DIMENSIONS

Daniel Quintillán Iván García Jorge Paz

March 2021

Índice

1. Desarrollo técnico 2D

1.1. Descripción

En términos generales, *Dimensions* es un videojuego de desplazamiento lateral (side scroller) con puzzles a resolver a través de sus distintas mecánicas. Estos puzzles combinan la exploración, modificación del nivel y el uso de portales con el desafío habitual de evitar enemigos y saltar obstáculos.

1.1.1. Personajes jugables

Figura 1: La protagonista

La protagonista de este videojuego nos permitirá explorar los laboratorios *Closure* mediante un desplazamiento de plataformas clásico, con saltos, escaleras y *sprint*.

Inicialmente el *gameplay* estará orientado a la interacción con paneles, donde el sistema de gafas de realidad aumentada de la protagonista le permitirá detectar servidores vulnerables. Más adelante, la *portal gun* jugará un papel crucial. En ambos casos, la protagonista empleará sus habilidades de sigilo, ya que su misión depende totalmente de no ser detectada.

1.1.2. Enemigos

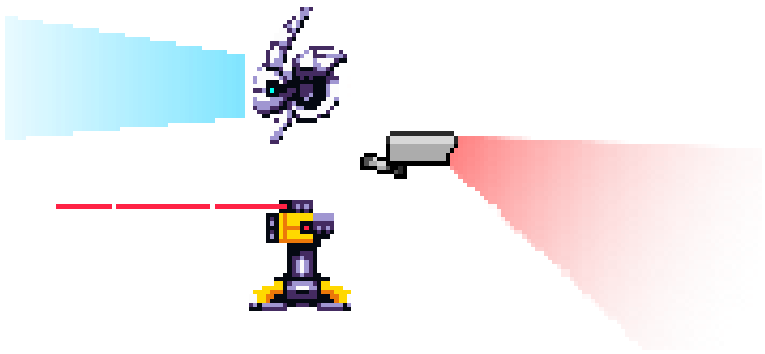


Figura 2: Enemigos

El sistema de seguridad de los laboratorios se compone de varios robots y cámaras de seguridad.

1.1.2.1. Dron terrestre Flota a nivel de suelo y patrulla los pasillos, detectando cualquier intruso que aparezca delante. Su escáner paramétrico precisa de proyección activa para su funcionamiento. La protagonista se aprovechará de la iluminación que genera este escáner para saber qué zonas evitar.

1.1.2.2. Dron aéreo Vuela cerca del techo para ofrecer mayor cobertura en zonas donde la seguridad es crucial. Emplea el mismo escáner que el dron terrestre. Cubre un área vertical importante, ondulando para mejorar su barrido. Sin embargo, tiene puntos ciegos al girar.

1.1.2.3. Torreta Centinela estacionario con un láser unidireccional para detectar intrusos. Vulnerable a ataques cibernéticos.

1.1.2.4. Cámaras Barren las habitaciones y detectan intrusos de manera inmediata. Utilizan una tecnología de escáner similar a la empleada por los drones.

1.1.3. Objetos

1.1.3.1. *Portal gun* Siendo el santo grial de los laboratorios *Closure*, la *Portal Gun* es la culminiación de años de investigación, y es resguardada como tal. Aunque se desconocen sus posibles efectos secundarios, es capaz de conectar regiones distantes del espacio a través de portales.

Para generar los portales, la *Portal Gun* dispara una sustancia indeterminada que produce alteraciones en el espacio al impactar con el material adecuado.

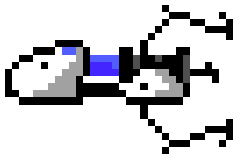


Figura 3: Portal Gun

1.1.3.2. Paneles de control La principal vía de interacción con el nivel en cada fase. Cada panel provocará un efecto distinto, por lo que la protagonista deberá prestar atención a las alteraciones en el entorno y cómo permiten avanzar en cada puzzle.

El sistema de realidad aumentada marcará con una flecha verde oscilante aquellos paneles que parecen vulnerables frente a un ataque.

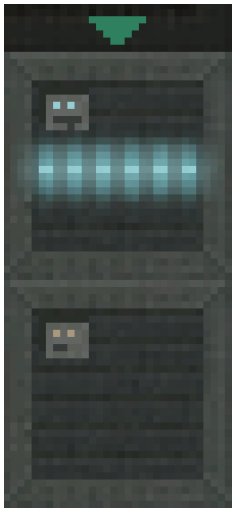


Figura 4: Panel de control

1.1.3.3. ***Portal walls*** Reciben este nombre las paredes de los laboratorios Closure que contienen el material adecuado para los portales. Curiosamente, son los bloques de construcción más vistosos de cada nivel.

Figura 5: Portal wall

1.1.4. Diseño

1.1.4.1. Guión La aventura comienza directamente en los laboratorios, en una zona de acceso restringido. A pesar de que nadie espera una intrusión, los sistemas de seguridad funcionan con normalidad. O funcionaban, hasta que la protagonista comience a alterar su funcionamiento desactivando robots, abriendo compuertas y cegando sistemas de vigilancia. En la zona más profunda y resguardada de los laboratorios obtendrá la protagonista la *portal gun*.

Una vez la protagonista consigue la pistola, el objetivo consiste en huir de los laboratorios. Para ello se han de resolver una serie de puzzles que precisarán de la creación de portales para su resolución. Sin embargo, en la zona superior y final de los laboratorios falta la escalera necesaria para terminar la escapatoria. Armada con la pistola de portales, la protagonista empleará la velocidad que proporciona la caída del silo central de los laboratorios para terminar su fuga.

Al final de los laboratorios nos espera un imponente portal distinto a todos los anteriores. El otro lado del portal precisará de otra aventura y una dimensión nueva para ser explorado.

1.1.4.2. Reglas La regla principal del juego es evitar ser detectado. Las posibles vías de detección son la colisión con los enemigos o con sus escáneres. Sin embargo, si la detección se produce en el segundo nivel, se ofrecerá al jugador la posibilidad de continuar al principio del nivel, sin necesidad de repetir el primero.

1.1.4.3. Mecánicas El diseño del juego se centra en forzar al usuario a explorar gran parte del nivel antes de llegar a completarlo. Esto se consigue gracias a varios factores:

Obstáculos: Hay varias puertas y trampillas que bloquean el paso a zonas importantes del nivel hasta que el usuario consigue abrirlas mediante paneles de control. Esto fuerza al usuario a hacer *backtracking* para avanzar en el nivel.

Cámaras: Otro de los objetos con los que podemos interactuar mediante los paneles de control. las cámaras también pueden bloquear el camino del usuario hasta que se desactiven.

