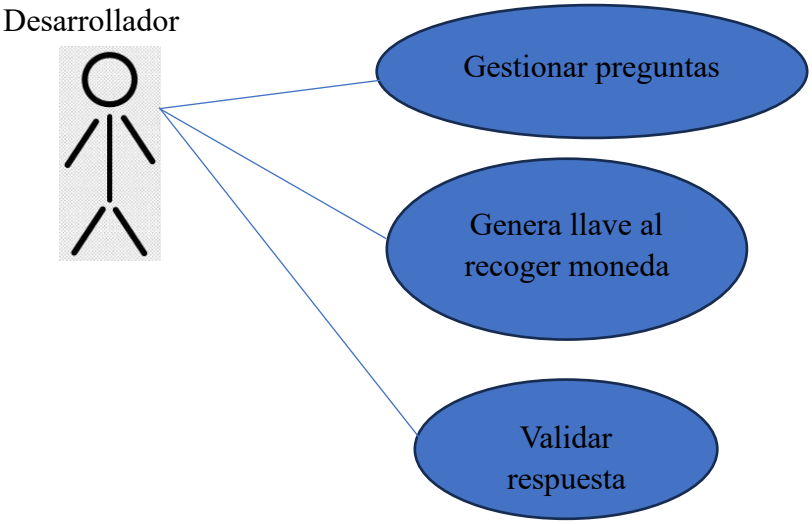
2. El jugador la recoge.
3. La puerta ahora puede abrirse.

Escenario 3: Apertura de la puerta y pregunta final

1. El jugador llega a la puerta con la llave.
2. La puerta se abre y aparece la pregunta sobre el Ecuador.
3. El jugador selecciona una respuesta.
4. El sistema valida la respuesta.
    - Si es correcta → Avanza.
    - Si es incorrecta → Reinicia el nivel.

