Tejón Jumper

Faro Pérez, Inés

Gómez Tejón, Marcos

Maciel Montero, Claudia

Rey López, Carolina

Suárez Calvo, Esteban

Índice

Índice de figuras

1. Desarrollo artístico

1.1. Antecedentes

1.1.1. Ambientación

Tejón Jumper es un juego de plataformas donde se toma de protagonista a un pequeño tejón que desea recuperar las bayas que le han sido robadas. En cada nivel tendremos que saltar y hacer uso de nuestras habilidades para evitar obstáculos y avanzar, recolectar bayas para obtener más vidas y otros efectos y rodar para alcanzar ciertas zonas en cada nivel.

1.1.2. Historia

En lo más profundo del Bosque de las Bayas, vive un valiente tejón. Ha pasado toda su vida en este bosque, explorando cada rincón del terreno y asegurándose de mantener su madriguera segura y abastecida con comida suficiente para mantener a toda su familia.

Sin embargo, una noche oscura y silenciosa, un depredador acecha entre las sombras. Un enorme oso y su banda, guiados por su insaciable hambre y egoísmo, encuentran la madriguera del tejón y deciden aprovecharse de su esfuerzo. Dicha banda irrumpe en la guarida y se apodera de todas las provisiones del tejón, dejándolo a él y a su familia sin recursos y con una profunda hambre.

Pero el tejón no se queda de brazos cruzados y decide emprender una aventura peligrosa para recuperar lo que le pertenece. El camino no será fácil, pues en el bosque habitan otros aliados del oso que no dudarán en hacerle frente: robustos erizos, astutos zorros, hábiles murciélagos y traviesas ardillas.

Durante su travesía a través del Bosque de Bayas, el tejón deberá correr, saltar y luchar para superar cada obstáculo, derrotando a sus enemigos y avanzando hacia su objetivo final. Además, durante este peligroso viaje irá recogiendo todas las bayas que vea por el camino.

Finalmente, tras un arduo viaje, el tejón llegará a la cueva del oso, donde lo encuentra disfrutando de las provisiones robadas. Sin miedo, se enfrenta al gigante adversario en una

batalla decisiva. Con rapidez y astucia, el tejón usa todas sus habilidades para vencer al oso y recuperar su comida.

Al final de su odisea, el tejón regresa triunfante a su madriguera, con sus provisiones nuevamente a salvo.

1.2. Personajes

- **Tejón.** Es el protagonista del juego (Figura 1). Puede correr rápidamente, saltar entre plataformas, rodar consumiendo su barra de energía e incluso volverse invulnerable por los efectos del estado de rabia.

Figura 1: Boceto del tejón

- **Oso.** Es el villano principal del juego (Figura 2). Le ha robado las bayas al tejón junto a su banda. Puede lanzar golpes devastadores con sus garras y causar temblores con su peso al caer además de recorrer la pantalla de un extremo al otro buscando hacer daño a nuestro protagonista.

Figura 2: Boceto del oso

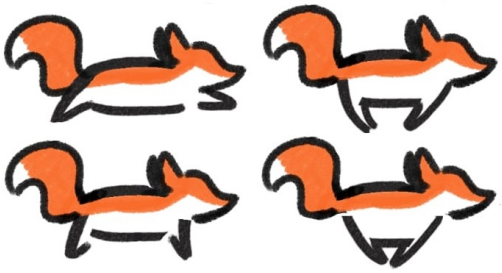
- **La banda del oso.** Son los enemigos menores del juego (Figura 3). Está compuesta por los siguientes animales:

  - **Zorros**. Caminan plácidamente por el bosque y son los enemigos más sencillos de derrotar, bastará con arañarlos o saltarles encima.

- **Erizos.** Se esconden en su capa de púas, haciendo daño al tejón si salta encima de ellas, por lo que únicamente podrán ser derrotados siendo arañados.

- **Murciélagos.** Son los enemigos voladores del juego, vuelan en trayectoria de U y se les podrá derrotar saltando encima de ellos.

- **Ardillas.** Estos enemigos viven plácidamente en las copas de los árboles. Sin embargo, cuando detectan al tejón pasar por debajo de su casa, no dudarán en lanzarle bellotas para alejarlo de su hogar. Este enemigo se puede derrotar saltándole encima, siempre y cuando estemos en un sitio suficientemente alto para hacerlo.



(a) Boceto de un zorro

(b) Boceto de un erizo

(c) Boceto de un murciélago

Figura 3: La banda del oso

1.3. Características de la ambientación

1.3.1. Objetos destacados

Los objetos destacados de este juego son la bayas. Cada baya tiene un efecto diferente.

- **Baya de energía.** La función de esta baya es la de rellenar la barra de energía.

  Esta barra está estrechamente relacionada con la habilidad de rodar: si dicha barra tiene algo de energía, el tejón podrá rodar durante un tiempo. Este tiempo será mayor o menor dependiendo de lo llena que esté dicha barra.

En la Figura 4 podemos observar tanto la baya de energía como la barra de energía.

(a) Boceto de la baya de energía

(b) Barra de energía

Figura 4: Energía

- **Baya de vida.** Baya que sirve para recuperar un corazón de la vida del tejón.

  Si el jugador se queda sin corazones o se cae por un agujero, el tejón pierde la partida. Podemos ver cómo son los corazones de vida del juego en la Figura 5b.

  En la Figura 5 podemos observar tanto la baya de vida como los corazones del tejón.

(a) Boceto de la baya de vida

(b) Corazones del tejón

Figura 5: Vida

- **Baya de rabia.** Baya que permite que el tejón entre en el modo rabia. En este modo, el tejón se vuelve más rápido y es inmune a los ataques enemigos.

  Podemos observar esta baya en la Figura 6.

Figura 6: Boceto de la baya de rabia

A parte de las bayas, también tenemos las **setas**, las cuales nos atacaran con sus esporas, haciéndonos daño. Sin embargo, estas también servirán para ayudar al tejón a alcanzar lugares más altos, pues le pueden servir como plataformas.

1.3.2. Lugares

- **Bosque de las Bayas.** Lugar donde se desarrolla toda la historia del juego. En él habita el tejón con su familia y otros animales.

- **Cueva del Oso.** Base del oso, donde esconde todas las bayas del tejón. En ella, el tejón y el oso lucharán para decidir quién se queda con el botín.

1.4. Guion

1.4.1. Desarrollo y transcurso del videojuego

El juego constará de 3 niveles. En el primero, recorreremos el bosque de día, encontrando los enemigos más básicos, es el nivel que servirá como base. En el segundo, recorreremos el bosque de noche y aparecerán enemigos más complejos, como los murciélagos. Y en el último nivel, llegaremos a la Cueva del Oso donde ocurrirá la batalla final contra el jefe de la banda. Tendremos que usar todo lo aprendido en los anteriores niveles para esquivar sus ataques y conseguir bajar su barra de vida con el propósito de derrotarlo y conseguir hacernos con las bayas.

La historia se contará al iniciar una nueva partida usando una pequeña animación donde se nos presentará a los personajes principales y la trama.

En los niveles nos encontraremos los distintos tipos de bayas indicados en la sección 1.3.1, con los que gracias a sus efectos podremos explorar nuevas zonas en cada nivel en las que no se puede entrar de forma convencional.