Ocultación del nivel: El *scrolling* bidimensional limita la cantidad de información disponible para el jugador en un momento dado.

Paneles mudos: Todos los paneles avisan de que se puede interactuar con ellos, pero ninguno especifica qué pasará cuando se active, por lo que el usuario se ve forzado a explorar de nuevo el nivel para identificar los efectos

de su acción en el entorno. Esto es una decisión consciente que, junto con todas las demás, fomenta el *backtracking* y la exploración.

Jugabilidad no lineal: Para lograr un *gameplay* más activo, que fomente que el jugador explore el nivel por completo y retenga más información del juego tras completarlo, los niveles presentan una disposición *no lineal*. Esto significa que, para avanzar en el juego, el jugador debe resolver puzles que tienen efectos en zonas opuestas del nivel, y debe realizar *backtracking* de forma asidua para disponer de la información suficiente y saber cómo progresar en cada fase. Esto provoca que el usuario necesite ver el nivel como un todo y no como la colección de pequeños puzzles o desafíos de los que está compuesto, a la vez que permite lograr una mayor cantidad de *gameplay* para el jugador como alternativa a desarrollar multitud de distintos niveles lineales, como demuestra el auge de los juegos no-lineales y *open world* de los últimos años.

1.2. Fases

1.2.1. Fase 1

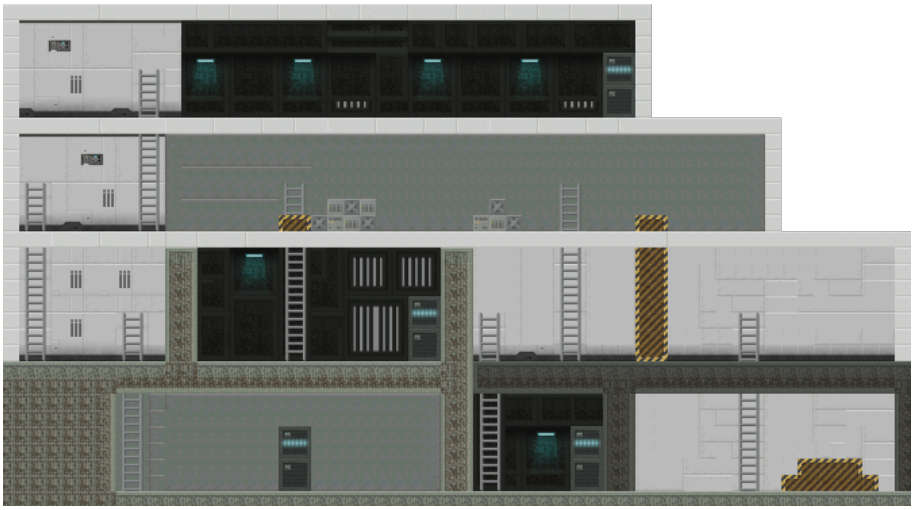


Figura 6: Vista del nivel 1 en *Tiled*

El juego empieza con un nivel de sigilo en el que el objetivo del jugador es conseguir llegar hasta la pistola de portales sin ser detectado por el sistema de seguridad del laboratorio.

1.2.2. Fase 2

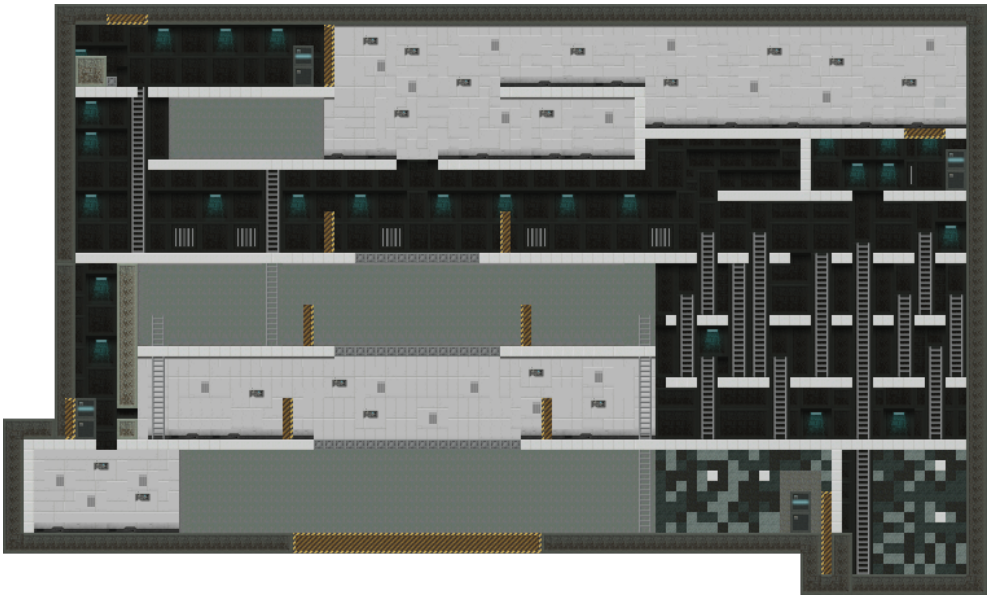


Figura 7: Vista del nivel 2 en *Tiled*

En la segunda fase se introduce la mecánica de los portales en la que se basará el diseño del nivel. El jugador tiene que conseguir encontrar la salida y utilizar los portales para abrirse camino hacia ella.

1.3. Aspectos destacables

1.3.1. Implementación de niveles mediante *tiling*

En vez de emplear una estrategia *ad-hoc* de creación de enemigos, muros, etc., hemos empleado un sistema de diseño de mapas mediante *tiles* que a través de un editor facilita sustancialmente la creación de niveles.

Para ello, hemos implementado el concepto de *Phase*, una clase con un *parser* que soporta un conjunto de primitivas a partir de las cuales se consti-

tuyen los niveles. Las primitivas se subdividen en *layers*, capas que contienen los *tiles* y determinan su orden de dibujado y *objects*, los elementos que interactúan, como el jugador, los enemigos o los paneles.

De este modo, una vez se ha implementado la funcionalidad de un tipo de elemento, este es libre de ser reutilizado tantas veces y en tantos niveles como se desee. El ejemplo paradigmático de este sistema son los paneles.

Los paneles emplean las propiedades de los *objects* en el programa *Tiled* para controlar otros objetos. El nombre de la propiedad del panel indica la acción a realizar, por ejemplo añadir un muro o desactivar una cámara. Es importante destacar que las acciones son totalmente independientes del objeto, es decir, la misma funcionalidad que se emplea para eliminar un muro del juego puede también eliminar un enemigo. A mayores, los paneles soportan varias acciones simultáneas, no necesariamente del mismo tipo.. En resumen, la implementación de niveles mediante *tiling* desacopla totalmente los detalles de implementación del diseño de niveles, ofreciendo una herramienta de edición con primitivas similar a la implementada por muchos juegos actuales, y que incluso abre la puerta a contenido generado por el usuario.

