UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Implementación de un detector de ritmos musicales para un videojuego

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Peral Tamarit, Pablo José

Tutor/a: Abad Cerdá, Francisco José

CURSO ACADÉMICO: 2023/2024

Resumen

Partiendo de un juego de ritmo, llamado Unwanted y desarrollado para la asignatura de Desarrollo de Videojuegos 3D, el objetivo de este proyecto es implementar un sistema de detección automática de ritmo para una canción cualquiera e introducir dicha canción en el juego. Actualmente el juego cuenta con una única canción cuyo ritmo fue extraído usando Audacity y corregido a mano. Este proyecto utilizará una librería específica para la detección de ritmo para cualquier canción. Se asume una determinada pérdida de precisión, al no poder ajustar a mano el ritmo detectado, a cambio de ganar en usabilidad al dejarle al usuario decidir con qué canción jugar. Para seleccionar la librería de detección de ritmos se hará una comparativa entre varias librerías y plugins de Audacity. Se analizarán varias canciones de géneros distintos para intentar encontrar la librería que ofrezca los mejores resultados. También habrá que estudiar la forma de integrar la librería seleccionada en un proyecto de Unity. El primer paso del proyecto es establecer un valor de referencia para la detección de ritmo usando Audacity como base para las comparativas. Audacity no cuenta de forma nativa con una función de detección de ritmos, pero sí con plugins capaces de dar dicho servicio. Una vez estudiadas las opciones, se procedería a implementar la detección de ritmo para así encontrar la librería que mejor se adapte a los requisitos del proyecto, tanto de comodidad de uso como especialmente de precisión en la detección. El siguiente paso sería adaptar Unwanted para que el usuario pueda, de manera cómoda e intuitiva, elegir la canción para jugar.

Palabras clave: videojuego, ritmo, detección de ritmo, Unity, Audacity, Essentia.

Abstract

Starting from a rhythm game called Unwanted, developed for the subject of 3D Video Game Development, the goal of this project is to implement an automatic rhythm detection system for any song and introduce that song into the game. Currently, the game has a single song whose rhythm was extracted using Audacity and manually corrected. This project will use a specific library for rhythm detection for- pondria of any song. It is assumed that there will be a certain loss of accuracy, as it will not be possible to manually adjust the detected rhythm, in exchange for gaining usability by allowing the user to decide which song to play with. To select the rhythm detection library, a comparison will be made between several libraries and Audacity plugins. Various songs from different genres will be analyzed to find the library that offers the best results. The way to integrate the selected library into a Unity project will also need to be studied. The first step of the project is to establish a reference value for rhythm detection using Audacity as a basis for comparisons. Audacity does not natively have a rhythm detection function, but it does have plugins capable of providing this service. Once the options are studied, the rhythm detection will be implemented to find the library that best meets the project's requirements, both in terms of ease of use and especially in terms of detection accuracy. The next step would be to adapt Unwanted so that the user can, in a comfortable and intuitive way, choose the song to play with.

Keywords : videogame, rythm, rythm detection, Unity, Audacity, Essentia

Tabla de contenidos

Índice general

Índice de figuras

1 Introducción

Este trabajo se ha desarrollado como una ampliación de un proyecto existente: ,un juego desarrollado en grupo para la asignatura optativa Desarrollo de Videojuegos 3D del Grado en Ingeniería Informática. La base es un videojuego del género *first person shooter* (un juego de disparos en primera persona), que se combina con elementos de ritmo. Este juego ha sido desarrollado para Windows utilizando Unity como motor para el juego y C# como lenguaje de programación.

El estado inicial del proyecto es un juego funcional que cuenta con un primer nivel prácticamente terminado y las mecánicas básicas de juego completamente implementadas. El juego consiste en avanzar por un mapa basado en caminos estrechos a lo largo de un bosque hasta llegar a zonas ligeramente más abiertas que representan las arenas de combate. En estas arenas podremos encontrar una serie de oleadas de enemigos. Para poder combatir con dichos enemigos, el jugador puede realizar una serie de acciones. La particularidad de este juego es que el jugador debe realizarlas al ritmo de la música del juego. Este ritmo se encuentra marcado en el centro de la pantalla con unas barras que se acercan a unos marcadores. El juego cuenta con una única canción cuyo ritmo ha sido extraído y retocado a mano para mayor precisión. En la Figura 1 podemos observar una captura de juego en la que se muestran las barras laterales que recorren la pantalla hacia las barras fijas, mas grandes, en el centro de la pantalla.

Figura 1: Captura de juego de Unwanted

Partiendo de esa base, el objetivo de este proyecto es ampliar el juego con un sistema que permita al jugador seleccionar una canción cualquiera dentro de su ordenador para incorporarla a la lista de canciones disponibles. El jugador será capaz de seleccionar cualquier canción anteriormente procesada para jugar El juego marcará el ritmo de la canción seleccionada en tiempo de ejecución, de la forma más precisa posible. Dicho sistema es transparente para el usuario, lo que permite jugar con cualquier canción sin necesidad de modificar la detección de ritmo a mano, como sí se hizo con la versión original.

Para comprobar la precisión del sistema de detección de ritmo implementado se ha comparado el resultado obtenido con distintos plugins de Audacity [11] que implementan detectores de ritmo. Las comparaciones se han realizado con 15 canciones de géneros distintos con el objetivo de comprobar la precisión de los sistemas de detección ante distintos tipos de canciones. De la misma forma, se ha publicado el juego y se ha hecho una encuesta para recoger la opinión de los usuarios.

1.1 Motivación

El desarrollo de Unwanted, el juego que sirve como base para este proyecto, ha sido uno de los trabajos que más ilusión y desarrollo personal me han aportado en la carrera. Siento que aprendí mucho durante el proyecto, pues nos enfrentamos a retos difíciles de diversa índole.

Uno de los retos más importantes, que a su vez era una pieza central del juego, era el sistema de ritmo. A lo largo del proyecto diseñamos 2 versiones completamente diferentes de este sistema. El principal problema de la primera versión de este sistema es que estaba pensado específicamente para la canción elegida para el juego y no era especialmente preciso. La segunda versión estaba más abierta para ser usada con otras canciones, pero para ello era necesario procesar, con anterioridad y con una herramienta externa, la canción elegida, retocando a mano el resultado del procesamiento de dicha canción.

El siguiente paso, que no fuimos capaces de alcanzar, es el de implementar un sistema que permitiera realizar el procesado de las canciones en tiempo de ejecución, permitiendo al usuario libertad total para jugar al juego con la canción que quisiera. El resultado de esto es un componente de ritmo llevado a su máxima expresión a nivel de funcionalidad y que a su vez permite desacoplar este componente del proyecto en cuestión para ser reutilizado en cualquier otro desarrollo que requiera de un sistema de ritmo, se trate de la aplicación que sea.

Debido a la complejidad del sistema y a la falta de tiempo de desarrollo proveniente de los plazos establecidos para la asignatura para la que fue diseñado, no se pudo implementar en el juego original esta última versión del sistema de ritmo. La principal diferencia que encontramos radica en el hecho de que la última versión implementada necesitaba de un análisis manual de cualquier canción que quisiera ser utilizada en el juego, mientras que la versión implementada para este TFG realiza ese análisis de manera automática y en tiempo de ejecución, otorgando al jugador una mayor libertad y sirviendo como característica diferencial que vuelve a Unwanted un juego único entre sus competidores.

1.2 Objetivos

Los objetivos principales de este Trabajo Fin de Grado son:

- Implementar un detector de ritmo en tiempo de ejecución para cualquier canción
- Permitir al usuario utilizar cualquier canción en un juego de ritmo
- Permitir al usuario jugar con canciones previamente analizadas en el juego.
- Desarrollar un componente de detección de ritmo fácilmente reutilizable en cualquier otro videojuego en Unity.
- Aprender sobre técnicas de detección de ritmo en canciones.
- Practicar en el uso de Unity y C#
- Practicar en el uso de metodologías ágiles

1.3 Colaboraciones

1.3.1 Estado inicial

Unwanted originalmente surge como un proyecto para la asignatura de Desarrollo de videojuegos en 3D en la que colaboramos cuatro programadores de Ingeniería Informática en la ETSINF junto a dos diseñadores de Diseño y Tecnologías Creativas de la Facultad de Bellas Artes. Este es el reparto original de tareas para el proyecto base.

Del bloque de programación, las tareas se han repartido de la siguiente manera:

- **Pablo José Peral Tamarit**
    - Sistema de ritmo inicial y detección de ritmo en las acciones necesarias
    - Lógica del sistema de combo
    - Movimiento de los enemigos (junto a Pablo y Jefferson)
    - Vida de los enemigos
    - Lógica de animaciones en el jugador y armas (junto a Pablo)
    - Lógica de la pistola (junto a Pablo)
    - Lógica de la katana (junto a Pablo)
    - Movimiento del personaje (junto a Pablo)
    - Gestión del Trello (junto a Rubén)
- **Jose Francisco García Sánchez**
    - Implementación del HUD (Junto a Rubén (diseños))
    - Implementación de animaciones del HUD (Junto a Rubén

(animaciones))

- Sistema de barras a ritmo
- Menú de pausa
- Menú de GameOver
- Arreglos en el sistema de ritmo
- Arreglos de problemas en los enemigos

- **Jefferson Paul Caiza Jami**
  - Sistema de ritmo mejorado
  - Ataque de enemigos (Junto a Pablo Granell)
  - Implementación menú principal
  - Implementación pantalla de carga
  - Créditos
  - Implementación cinemática inicial
  - Optimización (Junto a Pablo Granell)
  - Sistema de agua
  - Arreglo de los árboles
  - Compilación del juego
- **Pablo Granell Robles**
  - Movimiento de los enemigos (junto a Jefferson)
  - Terreno (junto a María y Rubén)
  - Implementacion animaciones enemigos
  - Implementación de enemigos, jugador y otros elementos de modelado 3D
  - Implementación animaciones jugador (junto a Pablo José)
  - Movimiento jugador (ayuda de una plantilla)
  - Implementación Tutorial
  - Feedback de los enemigos
  - Sistema de partículas + movimiento por un path (junto a María y Rubén)
  - Colliders (junto a María y Ruben)
  - Control de versiones + LFS (junto a Jefferson)
  - Implementación de sonidos (junto a Jefferson)
  - Mezclador de sonidos (junto a Jefferson)
  - Implementación de arenas (junto a María y Rubén)
  - Implementada katana + arreglarla basada en el feedback.

- Diseño de nivel (junto a María y Rubén)
- Diseño de iluminación (junto a María y Rubén)

Del bloque de artes, las tareas se han repartido de la siguiente manera:

- **María Manzanaro Alfaro**
  - Diseño de todos los enemigos del juego
  - Diseño de los personajes principales (junto a Rubén)
  - Realizado de todos los modelos 3D de enemigos excepto Boss
  - Realizado de todas las animaciones de jugador y enemigos
  - Diseño de escenario (junto a Rubén)
  - Terreno (junto a Pablo y Rubén)
  - Sistema de partículas (junto a Pablo y Rubén)
  - Movimiento por un path (junto a Pablo)
  - Diseño y edición de todos los sonidos y música
  - Organización y dirección general del proyecto
  - Diseño de nivel (junto a Pablo y Rubén)
  - Diseño de iluminación (junto a Pablo y Rubén)
  - Ideación del proyecto, guión e historia
  - Modelado de otros elementos del juego (junto a Rubén)
  - Diseño de vegetación (junto a Rubén)
  - Prototipado en blocking del nivel
  - Diseño de cinemática inicial
  - Grabación y edición de voz y sonidos cinemática inicial
  - Texturizado de enemigos y escenarios (junto a Rubén)
  - Implementación de arenas (junto a Pablo y Rubén)
  - Realización del video de feria de proyectos y video trailer
  - Creación de pegatinas promocionales para la feria (junto a Rubén)
  - Producción masiva de galletas Unwanted
  - Redacción mayoritaria del GDD
- **Rubén Rosaleny Benaches**
  - Diseño de los personajes principales (junto a María)
  - Diseño del HUD
  - Animación del HUD (Barras de vida, balas, sistema de combo)
  - Elección de fuentes tipográficas
  - Diseño de la pantalla de carga

- Animación de la pantalla de carga
- Diseño de la pantalla de inicio
- Animación de la pantalla de inicio
- Realizado de cinemática de introducción
- Realizado del modelo 3D del personaje principal
- Realizado del modelo 3D del Boss
- Realizado del modelo 3D de las raíces que bloquean los caminos
- Diseño de vegetación (Junto a María)
- Creación de árboles con la herramienta TREE de Unity
- Modelado de otros elementos del juego (junto a María)
- Realizado de animación de elementos 3D auxiliares (raíces)
- Diseño de escenario (junto María)
- Sistema de partículas (junto a Pablo y María)
- Diseño de nivel (junto a Pablo y María)
- Diseño de iluminación (junto a Pablo y María)
- Texturizado de enemigos y escenarios (junto a María)
- Implementación de arenas (junto a Pablo y María)
- Terreno (junto a María y Pablo)
- Colliders (junto a María y Pablo)
- Diseño de cajas de texto
- Diseño del logo de Unwanted
- Diseño del cartel de la feria de proyectos
- Creación de pegatinas promocionales para la feria (junto a María)
- Gestión del Trello (junto a Pablo José)
- Maquetación final GDD

1.3.2 División de tareas para el TFG

Partiendo de la división anteriormente planteada, se han desarrollado dos TFG en paralelo con temáticas distintas pero sobre la misma base. Las tareas específicas de cada desarrollo han sido realizadas por su respectivo autor mientras que otras tareas generales se han realizado en colaboración. Las tareas realizadas por cada autor han sido desarrolladas en su respectiva memoria:

- **Pablo Jose Peral Tamarit:** Implementación de un detector de ritmos para un videojuego
- **Jose Francisco García Sánchez:** Reingeniería de “Unwanted”: Mejora de un videojuego en Unity

De forma colaborativa se han desarrollado los siguientes apartados:

- Organización del proyecto mediante Trello
- Sprints y revisión de estos de forma conjunta
- Fusión de las ramas paralelas de trabajo
- Nueva versión del sistema de ritmo
- Página de Itch.io del juego
- Encuesta sobre el juego
- Sistema de encuesta en juego con estadísticas automáticas
- Arreglo de bugs previos al lanzamiento del juego
- Parche con corrección de bugs posterior al lanzamiento del juego

Junto a todo esto, este trabajo en concreto ha contado con la colaboración de Antonio Navas Cercós, un allegado personal con estudios de música que ha ayudado en el apartado 5.1 Evaluación interna a determinar cuáles de los distintos algoritmos de detección de ritmo de Audacity han obtenido las etiquetas de tiempo más precisas y a comparar estas con las obtenidas mediante el sistema de detección de Essentia.

2 Estado del arte

La música es uno de los componentes más importantes de un videojuego. Una buena dirección de sonido puede marcar mucho a un videojuego, tanto para bien como para mal. De la misma manera que ocurre en otras artes similares como el cine, los efectos sonoros son utilizados para transmitir emociones al jugador. Muchos de los combates más impresionantes o las escenas de terror más tensas que podemos recordar nos han impactado tanto gracias, en parte, a una buena banda sonora. Es imposible pensar en *Star Wars* sin recordar a *Darth Vader* y la Marcha imperial y así tantos otros ejemplos. Puede que las limitaciones técnicas del medio no permitieran que, en sus orígenes, muchos de los grandes títulos pudieran disponer de la variedad de sonidos a reproducir que más de un desarrollador le gustaría, pero eso ha hecho que las bandas sonoras de muchas de las principales sagas de la industria sean increíblemente memorables.

El videojuego como medio y como expresión artística tiende a reinventarse constantemente, buscando siempre nuevas propuestas con las que atraer a los jugadores y sorprender con nuevas formas de entender el medio. Muchos juegos tratan de utilizar el sonido de forma diferente. Es impactante para el jugador ver cómo cambia la forma en la que interactúa con un elemento tan básico de una forma que nadie antes había intentado y que se sale de lo común. De esta forma nacen los juegos de ritmo, que buscan convertir la música en la pieza central de la experiencia.

La vertiente más clásica dentro de los juegos de ritmo pasa por reproducir una canción y hacer que el jugador siga el ritmo. Probablemente el inicio del género de los juegos de ritmo, o como mínimo su popularización, se la podemos atribuir a Konami con su Dance Dance Revolution [1], que salió para máquinas *arcade* en 1998 (Figura 2). En dicho juego hay que pisar una serie de casillas marcadas en el suelo al ritmo de la música, simulando que se está bailando a su ritmo. Este juego ha sido tan popular y disruptivo que ha servido como base para lo que otras compañías han creado después.

Figura 2: Máquina arcade de Dance Dance Revolution

En 2006 salió Guitar Hero [2] para la consola Sony PlayStation 2. Este juego venía acompañado de una guitarra de plástico que servía de mando. En el mástil de dicha guitarra se encontraban una serie de botones de colores que el jugador tenía que presionar cuando el juego se lo indicase para así vivir la fantasía de ser una estrella de rock tocando algunas de las canciones más míticas del género. Esto se puede ver en la Figura 3. Las múltiples canciones con licencia de los grandes grupos ayudó al éxito del juego, tanto como para convertirse en una saga de prestigio. Este fue el gran juego que sirvió para popularizar el género de ritmo en el mundo de los videojuegos más allá de las máquinas *arcade.*

Figura 3: Captura de juego de Guitar Hero

Posteriormente a Guitar Hero salieron muchos clones dispuestos a competir en el nuevo género de videojuegos. Juegos como Rock band (Figura 4) [3], Piano Tiles (Figura 5) [4] o Friday Night Funkin' (Figura 6) [5] han seguido basando su jugabilidad en hacer que el jugador pulse una serie de botones a ritmo, aunque cada uno a su manera. Otra de las grandes sagas que ha definido el género es Just Dance (Figura 7) [6], que, en vez de hacer al jugador seguir el ritmo con secuencias de botones, tiene que seguirlo replicando los movimientos del bailarín en pantalla utilizando sistemas varios de captura de movimiento. Todos estos ejemplos se mantienen dentro de la base previamente establecida.

Figura 4: Captura de juego de Rock Band

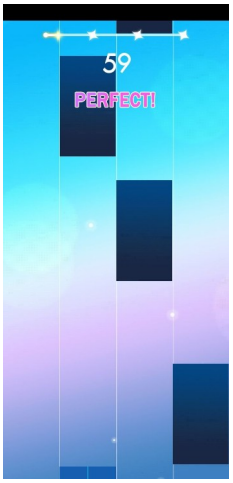


Figura 5: Captura de juego de Piano Tiles 2

Figura 6: Captura de juego de Friday Night Funkin'

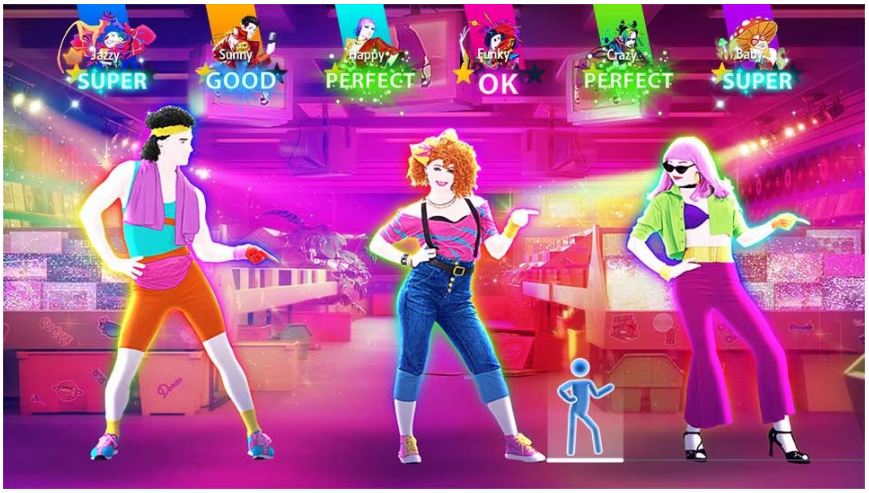


Figura 7: Captura de juego de Just Dance

Como el medio ya nos tiene acostumbrados, siempre se le puede dar un giro de tuerca a cada sistema. Y es que en los últimos años han ido apareciendo juegos que vuelven a reinventar el género trayendo consigo propuestas especialmente interesantes. Juegos como Metal Hellsinger (Figura 8) [7], Hi-Fi Rush (Figura 9) [8] y Crypt of the NecroDancer (Figura 10) [9] construyen algo más que simplemente un sistema de ritmo. Tomando por ejemplo Metal Hellsinger, que toma la base de otros juegos de disparos como el clásico Doom [10], en el que hay que matar demonios utilizando armas de fuego. Sin embargo, Metal Hellsinger añade un componente diferencial, que es el de añadir un sistema de ritmo, de tal manera que el juego se juega igual que si de un Doom (Figura 11) se tratase pero con la particularidad de que todas las acciones se tienen que realizar al ritmo de la música. Esto añade una profundidad y dificultad al juego nunca antes vista, pues ya no hay que hacer una acción en específico cuando el ritmo te lo indique si no que hay que jugar realizando cualquier acción pero solo cuando el ritmo te lo permita. Lo bueno de esta propuesta es que se puede aplicar a cualquier género, pues los Hi-Fi Rush y Crypt of the NecroDancer mencionados anteriormente son juegos de géneros completamente distintos entre sí con ese añadido del ritmo. Este tipo de propuestas son todavía consideradas de nicho dentro de la industria, pues no son muy comunes. Son especialmente complejas de desarrollar y tienen una gran barrera de entrada debido a su dificultad. Sin embargo son muy bien recibidas por su público y cuentan con un gran apoyo por parte de sus comunidades, lo que hace que cada vez más desarrolladores se animen a aportar su visión en este campo.

Figura 8: Captura de juego de Metal Hellsinger

Figura 9: Captura de juego de Hi-Fi Rush

Figura 10: Captura de juego de Crypt of the NecroDancer

Figura 11: Captura de juego de Doom Eternal

2.1 Crítica al estado del arte

Como hemos visto anteriormente, dentro de la industria del videojuego existen 2 vertientes de juegos de ritmo con características diferenciales. Vamos a analizar ambos casos.

La primera vertiente está centrada únicamente en la mecánica de ritmo. Estos juegos son los más populares dentro del género. Su enfoque es mucho más casual. Siempre juegan con la fantasía de bailar o tocar un instrumento a ritmo de la música, lo que hace que no haya mucha variedad en ellos más allá de las canciones en concreto o la estética del juego. La gran mayoría de jugadores, e incluso personas que no forman parte del colectivo habitual de jugadores, han jugado a alguno de estos juegos. Sin embargo son juegos en los que lo común no es dedicarle una gran cantidad de tiempo porque suelen volverse repetitivos. Son juegos perfectos para jugar un rato de vez en cuando pero no de manera habitual, en parte porque el contenido de estos juegos a nivel de canciones suele ser escaso. Estos juegos se basan en atraer jugadores mediante las canciones con licencias de grandes cantantes y grupos, pero esto hace que la variedad sea la justa porque conseguir estas licencias es caro. De la misma forma, grupos más pequeños o alternativos no suelen tener espacio en estos juegos.

La otra vertiente de los juegos de ritmo es la formada por juegos que construyen sus mecánicas en base a un género cualquiera y después le añaden un sistema de ritmo. Este tipo de juegos suelen ser de nicho. Es muy difícil conseguir atraer a un jugador a este tipo de propuestas, pero los jugadores de este género suelen exprimir al máximo sus mecánicas con una dedicación poco habitual en la otra vertiente de juegos de ritmo. Normalmente las propuestas que encontramos de este tipo de juegos suelen venir por parte de la escena independiente de desarrolladores. Se tratan de juegos más pequeños que suelen utilizar canciones especialmente compuestas para ellos, que simplifican su complejidad a nivel de ritmo para que el jugador sea capaz de seguir el ritmo sin depender de indicadores visuales, que están más pensados para enseñar al jugador a seguir la canción y estar de apoyo para el resto del juego. Por

norma general la variedad de canciones suele ser escasa, pues se prioriza la calidad antes que la cantidad. Los ajustados presupuestos de este tipo de juegos no suelen tampoco permitir grandes canciones licenciadas. El enfoque de este tipo de juegos está en una gran precisión en sus sistemas de ritmo. Sin embargo, el excesivo trabajo que puede llevar el implementar cada canción acaba derivando en una falta de variedad musical que puede llegar a lastrar los proyectos de este estilo.

2.2 Propuesta

Como hemos visto, uno de los problemas que tienen los juegos de este género es la variedad de canciones. Bien sea por limitaciones en presupuesto o por complejidad de procesar las canciones para crear contenido, ambas vertientes de los juegos de ritmo suelen adolecer de este problema. Es por esto que mi propuesta consiste en la implementación de un sistema de detección automática de ritmo en canciones que permita a los jugadores utilizar su propia música para jugar. Este sistema permitiría al jugador utilizar cualquier canción almacenada en su ordenador, pasando por un paso previo de procesado. De esta manera, el jugador podría crear su propia biblioteca de canciones previamente analizadas. Videojuegos similares podrían incluso utilizar esa misma biblioteca compartida para funcionar con una canción previamente procesada en varios juegos sin ningún coste y de una forma fácil, intuitiva y amigable para el usuario.

Implementar un sistema de detección de ritmo automático en los juegos de la primera vertiente sería muy complicado si no casi imposible, pues no solo hay que detectar el ritmo de dicha canción, que ya es un problema de gran dificultad, si no que también hay que construir las mecánicas para dicha canción. No solo tendríamos que crear un sistema que detectase ritmo si no uno que inventase coreografías de baile y / o secuencias de botones que simulasen las notas musicales. Para estos casos este tipo de juegos suele buscar un enfoque de calidad antes que cantidad, teniendo pocas canciones muy trabajadas. Debido a que los sistemas de detección de ritmo actuales no son todo lo precisos que desearíamos, los pasos añadidos al uso de canciones en estos juegos hacen que esta tarea sea casi inviable.

Sin embargo y pese a lo comentado anteriormente respecto a la segunda vertiente, considero que añadir un sistema que permita al jugador jugar con cualquier canción puede ser un punto diferencial y que puede ser factible de añadir en juegos que utilicen esta aproximación a los juegos de ritmo. Es particularmente difícil diseñar un sistema de detección automática de ritmo preciso, pero en muchos casos se podría llegar a una precisión lo suficientemente aceptable como para que el jugador pueda realmente disfrutar de un juego de este estilo con el añadido de usar su propia música. Tal vez no es un sistema lo suficientemente desarrollado a día de hoy como para lanzar un juego que apostase por darle al usuario libertad total de canciones a usar, pero si que puede ser un añadido interesante como recompensa para jugadores que logren terminar el juego de la manera establecida o como contenido de ampliación posterior al lanzamiento del juego base. Incorporar esta mecánica como un modo secundario de juego añade una gran rejugabilidad al título que es difícil de conseguir de otra forma. Además, es un sistema considerablemente más fácil de implementar en esta vertiente del género que en la otra, pues aquí solo hay que poder procesar la canción para detectar su ritmo ya que las acciones a realizar no vienen definidas por la canción en específico si no por el tipo de juego en cuestión. De la misma forma, como el trabajo en específico que requiere cada canción se resume en extraer el ritmo de la forma más precisa posible, un sistema automático puede ayudar como primer paso a la hora de trabajar con una canción que se quiera añadir al juego, aunque luego este proceso de análisis sea complementado con correcciones

manuales. Por último también cabe destacar que desarrollar un sistema de detección de ritmo desacoplado del juego es útil para el desarrollo de juegos de este estilo, pues todos los juegos que trabajan con este tipo de propuestas buscan lo mismo, extraer las marcas de tiempo de los golpes de ritmo. El análisis que se puede hacer para juegos como Metal Hellsinger es el mismo que se puede hacer para juegos como Hi-Fi Rush, permitiendo utilizar una canción pensada para un juego en el otro sin ningún cambio pese a que los juegos sean completamente distintos. De esta manera, un sistema automatizado de detección de ritmo puede ser útil en ambos juegos, permitiendo incluso que juegos de géneros diferentes puedan tener acceso a una biblioteca de canciones previamente procesadas por el usuario que podrían ser usadas directamente en ambos juegos sin coste alguno.

3 Análisis del problema

El punto de partida es un juego desarrollado para la asignatura de Desarrollo de Videojuegos 3D. Como se ha comentado en el apartado anterior, el proyecto fue desarrollado por cuatro estudiantes de Ingeniería informática en la ETSINF y dos estudiantes de Diseño y tecnologías creativas de la Facultad de BBAA. Casi la totalidad del contenido del proyecto fue creado desde cero por los 6 integrantes del grupo, por lo que se trata de un desarrollo completamente original.

En el momento de la entrega final del proyecto para la asignatura, el juego contaba con un menú principal, una pequeña cinemática inicial, un nivel tutorial previo al nivel principal, una demo del primer nivel de juego funcional y casi terminada y un apartado de créditos.

El menú principal (Figura 12) sirve como primera toma de contacto con el juego. En este se puede elegir empezar una nueva partida, mostrar los créditos o salir del juego. Junto a esto aparecen 2 opciones más, como son el menú de opciones y la opción de continuar una partida anterior. Dichas opciones nunca fueron implementadas y, de hecho, el botón de continuar se ha sustituido por una opción para jugar con las canciones seleccionadas por el usuario.

Figura 12: Menú principal de Unwanted

Al elegir una nueva partida, una pantalla de carga lleva a una pequeña cinemática (Figura 13) que introduce la historia del juego con una narración que simboliza los pensamientos de la protagonista al momento de despertarse aturdida en un mundo desconocido.

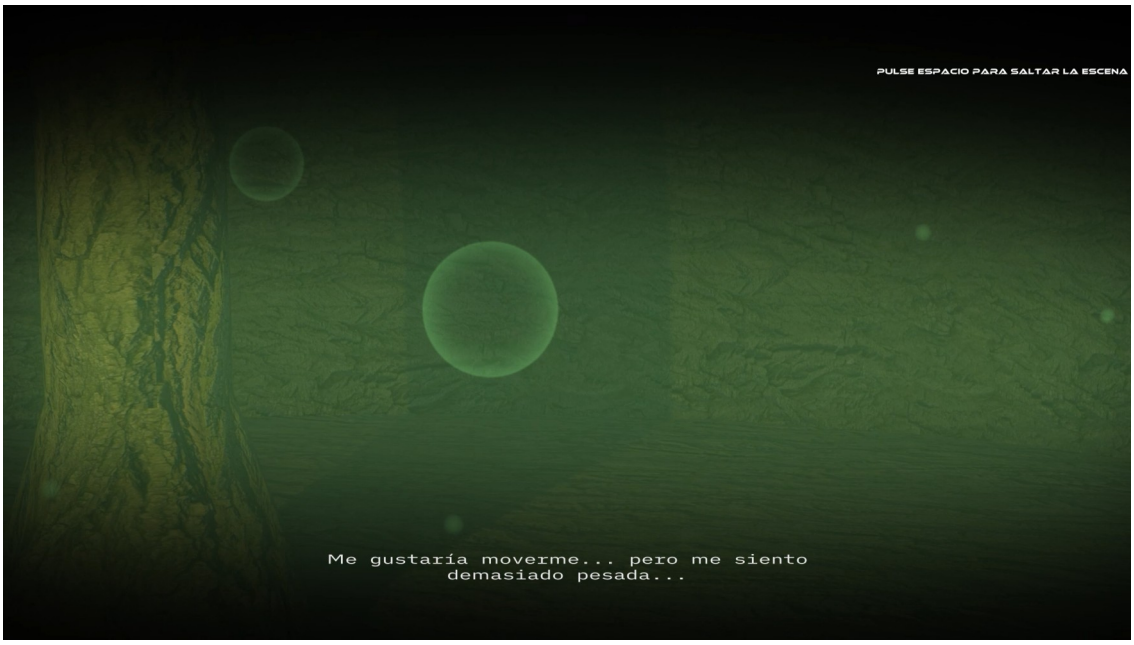


Figura 13: Cinemática inicial

Después de la cinemática se muestra un nivel tutorial (Figura 14) que introduce las mecánicas básicas del juego antes del inicio de una nueva partida, permitiendo así al jugador entrar en acción una vez conocidas las opciones que tiene a su disposición.

Figura 14: Tutorial de juego

Tras completar el tutorial se empieza directamente con el primer nivel. Este está constituido por un camino en un bosque oscuro que lleva a una arena de combate. Los caminos están indicados por unas mariposas que marcan la dirección a seguir. Al llegar a cada arena, el jugador queda atrapado, teniendo que enfrentarse a una serie

de oleadas de enemigos en un espacio cerrado para poder salir y continuar con la aventura. El mapa está dividido en varias secciones de camino que conectan con cuatro arenas básicas y una batalla contra un jefe final de nivel. En la Figura 15 podemos ver la primera arena de juego junto con algunos enemigos. Cada una de las arenas tiene una temática distinta y un nivel de dificultad incremental, con más enemigos y de mayor poder para así otorgar una cierta progresión al nivel. Hay cinco enemigos distintos con comportamientos únicos que otorgan variedad a la jugabilidad. Esto junto al diseño de las arenas con distintos obstáculos, alturas y temáticas hace que cada arena se sienta como una experiencia única. Por limitaciones de tiempo, no se pudo implementar ni la última de las 4 arenas ni la pelea final contra el jefe, pero sí se diseñaron.

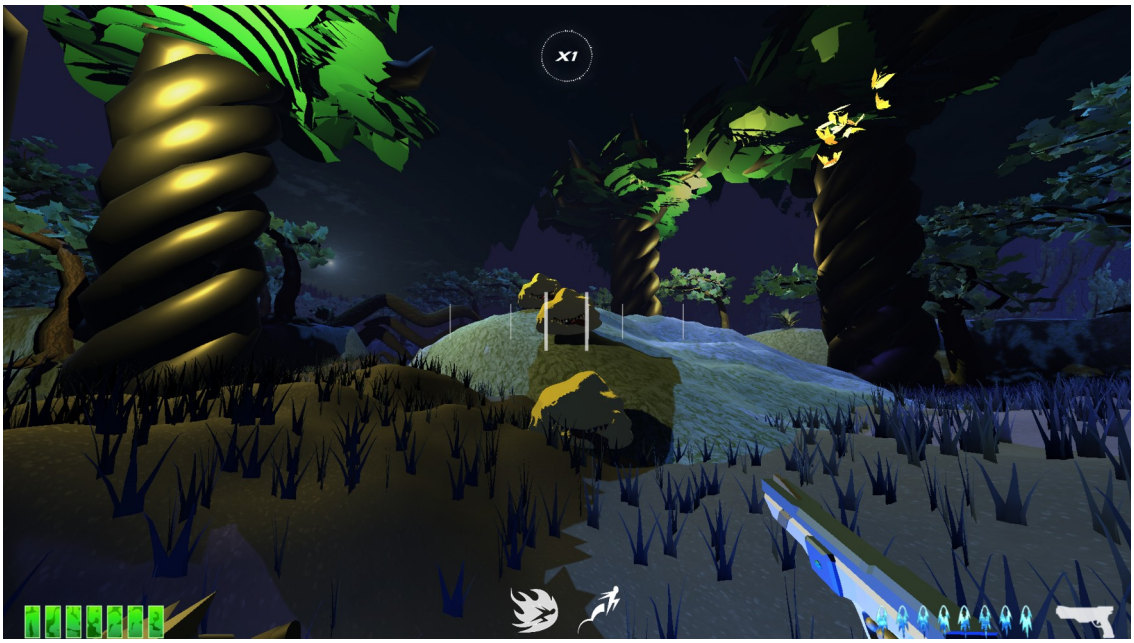


Figura 15: Primera arena de juego

En cuanto a las mecánicas del juego, que es la pieza más importante, las podemos dividir en dos tipos:

Por un lado tenemos las mecánicas de ataque:

- Pistola: arma principal de juego. Se puede disparar para dañar a los enemigos. Tiene munición infinita, pero se tiene que recargar cada ocho disparos. Es semiautomática, por lo que se tiene que soltar el clic para poder volver a disparar. Tiene alcance infinito.
- Katana: arma secundaria. Tiene mayor daño que la pistola, pero ataca más lento y su alcance está limitado a corta distancia. Tiene la particularidad de que cura al jugador un poco de vida al matar a un enemigo.

Al disponer de dos armas con usos diferenciados damos la oportunidad al jugador de crear una cierta estrategia para estudiar cómo enfrentarse a los combates de una manera óptima en cada caso.

Por otro lado tenemos las mecánicas de movimiento:

- Doble salto: aparte de saltar, el jugador puede realizar un doble salto en el aire, que le permite acceder a plataformas elevadas o esquivar los ataques de

los enemigos.

- *Dash*: un pequeño sprint que permite esquivar ataques enemigos y huir de estos.

Estas mecánicas otorgan una importante movilidad al jugador que le permiten desplazarse de manera ágil por todo el nivel de juego

Junto a estos 2 tipos de mecánicas hay que contar con la pieza clave para el juego, que es el sistema de ritmo. El juego cuenta con una única canción de rock. Gracias a que Snake, sus autores, nos proporcionaron dicha canción dividida en varias pistas de audio por cada instrumento, se analizó únicamente la pista que contenía la instrumental de la batería para extraer el ritmo. Normalmente la percusión es la que suele marcar de manera más clara y sencilla el ritmo de la canción, motivo por el cual se seleccionó esta pista en concreto. El proceso de análisis se hizo utilizando Audacity, al ser software libre y de fácil acceso. Como Audacity no incluye de manera nativa ninguna función para hacer el análisis de ritmo en canciones se utilizó un paquete de plugins conocido como Vamp Plugins [12] que disponen de varias utilidades para hacer este reconocimiento. Dichos plugins generan una serie de etiquetas de tiempo en Audacity que fueron extraídas a un fichero de texto. Ninguno de los algoritmos de Vamp Plugins es perfecto, por lo que partiendo de la base que proporciona el fichero se modificaron a mano estas etiquetas de tiempo para dar lugar a un fichero resultado que contiene una lista de segundos en los cuales se supone que existe un golpe de ritmo. Unwanted lee este fichero de texto y guarda en una lista estos tiempos. En pantalla se muestran dos barras transparentes al lado de la mira, junto a una serie de barras que se generan a los lados de la pantalla y viajan hasta cruzarse con las barras que hay fijas en el centro. Cuando las barras móviles se superponen con las barras fijas se considera que hay un golpe de ritmo, como indica el fichero preprocesado. Esto se muestra cambiando las barras fijas transparentes a un color blanco. Para poder realizar dentro del juego cualquiera de las acciones mencionadas anteriormente, ya sea atacar con la pistola o con la katana o hacer dobles saltos o *dashes*, es necesario coordinar estas acciones con los momentos en los que ocurre un golpe de ritmo. Para indicar que dichas acciones han tenido éxito, las barras fijas se pondrán de color amarillo. Fallas a ritmo un *dash* o un doble salto desactiva temporalmente estas acciones y no hace nada a nivel de movimiento. Las acciones de ataque si que se ejecutarán fuera de ritmo pero sin hacer daño a los enemigos, lo que obliga al jugador a atacar a ritmo. Realizar acciones de manera exitosa a ritmo otorga al jugador un multiplicador temporal que puede aumentar al seguir realizando acciones a ritmo o disminuir al fallar el ritmo o con el paso del tiempo. Este multiplicador aumenta el daño realizado a los enemigos y estaba planificado que otorgase puntos extra al jugador al superar el nivel. El sistema de puntos nunca fue implementado del todo. Como vemos, el sistema de ritmo es la pieza central del juego y el punto más importante de este, siendo necesario que el jugador aprenda a detectar el ritmo de la canción al escucharla o siguiendo las barras que se mueven por la pantalla para poder jugar de forma óptima.

El proyecto del juego era demasiado ambicioso para lo que el tiempo de la asignatura para la que fue desarrollado daba. Por este motivo existen muchas cosas que fueron planificadas para el juego pero que nunca llegaron a aparecer en la versión previa al desarrollo de este TFG. Entre ellas podemos encontrar el sistema de ritmo dinámico adaptable a cualquier canción, que es la característica objetivo a implementar en este trabajo. Otras características podrían ser el nivel de juego terminado con una pelea contra un jefe final, varios niveles con enemigos y ambientación distinta, un sistema de puntos que permitiría al jugador almacenar puntuaciones obtenidas al completar el nivel para añadir mayor rejugabilidad, cinemáticas que contarían la historia del juego entre otros.

Para más información se puede consultar en los anexos el *game design document* de Unwanted que contiene toda la información detallada de todo el contenido del juego, tanto planificado cómo implementado.

Por todo lo comentado en el apartado anterior parece especialmente interesante de cara a la escalabilidad del proyecto implementar un sistema que permitiese al propio juego, en tiempo de ejecución y sin necesitar de aplicaciones externas, detectar el ritmo de cualquier canción. Esto a su vez puede ser una opción más de cara al jugador para poder añadir al juego un componente de rejugabilidad y de mayor personalización al juego, la cual sería una característica diferencial para el proyecto al ser algo que otros juegos similares en el mercado no tienen.

Para poder implementar este sistema se han revisado diversos sistemas de detección de ritmo, desde librerías que contienen detectores de ritmo ya incorporados hasta librerías que permitirían programar una detección de ritmo utilizando sus funciones. Mediante el uso de estas librerías se plantea diseñar un sistema que permita procesar una canción cualquiera seleccionada por el jugador de entre cualquier archivo en formato mp3 que contenga en su ordenador. Esta canción seleccionada sería incorporada a una biblioteca de canciones previamente procesadas en una carpeta en específico del juego, junto a un archivo de texto que contendría las marcas de tiempo obtenidas en el procesamiento. De esta manera permitimos al jugador jugar directamente con cualquier canción previamente procesada sin tener que esperar a obtener el ritmo de dicha canción, lo cuál conlleva un determinado coste temporal, o cargar una nueva canción en la biblioteca de canciones para su uso en este momento o en cualquier otro.

Gracias a almacenar en ficheros de texto las marcas temporales de los golpes de ritmos de cada canción se puede reutilizar el sistema de ritmo previamente implementado sin la necesidad de modificar especialmente la función previa. Además, como el sistema genera archivos genéricos que solamente contienen estas marcas, este mismo sistema de detección de ritmo podría ser utilizado en cualquier otro desarrollo con Unity para el que fuera necesario utilizarlas, ya que el sistema que procesa las canciones para detectar su ritmo está desacoplado del sistema que muestra el ritmo en el juego.

Este sistema tiene dos problemas especialmente difíciles de gestionar. El primero es que la precisión de los algoritmos para la detección de ritmo tiene un cierto margen de error, lo que conlleva a que el sistema de ritmo en el juego funcione ligeramente peor con una canción previamente procesada y editada a mano que con una canción simplemente procesada por este sistema y sin ningún tipo de revisión humana. El segundo problema que tiene es que las implementaciones de librerías pueden ser dependientes de la plataforma en cuestión, limitando lo que en principio sería un proyecto ejecutable también en Linux o Mac a poder ser ejecutado únicamente en sistemas Windows, pudiendo necesitar de implementaciones específicas para otras plataformas. Este problema se considera algo asumible y aceptable, pues Windows es el principal sistema operativo elegido por la inmensa

mayoría de los jugadores de videojuegos en ordenador, reduciendo el posible mercado para este proyecto pero de una forma asequible.

Desarrollar Unwanted fue una experiencia multidisciplinar muy enriquecedora y, para no tener experiencia previa ninguno de los integrantes del equipo en el desarrollo de videojuegos a este nivel, creo que el resultado obtenido es de gran calidad. Sin embargo, la falta de conocimientos de la materia y la dificultad del proyecto, sumado al contexto de desarrollo para una asignatura más de la carrera, nos llevó a tomar decisiones equivocadas y a abordar problemas de formas mejorables.

Uno de los principales problemas que tuvo el juego se trata de un problema de rendimiento de este en ejecución y de organización del proyecto, tanto a nivel de código como de carpetas. Esto hizo especialmente difícil la escalabilidad del proyecto, tanto en el contexto de las últimas entregas de la asignatura como en el del desarrollo posterior de una ampliación como temática de TFG. Es por esto que, como se comenta en el apartado de colaboradores, mi compañero Jose Fran ha enfocado todo el desarrollo de su TFG en mejorar estos aspectos del juego.

