Universitat de Girona
Escola Politècnica Superior

Trabajo final de grado

Estudio: Grado en Diseño y Desarrollo de Videojuegos

Título: Diseño y construcción de una máquina recreativa junto a un juego arcade

Documento: Memoria

Alumno: Gonzalo Angel, Iara Mailen

Tutor: Patow, Gustavo Ariel
Departamento: Informática, Matemática Aplicada y Estadística
Área: Lenguajes y Sistemas Informáticos

Convocatoria (mes/año) Junio 2021

AGRADECIMIENTOS

A Gustavo Patow, tutor del trabajo, por confiar en este proyecto desde el inicio y por toda su ayuda brindada.

A Albert Figuerola, mi compañero en este proyecto, por aceptar embarcarse en este viaje junto a mí.

A Dana Gonzalo, mi hermana, por convertirse en mi incansable asesora de diseño; sin su ayuda y apoyo incondicional este trabajo nunca hubiera sido posible.

A Gerard Rutllant y Sandro Askerov, mis más leales compañeros, por aportar su gran granito de arena a este trabajo.

A todas las personas cercanas a mí, por aportar sus ideas, pero sobretodo, por creer y confiar ciega y absolutamente en mis capacidades, incluso cuando yo no lo hacía; mencionando especialmente a Yanela Oyarzo y Guillaume Letay, junto a mis padres.

Mil gracias a todos, por ayudarme a cumplir este sueño.

ÍNDICE DE CONTENIDO

1. INTRODUCCIÓN **6**

1.1 MOTIVACIÓN 6

1.2 PROPÓSITOS Y OBJETIVOS 7

1.3 BREVE HISTORIA DE LOS ARCADE 7

1.4 DISTRIBUCIÓN DE TAREAS 8

1.5 ESTRUCTURACIÓN DEL DOCUMENTO 9

2. ESTUDIO DE VIABILIDAD **11**

2.1 RECURSOS NECESARIOS Y VIABILIDAD 11

2.1.1 RECURSOS TÉCNICOS 11

2.1.2 RECURSOS HUMANOS 12

2.1.3 VIABILIDAD ECONÓMICA 12

2.2 ESTADO DEL ARTE 13

2.2.1 BÚSQUEDAS REALIZADAS 14

2.2.2 RESULTADOS OBTENIDOS 14

2.2.3 MATRIZ DE COMPETITIVIDAD 16

2.3 PÚBLICO OBJETIVO Y PERFIL DEL JUGADOR 16

2.4 CONCLUSIÓN VIABILIDAD DEL PROYECTO 17

3. PLANIFICACIÓN **18**

3.1 METODOLOGÍA 18

3.2 PLAN DE TRABAJO 20

3.3 TAREAS PLANIFICADAS 20

3.3.1 PLANTEAMIENTO 20

3.3.2 IMPLEMENTACIÓN E INTEGRACIÓN 21

3.3.3 ESTÉTICA 23

3.3.4 AUDIO 24

3.3.5 DOCUMENTACIÓN 24

3.4 TIEMPO ESTIMADO Y DEFINITIVO 24

3.5 RESULTADOS ESPERADOS 25

4. MARCO DE TRABAJO Y CONCEPTOS PREVIOS **26**
4.1 RASPBERRY PI 26
4.2 GODOT 26
5. DISEÑO UP, UP! NYUMI! **27**
5.1 NARRATIVA 27
5.2 ESPACIO DEL JUEGO 27
5.3 GAMEPLAY 28
5.3.1 ACCIONES 28
5.3.2 JERARQUÍA DE RETOS 29
5.3.2.1 SINGLEPLAYER 29
5.3.2.2 MULTIPLAYER 30
5.3.3 CONTROLES 31
5.3.3.1 CONTROLES PRINCIPALES 31
5.3.3.2 CONTROLES ALTERNATIVOS 33
5.4. ECONOMÍA 35
5.5 NIVELES 35
5.5.1 AUMENTO DE LA DIFICULTAD 35
5.6 SISTEMA DE GUARDADO 36
5.7 FLOWCHART 36
5.8 INTERFAZ GRÁFICA 36
5.8.1 PANTALLA INICIAL 36
5.8.2 PANTALLAS DE SELECCIÓN 37
5.8.3 PARTIDA 38
5.8.3.1 SINGLEPLAYER 38
5.8.3.2 MULTIPLAYER 39
5.8.4 PANTALLA FINAL 39
5.8.4.1 SINGLEPLAYER 40
5.8.4.2 MULTIPLAYER 40
5.9 ARTE DEL JUEGO 41
5.9.1 MENÚS DE SELECCIÓN 42
5.9.2 ESCENARIOS 43

5.9.2.1 BOSQUE MÁGICO 43
5.9.2.2 MUNDO PIRULETA 44
5.9.2.3 CASTILLO ENCANTADO 45
5.9.2.4 CIUDAD NOCTURNA 46
5.9.3 PERSONAJES 47
5.9.3.1 ORÍGENES 47
5.9.3.2 NYUMI BÁSICO 48
5.9.3.3 NYUMI BRUJA 49
5.9.3.4 NYUMI CARAMELO 50
5.9.3.5 NYUMI HONGO 50
5.9.3.6 NYUMI SERPIENTES 51
5.9.4 POWER-UPS 51
5.9.5 OTROS OBJETOS 53
5.10 SONIDOS 53
6. IMPLEMENTACIÓN 54
6.1 PANTALLAS DE SELECCIÓN 54
6.2 ESCENARIO 57
6.2.1 SINGLEPLAYER 57
6.2.2 MULTIPLAYER 62
6.3 ELEMENTOS PARTIDA 65
6.3.1 PERSONAJE 65
6.3.2 OBJETOS 71
7. RESULTADOS 74
7.1 PANTALLA INICIAL 74
7.2 SELECCIÓN DE ESCENARIO Y PERSONAJE 74
7.3 IN-GAME 76
7.3.1 SINGLEPLAYER 76
7.3.2 MULTIPLAYER 78
7.4 PANTALLA FINAL 80
7.4.1 SINGLEPLAYER 80
7.4.2 MULTIPLAYER 81

8. CONCLUSIONES **84**

9. TRABAJO FUTURO **85**

10. BIBLIOGRAFÍA **86**

11. ANEXOS **87**

11.1 PROYECTO GODOT UP, UP, NYUMI! 87

11.2 DISEÑO VINILOS 87

11.2.1 DISEÑOS DESCARTADOS 87

11.2.2 DISEÑO FINAL 89

1. INTRODUCCIÓN

A pesar de lo reciente que puede llegar a ser la industria de los videojuegos, teniendo en cuenta que arrancó a principios de los años 70, ésta ha conseguido ser uno de los sectores principales de entretenimiento, llegando a superar a los sectores de la industria musical y cinematográfica. Pero para ello, hubo (y sigue habiendo) un trabajo constante de desarrollo e innovación inmenso detrás.

En un pasado cercano, las máquinas recreativas eran muy populares; la gente se reunía en los locales de ocio y llegaba a pasar horas jugando, tanto con amigos como con desconocidos. Mas, con la aparición de las videoconsolas y ordenadores personales juntamente con Internet y los juegos en línea, cada vez era menos la gente que frecuentaba estos locales y, con los años, fueron desapareciendo paulatinamente.

Actualmente, las máquinas recreativas quedaron relegadas a objetos de coleccionismo para los más nostálgicos o, en algunos locales, aún las mantienen como segunda fuente de ingresos. Debido a todo esto, hoy en día es casi nulo el desarrollo de juegos nuevos para recreativas, a pesar de que son muchos los nostálgicos y/o amantes de los videojuegos más retro.

Por este motivo, es por lo que nos adentramos en el proyecto de desarrollar y construir una máquina recreativa low-cost, además de diseñar un juego específico para ella, dejando abierta la posibilidad de agregar otros tantos en un futuro.

1.1 MOTIVACIÓN

Nuestras motivaciones oscilan entre las personales y aquellas más profesionales. Entre las primeras se encuentra una pasión por el mundo de lo retro que se arraigó en nosotros desde un principio, quizá por el hecho de no haber tenido la oportunidad de haber vivido aquella época y las experiencias que conllevaba, y el no querer dejar caer en el olvido los orígenes de la industria de los videojuegos.

Respecto al lado más profesional, consideramos que las máquinas recreativas aún pueden dar mucho más de sí, debido a que proporcionan una experiencia de juego ni mejor ni peor que cualquier otra, pero sí completamente distinta a las que pueden ofrecernos las videoconsolas más recientes y esto debe aprovecharse. Son millones los usuarios y millones las distintas experiencias que se reclaman; es nuestro deber intentar brindarles a cada uno de ellos la posibilidad de disfrutar como tanto anhelan.

A nivel profesional, nuestra mayor motivación es aportar al mercado una máquina arcade que permita jugar a distintos videojuegos, no sólo para que los ya amantes de las recreativas recuperen su pasión, sino también para cautivar a nuevos usuarios y darles la oportunidad de vivir otra experiencia de juego y así poder conocer una posible nueva afición.

1.2 PROPÓSITOS Y OBJETIVOS

Los objetivos principales de nuestro proyecto son los siguientes:

- **Estudiar** las características de los juegos arcades y el funcionamiento de las máquinas recreativas.
- **Aprender a dominar Godot.**
- **Diseñar y desarrollar una máquina arcade *low-cost***, la cual cosa implica no solo el montaje de todos sus componentes, sino también el **diseño de sus vinilos** y el **programar su controlador** (para controlar la inserción de monedas y permitir al usuario seleccionar a qué videojuego desea jugar, entre otras funcionalidades).
- **Diseñar y desarrollar un videojuego funcional** para esta máquina, que combine los géneros **arcade y *endless runner*** y que sea capaz de cautivar a los usuarios y hacer que no quieran dejar de jugar.
- **Diseñar y desarrollar un videojuego simple** para demostrar el funcionamiento del seleccionador de juegos de la máquina.

1.3 BREVE HISTORIA DE LOS ARCADE

Cuando oímos la palabra *Arcade* enseguida lo asociamos a una máquina recreativa. Esto es debido a que los primeros juegos de este género estaban destinados a estos dispositivos. Pero debemos diferenciar entre una máquina arcade y el género de videojuegos.

Las primeras son “muebles” con controles, en su mayoría botones y una palanca, aunque hay máquinas más específicas que incluyen pistolas, volantes con pedales o una plataforma de baile. Entre sus características principales encontramos que funcionan con una placa que contiene un único juego[1] y el hecho de tener que pagar (introduciendo monedas, fichas o mediante una tarjeta) para poder jugar una partida, las cuales son de una escasa duración.

Por otro lado, si hablamos de un videojuego de género arcade, nos estamos refiriendo a un videojuego con las siguientes características:

- **Diseño minimalista,** en cuanto a escenarios y controles.
- **Sistema de vidas,** tradicionalmente se otorgan tres vidas, las cuales el jugador pierde al cometer un error.
- **Dificultad ascendente,** a medida que avanza el juego se va complicando aún más.
- **Puntaje como objetivo principal,** el objetivo de los usuarios no es completar el juego, sino obtener la máxima puntuación posible y poder así registrar su récord.

---

[1] Aunque posteriormente se desarrollaron sistemas para tener dos o más juegos en una misma placa

- **Límite de tiempo:** para que el jugador no pase demasiado tiempo en un nivel y saque ventaja de puntos. Cuando el contador llega a cero, el jugador pierde una vida.

- **Interrupciones mínimas,** entre pantallas hay escenas intermedias muy rápidas para pasar de inmediato a la siguiente fase.

1.4 DISTRIBUCIÓN DE TAREAS

La distribución de tareas se realizó conjuntamente en el momento de formar equipo, decidiendo que Albert (GEINF), se encargaría de todo el apartado de hardware (desarrollo y montaje de la máquina recreativa) y una parte del software (controlador de la máquina y un videojuego simple). Por otra parte, Iara (GDDV), se dedicaría exclusivamente al diseño y desarrollo de un videojuego más completo y al diseño de los vinilos de la máquina, por la cual cosa, su cuadro de autovaloración sería el siguiente:

Bloque	Porcentaje
Estética	60%
Narrativa	5%
Mecánicas	8%
Tecnología	27%

Tabla 1.4.1 Cuadro de autoevaluación de Iara

La estética es el bloque con más peso debido a que todo el apartado artístico del videojuego, junto a los vinilos de la máquina, son originales. Todos los sprites y animaciones fueron diseñados exclusivamente para este proyecto. Este bloque también incluye los elementos de feedback que recibe el jugador durante el juego.

A continuación lo sigue el bloque de tecnología, el cual representa toda la implementación en Godot: diseño del código y clases, control del jugador o jugadores en caso de ser el modo multijugador, implementación del sonido, flujo de las animaciones… y todo aquello referente al proyecto de Godot.

En cuanto a las mecánicas, el trabajo principalmente corresponde al diseño de estas, su testeo y balanceo.

Por último, es necesario comentar que el porcentaje del bloque de narrativa es tan bajo debido a que hay una historia muy básica tras el videojuego diseñado, la cual se explica implícitamente mediante la ambientación del juego.