1.3.2. Doble *scrolling* con cámara *lazy*

El desplazamiento a través de los niveles cuenta con un sistema de doble scrolling que carga dinámicamente los *tiles* que están ubicados en la zona a la que apunta la cámara del juego.

Tras la evaluación del rendimiento con una herramienta de *profiling*, se decidió optimizar el dibujo de los *tiles*, de manera que únicamente se pinten los que se precisen en cada momento. Esta implementación es escalable y permite implementar niveles de tamaño prácticamente arbitrario.

1.3.3. Física de portales

La implementación de un sistema de física de portales 2D constituye un añadido original al género de plataformas.

El sistema cuenta con un proceso de creación de portales a través de rayos disparados en ocho direcciones. Estos rayos deben responder correctamente a la multitud de casos de colisión posibles; por ejemplo un rayo disparado en diagonal no puede contar únicamente con su dirección para calcular la dirección del portal creado, debe estimarla en la colisión.

Para mantener un *gameplay* interesante, los portales únicamente pueden ser creados en superficies del material adecuado. A mayores, para facilitar el diseño de niveles, los portales automáticamente se alinean a la *grid* del mapa, y en caso de impactar en un borde del material son capaces de recolocarse de manera que no se desborde el material.

Finalmente, la propia mecánica de teletransporte incorpora ajustes para mejorar la jugabilidad. Los portales colocados en el suelo impulsarán más al jugador para permitir salir de ellos. El detalle más importante de la física del teletransporte es que conserva la velocidad de entrada del portal, con una eficiencia del 90 % para evitar bucles infinitos. De este modo se permite la mecánica final, donde la aceleración de la gravedad permitirá obtener la velocidad necesaria para terminar el nivel únicamente a partir de la altura de la última compuerta.

1.4. Manual de usuario

1.4.1. Dependencias

Las librerías necesarias para ejecutar el juego son:

- Pygame
- PyTMX: *parser* de archivos TMX (*tilemaps*)

Se encuentran reflejadas en el archivo *requirements.txt*.

1.4.2. Jugabilidad

Figura 8: Creación de portales apuntando en diagonal

1.4.2.1. ***Portal Gun*** La *Portal Gun* se controla con el ratón. Cuando desenfunda, el personaje se queda quieto y puede disparar portales en ocho direcciones: arriba y abajo, izquierda y derecha y las cuatro diagonales. Los portales solo pueden crearse en ciertas superficies, y aunque se disparen en diagonal, aparecerán en horizontal o en vertical dependiendo de si se crean en una pared, en un suelo o en un techo.
Para que un portal cumpla su función, se necesita disparar a dos superficies de portales diferentes. El segundo disparo conectará las dos superficies de manera que al entrar por un portal, el personaje aparece en el otro **conservando su velocidad**. Esto último será la clave para resolver el segundo nivel.

1.4.3. Controles

W	Subir escaleras
S	Bajar escaleras
A	Izquierda
D	Derecha
SHIFT	Correr
SPACEBAR	Saltar
X	Soltar la escalera
ESC	Pausa
E	Interactuar
LMB	Disparar portal
RMB	Apuntar con la pistola
Movimiento del ratón	Dirección de apuntado

1.4.4. *Visual cues*

En el entorno de los niveles aparecen indicadores verdes cuyo movimiento indica que un panel es interactuable.

1.4.5. Menús

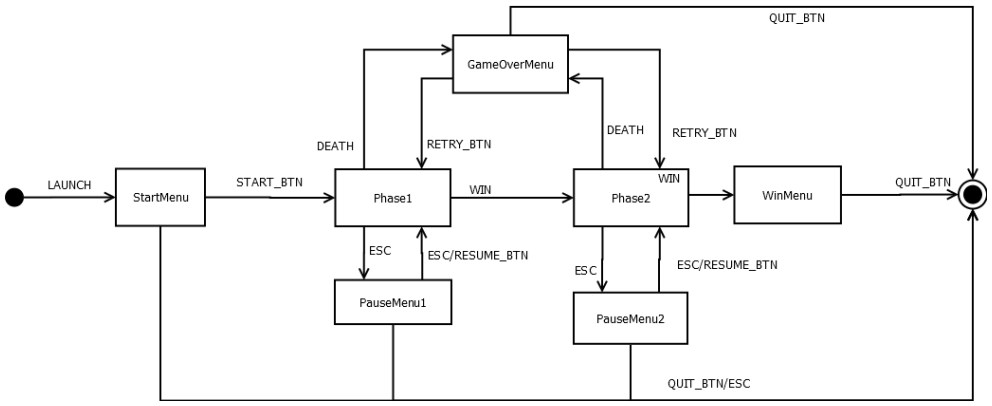


Figura 9: Diagrama de estados con las transiciones entre escenas

El flujo del juego se articula mediante menús y niveles.

Figura 10: Menú principal

En todo momento se puede salir del juego cerrando la ventana, y todos los menús dan la opción de salir mediante un botón. El menú de pausa permite parar el la fase conservando su estado y reanudarla cuando el usuario lo crea conveniente. El menú de *Game Over* permite instanciar de nuevo la misma fase en la que el usuario perdió. El menú de victoria permite volver a empezar el juego desde la primera fase. Las fases están conectadas directamente entre ellas sin ningún menú por el medio.

1.5. Proceso de desarrollo

Diseño incremental por niveles que precisan de nuevas *features*. Introducción de los portales en el segundo nivel. Hemos desarrollado, mediante un proceso incremental, un sistema de creación de niveles dirigido por datos. Es por esto que los detalles de diseño e implementación del sistema se discutirán haciendo referencia al soporte de fases genéricas, en vez de comentar las particularidades de cada nivel por separado.

1.5.1. Coordinación dentro del equipo

En este proyecto se ha utilizado la metodología de desarrollo ágil *Kanban*. Concretamente, una tabla con estas cinco columnas para asignar y actualizar tareas de manera dinámica.

Pending	*In progress*	*On hold*	*Completed*

Cada iteración del desarrollo comienza con un análisis de las tareas a realizar. A continuación se desgranan en tareas más pequeñas y se colocan en la columna de tareas pendientes. Una reunión del equipo aclara la prioridad con las que estas tareas deben ejecutarse y a quién se le asignan, moviéndolas a la columna de *In Progress*. De manera simultánea, se realizan hasta que se terminan o algún imprevisto provoca su interrupción.