Otro de los problemas con los que nos enfrentamos fue el sistema de detección de ritmo. Detectar ritmo en canciones es un problema de una gran envergadura. Para un músico experimentado no es difícil escuchar una canción cualquiera y marcar su ritmo. Una persona cualquiera con un poco de odio musical puede seguir el ritmo de una canción de manera casi instintiva y con una más que aceptable precisión. Sin embargo, realizar este proceso de manera automatizada por un programa es increíblemente complejo, hasta el punto de que a día de hoy sigue siendo un reto importante. Existen diversas técnicas para realizar esto, pero requieren de un conocimiento en la materia muy avanzado y especializado para conseguir un resultado aceptable, aunque la precisión de cualquiera de estos métodos nunca llega a obtener una precisión perfecta en cualquier tipo de canción.

Para poder implementar este sistema en el juego, la primera versión consistía en una serie de bucles que marcaban, durante un margen de tiempo, que en ese punto existía un golpe de ritmo. Este sistema contaba con la canción tenía golpes de ritmo cada intervalos regulares con una extensión fija y repetida en el tiempo hasta el fin de la canción. El problema que tenía este sistema es que no acababa de ser preciso, pues el ritmo no es constante en toda la canción, cambiando estos intervalos de tiempo en los que hay golpe de ritmo. De esta manera, al inicio todo funcionaba bien pero acababa por descuadrarse todo inevitablemente. Para solucionar este problema se implementó el sistema explicado en el apartado anterior, en el cual se pasó la canción en cuestión por un procesado previo en Audacity para obtener un fichero de texto que contuviera las marcas de tiempo en las cuales se supone que existe un golpe de ritmo. Esto evitaba los desfases pero generaba otro problema, y es que estos algoritmos de los plugins de Audacity no son todo lo precisos que nos gustarían, obligándonos a modificar el archivo a mano. De esta manera obtuvimos una versión final jugable de una calidad aceptable y que dejaba jugar sin mayor complicación al juego. El problema que este sistema genera es que sería necesario volver a preprocesar con Audacity y editar a mano cualquier canción que se quisiera implementar en el juego, cosa que estaba planteado que ocurriese de cara a haber implementado diversos niveles de juego.

3.1 Identificación y análisis de posibles soluciones

El problema principal que se ha abordado es la detección de ritmo propiamente dicha. Partimos de un sistema que espera un fichero de texto que contiene las marcas temporales en las que se considera que existe un golpe de ritmo. El juego original

sólo permitía jugar con la canción *Somewhere in the night*, que el grupo *Snake* nos cedió para el juego. De esta manera, el sistema de ritmo indica al jugador que en ese preciso instante, con un pequeño margen antes y después, existe un golpe de ritmo. Cuando el jugador realiza una acción, se tiene que comprobar que la canción se encuentra en un golpe para poderse realizar. El problema consiste en encontrar, de algún modo, un sistema que permita procesar cualquier canción, como paso previo al juego, para obtener la lista de marcas de tiempo. Originalmente se usó Audacity y un postprocesado manual para obtener dichas marcas de tiempo. Sin embargo es necesario implementar un sistema que pueda generar estas marcas en tiempo de ejecución, a costa de perder precisión por no poderse editar a mano el resultado obtenido.

La primera solución que se planteó es seguir con Audacity, pues era lo más cercano a la versión ya implementada del sistema de ritmo. Audacity es un programa de código abierto y gratuito utilizado para la edición de sonido. Permite desde tareas sencillas como cortar o mezclar audios, transformar archivos de un formato a otro, aplicar efectos de sonido y hacer análisis de audio. El problema es que Audacity por defecto no cuenta con ningún sistema para detectar ritmo en canciones. Sin embargo, una de las bondades del software libre y el amplio uso de esta herramienta es que existen multitud de *plugins* que se pueden instalar para ampliar sus funciones básicas.

Vamp Plugins es un conjunto de varios *plugins* para Audacity desarrollado en C++ y Python utilizados para extraer información de audios, principalmente canciones. Este conjunto de *plugins* ha sido desarrollado por múltiples organismos, como la Queen Mary University of London. En la web de Vamp Plugins se puede encontrar un pack [13] con algunos de los *plugins* más importantes del grupo que puede ser fácilmente instalado en Audacity y que contiene, entre otras funciones, cuatro implementaciones distintas de sistemas para la detección de ritmo. Estos *plugins* analizan la canción y generan una pista de etiquetas, con los instantes de tiempo en los cuales el sistema detecta que existe un golpe de ritmo, junto con información adicional. Estas etiquetas se pueden exportar en formato de archivo de texto. De esta manera y editando las marcas temporales a mano obtuvimos el ritmo de la canción base del proyecto.

Uno de los planteamientos que surgió para implementar el sistema de detección de ritmo fue tratar de utilizar la *api* de Audacity para poder procesar las canciones de la misma forma que lo habíamos hecho con anterioridad pero esta vez de manera automática. Sin embargo, esta opción acabó siendo descartada debido a que la herramienta de scripting de Audacity está disponible únicamente para Python y Perl. Unwanted ha sido desarrollado utilizando Unity utilizando C#, lo que limita las herramientas disponibles a este lenguaje. Es posible utilizar herramientas en otros lenguajes pero eso conlleva una mayor complejidad en el desarrollo. Además, la herramienta de análisis de ritmo no viene incorporada de base en Audacity, si no que se trata de un *plugin* externo. No existe mucha documentación alrededor de los Vamp Plugins y no parecía quedar claro si existe una forma de llamar a las funciones de los *plugins* mediante la *api*. Sin embargo este trabajo con los Vamp Plugins no ha sido desperdiciado pues, como se comentará más adelante, se han usado para hacer comparativas de la precisión de los sistemas implementados.

El siguiente planteamiento es el de implementar desde cero un sistema de detección de ritmo. Para ello realicé una investigación previa de métodos que ayudarían a hacer dicha implementación y de librerías que me podrían ayudar.

La aproximación a la que llegué sería la de utilizar la transformada de Fourier [14] para calcular los golpes de ritmo. La transformada de Fourier es una herramienta matemática que permite convertir señales de tiempo y frecuencia. Utilizando esta técnica se puede subdividir cada canción en ventanas de tiempo y calcular la energía

almacenada en dicha ventana. Si la diferencia de energía entre una ventana y sus contiguas es mayor que un determinado umbral, se considera que en dicha ventana ocurre un golpe de ritmo. Esto es así porque de manera tradicional se utilizan instrumentos de percusión como la batería para marcar el ritmo que siguen el resto de instrumentos y voces. Estos golpes de la percusión suelen estar especialmente marcados con grandes cantidades de energía respecto a la frecuencia de la canción, lo que los hace fácilmente reconocibles para un oyente y para un sistema informático. A primera vista se pueden reconocer estos golpes simplemente visualizando la onda de sonido en aplicaciones como Audacity.

Muchos de los sistemas de detección de ritmo en canciones utilizan esta técnica para hacer su análisis. Con distintos parámetros de tamaño de ventana, umbral de energía para un golpe de ritmo y técnicas estadísticas de repetición de patrones dentro de la canción se pueden obtener resultados bastante positivos. Como mínimo esto es una primera aproximación válida al problema.

Con la intención de aplicar la técnica anteriormente explicada procedí a buscar librerías que me permitieran hacer esta implementación. NAudio [15] fue una de las candidatas, pues es una librería de código abierto desarrollada para .NET de gran adopción. Permite cosas como grabar o reproducir audio o hacer conversiones de formato. También permite hacer un cierto análisis de audio. Sin embargo, la librería elegida para este proceso sería FFTSharp [16], pues es una librería de código abierto desarrollada por el MIT que implementa la transformada de Fourier de manera nativa, reduciendo en parte la complejidad del sistema.

Pese a todo el trabajo realizado anteriormente, acabé decidiendo descartar esta opción en el desarrollo. Crear mi propio sistema de detección de ritmo es una tarea excepcionalmente compleja que requiere de unos conocimientos en física y matemáticas de los que no dispongo. Se trata de una tarea que se escapa de mi especialidad. Habiendo hecho un cierto análisis de herramientas previas y tomando como referencia la experiencia con los *plugins* de Audacity llegué a la conclusión de que la detección de ritmo en canciones es una cuestión muy compleja incluso para expertos. No fui capaz de encontrar una herramienta que generase de manera eficaz resultados perfectos, incluso las desarrolladas por expertos. Por todo esto se decidió no hacer una implementación propia, y utilizar alguna librería que implementase de manera nativa estos métodos de detección de ritmo.

Dentro de las librerías estudiadas se encontraron dos alternativas: Aubio [17] [18] y Essentia [19]. Ambas alternativas generan el mismo problema con respecto a la forma de integrarlos en Unity, y es que Aubio está desarrollada para C y Essentia para C++ y Python. Al no disponer de manera nativa de una versión para C# se plantean dos posibles formas de proceder. Una de las alternativas consistiría en generar una dll con las funciones de detección de ritmo de la librería escogida y llamar a esta dll mediante P/Invoke desde un *script* en C#. La otra alternativa sería la de generar un pequeño ejecutable escrito en el lenguaje específico de la librería que proporcionase el resultado del análisis esperado y simplemente ejecutar dicho programa previamente compilado como si de una orden de terminal se tratase.

Respecto a si utilizar una dll o un ejecutable precompilado, la decisión era complicada. Por un lado, ambas soluciones tienen un problema y es que hacen el sistema de detección de ritmo dependiente de la plataforma. Windows es el sistema operativo mayoritario a nivel de usuario de ordenadores y prácticamente la única alternativa viable para aquellos que somos jugadores habituales en ordenador, pues ni Linux ni MacOS disponen del catálogo ni las herramientas de Windows. Según las encuestas de usuario de Steam (Figura 16) [23], el principal distribuidor de videojuegos en PC, el 96.21% de sus usuarios utilizan una versión de Windows a fecha de mayo de 2024, siendo Windows 10 y Windows 11 las mayoritarias. Sin

embargo, no deja de ser un inconveniente directamente hacer que un determinado porcentaje de usuarios no pueda disponer de un servicio solo por su sistema operativo, cosa que no pasaba antes de la implementación de esta funcionalidad.

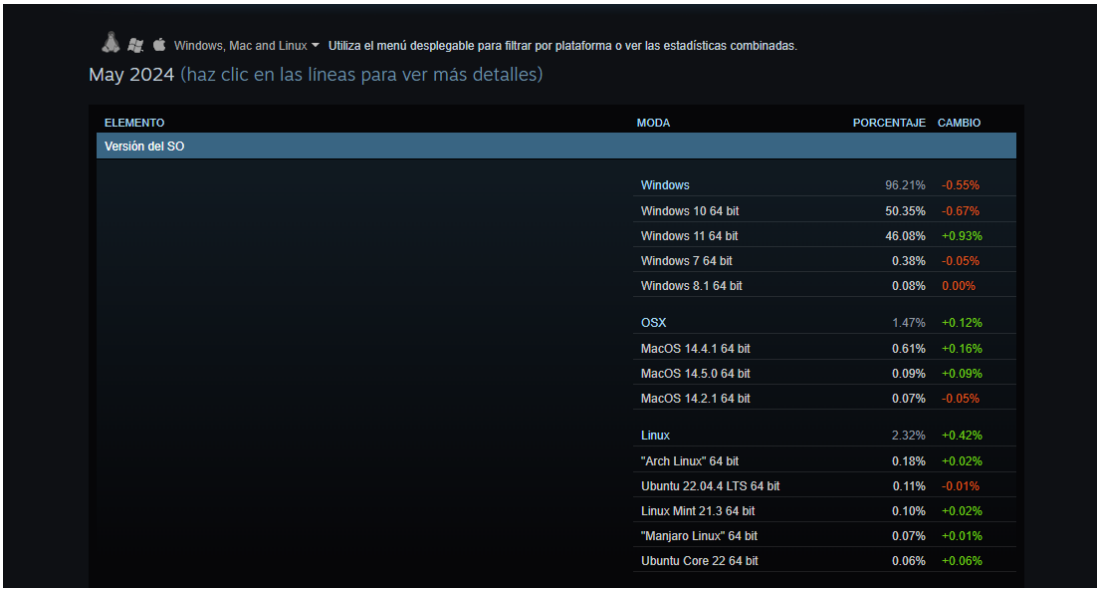


Figura 16: Distribución de usuarios de Steam por sistema operativo (Mayo de 2024)

Tras estudiar las diferencias entre ambas alternativas, se optó por utilizar un ejecutable. Entiendo Probablemente a nivel conceptual es más correcto utilizar una dll que un ejecutable, pero tratándose de un programa de dimensiones muy reducidas no lo considero un problema. Generar un ejecutable es mucho más sencillo y permite probarlo de una forma más fácil. De la misma manera, esto tiene una ventaja clara. Como se ha comentado en el apartado 2.2 Propuesta referente al estado del arte, este sistema de detección automática puede ser utilizado no solo como una característica a implementar directamente en un videojuego de ritmo, si no que se puede utilizar como paso previo de procesado de una canción cualquiera que el desarrollador decida incorporar al juego, modificando a mano los resultados para ajustarlos más a la realidad como previamente hicimos con Unwanted. Disponer de un pequeño programa de terminal que, en esencia, sea el que haga el procesado de canciones, puede ayudar a implementar esta funcionalidad incluso usando directamente el ejecutable.

La última decisión queda en si utilizar Aubio o Essentia. Resulta que uno de los *plugins* incorporados dentro del paquete de Vamp Plugins para Audacity utilizado para las comparativas que se muestran más adelante en el apartado 5.1 Evaluación interna utiliza Aubio para generar las etiquetas de tiempo resultado del análisis. Para comprobar el grado de precisión de la herramienta de análisis, se han probado los 4 análisis disponibles en este *pack* de *plugins* con la solución propuesta por Essentia. De dicha información se puede extraer que Aubio es siempre el sistema de los disponibles en Audacity que peor funciona y con bastante diferencia. Por ello se descartó su uso para la implementación definitiva del juego, optando por Essentia. Otra de las razones de seleccionar Essentia es que tiene un repositorio ampliamente detallado que contiene ejemplos de uso y sus versiones ya compiladas para ejecutar. Uno de estos ejemplos hace exactamente lo que se necesita para este proyecto, leer una canción y utilizar el método de detección de ritmo para imprimir en un fichero de texto las marcas temporales en las que se encuentra un golpe de ritmo.

3.2 Solución propuesta

La solución finalmente implementada consiste en una escena de Unity que se utilizará como paso previo a empezar una partida, si se selecciona el modo de juego con canciones personalizadas. En esta escena el jugador se encontrará con una lista de canciones almacenadas en una biblioteca local creada por el juego. El programa permitirá al usuario seleccionar un archivo, en formato .mp3, dentro de su ordenador y procesarlo utilizando Essentia. De entre todas las canciones procesadas previamente, en esa misma ejecución del juego o en una anterior, el jugador elegirá una y procederá a jugar con ella, utilizando el archivo resultado del análisis para marcar el ritmo en pantalla. En caso de que el jugador lo desee, el juego permite jugar la experiencia básica de Unwanted con la canción por defecto, siendo esta la experiencia originalmente planeada, a la vez que se le permite disfrutar del juego desde un punto de vista único y con una experiencia personalizada.

3.2.1 Especificación de requisitos

Partiendo de la idea base que da origen a todo el proyecto a desarrollar, es necesario definir los requisitos, tanto funcionales como no funcionales, que debe cumplir el proyecto. Estos requisitos marcan los objetivos que se deben de cumplir una vez terminado el desarrollo.

Requisitos funcionales:

1. Selección y procesamiento de canciones
    a. RF1: El sistema debe permitir al jugador seleccionar una canción cualquiera almacenada en su propio ordenador.
    b. RF2: El sistema debe añadir la canción seleccionada a una lista de canciones disponibles.
    c. RF3: El sistema debe procesar la canción seleccionada para extraer el ritmo de la misma en tiempo de ejecución.
    d. RF4: El sistema debe actualizar la lista de canciones disponibles de manera dinámica.
    e. RF5: El sistema debe detectar el ritmo de la canción seleccionada con precisión.
    f. RF6: El sistema debe ser transparente para el usuario, sin necesidad de configuraciones manuales.
    g. RF7: El sistema debe permitir al usuario utilizar canciones previamente procesadas sin ningún procesamiento adicional.
2. Interfaz de usuario
    a. RF8: La interfaz debe mostrar una lista de canciones disponibles para que el jugador seleccione.
    b. RF9: La interfaz debe permitir al jugador navegar entre diferentes menús, incluyendo la selección de canciones y el inicio del juego.

3. Partida

    a. RF10: El sistema debe reproducir la canción elegida por el jugador

    b. RF11: El sistema debe marcar correctamente en pantalla el ritmo de la canción seleccionada por el jugador

    c. RF12: El sistema debe permitir al jugador jugar con la configuración original de sonido y ritmo.

Requisitos no funcionales:

1. Rendimiento

    1.1. RNF1: El procesamiento de la canción seleccionada debe ser eficiente y no causar retrasos significativos en el inicio del juego.

2. Usabilidad

    2.1. RNF2: La interfaz de usuario debe ser intuitiva y fácil de navegar.

3. Escalabilidad

    3.1. RNF3:El sistema debe ser capaz de manejar un gran número de canciones en la lista de canciones disponibles sin degradar el rendimiento.

4. Fiabilidad

    4.1. RNF4: El sistema de detección de ritmo debe ser confiable y proporcionar resultados consistentes con diferentes géneros musicales.

    4.2. RNF5: El juego debe manejar adecuadamente errores o fallos en la selección y procesamiento de canciones sin causar interrupciones en la experiencia del usuario.

4 Diseño de la solución

En este apartado se detalla el diseño de la aplicación. Primeramente se analizará su arquitectura en grandes bloques para posteriormente analizar en profundidad como está construido el sistema. Por último se hará referencia a todas las tecnologías utilizadas para poder llevar a cabo el desarrollo.

4.1 Arquitectura del sistema

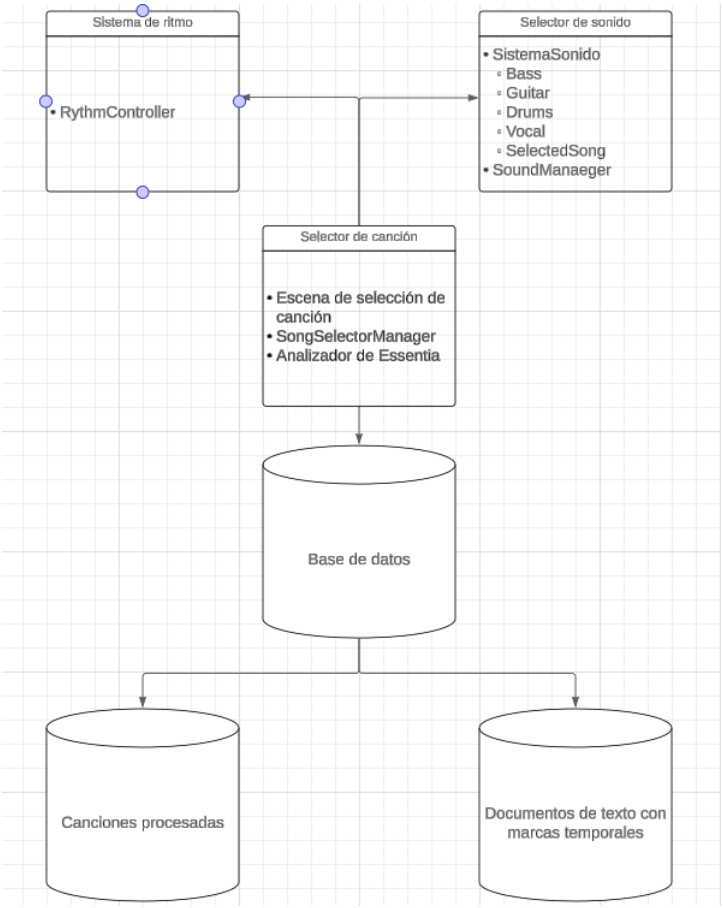


Figura 17: Diagrama de componentes UML

Como podemos ver en la Figura 17, el sistema está compuesto por cuatro bloques, siendo el selector de canción la pieza central y el punto de unión con la base de datos, el sistema de ritmo y el sistema de sonido.

Cuando se selecciona el modo de juego “Mis canciones” desde el menú principal del juego, se carga la ventana del selector de canciones. Este accederá a (o creará si no existe) la base de datos local de canciones procesadas. Desde aquí se

pueden procesar canciones nuevas y añadirlas a la base de datos utilizando el analizador de Essentia y seleccionar la canción con la que se desea jugar.

Cuando se selecciona una canción y se procede a jugar se carga la escena de la cinemática inicial del título y, posteriormente, el tutorial del juego. En este momento, los sistemas de ritmo y de sonido detectarán si existe en escena el componente específico del selector de canciones para pedirle el archivo de texto y la canción previamente seleccionados. En caso de que no exista el componente del selector de canciones significa que se está jugando a Unwanted de manera tradicional, por lo que ambos sistemas utilizan los valores por defecto.

4.2 Diseño detallado

Al iniciar el juego en pantalla se muestra el menú principal (Figura 18). En esta escena tenemos varias opciones disponibles para el jugador. “Salir del juego” cierra la aplicación, “Créditos” permite al jugador ver los créditos del juego. La opción “Configuración” no llegó a ser implementada, pero serviría para que el jugador pudiese configurar opciones como el volumen de la música. Las dos opciones restantes son las realmente interesantes. “Nueva partida” permite al jugador disfrutar de la experiencia por defecto de Unwanted tal y como fue concebida originalmente, con diversas correcciones y mejoras que se han implementado. La opción realmente interesante para este trabajo es “Mis canciones”, pues es el modo de juego alternativo que ha se ha desarrollado en este trabajo. En la Figura 19 podemos ver el diagrama de transición de escenas que resume toda esta interacción.

Figura 18: Menú principal

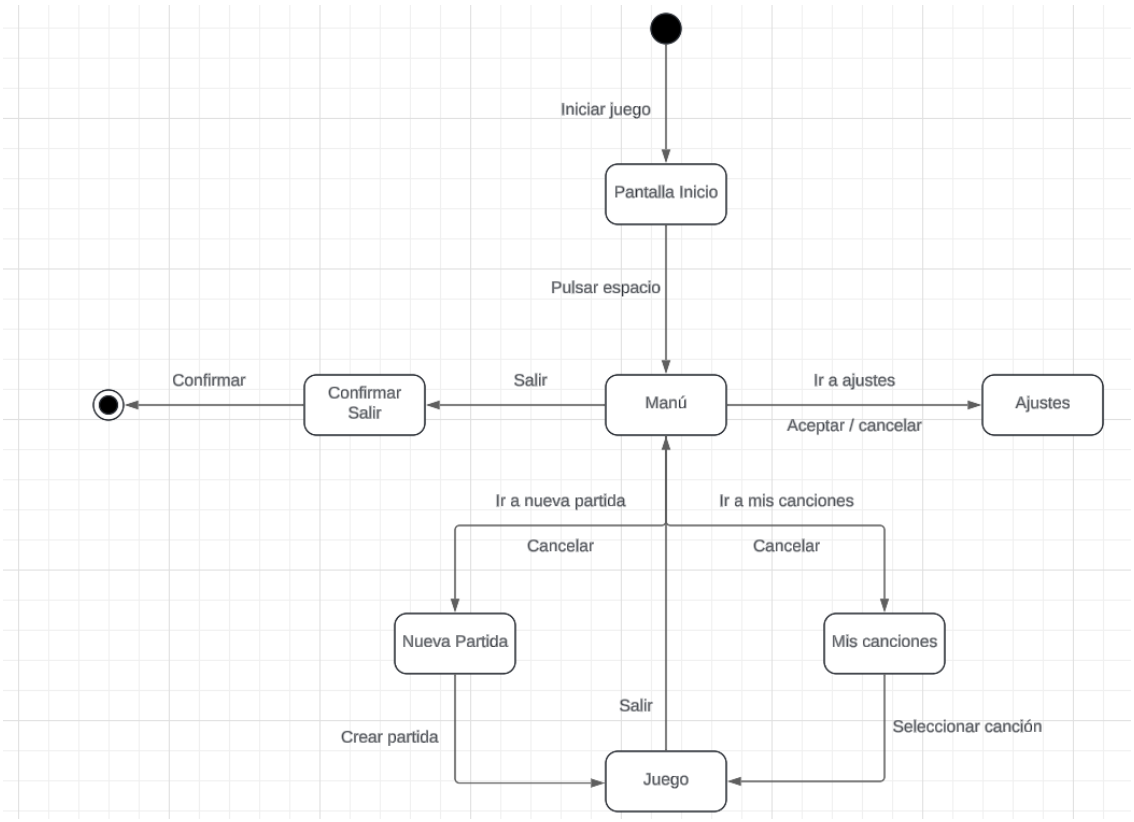


Figura 19: Diagrama de transición de escenas

Desde esta escena se puede volver en todo momento al menú principal, cumpliendo así con el RF9.

El uso en sí de la interfaz es bastante sencillo, como se puede observar en la Figura 21. Únicamente contamos con dos botones y una lista de canciones procesadas. Elegir canción también es intuitivo, ya que se abre directamente una ventana del explorador de archivos del sistema. De esta manera se cumple con el RNF2. La Figura 20 muestra el mockup utilizado para planificar.

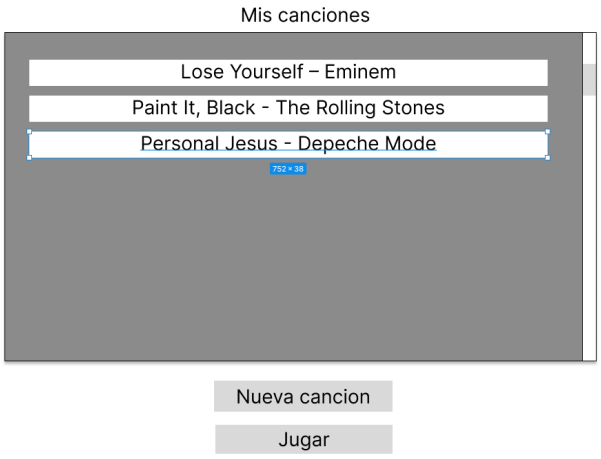


Figura 20: Mockup de la escena de selección de canción

Figura 21: Escena de selección de canción terminada

Al acceder al modo de juego “Mis canciones” se carga la escena de juego correspondiente. En la Figura 20 puede ver el mockup de la escena tal y como fue pensada y en la Figura 21 el resultado final de la escena.

Nada más cargarse la escena, se comprueba si existe la carpeta que será utilizada como base de datos dentro del equipo del jugador. En caso de que no exista, se creará de nuevo. Esta se encuentra en $LOCALAPPDATA%\Unwanted\Canciones, donde %LOCALAPPDATA% es una ruta dentro del perfil del usuario, que se encuentra normalmente en C:\Users\<NombreUsuario>\AppData\Local. Aquí se almacenarán todos los archivos de audio que el jugador decida procesar, manteniendo el nombre del archivo. Junto a los archivos de audio se guardan otros ficheros de texto que son el resultado del análisis de la canción. Dichos archivos se guardan bajo el nombre de “<NombreCancion>_result.txt”. Gracias a esta base de datos podemos acceder en cualquier momento, en esta ejecución y en una posterior, a cualquier canción procesada, cumpliendo así con el RF7.

A continuación a esto se llama a DontDestroyOnLoad(). Este método viene definido en la clase MonoBehaviour y se utiliza para que el GameObject que contiene este *script* no sea destruido cuando se decida cargar la siguiente escena. Esto es necesario para que los sistemas de ritmo y sonido puedan obtener la referencia al “SongSelectorManager” cuando sea necesario.

Posteriormente a esto se llama al método LoadSongs(). Este método recorre toda la carpeta de canciones creada anteriormente buscando archivos de extensión .mp3 para los que exista también en la misma carpeta un archivo con el mismo nombre pero que acabe con “_result.txt”. De esta manera se detectan las canciones previamente analizadas y se muestran en pantalla la lista de canciones que cumplen con las condiciones anteriores, cumpliendo así con el RF8. Esta lista estará formada por un botón con el nombre de cada canción sin la extensión de archivo. El jugador podrá seleccionar una única canción, quedando dicho botón marcado con un color distinto y desmarcando el botón seleccionado anteriormente, como indica el RNF5 y como se puede ver en la Figura 22. Este método también almacena la ruta del archivo de audio y texto relacionados a la canción seleccionada. No existe límite

predefinido en cuanto a la cantidad de canciones procesadas que se pueden almacenar, más allá de los límites del sistema operativo. De esta forma cumplimos con el RNF3.

Figura 22: Canción "Highway to Hell – AC_DC" seleccionada

El método Update() se encarga de comprobar que la escena actual no corresponda con el menú principal, pues el objetivo del GameObject al que está asociada dicha clase es la de mantenerse vivo durante toda la partida. Una vez la partida acaba, regresando al menú principal, este GameObject tiene que ser destruido para evitar así errores varios con múltiples instancias del GameObject. El problema de las múltiples instancias podría haberse resuelto mediante el uso de un patrón *singleton,* pero igualmente sería necesario eliminar la instancia del *singleton* una vez finalizada la partida, pues para el modo por defecto se necesita que no exista este GameObject. De esta forma se consigue el mismo resultado pero con menor complejidad.

Dentro de la propia escena encontramos un botón "Nueva Canción", cuya principal función es cumplir el RF2. Utilizando la librería Unity Standalone File Browser [24] se abre una ventana del explorador de archivos de Windows que permite al usuario seleccionar la canción con la que desea jugar, como indica el RF1 y como se muestra en la Figura 23. El primer paso es comprobar que no exista una canción con el mismo nombre previamente almacenada en la base de datos. En caso de existir, no se procesaría la canción y se mostraría un mensaje de error en pantalla (Figura 25), indicando que ya se ha añadido con anterioridad dicha canción. Esto cumple con el RNF5. En caso de no existir previamente en la base de datos dicha canción, se copia y se procede a su análisis. Para analizar dicha canción se genera un proceso del sistema que ejecutará de manera asíncrona y en segundo plano el comando de terminal necesario. Esto se ejecutará sin crear una ventana de PowerShell, permitiendo la máxima transparencia de cara al usuario final, como marca el RF3, el RF5 y el RF6. La orden ejecutada será de la forma

".\essentia_streaming_beattracker_multifeature_mirex2013.exe destinationPath outputFilePath".

"essentia_streaming_beattracker_multifeature_mirex2013.exe" corresponde con el ejemplo de uso precompilado que facilita la propia documentación de la librería Essentia [22]. Existen múltiples ejemplos de uso de las funcionalidades de Essentia proporcionados por sus desarrolladores. Algunos de los ejemplos comprobados llegan

a generar archivos JSON de varios cientos de líneas con todo tipo de datos extraídos de la pista de audio. Sin embargo, la cantidad de información es innecesariamente grande para las necesidades del juego. La implementación actual se limita únicamente a generar un archivo txt con las marcas temporales de los golpes de ritmo, una por línea del archivo. La simplicidad del ejemplo ayuda a su eficiencia y a su fácil tratamiento, al disponer únicamente de la información que necesito para mi sistema. “destinationPath” y “outputFilePath” corresponden a las rutas de la canción a analizar y el archivo resultado respectivamente, siendo ambas relativas a la base de datos. “essentia_streaming_beattracker_multifeature_mirex2013.exe” se encuentra dentro de la jerarquía de archivos del proyecto dentro de la carpeta Assets\StreamingAssets. Esto se hace porque al compilar el proyecto, esta carpeta queda guardada en Unwanted\Unwanted_Data\StreamingAssets sin que Unity modifique nada del contenido de dicha carpeta durante la compilación. Así se puede acceder de manera correcta al ejecutable. El procesado de las canciones con este método es variable dependiendo de la duración y complejidad de la canción pero no supera los 2 minutos de tiempo, lo que cumple con el RNF1.

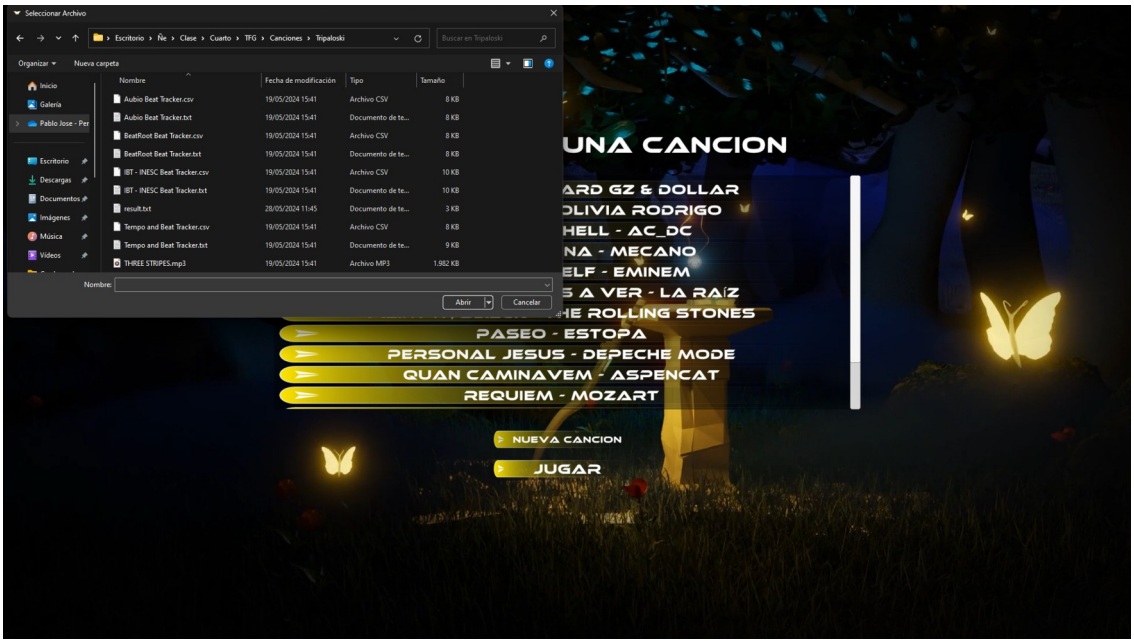


Figura 23: Explorador de archivos abierto para seleccionar una canción dentro del equipo

Figura 24: Análisis de una canción en proceso

Figura 25: Mensaje de error al intentar procesar una canción previamente analizada

Al ejecutarse de manera asíncrona este proceso no se bloquea la aplicación durante el procesado. Para indicar que se está procesando la canción se muestra un mensaje en la pantalla y se bloquean los botones de “Nueva Canción” y “Jugar”, pero el jugador aún puede navegar entre la lista de canciones y seleccionar cualquiera de estas. De esta manera se controlan posibles errores durante el procesado, evitando la sensación de que la aplicación se ha quedado bloqueada, tal y como indica el RNF5 y se muestra en la Figura 24. Se espera a que termine el proceso para poder a continuación devolver los mensajes de resultado y error por la consola de Unity, lo que ayuda al depurado y desarrollo. Una vez ejecutado el proceso se elimina el mensaje que indica que se estaba procesando, se desbloquean los botones y se

refresca la lista de canciones, lo que cumple con el RF4, permitiendo así al usuario utilizar dicha canción para jugar.

La última opción disponible es el botón de “Jugar”. Esta opción comprueba si se ha seleccionado previamente una canción de la lista. En caso negativo, se muestra un mensaje por pantalla (Figura 26), como indica el RNF5. En caso afirmativo, se hace la transición a la siguiente escena de juego, que corresponde con la cinemática inicial para proseguir con un desarrollo habitual del juego.

Figura 26: Mensaje de error al pulsar "Jugar" sin ninguna canción seleccionada

Una vez cargada la siguiente escena, es necesario realizar los ajustes necesarios por parte de los sistemas de ritmo y sonido para poder adaptarse a la elección realizada por el usuario.

Por la parte del sistema de ritmo, este está controlado desde el *script* “RythmController”, asignado al GameObject “Player”. Este, en su método Awake() busca en la escena al GameObject “SongSelector”. En caso de encontrarlo, llama a su método GetSelectedResult(), que devuelve la ruta en el sistema de archivos hacía el archivo de texto con el resultado del procesado de la canción. En caso de no encontrar el GameObject requerido quiere decir que se está realizando una ejecución en modo normal, por lo que se utiliza el *path* asignado por defecto.

En cuanto al sistema de ritmo propiamente dicho, este empieza con la llamada al método ProcessingSong() desde el Start() de la clase. Este método guarda en un *array* de *float* los valores leídos del archivo de texto, restando al valor en cuestión 1.45 y siempre y cuando el resultado sea mayor a cero. Durante el juego se va a estar comparando el instante actual de reproducción de la pista de audio de la canción con los valores almacenados en el array anterior. Si el tiempo de la canción coincide, con unos ciertos márgenes, con uno de los tiempos guardados, se lanza una rutina que genera la animación de las barras que se mueven desde los laterales hasta el centro de la pantalla. Dicha animación tarda exactamente 1.45 segundos en alcanzar el punto en pantalla en que se supone que hay un golpe de ritmo, por lo que se tiene

que lanzar 1.45 segundos antes de que ocurra. En el momento en que las barras coinciden con unas marcas fijas en pantalla, durante un corto lapso de tiempo se activa la *flag* que indica que el juego se encuentra dentro de un golpe de ritmo, permitiendo al resto de sistemas como el de disparo funcionar de manera normal. Esto cumple con los RF11 y RF12.

Por parte del sistema de sonido tenemos al *script* “SoundManager”, asignado al GameObject “SistemaSonido”. Este GameObject tiene a su vez varios GameObject hijos, correspondientes a cada una de las pistas de audio de *Somewhere in the night* junto a uno extra para la canción seleccionada por el jugador, teniendo a su vez cada uno un componente del tipo AudioSource. Esta clase gestiona de la misma forma la detección del GameObject “SongSelector” mediante el método Awake(), pero esta vez obteniendo la ruta a la canción con el método GetSelectedSong(). En este caso, a parte de obtener la ruta se desactivan los GameObject referentes a la canción por defecto, mientras que si no se encuentra “SongSelector” se desactiva el componente que contiene la referencia a la canción seleccionada. Para poder utilizar un archivo de audio seleccionado en tiempo de ejecución y asignado de manera dinámica es necesario hacer uso de las UnityWebRequest, siendo en este caso una petición del tipo multimedia al sistema local de archivos del ordenador. De esta manera se asigna el clip de audio al AudioSource de manera correcta. El resto de la clase se encarga de reproducir y pausar el sonido a voluntad. Esto cumple con los RF10 y RF12.

4.3 Tecnología utilizada

Para poder llevar a cabo este proyecto han sido necesarias múltiples tecnologías. Todas estas herramientas son la base del proyecto o han ayudado a su desarrollo y evaluación.

4.3.1 Unity

Figura 27: Logo de Unity

Unity (Figura 27) [25], lanzado al mercado en 2005 por Unity Technologies, es el motor de videojuegos sobre el que está construido este proyecto. Se trata del motor más utilizado en la industria, siendo la elección predilecta para la mayoría de desarrollos de la escena independiente, pero a su vez teniendo una importante representación dentro de los grandes desarrollos. Esto es así gracias a las políticas de Unity que permiten a cualquier desarrollador pequeño publicar juegos usando este motor sin coste. De la misma forma, las bondades de Unity lo han convertido en un servicio que escapa del mundo de los videojuegos, llegando incluso a la industria del cine.

Uno de sus puntos fuertes es que es fácil de utilizar y aprender, a la vez que facilita el trabajo para todo aquel desarrollador que no disponga de un motor propio y no quiera invertir en crear uno. Unity incorpora un motor de físicas tanto en 2D como en 3D que permiten al usuario crear todo tipo de juegos, junto a un motor de animaciones, de iluminación y de sonido entre otros. Todo esto es fácilmente configurable mediante el editor visual que incorpora, siendo este muy intuitivo y fácil de aprender. Unity es compatible con otras tecnologías de amplia adopción en el mercado, tales como modelados 3D almacenados en distintos formatos. Su lenguaje de *scripting* es C#, siendo este un lenguaje moderno de amplia adopción al que Unity incorpora funciones propias que permiten trabajar de forma muy sencilla.

El editor de Unity es multiplataforma, estando disponible en Windows, Linux y MacOS. A su vez, permite desarrollar juegos para plataformas móviles como Android y de videoconsolas como PlayStation o Xbox junto a las ya mencionadas plataformas de escritorio.

Unity cuenta con una inmensa comunidad organizada en foros propios y en general en internet. Esto facilita encontrar tutoriales de todo tipo. De la misma forma, Unity proporciona su Asset Store, lugar en que cualquier usuario puede publicar contenido para que cualquier otro desarrollador pueda descargarse gratis o comprar y utilizar en sus propios proyectos de manera cómoda.

Por todo esto considero que Unity es la mejor herramienta disponible para introducirse en el mundo del desarrollo de videojuegos y que a su vez permite a cualquiera continuar utilizando esta herramienta a nivel profesional con garantías de calidad.

4.3.2 GitHub

Figura 28: Logo de GitHub

GitHub (Figura 28) [26] es una herramienta imprescindible para el control de versiones en el desarrollo de software. Se trata de un servicio que actúa como repositorio remoto de código de manera gratuita. Cualquier usuario puede crear su propio proyecto en GitHub y subir el código. Cada vez que el desarrollador quiere publicar una nueva versión del código puede subir al repositorio dicho código, el cual queda almacenado remotamente. Cada nueva publicación viene acompañada de una lista de cambios que lo identifican. Desde cualquier dispositivo se puede acceder a dicho repositorio y descargarse la última versión fácilmente, permitiéndote así guardar una copia segura del trabajo fácilmente migrable entre equipos.

GItHub permite en todo momento consultar versiones anteriores del proyecto, compararlas con el actual y deshacer cambios en caso de ser necesario. De la misma forma, GitHub permite llevar ramas de desarrollo en paralelo con versiones diferentes del proyecto para la cual se puede estar trabajando en apartados distintos. Estas ramas se pueden combinar para lanzar versiones definitivas.

Una de las grandes bondades de GitHub es compartir el repositorio con varios desarrolladores. Cualquiera de estos puede comprobar el estado del trabajo de sus compañeros y descargar sus cambios. Esto facilita el trabajo en equipo. Debido a que Unwanted fue desarrollado originalmente por seis desarrolladores distintos, tanto de la parte artística como de programación, esto facilitó enormemente la comunicación entre los miembros. De la misma forma, el desarrollo ha podido continuar también en equipo, por lo que en todo momento ha sido imprescindible contar con una tecnología como esta.

4.3.3 Essentia

Figura 29: Logo de Essentia

Essentia (Figura 29) [19][20] es una librería de código abierto para C++ y Python que permite analizar audio, en especial música. Essentia ha cumplido con un rol fundamental en el desarrollo del proyecto, pues esta librería cuenta con algoritmos capaces de detectar ritmo en canciones, aunque la colección de algoritmos disponibles es mucho más amplia.

Essentia ha sido utilizado en multitud de proyectos. Ejemplos son webs como el repositorio Freesound.org [27] hasta juegos como *Crypt of the NecrodDancer* [9], y otros que podemos encontrar listados en su web [21].

Essentia cuenta con multitud de ejemplos de uso, tanto en código como ejecutables en web como ejecutables en escritorio. Uno de estos ejemplos de uso ha sido utilizado para obtener el ritmo de las canciones.

4.3.4 Audacity

Figura 30: Lodo de Audacity

Audacity (Figura 31) [11] es una herramienta de grabación y edición de audio de código libre. Es intuitiva y fácil de utilizar y aprender. Gracias a ser de código libre cuenta con una gran comunidad que facilita el proceso de aprendizaje gracias a tutoriales de todo tipo.