1.5 ESTRUCTURACIÓN DEL DOCUMENTO

Este documento se organizó en 11 capítulos, los cuales recogen toda la información del proyecto y conclusiones finales. Son los siguientes:

- **1. Introducción**

  En este apartado explicamos la razón por la cual decidimos realizar este proyecto, las motivaciones que nos llevaron a ello, cuáles son nuestros objetivos, cómo nos organizamos el desarrollo y cómo está estructurado el documento de la memoria.

- **2. Estudio de viabilidad**

  En este capítulo se justifican los parámetros necesarios para el desarrollo de nuestro proyecto.

- **3. Planificación**

  Define la estrategia seguida para alcanzar los objetivos planeados, junto a la temporización seguida a lo largo del proyecto.

- **4. Marco de trabajo y conceptos previos**

  En este capítulo se describen diversos aspectos del desarrollo general del juego, que facilitarán al lector la comprensión de los siguientes capítulos de la memoria. También se explicarán las acciones más significativas de las primeras etapas del diseño y desarrollo del proyecto y los pasos de estudio y aprendizaje de conceptos utilizados para el desarrollo.

- **5. Diseño del videojuego**

  Este capítulo contiene la hoja de diseño del videojuego *Up, Up! Nyumi!*, la cual contiene toda la información relativa al espacio del juego, objetos, acciones, jerarquía de retos, economía y diseño artístico (personajes, feedback, escenarios…)

- **6. Implementación**

  Este apartado contiene el proceso de construcción de la aplicación, las clases y los métodos implementados que resultan más significativos para la comprensión del funcionamiento del videojuego.

- **7. Resultados**

  En este capítulo se muestran pruebas de ejecución de la aplicación, junto a imágenes del videojuego, incluyendo interfaces, imágenes *in-game* del juego y todo aquello que puede visualizar el jugador del conjunto implementado.

- **8. Conclusiones**

  En este apartado se exponen las conclusiones que se extrajeron una vez finalizado el proyecto.

- **9. Trabajo futuro**

Este capítulo contiene todos aquellos aspectos que se podrían mejorar o ampliar de manera significativa en la aplicación.

- **10. Bibliografía**

  Contiene las referencias utilizadas para el desarrollo del videojuego.

- **11. Anexos**

  Esta sección contiene el diseño de los vinilos de la máquina recreativa e información sobre dónde encontrar los archivos del proyecto de Godot.

Nótese que no se incluye ningún capítulo con las especificaciones para el uso de nuestra aplicación. Esto es debido a que los juegos de las máquinas recreativas no vienen con un manual de usuario, sinó que el propio juego suele tener un vídeo explicativo, como se da en nuestro caso. No obstante, se explicarán los controles del juego en el capítulo 5.

2. ESTUDIO DE VIABILIDAD

Antes de empezar a desarrollar el proyecto, es necesario valorar si este es viable o no. Por ello, en este capítulo se calcularán los recursos necesarios para llevar a cabo el proyecto y se hará un estudio del mercado actual, de la tipología del jugador y del público objetivo.

2.1 RECURSOS NECESARIOS Y VIABILIDAD

En este apartado se comentarán las herramientas y recursos usados para el desarrollo del proyecto. Por otro lado, a pesar de ser un proyecto realizado por dos personas sin ningún tipo de remuneración, se hará una estimación de los costes y recursos humanos como si se tratara de un proyecto realizado por una empresa de videojuegos. Finalmente, valoraremos si es viable o no la realización de nuestro proyecto.

2.1.1 RECURSOS TÉCNICOS

Nuestro proyecto está dividido en dos partes: desarrollo de la máquina arcade y diseño y desarrollo de los videojuegos que hay en ella.

Para la implementación de los videojuegos no se requiere una gran cantidad de recursos técnicos, ya que puede ser llevada a cabo con un ordenador medianamente decente y un conjunto de programas gratuitos. No obstante, para el desarrollo de la máquina necesitamos todos sus componentes, incluida la estructura.

Dicho esto, los recursos que necesitamos para nuestro proyecto son los siguientes:

- **Hardware:**
    - **Ordenador de sobremesa y ordenador portátil:** para desarrollar el software (controlador de la máquina y juegos implementados).
    - **Raspberry Pi 4:** como núcleo de la máquina recreativa.
    - **Estructura máquina arcade,** compuesta por la parte superior (*bartop*) y la inferior (*pedestal*).
    - **Componentes máquina recreativa,** pantalla, altavoces, botones y palancas, tarjeta microSD, cables diversos.
- **Software:**
    - **Inkscape:** programa de dibujo vectorial, utilizado para el diseño artístico del videojuego casual.
    - **Photoshop:** programa para la edición de imágenes, usado para el diseño de los vinilos de la máquina arcade.
    - **HackNPlan:** para organización y control de las horas trabajadas y de las tareas planificadas.

- **Godot:** como motor de videojuegos sobre el que se desarrollará el proyecto.
- **Drive:** para los documentos e imágenes.
- **Diagrams.net:** para realizar cualquier tipo de diagrama.
- **Git:** para una mejor organización y seguridad del proyecto.
- **Meet:** programa de llamadas en línea. Usado para realizar las reuniones durante el desarrollo
- **Audiotrimmer.com:** web gratuita para la edición de la velocidad de audios
- **bearaudiotool.com:** web gratuita para la edición de archivos de audio

Todo el software empleado es de licencia gratuita o lo habíamos adquirido anteriormente, por lo cual no supuso ninguna inversión económica en este aspecto.

2.1.2 RECURSOS HUMANOS

Para el desarrollo de cualquier videojuego es necesario un equipo formado por profesionales que cubran todos los roles indispensables, que son los siguientes:

- **Diseñador del juego:** encargado de definir el juego, sus objetivos, narrativa y cualquier otro elemento necesario.
- **Programador:** encargado de la implementación del juego, lo que incluye el diseño de clases y métodos, el desarrollo del código y los aspectos más técnicos del proyecto.
- **Artista:** encargado de diseñar y desarrollar los componentes visuales del juego
- **Compositor:** encargado de componer la música y efectos sonoros del proyecto.

Cabe recalcar que un rol no es cubierto obligatoriamente por una única persona, ni que una misma persona debe desempeñar un único papel. Además, los perfiles de un equipo no se limitan a los explicados anteriormente, pero sí que son los más relevantes.

En nuestro caso, ambos tuvimos que cubrir los cuatro roles anteriores, aunque también obtuvimos ayuda externa a la hora de diseñar los elementos visuales y escoger la música y efectos sonoros.

2.1.3 VIABILIDAD ECONÓMICA

Desde el punto de vista económico, ya comentamos que la parte del software no nos supone ningún coste extra, pero sí que nos lo supone la maquinaria para desarrollar la máquina arcade en su totalidad. Para esta parte del proyecto las inversiones a realizar fueron las siguientes:

RECURSO	COSTE
Pack Raspberry Pi 4 (incluye cable de alimentación, HDMI y microSD)	87.83€
Botones y palancas + cables correspondientes	68.75€
Bartop (con vinilo personalizado)	198.21€
Pedestal (con vinilo personalizado)	152.22€
Pantalla	110.00€
Kit de sonido	25.90€
Total	**642.91€**

Tabla 2.1.1 Gastos de maquinaria

En lo que a los gastos de recursos humanos respecta, en caso de que realizáramos una estimación hipotética de los presupuestos de los integrantes del equipo, suponiendo que hay que contratar a un especialista por cada perfil principal, los gastos por hora estimativos serían los siguientes:

- **Diseñador del juego:** 18 € / hora
- **Programador:** 15 € / hora
- **Artista:** 15 € / hora
- **Compositor:** 13 € / hora

Una vez hecha la planificación, se podría calcular el coste total estimado del proyecto multiplicando las horas que dedicó cada profesional por su sueldo medio. Pero como mencionamos anteriormente, eso sería en un caso hipotético, ya que realmente el coste de recursos humanos total es nulo, debido a que entre los dos integrantes del grupo realizamos los diferentes roles y la ayuda que recibimos por terceras personas fue puramente altruista.

2.2 ESTADO DEL ARTE

Antes de empezar a implementar el juego “principal” de la máquina y tomar decisiones importantes, debemos tener en cuenta el estado actual del mercado de los videojuegos, especialmente del género de nuestra propuesta, para poder así estudiar la competencia.

En esta sección se analizará el mercado actual, seleccionando los juegos más apropiados para compararlo con el nuestro y valorar si es conveniente seguir adelante con el proyecto o no.

2.2.1 BÚSQUEDAS REALIZADAS

Nuestra mayor competencia son aquellos videojuegos que pertenezcan al género arcade y/o aquellos considerados como *endless runner* (juego sin final en el cual el jugador debe avanzar la mayor distancia posible en una misma dirección).

Teniendo en cuenta estos parámetros, las palabras claves para hacer las búsquedas fueron las siguientes:

- Arcade game
- Endless runner

Estas palabras se buscaron principalmente en *Google*, usándolas tanto sueltas como combinadas entre ellas.

2.2.2 RESULTADOS OBTENIDOS

De todos los resultados obtenidos, analizamos y seleccionamos los siguientes:

- **Pink Panther Jewel Heist**

  Este videojuego no cumple las condiciones necesarias para ser considerado un juego arcade y tampoco para ser un *endless runner*. Aún así decidimos escogerlo porque es un juego muy sencillo a la vez que adictivo, con un alto potencial de convertirse en competencia directa.

  El objetivo es conseguir llegar hasta el final del edificio; para ello, el jugador debe bajar cien plantas, evitando a los patrulleros de las plataformas y al rayo láser (el cual va bajando impidiendo que el jugador se quede quieto durante mucho tiempo).

Figura 2.2.1 Pink panther *in-game*

Figura 2.2.2 Màquina arcade Pink Panther Jewel Heist

- **Subway surfers**

  Es un videojuego originalmente destinado para móviles, aunque después fue adaptado a otras plataformas debido a su éxito, entre ellas, las máquinas recreativas.

  El juego original es un *endless runner*, en el cual el jugador controla a un grafitero que debe escapar de la policía, corriendo entre vías férreas, mientras evita los trenes que pasan por ellas, a la vez que puede recoger monedas y otros bonus.

  Para adaptarlo a la máquina recreativa el objetivo del juego fue modificado, haciendo que el jugador deba recoger una cantidad de letras durante la partida, la cual está limitada en el tiempo. Aún así, la jugabilidad sigue siendo la misma.

Figura 2.2.3 Subway surfers *in-game*

Figura 2.2.4 Subway surfers *máquina arcade*

- **Doodle Jump**

  Es un *endless runner* para móviles (también adaptado a máquinas arcade), en el cual el usuario controla a un personaje que salta sin parar y debe ir subiendo por las plataformas evitando los obstáculos. El objetivo es recorrer la mayor distancia antes de morir. Es un juego que fácilmente podría ser de género arcade, ya que su diseño es muy minimalista, el puntaje es el objetivo principal y la dificultad es ascendente.

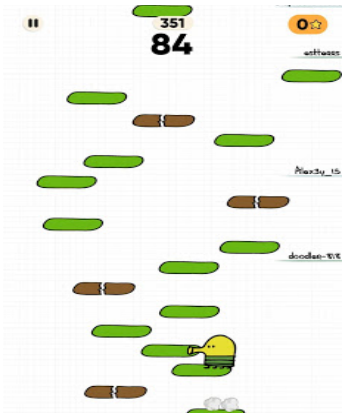


Figura 2.2.5 Doodle Jump *in-game*

Figura 2.2.6 Doodle Jump *máquina arcade*

2.2.3 MATRIZ DE COMPETITIVIDAD

Para comparar los videojuegos escogidos con nuestra propuesta, seleccionamos los parámetros que mejor nos permiten visualizar las flaquezas y fortalezas de cada obra y poder así observar sus diferencias más notorias:

NOMBRE	MULTIJUGADOR	COMPETITIVO	POWERUPS	DISEÑO	RITMO
Pink Panther	No	Sí	No	Sencillo	Rápido
Subway Surfers	No	Sí	Sí	Complejo	Medio
Doodle Jump	No	Sí	Sí	Sencillo	Medio
Up, up! Nyumi! (propuesta propia)	Sí	Sí	Sí	Sencillo	Rápido

Tabla 2.2.1 Matriz de competitividad

Nota: Consideramos Pink Panther Jewel Heist como juego competitivo a pesar de no tener una máxima puntuación registrada, ya que los usuarios compiten entre ellos para ver quién llega más lejos y, en caso de llegar a la misma planta (o superar el juego), compiten por quién consiguió más monedas durante el camino.

El juego propuesto se diferencia de los demás por ser el único que cumple con los tres primeros parámetros, además de tener un ritmo de juego rápido desde el principio. Esto hace que el juego sea más adictivo.

Por otra parte, el tener un diseño artístico más simple, hace que el jugador no acabe saturado por demasiada información y quiera dejar de jugar.

2.3 PÚBLICO OBJETIVO Y PERFIL DEL JUGADOR

A la hora de diseñar un videojuego, uno de los puntos clave es identificar a nuestro público objetivo, de manera que podamos conocerlo previamente y saber como atraerlo.

Para definir el perfil del jugador de nuestro juego principal, analizamos la tipología de jugadores definida por Richard Bartle, según el cual hay cuatro tipos de perfiles:

- ***Achievers,*** tienen como objetivo resolver el mayor número de retos y dificultades y conseguir una recompensa por ello. Obtienen placer resolviendo situaciones complejas. Están interesados en actuar en el mundo del juego.
- ***Explorers,*** les gusta explorar a fondo el videojuego porque quieren descubrir y aprender cualquier cosa nueva o desconocida del sistema. Están interesados en interactuar con el mundo del juego.

