# *DIMENSIONS 3D*

Daniel Quintillán Iván García Jorge Paz

Mayo 2021

# Índice

**1. Desarrollo técnico 3D** **2**
1.1. Descripción . . . 2
1.1.1. Personajes jugables . . . 2
1.1.2. Enemigos . . . 3
1.1.3. Objetos . . . 4
1.1.4. Armas . . . 7
1.1.5. Diseño del juego . . . 8
1.2. Aspectos destacables . . . 13
1.2.1. Sistema de oleadas . . . 13
1.2.2. Sistema de pagos . . . 13
1.2.3. Concepto de habitación . . . 14
1.2.4. Habitaciones con *spawners* sincronizados con las puertas 14
1.2.5. Habitaciones trampa . . . 14
1.3. Manual de usuario . . . 15
1.3.1. Dependencias . . . 15
1.3.2. Jugabilidad . . . 15
1.3.3. Controles . . . 16
1.3.4. *HUD* . . . 17
1.3.5. Menús . . . 18
1.4. Proceso de desarrollo . . . 18
1.4.1. Coordinación dentro del equipo . . . 18
1.4.2. Análisis . . . 19
1.4.3. Diseño Software . . . 20
1.4.4. Detalles de implementación . . . 26

1.5. Informe de *bugs* . . . . . 27

**2. Anexo: Desarrollo artístico** **28**

# 1. Desarrollo técnico 3D

## 1.1. Descripción

En términos generales, *Dimensions* es un juego en primera persona que combina componentes de *Round Based Survival Game*, como las oleadas de enemigos y la progresión del entorno, con mecánicas propias de los *Arena Shooters*, como las salas cerradas y el movimiento fluido, así como elementos de juegos *3D Platforming*.

### 1.1.1. Personajes jugables

La protagonista de este videojuego nos permitirá explorar los laboratorios *Closure* mediante un desplazamiento de *shooter* clásico (*crouch*, *strafe*, salto y *sprint*), enriquecido con un componente muy característico de los *arena shooters*: un *jetpack*.

La progresión del mapa estará orientada a la interacción con paneles, donde el sistema de gafas de realidad aumentada de la protagonista le permitirá calcular el número de piezas necesario para obtener nuevas armas y abrir compuertas. Las piezas se podrán obtener a partir de los restos de los *robots* enemigos tras ser neutralizados.

### 1.1.2. Enemigos

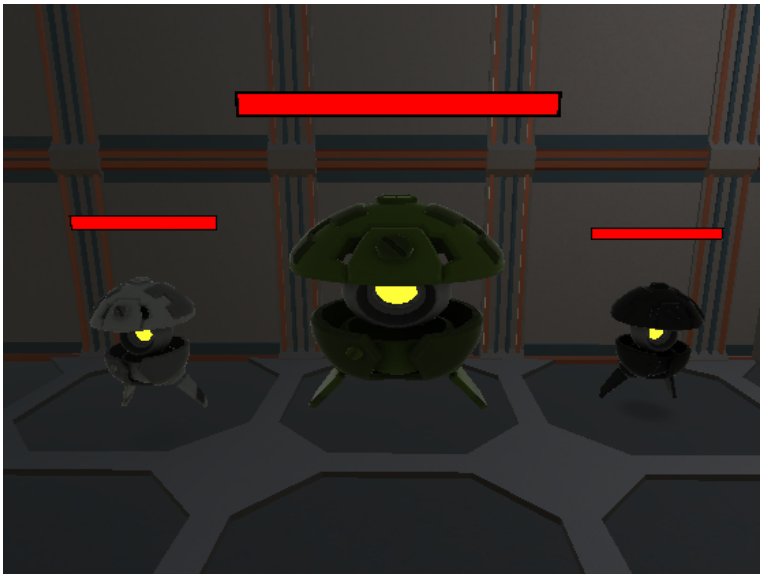


Figura 1: De izquierda a derecha: *melee*, tanque y *sniper*

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

**1.1.2.1. Melee** Flota a nivel de suelo y patrulla los pasillos. Su arma de campos electromagnéticos tiene un alcance de un par de metros, pero produce daño continuo. Se pueden distinguir fácilmente del resto de enemigos gracias a su apagado color gris, y las piezas que suelta al explotar resultan útiles para obtener armas y abrir compuertas.

**1.1.2.2. Tanque** Tras descubrir la vergonzosa falta de resistencia de sus robots *melee*, los ingenieros de *Closure* decidieron crear un robot más grande, con una armadura que le permite recibir cinco veces más daño hasta ser neutralizado, además de darle un reconocible color verde. Por suerte para el jugador, si conseguimos inutilizar a este enemigo podemos utilizar su sistema de regeneración para recuperar puntos de vida.

**1.1.2.3. Sniper** Como el más temible robot de los laboratorios *Closure*, el *sniper* cuenta con un arma láser en su ojo central que le permite atacar a una distancia mucho mayor que el resto de enemigos. Su camuflaje negro de élite lo hace invisible al radar de otros robots, pero no a la simple visión de nuestra protagonista.

### 1.1.3. Objetos

Figura 2: Modelo de una pieza

**1.1.3.1. Piezas** En concordancia con la estética cibernética del juego, su moneda no es otra que las piezas que pueden dejar caer algunos tipos de robots cuando los derrotamos. Con las suficientes piezas, podemos abrir puertas o fabricar armas en impresoras 3D especializadas. Un sistema monetario es un componente interesante en el diseño del nivel porque permite equilibrar y guiar la exploración de los mapas.

Figura 3: Modelo del *pickup* de salud

**1.1.3.2. *Pickup* de salud** Para recuperarse de los ataques de sus enemigos, a nuestra protagonista no le bastará con dejar que pase el tiempo. En su lugar, deberá considerar en sus estrategias qué enemigos pueden dejar caer este objeto que curará parte de sus heridas para poder seguir peleando.

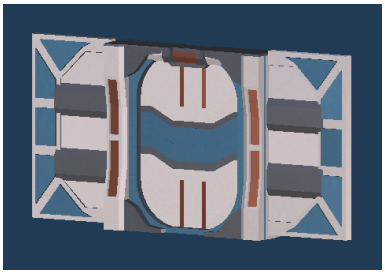


Figura 4: Modelo de la compuerta cerrada

**1.1.3.3. Compuertas de control de acceso** A lo largo, ancho y alto del mapa se encuentran distribuidas una serie de compuertas de seguridad que la protagonista deberá abrir para avanzar en el nivel. Estas puertas requieren un gasto de recursos, las piezas que se obtienen de los robot, para su apertura. Sin embargo, algunas compuertas tienen una funcionalidad extra para sorprender al jugador.

Figura 5: Modelo del *jetpack*

**1.1.3.4. *Jetpack*** En Marte la gravedad es menor, por lo que saltar por encima de los enemigos es más fácil. Pero llegará un punto en el que esto no sea suficiente para escapar de los laboratorios, y el usuario tendrá que equiparse con un *Jetpack* que cambia completamente la dinámica de movimiento del personaje.