1.5.2. Análisis

Desde un primer momento, nuestro objetivo es utilizar los *TiledMaps* para almacenar toda la información que podamos sobre los niveles y los elementos que los componen, desde muros y enemigos hasta el propio personaje jugable. Como vimos en clase, la arquitectura dirigida por datos tiene varias ventajas con respecto a una implementación *ad hoc*, siendo la principal el desacoplamiento de los comportamientos específicos de cada nivel de las primitivas internas del código.

No obstante, el objetivo del proyecto es llevar a término la mayoría de los requisitos plasmados en el desarrollo artístico, por lo que algunas decisiones priorizan la sencillez sobre la flexibilidad.

Partiendo de esa base, cada iteración analiza el desarrollo artístico e implementa el conjunto de *features* que en el momento se consideran prioritarias de cara a la reunión con el *Product Owner*: empezando por el primer nivel, seguido de la estructura de los menús, y concluyendo con las mecánicas de portales del segundo nivel.

1.5.3. Diseño

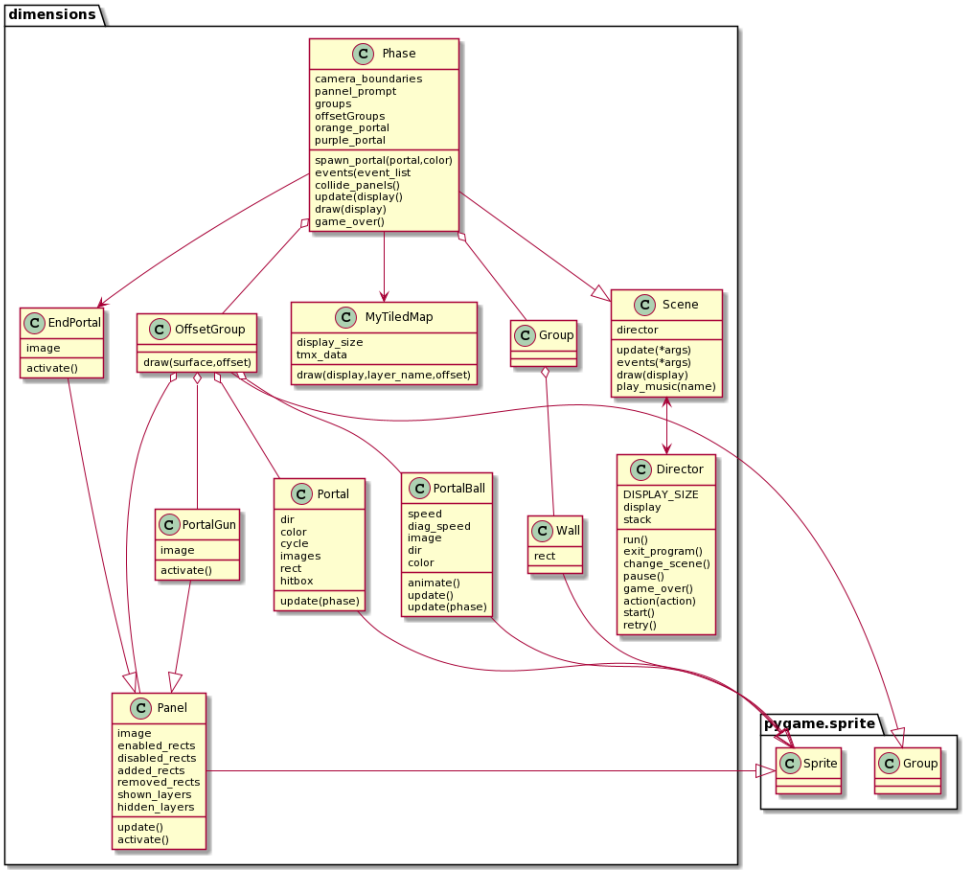


Figura 11: Diagrama de clases de la fase

Phase es una subclase de *Scene* responsable de encapsular el comportamiento y el estado de cada uno de los niveles. La elección de una única clase en vez de una fase genérica con varias que la especialicen viene motivada por la arquitectura dirigida por datos. La instancia de *Phase* recibe en su constructor el nombre del archivo del que tiene que extraer la información del nivel e instancia un *MyTiledMap* cuya única responsabilidad es cargar el archivo y dibujar las capas del escenario que se le indique. El objeto *Phase* extrae del archivo todos los objetos y le pide al mapa que dibuje las capas correspondientes en cada ciclo.

De esta manera, la mayoría del trabajo a la hora de crear o modificar niveles se hace en el editor de *TiledMaps* en vez de en el código. No obstante, tener la funcionalidad de todos los niveles en una sola clase puede ser difícil de mantener a medida que el proyecto escala. Además, podría ser más mantenible que la responsabilidad del *parsing* recayera sobre *MyTiledMap* y que la fase inicializase su estado y dibujase las distintas capas accediendo a atributos de la clase *MyTiledMap*

La interactividad con el entorno se articula mediante dos elementos: los grupos y los paneles. Por una parte, los grupos permiten manipular varios *Sprites* de manera cómoda, mientras que su especialización en *OffsetGroups* facilita aplicar el desplazamiento por *scrolling* a los *Sprites* que se van a dibujar. Por otra parte, la fase añade a cada panel las listas de los objetos y capas que se ven afectados por su activación. En el caso de *EndPortal* y *PortalBall*, su activación simplemente llama al director para que cambie de nivel.