- ***Socializers,*** son aquellos que sienten atracción por los aspectos sociales por encima de la misma estrategia del juego. Están interesados en interactuar con los jugadores.

- ***Killers,*** buscan competir con otros jugadores. Están interesados en actuar sobre los jugadores.

A nuestros jugadores les gusta competir contra los demás y obtener la máxima puntuación posible en cualquier juego, demostrando así su superioridad. Por lo tanto, el perfil que buscamos en nuestro jugador corresponde inequívocamente a los *Killers*.

Por otra parte, también es importante acotar el rango de edad de nuestros usuarios, ya que un niño no reacciona de la misma manera que un adulto a un mismo estímulo. En nuestro caso, decidimos orientar el videojuego hacia personas jóvenes-adultas (entre los 16 y los 45 años), ya que la gente nacida sobre la década de los setenta-ochenta pudieron vivir el auge de las máquinas recreativas y sentir cierta nostalgia hacia estas, mientras que los más jóvenes suelen mostrar más interés en probar todo tipo de juegos con tal de obtener nuevas experiencias y encontrar su verdadera pasión.

2.4 CONCLUSIÓN VIABILIDAD DEL PROYECTO

Por lo que al software respecta, se utiliza software gratuito y versiones de estudiante de aquellos que no lo son, consiguiendo que el coste sea cero. No obstante, los componentes para montar la máquina recreativa requieren de una inversión importante, pero asequible.

En cuanto a la remuneración de los trabajadores, en el caso hipotético de una contratación real, no debería suponer un precio excesivamente elevado, ya que es un proyecto relativamente simple si los roles son desempeñados por distintos profesionales. En nuestro caso, la inversión de horas es mayor, ya que estamos cubriendo cuatro roles diferentes con un nivel lejano al profesional; por ende, también se requiere una dedicación de horas considerable en cuanto a investigación y aprendizaje.

Respecto al estudio de mercado, podemos demostrar que el proyecto seleccionado es viable gracias a la matriz de competencia y, por el hecho de que nuestro público objetivo es lo suficientemente amplio como para ser un videojuego competente dentro del mercado.

En definitiva, este proyecto será viable, ya que es viable a nivel técnico y económico (a pesar de que el presupuesto no sea nulo, puede ser cubierto) y, además, puede llegar a ser una aportación significativa en el mercado de los videojuegos.

3. PLANIFICACIÓN

Antes de comenzar cualquier proyecto, hay que planificarlo, de manera que podamos estructurarlo de una manera más organizada y controlada, y poder calcular una estimación del tiempo de dedicación que requiere.

A lo largo de este capítulo se describe la metodología empleada junto a la planificación que se siguió durante la elaboración del proyecto, el cual consideramos de una duración total de 7 meses (desde noviembre hasta principios de junio).

3.1 METODOLOGÍA

Para la realización de este proyecto no se ha seguido una metodología de trabajo estándar, sino que se ha elegido y utilizado una metodología personalizada y que fue planteada con el tutor.

Esta metodología es la misma que se sigue en el proyecto *skylineEngine*, y los pasos que sigue son los siguientes:

1. Elegir el trabajo a desarrollar.
2. Decidir el lenguaje de programación y herramientas a utilizar.
3. Aprender el lenguaje de programación y las herramientas escogidas.
4. Estructurar el trabajo en partes según las funciones que se tengan que realizar.
5. Desarrollar la parte correspondiente siguiendo el orden de la estructura del trabajo.
6. Hacer comprobaciones para confirmar que el funcionamiento es correcto al finalizar cada parte.
    - Si al hacer las comprobaciones el resultado no es el deseado, se volverá al punto 5 para realizar los cambios oportunos en la última parte desarrollada o en las anteriores, si es necesario.
    - Si al hacer las comprobaciones el resultado es el deseado, se desarrollará la siguiente parte volviendo al punto 5. Una vez se hayan finalizado todas las partes con sus respectivas comprobaciones, se iniciará el punto 7.
7. Unir todas las partes desarrolladas y comprobar que el funcionamiento es correcto.
    - Si al hacer las comprobaciones el resultado no es el deseado, se volverá al punto 5 para realizar los cambios oportunos en la última parte desarrollada o en las anteriores, si es necesario.
    - Si al realizar las comprobaciones el resultado es el esperado, se iniciará el punto 8.
8. Generar diferentes situaciones de ejemplo para comprobar que el funcionamiento es el correcto.

- Si al hacer las comprobaciones el resultado no es el deseado, se volverá al punto 5 para realizar los cambios oportunos en la última parte desarrollada o en las anteriores, si es conveniente.
- Si al realizar las comprobaciones el resultado es el esperado, se iniciará el punto 9.

9. Presentar documentación.

La *Figura 3.1* muestra el esquema gráfico de los pasos anteriores.

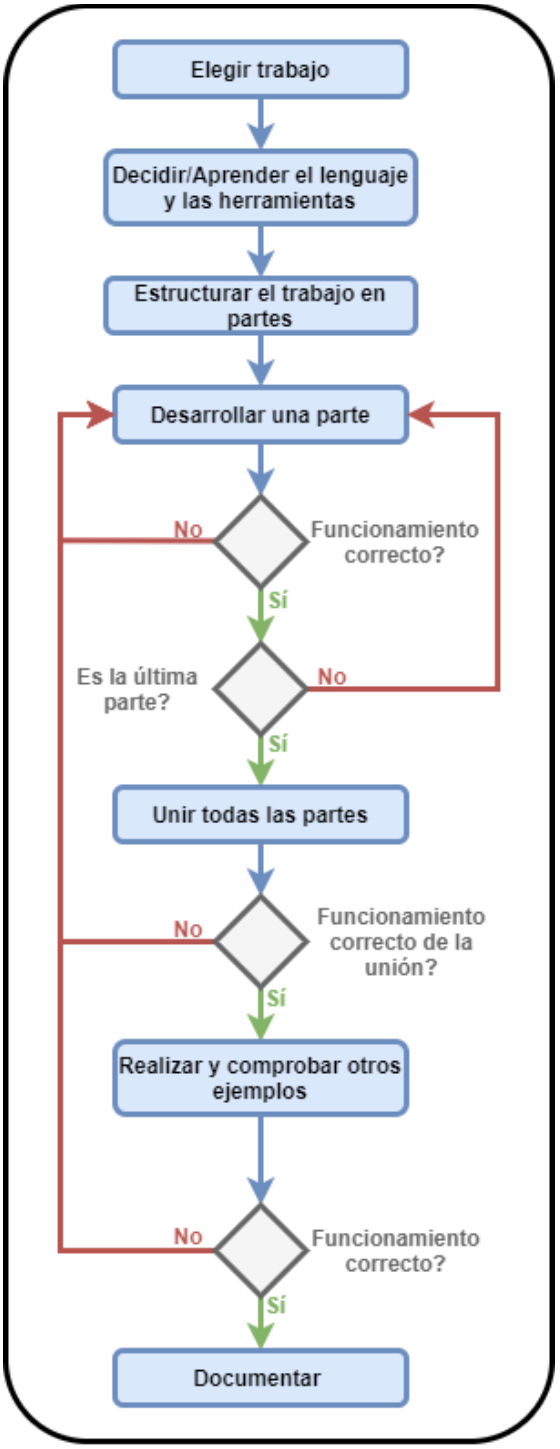


Figura 3.1 Diagrama metodología utilizada

3.2 PLAN DE TRABAJO

Una vez planteadas todas las tareas, las agrupamos en cinco bloques principales: planteamiento, implementación e integración, estética, audio y documentación. Véase *Figura 3.2.1 Organigrama principal*.

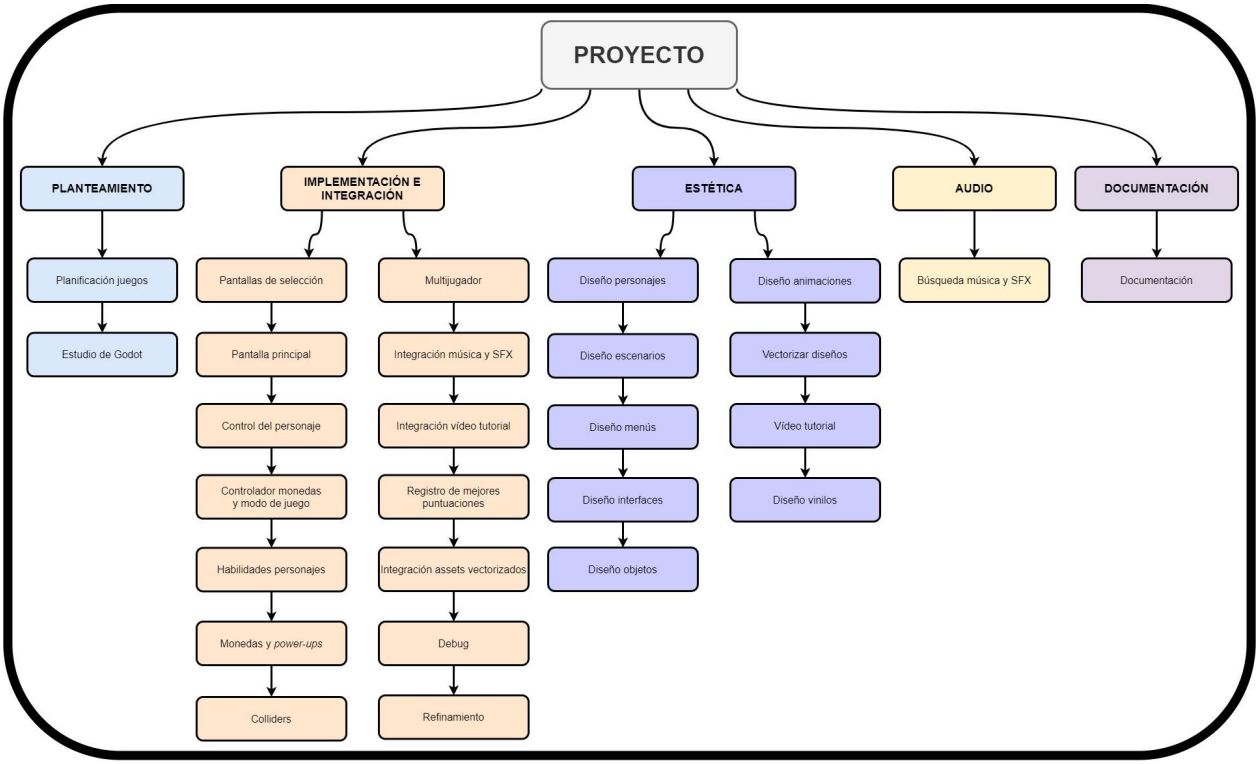


Figura 3.2.1 Organigrama principal

3.3 TAREAS PLANIFICADAS

En los siguientes subapartados se explicará brevemente en qué consiste cada tarea y el tiempo estimado de dedicación.

3.3.1 PLANTEAMIENTO

- **Planificación juegos** (dos semanas)

  En esta primera tarea se diseñó *Up, up! Nyumi!*, definiendo así sus objetivos, reglas y características principales, además de los objetos interactuables que contendría.

  Por otra parte, se estudiaron los distintos motores de juegos a utilizar, optando finalmente por Godot.

- **Estudio de Godot** (dos semanas)

En esta fase, se estudió el funcionamiento de Godot, el cual permite programar en los lenguajes de programación GDScript, C#, C++ y visual scripting. Optamos por desarrollar los juegos con GDScript.

3.3.2 IMPLEMENTACIÓN E INTEGRACIÓN

- **Pantallas de selección** (una semana)

  En esta fase programamos las pantallas de selección de personaje y selección de escenario. Incluye el guardar los elementos escogidos y el cambio entre pantallas.

- **Pantalla principal** (una semana y media)

  Esta tarea consiste en implementar el funcionamiento de la pantalla principal del juego: movimiento del escenario, aparición y desaparición de obstáculos, aparición de monedas y power-ups (no incluye la activación de estos objetos, sólo su visualización).

- **Control del personaje** (una semana)

  En esta fase programamos las mecánicas de movimiento del personaje (saltar a izquierda o derecha y deslizarse hacia abajo) y definimos sus parámetros de velocidad y altura. Esta tarea no incluye la implementación de las habilidades propias de cada personaje.

- **Controlador monedas y modo de juego** (dos días)

  Como nuestro videojuego se ejecuta sobre una máquina recreativa, debemos controlar si el jugador inserta alguna moneda, independientemente del momento de la partida en el que se encuentre y si decide jugar en modo de un solo jugador o multijugador. En esta fase nos encargamos de controlar los créditos de los que dispone el jugador, incrementando la cantidad cuando introduce monedas y decrementando al empezar una partida. También se encarga de negarle la partida al jugador en caso de no disponer suficientes créditos.

- **Habilidades personajes** (una semana)

  Por cada apariencia del personaje hay una habilidad especial diferente que se activa automáticamente transcurrida una determinada cantidad de tiempo. En esta fase nos encargamos de hacer que cada habilidad se cargue, se ejecute y se desactive automáticamente. Esta tarea no incluye la integración de las animaciones de cada habilidad.