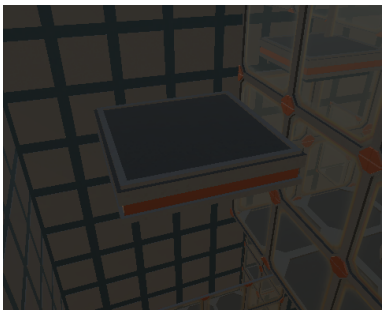


Figura 6: Plataforma móvil suspendida en el aire

**1.1.3.5. Plataformas móviles** Nuestra protagonista tendrá que utilizar su *jetpack* con agilidad y precisión para no caer al vacío en la última sección del nivel, donde en ocasiones tendrá que saltar rápidamente de plataforma en plataforma.

### 1.1.4. Armas

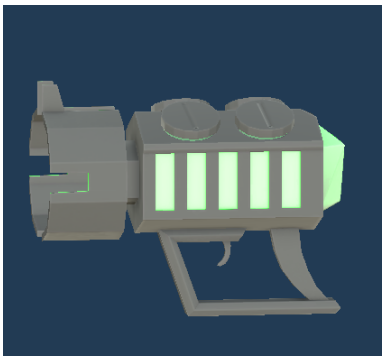


Figura 7: Modelo del *blaster*

**1.1.4.1. *Portal gun / Blaster*** El salto a la tercera dimensión parece haber neutralizado la capacidad de la *Portal Gun* de generar portales. Sin embargo, resulta muy útil su devastador efecto sobre los robots de *Closure*. Su reactor produce energía a un ritmo limitado, monitorizado en el HUD de realidad aumentada. Por lo tanto, es necesario esperar a recuperar energía tras una larga ráfaga de disparos.

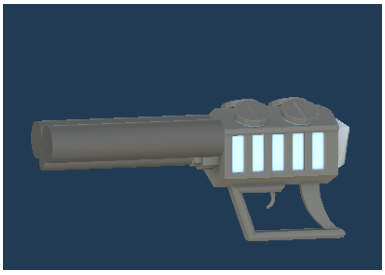


Figura 8: Modelo de la escopeta

**1.1.4.2. Escopeta** Si un rayo láser es eficaz, docenas de ellos a la vez lo serán más, ¿verdad? Este razonamiento llevará a nuestra protagonista a construir a través de las piezas de los robots un arma similar a una escopeta láser. Aunque la lluvia de rayos neutraliza los enemigos con facilidad, el reactor de la escopeta necesitará descansar para recuperar energía cada dos disparos.

Figura 9: Modelo del *Thunderfury*

**1.1.4.3. *Thunderfury*** Tras encontrar el prototipo de un cañón de partículas de portales nuestra protagonista descubrirá que el método más eficaz para reducir la marabunta de enemigos es construir un lanzagranadas láser. Si bien su barroco diseño necesita una gran cantidad de piezas, la capacidad de *Thunderfury* para *limpiar* una habitación es indudable ya que, si la cargamos lo suficiente, puede expulsar una ráfaga de devastadores proyectiles sobre grandes grupos de enemigos.

### 1.1.5. Diseño del juego

**1.1.5.1. Guión** La aventura comienza directamente en los laboratorios, en una zona de acceso restringido. Sin embargo, esta vez la protagonista ha sido detectada. Por lo tanto debe tratar de sobrevivir las consecutivas oleadas de robots de seguridad mientras explora los laboratorios buscando una vía de escape.

**1.1.5.2. Jugabilidad Base** La exploración de *Closure* se realiza principalmente abriendo las compuertas que separan las diferentes habitaciones. Su apertura requerirá de la creación de herramientas especializadas a cada cerradura y por lo tanto de piezas obtenidas de los robots. Si bien un estilo de juego consiste en limitar la exploración para emplear las piezas en la fabricación de armas más efectivas, la exploración se recompensará con la disponibilidad de nuevas armas y mejoras y la ventaja implícita de un mayor espacio para maniobrar.

La última sección del nivel es un segmento de *parkour* en el que aprovecharemos el *jetpack* para escapar de las instalaciones por un hueco en el tejado.

**1.1.5.3. Diseño de Niveles** El *level design* de Dimensions3D busca aprovechar al máximo la fusión de géneros que ofrece. De esta forma, el nivel puede ser dividido en tres áreas principales, en función del tipo de jugabilidad predominante en ellos:

- ***Central Rooms***:
  - ***Main Hall***: Éste es el nexo físico del nivel, donde el jugador comienza el juego. Con el objetivo de explotar al máximo la característica de *round-based surivival game*, el área contiene *spawners* de enemigos colocados de forma que el jugador precise siempre estar en movimiento, si no se quiere ver rodeado en una posición en la que no pueda escapar.

    Para dar dinamismo al *gameplay*, la decoración y el uso de múltiples niveles no es únicamente estético, si no que está organizado de forma que la navegación por este área sea menos monótona y el jugador deba pensar bien sus movimientos para no encontrarse en una situación desfavorable.
  - **Anexo**: Este área contigua al *Main Hall* tiene como principal función, además de ofrecer nuevas áreas que permiten una mayor maniobrabilidad por el nivel, proporcionar *foreshadowing* de cómo se puede desenvolver el resto del gameplay: al comenzar el nivel, es posible ver ya, a través de las cristaleras al lado izquierdo del *Main Hall*, y del balcón superior de la sala, que el jugador va a disponer de multitud de zonas explorables. Esto permite tanto aumentar el disfrute del juego, como indicar al jugador que efectivamente existen múltiples estrategias, en función de cuánto se priorice la *exploración* en vez de buscar obtener mejores armas desde un primer momento.
- ***Trap Room***: Éste área permite transicionar de forma *seamless* a un estilo *Arena Shooter*: el jugador se ve obligado a eliminar enemigos en un espacio mucho más reducido, lo que implica una velocidad de juego mayor y la necesidad de reaccionar rápidamente al movimiento de los robots. Para lograr este efecto, se ha utilizado una mecánica conocida como *"Monster House"* (en español *"Nido de Monstruos"*), utilizado en juegos *Roguelike*.

  De esta forma, la jugabilidad en este área queda dividida en dos fases:

- **Fase 1**: El jugador encuentra un área aparentemente sin enemigos, con el *Jetpack* claramente visible en el pequeño habitáculo interior que comunica por cuatro puentes al anillo exterior.

  Lógicamente, el jugador intentará obtener el jetpack; cuando avance por el puente hacia él, se activará la *traproom*. Todas las puertas de la sala son cerradas, y los *spawners* se activan, dejando encerrado al jugador en un área estrecha y de alto riesgo, pues requiere evitar en todo momento en los caminos sin salida los que se han convertido los puentes. El jugador sólo tiene una salida: abrir de nuevo las puertas que protegen el *jetpack*.
- **Fase 2**: Con el jetpack ya en su poder, el jugador puede utilizarlo para subir a las plataformas superiores de la *Trap Room*, y ya con algo más de margen de maniobra, continuar eliminando robots hasta poder abrir las puertas que dan al balcón del *Main Hall*, escapando del área.