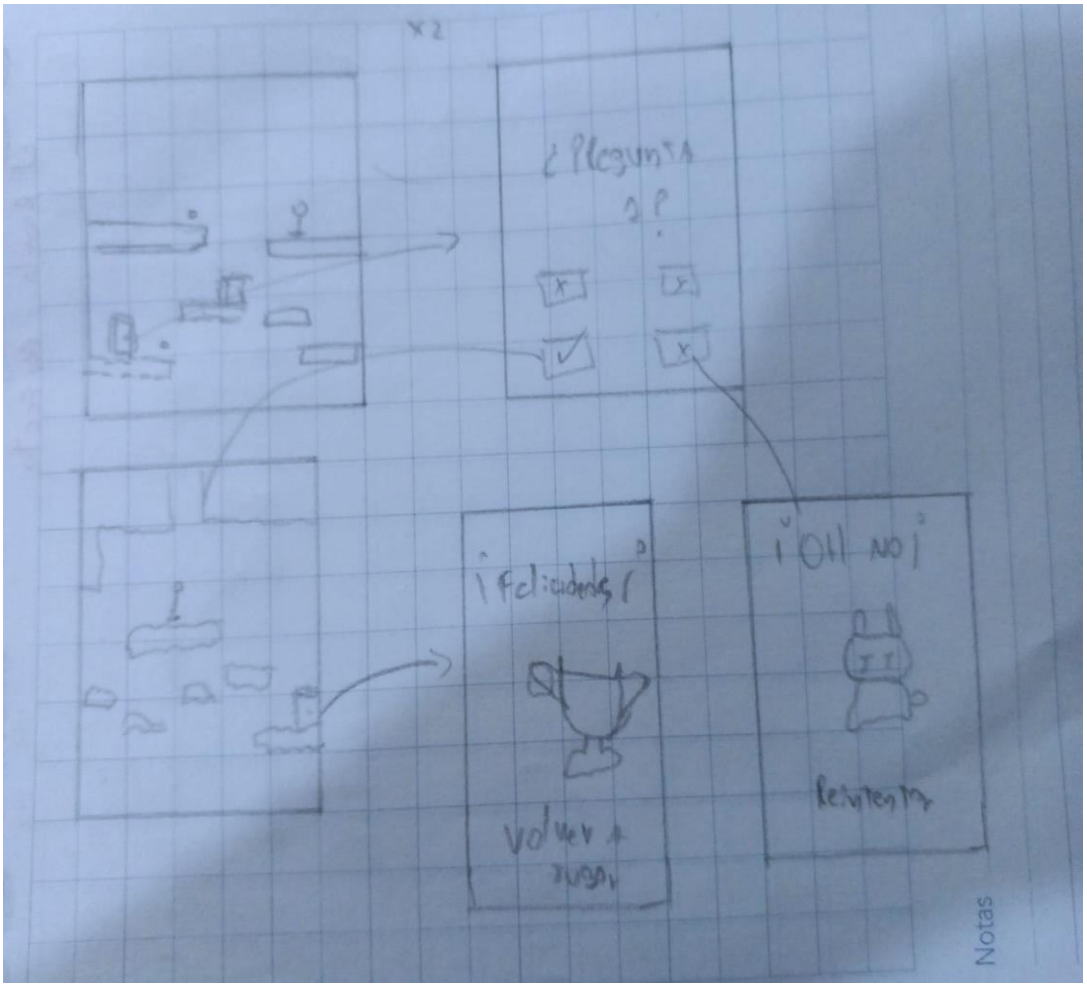
1.6. Diagrama de casos de uso





2. Diseño

2.1 WireFrames de Pantallas



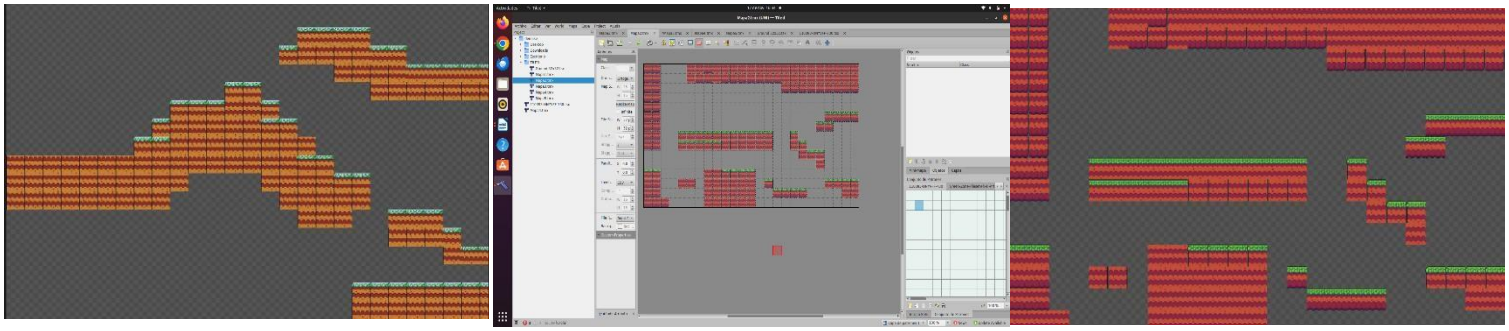
2.2 Diseño de personajes, fondos y recursos visuales.

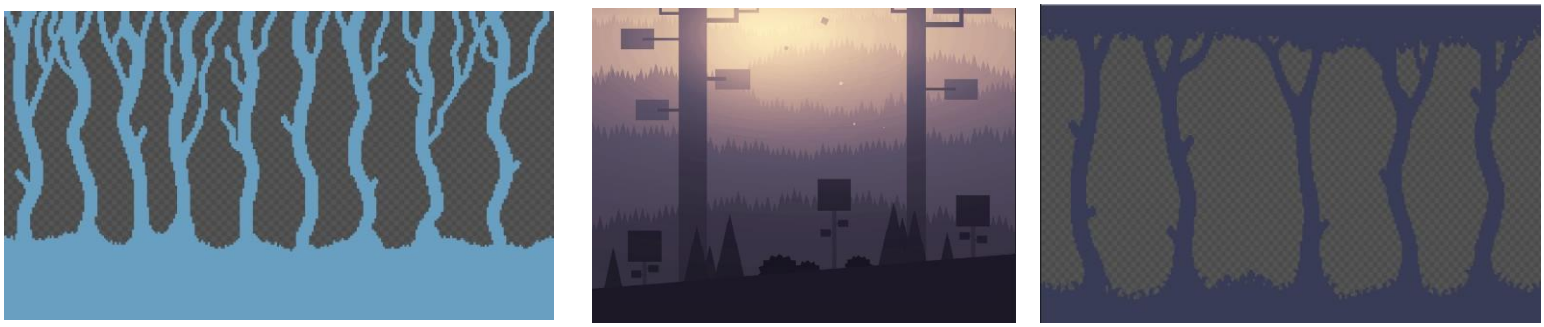
Figuras del diseño de personaje: Para la implementación del personaje, se utilizaron recursos gráficos preexistentes obtenidos de repositorios en línea para agilizar el desarrollo. La selección se centró en un estilo píxel art 2D coherente. Los Sprite se importaron y configuraron en GDevelop para responder a los controles de plataforma