Esta herramienta cuenta con gran cantidad de *plugins* que amplían sus funciones básicas. De entre toda la selección de *plugins* disponibles destacamos Vamp Plugins que, como comentamos anteriormente, son un conjunto de *plugins* desarrollados por instituciones como la Queen Mary University of London para el análisis de canciones. Dentro de estos *plugins* para el proyecto se utiliza un pack que viene ya configurado y de fácil instalación y que cuenta con cuatro sistemas de detección de ritmo distintos. Estos sistemas generan etiquetas con las marcas de tiempo de los golpes de ritmo. Para poder comprobar la precisión del sistema de Essentia se han utilizado estos *plugins* como referencia.

4.3.5 Trello

Figura 31: Logo de Trello

Trello (Figura 32) [28] es una herramienta web de administración de proyectos. Permite organizar un proyecto en tableros configurables al estilo de los tableros Kanban dentro de las metodologías ágiles de desarrollo de software. Estos tableros están divididos en columnas agrupadas por funcionalidades de la aplicación. Junto a

estas se crean tres columnas más, que representan el trabajo a realizar a corto plazo, el que se está realizando y el trabajo ya terminado. Dependiendo de la metodología aplicada se pueden añadir nuevas columnas para el proceso de desarrollo. En cada columna se pueden añadir nuevas tarjetas referentes a tareas a realizar dentro del desarrollo. Estas tarjetas son configurables, pudiendo distinguirse por colores, añadir descripciones detalladas de las tareas y asignar dichas tareas a desarrolladores en específico. Las tarjetas se pueden mover entre columnas conforme vayan pasando por el flujo de trabajo.

Esta herramienta ayuda a organizar el trabajo a realizar, agrupado por áreas diferenciadas, a la vez que permite tomar mejor decisiones sobre qué funcionalidades empezar a desarrollar y ayuda a controlar qué se ha hecho, qué se está haciendo y qué hay por hacer. Tanto en el desarrollo inicial de Unwanted (Figura 32) como en la ampliación para este TFG (Figura 33) se ha utilizado Trello como herramienta para mantener el proyecto organizado y controlado. De la misma forma, el proceso de desarrollo se ha llevado a cabo mediante sprints de dos semanas con reuniones en cada entrega. Trello ha ayudado a planificar dichos sprints y tener un control sobre lo que había que hacer en todo momento.

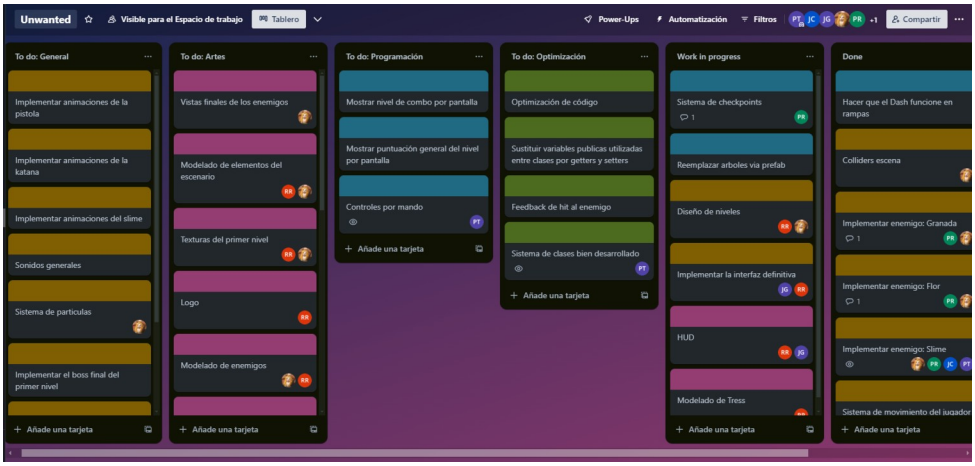


Figura 32: Estado en desarrollo del tablero Kanban de Unwanted

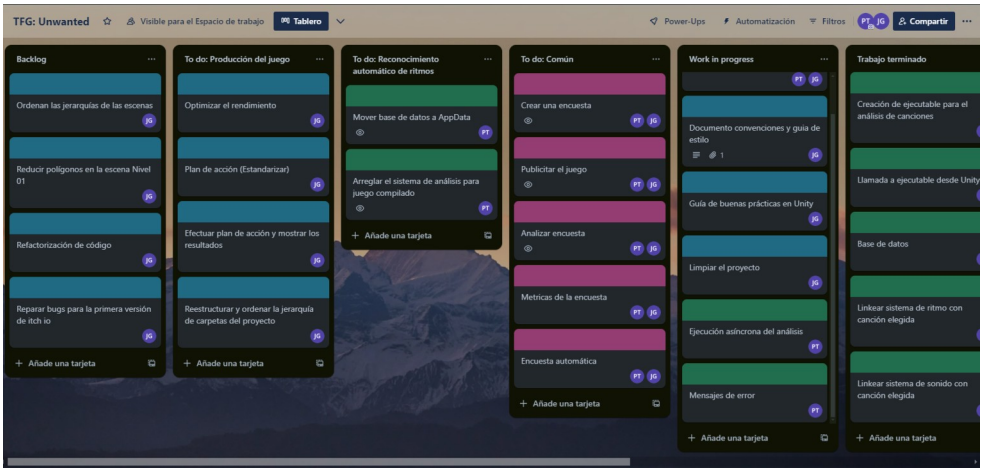


Figura 33: Estado en desarrollo del tablero Kanban de ambos TFG de ampliación de Unwanted

4.3.6 Itch.io

Figura 34: Logo de Itch.io

Itch.io (Figura 34) [29] es una página web de distribución de videojuegos y *assets*. Está centrada en la distribución de juegos independientes. Cualquier usuario puede publicar de manera gratuita su propio videojuego o contenido para crear videojuegos. Los usuarios pueden descargar estos juegos o contenido de manera gratuita, con opción de donar al creador, aunque también hay contenido de pago. Itch.io es un servicio para la distribución de juegos independientes que a su vez sirve de *portfolio* para desarrolladores. De la misma manera, es fácil encontrar *assets* de calidad para poder utilizar en tus propios proyectos.

Unwanted se ha publicado en Itch.io, tanto con la intención de que cualquiera persona pueda descubrirlo y jugarlo sin coste, como para dar a conocer el proyecto y así conseguir que los jugadores puedan rellenar una encuesta que acompaña a la publicación y que se presentará en el apartado 5.2 Evaluación externa. En la Figura 35 se puede ver la página de Itch.io del proyecto [30].

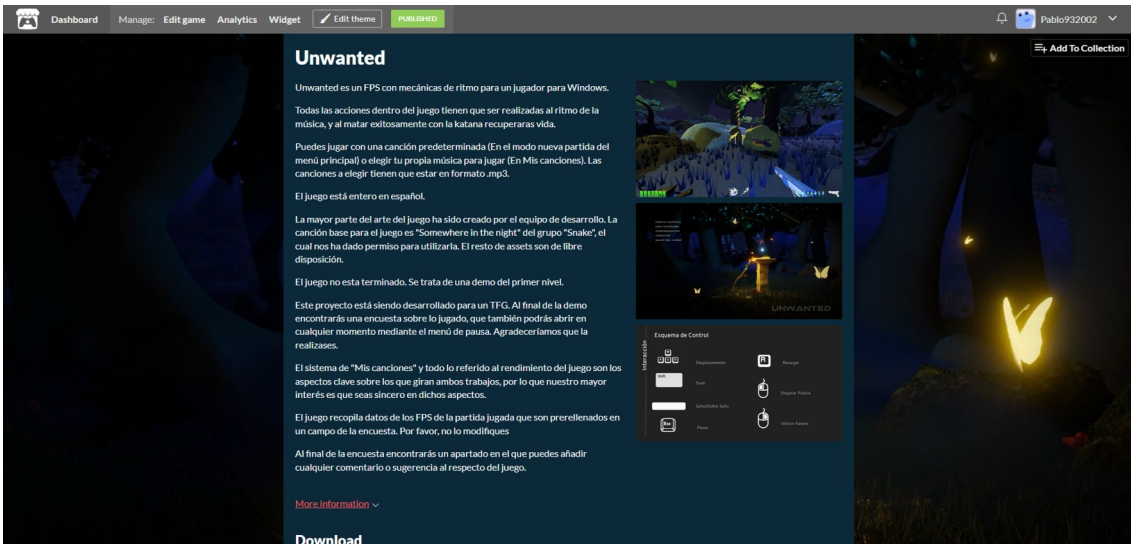


Figura 35: Página de Unwanted en Itch.io

5 Evaluación

Para evaluar el resultado del desarrollo se han seguido dos procesos distintos. El primero de ellos consiste en una comparación entre el sistema de detección de ritmo de Essentia con los sistemas incorporados en los Vamp Plugins de Audacity. El segundo consiste en una valoración general por parte de los usuarios del juego mediante una encuesta.

5.1 Evaluación interna

La primera parte del proceso de evaluación del desarrollo corresponde al análisis que se realizó al principio del desarrollo de este TFG para evaluar la precisión del sistema de detección de ritmo de la librería seleccionada. Para este análisis se han seleccionado 15 canciones de géneros diferentes que se han analizado tanto con el sistema de detección de ritmo de Essentia como con los cuatro detectores de ritmo incorporados en el *pack estándar* de Vamp Plugins de Audacity. Estas canciones pertenecen a géneros diferentes como pop, rap, trap, reggaeton, rock, hard o música clásica. Se entiende que no todos los géneros musicales funcionan igual de bien en este tipo de juegos pero se ha buscado trabajar con una gran variedad de géneros para comprobar los resultados de la implementación del sistema de detección de ritmo.

Se ha analizado cada canción de dos formas distintas. La primera es un análisis visual. Los archivos de texto generados por Essentia se han importado a Audacity y se han comparado de manera visual con las etiquetas generadas por los distintos *plugins*. Se ha reproducido la canción en cuestión y, con la ayuda de un músico experimentado se ha determinado cuál de las cinco opciones ha seguido mejor el ritmo y cómo de correcta es la interpretación de Essentia. Después de esta comparativa se ha procedido con un segundo análisis, que corresponde a una prueba de juego, en la cual se trata de jugar a Unwanted utilizando cada canción en concreto, valorándose la jugabilidad del juego utilizando dicha canción.

Lista de canciones:

- CAE LA NOCHE - HARD GZ & DOLLAR
- get him back - Olivia Rodrigo
- Highway to Hell – AC_DC
- Hijo de la Luna – Mecano
- Lose Yourself – Eminem
- Nos Volveremos a Ver - La Raíz
- Paint It, Black - The Rolling Stones
- Paseo – Estopa
- Personal Jesus - Depeche Mode
- Quan caminàvem – Aspencat
- Requiem – Mozart

- She Don't Give a FO – DUKI
- Tití Me Preguntó - Bad Bunny
- Trip to Ireland - Dr. Peacock
- ТРИ ПОЛОСКИ _ KOLM TRIIPU _ THREE STRIPES

Lista de algoritmos de Audacity:

- Aubio Beat Tracker: Beats
- IBT - INESC Beat Tracker
- Tempo and Beat Tracker: Beats (Queen Mary, University of London)
- BeatRoot Beat Tracker: Beats (Simon Dixon (plugin by Chris Cannam))

CAE LA NOCHE - HARD GZ & DOLLAR

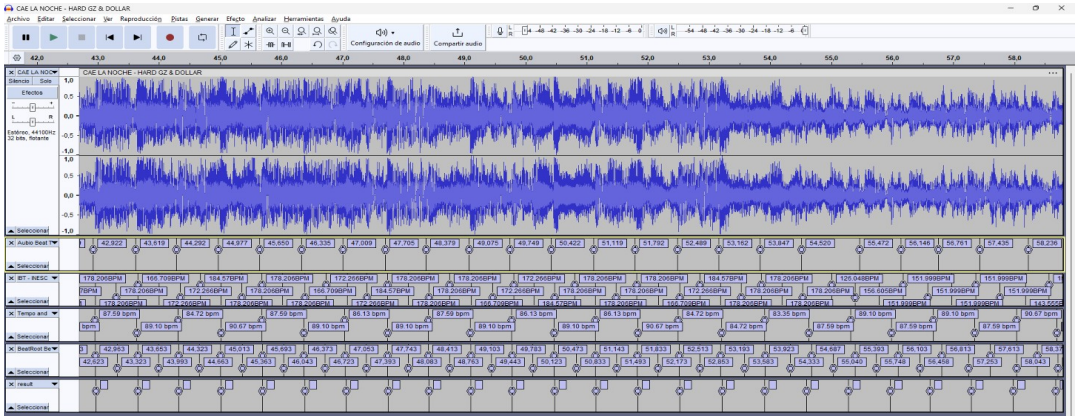


Figura 36: Onda de sonido de "CAE LA NOCHE - HARD GZ & DOLLAR"

El procedimiento para analizar cada canción será el mismo, por lo que se explicará utilizando esta canción como ejemplo y en el resto se procederá directamente con la conclusiones. La primera fila corresponde con la onda de sonido (Figura 36) de la canción en cuestión. Las cinco filas inferiores corresponden con las etiquetas generadas por los cuatro algoritmos en el orden en el que aparecen en la lista anterior y el último de ellos corresponde con las etiquetas generadas por Essentia e importadas a Audacity. Al reproducir la canción y con la ayuda de un músico se ha ido marcando el ritmo de manera manual. En este caso se valorará cuál de los algoritmos de Audacity es el más preciso en comparación con el obtenido manualmente y cómo de preciso es el resultado de Essentia en comparación con el más preciso de Audacity.

En el caso de esta canción se considera que todos los algoritmos más o menos aciertan, ya que esta canción tiene el ritmo muy marcado al tratarse de una canción de rap. En este caso consideramos que Aubio es el que peor lo hace en general y que el algoritmo de BeatRoot es el más cercano a la realidad.

En cuanto a la comparativa con Essentia, en esta canción las etiquetas de ritmo generadas son bastante precisas aunque en algunos puntos de la canción podemos encontrar ligeras diferencias. El principal problema encontrado es que solo se detecta la mitad de golpes de ritmo como tal, al igual que el algoritmo de la Queen Mary. En general se considera que es preciso y podría ser usable para el juego.

Un ligero error que aparece en esta canción y que es bastante habitual ocurre al inicio, ya que la introducción a la canción no cuenta con la misma instrumental que el resto de la canción y en algunos casos hay momentos de silencio. Algunos de los algoritmos en estas zonas de la canción “se inventan” las marcas temporales, probablemente mediante métodos estadísticos que generan estas marcas por repetición de lo que ocurre más adelante en la canción. Esto no suele ser un problema porque no suele durar mucho pero hace que los ritmos marcados al inicio de algunas canciones no tengan sentido. En esta canción esa parte es más molesta al tener 20 segundos diferentes al resto.

En cuanto a las pruebas de juego, las sensaciones son mixtas. El inicio lento hace que veamos zonas en las que se marcan ritmos que no se reconocen a nivel auditivo, aunque el hecho de existir y seguir un patrón similar al resto de la canción ayuda a habituarse. El problema de detectar solo la mitad de ritmos hace que el juego sea un poco lento. La parte detectada es buena pero se siente que falta algo. Si tuviéramos todos los golpes de ritmo detectados la experiencia sería perfecta.

get him back - Olivia Rodrigo

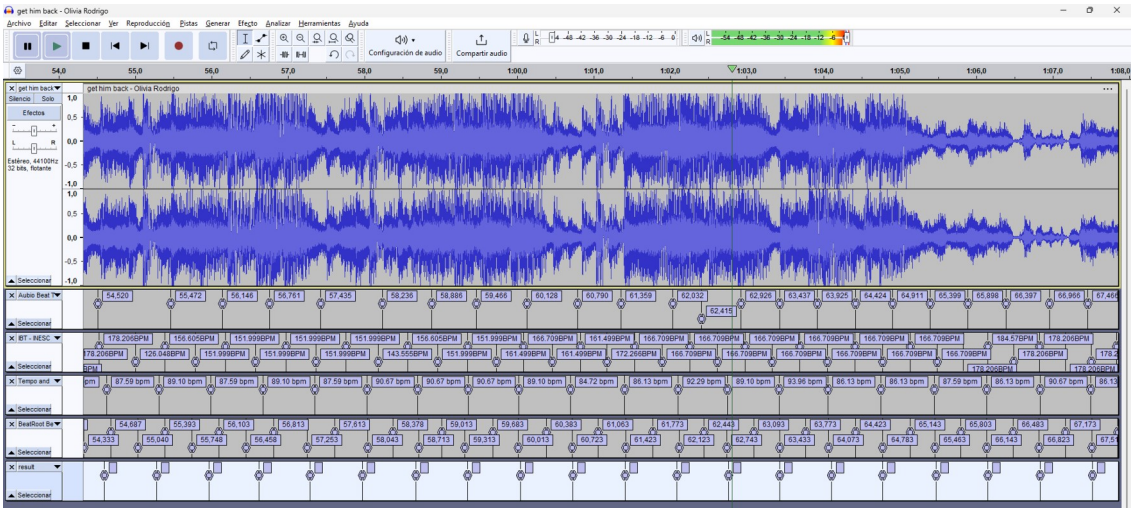


Figura 37: Onda de sonido de “get him back - Olivia Rodrigo”

En el caso de esta canción, prácticamente todos los algoritmos funcionan de manera más o menos correcta. La canción es de género pop-rock con golpes de batería muy marcados que hacen que sea relativamente sencillo. Como norma general Queen Mary es el que más se acerca a la precisión perfecta.

Al ser una canción identificable de manera más o menos sencilla, Essentia lo hace bastante bien en pruebas en Audacity, dando prácticamente el mismo resultado que el algoritmo de Queen Mary.

Las pruebas de juego son perfectas. El sistema se ajusta de manera muy precisa a la realidad y la canción elegida, tanto por velocidad como por estilo, encaja a la perfección con el juego.

Highway to Hell – AC_DC

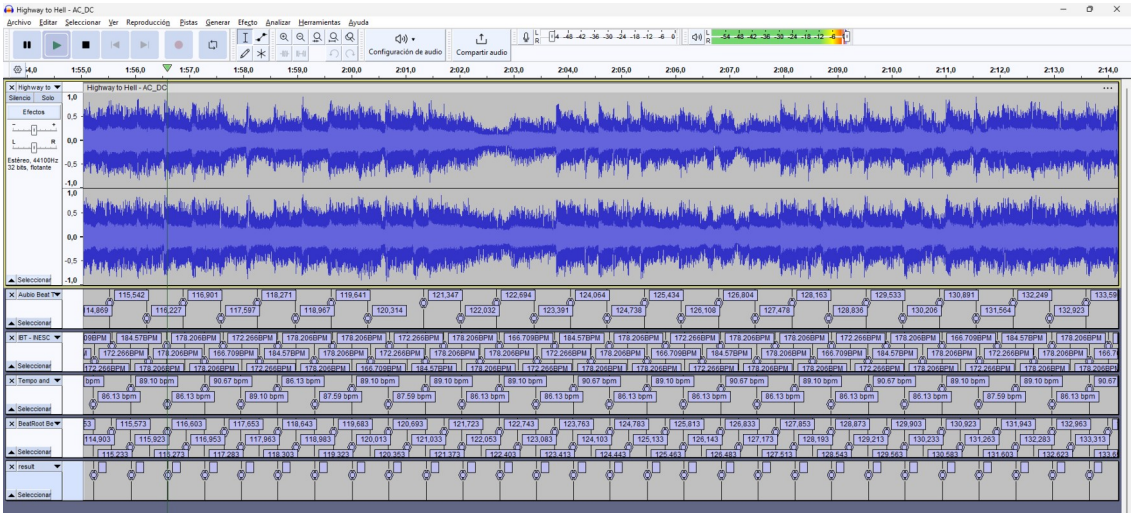


Figura 38: Onda de sonido de "Highway to Hell – AC_DC"

Otra canción de rock con el ritmo muy marcado y para la cual todos los algoritmos aciertan con una gran precisión. El resultado de Essentia es idéntico a los mostrados y de una gran precisión.

Esta canción comparte el problema del inicio con instrumental diferente pero la duración es menor que en otros casos.

Las pruebas en el juego son perfectas. Se ajusta a lo esperado.

Hijo de la Luna – Mecano

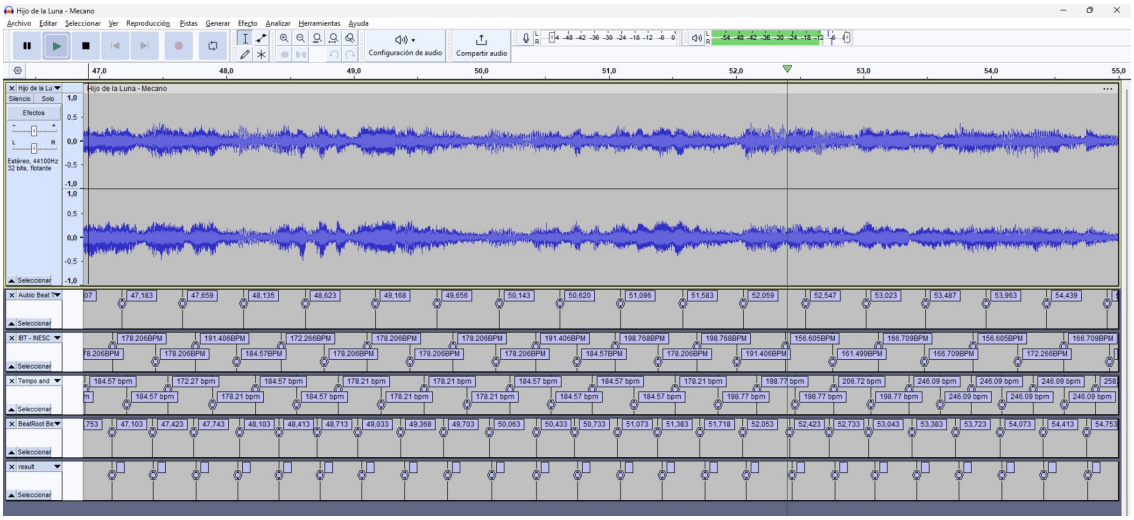


Figura 39: Onda de sonido de "Hijo de la Luna – Mecano"

La canción más difícil de analizar de todas las seleccionadas. Es la única que utiliza un compás compuesto de 6/8 en vez del clásico 4/4. Esto la hace muy rápida para ser utilizada en el juego, pero se ha incorporado para probar canciones muy diferentes. El algoritmo que más se acerca a la realidad es el de IBT. Los demás generan interpretaciones muy similares pero se descuadran en algunos momentos.

La interpretación de Essentia es similar a los resultados obtenidos por los demás algoritmos pero difiere ligeramente en algunos puntos.

Esta canción no está pensada para ser utilizada de esta forma y las pruebas en juego lo demuestran. La precisión es relativamente aceptable pero el hecho de seguir una estructura poco convencional la hace muy difícil de interpretar únicamente con el audio. Además el ritmo es demasiado rápido para lo esperado. El resultado al final es malo pero porque no es una canción buena para este juego en concreto y no tanto por el análisis de la librería.

Lose Yourself – Eminem

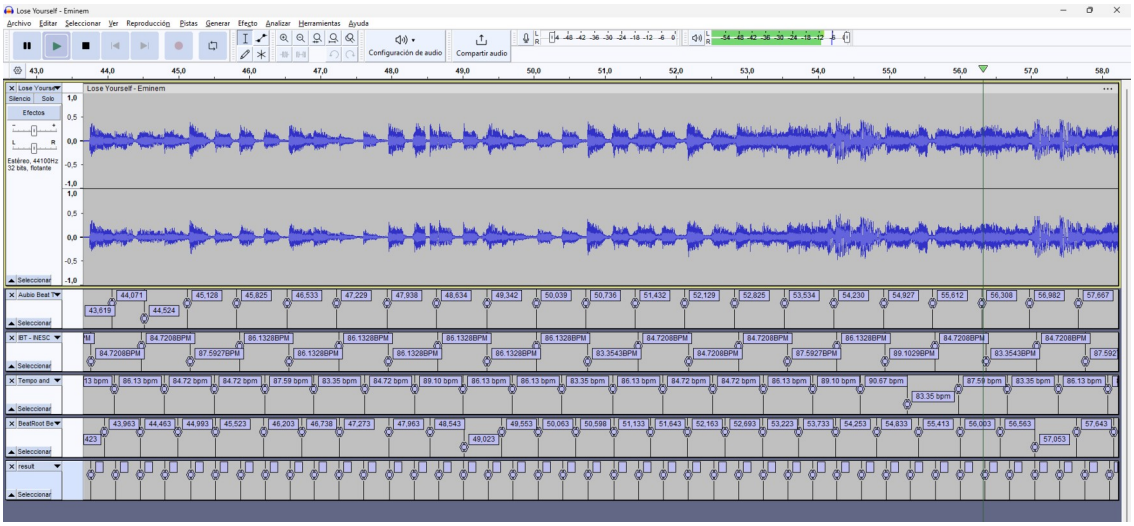


Figura 40: Onda de sonido de "Lose Yourself – Eminem"

Una de las sorpresas de este análisis. Lose yourself es uno de los clásicos del rap, género que destaca por tener unos ritmos muy marcados. Sin embargo y por algún motivo, solamente el algoritmo de IBT es capaz de detectar el ritmo correctamente. Mientras que este algoritmo roza la perfección, el resto se equivocan de manera estrepitosa. El algoritmo de Essentia es incluso el que peor lo hace. Se inventa demasiada información. Esto hace que las pruebas en juego sean desastrosas para una canción que se esperaba que diera buenos resultados en la teoría. En algunos puntos podemos encontrar zonas con buenos resultados pero al contar con demasiada información extra acaba generando mucho caos.

Esta canción tiene uno de los problemas más graves de inicio lento que hace que durante casi un minuto los resultados mostrados por los algoritmos sean inconsistentes.

Nos Volveremos a Ver - La Raíz

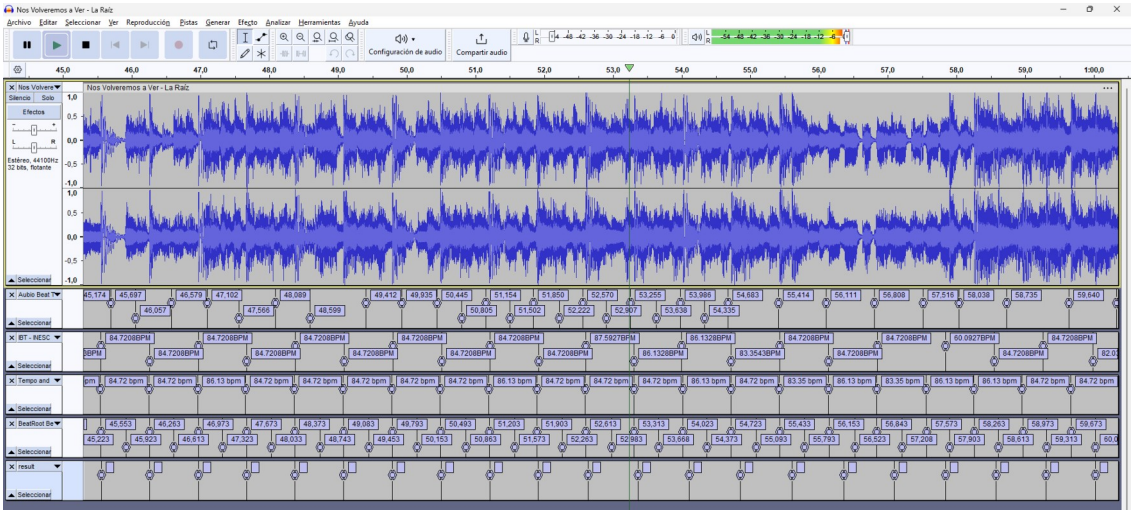


Figura 41: Onda de sonido de "Nos Volveremos a Ver - La Raíz"

Una canción especialmente difícil de analizar debido a todos los cambios de instrumental que tiene, lo que hace que en algunos segmentos de la canción un algoritmo pueda ir bien y en otros lo haga mal, por lo que es difícil establecer un ganador. En general la propuesta de Essentia es bastante aceptable.

Las pruebas en juego demuestran una correcta interpretación relativamente consistente y con una buena jugabilidad. El mayor problema deriva de la ligera lentitud del ritmo, provocada por la nula detección de los golpes intermedios. A esto hay que sumar el inicio extraño que provoca algunos conflictos.

Paint It, Black - The Rolling Stones

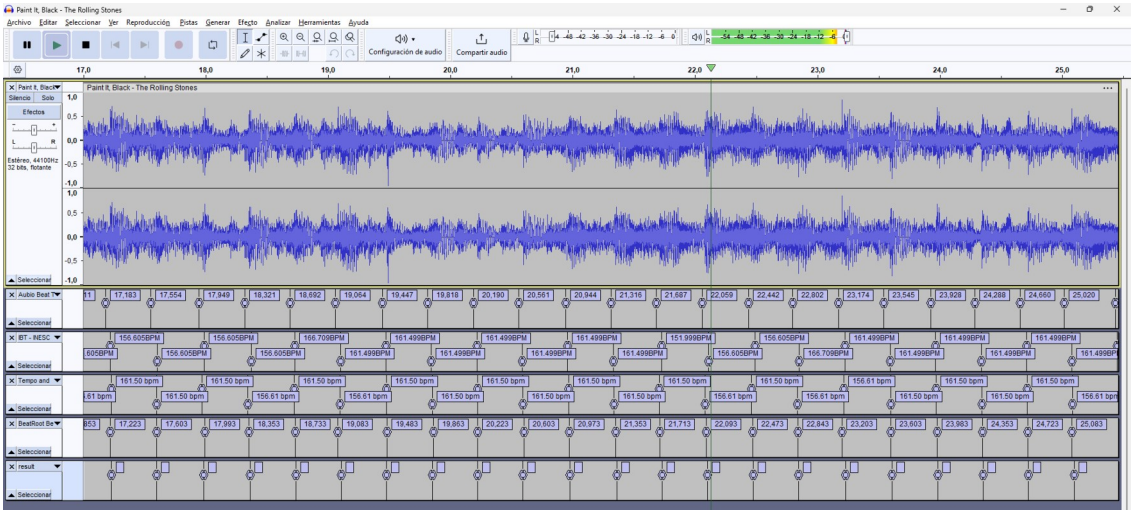


Figura 42: Onda de sonido de "Paint It, Black - The Rolling Stones"

La canción perfecta. Todos los algoritmos lo hacen bien con sus interpretaciones menos el de Aubio que se adelanta ligeramente. Esto es gracias a ser una canción de rock con un ritmo muy marcado.

Las pruebas en juego son muy buenas. Otra canción que se ajusta a la perfección a lo esperado.

Paseo – Estopa

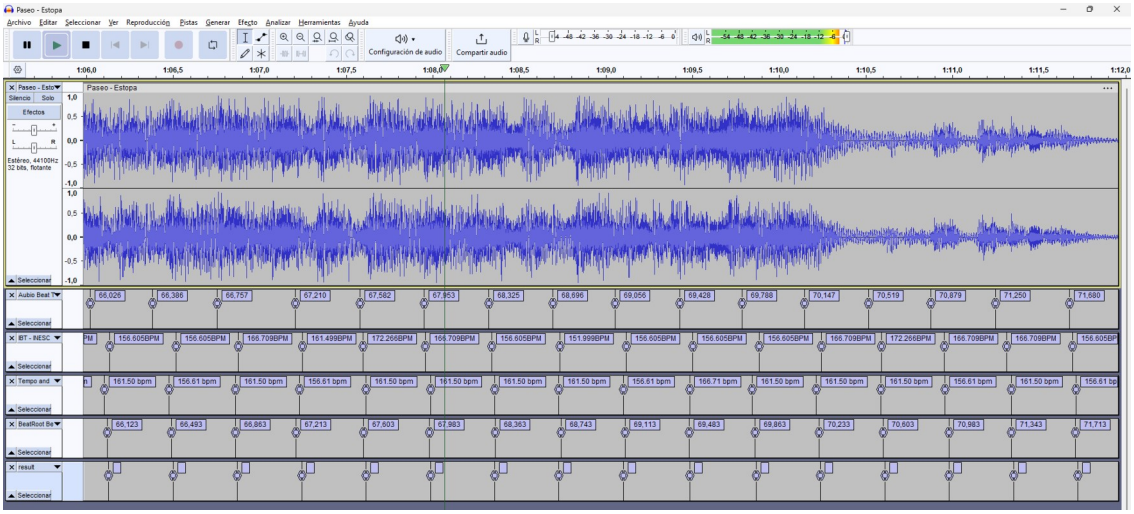


Figura 43: Onda de sonido de "Paseo – Estopa"

Otra canción con buenos resultados. La batería sirve mucho para marcar el ritmo, lo que hace que prácticamente los cinco algoritmos lo hagan bien.

Como era de esperar por los resultados obtenidos en Audacity, la canción funciona muy bien en el juego. El problema vuelve a ser que la canción es un poco lenta pero aceptable.

Personal Jesus - Depeche Mode

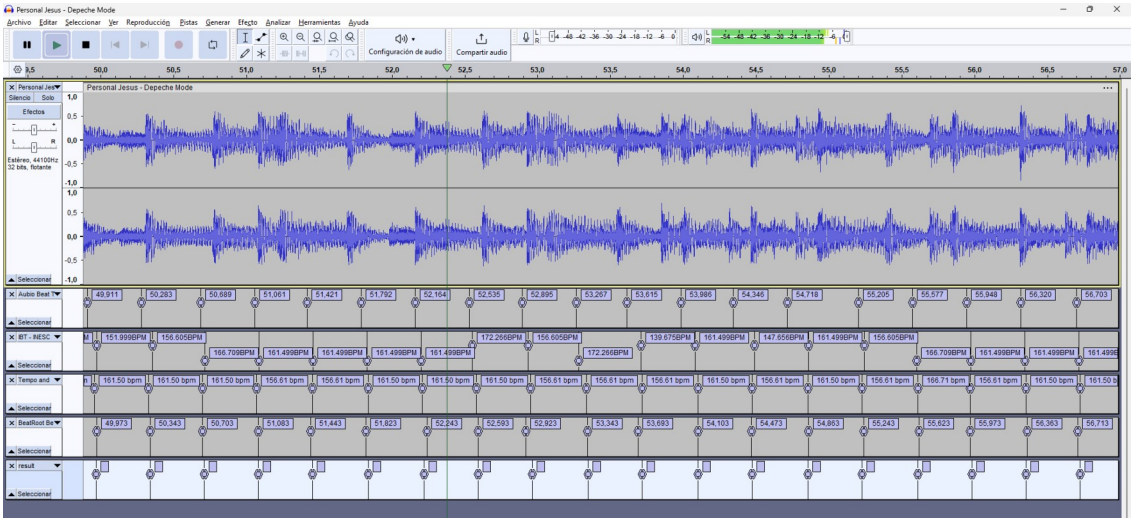


Figura 44: Onda de sonido de "Personal Jesus - Depeche Mode"

Otro resultado muy bueno. La música electrónica suele destacar por ritmos muy marcados, por lo que los cinco algoritmos cumplen su función a la perfección.

El ejemplo en juego también es muy bueno, como cabía esperar, aunque a veces el ritmo cuenta con un ligero retraso que es especialmente perceptible en canciones rápidas. De la misma forma, el inicio diferente de la canción hace que la detección sea regular en este punto.

Quan caminàvem – Aspencat

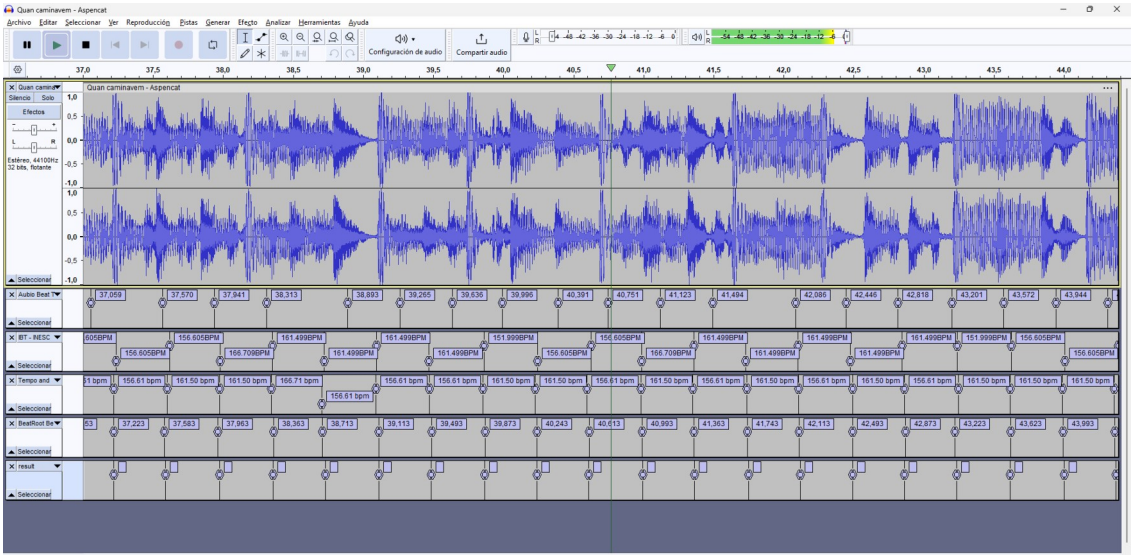


Figura 45: Onda de sonido de "Quan caminàvem – Aspencat"

Quitando del inicio diferente, la canción tiene un ritmo muy marcado que hace que los algoritmos lo hagan de manera bastante correcta, incluido el de Essentia. Sin embargo, este último parece que en algunos puntos se retrasa ligeramente. Aubio es el único que funciona de manera un poco extraña, sin ajustarse especialmente al ritmo

Las pruebas en juego son regulares debido a los retrasos y al inicio lento de la canción, lo que no la hace especialmente buena para jugar con ella

Requiem – Mozart

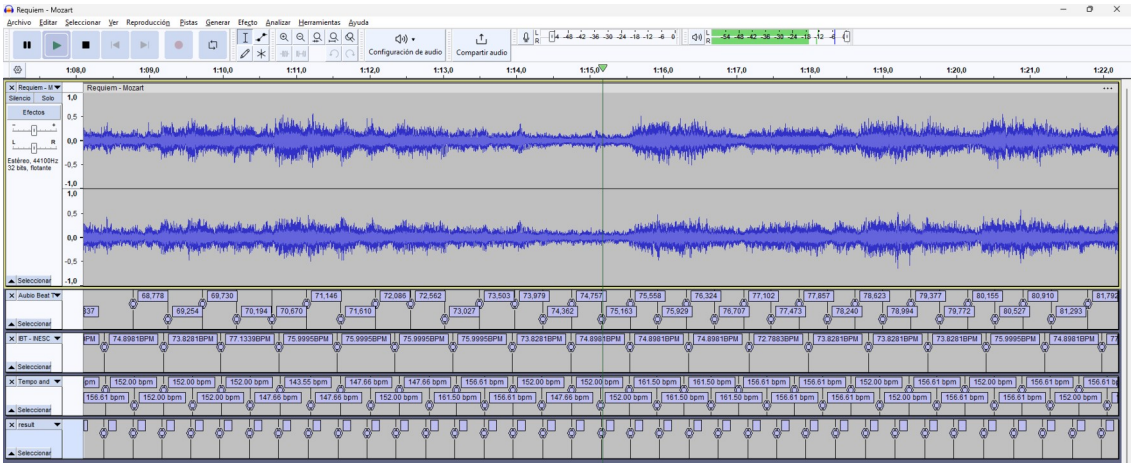


Figura 46: Onda de sonido de "Requiem – Mozart"

Uno de los casos más interesantes. Se trata de una canción con la que no se espera que la gente vaya a jugar pero que igualmente es interesante de comentar con la intención de abarcar más géneros musicales como es la música clásica.

En este caso, los resultados son muy dispares. BeatRoot directamente falla al analizar la canción y no devuelve ningún resultado. Aubio lo hace bastante mal y su análisis no tiene ningún tipo de sentido. Tempo se equivoca ligeramente en algunos puntos. IBT es el más preciso pero ignora golpes intermedios. Essentia devuelve un

resultado bastante parecido a IBT pero con esos golpes intermedios que pueden ser interesantes para el juego.

Definitivamente no es una canción con la que se espera que nadie decida jugar a un juego de estas características pero las pruebas en juego han sido interesantes. Es una pieza muy variable y con un inicio especialmente lento, lo que afecta a la jugabilidad. Sin embargo el ritmo es bastante constante y los golpes intermedios que detecta dan un resultado bastante bueno en cuanto a velocidad. Las canciones de rock dan mejores resultados pero creo que se podría jugar perfectamente con una canción así.

She Don't Give a FO – DUKI

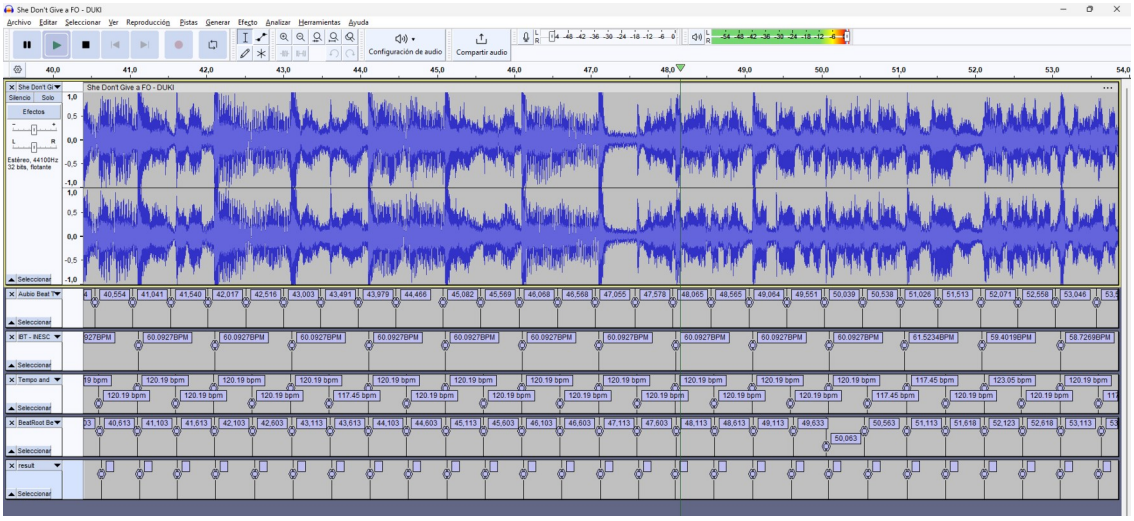


Figura 47: Onda de sonido de "She Don't Give a FO – DUKI"

El trap destaca por tener también ritmos muy marcados y sin embargo los resultados son ligeramente dispares. Los algoritmos de Queen Mary y BeatRoot son precisos y correctos y muy similares a los resultados devueltos por Essentia. Aubio se equivoca bastante e IBT pierde los golpes intermedios y se descuadra.

Las pruebas de juego son muy buenas, lo que demuestra que el trap puede ser otro género musical que encaja bastante con este juego. Como mucho la canción podría ser ligeramente más rápida para encajar mejor aunque ya funciona bastante bien.