Para evitar un salto abrupto en las mecánicas o el flujo del nivel, en éste área se aprecia de nuevo la presencia de múltiples estrategias implícitas: el diseño del nivel permite intuir que efectivamente la sala puede ser una *Monster House*, por lo que el jugador aún está a tiempo de volver al área principal y conseguir recursos de una forma menos frénetica, para después poder abrir las puertas y escapar inmediatamente; alternativamente, puede decidir arriesgarse e intentar terminar el nivel lo antes posible.

- **Parkour Room**: Como habitación o fase final del nivel y el juego, se encuentra una habitación que enfoca la jugabilidad hacia el género *3D Plataformer*, con el objetivo de aprovechar la maniobrabilidad en tres dimensiones que confiere el movimiento del jugador, en especial acompañado del *Power Up* del *jetpack*.

Con todo esto en mente, esta gigantesca habitación se centra lógicamente en la *verticalidad*. Un pequeño pasillo accesible por una puerta del *Main Hall* (creado deliberadamente para provocar un mayor contraste al entrar en esta útlima habitación) desemboca en una gran sala con multitud de retos lineales al estilo *Parkour*.

Estos retos de *3D Platforming* exigen la jugador hacer un buen uso del *jetpack*, en algunos casos requiriendo que piense bien como debe moverse antes de enfrentarse a ellos, pues cualquier pequeño fallo puede provocar que se precipite al vacío hasta volver al suelo de la sala. Debido a que los enemigos siguen fluyendo al interior de la habitación (aunque lógicamente no pueden perseguir al jugador), se consigue un efecto *lava floor*, que añade más tensión al que será el *endgame* del juego. Esto, añadido a la forma y la dirección verticales de la habitación, potencian la sensación de que efectivamente el juego llega a su clímax, hasta que el jugador logra escapar al exterior por el techo de la sala.

La dificultad de esta última habitación de *Parkour* va *in crescendo*. Esto se consigue "dinamizando"progresivamente los obstáculos, haciendo que dependan progresivamente más de la mecánica de plataformas móbiles implementada. Así existe un escalado gradual de dificultad: desde el primer obstáculo, que es completamente estático, hasta el zénit del nivel, donde no existe un suelo seguro donde el jugador pueda posarse y pensar calmadamente como debe utilizar el *jetpack*, requiriendo trazar un plan complejo (o reaccionar rápidamente al movimiento de las plataformas).

Una característica notable de esta habitación es el uso de pistas *implícitas* para guiar al jugador: las lámparas y la iluminación colocadas en las paredes y techos de la sala juegan un rol clave en esta fase, ya que indican de una forma sutil el siguiente lugar al que dirigirse, pero sin explicar el cómo hacerlo. Esta aproximación es cada vez más común en juegos que buscan cuidar el potencial narrativo del juego: al contrario que alternativas clásicas, que usan la GUI para mostrar *waypoints* explícitos, permite preservar la inmersividad narrativa del juego. A la vez, hace que el jugador se sienta gratificado de haber descubierto las "pistas"dejadas si las necesita, en vez de sentirse subestimado por haber sido guiado de forma rígida por el nivel.

**1.1.5.4. Reglas** La protagonista cuenta con un nivel de vida limitado y sin regeneración automática. El objetivo del juego es explorar los laboratorios hasta conseguir escapar, haciéndose paso a través de oleadas de enemigos.

Estos proporcionan a la protagonista las piezas necesarias para avanzar por el nivel así como *Pickups* que le permiten recuperar la salud perdida.

#### 1.1.5.5. Mecánicas

**Escapatoria y explotación del *pathfinding*:** Como es habitual en los juegos *round based survival*, hemos diseñado la interacción de forma que la supervivencia se base en la gestión del *pathfinding* de los enemigos: empleando pasillos y otros elementos que limitan el paso de los enemigos, el jugador puede obtener una posición ventajosa.

Por ejemplo, situarse en la zona superior del *hall* inicial supone una gran ventaja estratégica a la hora de usar el lanzagranadas sobre una multitud de enemigos. Además, la protagonista puede saltar la barandilla para escapar en un apuro.

***Jetpack*** Esta mecánica de saltar obstáculos que los enemigos no pueden atravesar se ve reforzada con la introducción del *jetpack*. Además de haber diseñado la última zona para su uso, el resto del mapa ha sido dispuesto de forma que el desbloqueo del *jetpack* introduzca mecánicas totalmente nuevas en zonas ya visitadas.

El ejemplo más claro lo supone el salto entre las zonas superiores del *hall* principal, donde tras el desbloqueo del *jetpack* no solo podemos evitar el daño por caída sino que podemos volar de una zona elevada a otra, ganando gran ventaja sobre los enemigos, que deben rodear todo el mapa.

**Mecánicas de *shooter*:** Como comentamos en la introducción, nuestro juego también se inspira en el género *arena shooter*. Esto se ve reflejado especialmente en el diseño de las armas, basadas en la recarga por *cooldown*. La mecánica de *vending* nos permite unir los dos géneros de manera fluida, de forma que se obtenga la progresión de los juegos tipo *round based* sin la necesidad de ahorrar munición, permitiendo un *gameplay* más *arcade*.

Además, este último punto se ve reforzado en el hecho de que disponemos de un sistema de cambio de armas bastante fluido que nos permite aprovechar potenciales sinergias entre varias armas alternando entre ellas cuando tenemos que dejar que se recarguen.

**Jugabilidad no lineal:** Como aspecto destacable general del *level design*, es importante mencionar el enfoque *no-linear* que se busca con el diseño de salas "modulares". El jugador no puede explorar todo el nivel desde el principio, si no que debe desbloquear el nivel progresivamente abriendo las distintas puertas. Cada nuevo área permite descubrir secretos o cambiar el enfoque a un género distinto sin resultar abrupto, aumentando la variedad existente en el *gameplay*. Añadido a esto, el jugador tiene la libertad de desbloquear las salas en el orden que prefiera, aunque lógicamente significará que el desarrollo y la velocidad de lo restante del juego será distinto en cada caso. Como forma de guiar de manera flexible al jugador, los costes de apertura de las puertas están deliberadamente seleccionados como pista al jugador de que el orden que proporciona la curva de dificultad más ideal es

```
Anexo -> TrapRoom -> Parkour Room
```

, pero dándole libertad de elegir el orden que desee. Además de mejorar la sensación de libertad en una *blind run*, potencia notablemente la re-jugabilidad, ya que los jugadores podrán buscar formas más rápidas de completar el juego, combinando distintos órdenes de desbloqueo de zonas y priorizando la exploración de forma distinta. Esto busca seguir la línea de los juegos no-lineales y de *mundo abierto*, que han demostrado con gran éxito la viabilidad de incorporar esta libertad en la jugabilidad y la narrativa del *gameplay*.