Fuente: https://www.youtube.com/watch?v=7zq22efUiMY (Todos los créditos del personaje son de él).

Fondos y plataformas

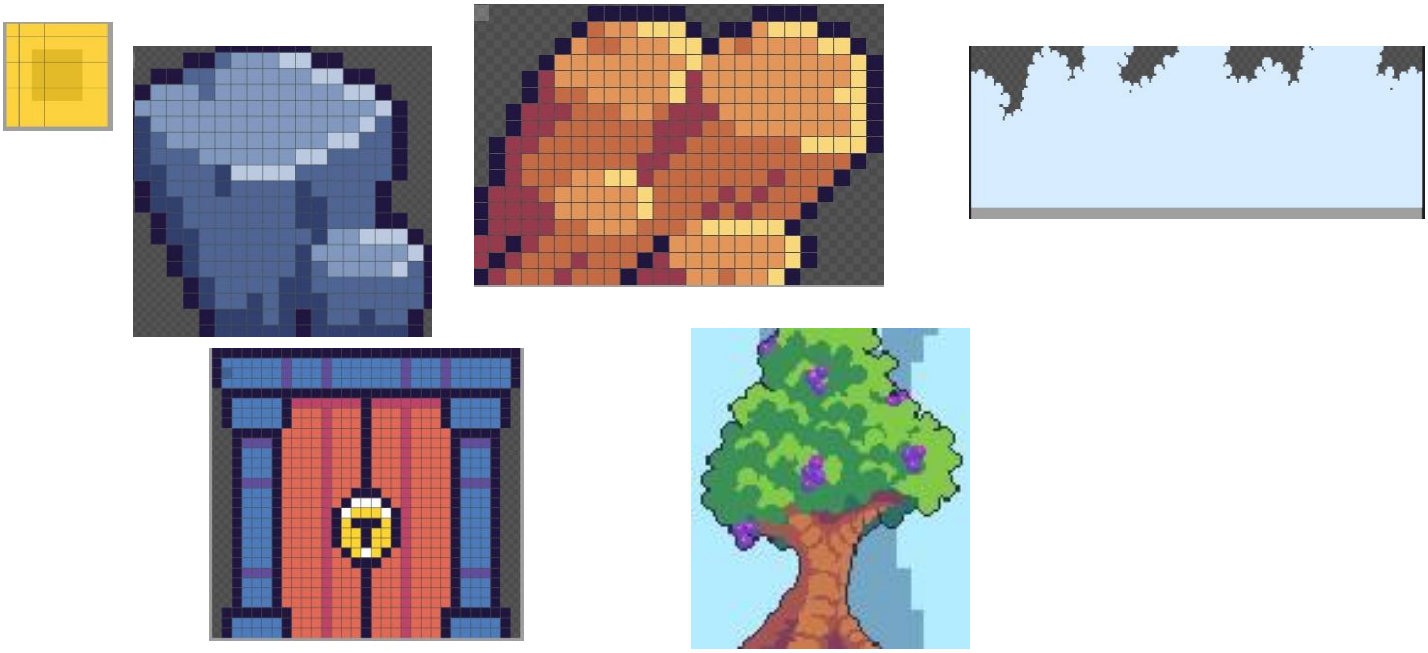
Se utilizo tilset (conjunto de baldosas) 2D con estilo píxel art para la construcción de los mapas representando las plataformas con colisiones de plataforma.