El juego tendrá 3 niveles de dificultad. Dependiendo del nivel de dificultad, tendremos más o menos corazones, además de tener distintos obstáculos en los niveles, como pueden ser charcos de barros que nos ralentizan, viento que nos entorpece en las secciones con saltos o ramas que caen de los árboles que pueden reducirnos la barra de vida

1.4.2. Diferencias entre el 2D y el 3D

La diferencia entre el videojuego 2D y 3D será que se añadirá una nueva dimensión tanto a los personajes como a los escenarios, pudiéndonos mover más libremente por las zonas del juego.

Los niveles pasarán a estar diseñados de otra forma teniendo en cuenta esta nueva dimensión. Por este motivo, las habilidades de los distintos personajes serán modificadas.

2. Videojuego 2D

2.1. Descripción

Tejón Jumper es un videojuego de plataformas donde manejaremos a un pequeño tejón que recorre un bosque a lo largo de 3 niveles, recolectando diferentes tipos de bayas y enfrentándose a varios enemigos.

2.1.1. Personajes

El personaje principal es el tejón, que será controlado por el usuario. El tejón no solo podrá saltar para llegar a lugares altos y derrotar enemigos, si no que también podrá rodar, lo que le permite hacer daño a sus enemigos, romper ciertos elementos del nivel y moverse más rápido.

Además, con la ayuda de la baya de rabia, podrá entrar en el estado de rabia, lo que le permite volverse invulnerable a los ataques enemigos, derrotarlos con tan solo tocarlos, moverse a gran velocidad y recuperar toda la energía.

2.1.2. Enemigos

Durante el viaje del tejón, nos encontraremos a los siguientes enemigos:

- **Zorros.** Caminan por el bosque en línea recta. Es el enemigo más básico que hay. Para derrotarlo, se puede, o bien saltar encima, o colisionar con él mientras el tejón está rodando.

- **Erizos.** Tienen el mismo movimiento que los zorros pero con una peculiaridad, únicamente los podremos derrotar si el tejón está rodando, ya que si saltamos encima de él sin estar rodando recibiremos daño debido a sus púas.

- **Ardillas.** Se sitúan encima de ciertas plataformas, lanzado bellotas tanto a la izquierda como a la derecha, según donde esté el jugador. Se derrotan igual que los zorros.

- **Murciélagos.** Vuelan por el nivel trazando una trayectoria con forma de V. Se derrotan de la misma forma que los zorros y las ardillas.

- **Setas.** Disparan esporas en línea recta para dañar al jugador. Sirven también como plataformas para que el tejón pueda alcanzar ciertos lugares. No se pueden derrotar.

- **Oso.** Es el enemigo final del juego. Se mueve en línea recta tal y como si fuese un zorro o un erizo, pero al recibir daño comienza a saltar e incrementa su velocidad. Su velocidad incrementa cada vez que recibe daño, de forma que cuanta menos vida tiene más difícil será de derrotar. La única forma de derrotarlo es saltarle encima.

2.1.3. Objetos

Los objetos principales del juego son las bayas. Existen varios tipos de bayas, y cada una de ellas tiene una funcionalidad diferente.

- **Baya de vida.** Baya roja en forma de corazón. Permite recuperar un corazón al jugador.

- **Baya de energía.** Baya amarilla en forma de rayo. Permite recuperar toda la energía al jugador.

- **Baya de rabia.** Baya naranja de forma redonda. Al obtenerla, el tejón entra en estado de rabia durante 10 segundos, lo que le permite correr más rápido, recuperar toda su energía, ser inmune al daño recibido y eliminar a sus enemigos únicamente tocándoles.

- **Moneda**. Baya azul con forma de moneda. Al recolectar 25 de estas bayas el tejón gana una vida.

También nos encontraremos con montones de rocas, las cuales nos impiden el paso. Para destruirlas será necesario rodar contra ellas.

Además, también tenemos una bandera al final de cada nivel, lo que nos permite transicionar al siguiente nivel (o a la pantalla de victoria si ya estamos en el último nivel).

2.1.4. Diseño

Dentro del juego, tenemos dos niveles de dificultad:

- **Fácil.** El tejón comienza con 3 vidas extra, su número máximo de corazones es 5 y el oso tiene 3 corazones.

- **Difícil.** El tejón comienza con 1 vida extra, su número máximo de corazones es 3 y el oso tiene 5 corazones.

Como se indicó en el Apartado 2.1.1, el tejón puede saltar, rodar y entrar en el modo de rabia. Únicamente podemos rodar si la barra de energía no está vacía.

Para terminar un nivel, deberemos llegar hasta la bandera del final sin caernos ni perder todos los corazones. Si perdemos una vida, reiniciaremos el nivel actual. Si perdemos todas las vidas, empezaremos desde el principio.

2.2. Escenas

El juego consta de las 3 fases que veremos a continuación.

2.2.1. Primera fase

La primera fase se ha planteado como una pequeña introducción al juego. Su ambientación presenta al Bosque de Bayas durante el día. En él nos encontraremos con zorros, erizos, ardillas y setas. Estos enemigos son tanto fáciles de derrotar como de esquivar. En este nivel se nos presentarán la baya de energía, y la de vida (Sección 2.1.2).

2.2.2. Segunda fase

La ambientación de la segunda fase representa al Bosque de Bayas de noche. En este nivel nos encontraremos enemigos más difíciles de derrotar como los murciélagos. En este nivel se introduce la baya de rabia (Sección 2.1.2).

2.2.3. Tercera fase

En el último nivel la ambientación se corresponde con la caverna del oso. En primera instancia, tendremos que saltar sobre varias setas para llegar al oso, teniendo que esquivar las esporas que lanzan. Una vez las hayamos superado, nos encontraremos con el oso. Tendremos dos opciones: derrotarlo o evitar esta pelea. Dependiendo de lo que hagamos, obtendremos una pantalla final distinta.

2.3. Detalles de implementación

2.3.1. Miembros y reparto del trabajo

- **Inés Faro:** programación y diagramas.
- **Marcos Gómez:** programación y diseño artístico.
- **Claudia Maciel:** programación y documentación.
- **Carolina Rey:** programación y documentación.
- **Esteban Suárez:** programación, diseño de interfaces y documentación.

2.3.2. Metodología de desarrollo

Para el desarrollo de este proyecto, se ha utilizado la metodología *kanban* para la gestión de tareas. Este enfoque permite una organización visual del flujo de trabajo, facilitando el seguimiento del estado de cada tarea. El tablero kanban se estructuró en cinco columnas:

- *Bloqueadas.*
- *Listas para comenzar.*
- *En progreso.*
- *En revisión.*

- *Completadas.*

Cada tarea fue asignada a un integrante del equipo, excepto en ciertos casos en los que dos desarrolladores trabajaron conjuntamente aplicando la técnica de *pair programming*. Esta metodología permitió mejorar la calidad del código y la resolución de problemas en tiempo real, además de fomentar el aprendizaje colaborativo. Gracias a este enfoque, se optimizó la gestión del trabajo en paralelo, asegurando que cada miembro del equipo pudiera contribuir activamente sin generar cuellos de botella en el proceso de desarrollo.

Para la gestión del control de versiones y la integración del código, se adoptó el modelo *git-flow*. Se establecieron dos ramas principales: *main*, donde se consolidan las versiones estables del proyecto, y *develop*, utilizada como rama de integración donde se fusionan las funcionalidades en desarrollo. Además, se crearon ramas específicas para cada tarea, siguiendo una nomenclatura estandarizada. Las tareas de desarrollo se implementaron en ramas denominadas *feature/XX*, mientras que las correcciones de errores se gestionaron en ramas *bugfix/XX*, donde *XX* representa el identificador de la tarea correspondiente.