Tití Me Preguntó - Bad Bunny

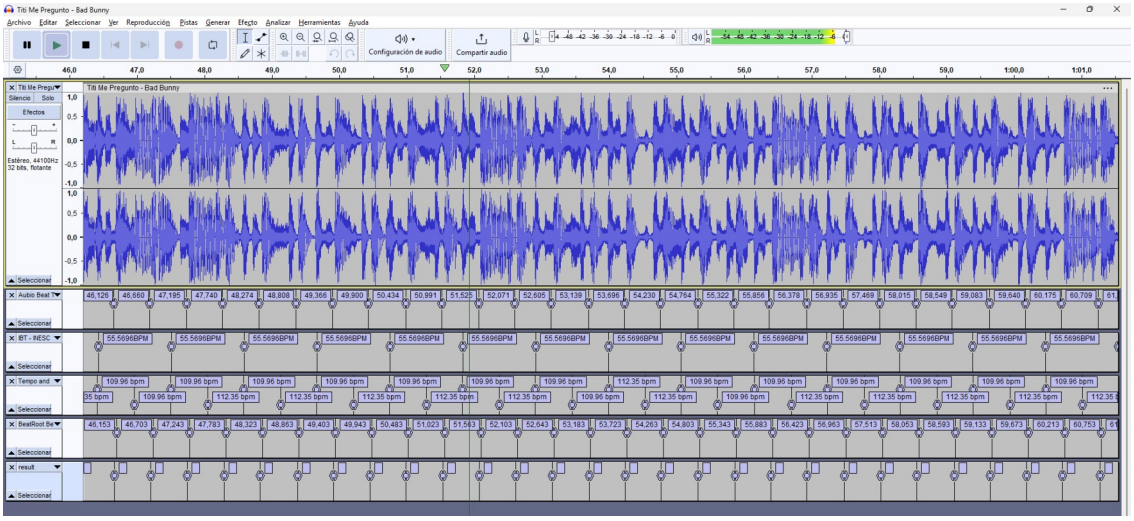


Figura 48: Onda de sonido de "Tití Me Preguntó - Bad Bunny"

BeatRoot, Aubio y Essentia son los únicos que reconocen bien el ritmo de la canción. Tanto IBT como Queen Mary se equivocan bastante. El reggaeton tiene un ritmo bastante marcado que lo hace sencillo de analizar, lo que me hace entender los resultados de los tres primeros algoritmos pero no el de los otros dos.

Esta canción funciona también bien en las pruebas en juego, lo que hace pensar que el reggaeton puede ser otro género que se adapta bien al juego, aunque esta canción también es ligeramente más lenta de lo que me gustaría.

Trip to Ireland - Dr. Peacock

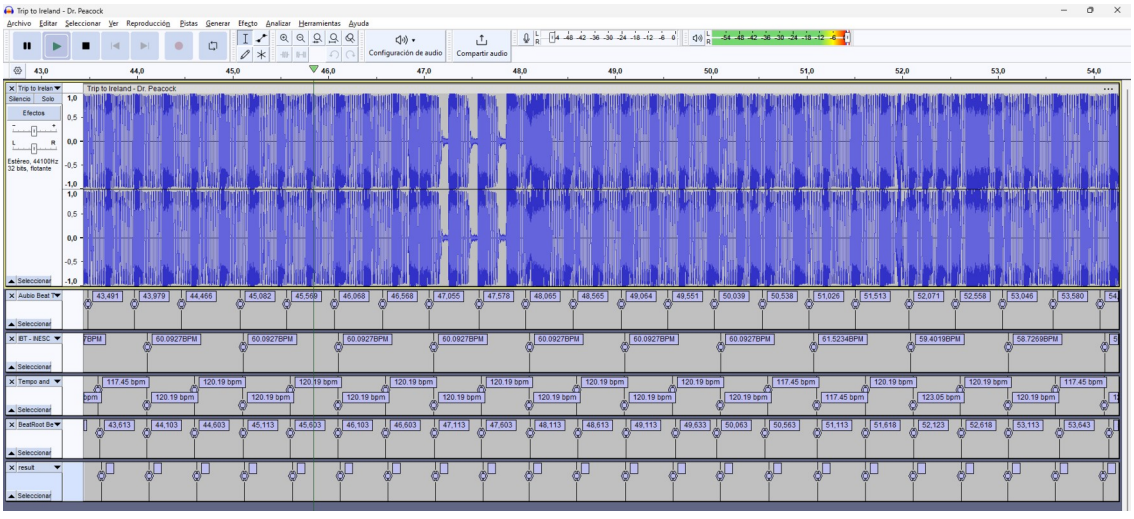


Figura 49: Onda de sonido de "Trip to Ireland - Dr. Peacock"

El hardcore techno destaca por tener golpes de ritmo muy marcados y rápidos. Quitando Aubio, los otros cuatro algoritmos dan buenos resultados, incluyendo Essentia.

Las pruebas en juego sin embargo son desastrosas. Este género musical se caracteriza por ser muy caótico con golpes muy constantes, lo que lo hace imposible de seguir en juego. Viendo ejemplos de otros jugadores con canciones similares, parece que el género no es bueno para este tipo de videojuegos, pues en general las

experiencias no han sido buenas.

ТРИ ПОЛОСКИ _ KOLM TRIIPU _ THREE STRIPES

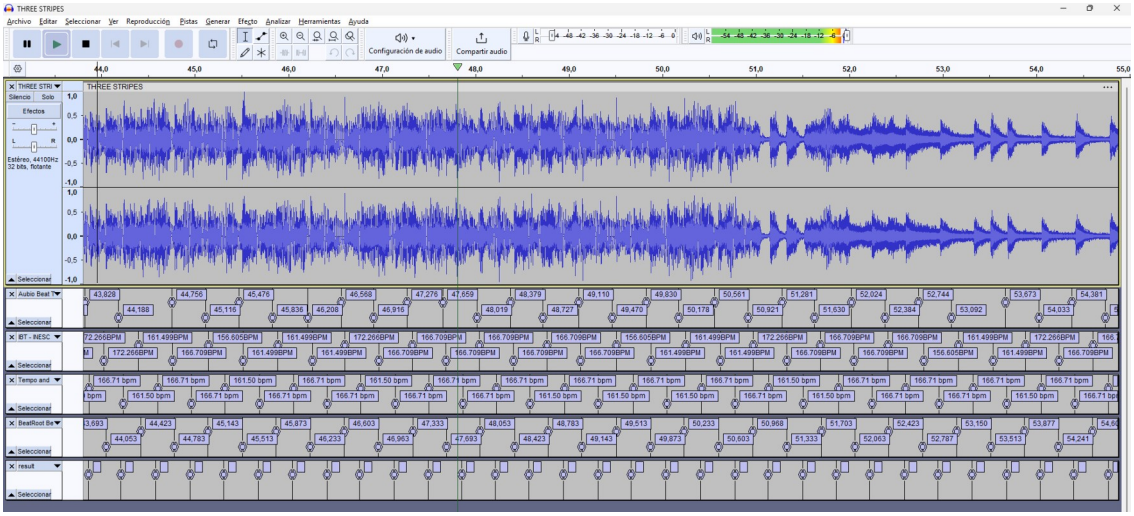


Figura 50: Onda de sonido de "ТРИ ПОЛОСКИ _ KOLM TRIIPU _ THREE STRIPES"

Esta canción pertenece a un género de música electrónica conocido como hard bass, que se caracteriza por unos golpes de ritmo muy constantes. Tanto el algoritmo de Queen Mary como el de Essentia son los que mejor lo hacen.

Las pruebas en juego son de las mejores. El género encaja bien gracias a la precisión de la detección del ritmo y la velocidad de la canción.

5.2 Evaluación externa

La segunda parte del proceso de evaluación del proyecto se ha realizado mediante una encuesta a usuarios. El juego fue publicado en Itch.io (Figura 51) y difundido entre nuestros allegados con el objetivo de conseguir la mayor cantidad de respuestas a la encuesta. La encuesta se abría automáticamente al finalizar la partida, aunque también es accesible desde el menú de pausa. Esta encuesta cuenta con una serie de preguntas, tanto generales sobre el juego en sí como particulares al trabajo relacionado con este TFG y con el de mi compañero.

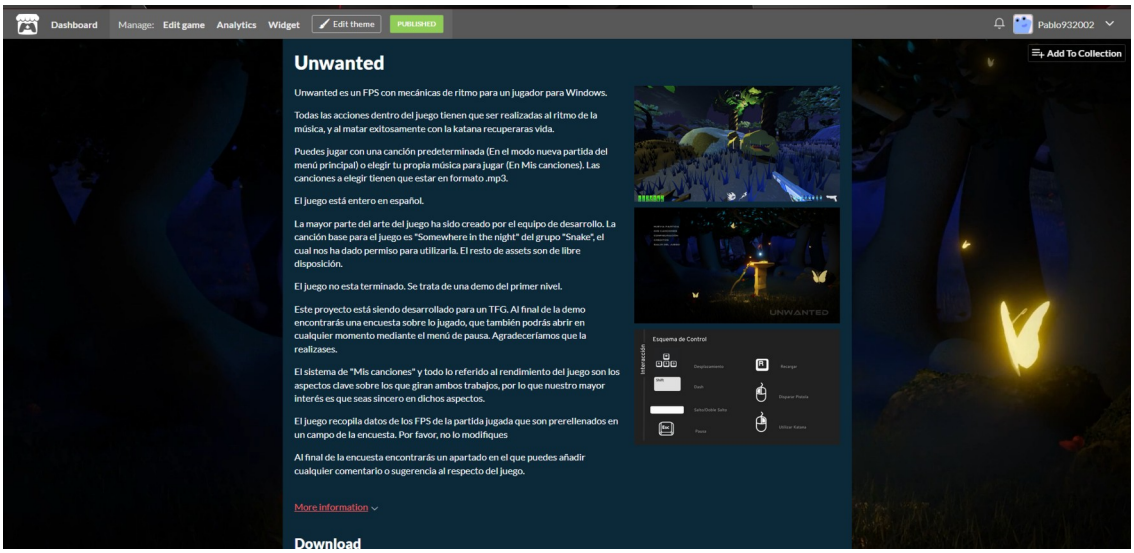


Figura 51: Página de Itch.io de Unwanted

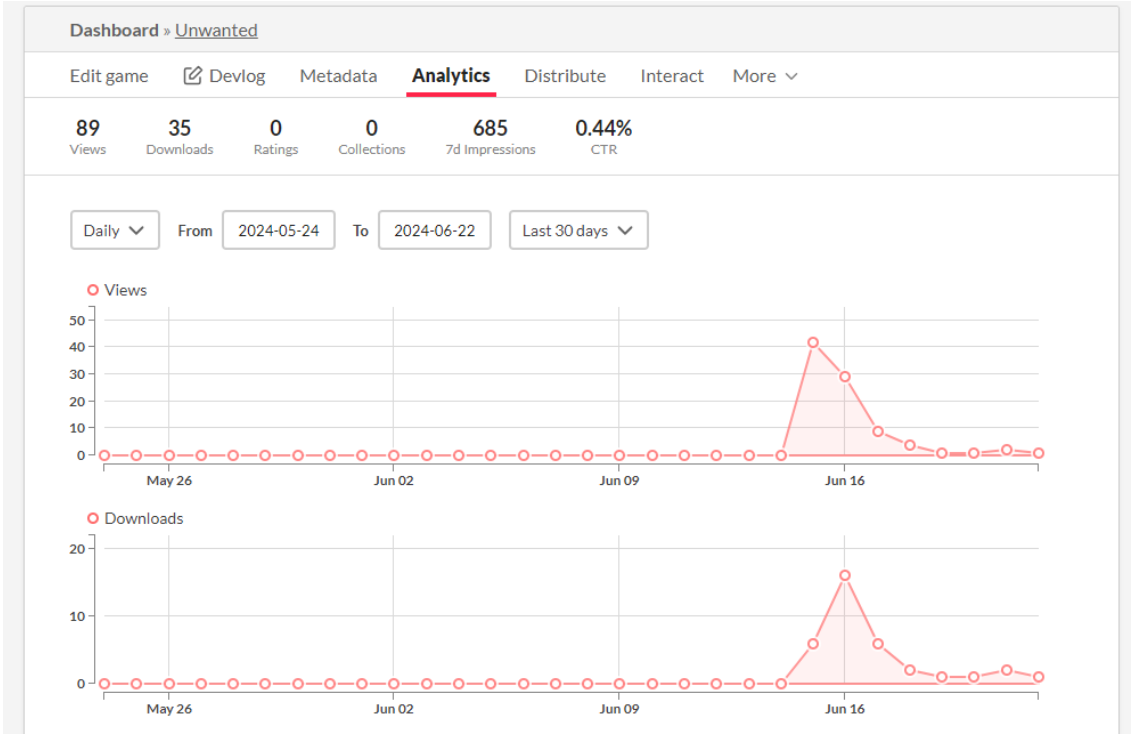


Figura 52: Estadísticas de la página de Itch.io

Unwanted se publicó en Itch.io [30] el sábado 15 de junio. El 22 de junio, una semana después, la página del juego cuenta con 89 visitas y 35 descargas. Estas se concentran mayoritariamente en el periodo que va desde el sábado 15, día de lanzamiento, hasta el martes 18. Estas estadísticas las podemos observar en los gráficos proporcionados por Itch.io en la Figura 52 y la Figura 53.

File download counts

File	Platforms	Downloads	Uploaded at
Unwanted.rar		22	7 days ago
Unwanted1.2.rar		13	6 days ago

Figura 53: Descargas de las dos versiones de Unwanted

Unwanted cuenta con dos versiones distintas. La primera versión cuenta con 22 descargas. Posteriormente al lanzamiento del título se hizo una revisión de bugs y se lanzó el domingo 16 de junio una segunda versión corregida. Esta cuenta con 13 descargas

A continuación se presentan las preguntas de la encuesta y los resultados. La encuesta a fecha del 22 de junio cuenta con 13 respuestas [31].

¿Has jugado alguna vez a un FPS de ritmo?

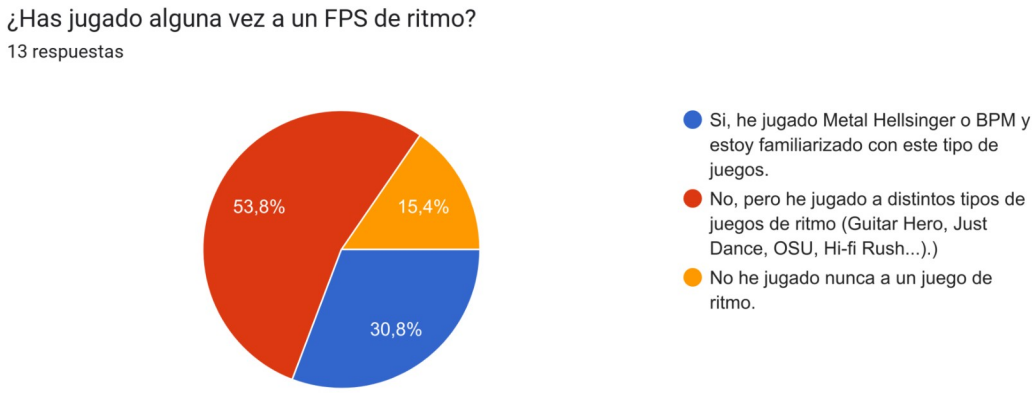


Figura 54: Estadísticas de la pregunta "¿Has jugado alguna vez a un FPS de ritmo?"

Como se ha comentado en el apartado 2 Estado del arte, por norma general podemos ver que la mayoría de usuarios han jugado anteriormente a juegos de ritmo casuales al estilo de Guitar Hero. La cantidad de gente que ha jugado juegos como *Metal Hellsinger* es menor. En este caso el 53,8% de los jugadores corresponden al primer grupo mientras que el 30.8% al segundo. Solamente un 15.4% de jugadores no ha jugado nunca juegos de ritmo. Todo esto lo podemos observar de manera gráfica en la Figura 54.

¿Qué te ha parecido la jugabilidad de Unwanted?

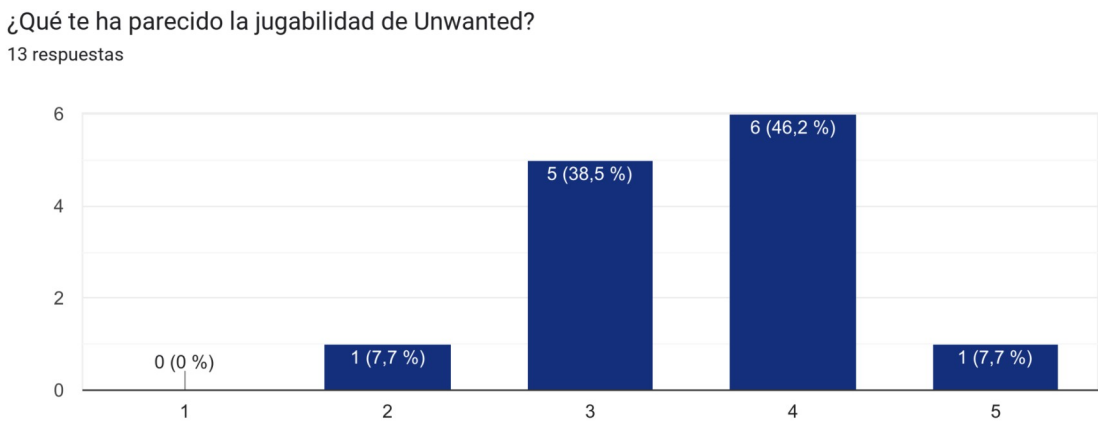


Figura 55: Estadísticas de la pregunta "¿Qué te ha parecido la jugabilidad de Unwanted?"

En cuanto a la jugabilidad podemos observar que un 46.2% la consideran buena, otro 38.5% mediocre, un 7.7% la considera excepcional y sólo un 7.7% la consideran mala, por ello podemos concluir que la jugabilidad ha tenido una aceptación bastante buena. Estos datos corresponden con la Figura 55.

¿Cómo de fluido te ha parecido el rendimiento del juego?

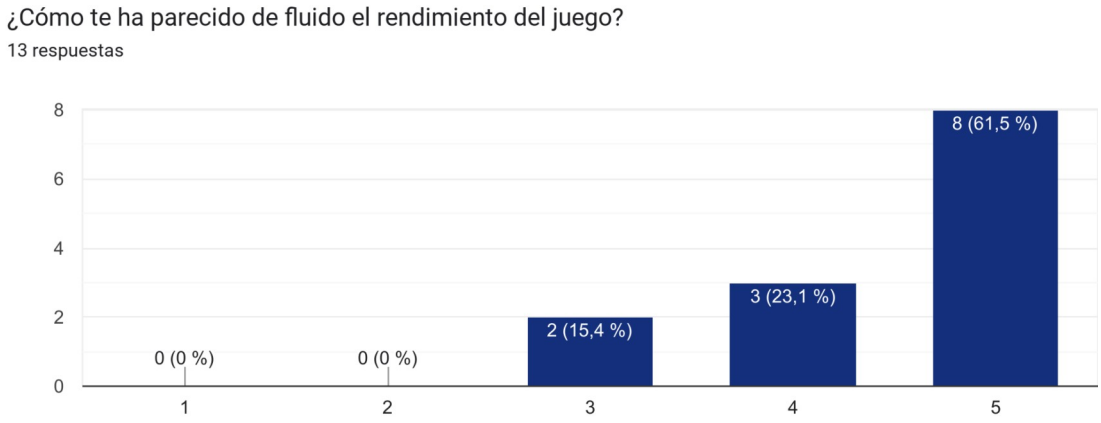


Figura 56: Estasdísticas de la pregunta "¿Cómo de fluido te ha parecido el rendimiento del juego?"

La valoración general por parte de los usuarios varía entre el cuatro y el cinco con un mínimo de tres. Este resultado parece más que aceptable. El resultado sin embargo es susceptible a variaciones debido a la gama del ordenador del usuario y las consideraciones personales de lo que es fluido. La Figura 56 muestra la gráfica de los resultados de esta pregunta.

¿De qué gama es el ordenador en el que has probado el juego?

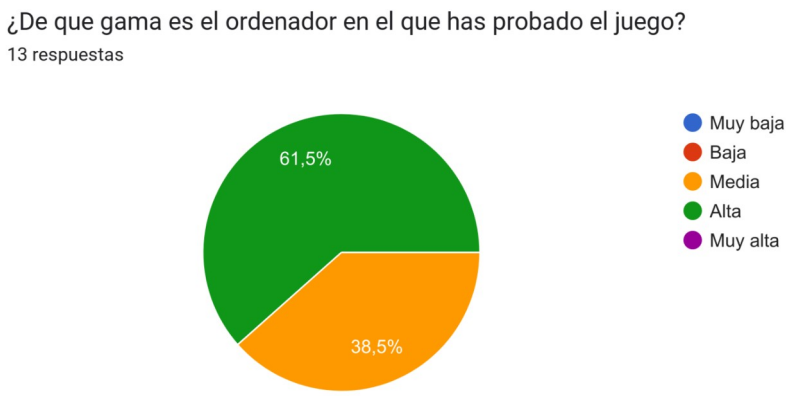


Figura 57: Estadístias de la pregunta "¿De qué gama es el ordenador en el que has probado el juego?"

El 61.5% de los jugadores cuenta con un ordenador de gama alta, mientras que el 38.5% con uno de gama media, esta información será contrastada con el apartado de media de fps para poder corroborar cómo ha funcionado el rendimiento en este tipo de dispositivos. La Figura 57 refleja estos datos.

¿Cómo de divertido te ha parecido jugarlo?

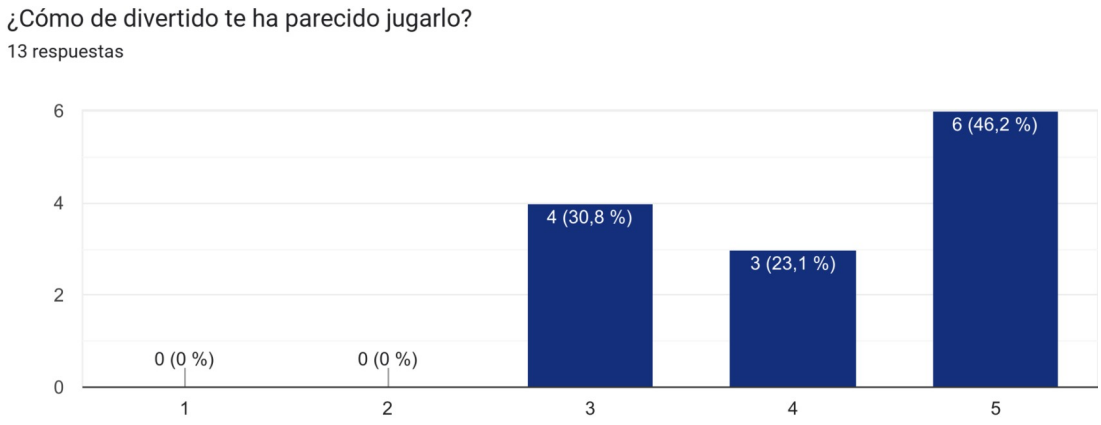


Figura 58: Estadísticas de la pregunta "¿Cómo de divertido te ha parecido jugarlo?"

Existe una gran polarización a nivel de diversión para los usuarios entre el tres y el cinco. Considerar una media de alrededor de cuatro es bastante aceptable. La diferencia de resultados puede deberse a la dificultad del título para gente que no está habituada al género, que son la mayoría, o por cosas como errores varios en general o el sistema de ritmo automático. En la Figura 58 podemos ver el gráfico de barras que representa esta información.

Datos de rendimiento (generados automáticamente, no tocar)

Figura 59: Estadísticas de juego generadas automáticamente

En este apartado, que se rellena automáticamente al finalizar una partida o con el botón en el menú de pausa, podemos observar que el pico máximo de fps de todos los usuarios esta en 600 fps, que debe de ser el límite de las gráficas, mientras que el pico mínimo oscila entre 1,5 fps y 7.5 fps, este ocurre al cargar una escena, y finalmente la media de fps que es el que más valor aporta ya que describe el rendimiento medio en una partida, y se encuentra entre los 55 fps a los 60 fps, lo que es un resultado muy bueno para dispositivos de gama media y alta. En algunos casos podemos encontrar medias de más de 120 fps, lo que se considera un rendimiento muy bueno. Solo hay un caso de media de alrededor de 30 fps, lo que probablemente esté relacionado con las prestaciones del ordenador en las que se jugó. La Figura 59 muestra algunos de los resultados de esta pregunta.

¿Con cuántas canciones has probado el modo "Mis canciones"?

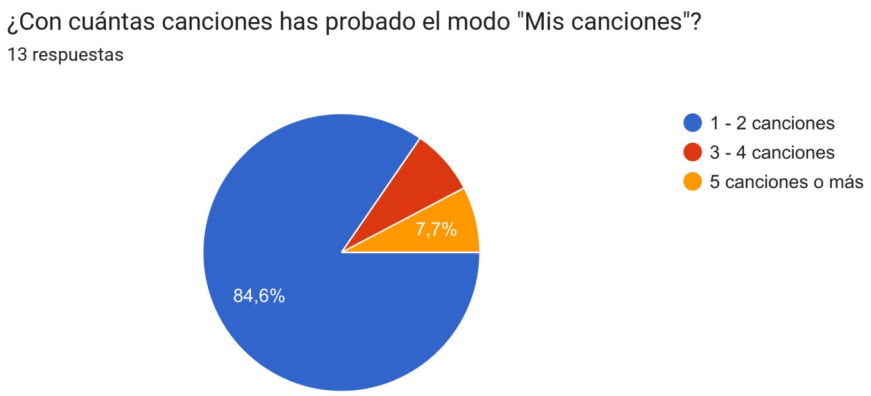


Figura 60: Estadísticas de la pregunta "¿Con cuántas canciones has probado el modo "Mis canciones"?"

La mayoría de usuarios han probado el juego utilizando de una a dos canciones extra. En general la implicación con este modo de juego ha sido baja, pues en algunos casos ni siquiera se ha llegado a probar pero han respondido que de una a dos porque era obligatorio responder a la pregunta. Esto lo podemos observar en la Figura 60. En general esto puede tener que ver con un desinterés por el modo, en parte derivado por la falta de precisión, o por el tiempo necesario para probar con varias canciones.

¿De qué géneros musicales son las canciones con las que has jugado a este modo?

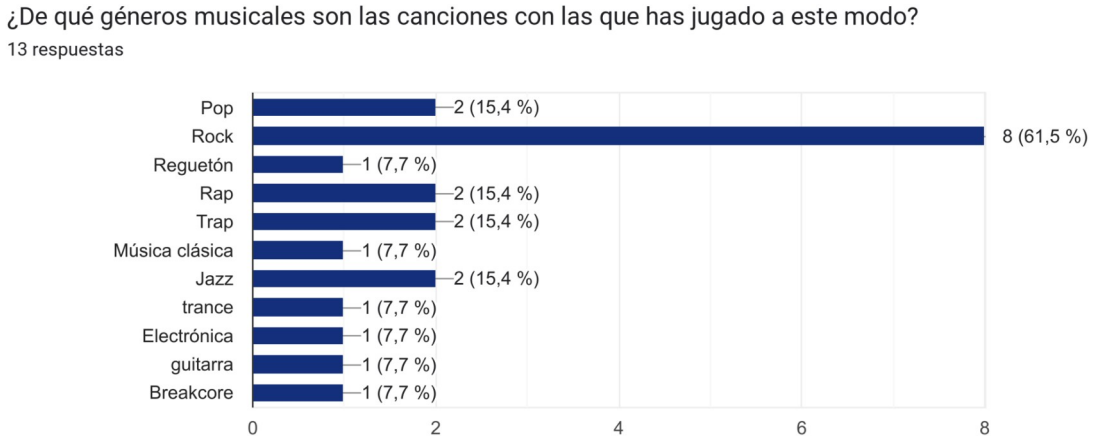


Figura 61: Estadísticas de la pregunta "¿De qué géneros musicales son las canciones con las que has jugado a este modo?"

En general el género musical más utilizado con gran diferencia es el rock. Este resultado era de esperar, pues juegos similares como *Metal Hellsinger* o *Doom* utilizan canciones de este género. Por esto y por la velocidad del ritmo también se considera que este género es el mejor para estos juegos. De la misma forma, las canciones de rock suelen contar con el ritmo más marcado gracias a la batería, por lo

que para el sistema de detección de ritmo debería de ser más fácil calcular bien los golpes de ritmo. La Figura 61 muestra las estadísticas de géneros más utilizados.

¿Cómo de preciso ha sido el ritmo en general en este modo?

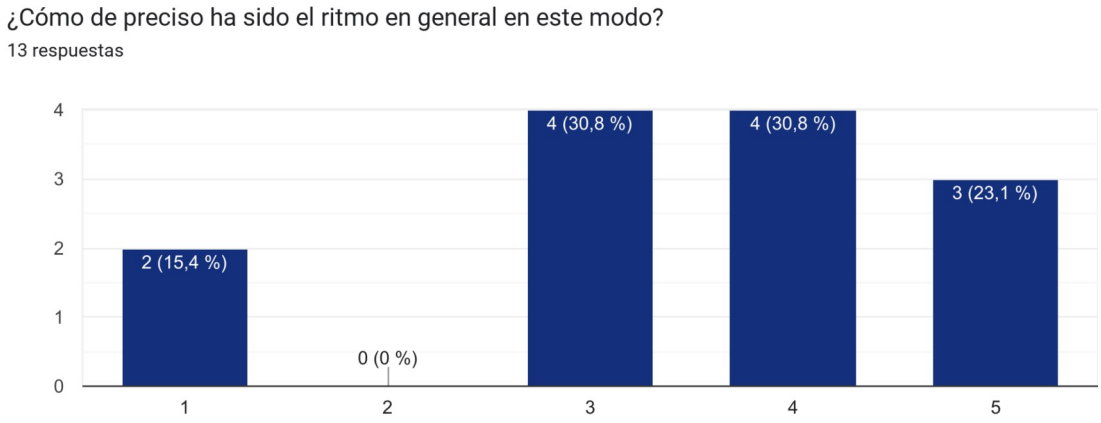


Figura 62: Estadísticas de la pregunta "¿Cómo de preciso ha sido el ritmo en general en este modo?"

Los resultados en cuanto a precisión del sistema de detección de ritmo han sido muy dispares. Junto a un bug que tenía las primeras versiones del juego hay que contar con que el sistema de detección de ritmo utilizado en el ejemplo de Essentia no es preciso en todos casos. Algunas canciones con el ritmo muy marcado son analizadas de manera muy precisa mientras que para otras no acierta nunca. A eso hay que sumar que algunas de las canciones, aun siendo analizadas de manera relativamente precisa, no son especialmente cómodas para jugar por contar con un ritmo demasiado lento. Todo esto repercute negativamente en la jugabilidad y provoca que en algunos casos el juego funcione muy bien y en otros muy mal. Estos resultados son visibles en la Figura 62

¿Cuál es la canción que mejor ha funcionado en este modo?

- Deathmatch
- heroine de Dutch Disorder
- Broken Reality
- Double Violin Concerto 1st Movement – J.S. Bach
- Pa Romperla - Bad Bunny ft. Don Omar
- Foreign Kidd Keo
- Baker Street
- I Really want to stay at your house, Rosa Walton
- Rammstein - Sonne

Por norma general podemos concluir que las canciones estilo rock o con ritmos muy marcados son los que mejor han funcionado, pues para el algoritmo de Essentia deben de ser las más fáciles de detectar.

¿Cuál es la canción que peor ha funcionado en este modo?

- traktor de saymoon
- The Calling - Angelwing
- Luv Sic (Instrumental) - Nujabes
- Blessed Kaidy Kain
- Can i Borrow a Feeling
- Die for you, Grabbitz
- Krystle URL Cyber Palace Mix

De la misma forma, canciones que no cuentan con un ritmo muy marcado por percusión son mucho más difíciles de analizar bien, por lo que no dan buenos resultados.

Cualquier comentario o sugerencia es bienvenida.

Se han agrupado las principales sugerencias y comentarios:

Bugs y errores:

- Los enemigos disparan a través de las paredes
- Las mariposas se quedan atascadas en la cueva
- Al iniciar el juego no aparecen las barras del ritmo
- Varios tipos de canciones tienen muy mala precisión del ritmo

Jugabilidad

- Hay zonas del mapa que son demasiado grandes y rompen el ritmo de juego
- Es difícil navegar el mapa
- No se sabe bien cuando spawnean enemigos ni cuando tienes poca vida

Sugerencias

- Enemigos con ataques únicos (Golem y Miñi)
- Añadir un parry a la Katana

Comentarios

- Es un estilo de juego que está dirigido a un grupo muy concreto de jugadores
- A pesar de que el juego tiene varios fallos la idea no es mala, y aunque esté completamente basado en otros juegos como Metal Hellsinger me sigue pareciendo una buena propuesta.

En general muchos de estos comentarios están relacionados con bugs del juego o por mecánicas del juego original, que son cosas que no corresponden a ninguno de los dos TFG que se han desarrollado a partir de este. Algunos de los bugs han sido corregidos pero los cambios en el diseño son demasiado grandes o directamente no son interesantes para las metas de este proyecto, aunque podrían ser tenidos en cuenta para una posible ampliación. El comentario general que se repite es que el

sistema de ritmo funciona de manera irregular, siendo muy preciso en algunos casos y muy poco preciso en otros.

6 Conclusiones

El resultado final de este TFG es un nuevo modo de juego añadido a Unwanted, un juego de disparos en primera persona con componentes de ritmo, que permite al jugador seleccionar su propia música para jugar al título.

Si revisamos los objetivos marcados al inicio de la memoria podremos comprobar el cumplimiento de cada uno.

Implementar un detector de ritmo en tiempo de ejecución para cualquier canción

Este es el principal objetivo que tenía el proyecto y se ha alcanzado. Actualmente se puede preprocesar cualquier canción que el usuario desee, en un paso previo a la partida pero dentro del juego. El procesamiento se realiza en un tiempo razonable y de manera asíncrona.

Permitir al usuario jugar con canciones previamente analizadas en el juego.

Este objetivo también se ha cumplido. Actualmente se puede utilizar cualquier canción dentro del juego, independientemente de su género. El mayor problema que tiene esto es que el sistema de detección de ritmo es bastante preciso para canciones de géneros como el rock, el pop, el rap, el reggaetón o algunos géneros electrónicos pero algunos géneros como bandas sonoras o algunos tipos de música electrónica fallan considerablemente. Es posible jugar con cualquier canción pero la experiencia puede ser positiva o negativa dependiendo de la canción elegida.

De la misma forma, algunos géneros musicales independientemente de la precisión del ritmo detectado dan peores resultados en cuanto a jugabilidad porque simplemente no son compatibles con el tipo de juego que es Unwanted, por su componente de juego de disparos y no exclusivamente de juego de ritmo.

Las canciones procesadas anteriormente se almacenan en una base de datos local. Esto permite al usuario procesar una vez una canción y utilizarla infinitas veces sin coste alguno y con el mismo resultado.

Desarrollar un componente de detección de ritmo fácilmente reutilizable en cualquier otro videojuego en Unity.

Debido a las características del sistema de detección, la misma escena que permite procesar canciones y seleccionar la elegida de la base de datos podría ser reutilizada sin necesidad de ninguna modificación por cualquier otro juego de ritmo similar a Unwanted. De la misma forma, esta misma base de datos podría ser compartida entre varios videojuegos, pues la detección de ritmo es genérica e independiente de la implementación del juego en cuestión.

Aprender sobre técnicas de detección de ritmo en canciones.

Este objetivo también está cumplido, pues he aprendido sobre cómo funcionan distintos algoritmos de detección de ritmo, tanto mediante Audacity como mediante librerías.

Practicar en el uso de Unity y C#

Para poder desarrollar este proyecto ha sido necesario investigar sobre técnicas de manejo de ficheros en Unity o ejecución asíncrona de aplicaciones de terminal en C#, lo que ha contribuido en mi desarrollo personal como programador.

Practicar en el uso de metodologías ágiles

Para poder llevar a cabo este proyecto ha sido necesario trabajar con técnicas de especificación de requisitos, mockups, tableros kanban, sprints, reuniones de revisión y control de versiones en GitHub. Todo ello ha sido un paso más en mi crecimiento personal como profesional en el sector de la ingeniería del software.

Así pues y habiendo revisado todos los objetivos marcados al inicio del proyecto, podemos afirmar que se han cumplido de manera correcta. El desarrollo ha tenido un buen grado de aceptación por los usuarios, y se ha publicado un juego completo que está listo para que los jugadores puedan disfrutar de él.

7 Trabajo futuro

Los objetivos de este proyecto han sido cumplidos tal y como se esperaba. Se había planteado añadir una nueva funcionalidad a un videojuego ya existente y se ha conseguido. A lo que el alcance de este trabajo respecta, no existe nada que se haya quedado pendiente, más allá de poder encontrar algún sistema de detección de ritmo que mejore los resultados del utilizado.

Si hablamos del juego en general, el alcance del proyecto original era mucho más amplio de lo alcanzado. No se pudo terminar el primer nivel de juego y a esto habría que añadirle el combate contra el jefe final, que estaba ya planteado y con el que se empezó a trabajar. El juego en sí también podría recibir muchas mejoras como más armas, niveles o enemigos. En caso de volvernos a juntar el equipo de desarrollo original del proyecto y en vistas de la relativamente buena recepción que ha tenido el título se podría llegar a ampliar para pasar de lo que actualmente es una demo a disponer de un juego terminado.

8 Referencias

Todos los enlaces se han accedido antes del 27/6/2024

[1] Dance Dance Revolution (serie) – Wikipedia, la enciclopedia libre. Consultado en: https://es.wikipedia.org/wiki/Dance_Dance_Revolution_(serie)

[2] Guitar Hero – Wikipedia, la enciclopedia libre. Consultado en: https://es.wikipedia.org/wiki/Guitar_Hero

[3] Rock Band (serie) – Wikipedia, la enciclopedia libre. Consultado en: https://es.wikipedia.org/wiki/Rock_Band_(serie)

[4] Piano Tiles 2 – Wikipedia, la enciclopedia libre. Consultado en: https://es.wikipedia.org/wiki/Piano_Tiles_2

[5] Friday Nigth Funkin' – Wikipedia, la enciclopedia libre. Consultado en: https://es.wikipedia.org/wiki/Friday_Night_Funkin%27

[6] Just Dance (serie) – Wikipedia, la enciclopedia libre. Consultado en: https://es.wikipedia.org/wiki/Just_Dance_(serie)

[7] Metal Hellsinger – Wikipedia, la enciclopedia libre. Consultado en: https://en.wikipedia.org/wiki/Metal:_Hellsinger

[8] Hi-Fi Rush – Wikipedia, la enciclopedia libre. Consultado en: https://es.wikipedia.org/wiki/Hi-Fi_Rush

[9] Crypt of the NecroDancer – Wikipedia, la enciclopedia libre. Consultado en: https://en.wikipedia.org/wiki/Crypt_of_the_NecroDancer

[10] Doom (franquicia) – Wikipedia, la enciclopedia libre. Consultado en: https://es.wikipedia.org/wiki/Doom_(franquicia)

[11] Audacity: https://www.audacityteam.org/

[12] Vamp Plugins: https://www.vamp-plugins.org/

[13] Vamp Plugins pack: https://www.vamp-plugins.org/pack.html

[14] Transformada de Fourier – Wikipedia, la enciclopedia libre. Consultado en: https://es.wikipedia.org/wiki/Transformada_de_Fourier

[15] NAudio – GitHub: https://github.com/naudio/NAudio

[16] FftSharp – GitHub: https://github.com/swharden/FftSharp

[17] Aubio: https://aubio.org/

[18] Aubio – Github: https://github.com/aubio/aubio

[19] Essentia: https://essentia.upf.edu/index.html

[20] Essentia – GitHub: https://github.com/MTG/essentia

[21] Essentia – Applications: https://essentia.upf.edu/applications.html

[22] Essentia – Ejecutables compilados para Windows: https://essentia.upf.edu/extractors/essentia-extractors-v2.1_beta5-356-g673b6a14-win-i686/

[23] Encuesta sobre hardware y software de Steam: May 2024: https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam?l=spanish

[24] StandaloneFileBrowser – GitHub: https://github.com/gkngkc/UnityStandaloneFileBrowser

[25] Unity: https://unity.com

[26] Github: https://github.com/

[27] Freesound.org: https://freesound.org/

[28] Trello: https://trello.com/

[29] Itch.io: https://itch.io/

[30] Unwanted – Itch.io: https://pablo932002.itch.io/unwanted

[31] Encuesta TFG Unwanted: https://docs.google.com/forms/d/e/1FAIpQLSfMFOLmyo0daBWQUrJsDnVLp4uxUCpHxzxKTHHCzqSoT0nF9w/viewform

9 Anexos

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
Fin de la pobreza.				X
Hambre cero.				X
Salud y bienestar.				X
Educación de calidad.				X
Igualdad de género.				X
Agua limpia y saneamiento.				X
Energía asequible y no contaminante.				X
Trabajo decente y crecimiento económico.				X
Industria, innovación e infraestructuras.		X		
Reducción de las desigualdades.				X
Ciudades y comunidades sostenibles.				X
Producción y consumo responsables.	X			
Acción por el clima.				X
Vida submarina.				X
Vida de ecosistemas terrestres.				X
Paz, justicia e instituciones sólidas.				X
Alianzas para lograr objetivos.				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Los objetivos de desarrollo sostenible son una serie de metas marcadas por la ONU que tienen como objetivo encaminar el desarrollo de todos los países alrededor del mundo hacía una versión más sostenible de este. Esto a largo plazo pretende mejorar la vida en el planeta y asegurar un futuro mejor. Es por esto que es de especial relevancia el tratar de cumplir siempre con estas metas.

Debido a la naturaleza del proyecto desarrollado, encontramos dos ODS con los que este se alinea de cierta forma. Estos son el ODS 9: Industria, innovación e infraestructuras y el ODS 12: Producción y consumo responsables.

Respecto al ODS 9: Industria, innovación e infraestructuras, se considera que este proyecto cumple en cierta medida con este debido a que la principal característica desarrollada en este TFG es un sistema de detección automático de ritmo de canciones en un videojuego, lo cual es una característica novedosa y de alta innovación que no encontramos en productos similares en el mercado

Respecto al ODS 12: Producción y consumo responsables, se considera que esta ampliación de Unwanted también se alinea con este objetivo en gran medida. Al permitir al usuario utilizar su propia música en el juego estamos evitando invertir recursos en producción musical innecesaria, reutilizando los recursos previamente existentes. De la misma forma, el componente de análisis de canciones desarrollado es reutilizable, permitiendo también reutilizar recursos sin invertir nada en el desarrollo de cero de este componente.

UNWANTED

Game Design Document

ETSINF - Ingeniería informática

Jefferson Paul Caiza Jami

Pablo José Peral Tamarit

Jose Francisco García Sánchez

Pablo Granell Robles

BBAA - Diseño y tecnologías creativas

María Manzanaro Alfaro

Rubén Rosaleny Benaches

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice

1. General

- **Género:**

Ritmo FPS

- **Plataforma:**

PC (Windows)

- **Público objetivo:**

Jugador casual, con un atractivo especial a los juegos de ritmo y a los juegos frenéticos.

Jugador de FPS que busque una experiencia distinta

- **Clasificación PEGI justificada:**

PEGI 12. Se trata de un juego de fantasía con violencia poco explícita ejercida contra criaturas irreales. No se muestra sangre, vísceras o desmembramientos de ningún tipo. No existe lenguaje soez ni ningún tipo de contenido para adultos.

- **Novedad:**

FPS de ritmo con un estilo de fantasía. El mercado de los juegos de ritmo está poco explotado. Otros competidores utilizan estéticas con inspiración demoníaca o de ciencia ficción. La temática de fantasía hace a Unwanted una experiencia diferente al resto.

- **¿Qué lo hace emocionante?:**

El componente de ritmo como mecánica central a través de la cual giran las demás añade un componente de dificultad al juego diferente al resto. Seguir el ritmo durante el combate de manera correcta genera una gran satisfacción al usuario como recompensa al aprendizaje necesario para llevarlo a cabo.

2. Gameplay

- **Acciones personajes:**

El personaje tendrá una única velocidad de movimiento (No sprint) que deberá ser rápida para favorecer la velocidad global del juego. También dispondrá de habilidades complementarias como el dash o el doble salto. La forma principal de ataque será mediante armas de fuego usando el click izquierdo y un ataque secundario con una

katana usando el click derecho.

- **Acciones NPCs:**

Los NPCs o enemigos atacarán al personaje, sus ataques cambiarán dependiendo del tipo de enemigo el cual dependerá del entorno o bioma en el que se encuentre. Por ejemplo en un bioma de bosque los enemigos podrían tener un aspecto de planta y tener ataques a melee con látigos de raíces de plantas.