Recursos visuales

Los objetos recolectables como la llave y las monedas, se obtuvieron de un paquete de ítems píxel art antes ya mencionado en la creación del personaje. Aunque los recursos se obtuvieron de fuentes de terceros, la totalidad de la lógica de juego y la interactividad fue implementada por el equipo de desarrollo utilizando el sistema de eventos de GDevelop. Esto incluye: Mecánica de recolección y mecánica de peaje.



2.3 Arquitectura del Videojuego. Diagrama de Módulos

Módulos del videojuego: “Las flipantes aventuras del furro guayaco”

- **Módulo de control de Entrada (input)**

Captura las teclas izquierdas, derecha, salto(arriba) y las traduce en acciones para el personaje.

- **Módulo de Lógica**

Controla las colisiones del personaje y actualiza las animaciones y el manejo de la cámara.

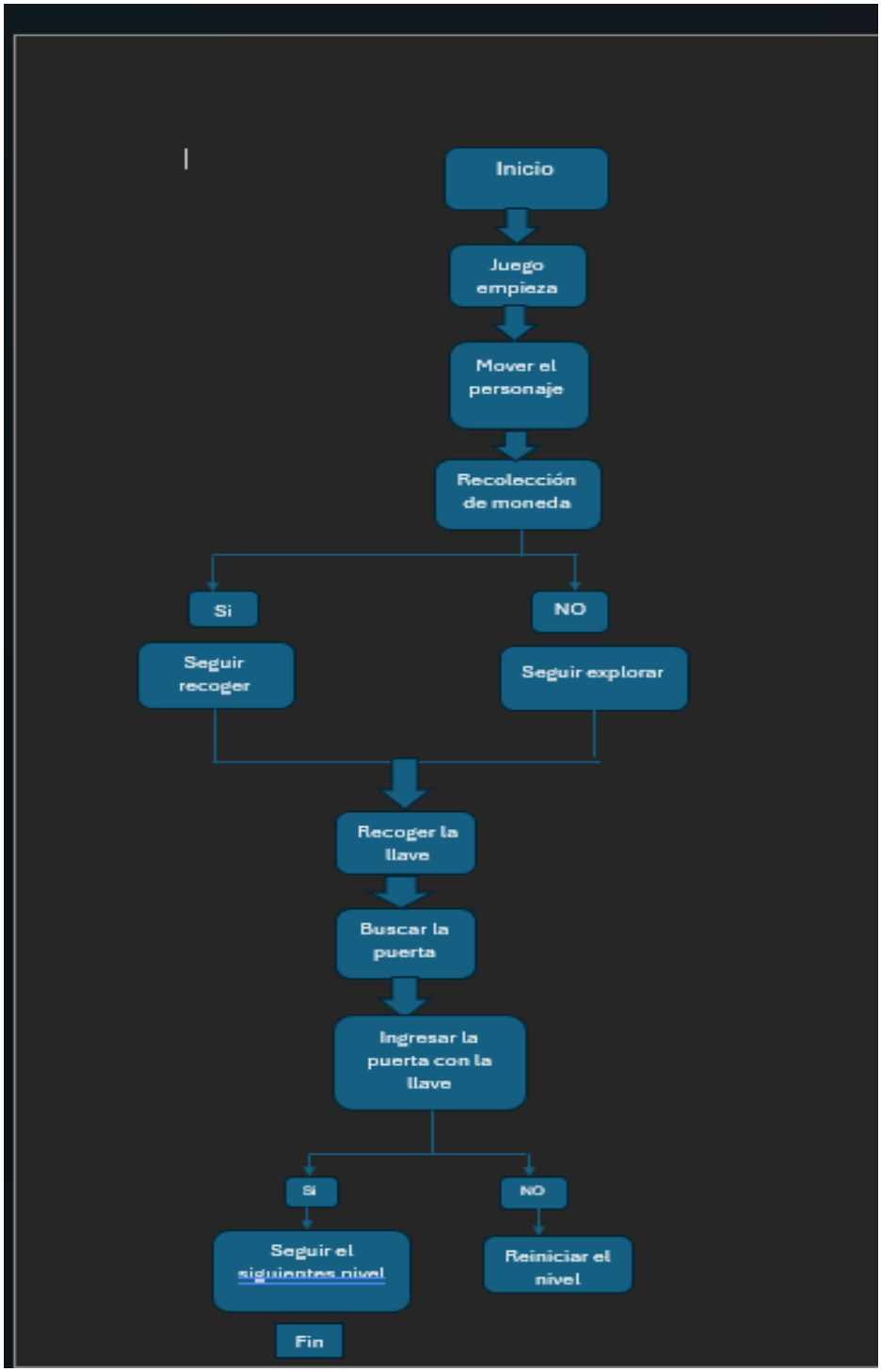
- **Módulo de interacción**

Gestiona la lógica de los coins, suma la variable global coin y maneja la mecánica del peaje en la llave. Comprueba si coin cumple con esta condición >= 10 y que si la variable “Tiene llave” sea true y se use en una puerta.

- **Módulo de navegación**

Este módulo se activa luego de que el módulo de interacción se cumpla en todos sus criterios de uso, redirigirá al jugador a una escena trivia de selección múltiple. Escoger la correcta o incorrecta te dirigirá al siguiente nivel o a la pantalla de derrota.

2.4 Diagrama de flujo



3. Implementación

3.1 Construcción del videojuego en GDevelop

El proyecto se organizó en escenas modulares para separar la lógica de juego de la lógica de trivia y los menús:

Escena de Nivel (ej. "Nivel_1"): Contiene la mecánica principal de plataformas, el personaje (jugador), los coleccionables (Coins), la llave y las puertas.

Escena de Trivia (ej. Pregunta1): Una escena separada que se carga al interactuar con una puerta. Muestra la pregunta y los botones de respuesta.

Escena de Victoria: Muestra la pantalla de "¡Felicidades!" con el botón invisible de reinicio.

Escena de Derrota: Muestra la pantalla de "Vuelve a intentarlo" con el botón invisible de reinicio.