## 1.2. Aspectos destacables

### 1.2.1. Sistema de oleadas

El aspecto *Round Based Survival* de nuestro juego se ha implementado mediante la modificación del componente *CustomEnemyManager*. Este componente encapsula la funcionalidad de llevar cuenta de los enemigos con la generación de los mismos mediante oleadas programables.
Además de las propias oleadas, configurables en la cantidad, tipo y velocidad de aparición de los enemigos, se pueden configurar desde el inspector o desde otros componentes los puntos donde se desea que aparezcan los enemigos.

### 1.2.2. Sistema de pagos

El jugador tiene un componente *Wallet* que mantiene la cuenta de las piezas que tiene en cada momento. Para activar impresoras y puertas por

proximidad, cada una tiene un *script* de *Vending* y un *trigger* de manera que, cuando el jugador entra en su zona de influencia, el *script* se añade a sí mismo como alcanzable en la cartera del jugador, para que se generen los *prompts* oportunos y el jugador sepa que puede interactuar con esa estación. Lógicamente, cuando el jugador sale del *trigger* la referencia en la cartera se rompe y los *prompts* se restauran.

### 1.2.3. Concepto de habitación

Al abrir una puerta, el componente *VendingDoor* se comunica con el *RoomManager* de la habitación con la que conecta para que este abra todas sus puertas. De esta manera se facilita la navegación por el mapa para el personaje y para los enemigos.

### 1.2.4. Habitaciones con *spawners* sincronizados con las puertas

Una característica destacable del diseño de nuestros niveles es la progresión a través de diversas habitaciones, a las cuales se accede mediante el pago en las puertas. Para que esta característica se integre de forma correcta con el sistema de oleadas hemos sincronizado los componentes.
Cuando el jugador abre una puerta, o en general un grupo de puertas que permiten acceder a una nueva habitación, esta situación se comunica a *CustomEnemyManager* para activar el *spawning* de enemigos en la habitación correspondiente. Concretamente, la puerta notifica a su respectivo *RoomManager* para que se añadan los *spawners* a la lista de *spawners* activos.
La flexibilidad del sistema nos permitió también implementar la funcionalidad de habitación trampa, donde se cierran las puertas de entrada y el jugador debe encontrar otras por las que salir.

### 1.2.5. Habitaciones trampa

Para implementar el concepto de *Monster Nest*, tan común en los *Roguelike*, se especializa la clase *RoomManager* para que, cuando un *trigger* se lo comunique, cierre todas sus puertas y comunique al *CustomEnemyManager* que tiene que desactivar todos los *spawners* del mapa excepto los presentes en esta habitación trampa. El jugador queda atrapado, ya que los *triggers* de las puertas de entrada se colocaron para que solo se pudieran abrir desde fuera. Cuando encuentre una puerta que se pueda abrir desde dentro, esta

notificará al *TrapRoomManager* mediante un evento insertado en el inspector para que vuelva a abrir todas las puertas y restaure los puntos de *spawn*.

## 1.3. Manual de usuario

### 1.3.1. Dependencias

Los paquetes necesarios para ejecutar el juego son:

- FPS Microgame
- Sci-Fi Asset Pack
- Navmesh Components
- TextMesh Pro

### 1.3.2. Jugabilidad

El principal obstáculo a la hora de explorar los laboratorios *Closure* son los enemigos, especialmente aquellos con ataque a distancia. Para lidiar con ellos, el jugador dispone de la posibilidad de explotar el *pathfinding* y la puntería de los enemigos.

Si bien cada enemigo se mueve a una velocidad ligeramente diferente, es posible desplazarse de forma que se agrupen. De este modo se puede incluso conseguir que los enemigos con armas a distancia dañen a los demás. Además, si el jugador se mueve perpendicularmente a los enemigos, no recibirá disparos.

El otro gran reto consiste en evitar ser rodeado. Para ello, la mejor estrategia es emplear el diseño del mapa, planeando las rutas. El mapa está organizado de forma que los pasillos forman anillos, por lo que es sencillo seguir intuitivamente un camino que permita escapar.

Los pasillos pueden ampliarse mediante la apertura de puertas. Al igual que el desbloqueo de armas, la progresión en el nivel depende de un sistema monetario basado en las piezas que se obtienen al inutilizar a los enemigos.

Por otro lado, aunque no es suficiente por sí misma para resistir una oleada, la mecánica de alternar rápidamente las armas según su *cooldown* recompensa al jugador que sabe escoger cuándo emplear cada una con mayor potencia de fuego.

Finalmente el *jetpack* inclina la jugabilidad más hacia el lado *arena shooter* de nuestro juego, donde el uso inteligente del impulso permite evitar el daño por caída. En la zona final de *parkour*, además del camino principal, el jugador experto cuenta con trayectorias más complicadas, que permiten ahorrar tiempo y suponen una adición interesante para quienes disfrutan del *speedrunning*.

### 1.3.3. Controles

| | |
|---|---|
| W | Avanzar |
| S | Retroceder |
| A | Izquierda |
| D | Derecha |
| C | Agacharse/Levantarse |
| SHIFT | Correr |
| SPACEBAR | Saltar |
| TAB | Pausa |
| E | Interactuar |
| LMB | Disparar |
| LMB(mantener) | cargar lanzagranadas |
| RMB | Apuntar |
| Movimiento del ratón | Dirección de apuntado |
| Q / MOUSEWHEELUP | Arma anterior |
| E / MOUSEWHEELDOWN | Siguiente arma |
| 1,2,3 | Arma número n |