Para garantizar la calidad del código y evitar problemas de integración, se estableció un proceso de revisión antes de fusionar cualquier rama con *develop* o *main*. Cada solicitud de incorporación de cambios (*pull request*) debía ser revisada y aprobada por al menos dos integrantes del equipo que no estuvieran involucrados en el desarrollo directo de la tarea (destacar que si una tarea recibe peticiones de cambios sin resolver, no se puede *mergear* independientemente del número de aprobados). Este proceso contribuyó a mantener un código limpio, reducir errores y asegurar la coherencia del proyecto en su conjunto.

2.3.3. Diagramas de componentes

A continuación se muestran varios diagramas de clases (Figura 8 y Figura 9) donde se puede ver con mejor lujo de detalle las relaciones previamente señaladas en el diagrama de componentes (Figura 7).

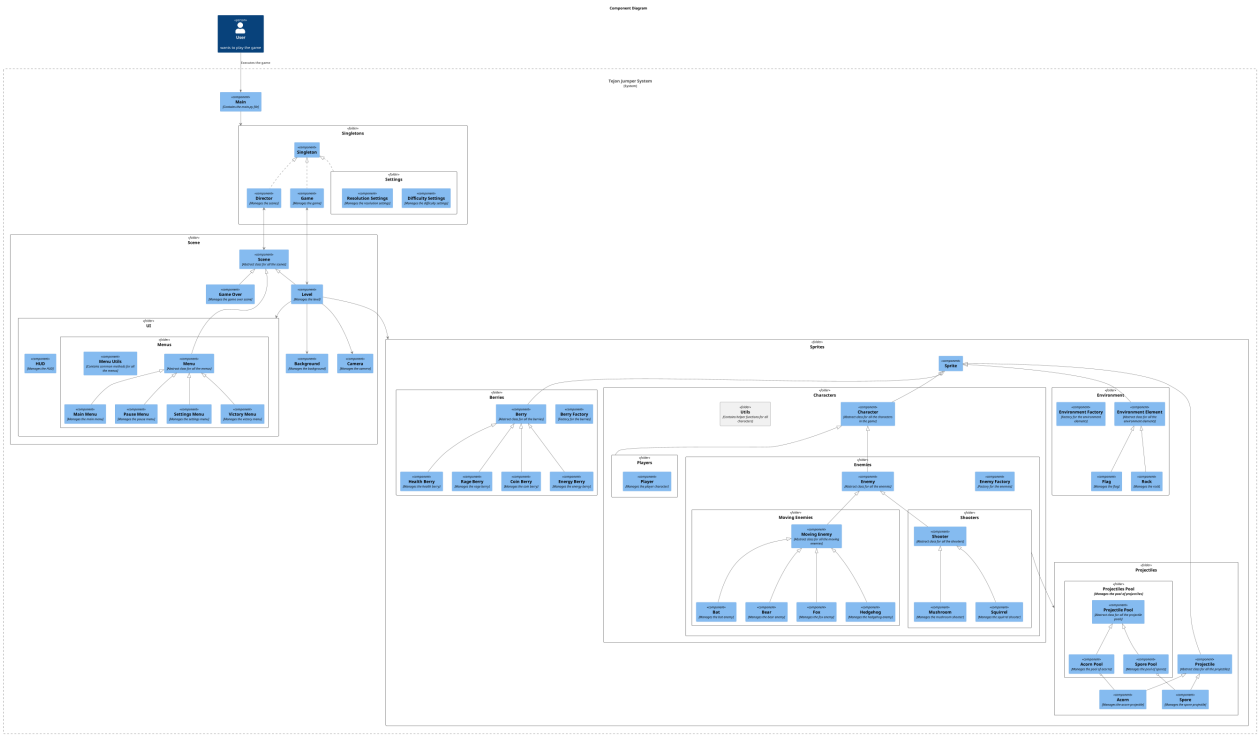


Figura 7: Diagrama de componentes

2.3.4. Diagramas de clases

La Figura 8 incluye clases esenciales para la gestión del juego, como el *Director* que controla el flujo de escenas o *Scene*, clase base para todas las escenas. También resalta componentes auxiliares como *HUD*, *Camera* y *Background*. Cabe destacar las relaciones de *Director*, *ResolutionSetting*, *DifficultySettings* y *Game* con *SingletonMeta*, que son relaciones de metaclase. Así como las relaciones de herencia, entre *Level* y *Scene* (entre otras) y las relaciones de uso como podemos ver entre *Level* y *PauseMenu*, por ejemplo.

La Figura 9 se enfoca en elementos del gameplay como *Level* (diseño de niveles), *Character* (jugador y enemigos como *Bear* o *Squirrel* entre otros), *Projectile* (como las bellotas), y recursos como *EnergyBerry* o *CoinBerry*, por ejemplo. También cabe destacar las relaciones de herencia presentes, así como las relaciones de agregación entre *Acorn* y *AcornPool* o *Spore* y *SporePool*.

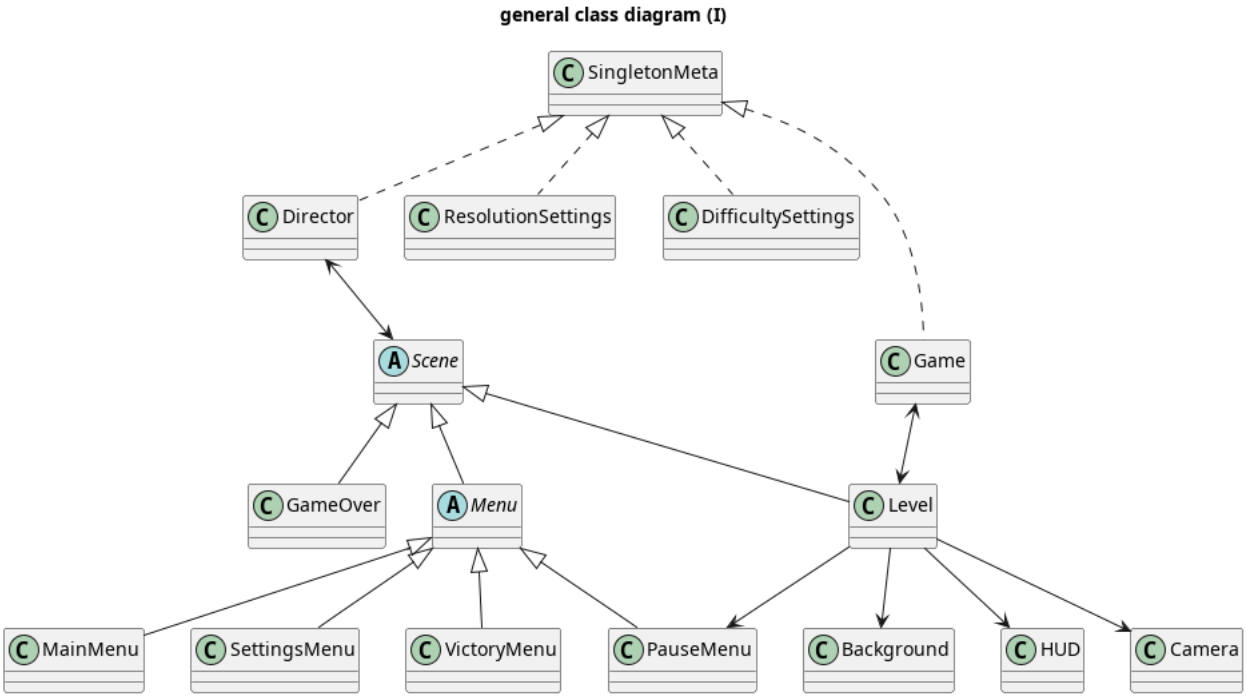


Figura 8: Diagrama de clases (I)

2.3.5. Diagramas de flujo

Estos diagramas ilustran la interacción en tiempo real entre el usuario, el sistema y los componentes del mismo, destacando decisiones y manejo de estados.