Mecánicas implementadas

- **Movimiento del Jugador:** Se implementó usando el **Comportamiento de Plataformas** estándar en el objeto jugador. Las animaciones (caminar, saltar, etc.) se controlan mediante eventos de estado (ej. El jugador está en movimiento).
- **Sistema de Recolección y Peaje:** Esta es la mecánica central.

    1. **Recolección:** Un evento detecta la colisión jugadora con Coins, reproduce un sonido, añade 1 a la variable de escena coin y destruye el objeto Coins.
    2. **Peaje:** Se programó un evento condicional para la llave. Requiere colisión Y que la variable coin sea >= 10.
    3. **Pago:** Al cumplirse, un evento Disparar una vez asegura que solo se cobre una vez, restando 10 a coin y estableciendo la variable TieneLlave en true.

    • **Lógica de Trivia y Navegación:**

    1. La interacción con la puerta requiere tres condiciones: colisión, variable TieneLlave == true y Tecla "Up" presionada.
    2. Al cumplirse, la acción principal es Cambiar la escena a la escena "Pregunta1".
    3. En "Pregunta1", los botones usan las condiciones Cursor/táctil sobre objeto y Botón del ratón soltado para cambiar a la escena "Victoria" o "Derrota".

- **Botones de Reinicio:** En las escenas finales, se implementó un objeto Sprite invisible (BotonVolver_Hitbox) con Opacidad 0, colocado sobre la imagen del botón para detectar el clic y reiniciar el juego.

3.2 Integración de sprites, fondos, sonidos y animaciones.

- **Sprites:** Se importaron los recursos gráficos (jugador, Coins, llave) en formato PNG. Al objeto puerta se le configuraron dos animaciones (abierta y cerrada).
- **Fondos y Cámara:** Se utilizó una capa "Fondo" para la imagen de fondo estática. Al inicio de la escena, se usan las acciones Cambiar zoom de la cámara a 2 y Forzar límites de la cámara para centrar la vista y evitar que muestre el vacío.
- **Animaciones:** Las animaciones del jugador se activan por eventos condicionales (Si el jugador está en movimiento ---> reproducir animación caminar).
- **Sonidos:** Se usó la acción Reproducir sonido vinculada al evento de colisión con Coins.