### 1.3.4. *HUD*

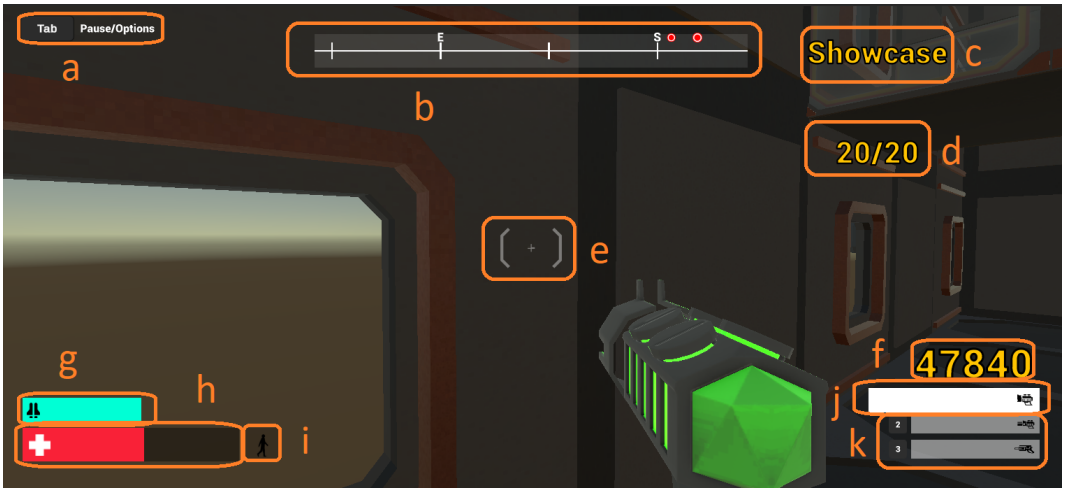



Figura 10: *Heads Up Display*

(a) Indicador de menú de pausa

(b) Brújula con indicador de enemigos a varias alturas

(c) Nombre de la ronda actual

(d) Número de enemigos que quedan por derrotar en esta ronda

(e) Mirilla del arma

(f) Número de piezas actual

(g) Barra de carga del *jetpack*

(h) Barra de salud

(i) Indicador de *standing*/*crouching*

(j) carga del arma actual

(k) lista de otras armas disponibles

### 1.3.5. Menús

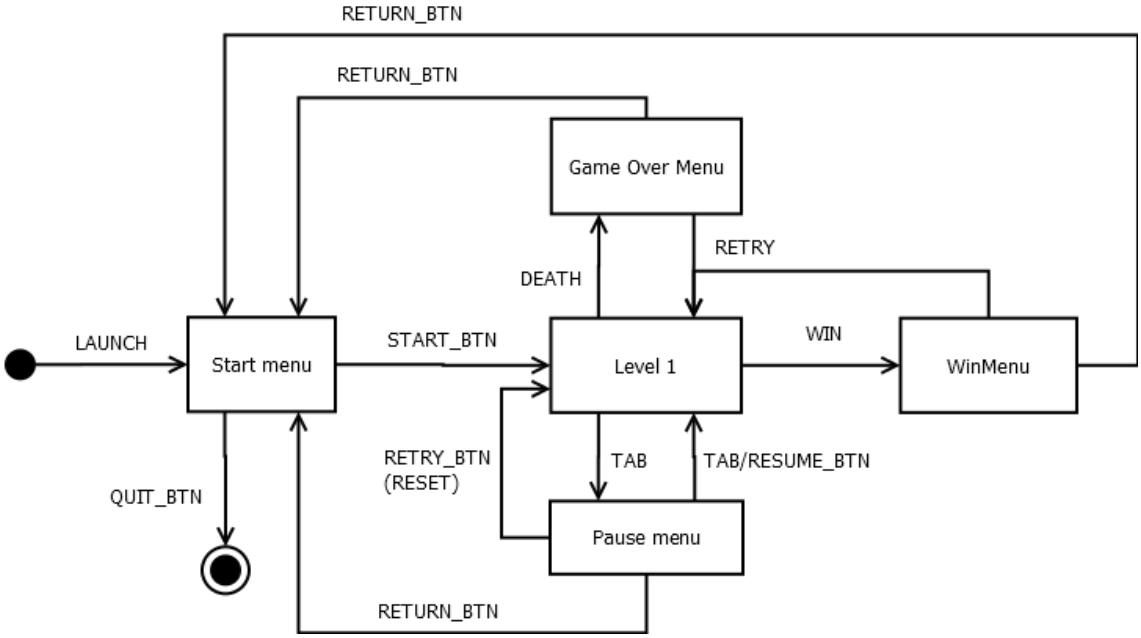



Figura 11: Transición entre escenas

En todo momento se puede salir del juego cerrando la ventana, El único menú que puede salir de la aplicación es el de inicio, al que todos los menús pueden acceder. Adicionalmente, los menús de muerte, pausa y victoria tienen botones de reintento.

## 1.4. Proceso de desarrollo

### 1.4.1. Coordinación dentro del equipo

En este proyecto se ha utilizado la metodología de desarrollo ágil *Kanban*. Concretamente, una tabla con estas cuatro 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.

### 1.4.2. Análisis

El proceso de análisis parte de dos ideas principales: tener claro el tipo de juego que se quiere implementar y aprovechar lo máximo posible los *assets* disponibles en la tienda de Unity. La base de la jugabilidad de *Dimensions* es la de un *First Person Shooter*, que requiere la laboriosa implementación de varios componentes característicos como el apuntado, el daño por proyectiles y la inteligencia artificial de los enemigos. Para poder centrarnos en los componentes más específicos de nuestro juego, utilizamos el paquete *FPS Microgame* de *Unity Learn*.

Los *scripts* de este paquete están implementados para permitir *customización* dinámica de *prefabs* mediante el uso del inspector de Unity. Gracias a esta característica las siguientes *features*, entre otras, no tuvieron que ser reimplementadas:

1. Gestión de audio (AudioUtility)
2. Gestión de entrada (PlayerInputHandler)
3. Navegación de menús (MenuNavigation)
4. Transición a los menús de muerte y victoria (GameFlowManager)
5. Gran parte de la IA de los enemigos (EnemyController y EnemyMobile)
6. Control del personaje: salud, movimiento, vuelo, manejo de armas (Health, PlayerWeaponsManager, Jetpack, PlayerCharacterController, PlayerInputHandler)

No obstante, algunos componentes no son lo suficientemente genéricos como para acomodar la jugabilidad que tenemos en mente. En estos casos, optamos por incluir en el *namespace* de la librería FPS una versión *Custom* que respeta la funcionalidad del código original y añade los comportamientos específicos que necesitamos.

### 1.4.3. Diseño Software

A lo largo de todo el diseño hemos seguido una serie de principios que guían el desarrollo. Además de las buenas prácticas establecidas en la arquitectura al emplear un ejemplo proporcionado por los propios desarrolladores de *Unity*, hemos prestado atención a cómo nuestras decisiones de diseño influencian el rendimiento. Por ejemplo, desde la interfaz de usuario hasta la comunicación de las puertas, hemos preferido sistemas basados en *observers*, *triggers*, eventos, etc. frente a la espera activa con *raycasts* o cálculo de distancias.

Como comentamos, nuestra arquitectura parte de la establecida en *FPS Microgame*. Los *GameObjects* principales de esta arquitectura son *Player* y *GameManager*, que adaptaremos a sus versiones *custom*.

Mientras que el primero se encarga de la cámara, armas, locomoción, etc. el segundo se encarga de la gestión general del juego, como el audio o el HUD y, crucialmente, de la gestión de los enemigos. Hemos modificado especialmente el componente *EnemyManager*, para implementar la funcionalidad específica de nuestro juego, desde la generación de enemigos en sus *spawners* hasta mostrar por pantalla el número de enemigos restantes en la oleada actual o el precio a pagar en una *vending machine*.