- **Objetos:**

No hay objetos en el juego ya que un inventario haría que la experiencia de juego fuera lenta.

- **Armamento:**

El jugador llevará siempre equipadas 2 armas a la vez, una en cada mano. La principal será un arma de fuego y la secundaria una espada. Ambas no tendrán retroceso ni excesivo cooldown para evitar romper el ritmo de juego.

- **Posibilidades:**

Cuando el jugador cuadre el ritmo con la música o con un patrón fijado previamente se ofrecerá la posibilidad de un combo que proporcionará un multiplicador de puntos o de daño. Una vez terminado el nivel se mostrarán stats como cuántos aciertos de ritmo el jugador ha realizado, cuánto tiempo del total del nivel se ha mantenido al ritmo, cuanta puntuación ha conseguido por ello, etc.

- **Potenciadores:**

No hay potenciadores.

- **Efectos negativos:**

Si atacas fuera de ritmo recibes una penalización, perdiendo el multiplicador del combo y no haciendo el daño.

3. Lore

3.1 Prólogo

Tress despierta en un lugar oscuro, sin recordar nada y agarrada a una cápsula que contiene una niña dentro. Cuando habla con esta niña, Gaia, se dan cuenta de que no recuerdan nada de dónde están ni de quiénes son, pero deciden trabajar en equipo.

Mientras piensan qué hacer, el colgante de Gaia empieza a brillar y a elevarse en el aire, como indicando hacia una dirección. Cuando lo siguen, son llevadas al exterior de la cueva donde descubren estar en un planeta desconocido lleno de naturaleza, pero enseguida son atacadas.

Gaia, asustada, se fusiona con el colgante que llevaba y Tress, al ponérselo, empieza a escuchar una música desde este. A partir de ahí Tress se ve poseída por el ritmo y empieza a enfrentarse a los enemigos.

3.2 Gameplay

Tras atravesar el primer bioma y haber derrotado al "guardián" (boss del primer nivel), las protagonistas llegan a una construcción en ruinas en la que refugiarse por un tiempo. En esta descubren unos dibujos que describen una guerra entre androides, como Tress, contra la naturaleza del planeta. La naturaleza aparece como ganadora al haber derrotado a todos los androides, aunque ahora saben que no es cierto. También aparece un retrato que es parecido a Gaia, pero no parece haber más contexto sobre ello. A partir de aquí las protagonistas deciden avanzar, ahora seguras de que la naturaleza las quiere muertas, aunque no saben por qué. Primero deben descubrir quienes son ellas mismas.

Una vez han conseguido atravesar el desierto, y después de derrotar al "androide corrupto" (boss del segundo nivel), llegan a unas ruinas más grandes. Están todas repletas de androides destruídos y despedazados, y tras mayor inspección parece que se trataba de una ciudad androide. Cuando entran a una de las casas para descansar descubren unos documentos escritos, como un diario. En él explica que todos en la ciudad están temerosos por la hostilidad que la naturaleza tiene hacia ellos, y que algunos están empezando a entrenar por si la situación llegase a más. También habla de que "la creadora" está desaparecida, y sin una mediadora como ella no pueden comunicarse bien con la naturaleza para llegar a la paz.

Ahora que Tress comprende mejor de dónde proviene, siente tristeza y rabia por la desaparición de su pueblo y se pone en marcha de nuevo hasta lo que en algunos mapas estaba descrito como el hogar de "la creadora", ya que es la única que puede darle todas las respuestas que necesita. Gaia, también triste por la situación de su compañera, decide guardar silencio y no interponerse, a pesar de que sospecha que ella ya sabe quién es "la creadora".

Tras derrotar al "ser de luz" (boss del tercer nivel), y que esta se despida con un "bienvenida de vuelta", Gaia sale del colgante, entra a la torre, y confiesa que sabe que ella es "la creadora", y se disculpa con Tress entre lágrimas por todo lo que ha pasado. Cuando Tress se entera, apunta su katana a Gaia y le pide respuestas.

Gaia explica que ella creó todo aquel planeta, que primero creó la naturaleza para que se desarrollará por el lugar, y una vez estuvo feliz con el resultado, se dió cuenta de que se sentía muy sola porque quería tener a su lado gente como ella. Entonces recogió productos de la naturaleza y creó a los androides, que aunque fuesen de metal, compartían apariencia con ella y le hacían más compañía. La naturaleza, tomándose la creación de los androides como una amenaza hacia ellos, empezaron a destruirlos y a pedir el poder máximo de la naturaleza de vuelta. Gaia, desesperada

por la situación de sus creaciones destruyéndose entre sí, viendo el final de su planeta cerca, y la muerte de sus amigos por todas partes, se encerró en la torre a pensar cómo parar todo, pero no le quedaba poder para crear nada más, y no podía encontrar ninguna solución.

Entonces Tress, en medio de la guerra, recordó a su creadora y sabiendo que ella era la última esperanza del planeta, la sacó de la torre y se escondió con ella en un lugar alejado y seguro, esperando dormir hasta que todo pasase.

Tras escuchar la historia y conseguir sus memorias también de vuelta, Tress baja su katana y se arrodilla ante Gaia, entregando el colgante y pidiéndole que traiga de vuelta a todos sus hermanos y a todos los seres de la naturaleza que ha destruído en el camino, para poder reconstruir el planeta de nuevo. Gaia, sacando toda la magia del colgante, revive a todos y habla tanto con los androides como con la naturaleza, contando su historia, asegurando que ninguno está allí para reemplazar al otro, y que nadie se debe sentir no deseado (“unwanted”). Gaia agota toda su magia y pierde su poder de “creadora”, pero recupera la paz en el planeta y todo lo que siempre había querido.

3.3 Epílogo

Una vez terminado el juego, podemos resumir el resto como que la guerra y las tensiones entre ambos bandos desaparecieron. Los androides se desarrollaron en distintas profesiones y papeles que resultaban de su interés, y la naturaleza siguió creciendo hasta el nacimiento de nuevos seres: los animales. Trabajando en equipo consiguieron crear preciosas construcciones y crecer aprendiendo unos de los otros, y Gaia ayudó a ambos con sus ideas mientras investigaban sobre cómo hacer de su querido planeta un sitio mejor.

4. Objetivos de nivel

El objetivo de cada nivel es ir sobreviviendo a las oleadas de enemigos repartidas por diferentes áreas del nivel hasta llegar al checkpoint y derrotar a un boss.

El boss tendrá una barra de vida visible en el HUD, varias fases y diferentes tipos de ataques por cada fase. Al derrotar al boss hay una pequeña secuencia de finalización y termina la pelea. Esa secuencia se encadena con la secuencia del final del nivel, que incluye una tabla con la puntuación.

Después de eso escapas del nivel, se desbloquea algo de historia, se regenera la vida y se guarda la partida.

La iluminación se usa de forma que el jugador puede encontrar pistas implícitas de cómo seguir el nivel.

Los niveles se ensanchan tipo coliseo cuando hay una pelea contra un boss y al

terminar la pelea o se activa la secuencia o el boss suelta algo y el jugador la activa manualmente.

Los niveles tendrían rejugabilidad desde el punto de vista de la jugabilidad y de la historia:

1. Desde el punto de vista de la jugabilidad por tener un sistema de combos y ritmo que premia al jugador consistente que consigue seguir el ritmo y al jugador ingenioso que experimenta con diferentes combos.

2. Desde el punto de vista de la historia por saber el final e ir desde el principio viendo la historia desarrollarse.

Los retos individuales de cada nivel serían:

1. Limpiar oleadas de enemigos, descubriendo diferentes tipos de enemigos y buscando sus debilidades.

2. Disfrutar del ritmo, si sigues el ritmo el juego se hace fácil, si no se complica.

3. Explorar los niveles, y encontrar la narrativa implícita del entorno.

La historia del juego se va enseñando a lo largo de los niveles y con una pequeña transición entre cada nivel del juego. Mientras vas navegando el nivel encuentras pequeños símbolos/ pequeños elementos (estilo dark souls) que van descubriendo poco a poco la historia. Al final del nivel desvelamos alguna otra pieza de información (estilo plants vs zombies).

No hay nuevos personajes, solo nuevos tipos de enemigos dependiendo del checkpoint y del nivel. Además esto ayudaría a la curva de dificultad.

Los enemigos están creados por Gaia y vuelven a ella cuando mueren. Algunas otras partes del nivel dejarían de alguna forma implícita que Gaia es la creadora del mundo (como diferentes inscritos en las paredes parecidos a los grabados de la cápsula que contiene a Gaia) lo que recompensará a los jugadores observadores.

Resumen:

- **Objetivo de cada nivel:**

Sobrevivir hasta escapar.

- **Retos y su relación:**

Limpiar oleadas de enemigos, seguir el ritmo de la música, explorar los niveles y continuar la historia. El juego debería de ser lo suficientemente fácil si sigues el ritmo y difícil si no lo sigues.

- **Justificación diferentes escenas/niveles:**

Tienes que escapar de la zona y seguir avanzando para descubrir por qué estabas ahí en primer lugar.

- **Retos a superar:**

Eliminar a los enemigos de las distintas zonas perdiendo la mínima vida posible.

- **Nuevos personajes:**

No, solo enemigos nuevos.

- **Justificación nuevos personajes:**

Los enemigos son creados por Gaia, cuando Tress los mata vuelven al colgante de Gaia.

5. Personajes

5.1 Gaia

- **Descripción de personaje:**

Gaia es una de las 2 protagonistas, pero no es jugable. Irá acompañando al jugador de forma no física (dentro de un colgante) y la mayoría del juego solo podremos escuchar su voz. A pesar de esto, es el personaje más influyente y principal de la historia.

- **Descripción física:**

Gaia tiene el aspecto de una niña de 10-12 años, es negra, con el pelo negro corto y rizado, y ojos azules (del color del colgante). Mide 139 cm, sin estar aún muy desarrollada. Dos cuernos salen de su cabeza, y tiene marcas blancas estilo tribal en las mejillas, brazos y tobillos.

- **Descripción psicológica:**

A pesar de ser en un principio un ser omnipotente, Gaia se desmorona fácilmente y le cuesta actuar en situaciones de presión o miedo, en las que se queda paralizada. Es una niña amigable y respetada por todos, bastante madura pero que aún demuestra comportamientos de ser joven.

- **Atuendo:**

Gaia siempre lleva un velo cayendo sobre su cara, y su atuendo se basa en telas y recortes unos encima de otros. La mayoría de su ropa está hecha jirones ya que

siempre viste con lo mismo y está desgastado. Tampoco utiliza zapatos, pero sí tiene algunos elementos de joyería como brazaletes y pendientes, a parte, por supuesto, de su colgante mágico.

- **Historia de personaje:**

No se sabe demasiado sobre cómo nació, pero sí se sabe que desde pequeña fue creando un planeta a su agrado donde decidió quedarse a vivir, creando así la naturaleza y, más tarde, los androides.

- **Evolución de personaje:**

El arco de personaje de Gaia se basa en aceptar los errores y tomar responsabilidad por el mal que ha provocado, pasando así de ser un personaje cobarde y pasivo a un personaje empoderado y con voluntad de redimirse de sus errores.

5.2 Tress

- **Descripción de personaje:**

Tress es una de las 2 protagonistas, y la única personaje jugable. Como el juego es en 1ª persona, el jugador verá todo el juego desde los ojos de ella, y la mayoría del tiempo solo verá sus brazos.

- **Descripción física:**

Tress es una androide con atributos femeninos de una mujer adulta. Está toda hecha de metal en distintas tonalidades de plateado, y como cara solo se reconocen 2 agujeros en los ojos. Mide 183 cm y es de complexión delgada, lo que la hace un poco más ligera. En contraste con el plateado brillante, Tress tiene muchos rasguños que han dejado marcas negras en su cuerpo, como arañazos, cortes, o zonas sin chapa, que obtuvo tras la guerra.

- **Descripción psicológica:**

A pesar de que no tenga un aspecto muy expresivo (no tiene prácticamente rostro) eso no le impide demostrar una personalidad muy decidida y segura de sí misma a través de sus acciones. Suele tener pensamiento y soluciones rápidas, no suele demostrar debilidad y no suele perder la calma. Aunque parezca fría y radical, también tiene un lado sentimental.

- **Atuendo:**

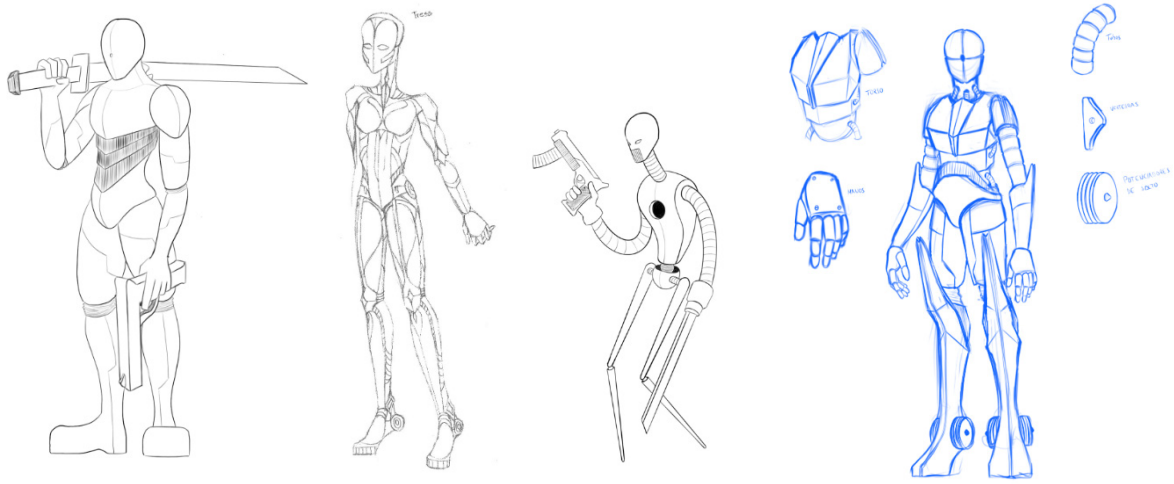
No lleva ningún tipo de atuendo sobre su estructura metálica.

- **Historia de personaje:**

Tress es una androide creada por Gaia. No parece que tuviese una relación más íntima que los demás androides con su creadora, pero sí que podemos deducir que fue una de las androides que vivía en la ciudad y que empezaron a entrenar viendo que la guerra se aproximaba, y la única que pensó en una solución a largo plazo para el planeta. Tress tendría que haber peleado en la guerra para llevar a Gaia escondida en la cápsula desde su torre hasta el escondite donde se encontraron.

- **Evolución de personaje:**

El arco de personaje de Tress se basa en el perdón. Gracias a Tress la historia avanza como es, y a pesar de que en un momento del juego “la creadora” se convierte en su enemigo, después sabe dar un paso atrás, entender la situación de Gaia y perdonarla, dando así las fuerzas a Gaia de enmendarse.



(Este último es el diseño final de Tress)

5.3 Jefes

Son los enemigos más fuertes que se encuentran al final de cada nivel. Son 3 en total: “Guardián”, “Androide Corrupto” y “Ser de Luz”. Todos ellos tendrán un pequeño diálogo con el jugador antes y/o después de su batalla.

- **Guardián:**

Es un gran y viejo árbol que delimitó los dos territorios (bosque y desierto) en la guerra. Fue de los primeros que nacieron en el planeta y su odio por los androides era muy profundo, de manera que fue uno de los líderes de la naturaleza durante el conflicto. Es orgulloso y charlatán, y aunque sus ataques sean lentos por su tamaño, hacen mucho daño.

- **Androide Corrupto:**

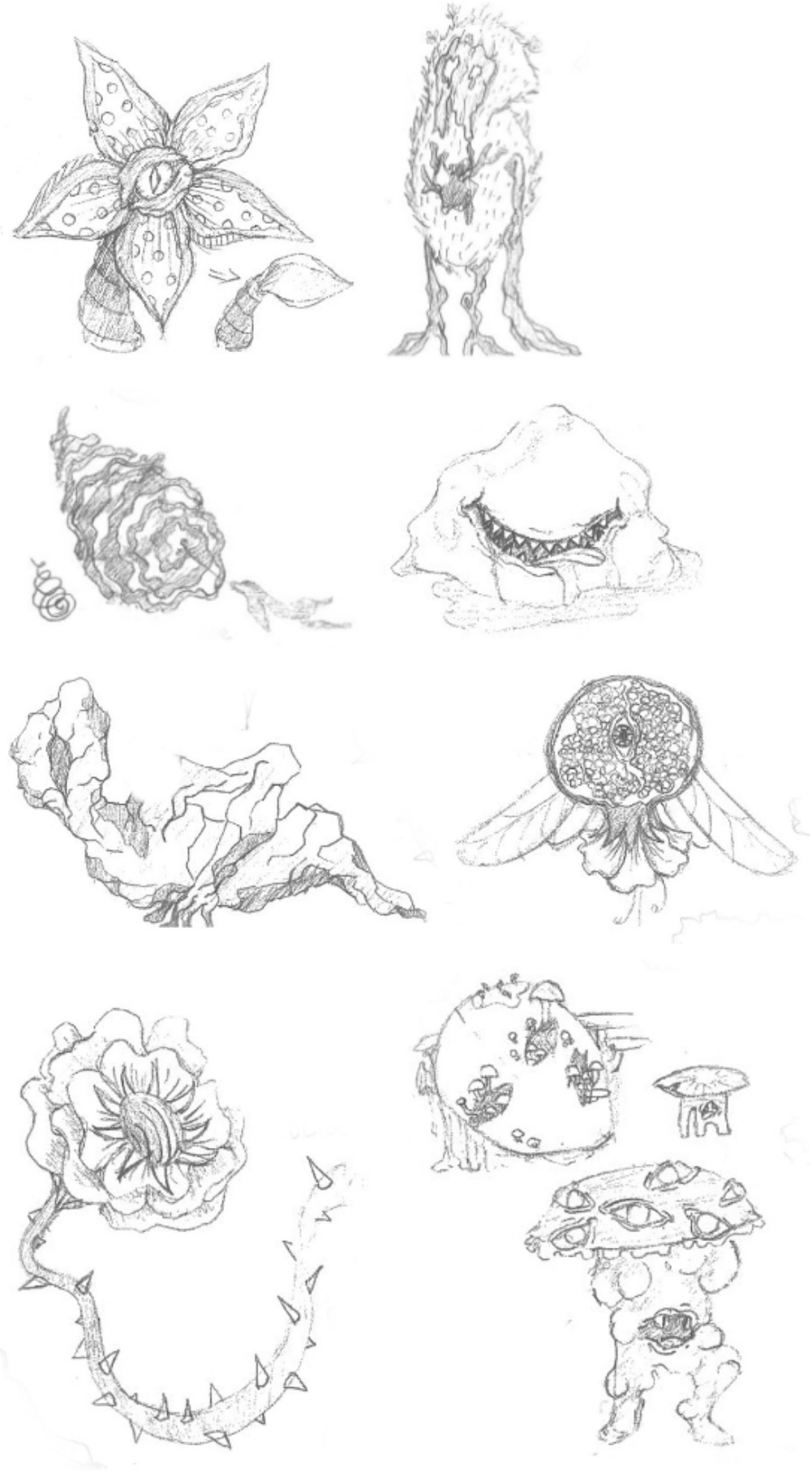
Es el cuerpo de un androide que murió en la guerra y que distintos tipos de hongos y plantas han utilizado como recipiente para ser más fuertes. No queda nada de la consciencia del robot, así que se mueve torpemente a voluntad de los nuevos ocupantes. A pesar de ello, el androide sigue manteniendo la velocidad y habilidad que tenía en sus tiempos, lo que lo hace un enemigo muy peligroso al estar al mismo nivel de entrenamiento que Tress.

- **Ser de Luz:**

El Ser de Luz es el antiguo guardián y confidente de la torre donde vivía Gaia. Fue creado por Gaia a partir de solo magia y su responsabilidad era protegerla y evitar que los desconocidos entren a la torre. El Ser de Luz reconoce a Tress, pero Tress llega a la torre muy enfurecida y no quiere escucharlo, de manera que el Ser de Luz decide cumplir su papel de defender la torre y luchar contra ella.

5.4 Otros enemigos

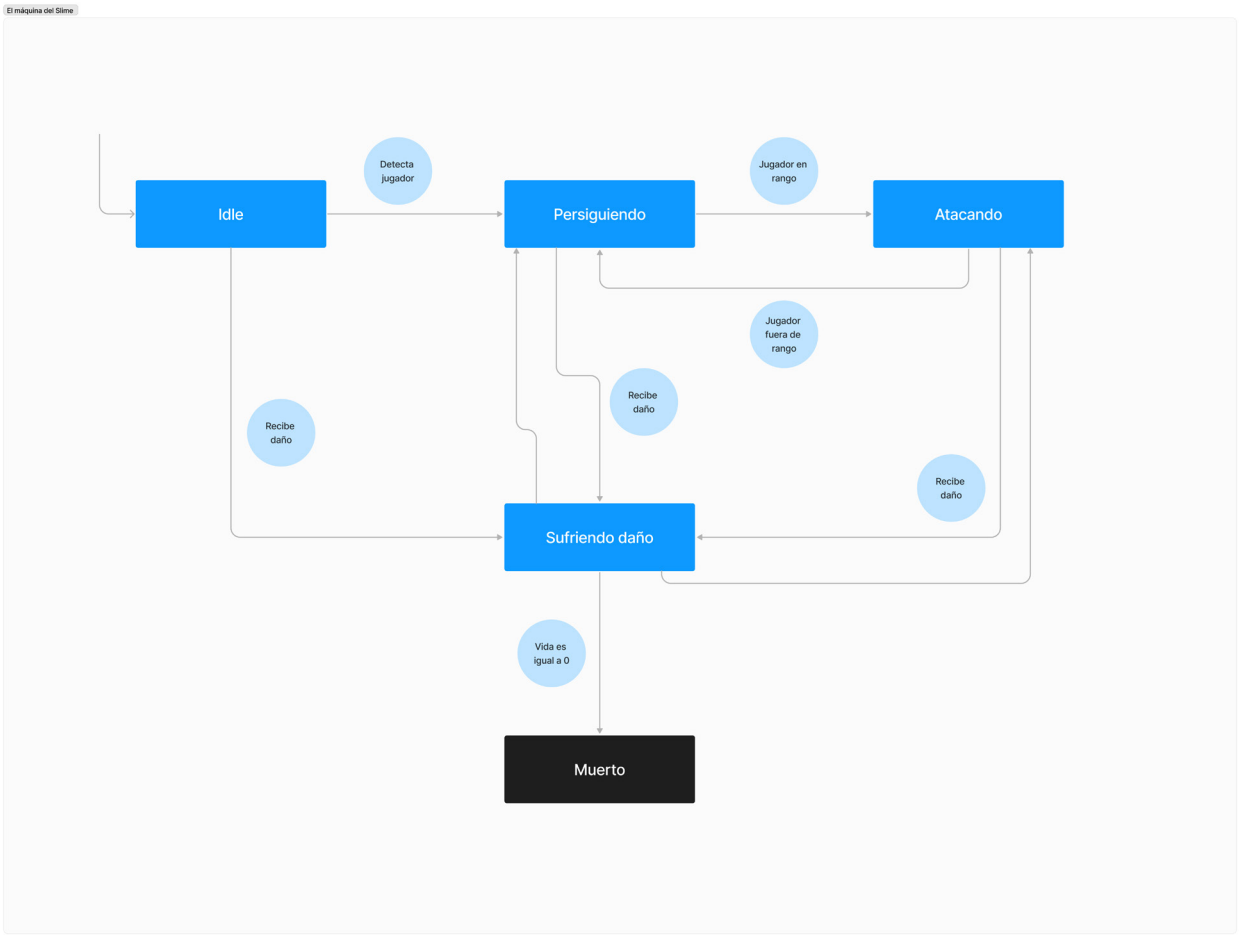
El resto de enemigos a lo largo de los niveles son parte de la flora del planeta que ha evolucionado para poder luchar. Todos comparten la misma mentalidad, estando en contra de los androides y sintiéndose insuficientes para Gaia. Cada uno tiene distintas habilidades y ataques que impedirán el rápido avance del jugador por el nivel.



***Se pueden encontrar los modelos 3D de los enemigos y el boss final en los anexos**

6. Diagramas de estado

6.1 Diagrama de estados y transiciones de Slime

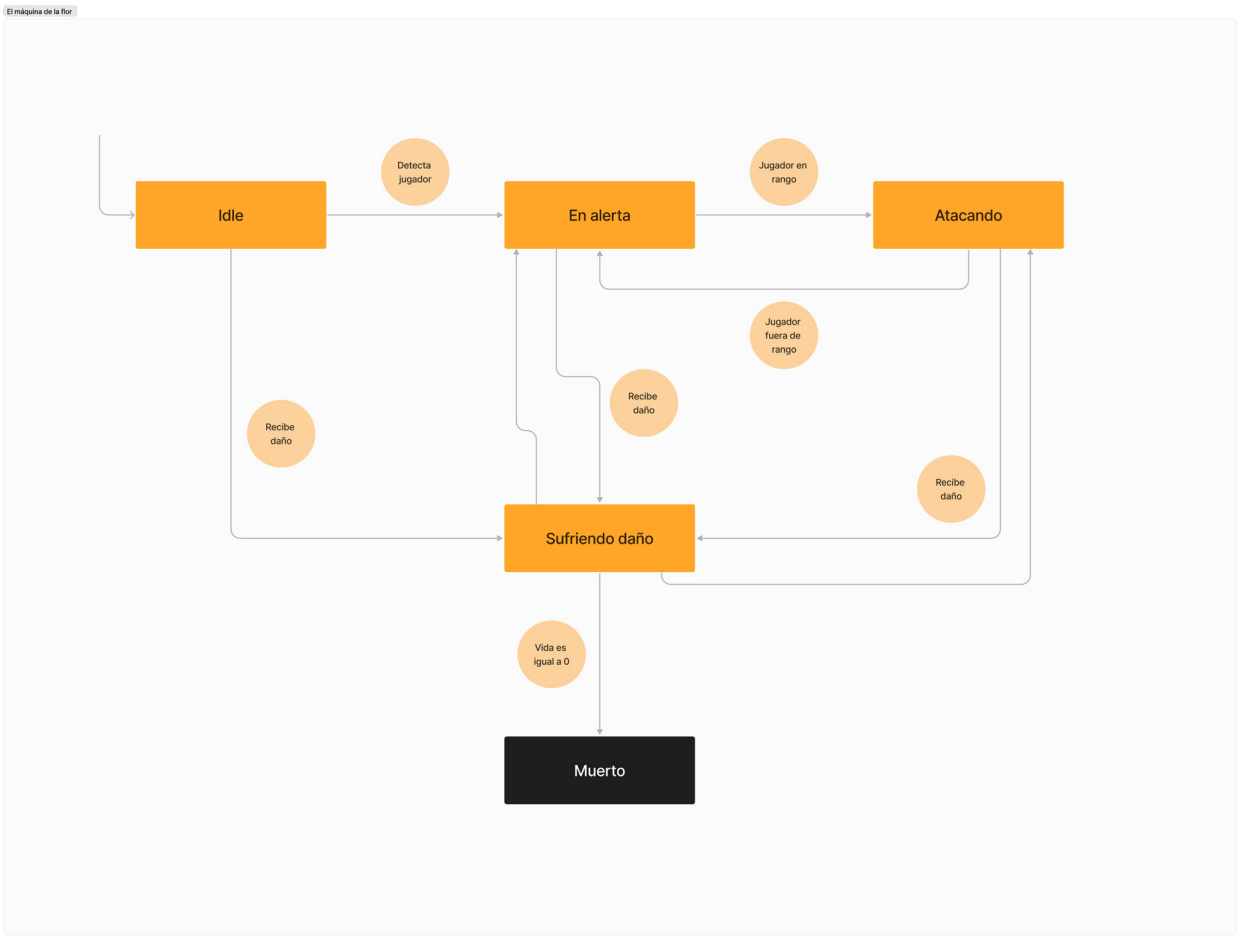


Estados	
Idle	Estado en el cual el enemigo se mueve aleatoriamente en un espacio concreto
Persiguiendo	Estado en el cual el enemigo se desplaza a la posición en la que está el jugador
Atacando	El enemigo quita un punto de vida al jugador
Sufriendo Daño	El enemigo recibe daño
Muerto	Al llegar a 0 de vida se destruye el objeto del enemigo

Eventos	
Detecta Jugador	Cuando el jugador está en el rango de visión del slime
Jugador en Rango	Cuando el jugador está a rango de ataque del slime
Jugador fuera de Rango	Cuando el jugador está fuera del rango de ataque del slime
Recibe daño	Cuando el enemigo recibe cualquier instancia de daño del jugador
Vida = 0	Cuando la vida del enemigo llega a 0

Eventos/Estados	Idle	Persiguiendo	Atacando	Sufriendo Daño	Muerto
Detecta Jugador	Persiguiendo				
Jugador en Rango		Atacando			
Jugador fuera de Rango			Persiguiendo		
Recibe Daño	Sufriendo Daño	Sufriendo Daño	Sufriendo Daño		
Vida = 0				Muerto	

6.2 Diagrama de estados y transiciones de Flor

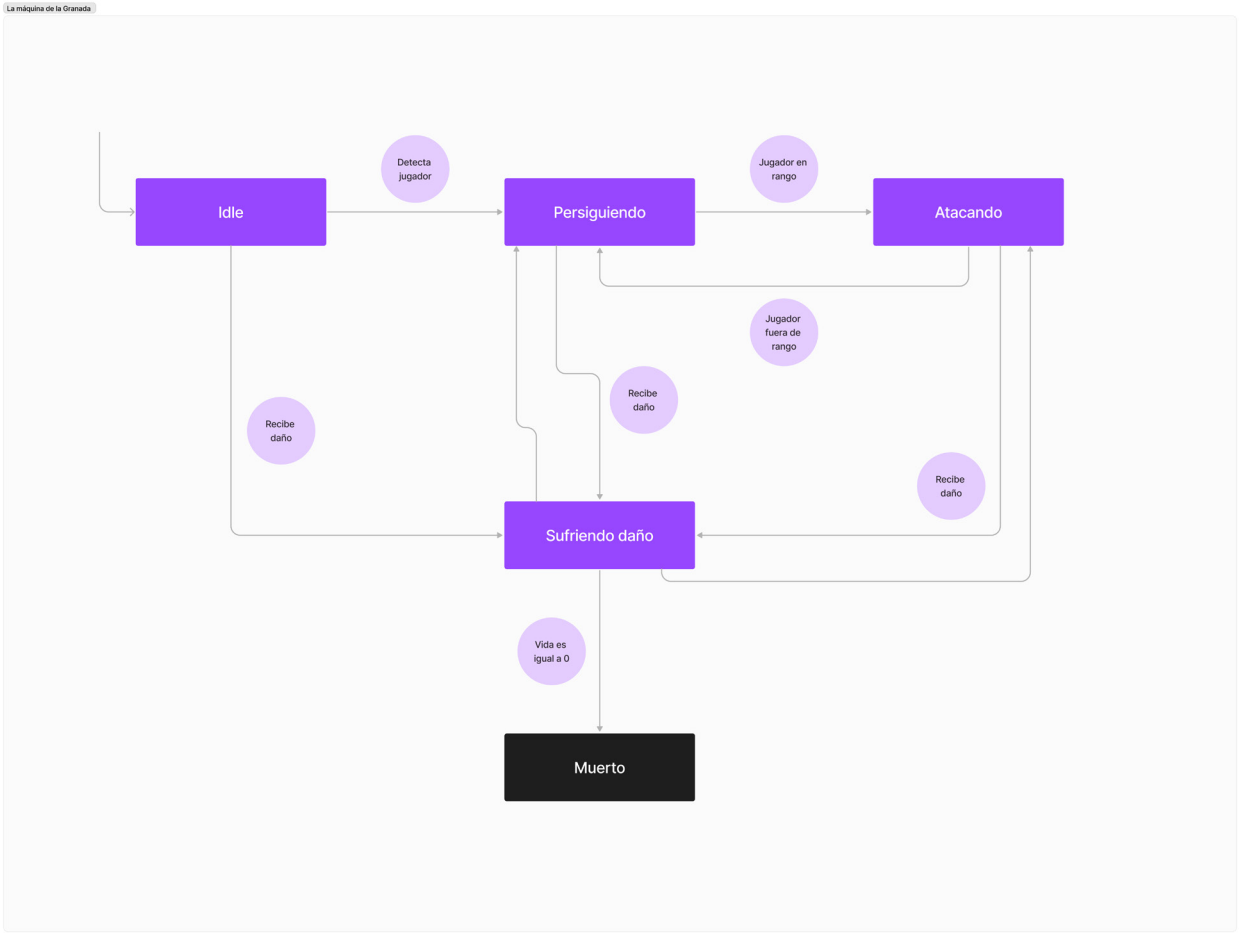


Estados	
Idle	Esta parada contoneándose
En alerta	Estado en el cual el enemigo apunta hacia el jugador
Atacando	El enemigo quita un punto de vida al jugador
Sufriendo Daño	El enemigo recibe daño
Muerto	Al llegar a 0 de vida se destruye el objeto del enemigo

Eventos	
Detecta Jugador	Cuando el jugador está en el rango de visión del enemigo
Jugador en Rango	Cuando el enemigo está a rango de ataque del jugador.
Jugador fuera de Rango	Cuando el enemigo está fuera de rango del ataque del jugador.
Recibe daño	Cuando el enemigo recibe cualquier instancia de daño del jugador
Vida = 0	Cuando la vida del enemigo llega a 0.

Eventos/Estados	Idle	En alerta	Atacando	Sufriendo Daño	Muerto
Detecta Jugador	*Persiguiendo*				
Jugador en Rango		Atacando			
Jugador fuera de Rango			En alerta		
Recibe Daño	Sufriendo Daño	Sufriendo Daño	Sufriendo Daño		
Vida = 0				Muerto	

6.2 Diagrama de estados y transiciones de Granada

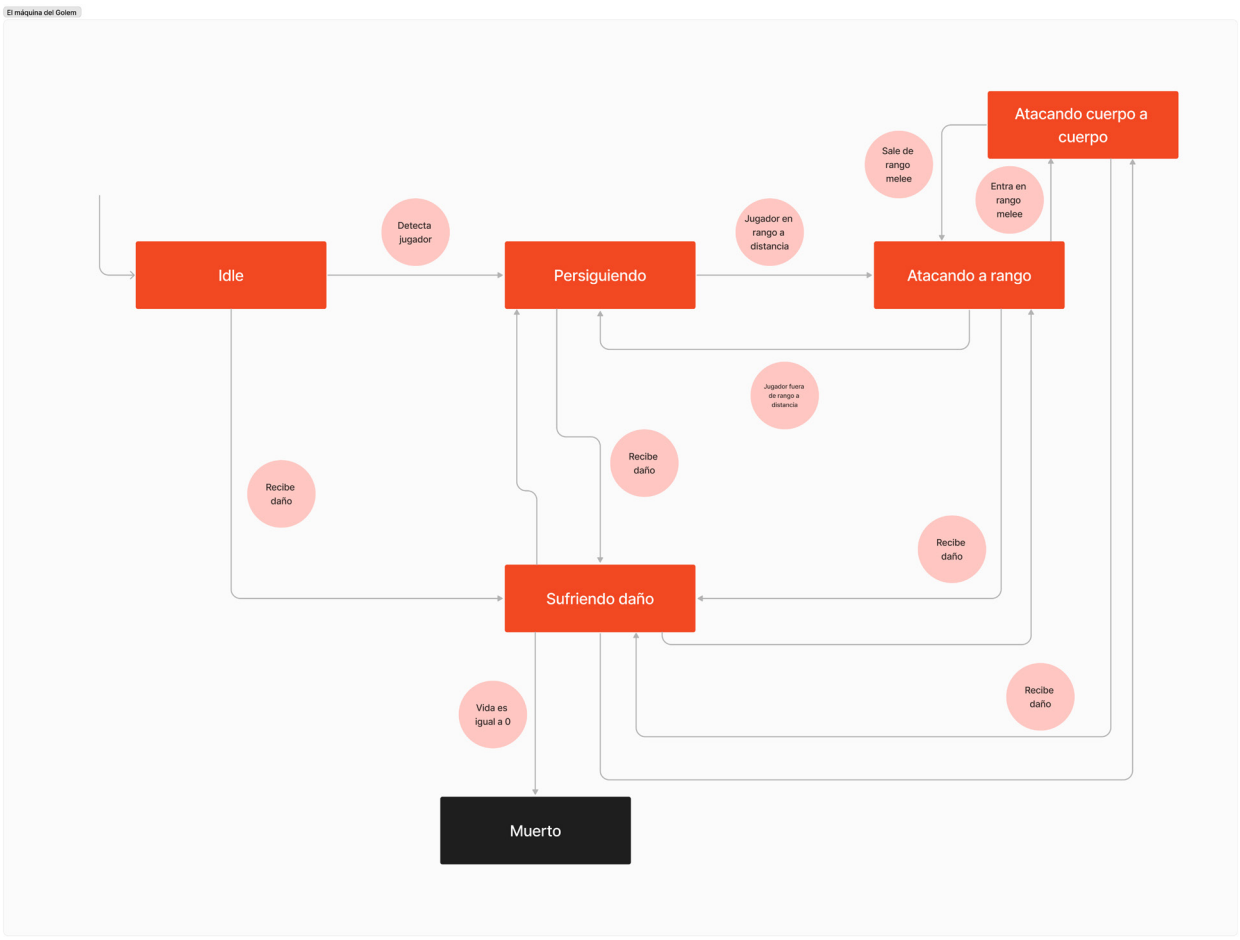


Estados	
Idle	Estado en el cual el enemigo se mueve aleatoriamente en un espacio concreto.
Persiguiendo	Estado en el cual el enemigo se desplaza a la posición en la que está el jugador.
Atacando	El enemigo quita un punto de vida al jugador
Sufriendo Daño	El enemigo recibe daño
Muerto	Al llegar a 0 de vida se destruye el objeto del enemigo

Eventos	
Detecta Jugador	Cuando el jugador está en el rango de visión de la granada.
Jugador en Rango	Cuando el enemigo está a rango de ataque del jugador.
Jugador fuera de Rango	Cuando el enemigo está fuera de rango del ataque del jugador.
Recibe daño	Cuando el enemigo recibe cualquier instancia de daño del jugador
Vida = 0	Cuando la vida del enemigo llega a 0.

Eventos/Estados	Idle	Persiguiendo	Atacando	Sufriendo Daño	Muerto
Detecta Jugador	*Persiguiendo*				
Jugador en Rango		Atacando			
Jugador fuera de Rango			Persiguiendo		
Recibe Daño	Sufriendo Daño	Sufriendo Daño	Sufriendo Daño		
Vida = 0				Muerto	

6.2 Diagrama de estados y transiciones de Golem

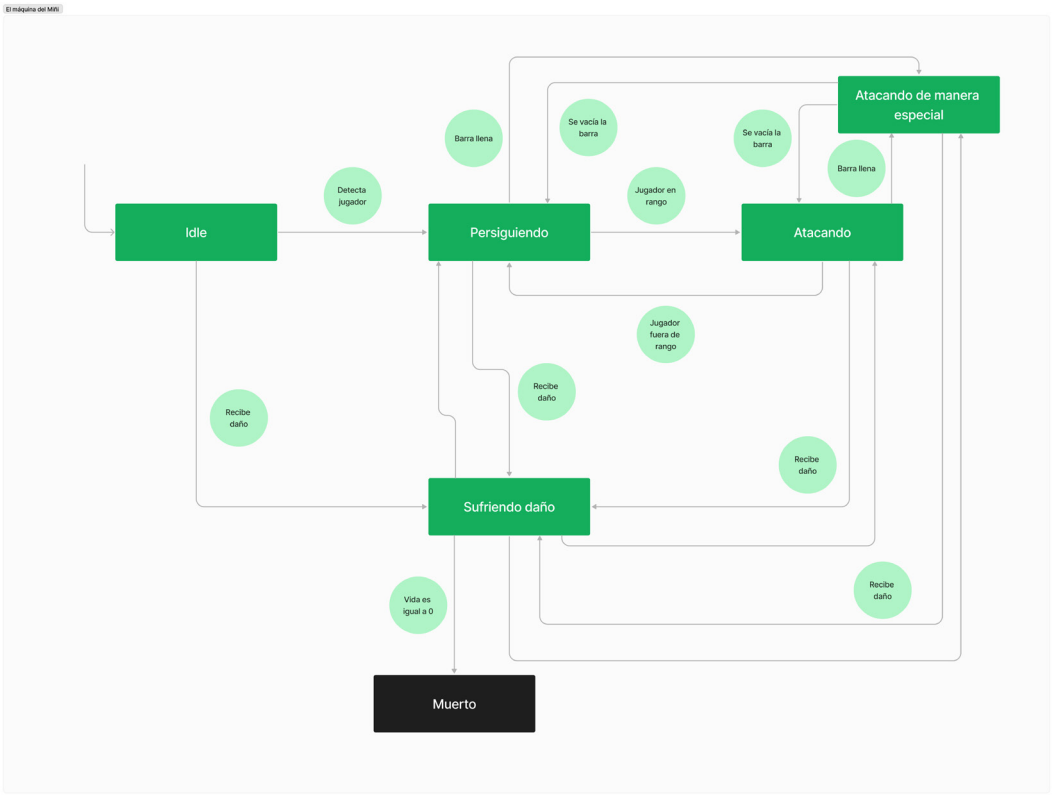


Estados	
Idle	Estado en el cual el enemigo se mueve aleatoriamente en un espacio concreto.
Persiguiendo	Estado en el cual el enemigo se desplaza a la posición en la que está el jugador.
Atacando a Rango	El enemigo quita un punto de vida al jugador a una distancia elevada
Atacando Cuerpo a cuerpo	El enemigo quita un punto de vida al jugador a una distancia corta
Sufriendo Daño	El enemigo recibe daño
Muerto	Al llegar a 0 de vida se destruye el objeto del enemigo

Eventos	
Detecta Jugador	Cuando el jugador está en el rango de visión del enemigo.
Sale de rango melee	El enemigo sale de la distancia de ataque a melee
Entra en rango melee	El enemigo entra en la distancia de ataque a melee
Jugador en Rango a distancia	Cuando el enemigo está fuera del rango de ataque a distancia.
Jugador fuera de Rango a distancia	Cuando el enemigo está fuera de rango del ataque cuerpo a cuerpo.
Recibe daño	Cuando el enemigo recibe cualquier instancia de daño del jugador
Vida = 0	Cuando la vida del enemigo llega a 0.