En la Figura 10 podemos observar la secuencia en la que un jugador ejecuta el juego y navega por el menú de inicio. En primera instancia, el director carga el menú principal para luego cargar el primer nivel.

En la Figura 11 se exhibe el caso en el que el jugador presiona ajustes en la pantalla de inicio.

En la Figura 12 se muestra el diagrama correspondiente al ciclo del juego, donde el jugador se mueve por los distintos niveles siendo el director el que gestiona las diferentes transiciones entre escenas. Se exponen en este diagrama diferentes alternativas que pueden ocurrir durante el periodo de juego como puede ser que el jugador pierda una vida, pierda todas las vidas, reinicie el nivel, complete un nivel o finalice el juego ya sea saliendo de él o consiguiendo la victoria.

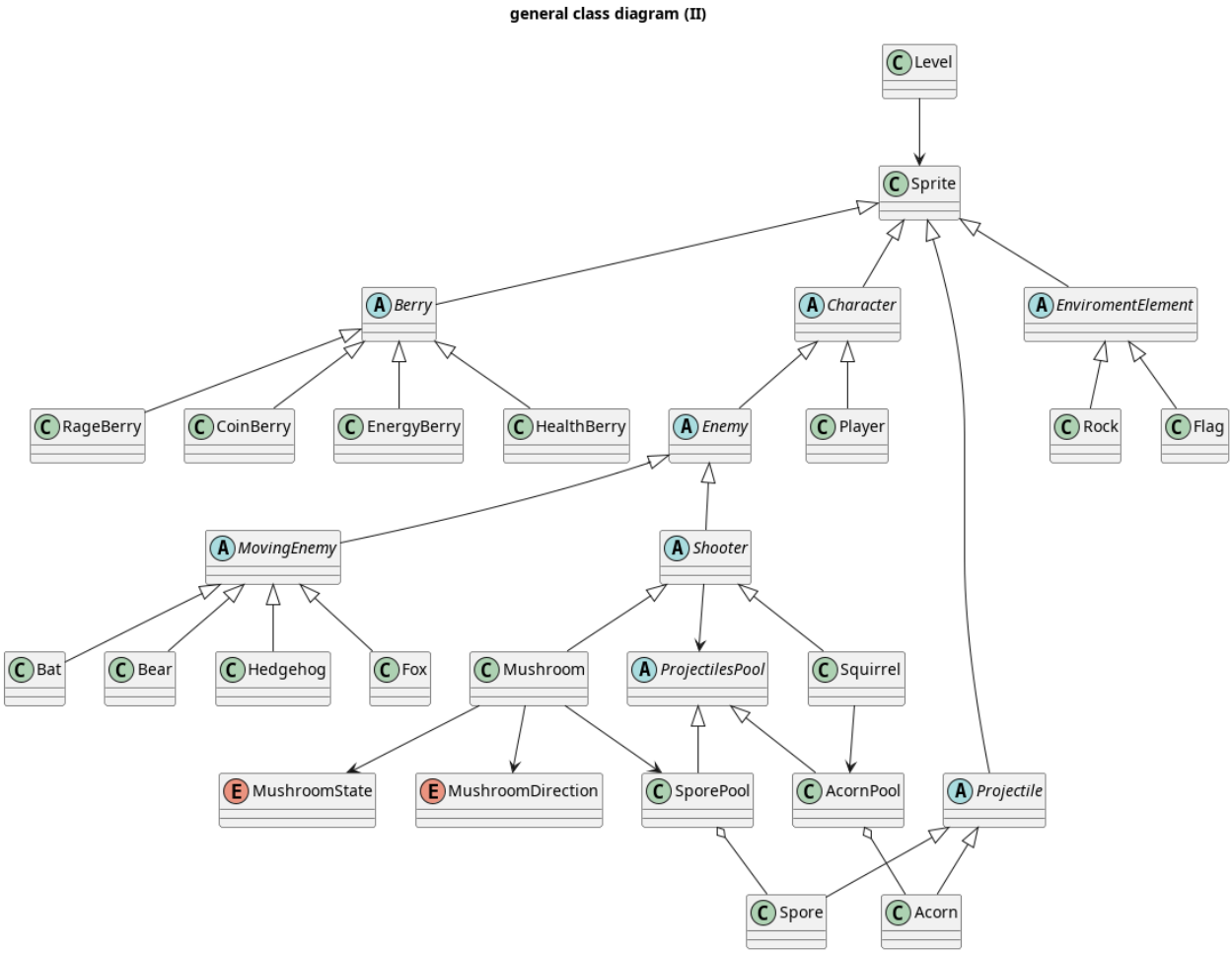


Figura 9: Diagrama de clases (II)