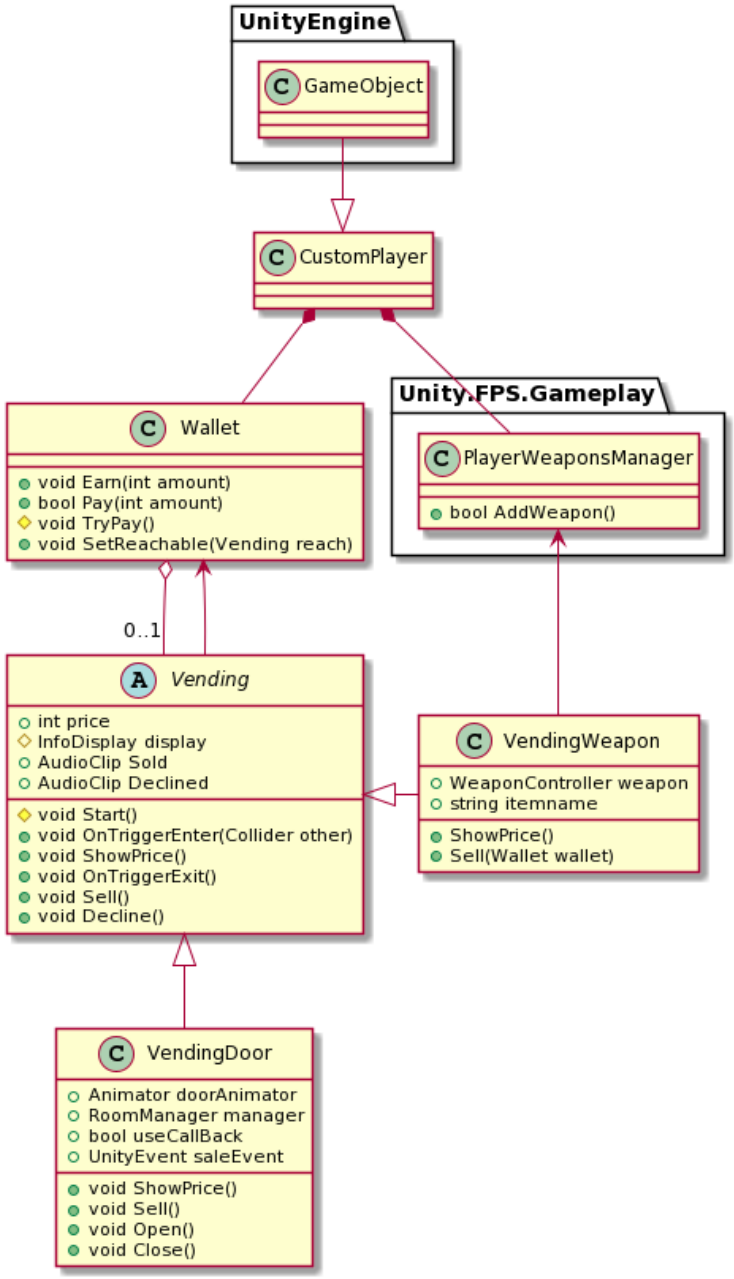



Figura 12: Diagrama de clase de *Vending*

Las *weapon vending machines* son componentes totalmente independientes de *FPS Microgame*, implementados expresamente para este videojuego. Por similitud en su interacción, estas máquinas comparten la mayoría de su código con las *vending doors*, las compuertas que regulan la progresión en el nivel. Concretamente, esta similitud se ve reflejada en que ambas clases heredan de *vending*, la abstracción de la funcionalidad común.

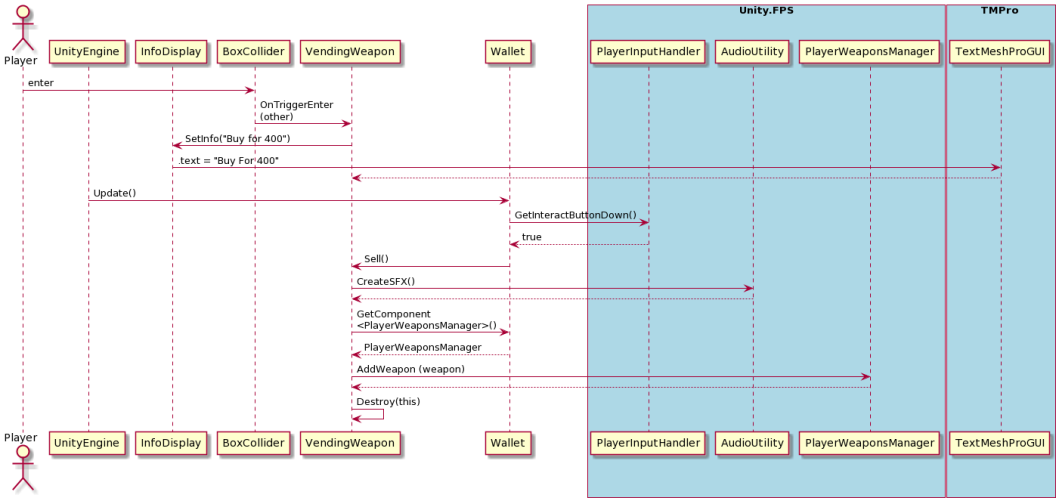



Figura 13: Diagrama de secuencia de compra de un arma

Sin embargo, la mayor diferencia a nivel de diseño software de las dos es la interacción que las puertas presentan con *RoomManager* y su especialización *TrapRoomManager*. Estos dos componentes son los encargados de comunicar las puertas entre sí y con *EnemyManager*.

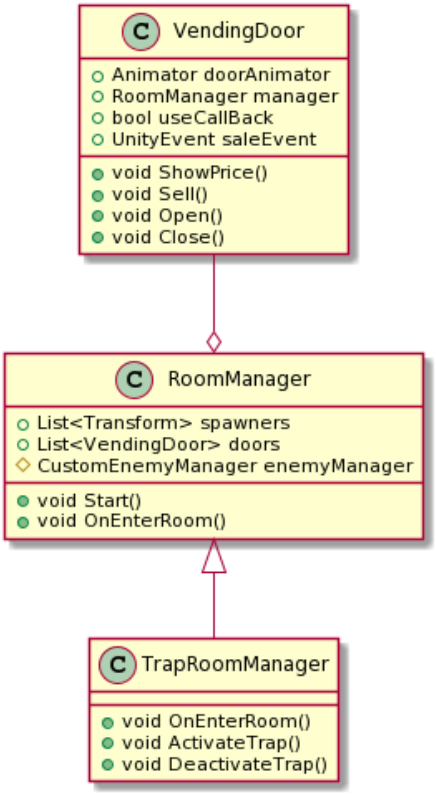



Figura 14: Diagrama de clase de *RoomManager*

Esencialmente, un *RoomManager* recibe la notificación de una puerta de que se debe de abrir la habitación, incluyendo todas sus puertas. Esta apertura también activa los *spawners* de la habitación. *EnemyManager* dispone de una lista de *spawners* activos, que *RoomManager* modificará. Esta arquitectura nos permite un control fino, a nivel de cada *spawner*, que brinda flexibilidad a la hora de diseñar el nivel.

Por ejemplo, *TrapRoomManager* desactiva todos los *spawners* situados fuera de la habitación durante su activación, que consiste en encerrar al jugador de manera que solo algunas puertas permitan salir. Resultaría sencillo extender la clase para activar *spawners* determinados, pudiendo incluso rotarlos, gracias al control proporcionado por la interfaz.