Eventos/Estados	Idle	Persiguiendo	Atacando a rango	Atacando cuerpo a cuerpo	Sufriendo Daño	Muerto
Detecta Jugador	*Persiguiendo*					
Sale de rango melee				Atacando a rango		
Entra en rango melee			Atacando cuerpo a cuerpo			
Jugador en Rango a distancia		Atacando a rango				
Jugador fuera de Rango a distancia			Persiguiendo			
Recibe Daño	Sufriendo Daño	Sufriendo Daño	Sufriendo Daño	Sufriendo Daño		
Vida = 0					Muerto	

6.2 Diagrama de estados y transiciones de Miñi

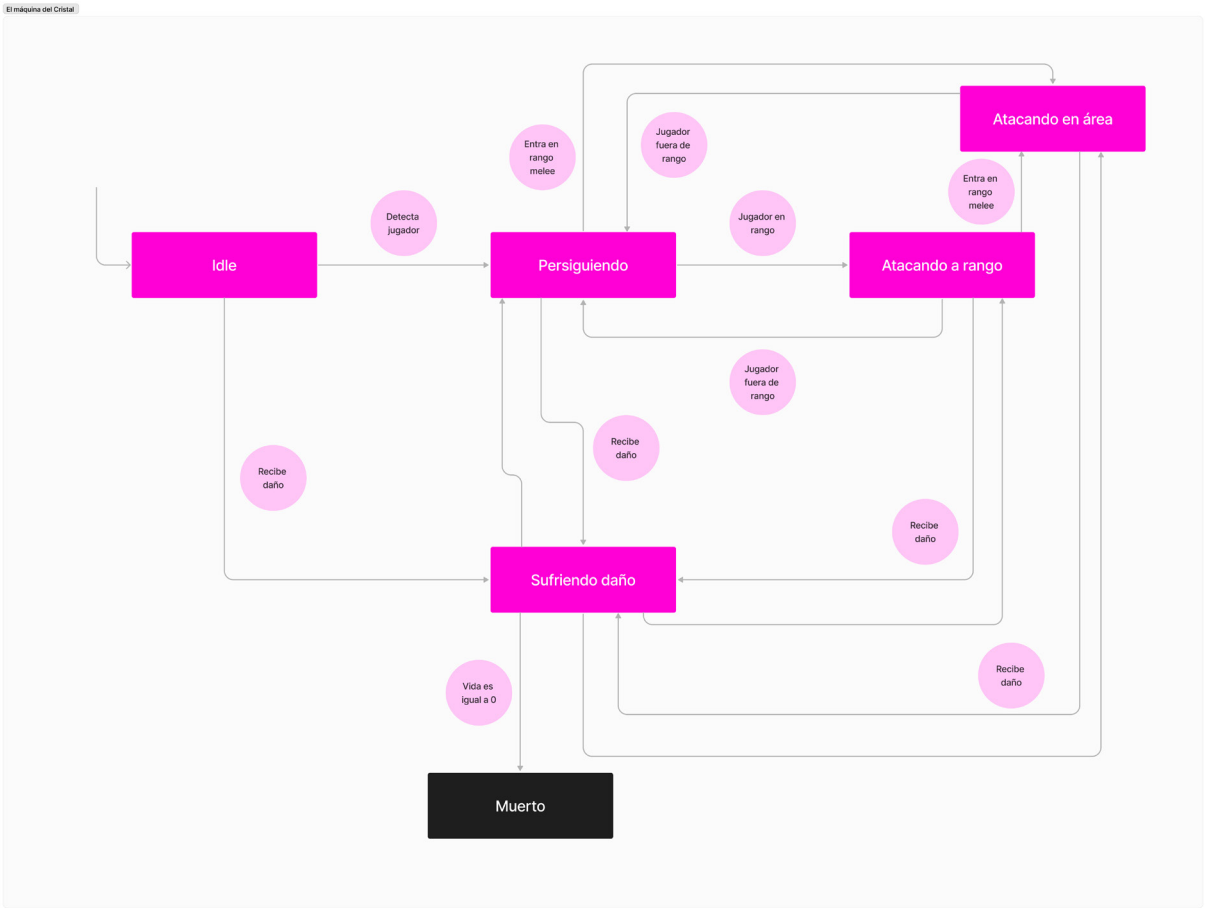


Estados	
Idle	Estado en el cual el enemigo se mueve aleatoriamente en un espacio concreto.
Persiguiendo	Estado en el cual el enemigo se desplaza a la posición en la que está el jugador.
Atacando	El enemigo quita un punto de vida al jugador a una distancia elevada
Atacando de manera especial	El enemigo quita un punto de vida al jugador a una distancia corta
Sufriendo Daño	El enemigo recibe daño
Muerto	Al llegar a 0 de vida se destruye el objeto del enemigo

Eventos	
Detecta Jugador	Cuando el jugador está en el rango de visión del enemigo
Jugador en Rango	Cuando el enemigo está dentro del rango de ataque
Jugador fuera de Rango	Cuando el enemigo está fuera de rango de ataque
Recibe daño	Cuando el enemigo recibe cualquier instancia de daño del jugador
Vida = 0	Cuando la vida del enemigo llega a 0.
Barra llena	La barra de cooldown para el ataque especial se llena
Se vacía la barra	Tras efectuar el ataque especial, la barra vuelve al valor 0 para rellenarse con el tiempo

Eventos/Estados	Idle	Persiguiendo	Atacando	Sufriendo Daño	Muerto	Atacando de manera especial
Detecta Jugador	*Persiguiendo*					
Jugador en Rango		Atacando				
Jugador fuera de Rango			Persiguiendo			
Recibe Daño	Sufriendo Daño	Sufriendo Daño	Sufriendo Daño			Sufriendo Daño
Vida = 0				Muerto		
Barra llena		Atacando de manera especial	Atacando de manera especial			
Se vacía la barra						Atacando

6.2 Diagrama de estados y transiciones de Cristal



Estados	
Idle	Estado en el cual el enemigo se mueve aleatoriamente en un espacio concreto.
Persiguiendo	Estado en el cual el enemigo se desplaza a la posición en la que está el jugador.
Atacando a Rango	El enemigo quita un punto de vida al jugador a una distancia elevada
Atacando en Área	El enemigo quita un punto de vida al jugador en un área determinada
Sufriendo Daño	El enemigo recibe daño
Muerto	Al llegar a 0 de vida se destruye el objeto del enemigo

Eventos	
Detecta Jugador	Cuando el jugador está en el rango de visión del enemigo
Jugador en Rango	Cuando el enemigo está dentro del rango de ataque a distancia
Jugador fuera de Rango	Cuando el enemigo está fuera de rango de ataque a distancia
Jugador en Rango Melee	Cuando el enemigo está dentro del rango de ataque melee
Jugador fuera de Rango Melee	Cuando el enemigo está fuera de rango de ataque melee
Recibe daño	Cuando el enemigo recibe cualquier instancia de daño del jugador
Vida = 0	Cuando la vida del enemigo llega a 0.

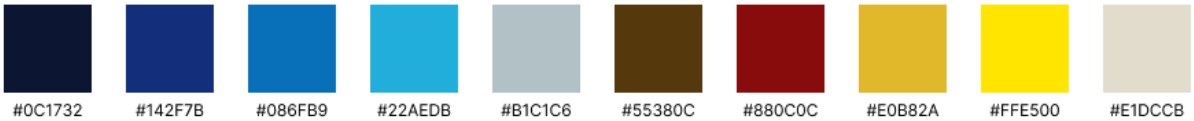
Eventos/Estados	Idle	Persiguiendo	Atacando a distancia	Atacando en área	Sufriendo Daño	Muerto
Detecta Jugador	*Persiguiendo*					
Jugador en Rango		Atacando a distancia				
Jugador fuera de Rango			Persiguiendo			
Jugador en Rango Melee			Atacando en área			
Jugador fuera de Rango Melee				Atacando a distancia		
Recibe Daño	Sufriendo Daño	Sufriendo Daño	Sufriendo Daño	Sufriendo daño		
Vida = 0					Muerto	

7. Estilo visual y gráfico

Para el estilo visual de Unwanted perseguíamos una estética de fantasía oscura en la que se mezclaba una naturaleza exuberante y con elementos primitivos, con elementos de tecnología muy futurista, como serían los androides, creando así una contraposición que llamase la atención por su atrevimiento. Este fue el moodboard inicial del proyecto:

https://www.pinterest.es/ladodelaoscuridad/unwanted/

También decidimos desde el inicio que nuestra paleta se formaría principalmente por azules y ocres, y a partir de aquí creamos esta paleta que hemos ido respetando al máximo en la creación de personajes y escenarios:



Ya que es un juego de fantasía, buscamos integrar elementos que nos acercasen a esa idea: luces para dar un aspecto más “mágico”, enemigos diseñados con aspecto monstruoso pero con elementos humanoides (rozando un uncanny valley), escenarios irreales e imposibles donde la naturaleza es la principal protagonista...

8. Escenarios y entornos

El juego hasta donde hemos creado es un bosque de noche, oscuro, donde escasea la luz y abundan los enemigos. El escenario que principalmente ve el jugador es el de un camino rodeado de abundante vegetación en un valle rodeado de altas montañas. El primer escenario, hasta la caída a la arena 2, es un camino rodeado de árboles que al llegar a la arena se abre hacia un área más amplia y con otros tipos de árboles que te permiten ocultarte o jugar con el entorno durante la pelea. El suelo durante toda esta primera parte es irregular para dar la impresión de ser un suelo de barro o rocoso, y el camino que el jugador sigue simula aquel que podría ser el cauce de un río vacío.

A partir de la arena 2 encontramos un entorno más cerrado bajo tierra donde encontramos raíces que soportan la zona y setas que aportan la vegetación que podría ser propia de este entorno. La textura de esta arena es más árida y rocosa, con una cuesta que abarca toda la zona circular añadiendo dinamismo y un juego de alturas al entorno.

Al salir de este área tenemos de nuevo una transición hacia la superficie pero nos encontramos esta vez en un gran y largo laberinto, integrado dentro de un tronco muerto. Aquí dentro no existe vegetación y resulta en un escenario bastante diferente y simple pero que aporta a la vibra oscura y un pelín tenebrosa que también tiene el juego. El área de combate es este mismo laberinto donde no sabes dónde encontrarás el próximo puñado de enemigos.

A partir de esta arena pasamos al área final del nivel, y si antes veníamos de una zona “muerta” aquí nos encontramos un escenario de “vida” cuando se integran las flores (que hasta ahora no habían tenido nada de protagonismo) y el agua a la estética de nuestro nivel, creando una ambientación que parece ser más suave y delicada que las anteriores, aunque precisamente sea la más mortal porque es donde se encuentra el boss final.

El primer escenario, donde se desarrolla el tutorial, consiste simplemente en una cueva bajo tierra, con pocos elementos y una iluminación cálida en su interior pero fría en la salida. Ya que es un escenario de prueba lo dejamos lo más despejado y simple posible.

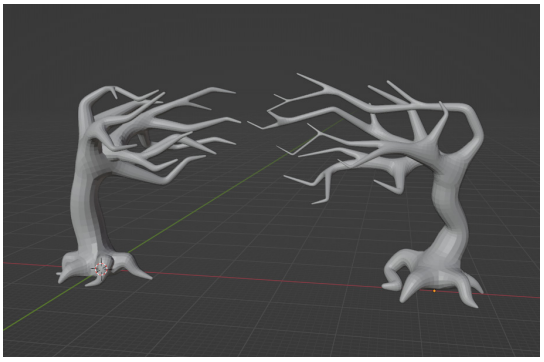
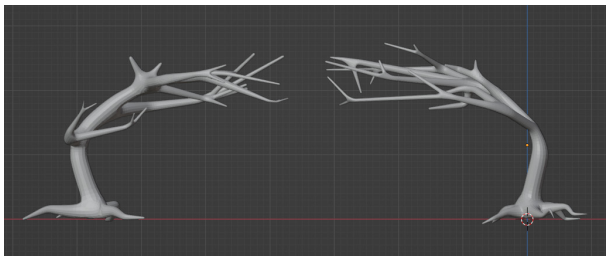
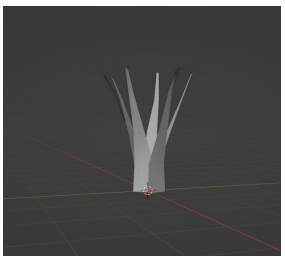
9. Elementos de juego

El fondo del escenario en el primer nivel estará compuesto sobre todo de vegetación (grandes árboles, ramas que sobresalen del suelo, maleza, etc.) y diferentes partículas que servirán de guía para el jugador.

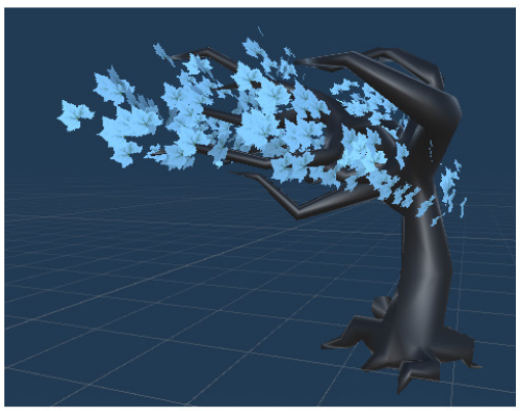
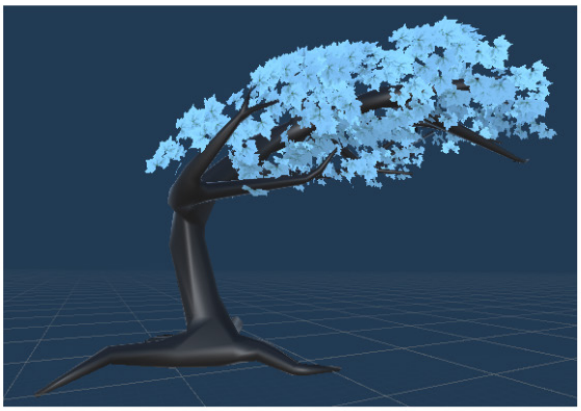
Algunos de los elementos que encontraremos a lo largo de la aventura serán únicamente decorativos o cumplirán una función de “muro” (árboles, acantilados, rocas gigantes, etc.) para que el jugador no pueda salirse de los límites. En cambio,

habrá otros objetos que harán las veces de plataformas y podremos usarlos para desarrollar nuestra estrategia de combate (ramas, rocas, paredes, etc.).

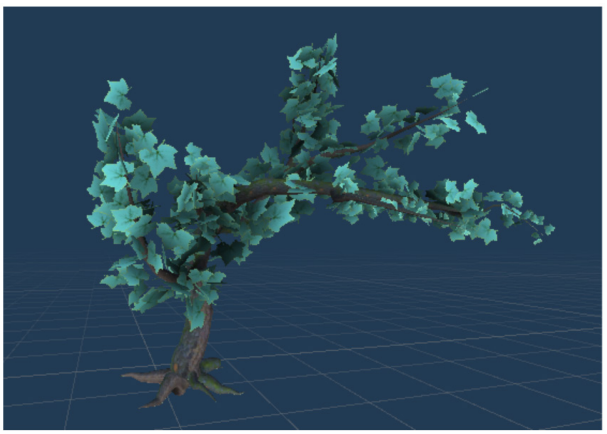
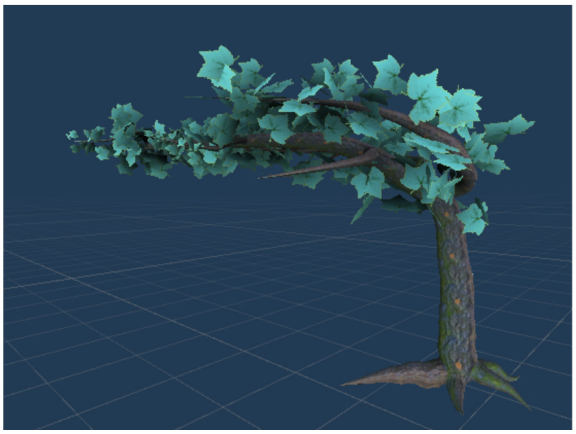
(Como hemos tenido problemas con los árboles, los hemos tenido que rehacer varias veces y de distintas maneras)

Estos árboles son los realizados anteriormente pero con una decimación de polígonos considerable (de 50.000 a 700) y con las copas de los árboles hechas a partir de un sistema de partículas en blender.

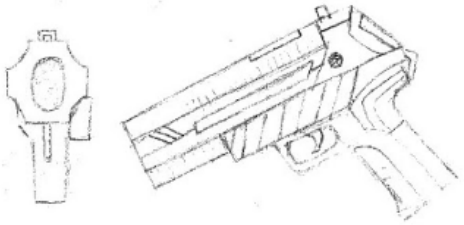



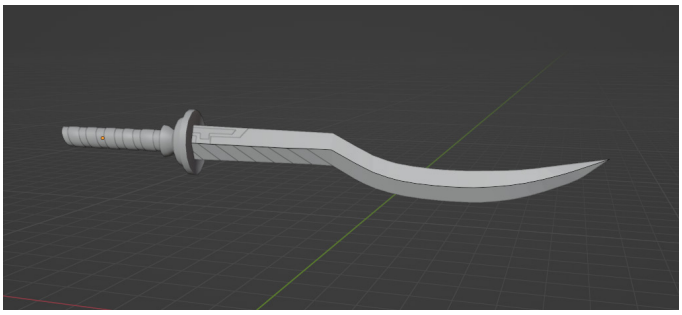
Y estos otros árboles están realizados en Unity a través de la herramienta de Terrain. (Son los definitivos)

Durante casi todo el juego predominará una iluminación oscura, con intención de transmitir un aura de dramatismo postapocalíptico. Además, la paleta cromática de los escenarios variará completamente de un nivel a otro.

En cuanto al combate, nuestro personaje principal entablará los combates usando un arma de fuego (pistola) y una espada (katana). Ambas armas estarán disponibles desde el inicio de la aventura y podrán ser usadas indefinidamente.





Las principales diferencias entre una y otra serán su diferencia en cuanto al alcance, siendo la katana únicamente un arma cuerpo a cuerpo y la pistola un arma a distancia. Además, la katana contará con una mecánica de juego única, que premiará el riesgo del combate cercano con un ligero robo de vida cada vez que golpees con ella. Esto servirá para que los jugadores más habilidosos puedan arriesgar más y vean recompensado su atrevimiento.

Durante el transcurso de los niveles no encontraremos objetos que sean utilizables o que se tengan que gestionar, debido a ello, hemos decidido prescindir de un sistema de inventario.

10. Diseño de nivel

Aquí describiremos el desarrollo de diseño de nivel. Nos hemos basado en este documento para la realización del diseño:

LD - In pursuit of better levels

10.1 Restricciones en el primer nivel

Camino único / dificultad escalante / aprender uso de armas / 5 enemigos distintos (Granada, Slime, Flor, Miñi, Golem) / 1 boss final (Guardián).

10.2 Objetivos en el primer nivel

Escenario tenebroso / oscuridad / juegos de luz / horizontalidad/verticalidad / división en arenas / plataformeo.

10.3 Contexto en el primer nivel

1er nivel (el jugador se adapta) - ayuda a comprensión de controles / algo que enganche.

- **¿Dónde?**

Bosque fantástico.

- **¿Cuándo?**

Noche, mundo devorado por la naturaleza después de guerra.

- **¿Mecánicas?**

Doble salto, dash, ataque 1, ataque 2

- **¿Diferencias?**

Es el primer nivel y cada uno sucede en un bioma, tiene que producir un poco de angustia.

- **¿Adecuación?**

El escenario es amplio, ya que nuestro personaje tiene mucha movilidad.

- **¿Historia?**

El bosque es el sitio más alejado de la torre de Gaia y donde la influencia de los seres de la naturaleza es más fuerte. Todo se basa en lo orgánico y lo natural.

- **¿Comunicación?**

Atender a todas las opiniones, considerar posibilidades con informáticos y tener una idea clara del concepto entre artistas. La música debe ir más adecuada al contexto emocional de Tress y el gameplay que al aspecto visual del nivel.

- **¿Posibilidades?**

Es posible realizar el nivel como queremos, nos limita hasta dónde sepamos hacer.

- **¿Dificultades?**

Creemos que lo más complicado es un buen flujo de nivel y la iluminación en la escena.

10.4 Path

Nuestro Critical Path (el camino más rápido a completar el nivel) y nuestro Golden Path (el camino que esperamos que siga el jugador) son el mismo, ya que no hay ningún tipo de shortcut en nuestro juego.

10.5 Métrica

Tenemos que conseguir armonía entre todos los elementos del nivel:

- **Mundo :**

largo y ancho / distancias entre arenas / obstáculos

- **Jugador :**

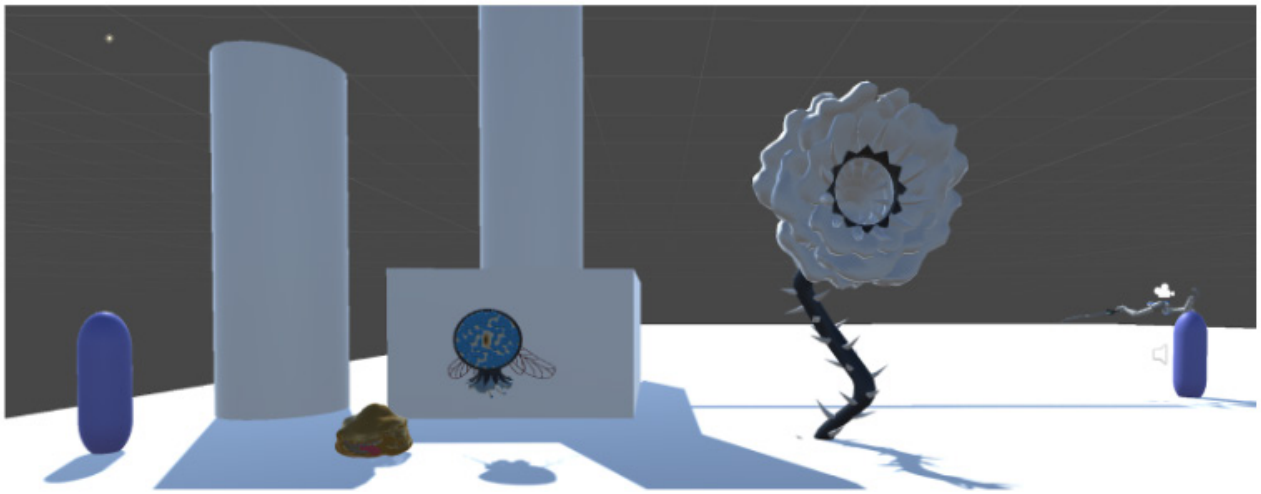
dimensiones / salto (altura y distancia) / ...

- **Armas:**

rangos de ataque - efectividad

- **Enemigos:**

rango visión / rango ataque / dimensiones / movimientos



(“Gym” para probar las relaciones de tamaño entre elementos del mapa)

10.6 Prototipado (diagrama)

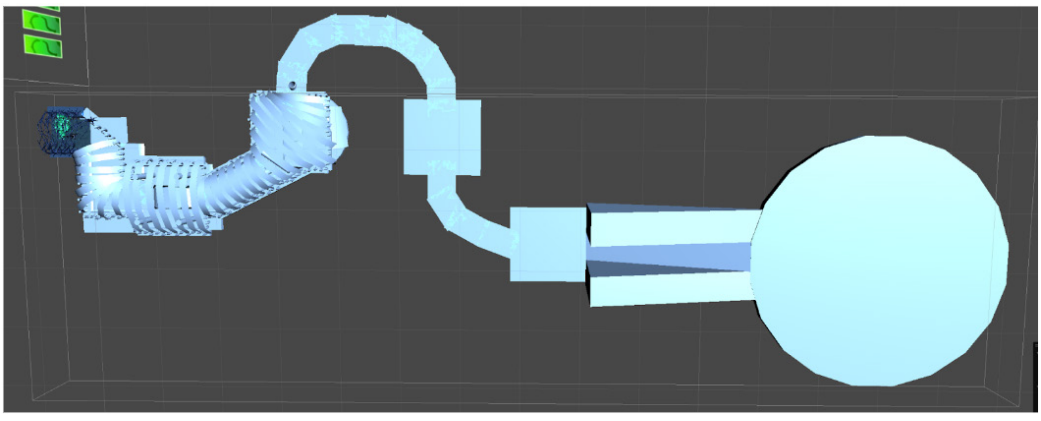
Arena 1: Primera zona, pocos enemigos (Slimes y Flores), para poder probar armas. Estéticamente es una zona sencilla con un relieve de altura y bastante espacio, se centra en la comodidad del jugador.

Arena 2: Slimes, Flores y Granadas, más cantidades pero son los más fáciles de derrotar. Estéticamente es un espacio limitado bajo tierra, que pretende dar la sensación de encierre y un poco más de agobio al jugador. Cuenta con una cuesta en diagonal que también añade verticalidad y produce un poco de vértigo.

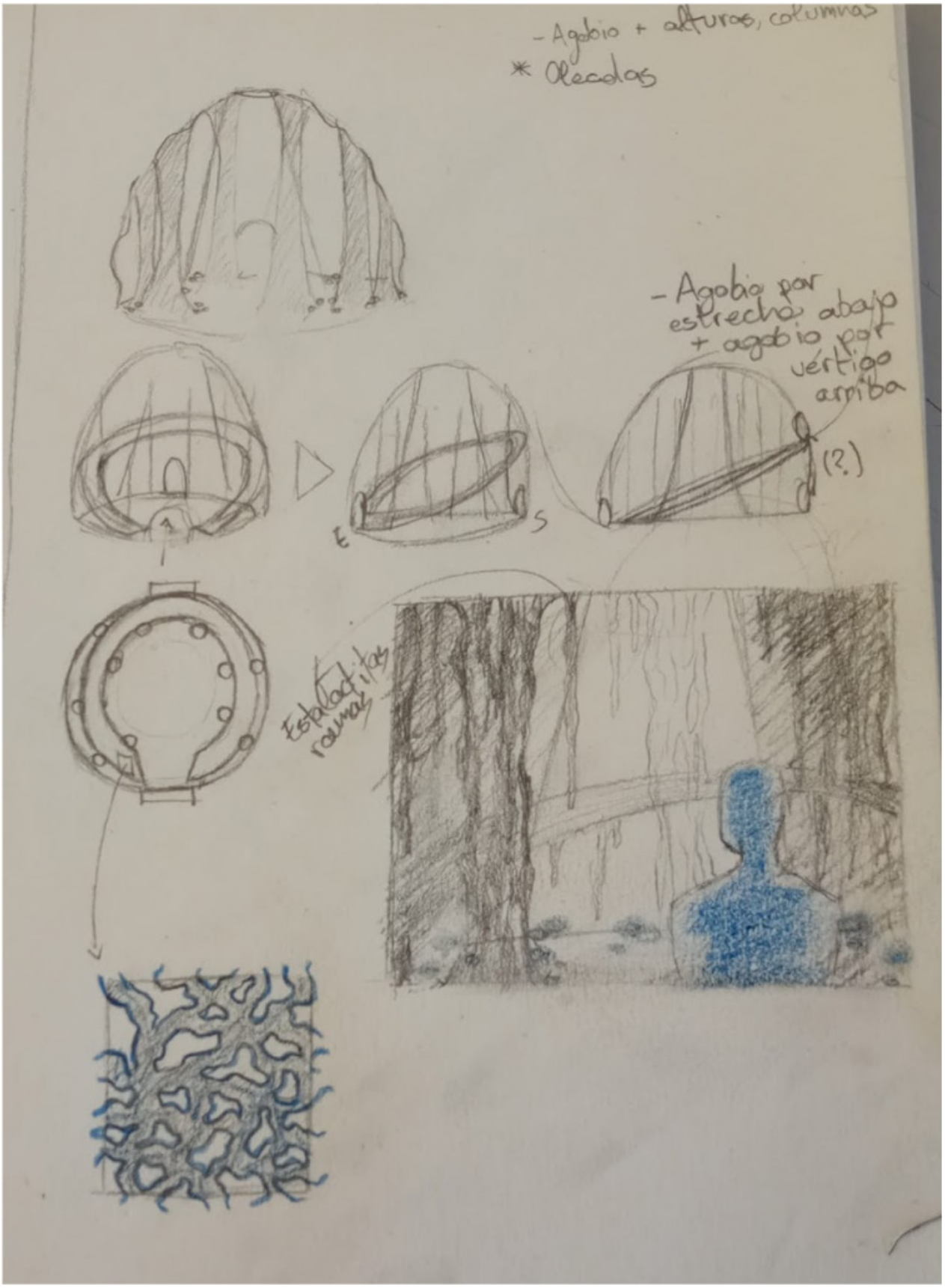
Arena 3: Se añaden los enemigos de Miñi y Golem (más complicados de vencer). Estéticamente es un tronco muerto tumbado en diagonal, y dentro hay un laberinto. Pretende crear diferencia a las otras arenas siendo una arena más interactiva con el entorno y la más angustiante de todas.

Arena 4: Todos los enemigos y en gran cantidad. Estéticamente tiene naturaleza suave (agua, flores) pero es la más agresiva y difícil de todas. Consta de un área superior de forma circular, donde se limita (con montañas, es como un agujero) el movimiento a las paredes (como un donut) y en el centro se encuentra un lago con una enorme flor. Al terminar la oleada en esa zona, se da acceso a una escalera de caracol que entra hacia la tierra y limita el movimiento hacia arriba o abajo, a parte de mucho control ya que hay un agujero en el centro por donde caes al vacío.

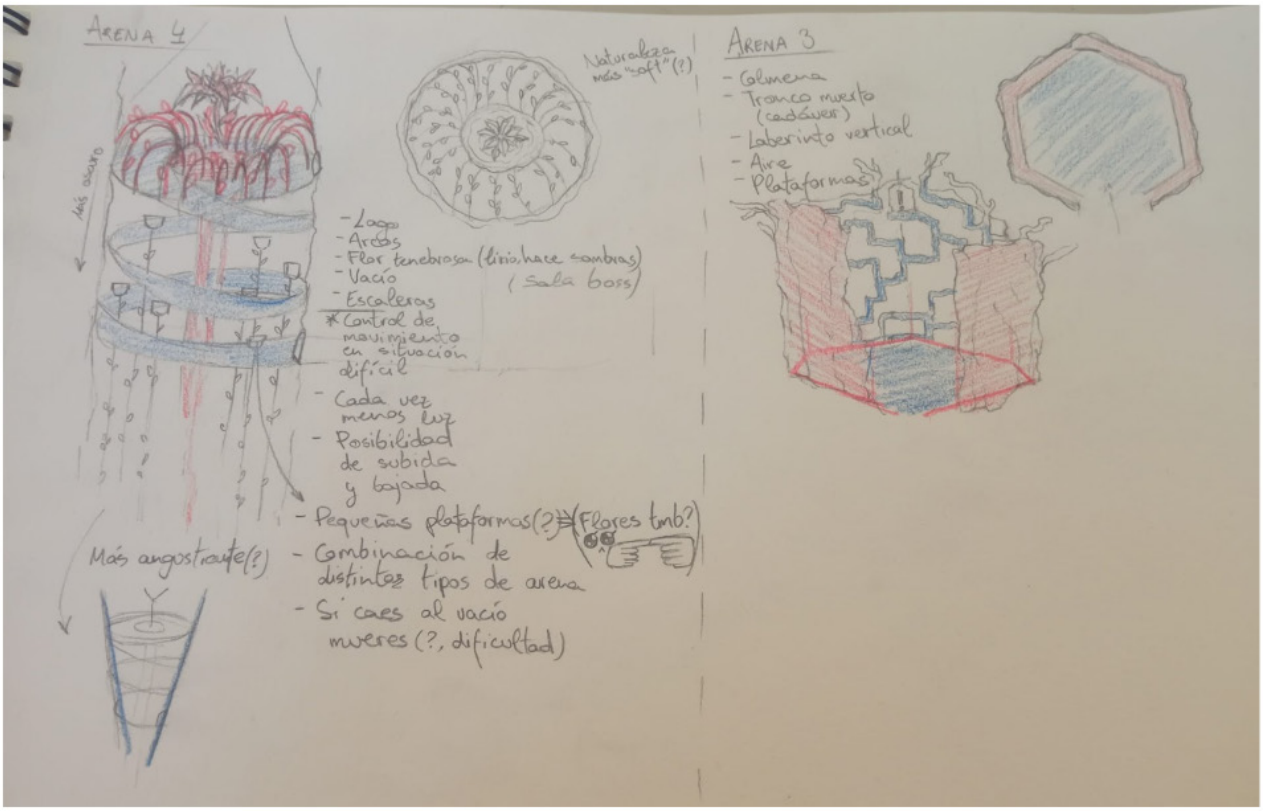
*En nuestro proyecto, ahora que ya está finalizado por la asignatura, hemos tenido que unir el diseño de la arena 4 con el de la arena del boss, manteniendo todos los elementos posibles pero igualmente simplificando la dinámica y posibilidades de esta.



(Prototipado de la estructura de nivel con “blocking”)



(Bocetos de diseño de la arena 2)



(Bocetos del diseño de la arena 3 y 4)

11. Animaciones y efectos especiales

Las animaciones de todo el juego han sido creadas por los encargados de arte del equipo, en el programa que principalmente también hemos usado para el modelado 3D, que es Blender.

Primero, se realizaron las animaciones de movimiento de nuestra protagonista, TRESS, para todos sus posibles estados: Idle, corriendo, salto, dash, ataque de katana, ataque de pistola y recarga de pistola. Estas animaciones están realizadas con los mismos modelos de los brazos y sin utilizar huesos ya que aún no sabíamos cómo trabajar con ellos.

Después realizamos las animaciones para los enemigos. Cada uno de los 5 enemigos cuenta con una animación de movimiento o corriendo, y otra de ataque. Para estas (excepto para el enemigo Granada) sí que utilizamos esqueletos que creamos nosotros mismos a partir de los modelos, colocando los pesos y constraints a mano, y animando cada uno según su tipo y "personalidad":

- Las animaciones del Slime tienen un movimiento similar al de la gelatina, con mucho balanceo y movimiento en la parte superior de su masa, mientras su animación de ataque es un movimiento de mordisco.

- Las animaciones de la Flor tienen un movimiento muy dinámico (ya que su posición es estática) y serpenteante que nos puede dar la impresión de ser una

especie de baile, mientras que su ataque se parece más a la acción de escupir su bala.

- La Granada dió problemas al no haber sido animada con huesos y finalmente solo cuenta con una animación, la de aleteo constante, ya que no cuenta con animación de ataque.

- Las animaciones del Golem cuentan con movimientos lentos pero fuertes, teniendo las animaciones que más se parecerían a aquellas de los humanos, aunque aún así en parte incorrectas. En la animación de caminado intenta simular el andar de un humano, y en la de ataque da una fuerte palmada hacia lo que sería la cabeza del jugador.

- Las animaciones del Miñi son las que cuentan con más huesos y tiene, al contrario de su apariencia bastante oscura, las animaciones más "animadas" o "contentas" del grupo. Su andar es parecido al de un perro o una mascota, lo cual es empatizado por que tiene 3 patas, y puede llegar a ser inquietante por su abundancia de movimiento y aparente felicidad. Por otro lado, su ataque es un movimiento rápido y seco con el brazo que simula ser un corte con una guadaña.

A parte de las animaciones de los personajes, también encontramos otras animaciones como las de las raíces que abren y cierran las arenas, en las que grandes raíces se mueve entrelazándose para bloquear o abrir el camino.

En efectos especiales contamos con las reacciones de los enemigos (un parpadeo en amarillo y un sistema de partículas que explota en forma de esfera), simulando el disparo y el dolor del enemigo pero sin caer en tópicos más grotescos como podría ser una salpicadura de sangre.

También tenemos otro efecto visual muy destacable que es el sistema de partículas de mariposas que guía al jugador por el juego. En este se utilizan modelos 3D de mariposas muy simplificadas que van dejando tras ellos una estela o rastro de tonalidad amarilla que se va desvaneciendo y que a su vez también emite luz, proporcionando un efecto muy mágico.

12. Sonido y música

La banda sonora del juego se compone de 2 canciones:

- "Punch Deck - I Can't Stop" is under a Creative Commons (cc-by) license, Music promoted by BreakingCopyright: https://bit.ly/bkc-stop, la cual es nuestra música de menú de inicio.

- "Somewhere in the night" por el grupo murciano S.N.A.K.E, al cual estamos muy agradecidos por habernos prestado su música sin ningún coste ni condición. Se puede encontrar al grupo en Spotify: https://open.spotify.com/intl-es/artist/79fLTbapuoDUq8lmFtrOax.

Buscamos principalmente una banda sonora de música rock, que tuviese un ritmo lo más constante posible en un compás 4/4 y que fuese un rock no muy ruidoso pero inspirador. Somewhere in the night incluso tiene una letra parecida a las sensaciones y emociones que se representan en el juego. Y es que nuestra elección de música no quería acompañar al ambiente, sino a la perspectiva y situación de nuestras dos protagonistas en el camino.

En la música de menú de inicio encontramos una composición sin voz y bastante diferente a nuestra otra canción elegida, que sirve más de preparación para el jugador, con un ritmo muy constante, lineal y grave.

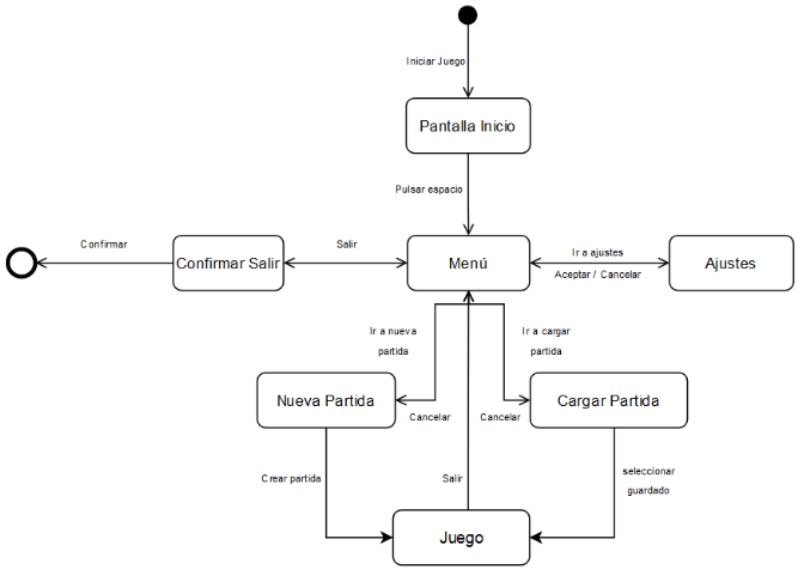
En cuanto a los sonidos, hemos querido experimentar en este campo utilizando sonidos inusuales y distintos al tópico en nuestro juego. TRESS por ejemplo cuenta con sonidos que serían adecuados al movimiento de un robot a partir de sonidos de cafeteras o otros elementos tecnológicos, mientras que los enemigos presentan sonidos un tanto curiosos, mezclando elementos humanoides (el sonido de ataque de la flor es sonido de escupir) y otros totalmente irreales (el sonido agudo, como si estuviese masticando del slime). Se tratan de sonidos que mezclan lo tecnológico y lo muy natural o humano, aportando a ese escenario que también combina estos ambos.

Todos los sonidos son sacados de bancos de sonidos gratuitos (Pixabay en su mayoría) y todos han sido editados previamente a ser introducidos en el juego por nosotros para adaptarlos al ambiente o situación que queríamos representar.

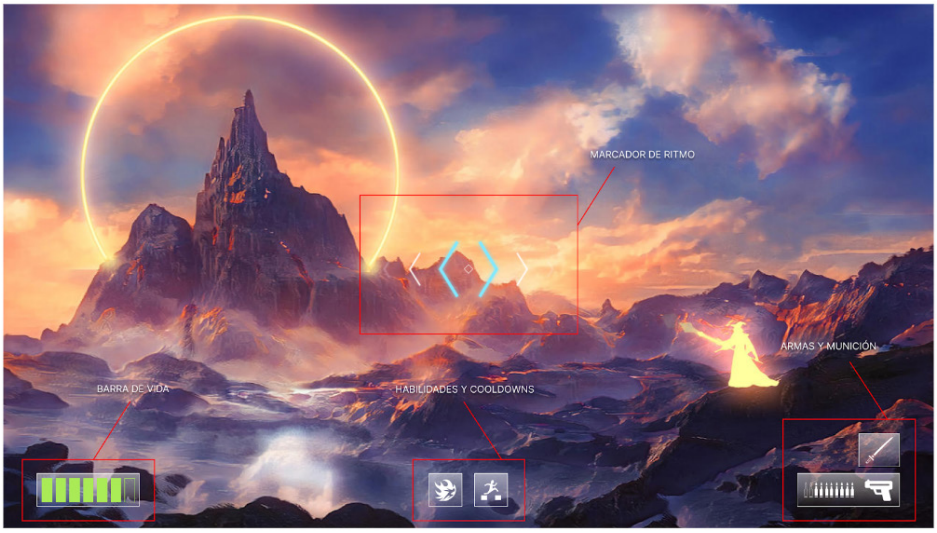
Agradecer la ayuda de Marta Lola López Cosano por ser la actriz de voz de Gaia en la cinemática.

13. Interacción

13.1 Diagrama de navegación



13.2 HUD/OSD



(Esta es la imagen del prototipo del HUD)

(Esta es la imagen del HUD final)

- Indicador de combo : Muestra el nivel de combo del jugador, aumenta al realizar acciones a ritmo, y al subir un nivel de combo aumenta el ataque del jugador. Disminuye al fallar acciones a ritmo y con el paso del tiempo.

- Mirilla: Puntero que indica el lugar al que se está apuntando en todo momento.

- Indicador de ritmo: Barras fijas en pantalla. Cuando la barra de ritmo se superpone con este indicador es cuando se produce el golpe de ritmo. Se cambia en color de este componente a amarillo en el momento en que se efectua una acción que requiere de ritmo en el momento exacto.

- Barras de ritmo: Barras que se desplazan desde los laterales de la pantalla hasta el centro de esta. Son un indicador visual que recibe el jugador sobre el momento en que se producirá el golpe de ritmo, pues este justo coincide con el momento en que las barras se solapan con el indicador de ritmo.

- Dash: Cuando está relleno de color blanco indica que el dash esta disponible, al fallar el dash el indicador se vacia y vuelve a estar disponible con un cooldown.

- Doble Salto: Cuando está relleno de color blanco indica que el doble salto esta disponible, solo se puede hacer si se hace la acción a ritmo.

- Barra de vida: Indicador visual que tiene el jugador respecto a la cantidad de vida que tiene este. Esta barra está dividida en 7 secciones y cada golpe le quita una cantidad de secciones dependiendo del enemigo que golpee al jugador. Se puede rellenar al matar con la katana hasta el máximo de 7 secciones. Cuando las 7 desaparecen, el jugador muere.

- Munición: Indicador visual de la cantidad de balas que le quedan al arma del jugador. El jugador dispone de 8 balas que se representan con un icono individual y desaparecen conforme se van gastando. Al recargar se recuperan todas las balas de golpe.