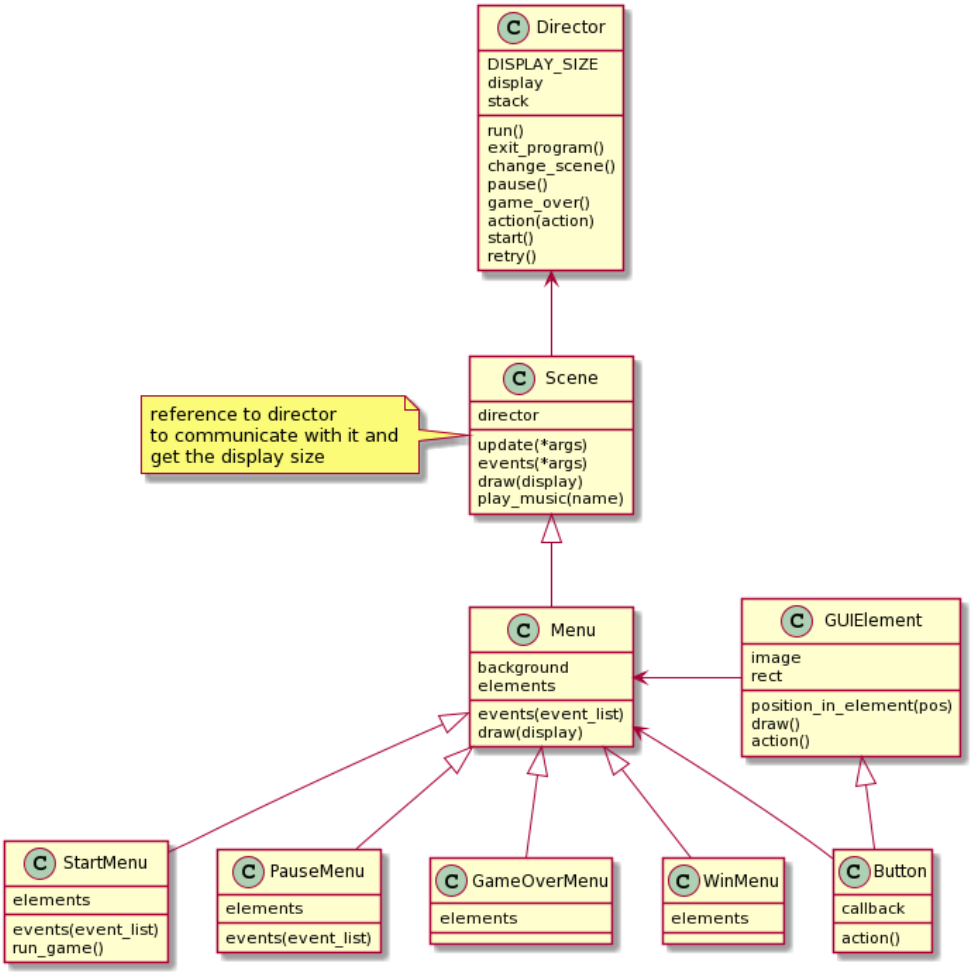


Figura 12: Diagrama de clases de los menús

Los menús son otra subclase de *Scene* que articulan la transición entre fases. La maquetación de cada menú tiene dos elementos: la imagen de fondo y los *GUIElements*. Los *GUIElements* son imágenes que ocupan una posición en el menú. El menú se encarga de manejar los eventos para comprobar si el usuario hace click en uno de los elementos y ejecutar la acción correspondiente. Si se trata de instancias de *GUIElement* (como podría ser el texto del título) la acción no existe, pero en el caso de los *Buttons* la acción es un callback que su constructor recibe como argumento.

Las subclases de *Menu* se diferencian por los *GUIElements* que contienen

y la manera en la que reaccionan a los eventos. Hemos prescindido de la estructura con múltiples pantallas para simplificar el diseño y centrarnos en otras *features* más importantes.

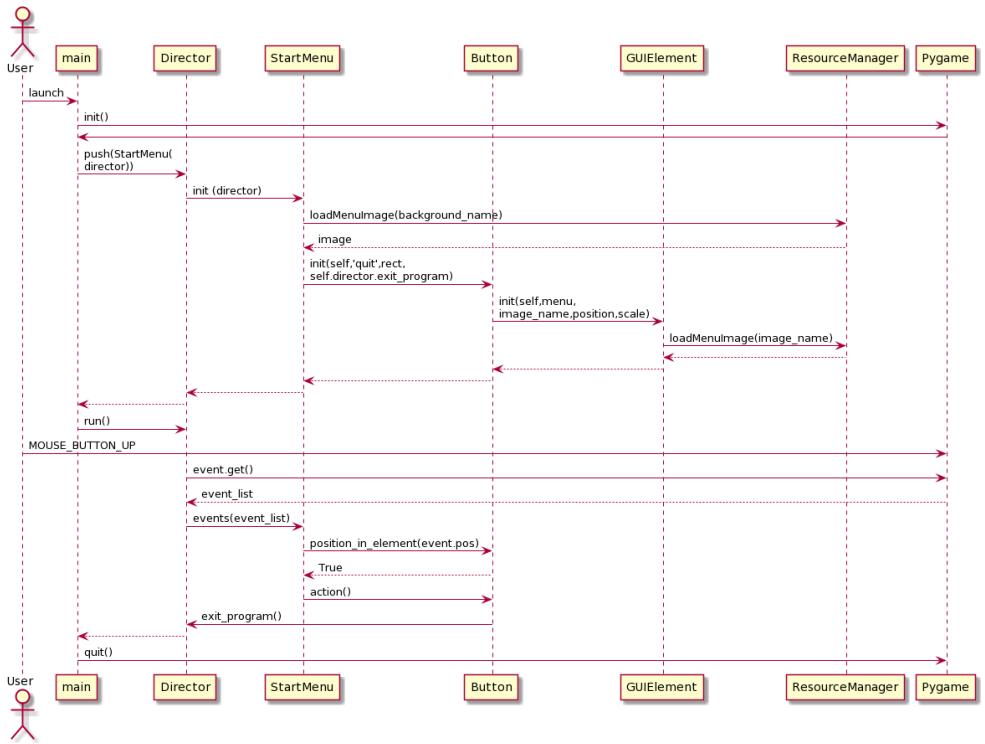


Figura 13: Diagrama de secuencia de instanciar un menú y salir del juego

En este diagrama se puede apreciar la interacción entre el director y las escenas. Concretamente, se detallan los pasos más importantes a la hora de instanciar el menú de inicio y utilizarlo para salir del juego. En general, las escenas necesitan una comunicación bidireccional con el director, ya que los niveles y menús necesitan poder comunicarle al director a qué otra escena transicionar y el director necesita saber en qué escena se encuentra el juego en un momento dado.

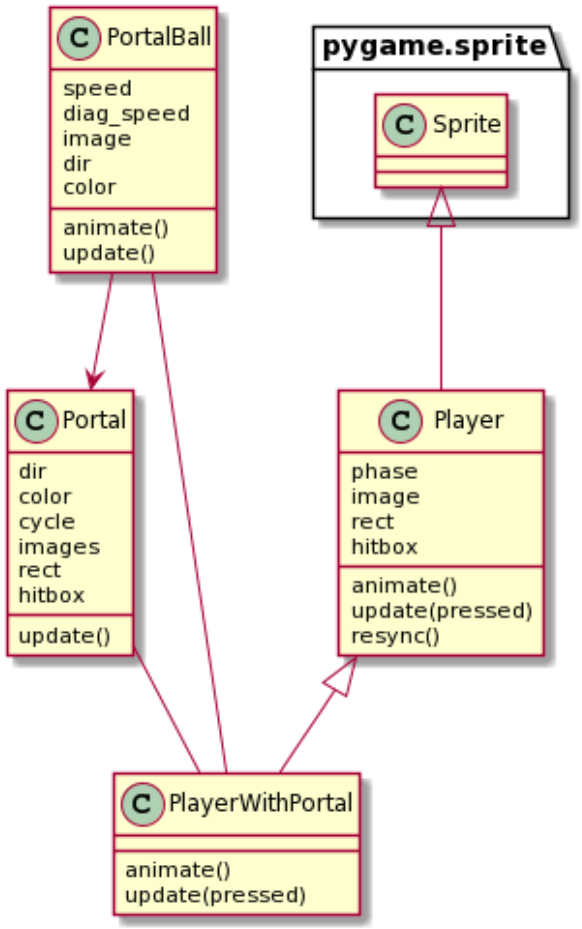


Figura 14: Diagrama de clases del jugador

El jugador es una subclase de *Sprite*, y por lo tanto se puede manejar mediante grupos desde la fase. Su método *update* requiere, además de las teclas pulsadas, información sobre un número arbitrario de grupos de la fase que pueden colisionar con el jugador, por lo que cada instancia de *Player* se guarda una referencia a su fase.