3.3 Configuración de variables, condiciones y eventos.

El núcleo del juego se basa en una combinación de Variables de Escena (para el estado general) y Variables de Objeto (para estados específicos de ítems).

Variables Utilizadas

Se utilizaron las siguientes variables para gestionar el estado del juego:

- **coin** (Variable de Escena): Variable numérica que almacena la cantidad de monedas que el jugador ha recolectado.
- **TieneLlave** (Variable de Objeto, en llave): Variable booleana que funciona como un "permiso". Se activa (true) solo si el jugador toca la llave mientras tiene 10 o más monedas.
- **checkpointX / checkpointY** (Variables de Escena): Variables numéricas que guardan la posición X/Y del último punto de control para reaparición del jugador.

Eventos y Condiciones Clave

- **Evento de Muerte y Respawn:**
    - Condición: La posicion Y de jugador > 719 (detecta si el jugador se ha caído del mapa). ○ Acción: Cambia la posición de jugador a las coordenadas almacenadas en las variables checkpointX y checkpointY.
- **Evento de Recolección (Coins):**
    - Condición: jugador está en colisión con Coins. ○ Acciones: Añadir 1 a la variable de escena coin, Reproducir el sonido "PickupCoin" y Eliminar el objeto Coins.

- **Evento de Activación de Llave (Pre-peaje):**
    - Condiciones: llave está en colisión con jugador Y la variable de escena coin es >= 10. ○ Acciones: Establecer la variable TieneLlave del objeto llave en verdadero y Cambia la posición de llave (haciendo que la llave siga al jugador, indicando que está "activada").
    - *Este evento solo "activa" la llave, no cobra las monedas.*
- **Evento de Abrir Puerta (Pago de Peaje):**
    - Condiciones: La variable TieneLlave de llave es verdadero, jugador está en colisión con puerta, La animación de puerta = "abierta", y La tecla "Up" está presionada. ○ Acciones: Ir a la escena "pregunta1" Y Cambiar la variable coin: sustraer 10. ○ *Es en este evento donde se cobra el peaje de 10 monedas y se avanza al módulo de trivia.*
- **Evento de Actualización de HUD (Monedas):**
    - Este evento se ejecuta sin condiciones (en cada fotograma) para mantener el contador actualizado. ○ Acción 1: Cambiar el texto de Contador monedas: establecer a coin (Intenta vincular el texto a la variable coin). ○ Acción 2: Cambia la posición de Contador monedas (Fija el contador para que siga al jugador, usando expresiones como jugador. X () - 30).

4. Prueba

4.1 Listado de errores detectados corregidos.

Descripción del Error	Causa Principal	Solución Aplicada
El jugador "agarra y suelta" la llave instantáneamente al pagar el peaje.	El evento de peaje se ejecutaba 60 veces por segundo. En el fotograma 1, el jugador tenía 10 monedas (se restaban). En el fotograma 2, ya tenía 0, por lo que la condición (coin >= 10) era falsa.	Se añadió la condición avanzada Disparar una vez al evento "COGER LLAVE" para que la lógica de pago se ejecute una sola vez por colisión.
El contador de monedas en pantalla mostraba texto incorrecto (ej. la palabra "coin").	La acción Cambiar el texto del objeto contador monedas estaba configurada para mostrar el valor de texto estático "coin", en lugar	Se ajustó la acción del evento para usar la expresión "Monedas: " + VariableString(coin), vinculando dinámicamente el texto a la variable de escena.
El evento de peaje cobraba incorrectamente (ej. fijaba las monedas en -10).	La acción de cobro usaba el operador establecer en (=) con un valor de -10, en lugar de restar al total acumulado.	Se corrigió la acción para usar el operador sustraer (-) y el valor 10, asegurando un cobro correcto sobre el total de monedas.