- Nueva partida: Comienza una nueva partida desde el tutorial de juego

- Continuar: No se ha implementado por falta de tiempo

- Configuración: No se ha implementado por falta de tiempo

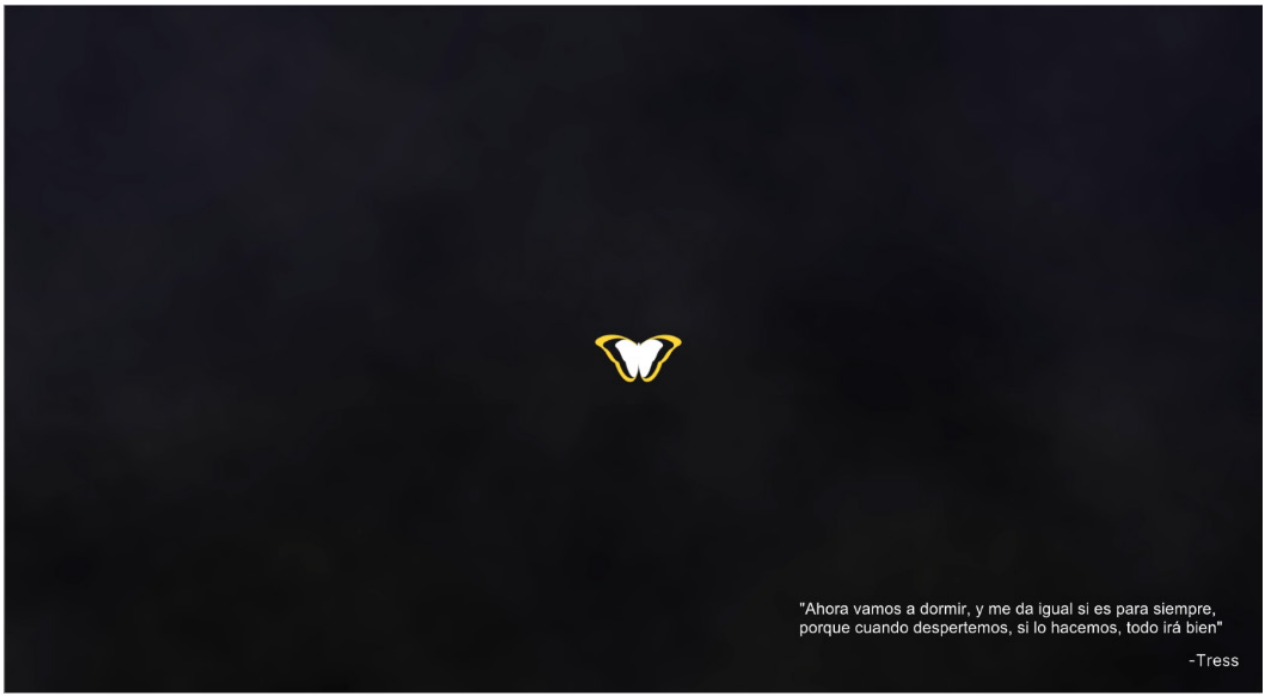
- Créditos: Abre la pestaña de créditos

- Salir del juego: Cierra la aplicación

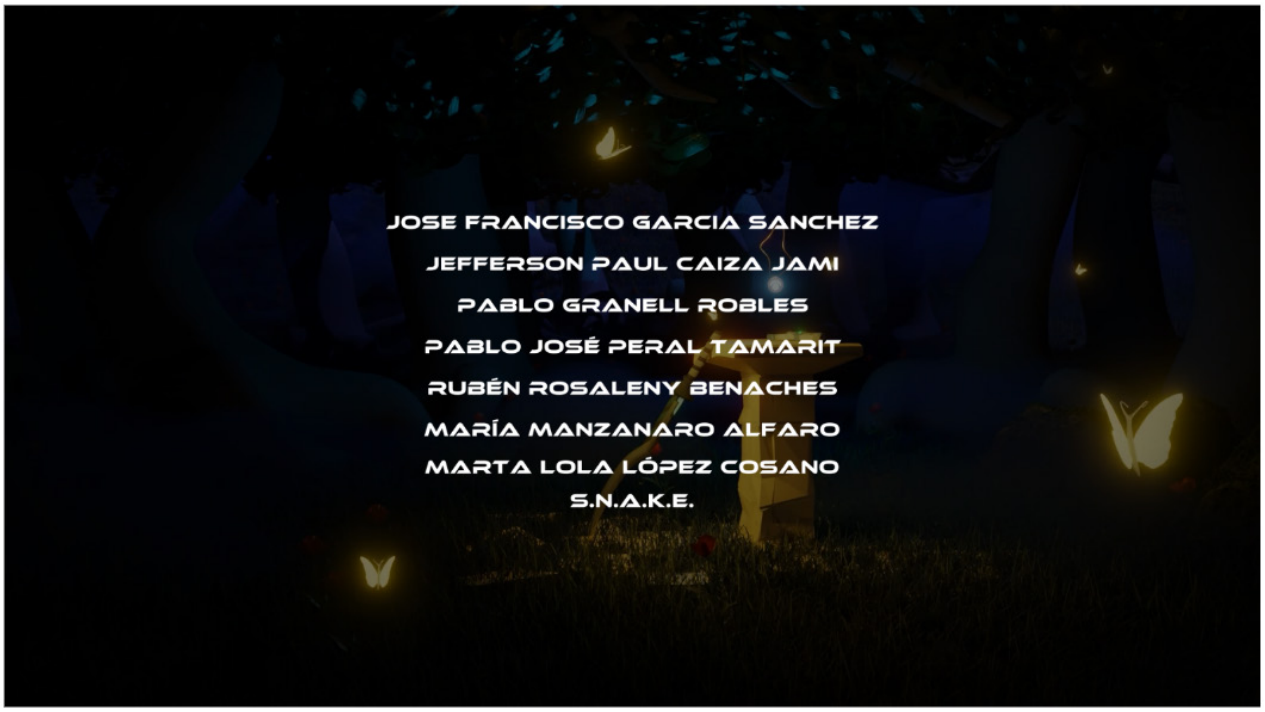
- Reanudar: Cierra el menú de pausa
- Reiniciar: Comienza de 0 el nivel de juego
- Salir: Cierra el juego

13.3 Mapa de Teclas

- Teclado y ratón
- WASD: Movimiento
- Click derecho: Katana
- Click izquierdo: Pistola
- Espacio: Salto / doble salto
- Shift: Dash
- R: Recargar
- Esc: Pausa



(Pantalla de carga)



(Pantalla de créditos)

14. Referencias

- **Estéticas:**

Risk of Rain 2: Estética cell-shading, referencia de personajes.

Slime Rancher: Estética cell-shading.

Sky: Children of the Light: Iluminación.

V.A Proxy: Estética cell-shading.

- **Gameplay:**

BPM: Referencia de juego de ritmo, interfaz.

Metal Hellinger: Referencia de juego de ritmo, interfaz.

Hi-fi Rush: Referencia de juego de ritmo.

No Straight Roads: Referencia de juego de ritmo.

Hades: Referencia de juego frenético.

15. Planificación

Respecto a la planificación, se ha realizado un reparto de áreas de desarrollo basado en las aptitudes y preferencias de los integrantes del equipo:

- Modelado 3D (Rubén y María)
- Texturas (Rubén)
- Animación 3D (Rubén y María)
- Sonido y música (María)
- Diseño interfaz (Rubén)
- Montaje de escena (Rubén y María)
- Diseño del cartel, logo, marca general (Rubén)
- Diseño personajes (María y Rubén)
- Guión (María)
- Partículas (María y Rubén)
- Programación enemigos (Pablo + Jefferson)
- Programación sonido (Todos)
- Programación armas (PJ + Pablo)
- Programación entorno (Jefferson)
- Sistema de combo (PJ)
- Sistema de navegación de enemigos (Pablo)
- Programación protagonista (Todos)
- Optimización (Jefferson)
- Organización (PJ)
- Solucionado de errores (Josefran + Jefferson)
- Programación física (Pablo + Jefferson + María)
- Programación interfaz (Josefran + Jefferson)

- Control de versiones (Pablo)
- Restricciones al jugador (PJ)

*Mucha de esta planificación ha ido cambiando y progresando según las necesidades y complicaciones del desarrollo, y no es fiable como prueba de qué ha realizado cada uno. Eso será especificado en el informe de tareas.

16. Relación de hitos

16.1 Funcionalidad Desarrollada

- Desarrollo de sistemas de sonido generales y específicos.
- Modelado de elementos del escenario y texturización del primer nivel.
- Creación e implementación de enemigos como el Slime, la Granada y la Flor.
- Integración de música y efectos sonoros principales.
- Desarrollo de la iluminación del primer nivel.
- Sistema de movimiento del jugador, que incluye acciones como disparar, recargar, y el uso de la katana.
- Implementación de un sistema de combo y sistema de puntos.
- Sistema de armas, de vida y de daño a los enemigos.
- Integración de la mayoría de los elementos.
- Diseño y desarrollo de HUD.
- Implementación de menús, como el menú principal y de pausa.
- Desarrollo y pulido del sistema de armas, incluyendo tiempos, daño y restricciones.

30 de noviembre

- Enemigos de flor y granda atacan con patrones de ataque diferentes.
- Pulido general el NavMesh Surface de cada enemigo, ahora es más tolerante a artefactos del terreno.
- Sistema de oleadas completo, ahora existe una zona de combate como tal.

- Sonidos básicos implementados, faltan los sonidos de ambiente y cuadrar la canción.
- Optimización de las mallas, desde blender a lo largo de todo el pipeline las mallas tienen ahora menos triángulos (10M total max.)
- Optimización de las iluminación, ahora se hacen todas al final del frame.
- HUD casi totalmente implementado, falta animar las barras de ritmo, la mirilla y el contador de combo. Todos los elementos están realizados pero no implementados.
- Unificación de todo el mapa en la herramienta terrain, mejora el rendimiento y nos permite tener un LOD efectivo y singular.
- Reiteración de los árboles, esto incluye su estructura, sus mallas y su iluminación.
- Completado el diseño del mapa, falta implementar las 2 últimas arenas y el boss.
- Diseño completo de la interfaz completo y realizado.

Entrega final

- Implementada escena inicial con tutorial.
- Implementado cambio de escena de tutorial al nivel inicial.
- Creada e implementada cinemática inicial.
- Sistema de combo perfilado.
- Realizados e implementados enemigos Miñi y Golem.
- Realizadas e implementadas animaciones de los enemigos.
- Realizadas e implementadas animaciones jugador.
- Sistema de pathing para el sistema de partículas guía.
- Implementadas arenas 3 y 4/boss.
- Mejora de los árboles en Unity.
- Añadido más Terrain.
- Mejora del funcionamiento de la katana.
- Implementación y finalización de la interfaz con el sistema de combo, con todos los demás elementos también funcionales.

- Mejora en el feedback de los enemigos con parpadeo y sistema de partículas.
- Sonidos implementados en su totalidad.
- Implementados cierres de la arena con animación propia (raíces).
- Terminado menú de inicio con animación en bucle.
- Añadidos más elementos a escena.
- Terminada y optimizada la iluminación .
- Implementada nueva oleada de enemigos en arena 3.
- Creada nuevas interfaces para tutorial y final de la demo.
- Ajustado juego a plataforma Steam Deck.
- Añadidos viento y niebla a la escena.
- Mejoras generales en aspecto (texturas) y funcionalidad de los enemigos.

16.2 Dificultades Observadas y Soluciones Aplicadas.

- Pulir de colliders de enemigos y la corrección del movimiento del slime.
- El sistema de navegación de enemigos no es muy intuitivo. Lo solucionamos con buen código y una administración inteligente de los recursos.
- La relación del sistema de físicas y los tiempos con los fotogramas por segundo. Hubo que investigar bastante y aplicamos soluciones fotograma-independientes.
- La cantidad de triángulos, sobre todo en los árboles.
- La conversión de assets desde blender a unity. Lo solucionamos seleccionando la opción “X-forward =Y Up” a la hora de exportar FBX.
- Unity tiene sus propias formas de trabajar así que hemos tenido que unificar la rutina de trabajo entre BBAA e informática.

30 de noviembre

- Los enemigos daban problemas al tener un collider separado de ellos, para ello hemos activado una función el el FBX que nos permite editarlos en tiempo de ejecución.
- Para el NavMesh Surface hemos cocinado uno general para todo el mapa, además hemos ajustado la detección de obstáculos para que sea más permisiva.
- El sistema de oleadas crea un zona impasible mientras queden enemigos en esa

zona y la destruye al terminar el combate.

- Los sonidos se reproducen en un audio source que se mueve con el jugador, cada función que necesita audio lo importa.

- La optimización de las mallas es un problema que abarca una gran cantidad de apartados, desde blender a los ajustes de unity:

  1. Reducción del detalle de los modelos (mid poly)

  2. Reducción de la cantidad de triángulos en los FBX.

  3. LOD basado en lejanía e importancia.

- Respecto a la iluminación hemos encontrado que es un problema complejo, y que la precisión condena su rendimiento:

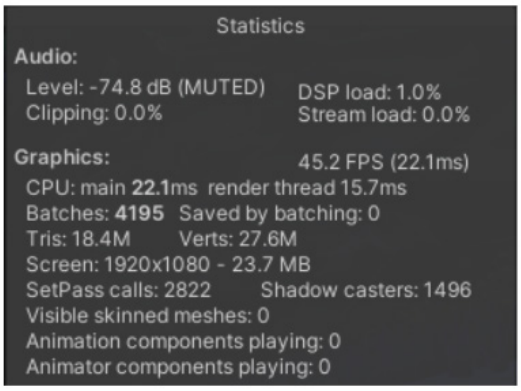
  1. Las tecnologías actuales plantean un enfoque en tiempo real y preciso (ray(path) tracing), pero nosotros no podemos permitirnos esto.

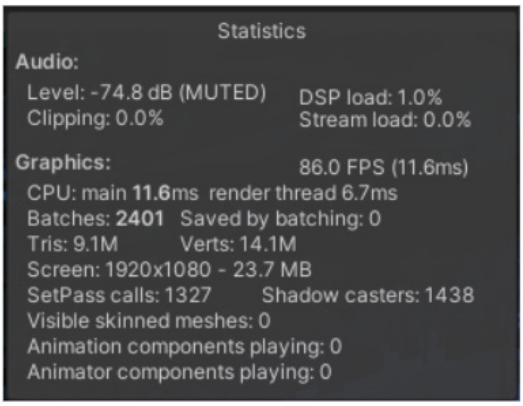
  2. Por esto hemos preferido cocinar las sombras y renderizar todas las sombras por separado al final del frame, aunque esto le quite precisión.

  3. El sistema de renderizado en diferido renderiza la geometría y texturas sin luz y luego hace una última pasada (como haría con la niebla no volumétrica) a cada frame para crear la luz.

  4. Esto es muy interesante cuando tenemos muchos focos de luz, ya que tienen sus apartado especial donde Unity puede simular la iluminación tomándose la libertad de bajar la precisión.

  5. El resultado es un ambiente que simula el anterior con la mitad de coste de CPU en milisegundos.





(Antes vs después)

  6. La interfaz está ahora completamente implementada en con el sistema de canva de Unity, por lo que ahora todo el juego escala a cualquier resolución y ratio de aspecto.

7. Los árboles, los arbustos y el LOD están ahora implementados junto con la herramienta de terreno de Unity, esto nos permite utilizar impostores y carteles para los árboles lejanos y tener un LOD medio para los árboles y la vegetación.

8. Los árboles han sido rehechos desde el punto de vista del mid-poly y el rendimiento, ahora usan la herramienta de árboles de Unity, esto nos da más granularidad y la posibilidad de poder poblar más rápidamente.

9. El mapa está planificado y construido, solo falta implementarlo en Unity.

10. La interfaz está terminada, solo tenemos que cambiar el fondo.

Entrega final

- Cambio del sistema de ritmo

  1. Anteriormente, para marcar el ritmo, se utilizaban unos tiempos aproximados constantes. Esto partía de la base de que el ritmo de la canción era constante, pero al ser grabada con instrumentos reales siempre presentaba pequeñas variaciones que desfasaban el ritmo constantemente (además de redobles, pausas, etc).

  2. Para la mejora de precisión se utilizó VampPlugins, un sistema de plugins para Audacity que permite extraer información de pistas de sonido. En este caso se utilizó para obtener los tiempos exactos donde se producían los pulsos de ritmo de batería de la canción.

  3. Posteriormente, con los tiempos mejorados, tan solo tuvimos que modificar el sistema de ritmo para que esperase dichos tiempos entre pulso y pulso.

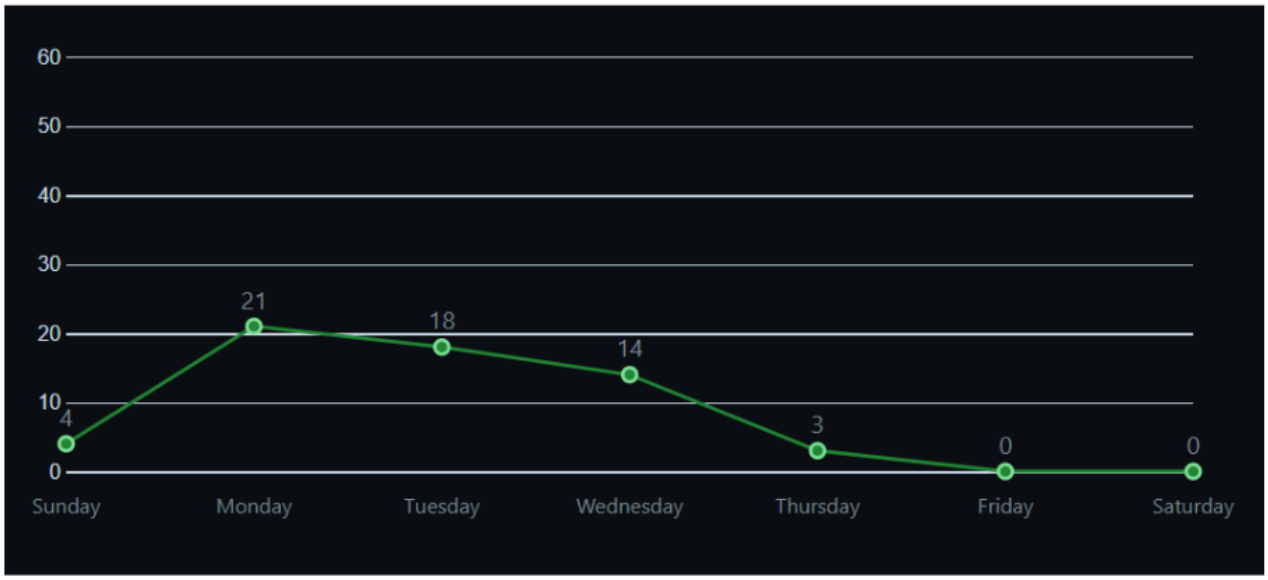
- Arreglados colliders de los enemigos antiguos y nuevos

- Ajustado sistema de voz al nivel de combo, cuanto más combo más alto se escucha la voz y esta solo se empieza a escuchar pasado el multiplicador de combo x1.

17. Resultados Feria de proyectos

La feria de proyectos nos presentaba la posibilidad de conseguir valioso feedback de un público general, variado y objetivo. Nos centramos en presentar la mayor posibilidad de plazas de juego (teníamos dos portátiles y la Steam Deck disponibles) y de asegurarnos de que la gente que viniese a probar el juego tuviese recompensa por darnos este feedback (llevamos pegatinas personalizadas del juego, caramelos y galletas personalizadas también con el logo del juego. Todo esto se puede ver en los anexos). Mucha gente quiso probar el juego y las respuestas que recibimos fueron sinceras pero positivas, lo cual apreciamos mucho. A partir de aquí modificamos el plan de trabajo para añadir estos nuevos cambios y sugerencias a lo que quedaba por hacer. En el informe de la encuesta de la Feria de proyectos se encuentra información más detallada sobre esto.

18. Estimación de Tiempo



Quedamos los lunes presencialmente para trabajar y discutir ideas y problemas vigentes. Como se puede ver en el gráfico después del lunes continuamos trabajando, solucionando los problemas y añadiendo nuevas funciones. Los fines de semana eran de descanso. Le hemos dedicado una media de 3 horas presencialmente a la semana y 15 horas de media a la semana de trabajo telemático.

La estimación de tiempo no ha cambiado, seguimos quedando los lunes y descansamos los fines de semana. Intentamos implementar el control por mando y funcionaba bien, con algunos matices, pero también intentamos hacer una reestructuración del código para que fuera más legible y eficiente lo que nos llevó a condiciones de carrera.

Para solucionar esto hicimos un rebase hasta un commit que fuera totalmente funcional (borrando por el camino el control por mando) y aplazamos esta reorganización para cuando el juego esté terminado, ya que algunos integrantes del grupo de desarrollo de Unwanted van a usar el proyecto como punto de partida para el TFG, así que es interesante que sea lo más claro posible.

19. GitHub

Solo los profesores tienen acceso al repositorio privado. Si necesitáis añadir a alguien no dudéis en preguntar.

https://github.com/pablogranell/Unwanted

Ief0X -> Jefferson Paul Caiza Jami

Wekkax -> María Manzanaro Alfaro

Moski-24 -> Jose Francisco García Sánchez

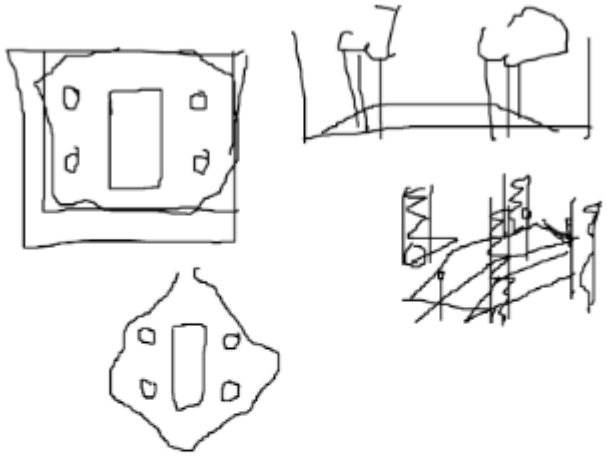
PolloTeriyakii -> Rubén Rosaleny Benaches