Se podría implementar un patrón decorador para que. una vez instanciado el jugador, se convierta en *PlayerWithPortal* de manera reversible, pero en este caso no es necesario ya que en el desarrollo artístico se especifica que el cambio se produce entre dos fases. *PlayerWithPortal* especializa a su superclase para permitir al usuario disparar *PortalBalls*, que cuando detectan una

colisión con el grupo de muros adecuado instancian un *Portal*. Este cambio también requiere que en el método *update* la subclase gestione colisiones con el grupo de los portales para implementar la mecánica del teletransporte.

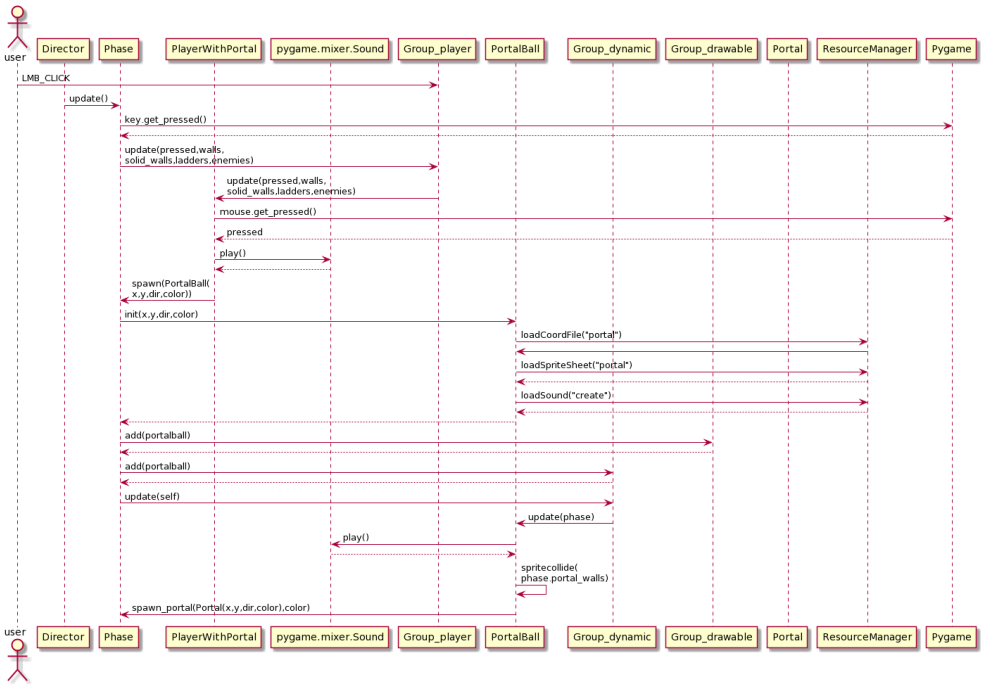


Figura 15: Diagrama de secuencia de la creación de un portal

En este diagrama se enumeran los pasos que tienen desde que el usuario dispara una *PortalBall* hasta que se genera el portal en el muro con el que colisiona.

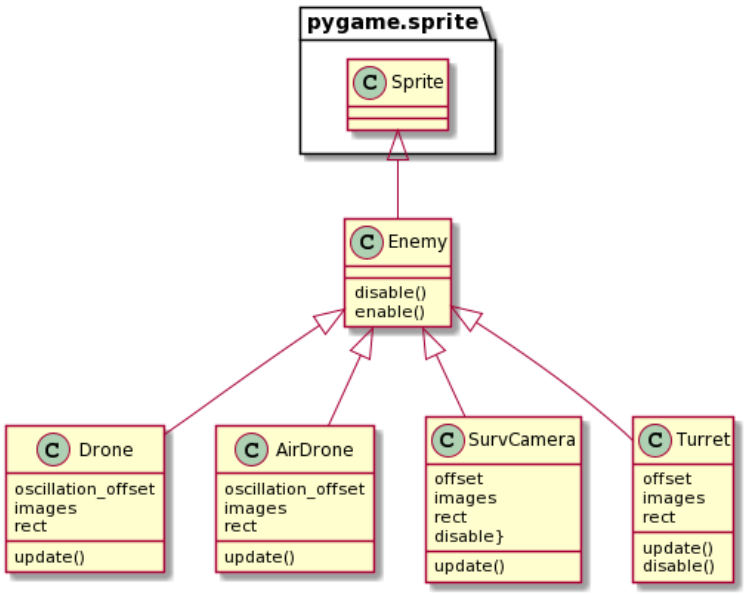


Figura 16: Diagrama de clases de los enemigos

La clase *Enemy* es un *Sprite* que ofrece a los grupos de la fase las funciones de activar y desactivar, aunque son sus subclases las que deben implementar estos métodos atendiendo a sus distintos comportamientos.