- **Monedas y *power-ups*** *(dos semanas)*

  Durante el juego aparecen monedas y *power-ups* que el jugador puede recoger (implementado en la tarea *Pantalla principal*). En esta fase, programamos la activación de estos objetos. Las monedas hacen que la habilidad del personaje se cargue más rápido y al final de la partida suma puntos, mientras que los *power-ups* pueden usarse en beneficio propio o, en caso de que sea una partida multijugador,

pueden usarse contra los contrincantes. En este punto, solo implementamos los *power-ups* que benefician al jugador, mientras que los perjudiciales se implementaron junto al modo multijugador.

- **Colliders** (tres días)

  Al inicio se usaron *hitboxes*, menos precisas para probar el funcionamiento general del juego. En esta fase, hicimos que los colliders de cada objeto y personaje delimitaran lo mejor posible cada sprite, de manera que las colisiones del juego fueran lo más exactas posibles.

- **Multijugador** (una semana)

  Una vez que tuvimos la base del *singleplayer* en funcionamiento, pasamos a hacer el modo multijugador local. Esto incluye no sólo la pantalla dividida para jugar en una misma máquina, sino también la implementación de los *power-ups* para perjudicar al contrincante y tener en cuenta cuál de los dos jugadores es el ganador, o si hubo empate.

- **Integración música y SFX** (un día)

  Una vez finalizado el juego, integramos la música y los efectos sonoros que habíamos escogido previamente.

- **Integración vídeo tutorial** (un día)

  En esta fase nos encargamos de integrar el vídeo tutorial que habíamos montado con anterioridad. Este se reproduce en bucle mientras la máquina está inactiva.

- **Registro de mejores puntuaciones** (tres días)

  Como juego perteneciente al género arcade, el objetivo es conseguir la máxima puntuación de la máquina en la que se juega. Esta tarea consiste en llevar un registro con las tres mejores puntuaciones de la historia, guardadas en un archivo de texto (para tener una copia de seguridad en caso de que se apague el dispositivo), junto a los nombres de los usuarios. Además, en esta fase implementamos la visualización por pantalla y la actualización de los récords en caso de que fuera necesario, permitiendo que el jugador registre su nombre.

- **Integración assets vectorizados** (dos días)

  Esta tarea consiste en sustituir todos los sprites provisionales por los definitivos, ya vectorizados, coloreados e iluminados. También incluye la implementación e integración de todas las animaciones.

- **Debug** (una semana)

  En esta fase nos dedicamos a hacer diversas pruebas con dos objetivos principales: balancear el juego y corregir cualquier *bug* o error que podamos encontrar en el proceso.

- **Refinamiento** (dos días)

Una vez que el funcionamiento correcto del juego está comprobado, pasamos a pulir y refinar los pequeños detalles que puedan haber.

3.3.3 ESTÉTICA

- **Diseño personajes** (tres días)

  Up, up! Nyumi! está compuesto por cinco personajes que comparten la misma base. Todas sus apariencias fueron diseñadas a color.

- **Diseño escenarios** (tres días)

  En esta tarea diseñamos a color los diferentes escenarios que tendría el juego. Optamos por hacer diseños que acompañaran a cada una de las apariencias del personaje, aunque después el jugador pueda escogerlos de manera independiente.

- **Diseño menús** (cuatro días)

  Con el diseño de los menús nos referimos al diseño global y esquemático de las pantallas de selección de escenario, de personaje y la pantalla de final de partida.

- **Diseño interfaces** (tres días)

  En esta fase diseñamos los elementos que componen el HUD de cada pantalla, por ejemplo: cómo visualizamos los *power-ups* o cómo mostramos si se superó un nuevo récord.

- **Diseño objetos** (cuatro días)

  Con objetos nos referimos a todos aquellos elementos, interactuables o no, que no forman parte del HUD. Lo son principalmente: los obstáculos de cada escenario, los *power-ups* y las monedas.

- **Diseño animaciones** (una semana y media)

  En esta fase pensamos en todas las animaciones necesarias para el juego (voltereta al saltar, jetpack, congelar la puntuación del adversario, etc) e hicimos sus bocetos básicos.

- **Vectorizar diseños** (tres semanas)

  Esta es una de las tareas más laboriosas, la cual consiste en pasar cada uno de los diseños anteriores, incluidas las animaciones, a vectorial, además de recolorizarlos y ponerles la iluminación necesaria.

- **Vídeo tutorial** (una semana)[2]

---

[2] Esta tarea forma parte del bloque de estética ya que el **montaje** del vídeo encaja mejor en este grupo que en los otros cuatro.

Cuando una máquina recreativa está en desuso, reproduce en bucle un vídeo hasta que un usuario interactúa con ella. Así que en esta fase nos encargamos de diseñar y montar un vídeo que sirviera a la vez como demostración y como tutorial del juego.

- **Diseño vinilos** (una semana)

  En esta fase diseñamos los vinilos que cubren la estructura de la máquina arcade.

3.3.4 AUDIO

- **Búsqueda música y SFX** (dos días)

  Esta tarea consistía en buscar la música y los efectos sonoros que formarían parte del juego. Con música nos referimos a la canción de fondo en las pantallas de selección y la canción que suena durante una partida. Por otra parte, necesitábamos efectos sonoros que nos sirvieran como *feedback* hacia el jugador.

3.3.5 DOCUMENTACIÓN

- **Documentación** (7 meses)

  Consiste en escribir la memoria del proyecto, la cual debe explicar todo el proceso creativo, de producción y la valoración del resultado final. Es una tarea constante que realizamos a lo largo de todo el proyecto.

3.4 TIEMPO ESTIMADO Y DEFINITIVO

En esta sección, veremos la comparativa entre la planificación inicial y la planificación real.

Nótese que no se mostrarán los esquemas de las tareas de Planteamiento y Documentación ya que el tiempo estimado y definitivo coincidieron con un margen de error de uno o dos días.

En la *Figura 3.4.1 Diagrama de Gantt previsto* queda reflejado el estudio esperado del tiempo inicial, mientras que el resultado final, una vez concluido el proyecto, está reflejado en la *Figura 3.4.2 Diagrama de Gantt real*.

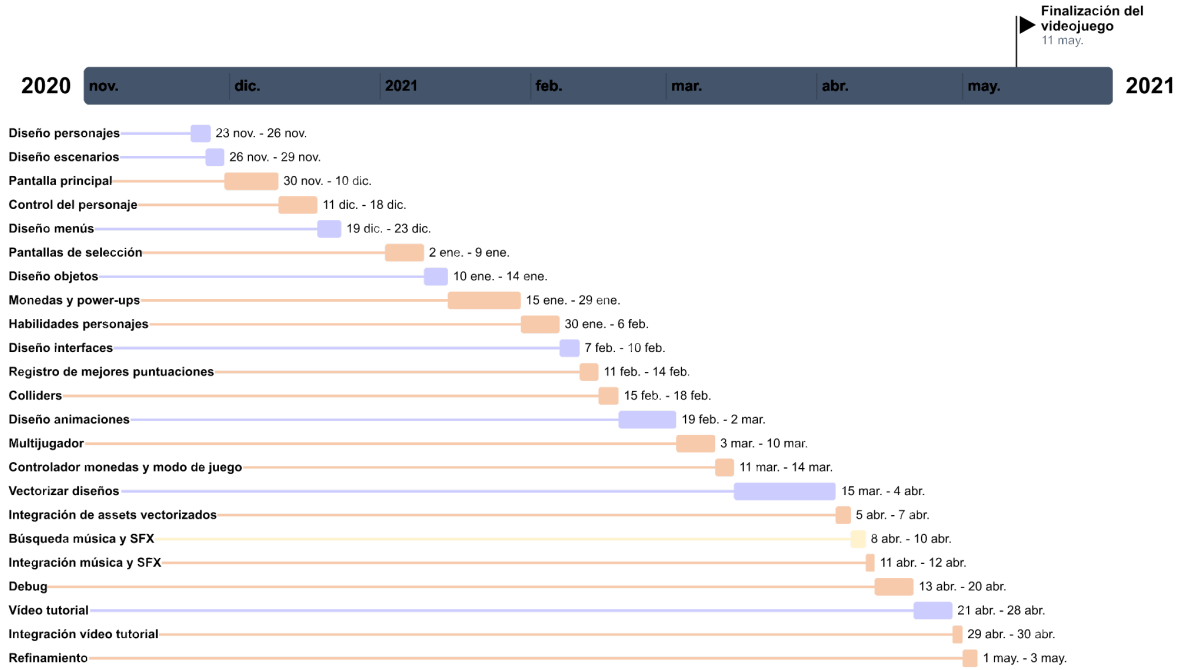


Figura 3.4.1 Diagrama de Gantt previsto

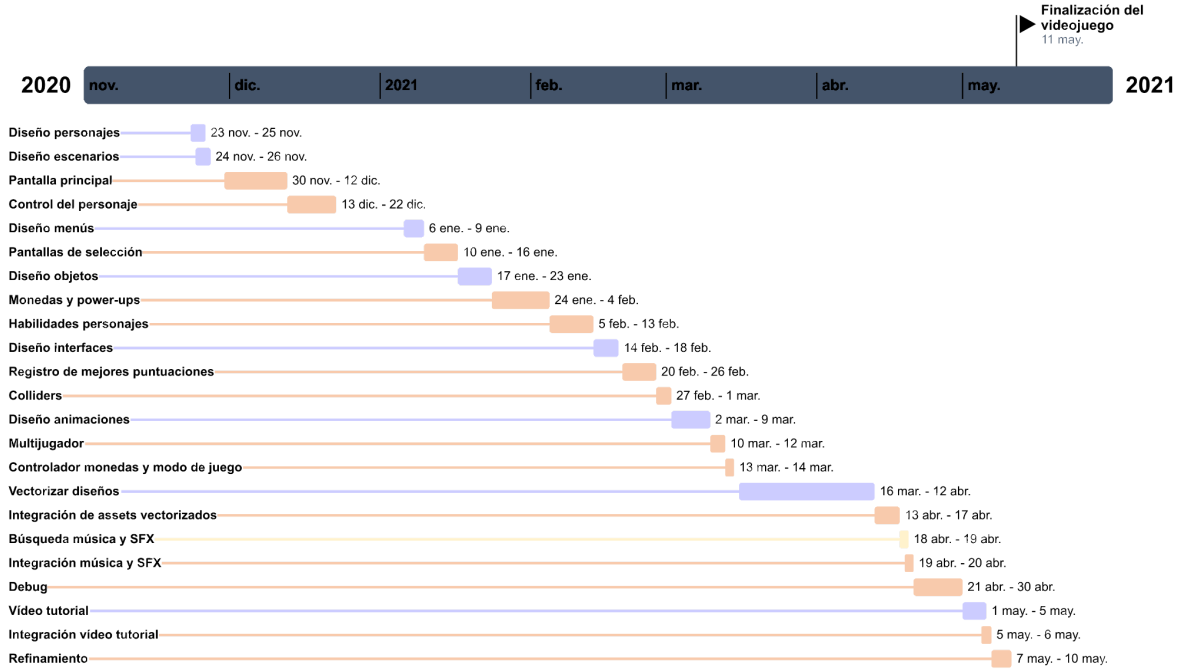


Figura 3.4.2 Diagrama de Gantt real

3.5 RESULTADOS ESPERADOS

Se espera poder diseñar e implementar un videojuego arcade completamente funcional, además de realizar el diseño para los vinilos de una máquina recreativa.

Del proyecto en su totalidad se espera que sea entretenido a la vez que adictivo para sus futuros usuarios, y que cumpla con todas las premisas y expectativas presentadas a lo largo de este documento.

4. MARCO DE TRABAJO Y CONCEPTOS PREVIOS

En este apartado enfatizaremos sobre dos conceptos básicos sobre los cuales trabajamos: la placa computadora Raspberry Pi y el motor de juegos Godot.

4.1 RASPBERRY PI

La Raspberry Pi es una placa de ordenador de bajo coste y tamaño reducido, la cual está formada por varios elementos: CPU, memoria RAM, puertos de entrada y salida de audio y vídeo, conectividad de red, ranura SD para almacenamiento, etc. Además, este microordenador es capaz de soportar varios componentes necesarios en un ordenador común, tales como teclados, ratones y monitores. Véanse las Figuras *4.1.1* y *4.1.2*.

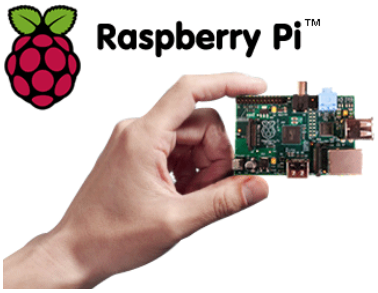


Figura 4.1.1 Raspberry Pi

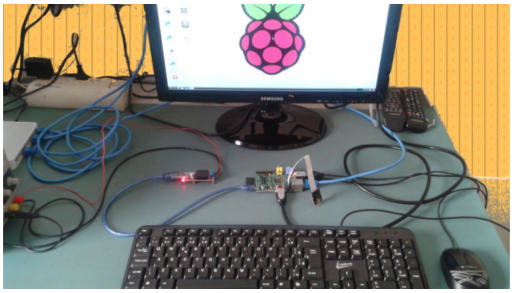


Figura 4.1.2 Raspberry Pi conectada

Nuestros únicos requisitos para el núcleo de la máquina recreativa eran la capacidad de ejecutar juegos sencillos a nivel de gráficos y jugabilidad, y la capacidad de controlar las interacciones con el usuario, las cuales están muy limitadas. Por ello, optamos por una Raspberry Pi, ya que es una placa económica y simple, mas con el potencial suficiente para nuestro proyecto.