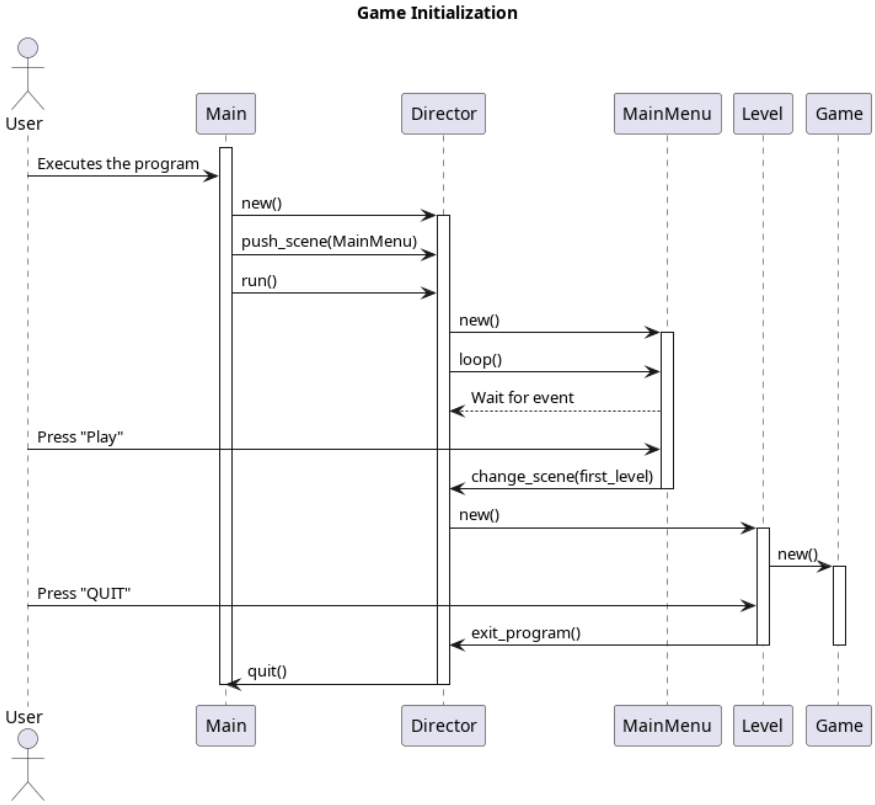


Figura 10: Diagrama de secuencia - Inicialización del juego

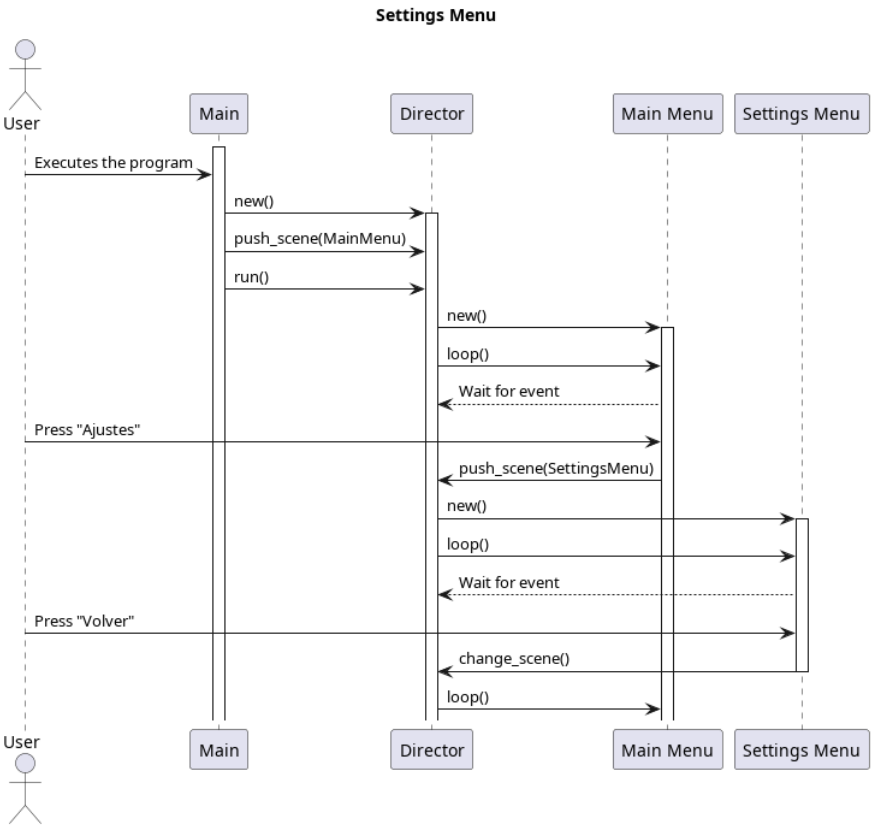


Figura 11: Diagrama de secuencia - Menu de ajustes

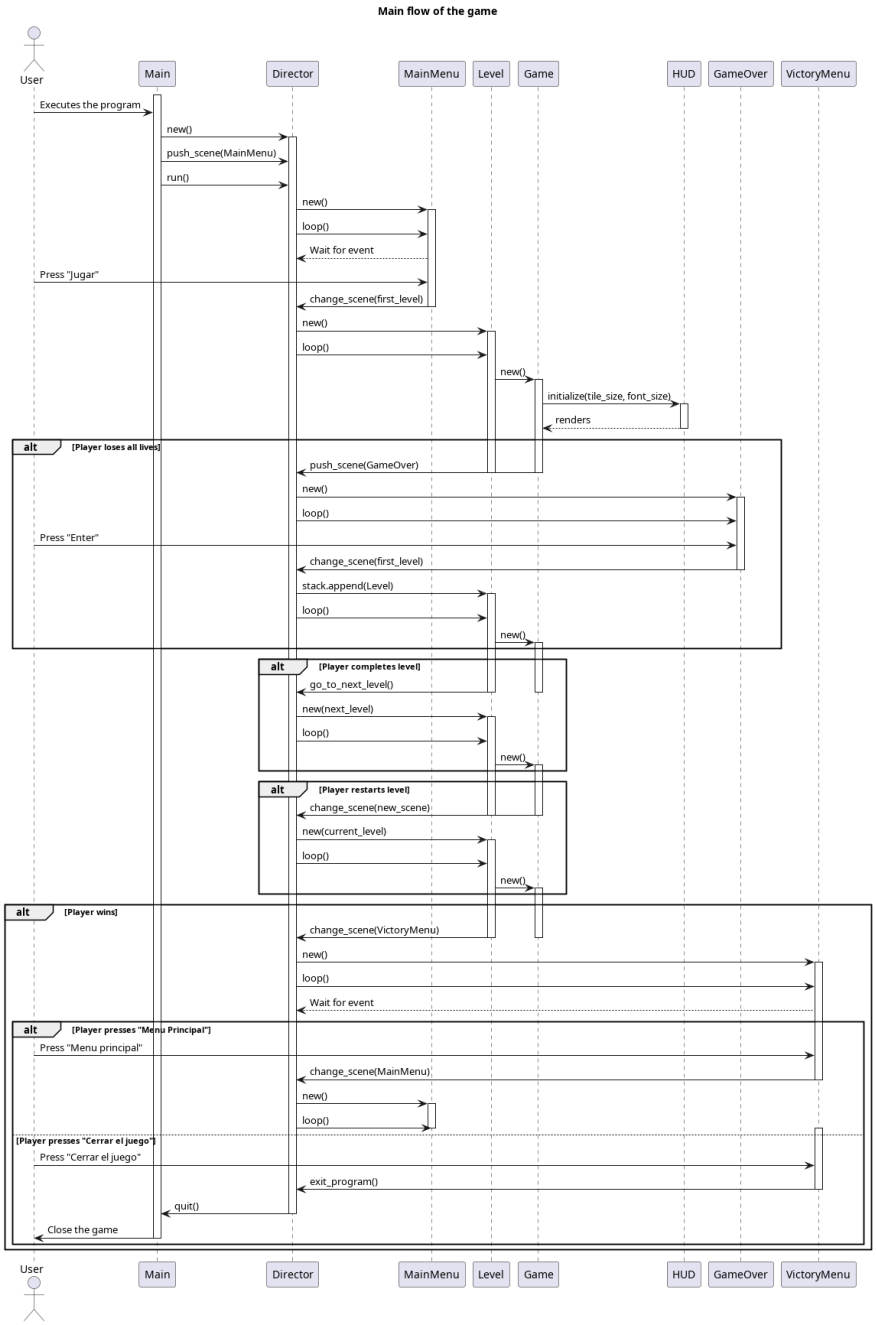


Figura 12: Diagrama de secuencia - Flujo del juego

Finalmente, en la Figura 13, se presenta la interacción del usuario al pulsar la letra 'P' para pausar el juego y las diferentes alternativas del menú, donde el usuario puede reiniciar el nivel o continuar con el mismo.