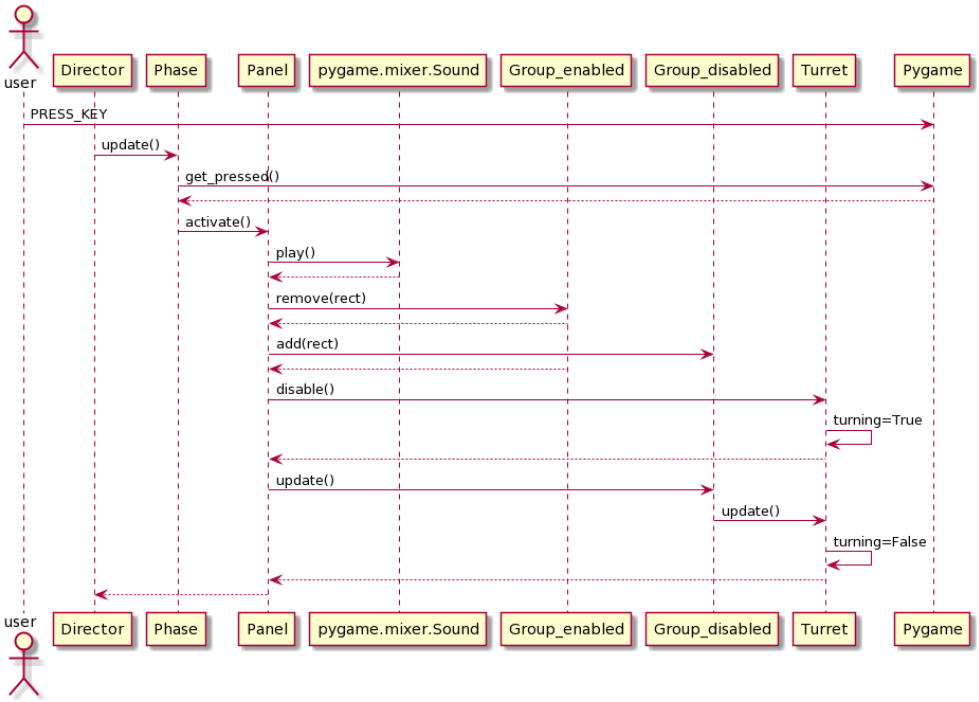


Figura 17: Diagrama de secuencia la desactivación de una torreta

En el caso concreto de la torreta, este diagrama ilustra todos los pasos desde que el usuario interactúa con el panel que la desactiva hasta que la torreta termina su animación de desactivación. El panel cambia al objeto *Turret* del grupo *enabled* al*disabled*, de manera que el *Player* no lo tenga en cuenta en su detección de colisiones. El atributo *turning* de la torreta es el responsable de activar y desactivar la animación. Al invocar el método *disable* de la torreta este *flag* se activa para que la animación inicie en el siguiente ciclo, y después de cierto número de ciclos se vuelve a desactivar.

1.5.3.1. Patrones de diseño

Arquitectura dirigida por datos Como hemos comentado a lo largo de la memoria, la implementación de *data-driven architecture* nos brinda las ventajas de independencia de la implementación respecto al diseño de niveles.

Polimorfismo en las acciones de los paneles Los objetos *panel* en los archivos TMX contienen sus acciones como propiedades. El nombre de la propiedad indica al *parser* la acción a realizar y el valor indica el objeto sobre el cual se aplica. La implementación de las acciones es agnóstica al tipo del objeto afectado, por ejemplo, el código de la acción *add* se aplica por igual al muros y enemigos. Se trata de una situación similar al polimorfismo basado en *generics*, como los algoritmos sobre vectores genéricos que emplean el mismo código para vectores de distintos tipos.

Resource manager Para optimizar el manejo de recursos, todos (excepto la música) se cargan mediante un gestor de recursos. El gestor contiene como atributo estático un diccionario mediante el cual el resto de clases pueden cargar archivos guardando una referencia a ellos una sola vez y recuperarlos por clave.

1.5.4. Detalles de implementación

- El *Resource manager* se complementa, cuando es posible, con una estrategia de carga estática que permite compartir recursos a nivel de clase. Al cargar y segmentar la *spritesheet* a nivel de clase, varias instancias de la misma clase comparten todas las imágenes de su animación en vez de tener que recortarlas individualmente.

- Las *Spritesheets* se segmentan extrayendo sus dimensiones y posición de unos archivos de coordenadas obtenidos con el programa TexturePacker.

- Debido al modo en el que se tratan los *Sprites* la distribución de la *Spritesheet* es ineficiente.

- Ya que varias clases en distintos archivos necesitan el mismo código para segmentar la *spritesheet* en base al archivo de coordenadas, esta función está en el archivo **utils.py** para que se pueda importar cuando haga falta.

- Debido a que los *Sprites* de algunos enemigos tienen gradientes de transparencia, las *Spritesheets* se guardan con el canal alfa en vez de utilizar una *colorkey*.

- Las colisiones con los enemigos se realizan por máscara y las de los muros, por el rectángulo del mismo contra una *hitbox* personalizada para el jugador que siempre tiene las mismas dimensiones.

- Sincronizacion por FPS. Como nos encontramos en un entorno bidimensional, donde es mucho más sencillo estimar la carga computacional que en un videojuego 3D, hemos optado por la sincronización por *framerate*. Esta decisión nos permite simplificar el código de físicas y proporcionar una experiencia precisa y consistente entre ordenadores, siempre que se cumplan los requisitos mínimos.

1.6. Informe de *bugs*

A continuación se listan los *bugs* encontrados en el videojuego, pendientes de resolución por limitaciones temporales.

- Hay rejillas transparentes en el nivel 1 que generan artefactos visuales al superponer estelas del jugador. Arreglarlo sería tan fácil como taparlas o poner algo opaco en el fondo, pero nos gusta el efecto que crea.

- Cuando se mantiene presionada una tecla de movimiento antes de que cargue el nivel, la jugadora aparece fuera del nivel. Este *bug* puede deberse al procesamiento de eventos.

- Al subir o bajar escaleras que estén cerca del borde del mapa es posible que el personaje se teletransporte fuera del mapa. El sistema de manejo de colisiones parece ser el culpable.

- Se pueden disparar dos portales al mismo sitio. La solución implicaría eliminar el antiguo tras detectar que la colisión del nuevo está en la misma región.

- El último portal colocado desaparece en cuanto se dispara una nueva *PortalBall*, independientemente de si esta va a general o no un portal nuevo. Esto puede provocar que el usuario se quede atascado en algunas partes del nivel al no poder volver deshacer su camino y se solucionaría retrasando la eliminación del último portal hasta que la *PortalBall* que está en el aire genere un portal nuevo.

- Cuando se genera un portal, si la pistola se encuentra demasiado cerca de un muro puede generar un portal al otro lado. Esto se debe a que el rayo se instancia hacia donde apunta la pistola. La solución implicaría un complejo sistema de detección de esta situación.

- Actualmente el nivel de opacidad necesario para la colisión con los escáneres no es distinto para cada tipo de enemigo, sino que emplea el valor por defecto de 127. Esto provoca que la detección de colisiones con los enemigos que tienen gradiente de transparencia tenga falsos negativos. La solución consistiría en la creación de atributos *mask* a medida para cada enemigo.

- Cuando se están subiendo las escaleras, se permite saltar. Si se mantiene presionado el botón de subir y el de saltar la jugadora ascenderá a la velocidad de salto continuamente. La solución consistiría en añadir un *cooldown* tras saltar desde una escalera.

- La animación de apuntar desde una escalera carece de imagen propia, habría que crear otros assets específicos.

- Al tratar de apuntar al caer se puede congelar la animación de caída. Para evitar esto, habría que crear animaciones de caída en las que se apunta y afinar el manejo de la animación de apuntar.