4.2 GODOT

Godot es un motor gráfico de desarrollo de videojuegos, multiplataforma, gratuito y de código abierto, distribuido bajo la licencia MIT. Permite desarrollar videojuegos 2D y 3D mediante un sistema jerárquico de nodos y escenas. Además, incluye las herramientas necesarias para el desarrollo de manera centralizada y visual.

En nuestro caso, prácticamente cualquier motor de juegos podría haber sido un candidato prometedor, ya que nuestro videojuego arcade es 2D y no tiene gráficos potentes ni mecánicas complejas. No obstante, nos vimos condicionados por nuestro *hardware*.

Raspberry Pi es un procesador ARM[3] el cual trabaja sobre GNU/Linux, mientras que la mayoría de motores están pensados para procesadores x86 o para ARM que permiten exclusivamente sistemas operativos Android o iOS. Debido a esto, optamos por Godot, debido a que a pesar de no soportar la Raspberry Pi de forma nativa, al ser de código libre, nos permite compilarlo a partir del código fuente.

[3] Machine RISC Advanced (Máquina RISC Avanzada), donde RISC significa Reduced Instruction Set Computer (Ordenador con Conjunto Reducido de Instrucciones)

5. DISEÑO *UP, UP! NYUMI!*

Up, up! Nyumi! es el juego principal de nuestra recreativa. Es un videojuego de género arcade destinado al entretenimiento y disfrute de sus usuarios. Al diseñar *Up, up! Nyumi!* tuvimos presente desde un principio que su objetivo principal sería el de atraer y cautivar al mayor número de jugadores posible. Por ello, quisimos que fuera un juego rápido, simple y, por encima de todo, adictivo. Para lograr este último rasgo consideramos que la competitividad era un punto crítico y, por este motivo, quisimos implementar un modo multijugador local.

En este capítulo describiremos con mayor detalle el diseño de nuestro videojuego junto a sus características y propiedades más notables.

5.1 NARRATIVA

Nyumi es una especie de otra dimensión compuesta por un material gelatinoso desconocido. Actualmente se tiene conocimiento de cinco especímenes diferentes con habilidades únicas, aunque ninguno de ellos pudo ser capturado todavía. No obstante, los datos recogidos mediante la observación son los siguientes:

- Esta especie sufre un tipo de disfunción visual que no le permite ver más allá de los objetos más cercanos a ella.
- Pueden poseer una de estas habilidades:
    - Capacidad de eliminar una cantidad de objetos limitada
    - Capacidad de saltar ilimitadamente sin necesidad de reposo durante unos segundos
    - Mejora en la detección de objetos brillantes de manera momentánea
    - Volar un breve período de tiempo
- Cuando utilizan su habilidad, necesitan un período de descanso hasta poder volver a usarla.

El jugador debe escoger a uno de ellos para controlarlo y tratar de escapar de todos los científicos y militares que lo quieren aprehender. Para ello, deberá saltar entre paredes, subiendo lo máximo posible mientras evita los obstáculos que se interponen en su camino.

5.2 ESPACIO DEL JUEGO

Nuestro videojuego sucede en un único espacio, el cual tiene cuatro diseños diferentes: una ciudad, un bosque, un castillo y un mundo de caramelo. La única diferencia entre estos espacios es meramente su estética, las demás cualidades (tipos de power-ups o ratio de aparición de objetos, por ejemplo) son compartidas.

En la *Figura 5.2.1 Esquema espacio del juego* pueden verse reflejados estos cuatro escenarios.

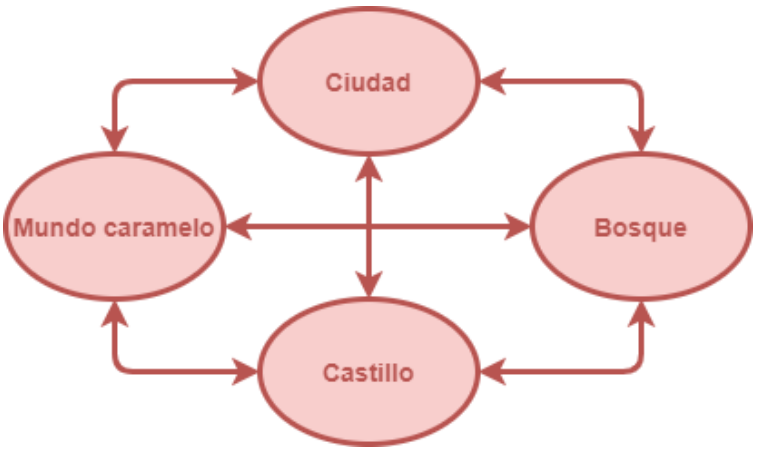


Figura 5.2.1 Esquema espacio del juego

Nótese que en la figura anterior los arcos son bidireccionales debido a que todos los espacios están disponibles desde un inicio, es decir, el usuario puede jugar en cualquiera de los cuatro ambientes, sin la necesidad de desbloquear uno u otro primero.

5.3 *GAMEPLAY*

En este capítulo explicaremos las acciones que puede realizar el jugador en *Up, up! Nyumi!*, con qué controles puede ejecutarlas y, además, se detallarán los retos y objetivos.

5.3.1 ACCIONES

Las acciones que puede realizar el jugador son las siguientes:

- **Saltar:** el jugador puede saltar hacia la derecha o hacia la izquierda. De base puede realizar hasta cuatro saltos sin la necesidad de tocar una pared[4].
- **Acelerar caída:** cuando el personaje no está saltando, se desliza automáticamente hacia abajo por la pared en la que está apoyado. El jugador puede acelerar esta caída si lo desea.
- **Volar:** cuando el personaje usa un jetpack o se activa la habilidad de volar, el jugador puede moverse por el escenario sin necesidad de saltar entre las paredes. Mientras vuela el jugador puede moverse en las siguientes direcciones: arriba, abajo, izquierda y derecha.
- **Recoger objetos:** para recolectar una moneda o un power-up (en caso de que haya un espacio libre), el jugador sólo debe hacer que su personaje toque ese objeto, el cual se recogerá automáticamente.

[4] La habilidad de uno de los personajes es la capacidad de saltar de manera ilimitada durante unos segundos.

- **Usar power-up:** durante la partida se pueden acumular hasta tres power-ups, los cuales se pueden usar en cualquier momento activándose manualmente. Cada power-up tendrá un funcionamiento distinto.

  Nota: hay dos tipos de power-ups: beneficiosos y perjudiciales. Los primeros aportan algún tipo de ayuda/mejora al jugador que los activó, mientras que los segundos sólo aparecen en el modo multijugador y sirven para perjudicar al contrincante.

5.3.2 JERARQUÍA DE RETOS

En los siguientes subapartados encontraremos las representaciones gráficas de las jerarquías de reto para los dos modos de juego: un solo jugador y multijugador.

En ambos la leyenda del esquema es la siguiente:

- **Rectángulos rojos:** representan las misiones
- **Elipses naranjas:** representan las submisiones
- **Elipses amarillas:** representan los retos no atómicos
- **Círculos grises:** representan los retos atómicos

5.3.2.1 *SINGLEPLAYER*

Cuando el usuario juega solo, su misión principal es superar uno de los tres mejores récords registrados. Para ello debe evitar los obstáculos y no morir rápidamente.

Para esquivar los obstáculos debe:

- **Moverse**, ya sea saltando, volando o cayendo más deprisa
- **Usar power-ups**, los cuales debe recoger previamente
- **Activar su habilidad especial** (si es que la tiene), la cual puede recargar con mayor facilidad si recoge todas las monedas que encuentre en su camino.

La *Figura 5.3.1 Jerarquía de retos singleplayer* representa la jerarquía de retos de manera visual:

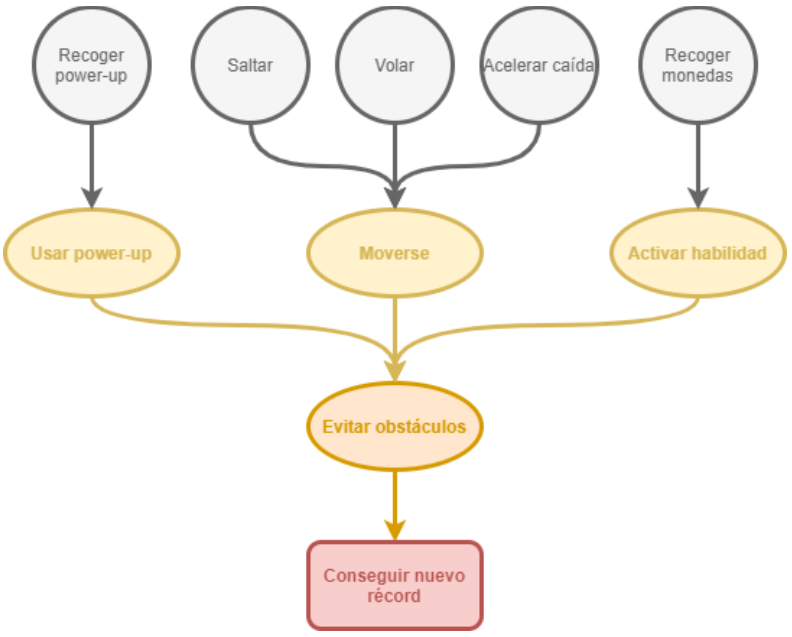


Figura 5.3.1 Jerarquía de retos *singleplayer*

5.3.2.2 *MULTIPLAYER*

Cuando se decide jugar en multijugador, el objetivo principal pasa a ser derrotar al contrincante, mientras que conseguir una de las mejores puntuaciones queda en segundo plano.

Para ganar al oponente, se debe sobrevivir más tiempo que él, con lo cual hay que realizar las mismas acciones que si se quisiera batir un récord, pero además de esto, podemos perjudicarlo. Para sabotear al otro jugador se pueden usar los power-ups perjudiciales, aunque, al igual que los beneficiosos, deben recogerse previamente.

La *Figura 5.3.2 Jerarquía de retos multiplayer* representa la jerarquía de retos para el modo multijugador:

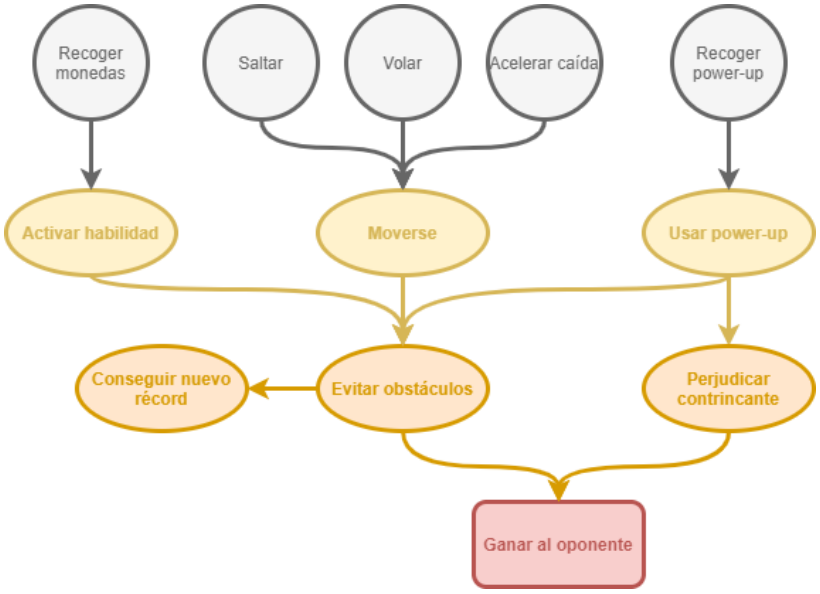


Figura 5.3.2 Jerarquía de retos *multiplayer*

5.3.3 CONTROLES

Up, up! Nyumi! está pensado para que se ejecute sobre una máquina recreativa; no obstante, también está adaptado para ordenador. En esta sección explicaremos los controles (para ambos dispositivos) en las situaciones siguientes:

- Pantalla inicial
- Pantalla de selección de escenario y personaje
- Cuando el jugador se encuentra en partida
- Pantalla final

5.3.3.1 CONTROLES PRINCIPALES

En este apartado se describirán los controles del juego para una máquina recreativa.

Como podrá observarse, los controles están duplicados, ya que *Up, up! Nyumi!* permite el modo multijugador. En este caso, los controles del segundo jugador son los situados a la derecha de la imagen.

PANTALLA INICIAL

Cuando la máquina está inactiva, muestra un vídeo en bucle hasta que se pulse cualquier botón o se mueva el joystick. En este caso, se procederá a mostrar la pantalla inicial, donde pueden visualizarse las tres mejores puntuaciones, los créditos actuales y el modo de juego.

Una vez el jugador se encuentre en esta pantalla, los controles serán los mostrados en la *Figura 5.3.3 Controles pantalla inicial*.