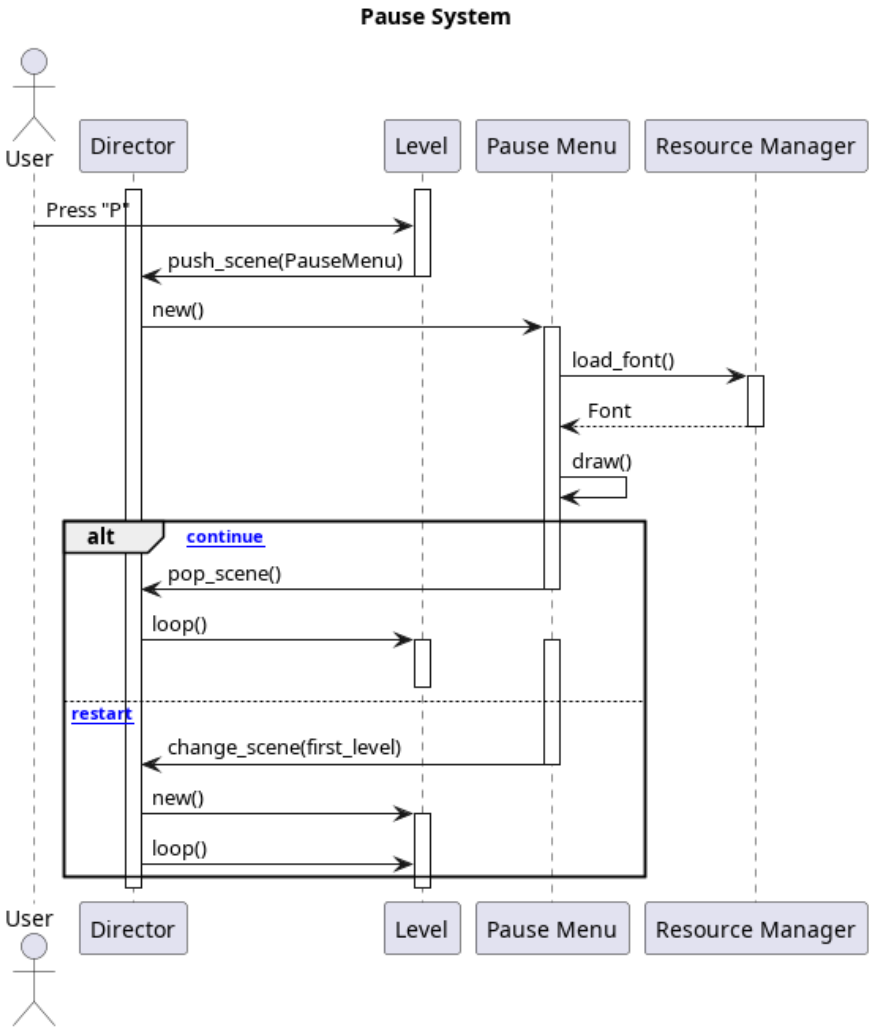


Figura 13: Diagrama de secuencia - Menú de pause

2.3.6. Diagrama de estados

De igual modo que en los apartados anteriores, se ha diseñado un diagrama de estados (Figura 14) para mostrar las transiciones existentes entre las diferentes escenas del juego, iniciando en el momento en el que el jugador ejecuta el programa y finalizando cuando gana el juego o sale de él.

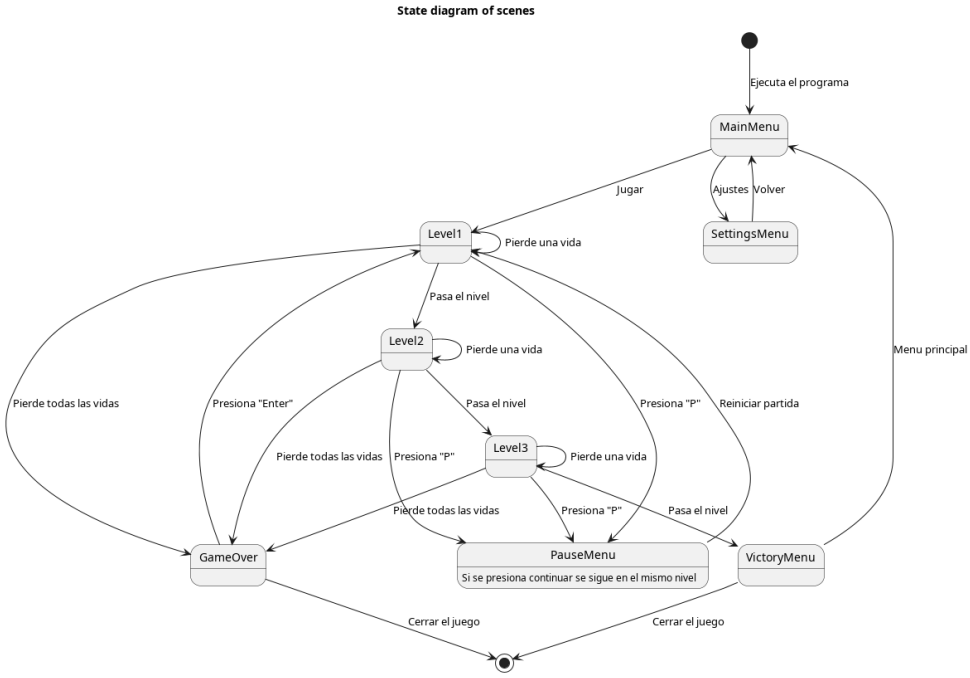


Figura 14: Diagrama de estados

2.3.7. Patrones de diseño

Los patrones de diseño aplicados en este proyecto son los siguientes:

- **Patrón Plantilla (Template Method)**

  El patrón Plantilla (Template Method) es un patrón de diseño que define el esqueleto de un algoritmo en un método, pudiendo dejar algunos pasos específicos a ser implementados por las subclases. El método de la clase base proporciona una secuencia de pasos fijos, pero permite que las subclases sobrescriban ciertos métodos o pasos sin cambiar la estructura general del algoritmo. Esto promueve la reutilización de código y la extensión del comportamiento, ya que se puede personalizar la parte variable de un proceso sin modificar su flujo principal. Este patrón es útil cuando varios procesos comparten una estructura común, pero con pequeñas variaciones.

  Se utiliza para definir el comportamiento de los enemigos.

  Primero, se utiliza en la clase *Enemy* para definir el algoritmo base *update* para ges-

tionar las colisiones con el jugador. Las clases que heredan de esta (*MovingEnemy* y *Shooter*) extienden este algoritmo para incluir los pasos relativos al movimiento de los enemigos o de disparar respectivamente.

Por un lado, la clase *MovingEnemy* define el algoritmo base *update* para el movimiento de un enemigo, definiendo además una implementación por defecto del movimento de un enemigo (ya que hay varios enemigos que se mueven igual o de forma muy similar). En algunos casos, como en el de la clase *Bat*, la cual hereda de *MovingEnemy*, el algoritmo base se extiende para añadir pasos adicionales que únicamente son necesarios dentro de esa clase.

Por otro lado, la clase *Shooter*, define el algoritmo base *update* para los enemigos que disparan algún tipo de proyectil. El algoritmo base es extendido en una subclase, en este caso en la clase *Squirrel*.

También se utiliza para definir la gestión de eventos en de los Menús, definiendo el algoritmo base en la clase *Menu*.

- **Patrón Gestor de Estado (State Manager)**

  El patrón Gestor de Estado (State Manager) es un patrón de diseño que permite a un objeto cambiar su comportamiento dinámicamente según su estado interno, delegando la lógica de cada estado a clases separadas. En lugar de usar múltiples condicionales para manejar distintos estados, el objeto principal mantiene una referencia a un estado actual y delega las acciones a esa instancia, lo que mejora la organización y facilita la extensibilidad.

  Este patrón está implementado en la clase *Director*, cuya función es orquestar el juego e ir cargando las diferentes escenas del juego (las cuales se almacenan en una pila), el bucle de eventos, etc.

- **Patrón Factoría (Factory Method)**

Es un patrón de diseño creacional que delega la responsabilidad de instanciar objetos a una clase especializada, evitando así la creación directa con new o su equivalente. En su lugar, se define un método (o función) fábrica que encapsula la lógica de creación y permite que las subclases determinen qué tipo de objeto crear. Esto facilita la extensibilidad, el mantenimiento y el desacoplamiento, ya que el código principal no necesita conocer las clases concretas, sino solo la interfaz o superclase común.

Se utiliza a la hora de crear los enemigos (*enemy_factory*), los elementos del entorno con los que el jugador puede interactuar (*environment_factory*) y las bayas (*berry_factory*).