Al hacer clic en el texto "¡Felicidades!" se reiniciaba el juego. (en toda la escena se reiniciaba y no en el botón)	**El evento de clic en las escenas de "Victoria" y "Derrota" estaba vinculado al objeto Sprite que contenía *toda* la imagen de fondo.**	**Se creó un nuevo objeto Sprite invisible (BotonVolver_Hitbox) con Opacidad 0. Este se colocó solo sobre el área visual del botón y el evento de clic se vinculó a este objeto.**

4.2 Evaluación del Aprendizaje Logrado

La mecánica educativa de este videojuego está integrada directamente en la progresión y funciona como una puerta de conocimiento.

El aprendizaje se evalúa de la siguiente forma:

1. **Mecánica de Peaje:** El juego fuerza al jugador a dominar la mecánica de plataformas (exploración y recolección de monedas) *antes* de permitirle acceder al desafío de conocimiento.
2. **Módulo de Trivia Dedicado:** El juego interrumpe el flujo de plataformas y cambia a una escena dedicada (pregunta1). Esto enfoca al 100% la atención del jugador en la pregunta.
3. **Evaluación de respuestas:** El éxito (escena Victoria) o el fracaso (escena Derrota) dependen directamente de la respuesta en la escena de trivia. Esto asegura que el jugador debe retener el conocimiento para poder ganar, validando la efectividad del aprendizaje.

Evaluación del Aprendizaje logrado (Desarrollador)

El desarrollo de este proyecto representó un ejercicio práctico en el motor GDevelop, yendo más allá de la simple colocación de *sprites* para enfocarse en la implementación de lógica de juego un poco compleja y la depuración de errores de estado.

El aprendizaje como desarrollador se puede resumir en los siguientes puntos clave:

1. **Gestión de Variables:** El mayor desafío y aprendizaje fue la gestión de las variables. Inicialmente se intentó usar Variables de Objeto (TieneLlave en el objeto llave). Se aprendió de forma práctica que esta lógica falla cuando el objeto es destruido o las condiciones cambian. La solución fue adoptar Variables de Escena (como coin y TieneLlave), que son persistentes y actúan como *flags* globales para el estado del nivel.
2. **Lógica de Eventos y el Bucle de Juego:** Se identificó un error crítico donde el jugador "agarraba y soltaba" la llave instantáneamente. Esto se debió a una falta de comprensión del bucle de juego (que se ejecuta 60 veces por segundo). Se

aprendió la importancia vital de la condición **Disparar una vez**. Esta condición es esencial para pagar el peaje, asegurando que la acción de restar monedas se ejecute una sola vez y no falle.

3. **Implementación de Mecánicas Condicionales (Peaje):** No solo se implementó una recolección simple (+1), sino una mecánica de "peaje". Se aprendió a combinar múltiples condiciones (colisión + variable >= 10) para disparar un evento, y a combinar múltiples acciones (restar 10 + activar flag TieneLlave) para ejecutar una mecánica de juego compleja.
4. **Técnicas de Interfaz de Usuario (UI/UX):** Para las escenas de "Victoria" y "Derrota", la solución inicial de hacer clic en toda la imagen de fondo era funcional pero imprecisa. Se aprendió una técnica de desarrollo estándar: la creación de **botones invisibles (Hitbox)**. Colocar un objeto Sprite con Opacidad 0 sobre el área visual del botón permitió una detección de clics precisa y una mejor experiencia de usuario.
5. **Modularidad de Escenas:** Se tomó la decisión de diseño de mover la trivia de una capa (UI_Pregunta) a una **escena completamente separada** (pregunta1).

   El proyecto fue de prueba y error. Sirvió como validación de que en el diseño de videojuegos 2D no son solo gráficos planos moviéndose, sino que existe toda una lógica robusta y una depuración de eventos intensiva.

ENLACE DEL JUEGO:

https://gd.games/games/f26f45a4-b030-4384-b2cc-bce211f59c04

ENLACE DE LA DEMOSTRACIÓN DEL

JUEGO:

https://drive.google.com/drive/folders/13FI3Teoj2l0k-kXH8A3e8tJwpG_o3kK?usp=sharing