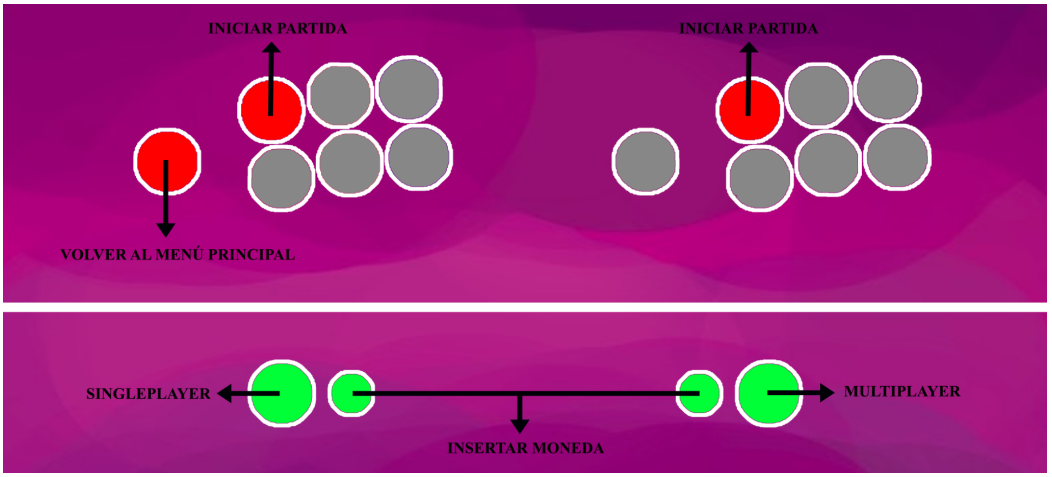


Figura 5.3.3 Controles pantalla inicial

En esta pantalla se podrá seleccionar el modo de juego, insertar monedas e iniciar la partida pulsando el primer botón de cada jugador. En caso de modo multijugador, la partida no empezará hasta que ambos jugadores hayan pulsado su botón de inicio.

Por otra parte, si el jugador mueve el joystick hacia abajo y lo mantiene en esta posición durante algunos segundos, puede volver al menú principal de selección de juegos.

PANTALLAS DE SELECCIÓN

Antes de empezar a jugar, el usuario debe escoger tanto el escenario como el personaje. Los controles son los representados en la *Figura 5.3.4 Controles pantalla selección*:

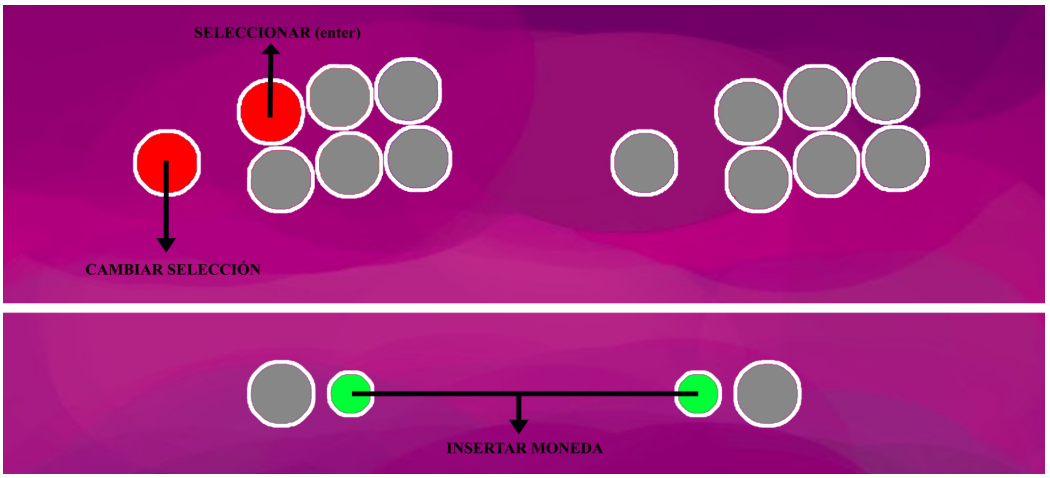


Figura 5.3.4 Controles pantalla selección

Como puede verse, se utiliza el joystick para cambiar de selección, mientras que se emplea el primer botón del jugador 1 para confirmar esta decisión. En el modo multijugador también es el jugador 1 el que controla la pantalla. Por otra parte, también se pueden insertar monedas.

IN-GAME

En la *Figura 5.3.5 Controles in-game* podemos ver un esquema básico de los controles de la máquina cuando el jugador se encuentra en partida.

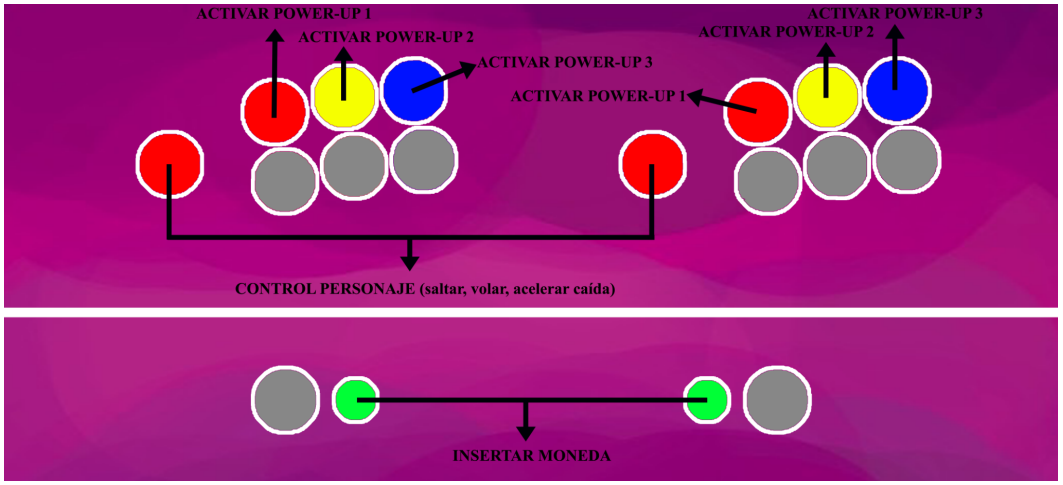


Figura 5.3.5 Controles *in-game*

Los controles situados a la izquierda controlan la partida del primer jugador, mientras que los situados a la derecha controlan al segundo en caso de jugar en multijugador.

El joystick se utiliza para mover al personaje, mientras que los tres botones superiores se usan para activar los power-ups que se hayan conseguido. Por último, el jugador puede insertar monedas pulsando los botones inferiores situados más al interior.

PANTALLA FINAL

Una vez finalizada la partida, llegamos a la pantalla final, en la cual se muestra la puntuación obtenida por cada jugador junto a las tres mejores, si se superó uno de los récords registrados, los créditos disponibles y el modo de juego.

En caso de superar uno de los récords, el jugador deberá introducir su nombre. En este caso, los controles son los mostrados en la *Figura 5.3.6 Controles introducción nombre*.

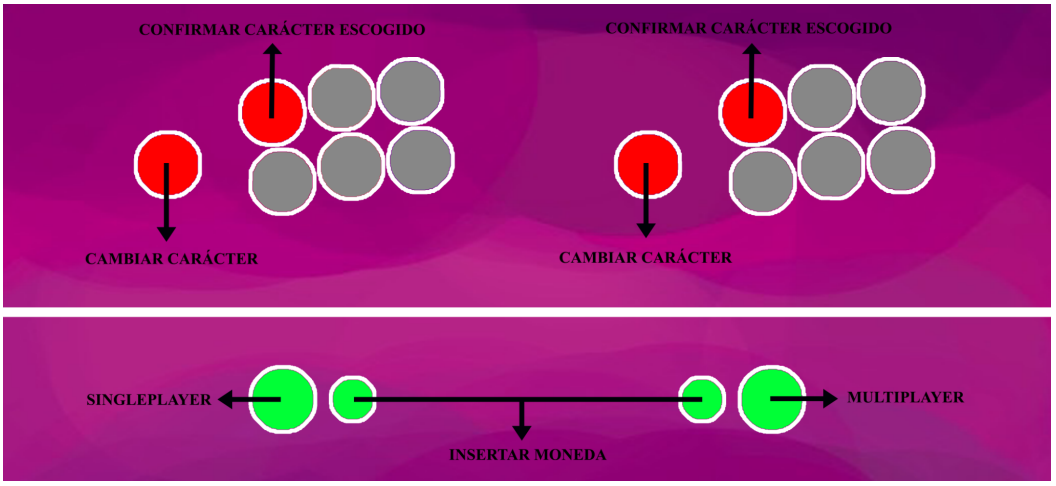


Figura 5.3.6 Controles introducción nombre

Una vez escogido el nombre del jugador, los controles cambian, pasando a ser los mismos que en la pantalla inicial (véase *Figura 5.3.3 Controles pantalla inicial*)

5.3.3.2 CONTROLES ALTERNATIVOS

En esta sección, explicaremos los controles del juego para ordenador. Debe tenerse en cuenta que el usuario puede insertar una moneda en cualquier momento del juego pulsando la tecla C o salir de éste pulsando *Esc*.

PANTALLA INICIAL

Antes de llegar a la pantalla inicial, el juego reproduce un vídeo. Éste puede saltarse pulsando cualquiera de las teclas útiles[5] del juego.

[5] Aquellas que se usan en cualquier pantalla del juego, independientemente de su función.

Una vez que el jugador se encuentre en la pantalla inicial, podrá realizar las siguientes acciones:

- **Seleccionar modo *singleplayer***, pulsando la tecla N.
- **Seleccionar modo multijugador**, pulsando la tecla M.
- **Volver al menú principal**, manteniendo pulsada la tecla B durante unos segundos.
- **Iniciar partida**, apretando la tecla Enter, ya sea del teclado principal o del numpad.

PANTALLAS DE SELECCIÓN

En la pantalla de selección, de escenario o personaje, el usuario puede hacer lo siguiente:

- **Cambiar la selección**, pulsando A para ir a la izquierda o D para ir a la derecha.
- **Confirmar selección**, pulsando la tecla Enter del teclado principal.

IN-GAME

Por otro lado, los controles durante una partida son los siguientes:

- **Jugador 1:**
    - **WASD** para el movimiento del personaje (ya sea saltar, volar o acelerar la caída).
    - Teclas **1**, **2** y **3** para activar los power-ups correspondientes.
- **Jugador 2:**
    - **Flechas** para moverse.
    - Teclas **1**, **2** y **3** del **numpad** para activar los power-ups.

PANTALLA FINAL

En caso de que un jugador haya superado uno de los récords, primero deberá registrar su nombre. Para ello, las acciones que debe hacer son las siguientes:

- **Cambiar carácter**
    - El jugador 1 podrá hacerlo con las teclas A y D.
    - El jugador 2 deberá pulsar las flechas izquierda o derecha.
- **Seleccionar carácter**
    - Pulsando la tecla Enter del teclado principal para el jugador 1.

- Pulsando la tecla Enter del numpad para el jugador 2.

Del mismo modo que en la máquina recreativa, una vez escogido el nombre del jugador, los controles cambian, pasando a ser los mismos que en la pantalla inicial.

5.4. ECONOMÍA

La economía de nuestro juego es un tanto peculiar. El jugador no interactúa directamente sobre los elementos que la conforman, sino que es el propio juego quien los gestiona.

Por un lado, tenemos las **monedas** del juego, que actúan simultáneamente como *sources* y como *traders*. Las monedas son un recurso ilimitado con dos funciones básicas: durante el juego cada moneda recogida recarga la habilidad especial del personaje escogido (de manera casi imperceptible para el jugador); por otra parte, una vez finalizada la partida, cada moneda acumulada se cambia por puntos que se suman a la puntuación final obtenida.

Por otro lado, tenemos los **créditos**, los cuales se consiguen a cambio de dinero real del jugador. El usuario puede intercambiar cierta cantidad monetaria por créditos, los cuales sirven para poder jugar partidas. Este elemento funciona como *drain*, ya que es un recurso que se gasta para poder jugar, aunque puede ser recuperado si se supera uno de los tres mejores récords registrados en esa máquina.

5.5 NIVELES

Up, up! Nyumi! consta de un único nivel interminable y rejugable. Para evitar que el jugador aprenda patrones y memorice el escenario, haciéndose un juego demasiado repetitivo, las posiciones de los obstáculos y la aparición de objetos es completamente aleatoria[6].

5.5.1 AUMENTO DE LA DIFICULTAD

Por otra parte, para evitar el hastío del jugador y mantenerlo en el flow-state[7] hacemos que la partida aumente progresivamente su dificultad. Para ello, lo que hacemos es:

- Aumentar la velocidad a la que se mueven los escenarios y, por ende, los obstáculos.
- Variar los valores del movimiento del jugador, haciendo que caiga más rápido, pero aumentando su velocidad de salto para compensarlo.
- Aparición de obstáculos con mayor frecuencia.
- Mantener la frecuencia de aparición de monedas y power-ups, para que el jugador aprenda a gestionarlos mejor y jugar no se haga demasiado fácil.