- **Patrón Singleton**

  El patrón Singleton es un patrón de diseño creacional que garantiza que una clase tenga una única instancia en todo el programa y proporciona un punto de acceso global a ella. Para lograrlo, se usa una variable estática que almacena la instancia y un método que la crea solo si aún no existe, devolviendo siempre la misma referencia. Esto es útil cuando se necesita un único objeto compartido.

  En este proyecto aparece implementado en las clases *Game*, *Director* y las relativas a la configuración del juego (*ResolutionSettings* y *DifficultySettings*).

2.4. Aspectos destacables

En lo que respecta al apartado artístico, todos los sprites de los personajes, elementos del entorno (bandera y roca), las plataformas de madera, logos, bayas, vidas del oso y el fondo de la cueva han sido creados por miembros del equipo de desarrollo.

Dentro de la parte de interfaces de usuario, hemos añadido un menú de inicio con un apartado de configuración donde se puede configurar la dificultad, el volumen de la música y el de los efectos de sonido y modificar la resolución.

También se ha implementado la funcionalidad de pausa. Al activar la pausa, además de pausar el nivel, se muestra un menú que permite modificar el volumen de la música, los efectos de

sonido, y se muestran además 2 botones, uno para volver al nivel (también se puede volver presionando la tecla con la que se abrió el menú de pausa) y otro para reiniciar la partida, volviendo al nivel 1.

Se ha implementado el videojuego en 2 resoluciones, 1280×720 y $1880x1030$.

Por último, se ha añadido el sonido estéreo al ser golpeado por un enemigo, de forma que si la colisión se produce por la izquierda suena el audífono izquierdo y viceversa.

2.5. Manual de usuario

2.5.1. Interfaz del nivel

- **Indicador de corazones:** En la esquina superior izquierda de la pantalla del nivel, se muestran los corazones que le quedan al tejón. Cada vez que recibe un golpe de un enemigo, se le resta 1 corazón. Figura 15 (1).

- **Barra de energía:** Situada debajo del indicador de corazones, la barra de energía indica cuánta energía le queda al tejón. La energía se consume si el tejón va rodando. Figura 15 (2).

- **Indicador de vidas:** Debajo de la barra de energía se muestra el número de vidas que le quedan al tejón. Si el tejón se cae, o se le acaban todos los corazones, pierde una vida. Si se acaban todas las vidas, el tejón volverá al nivel 1. Figura 15 (3).

- **Contador de monedas:** Por último, en la esquina superior derecha, se muestra el número de monedas que tiene el tejón durante la partida. Cada 25 monedas recogidas, el tejón recupera 1 vida. Si el tejón se cae, se pierden todas las monedas obtenidas hasta el momento. Figura 15 (4).

2.5.2. Controles

- **Movimiento lateral del tejón:** Flechas izquierda y derecha.

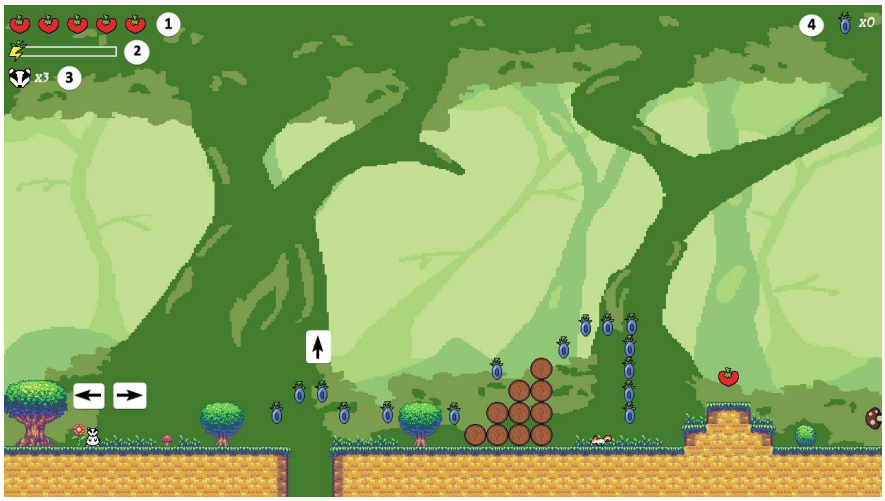
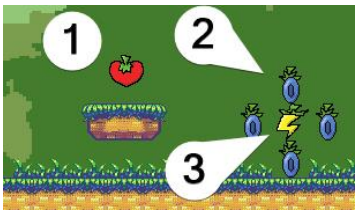


Figura 15: Controles del nivel

- **Saltar:** Flecha de arriba.
- **Rodar:** SHIFT.
- **Pausar el juego:** Tecla P.

2.5.3. Bayas

- **Baya de vida:** La baya de vida permite al tejón recuperar 1 corazón. Si tiene los corazones al máximo, no pasará nada. Figura 16 (1).
- **Moneda:** Monedas que se pueden recoger a lo largo de los niveles. Figura 16 (2).
- **Baya de energía:** El tejón recupera toda la energía al cogerla. Si la barra de energía ya está llena al máximo, no pasará nada. Figura 16 (3).
- **Baya de rabia:** El tejón entrará en el estado de rabia. Mientras se encuentre en este estado, no recibirá daño de ningún enemigo, ni se consumirá la energía al rodar. Figura 16b.



(a) Bayas de vida, energía y monedas

(b) Baya de rabia

Figura 16: Bayas del nivel

2.5.4. Enemigos

- **Zorro:** El zorro camina de un lado a otro en el nivel. Si el tejón se choca con él, se hará daño. Para eliminarlo, el tejón tendrá que saltar encima de él, chocar con él mientras está rodando o en el estado de rabia.

- **Erizo:** Al igual que el zorro, el erizo camina de un lado a otro. Pero, si el tejón salta encima de él, se hará daño. Para eliminarlo, el tejón tiene que chocar con él mientras está rodando o en el estado de rabia.

- **Ardilla:** La ardilla lanza bellotas en la dirección del tejón. El tejón se hará daño si recibe un golpe de alguna bellota, o si se choca con la ardilla. Para eliminar la ardilla, basta con saltar encima de ella, chocar con ella mientras el tejón está rodando, o, en el estado de rabia.

- **Seta:** La seta dispara esporas en la dirección del tejón. La seta no le hará daño al tejón, pero las esporas sí. No se puede eliminar, pero el tejón puede saltar encima de ella para usarla como plataforma y llegar a lugares altos.

- **Murciélago:** A diferencia de los demás enemigos, el murciélago se mueve volando de un lado a otro del nivel. Si el tejón se choca con él, se hará daño. Para eliminarlo, el tejón tiene que saltar encima de él, chocar contra él mientras rueda, o chocar contra él en el estado de rabia.

- **Oso:** El oso es el jefe final del juego. Para eliminarlo, el tejón tiene que quitarle todas las vidas, que se muestran en la parte superior de la pantalla. Para quitarle una vida,

el tejón tiene que saltar encima de él.

2.5.5. Mecánicas del juego

- **Rocas:** En algunos niveles hay rocas que bloquean el camino. Para seguir adelante, el tejón tendrá que rodar para romperlas (Figura 17).

- **Bandera:** Al final de cada nivel hay una bandera para pasar a siguiente nivel. Bastará con tocarla para avanzar en la aventura (Figura 17).

(a) Rocas

(b) Bandera del final del nivel

Figura 17: Mecánicas del juego

2.5.6. Menú de ajustes

- **Modificar el volumen de la música y efectos de sonido:** En la pantalla de ajustes del juego, se puede modificar tanto el volumen de la música del juego como de los efectos de sonido, para que se ajusten a las preferencias de usuario (Figura 18 (1)(2)).