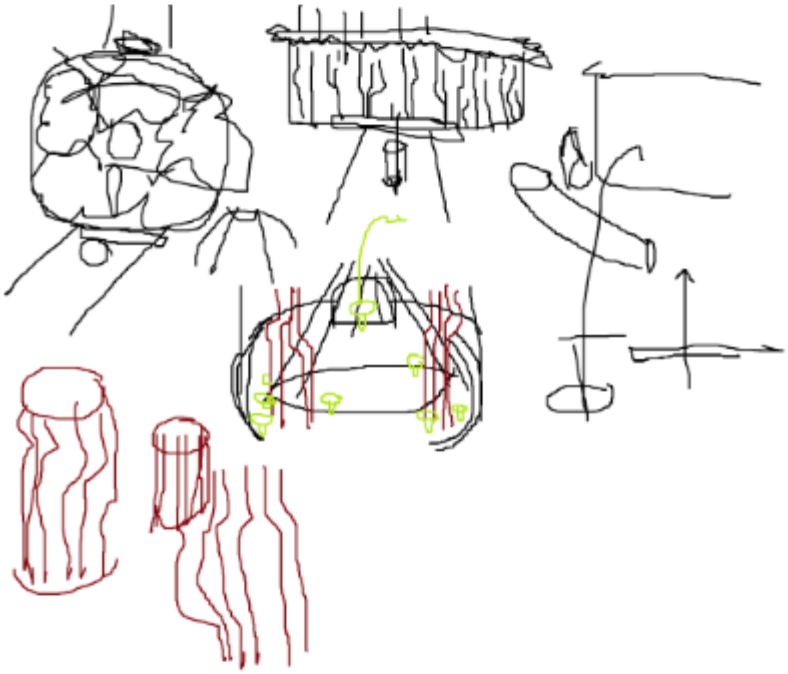
pablogranell -> Pablo Granell Robles

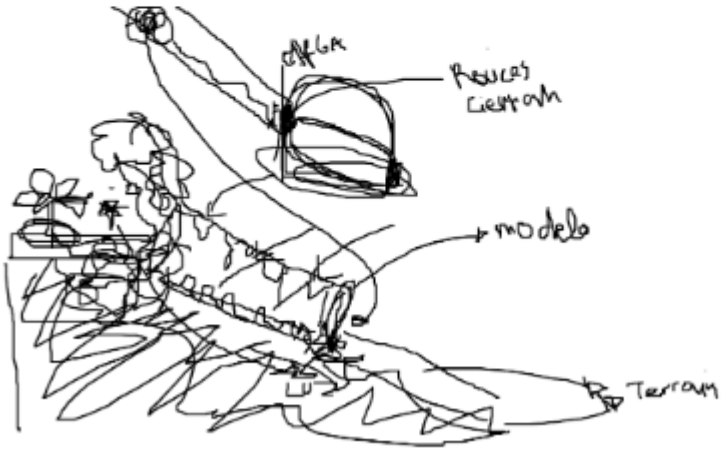
Pablo932002 -> Pablo José Peral Tamarit

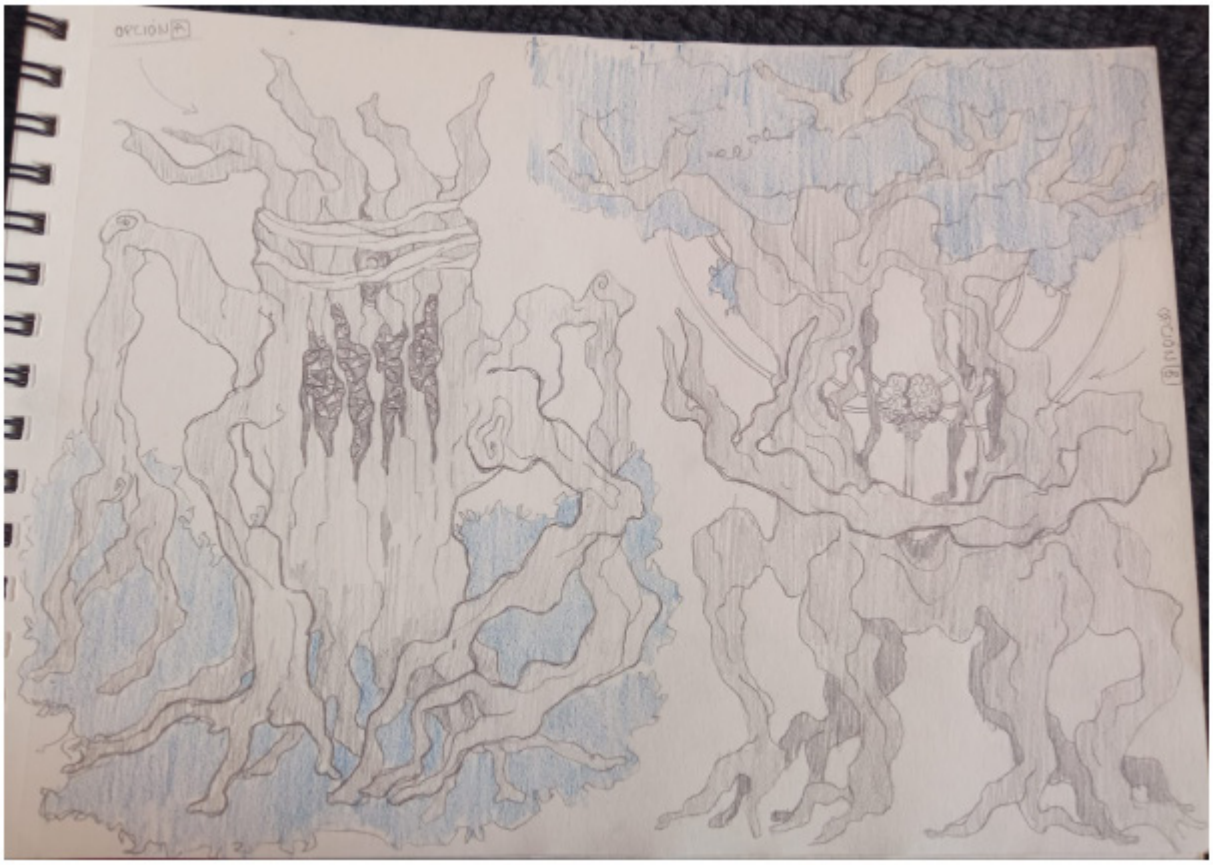
20. Anexos

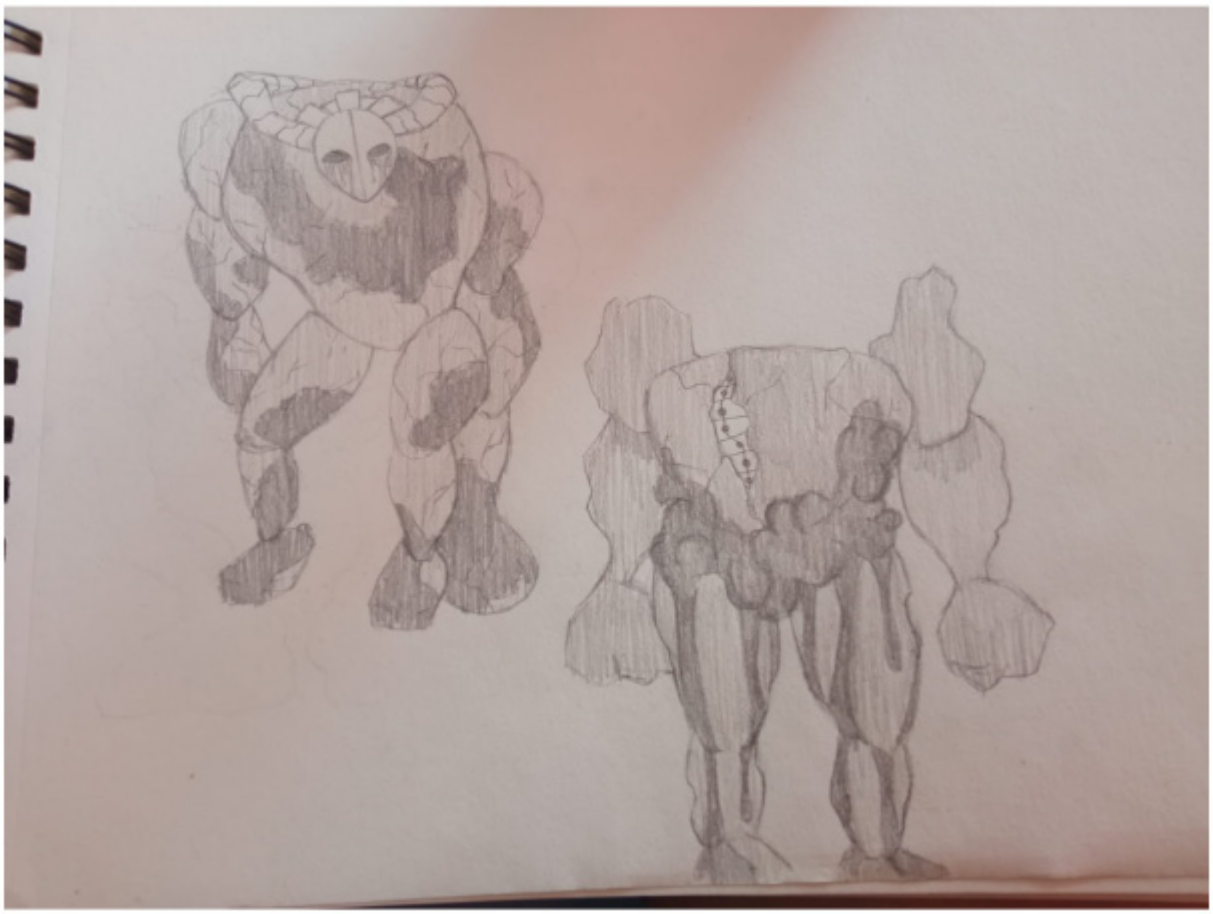
20.1 Bocetos

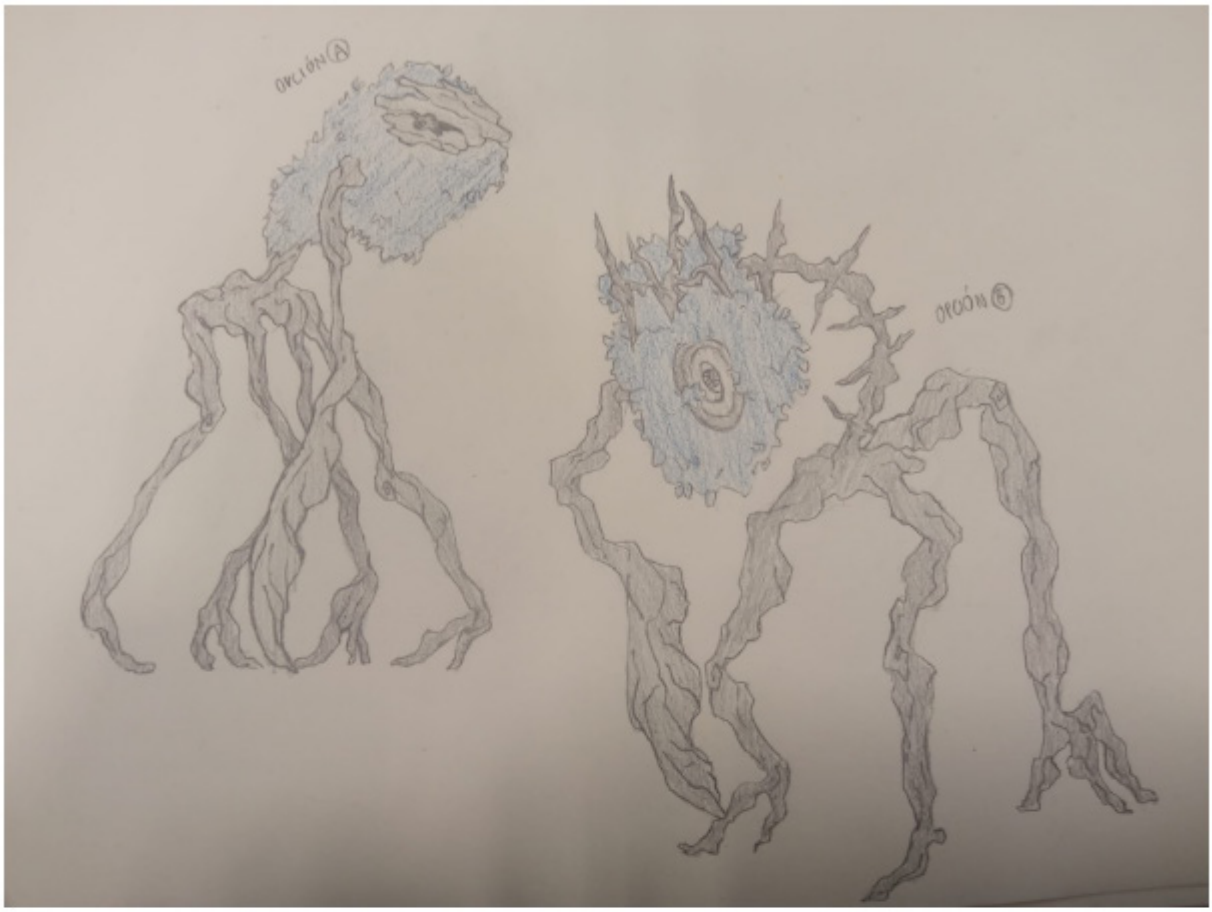


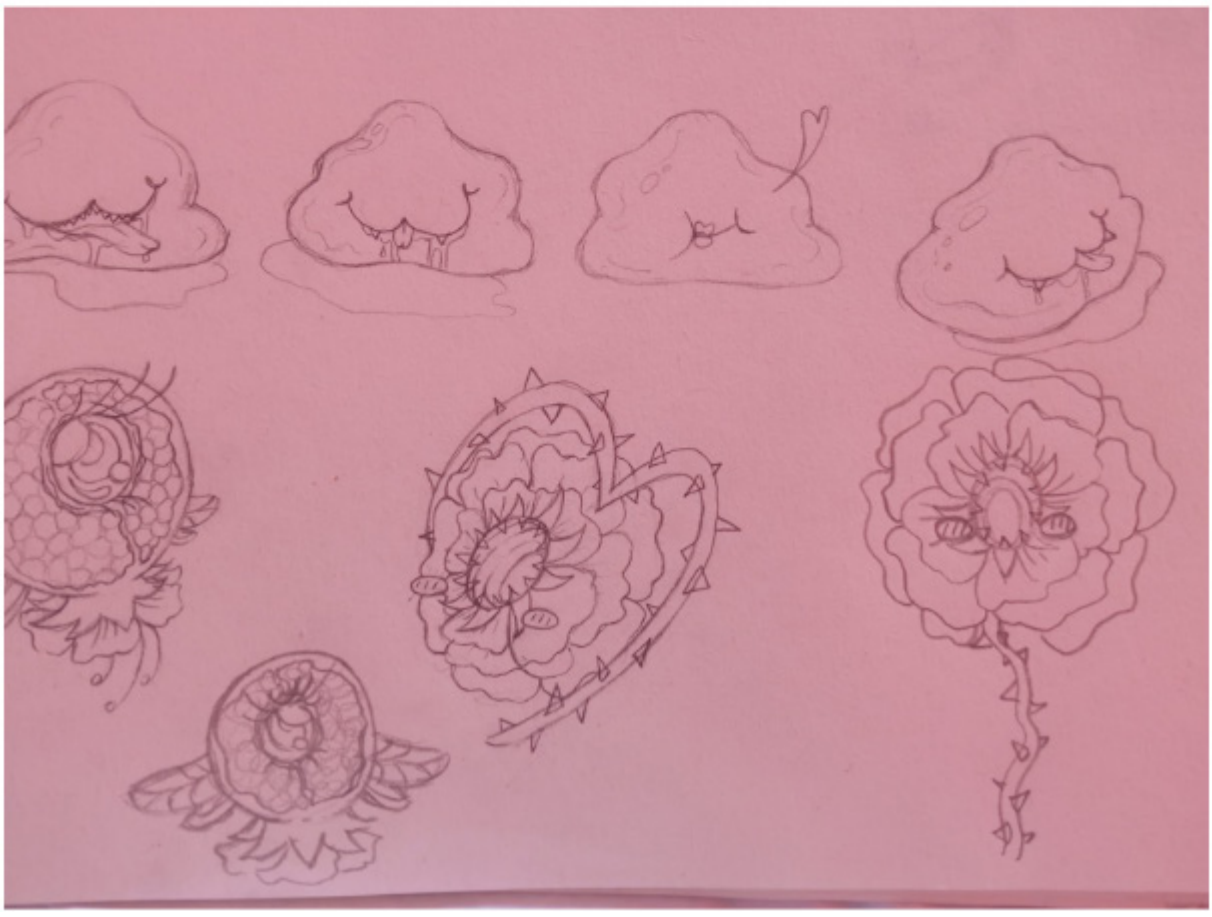


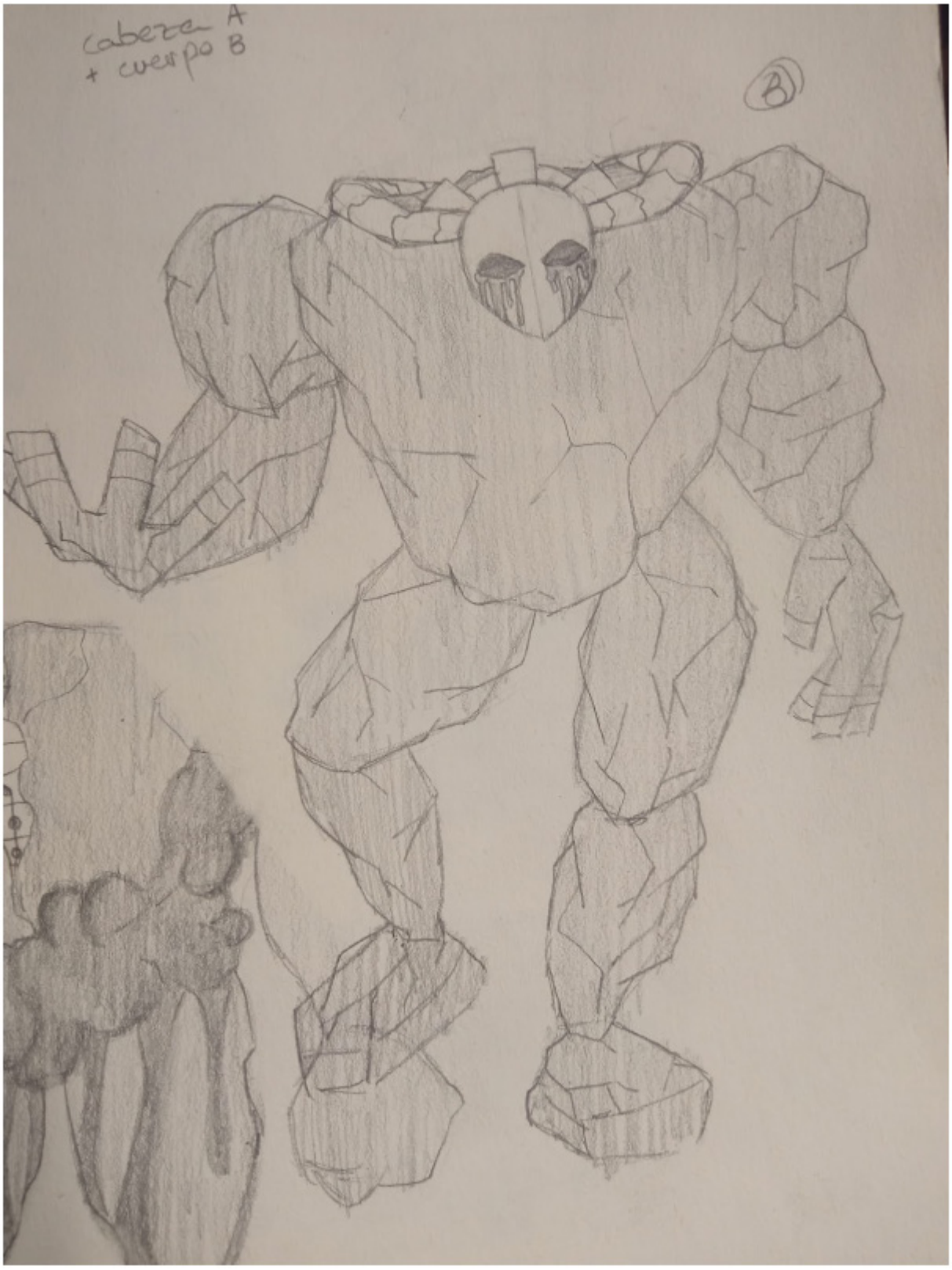

modelo
Terreno

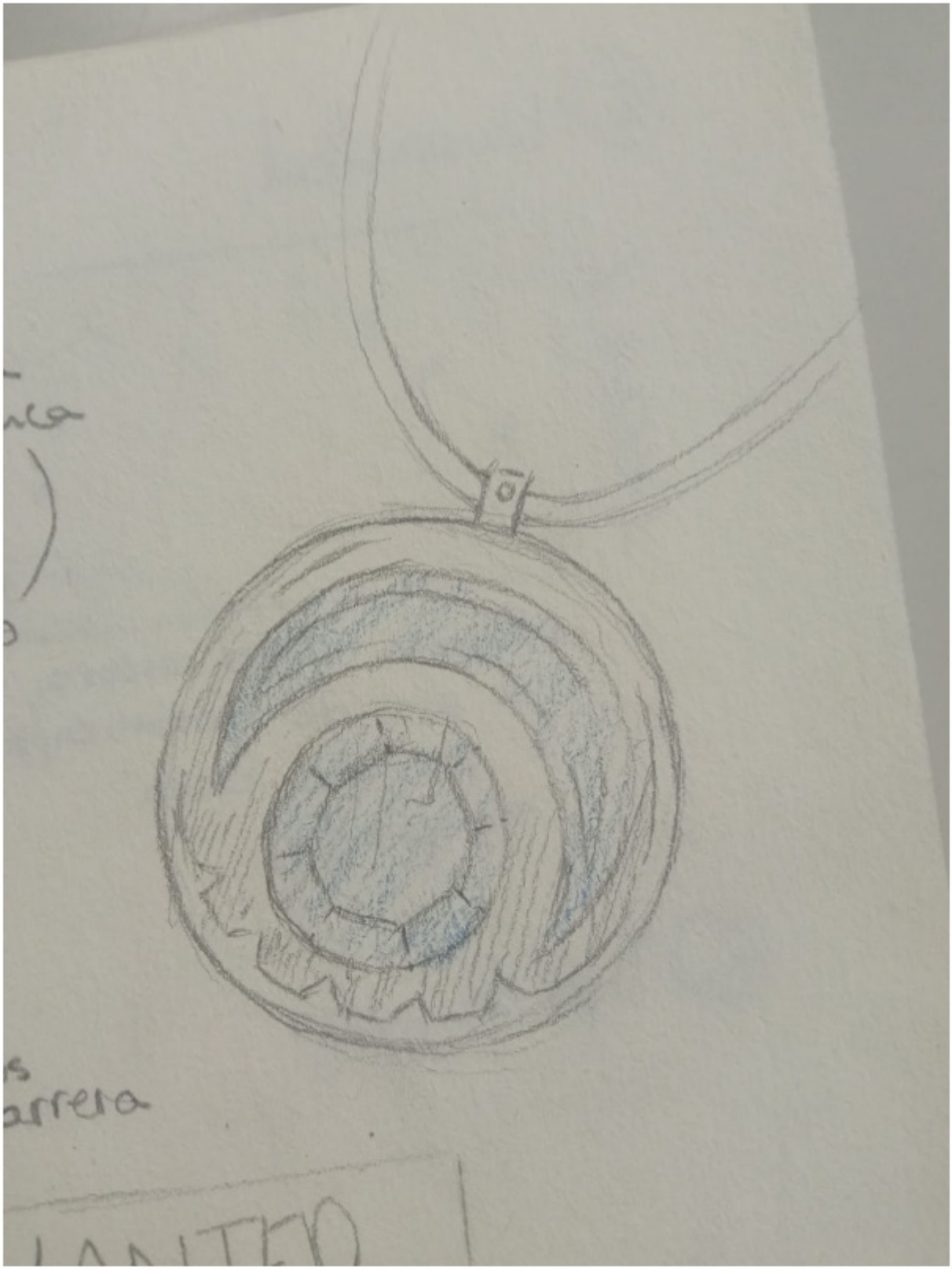


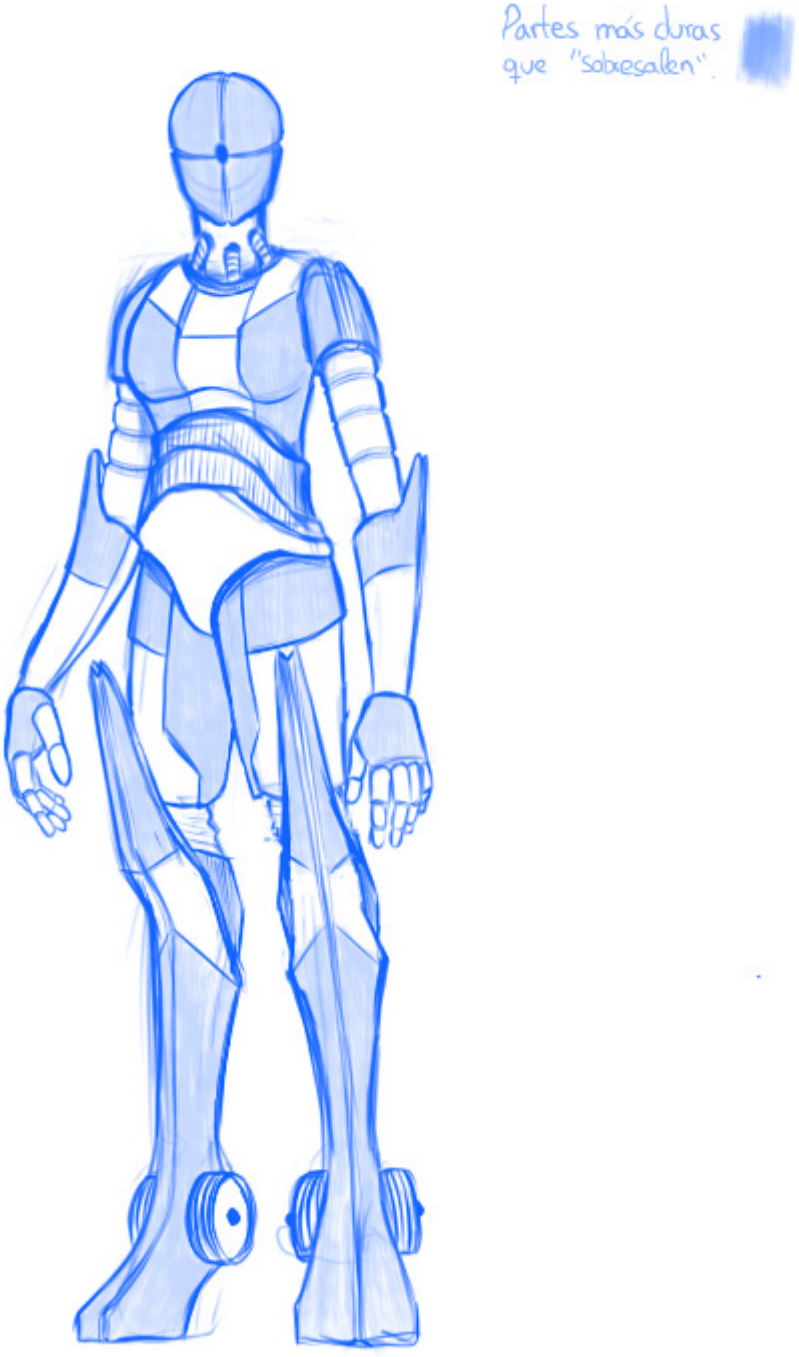


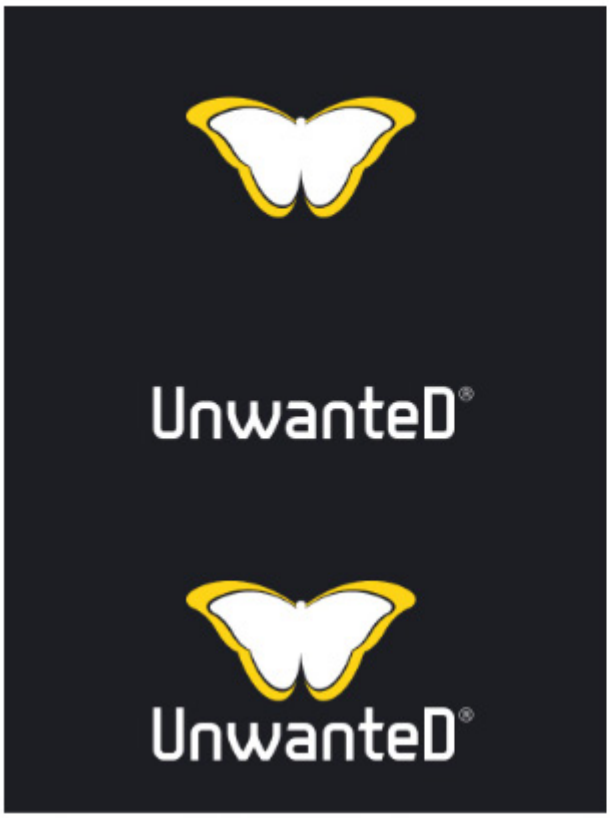





cabeza A
+ cuerpo B
B

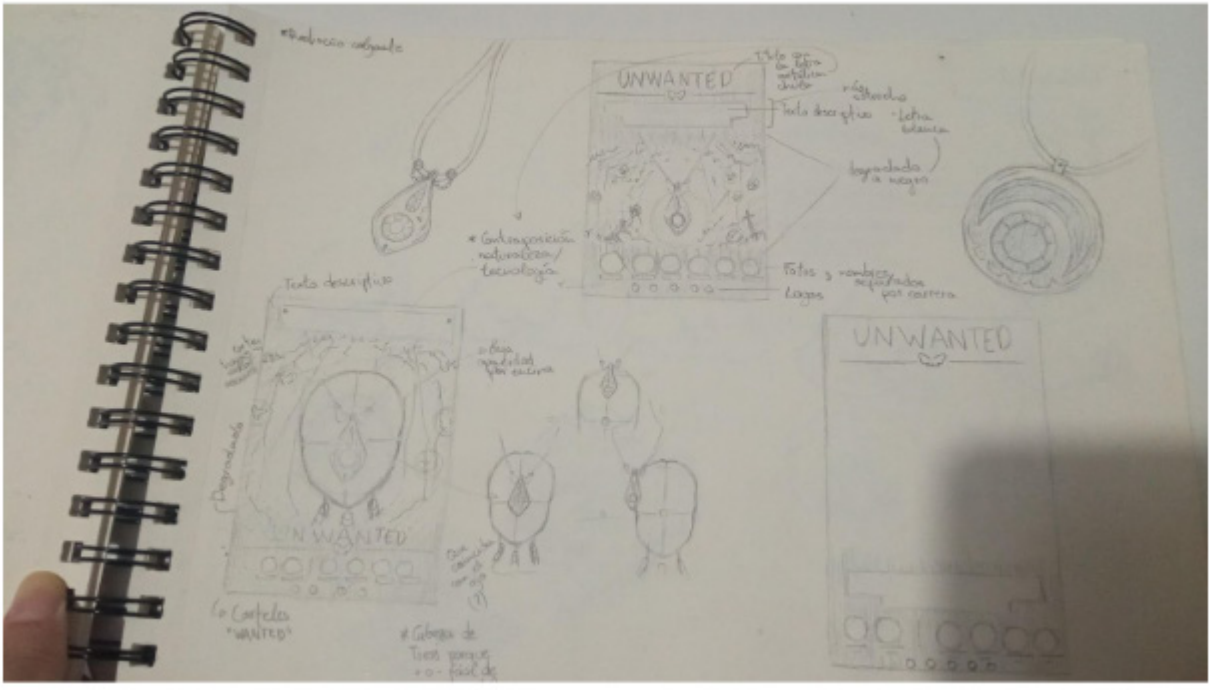



Partes más duras
que "sobresalen".

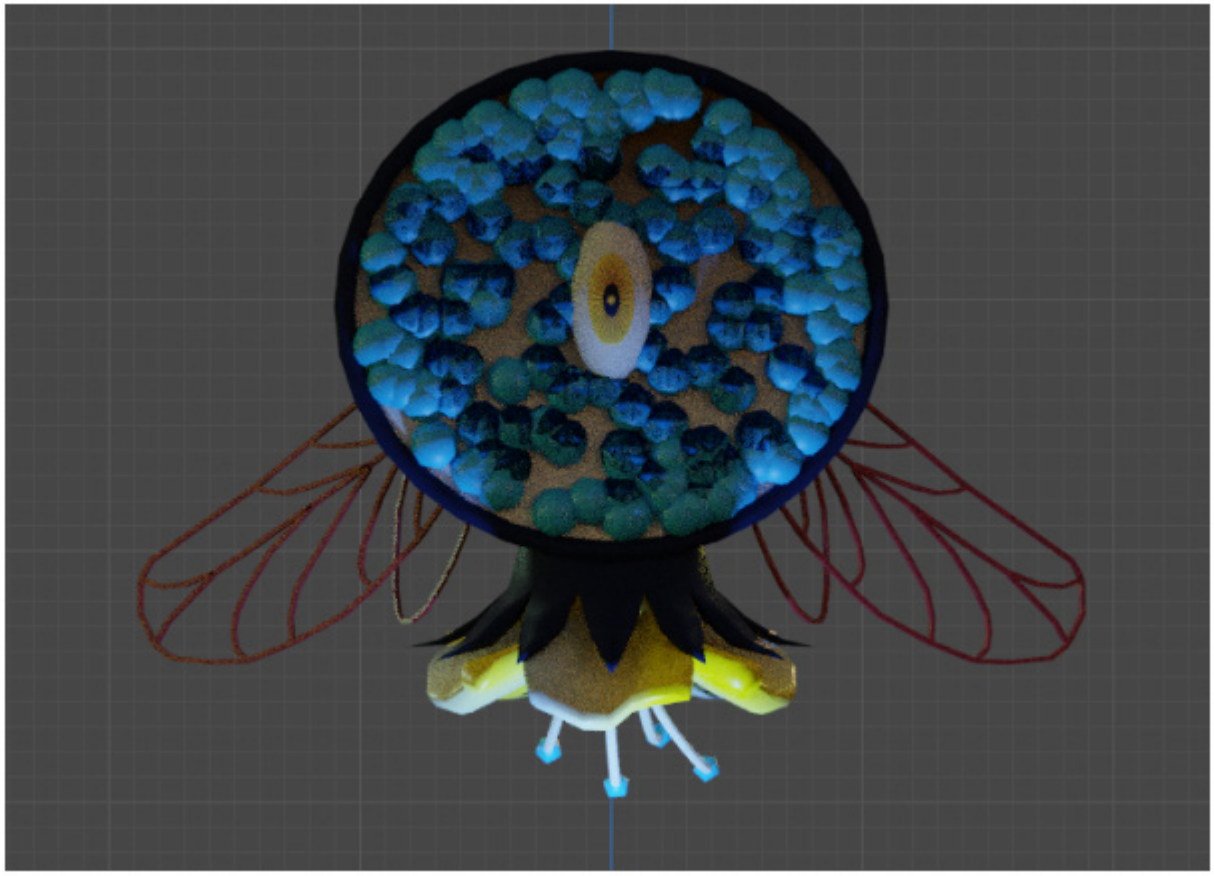


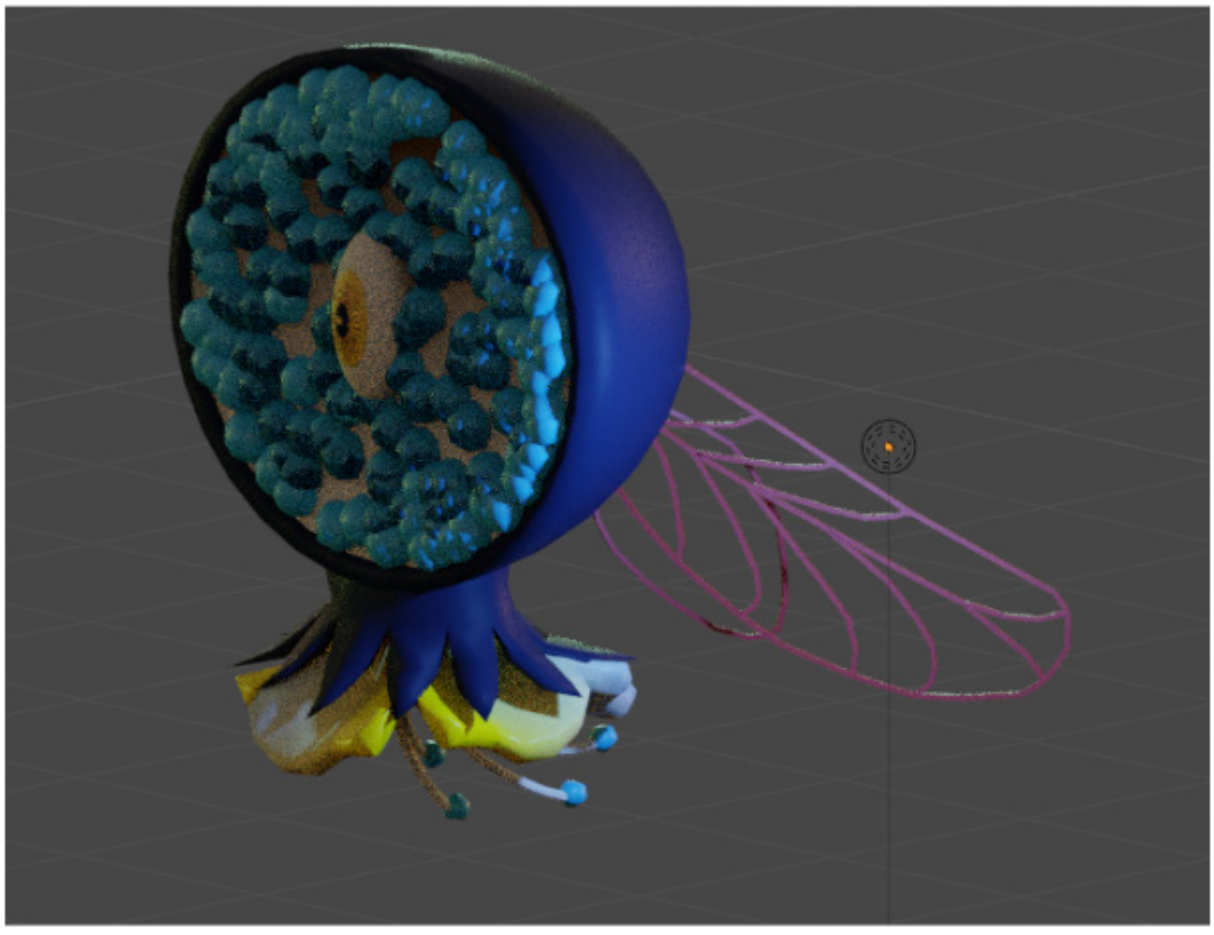
UnwanteD®

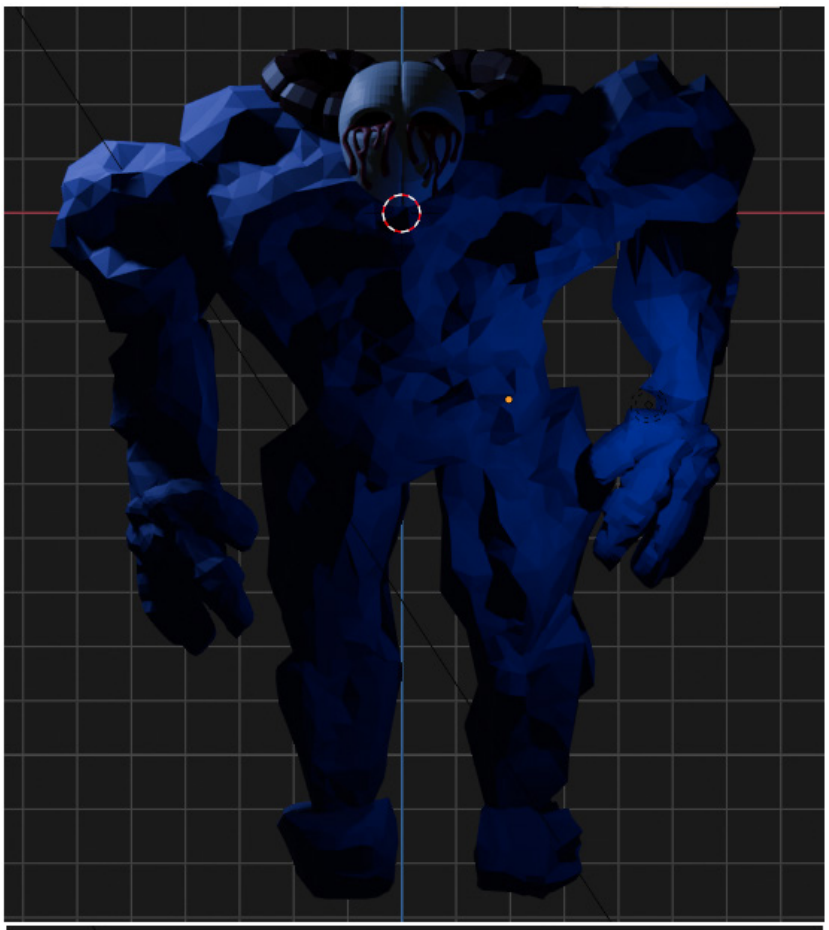
UnwanteD®

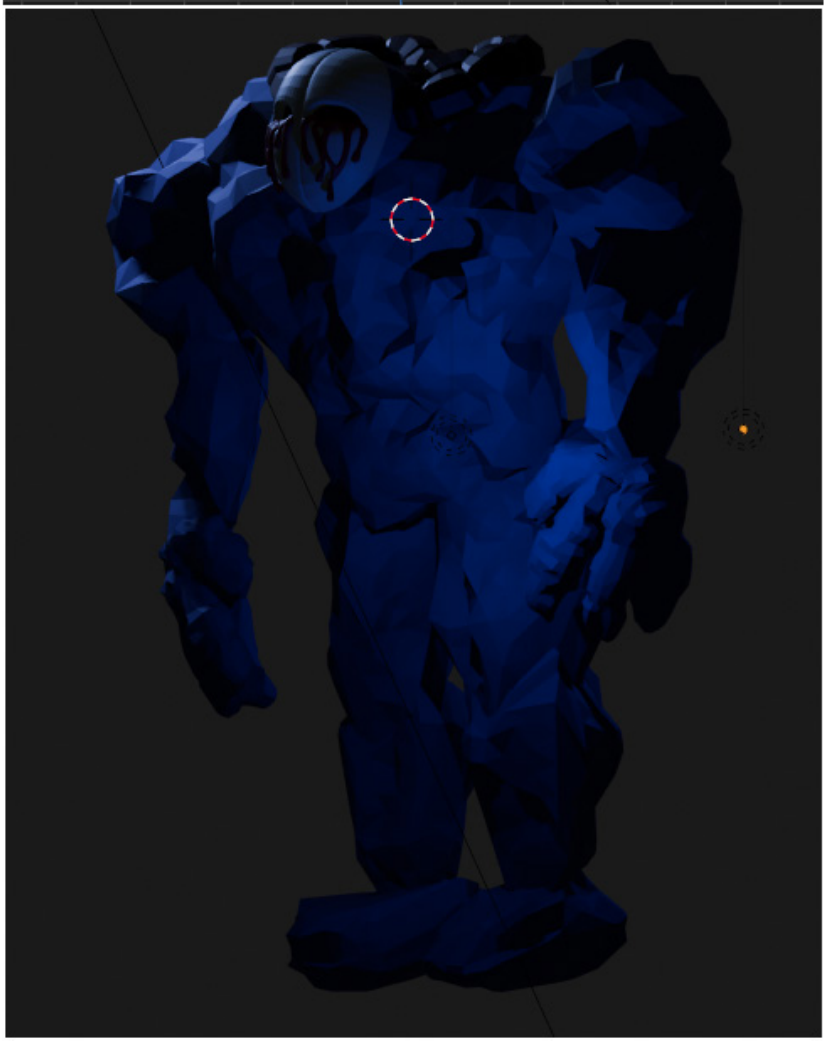


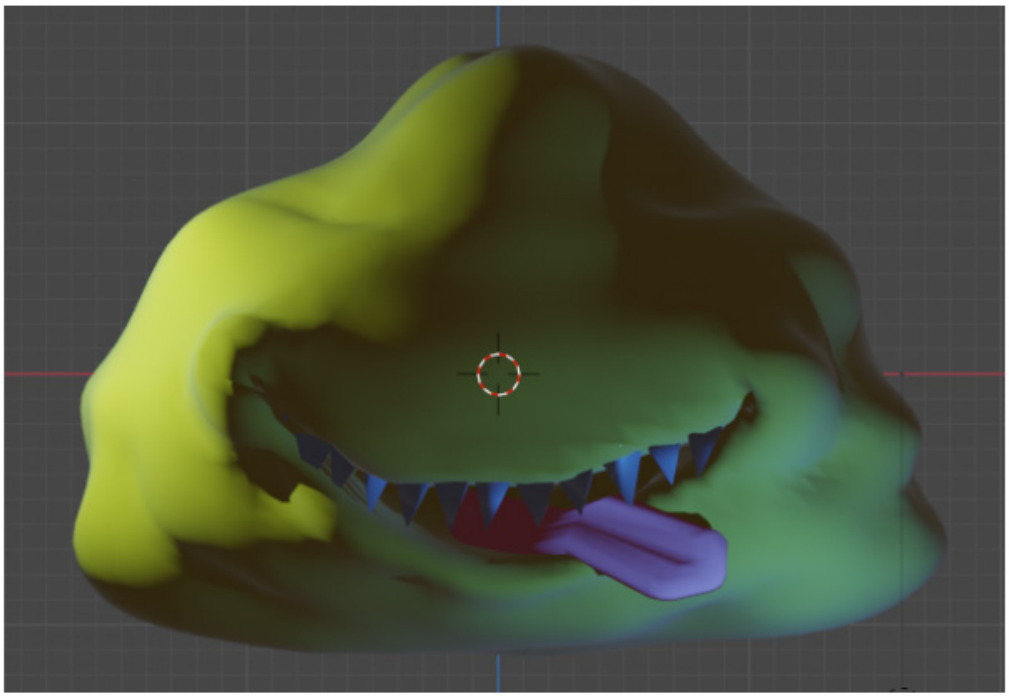
20.2 Modelado

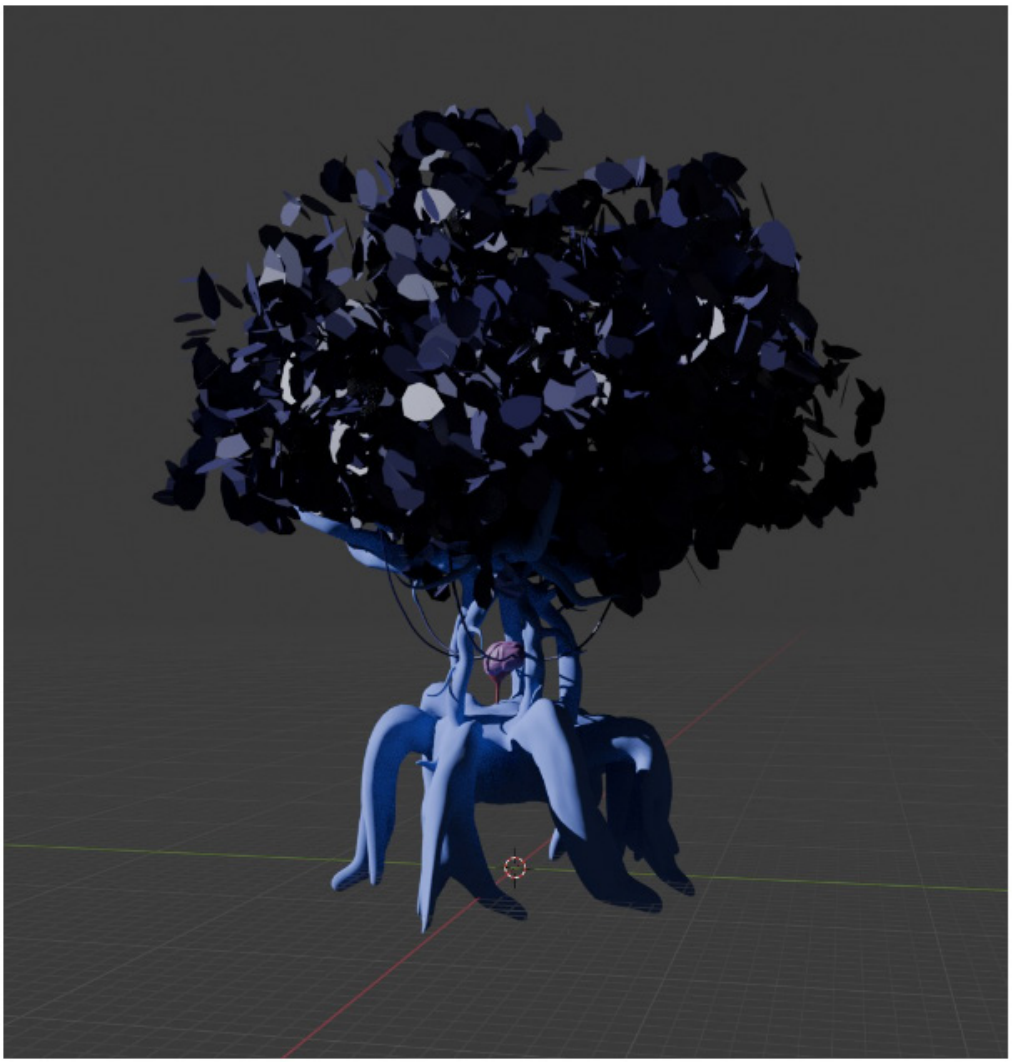


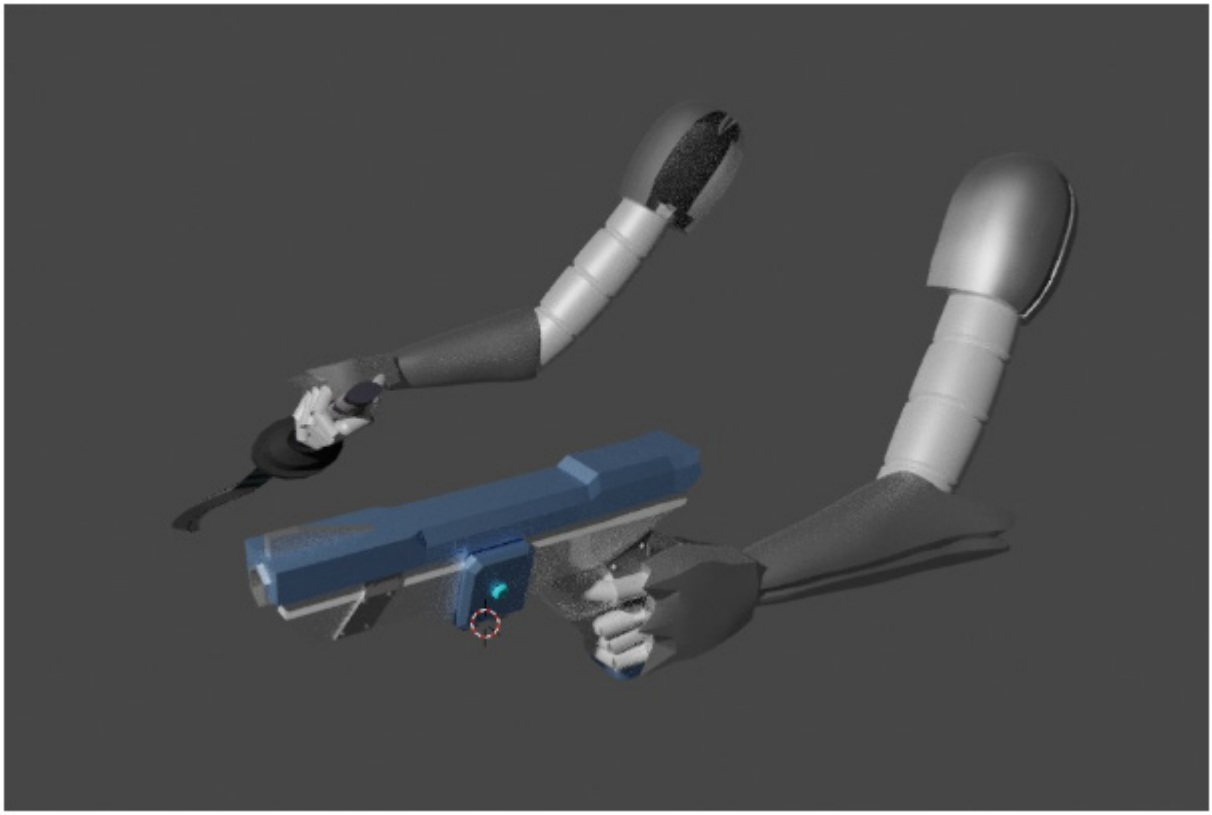


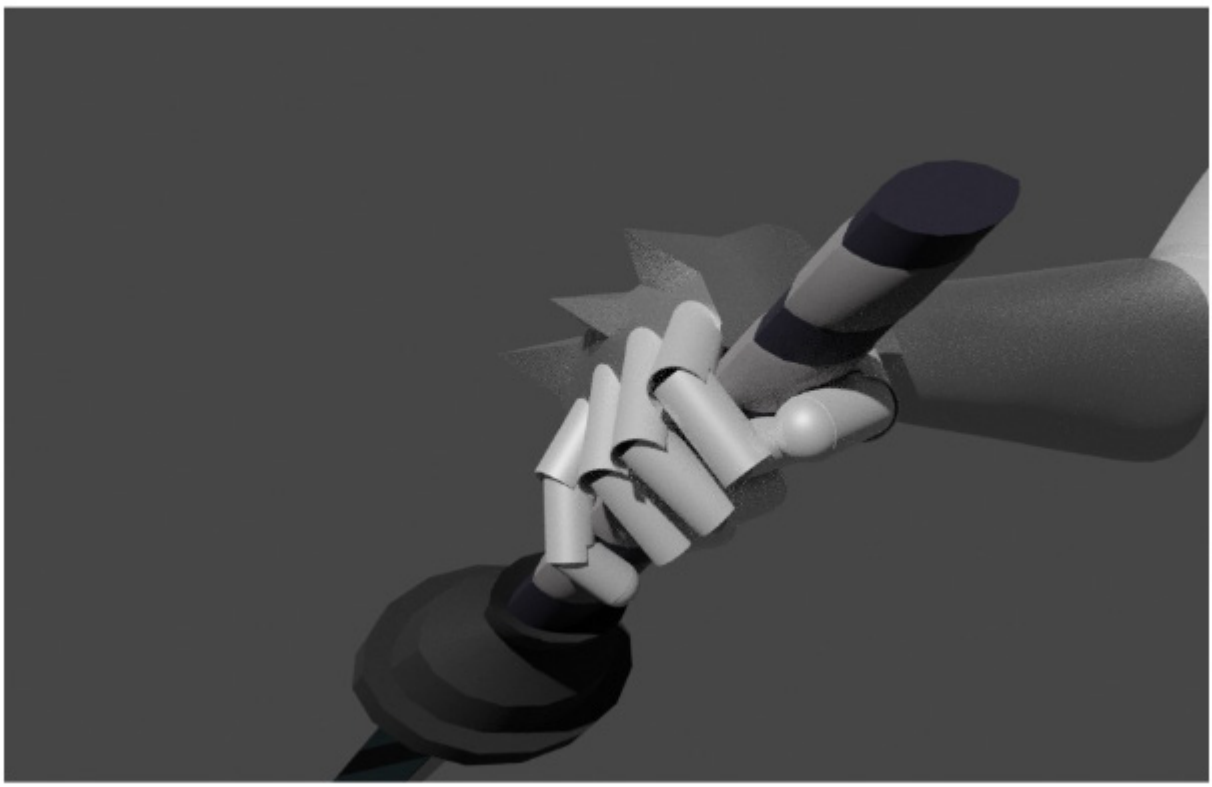


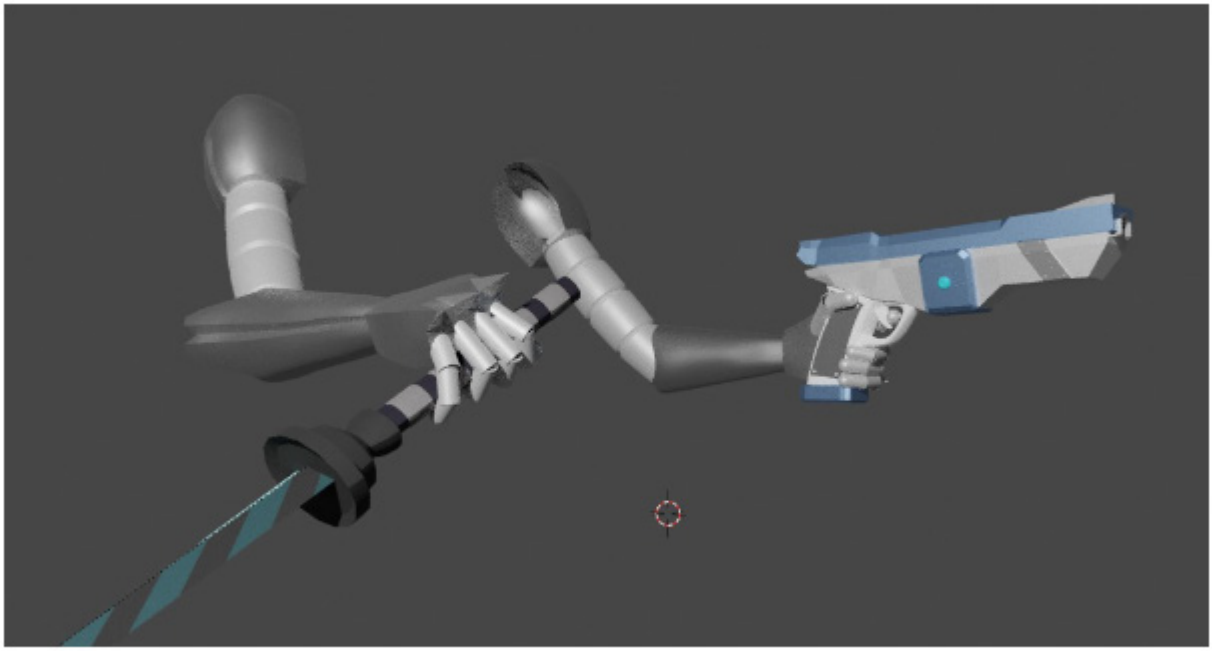


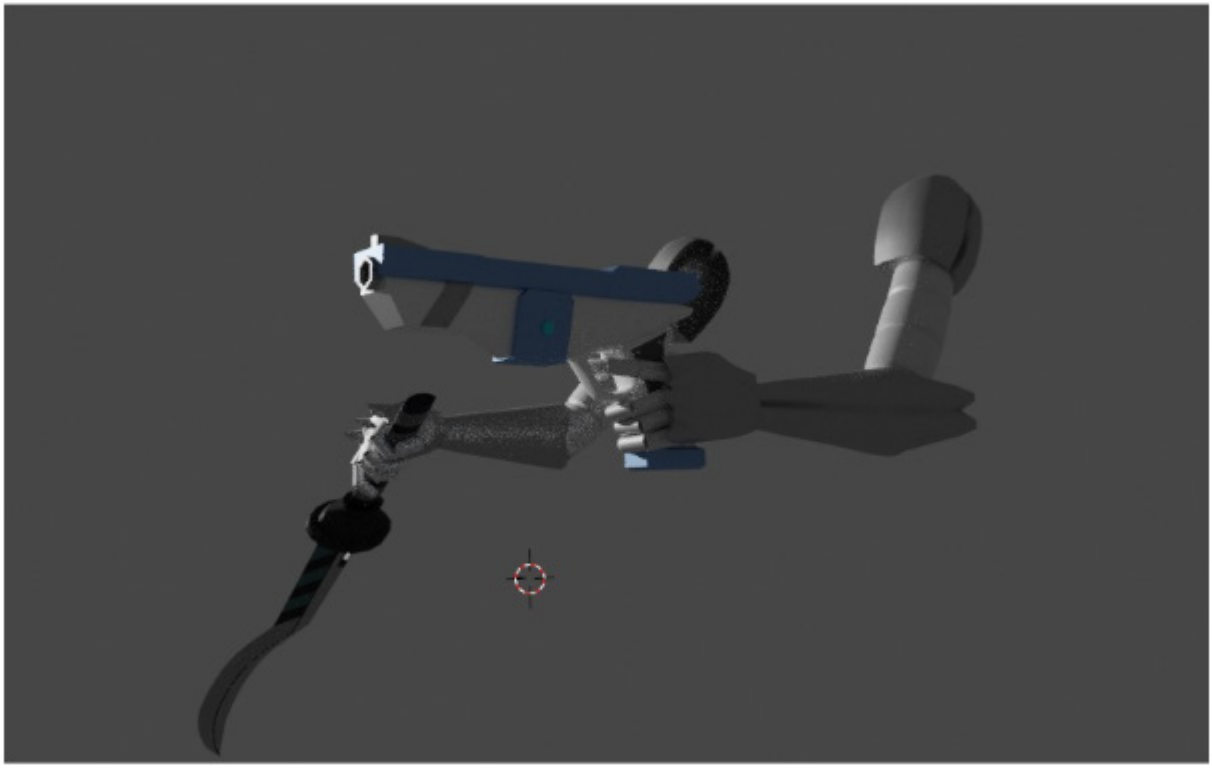




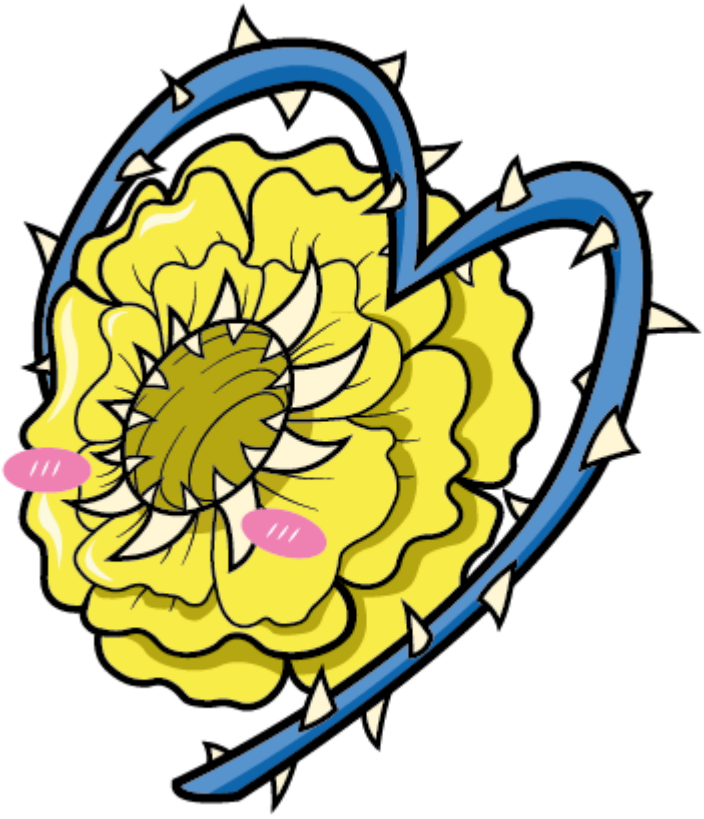


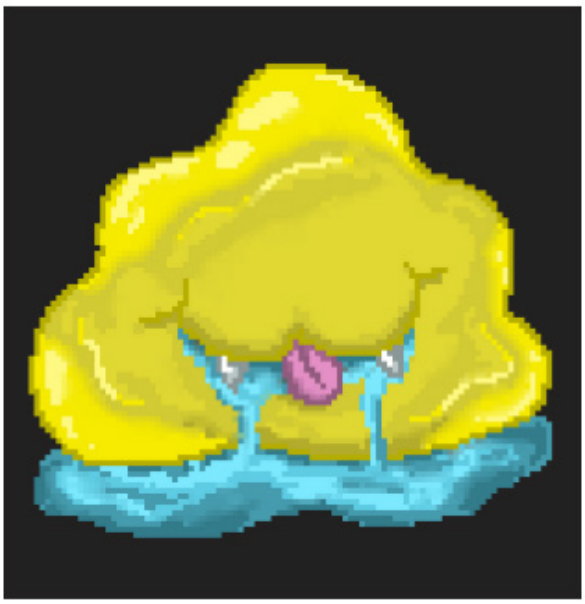




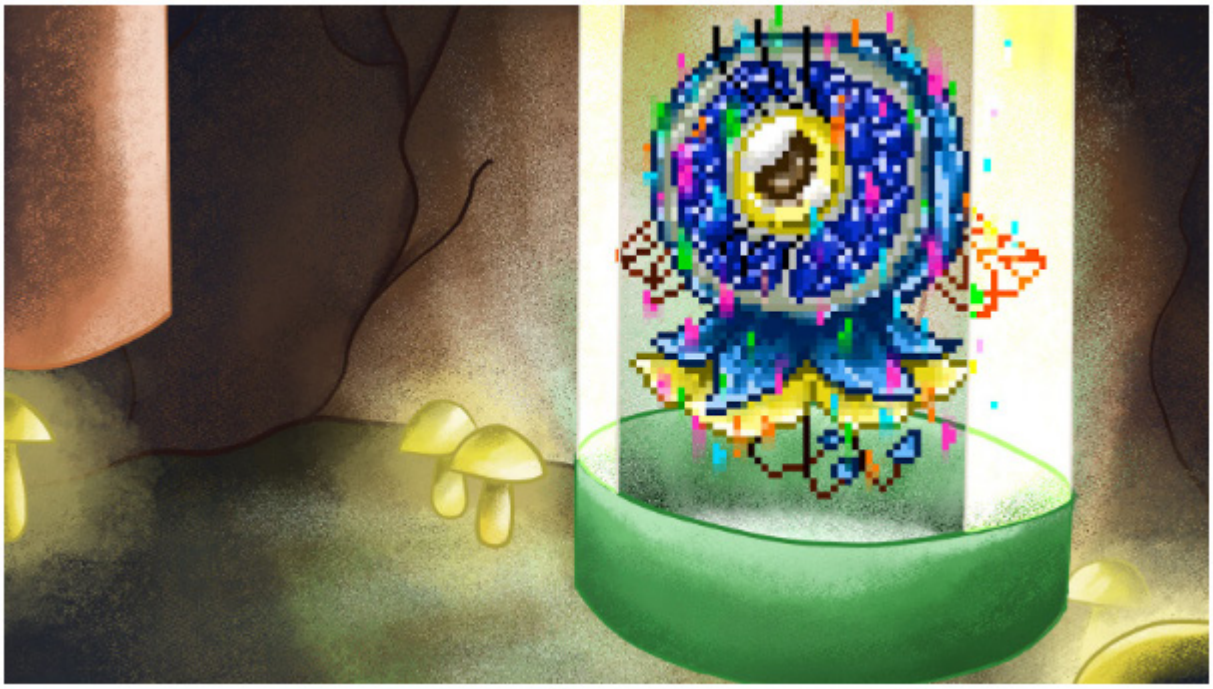


20.3 Otros proyectos de Unwanted






BAD GUYS CLUB



https://www.canva.com/design/DAF6ItYxTlo/bVaEBsvrQ16KtuwpSqA7qA/view?utm_content=DAF6ItYxTlo&utm_campaign=designshare&utm_medium=link&utm_source=editor

20.4 Otros


UNWANTED


UNWANTED
RUBÉN


UNWANTED
GAME 03
Unwanted