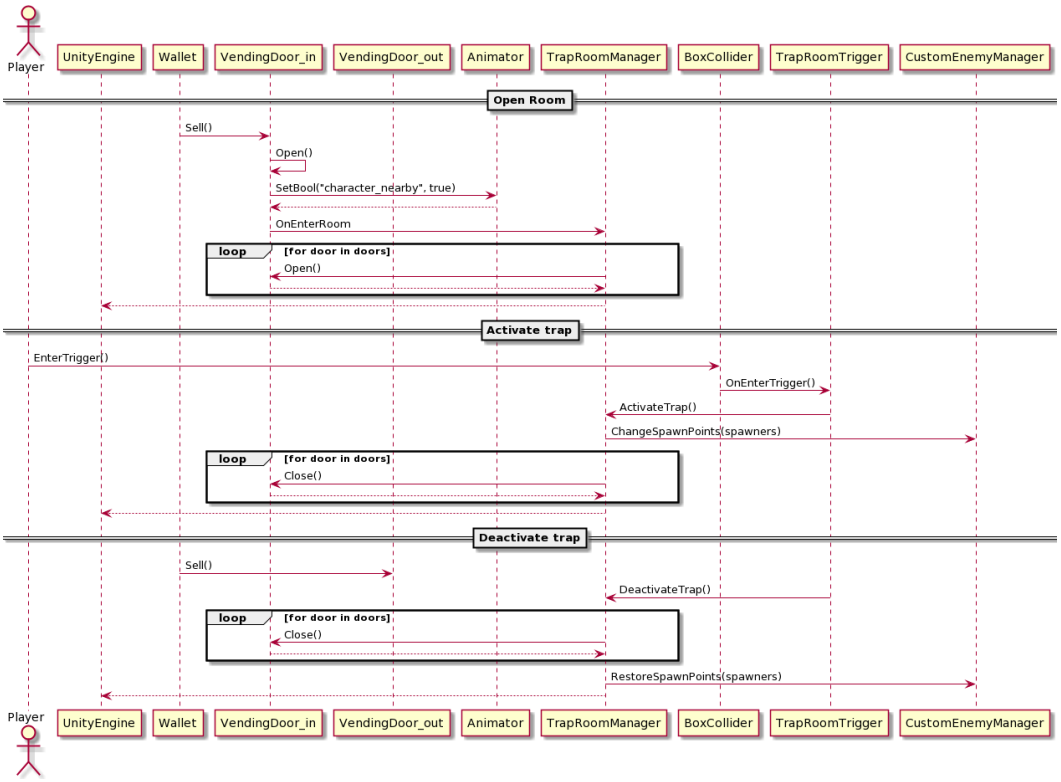



Figura 15: Diagrama de secuencia de la habitación trampa

Por último, *EnemyManager* emplea también un sistema de listas para la ejecución de las oleadas. Estas listas permiten establecer oleadas con un número, tipo y velocidad de aparición de los enemigos determinados. Se permite a los diseñadores de nivel refinar a mano las oleadas, eligiendo cuidadosamente cuántos enemigos y en qué orden deben aparecer, todo ello sin abandonar la comodidad del editor y su sistema de listas.

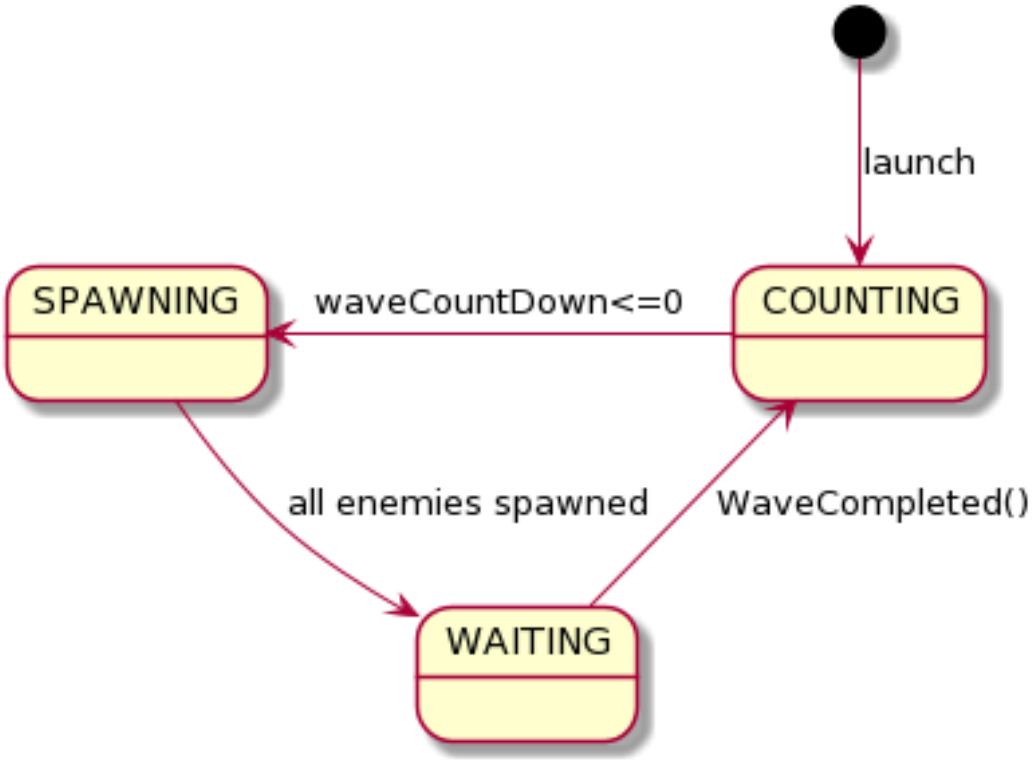



Figura 16: Diagrama de estados del gestor de oleadas del *CustomEnemyManager*

Esta elección además no limita la posibilidad de generar automáticamente las oleadas. Si bien hemos elegido diseñar a mano las mismas para garantizar una experiencia concreta, nuestro diseño hace sencillo generar la lista de la oleada aleatoriamente. Esta aproximación incluso presenta ventajas frente a la generación aleatoria *just in time*, ya que permite acotar la "mala suerte"que se puede tener en una ronda o randomizar partes de la misma manteniendo otros parámetros constantes.

Un último detalle a mencionar sobre las oleadas es que, para que no sea necesario planear decenas de ellas manualmente, el *manager* recorre la lista de oleadas hasta su último elemento y, cuando llega a este, lo repite indefinidamente. De esta manera se da soporte a partidas potencialmente infinitas, aunque esto requiera calibrar con cuidado la dificultad para que no sienta que la dificultad es inadecuada o injusta.

**1.4.3.1. Patrones de diseño** Los principios de diseño que mencionábamos anteriormente se ven reflejados en los patrones concretos que siguen los componentes. Podemos clasificar los patrones en aquellos que imponen tanto el propio motor como el paquete *FPS Microgame* y aquellos que hemos empleado para organizar el diseño de las *features* propias de nuestro juego.