---

[6] Exceptuando los tres primeros obstáculos, los cuales aparecen siempre en las mismas posiciones para que el jugador tenga unos segundos de respuesta automática antes de concentrarse para jugar.
[7] Estado subjetivo en el cual una persona se involucra completa y exclusivamente en la actividad que está ejecutando, olvidando todo lo externo a ella.

5.6 SISTEMA DE GUARDADO

Nuestro videojuego tiene un sistema de guardado automático de las mejores puntuaciones (junto a los nombres) de la máquina, evitando así que se pierdan los récords en caso de una avería o fallo inesperado.

El fichero que contiene estos datos es local, de manera que, si hubiera más de una máquina recreativa que ejecute *Up, up! Nyumi!*, cada una de ellas tendría sus propias puntuaciones. No obstante, no se descarta la opción de en un futuro guardar los puntos en un servidor web y que todas las máquinas compartan los mismos datos.

5.7 *FLOWCHART*

La *Figura 5.7.1 Flowchart* muestra la visión global del juego a nivel básico.

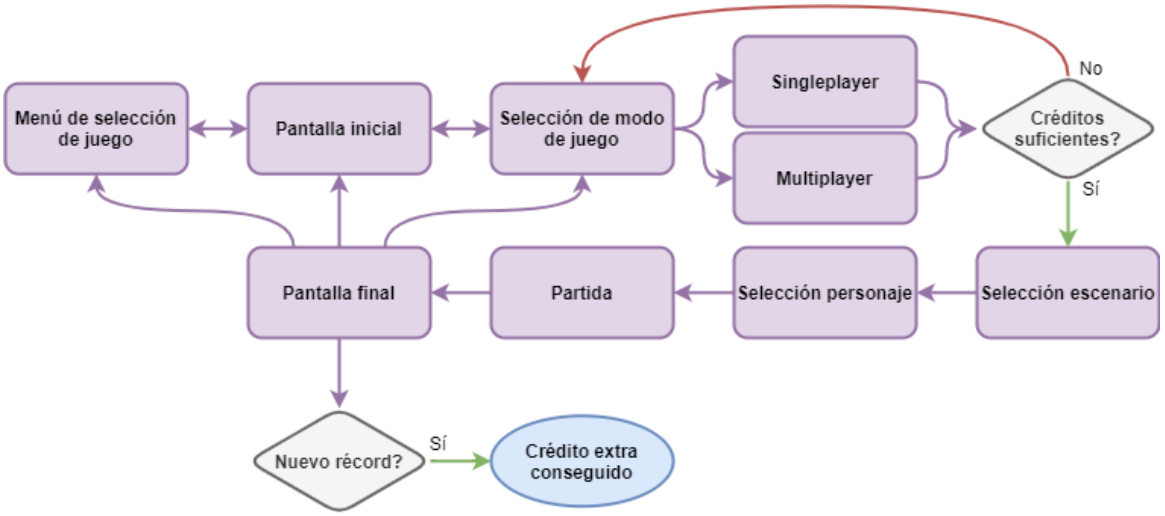


Figura 5.7.1 *Flowchart*

Nota: la acción “Insertar moneda” no se representa en el esquema ya que el jugador puede hacerlo en cualquier momento del juego, ya sea en la pantalla inicial o final, durante la partida o en el menú de selección. Se optó por omitirlo en el esquema para no perder legibilidad.

5.8 INTERFAZ GRÁFICA

En este capítulo explicaremos el conjunto de escenas que tiene nuestro videojuego y detallaremos los elementos más importantes de estas.

5.8.1 PANTALLA INICIAL

Es la pantalla que se ve antes de empezar a jugar. Desde esta se puede volver al menú principal de selección de juegos o iniciar una nueva partida. Véase la *Figura 5.8.1 Pantalla inicial*.

Figura 5.8.1 Pantalla inicial

En el centro podemos ver las mejores puntuaciones y las instrucciones para el jugador, ya sea “Insertar moneda” o “Enter para jugar”; mientras que en la parte inferior izquierda se muestran los créditos totales y a su derecha, el número de jugadores de la siguiente partida.

5.8.2 PANTALLAS DE SELECCIÓN

En el juego hay dos pantallas de selección: una para el escenario y la otra para el personaje. Véanse las figuras *5.8.2* y *5.8.3*. Ambas siguen el mismo modelo, cambiando únicamente las imágenes a seleccionar y su descripción.

Figura 5.8.2 Interfaz selección escenario

Figura 5.8.3 Interfaz selección personaje

La información más importante para el jugador es el tiempo restante (esquina inferior izquierda), las flechas (que indican si hay más elementos a izquierda y derecha) y cómo seleccionar el elemento (animación de un botón en la esquina inferior derecha).

5.8.3 PARTIDA

En los siguientes subapartados veremos la interfaz de una partida en ambos modos de juego, junto a sus elementos más destacables.

5.8.3.1 *SINGLEPLAYER*

En la *Figura 5.8.4 Pantalla singleplayer* podemos ver la estructura general de la interfaz de una pantalla en modo de un solo jugador.

Figura 5.8.4 Pantalla *singleplayer*

La pantalla está compuesta por los siguientes elementos de información:

- **Monedas recogidas**, situadas arriba a la izquierda
- **Puntuación**, situada arriba a la derecha
- **Barra de carga de la habilidad**, situada en la parte inferior. Empieza con un relleno blanco y comienza a pintarse de amarillo, simulando que la habilidad del personaje se está cargando. Cuando está completa, se indica parpadeando y cambiando de color (véanse *Figuras 5.8.5 y 5.8.6*).

Figura 5.8.5 Barra de habilidad cargada en distintos porcentajes

Figura 5.8.6 Barra de habilidad activada

- **Power-ups en posesión,** situados a la derecha de la barra de carga de la habilidad. El jugador puede tener hasta un máximo de tres power-ups simultáneos. Cada uno de ellos se representa con un círculo, en gris si está vacío o, en caso contrario, con el dibujo del power-up que lo está ocupando. Ver *Figura 5.8.7 Interfaz power-ups*.

Figura 5.8.7 Interfaz power-ups

5.8.3.2 *MULTIPLAYER*

En la *Figura 5.8.8* podemos ver la pantalla de juego en modo multijugador.

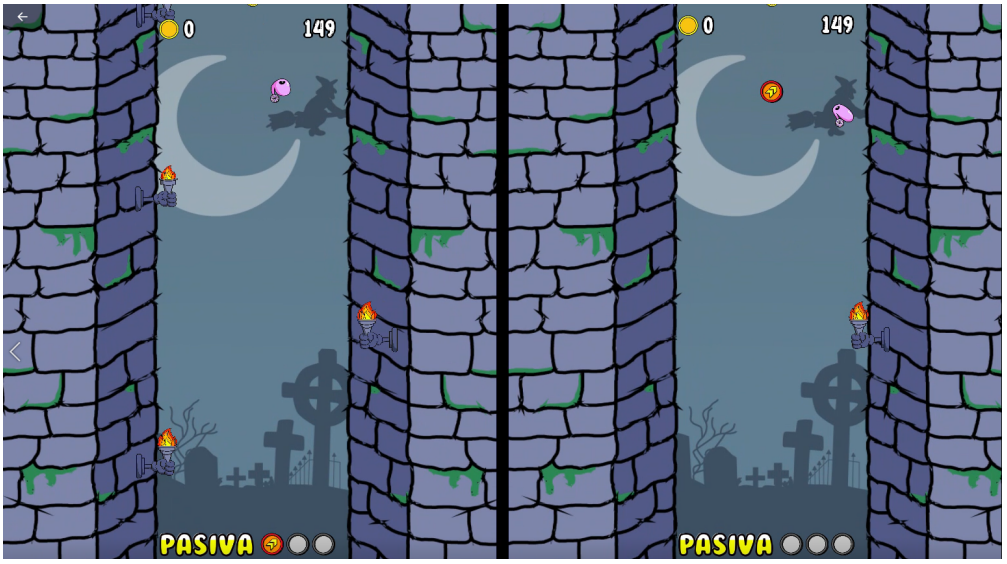


Figura 5.8.8 Pantalla multijugador

Como se puede observar en la figura anterior, se trata de una pantalla dividida, en la que se muestra la partida del primer jugador a la izquierda y del segundo a la derecha. Ambas zonas tienen la misma estructura que en el modo de un solo jugador, con los mismos elementos y las mismas posiciones.

5.8.4 PANTALLA FINAL

Una vez finalizada la partida, se muestra una pantalla con la puntuación lograda por el jugador, los tres récords registrados y si se superó alguno de ellos, en cuyo caso se deberá insertar el nombre. Además, se muestra la totalidad de créditos y el número de jugadores de la siguiente partida.

5.8.4.1 *SINGLEPLAYER*

En este caso, la estructura es prácticamente idéntica a la de la pantalla inicial. Los cambios principales son la añadidura de las visualizaciones de:

- Los **puntos conseguidos** por el usuario, situados en la parte central entre las mejores puntuaciones y el texto informativo. Ver *Figura 5.8.9*.

Figura 5.8.9 Pantalla final *singleplayer*

- Si se logró un nuevo récord, en cuyo caso debe introducirse el nombre del jugador. Ver *Figura 5.8.10.*

Figura 5.8.10 Nuevo récord *singleplayer*

5.8.4.2 *MULTIPLAYER*

La siguiente imagen muestra la pantalla final de una partida en multijugador, en la cual el jugador 2 se proclamó campeón, aunque sin superar ninguna de las tres mejores puntuaciones. Ver *Figura 5.8.11*.

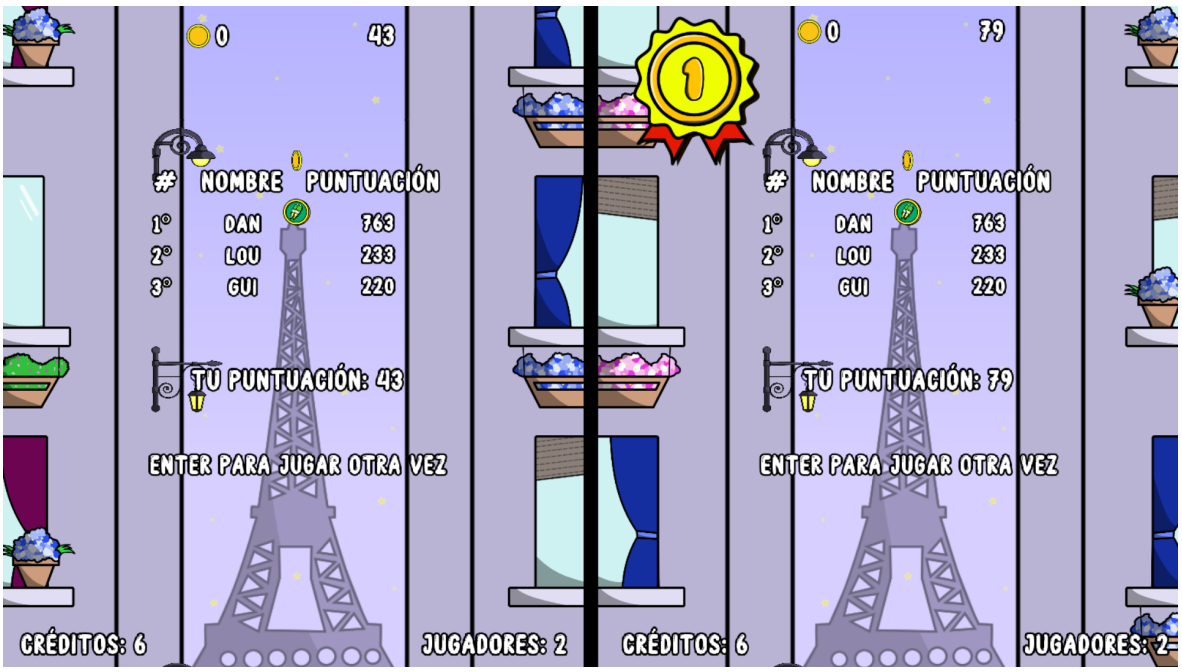


Figura 5.8.11 Pantalla final *multiplayer*

En caso de haberse superado alguna de las tres mejores puntuaciones, la pantalla final tendría la visualización mostrada en la *Figura 5.8.12*.