- **Cambiar la dificultad:** Se puede cambiar la dificultad del juego, permitiendo las dificultades Fácil y Difícil (Figura 18 (3)).

- **Cambiar la resolución:** Al igual que la dificultad, también se puede modificar la resolución del juego. Las resoluciones disponibles son: 1280×720 y 1880×1030 (Figura 18 (4)).

- **Volver a la pantalla de título:** Para regresar a la pantalla de título (Figura 18 (5)).

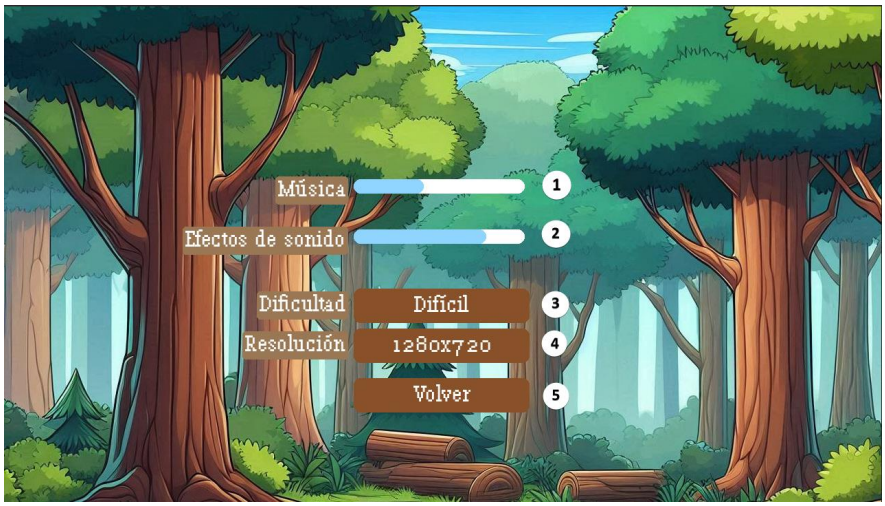


Figura 18: Menú de ajustes

2.5.7. Menú de pausa

- **Modificar el volumen de la música y efectos de sonido:** Al igual que en el menú de ajustes, se puede modificar el volumen de la música y efectos de sonido durante la partida (Figura 19 (1)(2)).

- **Continuar:** Salir del menú de pausa y continuar jugando en el nivel actual (Figura 19 (3)).

- **Reiniciar partida:** Se reinicia la partida, y se vuelve a empezar desde el principio, es decir, desde el primer nivel (Figura 19 (4)).

2.6. Manual de instalación

Para poder jugar al videojuego Tejón Jumper, es necesario cumplir con una serie de requisitos e instalar algunas dependencias. A continuación, se detallan los pasos necesarios:

Requisitos previos

- Tener instalado Python 3.12 o superior.

- Tener *pip* instalado y actualizado para la gestión de paquetes.

Figura 19: Menú de pausa

- Opcionalmente, se recomienda utilizar un entorno virtual para evitar conflictos entre dependencias.

Uso de entorno virtual (opcional)

Se recomienda crear un entorno virtual para el proyecto. Para ello, ejecutar el siguiente comando dentro del directorio del juego:

```
python -m venv venv
```

Luego, para activar el entorno virtual:

- En Windows: `venv\scripts\activate`
- En Linux y macOS: `source venv/bin/activate`

Instalación de dependencias

Es posible instalar todas las dependencias del juego utilizando el archivo requirements.txt, que contiene las versiones necesarias de cada paquete.

Para instalar los paquetes automáticamente, se debe ejecutar el siguiente comando en la terminal, dentro del directorio donde se encuentra el archivo:

```
pip install -r requirements.txt
```

Ejecución del juego

Para ejecutar el juego, basta con situarse en la carpeta del proyecto y ejecutar el archivo principal con el siguiente comando:

```
python src/main.py
```

2.7. Reporte de bugs

Se identificaron los siguientes bugs:

- **Pantalla flotante en Windows 11.** Al mover la ventana en Windows, el juego se queda bloqueado y, al soltarla, se producen comportamientos extraños relativos al movimiento de los enemigos y de los proyectiles.

  Este error únicamente se ha reportado en el sistema operativo Windows 11, pero es posible que también suceda en otros sistemas operativos en los que no se ha testeado el videojuego, como Windows 10 o MacOS.

- **Movimiento del murciélago.** En ciertas ocasiones, el murciélago se queda bloqueado cambiando infinitamente de posición en la posición más alta de su trayectoria.

  Este bug ha sido reportado en muy pocas ocasiones y únicamente ejecutando el videojuego en Windows 11, por lo que es posible que anteriormente se haya movido la ventana y este bug sea causado por el bug mencionado anteriormente.

- **Barras de sonido.** Las barras de sonido, al modificarse, pueden presentar pequeños bordes al final de ella en color azul, los cuales deberían estar en blanco, así como otros bugs visuales como que el final de la barra se muestra de forma cuadrada, o redonda pero con pequeñas secciones azules.

  Esto se debe a que las barras de sonido están siendo implementas utilizando la clase

pygame.rect.Rect, la cual parece no estar planteada para el diseño de interfaces.

- **Movimiento del oso.** En algunas ocasiones, si el oso salta y choca contra una pared, al realizar el cambio de dirección vuelve a saltar sin tocar el suelo, lo que produce que el oso salte mucho más de lo que debería. Este bug se ha reportado en muy pocas ocasiones, por lo que resulta difícil comprobar cuál es su causa exacta.

- **Vidas del oso.** En ciertas ocasiones, el oso sigue vivo teniendo 0 corazones, lo que obliga al jugador a inflingir daño una vez más para eliminarlo. Este bug es poco común por lo que no se ha identificado una posible causa.

- **Tamaño de ventana en la resolución grande.** En la resolución más grande de las dos (1880×1030), el tamaño de ventana es mucho más grande de lo que debería ser si el escalado de pantalla está activado en los ajustes del sistema operativo.

  El efecto de este bug es que, si, por ejemplo, se juega al juego en una pantalla (1920×1080) y tenemos un escalado de, por ejemplo, el 125 %, entonces la ventana se mostrará más grande que el monitor, a pesar de que, evidentemente, la resolución de la pantalla es mayor que el tamaño de la ventana, lo que hace que no se pueda jugar, puesto que no se ve la ventana del juego al completo. Por lo tanto, si se desea jugar en esta resolución en un monitor con esa resolución, se recomienda desactivar el escalado de pantalla en los ajustes del sistema operativo.

  Todo apunta a que este bug sucede porque el escalado del sistema operativo se aplica a la ventana del juego, cuando no debería, por lo que es posible que este bug también ocurra con la resolución pequeña si se utilizan pantallas con baja resolución.

- **Colisiones con el jugador.** Es posible que el sprite del jugador colisione desde la parte superior con otro sprite (como por ejemplo un proyectil o una plataforma) sin que visualmente el dibujo del tejón llegue a tocar el del otro sprite.

  Este comportamiento se debe a que el rectángulo asociado a la imagen tiene un tamaño cuadrado, (por ejemplo de 32×32 en la resolución pequeña), mientras que el dibujo del

tejón en movimiento presenta una forma más rectangular. Por lo tanto, para solucionar este bug se debería de crear un rectángulo distinto para cuando el jugador está en movimiento y ajustarlo mejor al tamaño de la imagen.