- Si al apuntar se sitúa el cursor en un píxel situado exactamente entre dos de las ocho direcciones posibles, el error de redondeo puede hacer que se parpadee entre las dos direcciones adyacentes. La solución implicaría tener un umbral distinto para cambiar de dirección dependiendo del sentido de giro.

- La detección de colisiones con la torreta desactivada provoca que el personaje se deslice dentro de la que sería su *hitbox* cuando debería pasar por detrás sin colisionar. La solución podría pasar por convertir la torreta desactivada en un muro.

- Si se colocan dos portales formando una esquina inferior (uno en el suelo y otro en una pered cercana) y el jugador se mete en el hueco entre ambos, se teletransportará constantemente y no podrá salir del bucle.

2. Anexo: Desarrollo artístico

Desarrollo artístico CIIE- *DIMENSIONS*

Iván García Fernández, Jorge Paz Ruza, Daniel Quintillán Quintillán

1. Antecedentes

a. Ambientación:

Año 2070. Elon Musk y su empresa SpaceXY lideran el viaje espacial comercial a Marte. La labor incansable de sus primeros clientes facilitó que el proceso de terraformación del planeta rojo durase menos de lo esperado, por lo que las colonias ya están perfectamente asentadas.

b. Historia

Gracias a una beca de SpaceXY, por fin consigues escapar del planeta Tierra y empezar tu carrera de investigación en el campo de la física teórica. El último proyecto del señor Musk lleva años en desarrollo, pero ni el público general ni los propios inversores conocen los detalles del mismo por ser altamente confidencial.

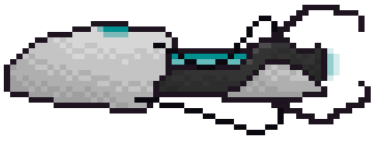
Tras descubrir grandes partes del código que ejecuta nuestro universo, los científicos de tu equipo descubren una combinación de estados que produce que la información se replique de manera no local. En idioma humano: teletransporte.

Sin embargo, tus teorías muestran que el teletransporte desestabiliza la ejecución del universo, y las alteraciones locales en la gravedad producidas por los experimentos que tanto se rumorean no te dejan dormir.

Tras ser ignorada (obviamente el teletransporte es lucrativo), decides resolverlo tú misma. Recuperarás un prototipo de la compañía para poder estudiarlo en detalle.

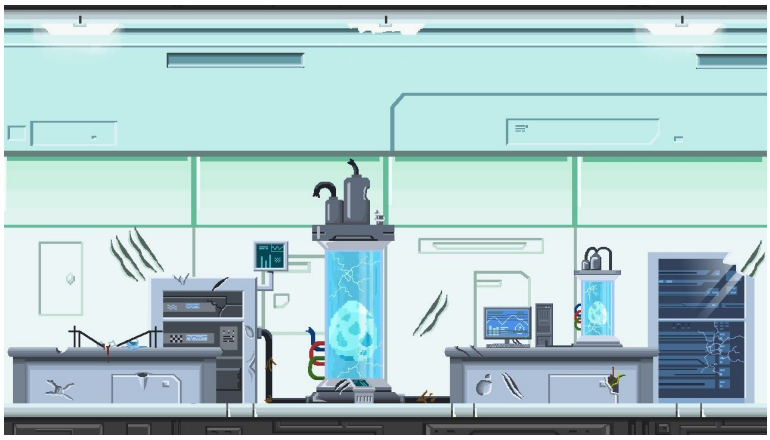
2. Otras características de la ambientación, y elementos que aparecerán en el juego

a. Objetos destacados:



i. **El arma de portales:**

b. Lugares:



i. **El laboratorio de Investigación:** La mayor parte de la acción del juego transcurre en los niveles más bajos del edificio, donde se encuentran todos los laboratorios. Son entornos estériles, protegidos y muy bien iluminados, así que tendrás que usar tu ingenio para pasar desapercibida.

ii. **Azotea:** Con la imponente metrópolis de fondo, llega el momento de enfrentarte al jefe final. Tendrás que usar el arma de portales con astucia para evitar sus ataques y derrotarle.

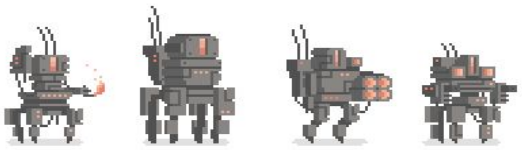
c. Personajes:

i. **Protagonista**:

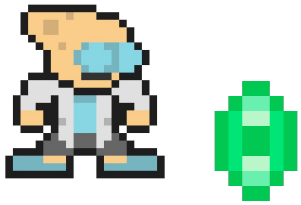
ii. **Enemigos:**

1. Cámaras que nos detectan
2. Drones que nos disparan

3. Androides que nos disparan



iii. **Boss final**: Elon Musk. A medida que los años pasaban, las sospechas de cómo el magnate se mantenía vivo se popularizaban con más frecuencia. Resulta que las esmeraldas de la mina que su padre explotó durante el Apartheid sudafricano tenían propiedades mágicas, y no dudará en utilizarlas para defender su imperio marciano a toda costa.



3. Guión de desarrollo y acción del juego

2D:

1. **Introducción no jugable:** Presentación de la historia
2. **Primera fase:** La protagonista entra en el edificio sin armas. En el primer nivel tiene que conseguir robar la pistola de portales esquivando ser vista por robots centinela y drones con cámaras. Para ello tendrá que cortar el suministro eléctrico de cada sala en la que vaya entrando.
3. **Segunda fase:** La protagonista se abre camino a través del segundo piso aprovechando la pistola para resolver puzzles y evadir a los enemigos, que ya conocen su plan.
4. **Interludio no jugable**(sonidos de pelea y subir escaleras)
5. **Boss final:** ya en la azotea, un Elon Musk decrépito interrumpe su presentación al vernos con el prototipo y tenemos que pelear con él ayudándonos de los portales (caen cajas del cielo porque es parte de la demostración y tenemos que conseguir que le hagan daño a él con los portales).
6. Epílogo no jugable:El arma se sobrecarga y hay una gran explosión. Cliffhanger para la parte en 3D

3D:

- Introducción no jugable: La explosión nos transportó a un mundo 3D
- Escena 1: el arma ahora dispara rayos de energía y los usamos para completar puzzles con circuitos y atacar a drones y androides mientras intentamos bajar de la azotea. Hay que recargar la pistola con Powerwalls (™) de la marca Voltios.

- Escena 2: Nos siguen persiguiendo dentro del edificio semiderruido mientras se oyen explosiones que nos indican que va a colapsar y tenemos que darnos prisa.