La arquitectura por componentes de *Unity* implica un uso masivo del patrón decorador, ya que nos permite añadir componentes de forma dinámica a cada uno de los *GameObjects* de una escena. De manera similar, el uso de *prefabs* en nuestra práctica supone también que estamos empleando un patrón plantilla, que permite una reutilización intensiva de recursos a un coste mínimo.

En el caso de *FPS Microgame*, *GameManager* funciona como un componente que sigue el patrón Singleton: una sola instancia puede controlar una escena entera, si bien es cierto que la arquitectura no nos impide instanciar varios.

En nuestra implementación, la naturaleza cohesiva de la interacción con los elementos del nivel nos llevó a implementar en varias ocasiones el patrón observador. Como se puede ver en los diagramas 12 y 14, *RoomManager* es observador de *VendingDoor*, *PlayerWeaponsManager* es observador de *Vending* y *Wallet* es observador de *Vending*.

observer VendingDoor es subject de RoomManager, que es observer

Wallet es observer de Vending

plantilla decorador - cada uno de los componentes

### 1.4.4. Detalles de implementación

- En las situaciones que involucran acciones periódicas (la aparición de enemigos por oleadas y la espera antes del cambio de dirección de las plataformas móviles) se utilizan corrutinas para manejar las esperas de manera eficiente.

- Para que todos los enemigos empiecen a perseguir a la protagonista en cuanto *spawneen* y no la pierdan nunca de vista, se modificó el componente *DetectionModule* en el *script CustomDetectionModule*.

- El daño por contacto de los enemigos *melee* se implementó mediante un *trigger*. El método *OnTriggerStay* se ejecuta cada *FixedUpdate*, por lo

que es robusto a cambios en la tasa de refresco y el candidato perfecto para implementar un sistema de daño continuo por contacto.

- Por defecto, cada puerta del juego llaman al gestor de su habitación para que provoque los efectos secundarios de abrir esa puerta en concreto. Para dar todavía más versatilidad a la hora de diseñar el nivel, el componente *VendingDoor* tiene un evento vacío al que se le pueden agregar métodos de cualquier *script* de la escena. Esta característica nos permite crear desde el inspector las puertas de salida de las habitaciones trampa, que tienen que desactivar la trampa además de abrir todas las puertas.

- Para mostrar texto en el HUD de manera sencilla y eficiente se utilizan los componentes de texto del paquete *TextMeshPro*. Como queremos actualizar la información *on demand*, un componente intermediario *InfoDisplay* se encarga de pasarle la información actualizada al componente *TextMeshPro*. Si un componente maneja información que queremos tener en el HUD, necesita una referencia al *InfoDisplay* concreto que actualiza el componente *TextMeshPro* correspondiente.

- Para que los enemigos *melee* no disparen, su comportamiento se implementa en una subclase de *CustomEnemyController* que sobreescribe su método *TryAttack* para que no haga nada.

- Cuando el personaje está pisando una plataforma móvil, sus movimientos tienen que ser relativos a la plataforma. Para ello, la plataforma se convierte en el padre del personaje mientras este último se encuentre dentro de un *trigger* situado a unos centímetros por encima de la plataforma.

- El movimiento de las plataformas móviles está sincronizado por tiempo y también es robusto a cambios en la tasa de refresco.

- Para abrir y cerrar las puertas del nivel, el componente *VendingDoor* le indica al animador del *prefab* que hay un personaje cerca.

## 1.5. Informe de *bugs*

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

- La mirilla de las armas aumenta y disminuye de tamaño erráticamente en ciertas situaciones. Este error es difícil de reproducir
- Los muros del nivel son polígonos muy finos, por lo que hay situaciones en las que los enemigos pueden *clippear* visualmente e incluso el jugador puede atravesar paredes.
- Al volver a entrar en una escena, activar el menú de pausa requiere dos *taps* del botón de pausa. El primero para el tiempo en el juego y nos permite disparar y mirar a nuestro alrededor, pero por algún motivo el *canvas* del *InGameMenu* no se activa hasta el segundo *tap*. Una vez que hayamos pulsado el mismo botón dos veces, el error no se repetirá hasta que reintentemos el nivel.
- Al activar el menú de pausa mientras se reproducen ciertos sonidos (como el del *cooldown* de las armas), el sonido se sigue reproduciendo en bucle.
- Al subirse en una plataforma móvil, el *parenting* del componente *Transform* del personaje al de la plataforma hace que se reproduzca la animación de caminar aunque estemos parados. Además, el *parenting* también provoca que podamos atravesar paredes si estamos quietos encima de una plataforma.
- Al activar una habitacion trampa, los *spawners* de las otras habitaciones se desactivan, pero como la *NavMesh* de los enemigos no tiene en cuenta las puertas, los que ya están vivos pueden entrar y salir de la habitación aunque nosotros no.

# 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.

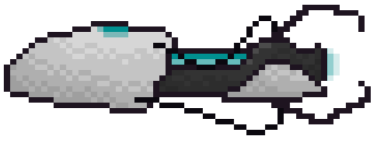
Tras descubrir grandes partes del código que ejecuta nuestro universo, los científicos de tu equipo descubren una combinación de estados que produce que la información se replique de manera no local. En idioma humano: teletransporte.

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.

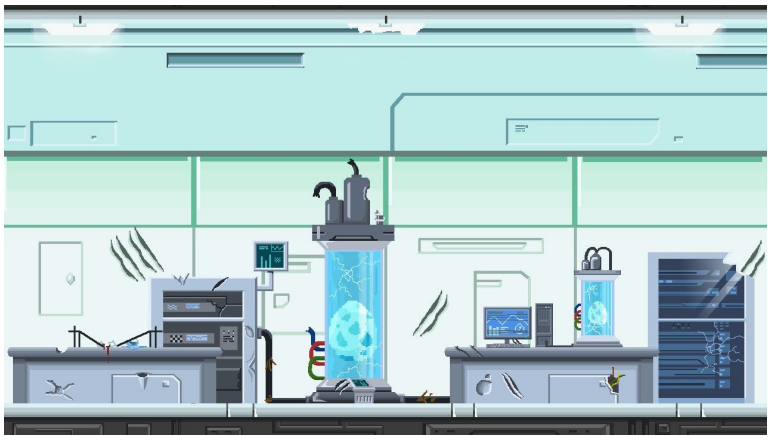
# 2. Otras características de la ambientación, y elementos que aparecerán en el juego

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.

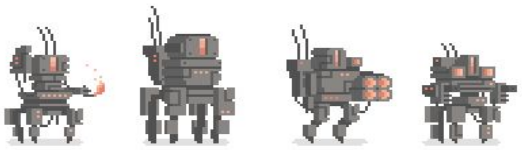
c. Personajes:

i. **Protagonista**:

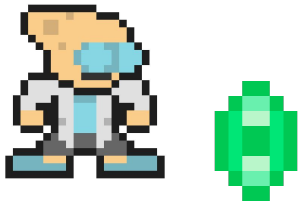
ii. **Enemigos:**

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.