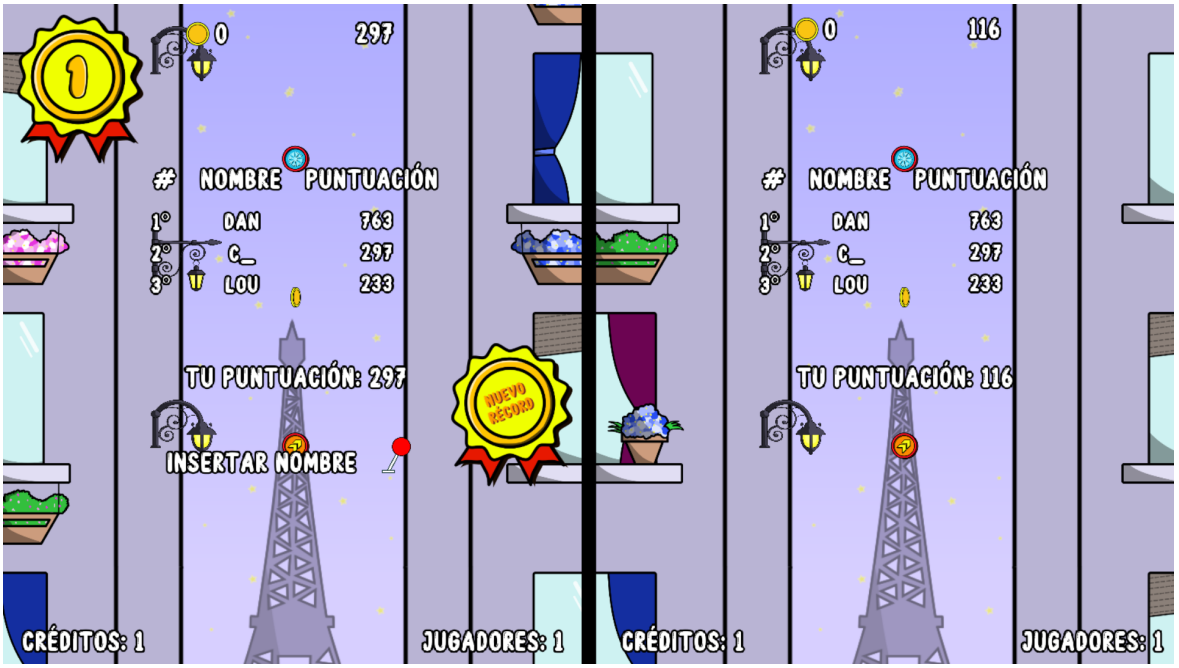


Figura 5.8.12 Nuevo récord *multiplayer*

5.9 ARTE DEL JUEGO

Una de las características principales de los juego arcade es su diseño minimalista a nivel de diseño y de controles. En este capítulo mostraremos el diseño artístico de todos los

elementos que conforman nuestro juego, el cual decidimos que tuviera un estilo *cartoon*, amigable y colorido.

Como podrá observarse no se usan elementos con puntas, ya que estos denotan agresividad, por lo cual se redondearon prácticamente todas las esquinas, exceptuando aquellos detalles en los que era imprescindible.

Cabe remarcar que **todos** los assets y spritesheets utilizados en nuestro juego son de creación propia.

5.9.1 MENÚS DE SELECCIÓN

Para los menús de selección tuvimos que crear el cuerpo central (*Figura 5.9.1 Spritesheet menú selección*) y una pequeña animación para indicar al jugador qué botón ha de pulsar para continuar (*Figura 5.9.2 Spritesheet animación OK*).

Figura 5.9.1 *Spritesheet* menú selección

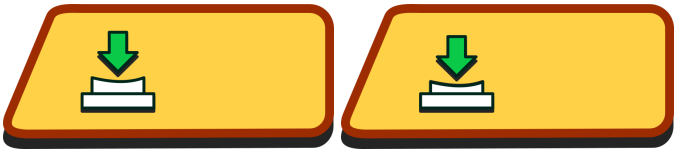


Figura 5.9.2 *Spritesheet* animación OK

Por otra parte, hicimos spritesheets con los escenarios y personajes posibles a seleccionar (*Figura 5.9.3 Spritesheet selección escenarios* y *Figura 5.9.4 Spritesheet selección personajes*). Éstos se muestran de uno en uno en el centro del cuerpo central junto a su descripción.

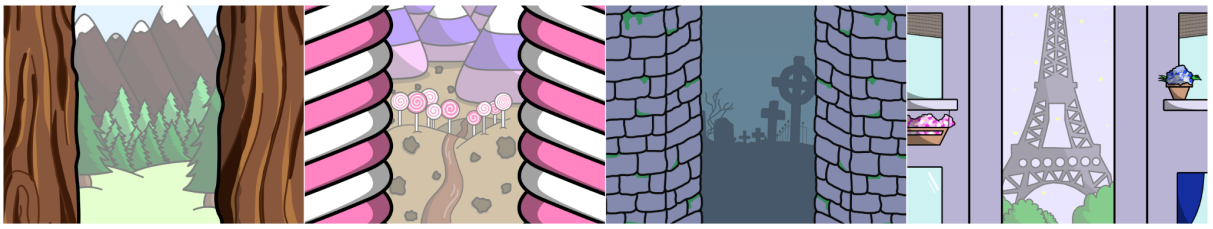


Figura 5.9.3 *Spritesheet* selección escenarios

Figura 5.9.4 *Spritesheet* selección personajes

5.9.2 ESCENARIOS

Como comentamos en capítulos anteriores, *Up, up! Nyumi!* actualmente consta de cuatro escenarios diferentes implementados: una ciudad, un castillo, un bosque y un mundo de caramelo. Por cada uno de ellos hicimos un fondo, las paredes que hay por encima y diferentes obstáculos.

5.9.2.1 BOSQUE MÁGICO

La *Figura 5.9.5 Fondo bosque* muestra el fondo del bosque, el cual queda tapado por las paredes (*Figura 5.9.6 Paredes bosque*).

Por otra parte, podemos ver una parte del conjunto de las imágenes anteriores en la *Figura 5.9.7 Escenario bosque*.

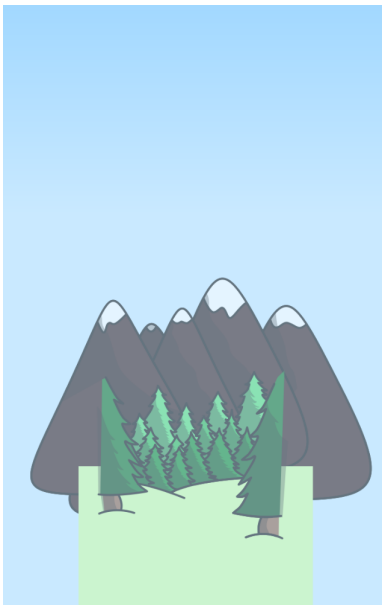


Figura 5.9.5 Fondo bosque

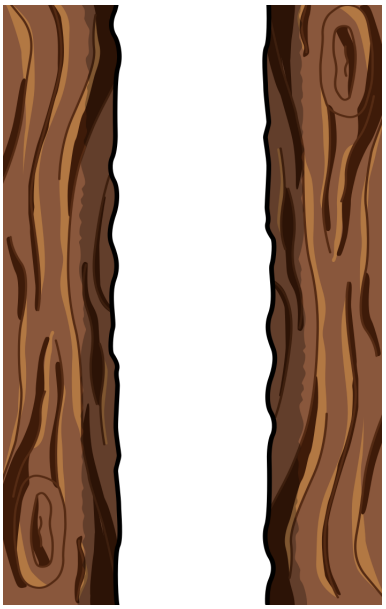


Figura 5.9.6 Paredes bosque

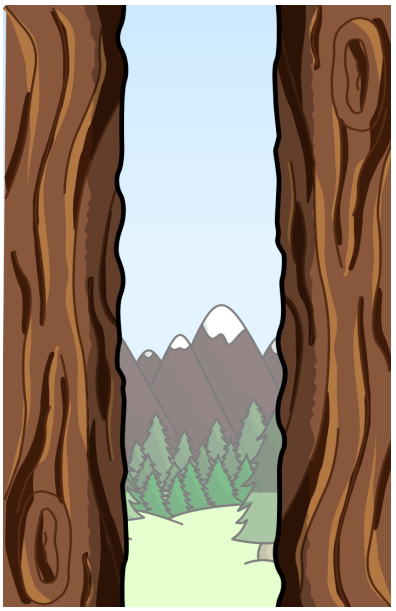


Figura 5.9.7 Escenario bosque

Por otra parte tenemos los obstáculos, mostrados en las Figuras *5.9.8* y *5.9.9*.

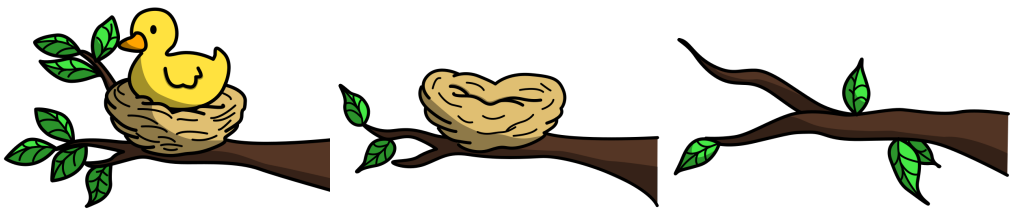


Figura 5.9.8 Obstáculos bosque derecha

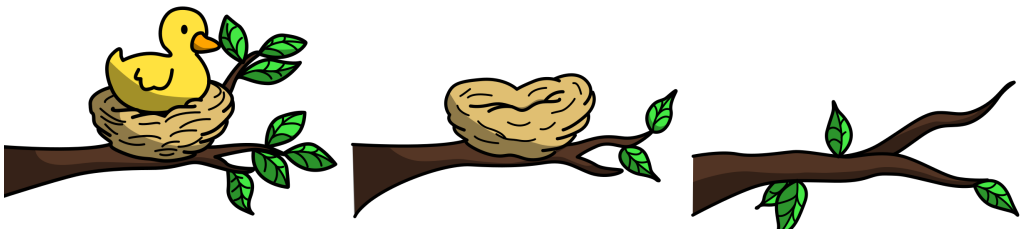
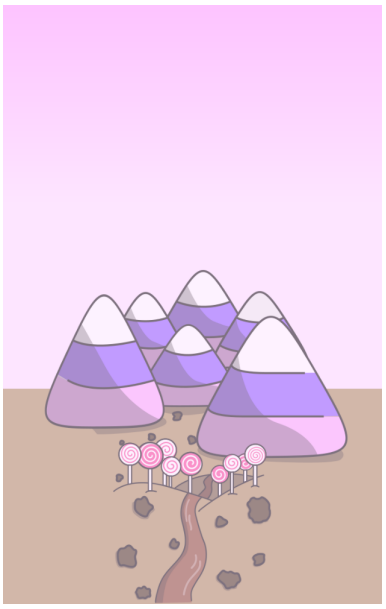


Figura 5.9.9 Obstáculos bosque izquierda

5.9.2.2 MUNDO PIRULETA

La *Figura 5.9.10 Fondo caramelo* muestra el fondo de este mundo, mientras que la *Figura 5.9.11 Paredes caramelo*, muestra sus paredes.

Por otra parte, podemos ver una parte del conjunto de las imágenes anteriores en la *Figura 5.9.12 Escenario caramelo*.



5.9.10 Fondo caramelo

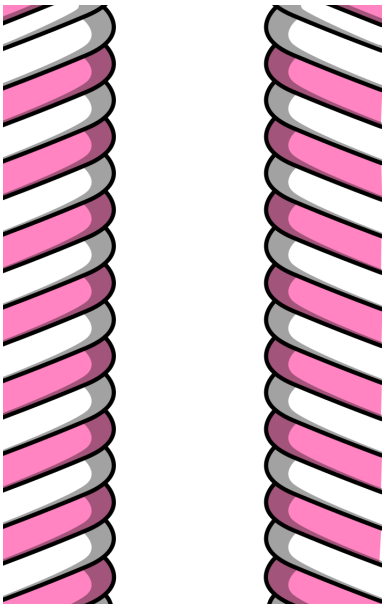


Figura 5.9.11 Paredes caramelo

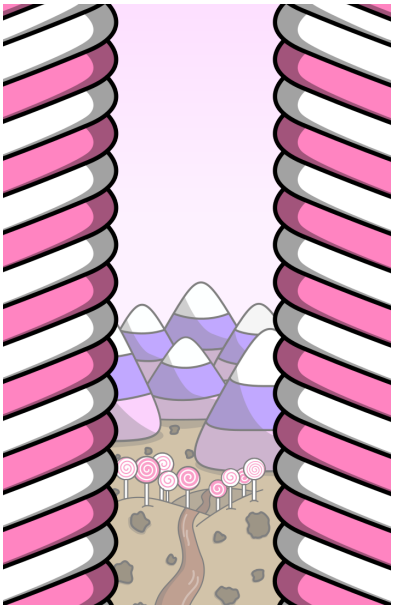


Figura 5.9.12 Escenario caramelo

Por otra parte tenemos los obstáculos, mostrados en las Figuras *5.9.13* y *5.9.14*.

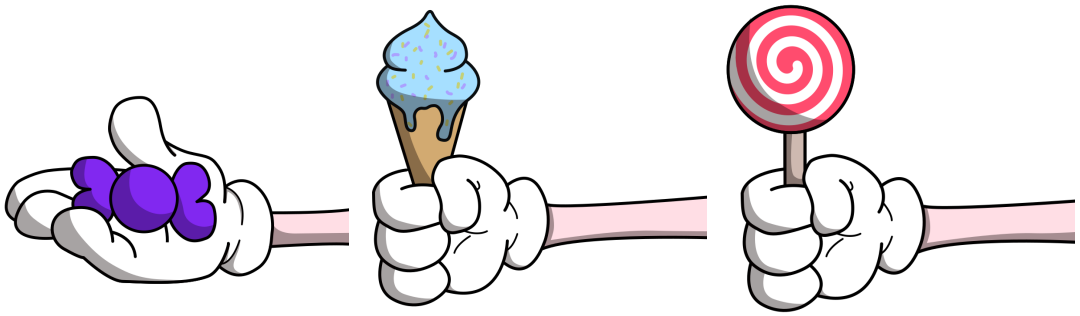


Figura 5.9.13 Obstáculos caramelo derecha

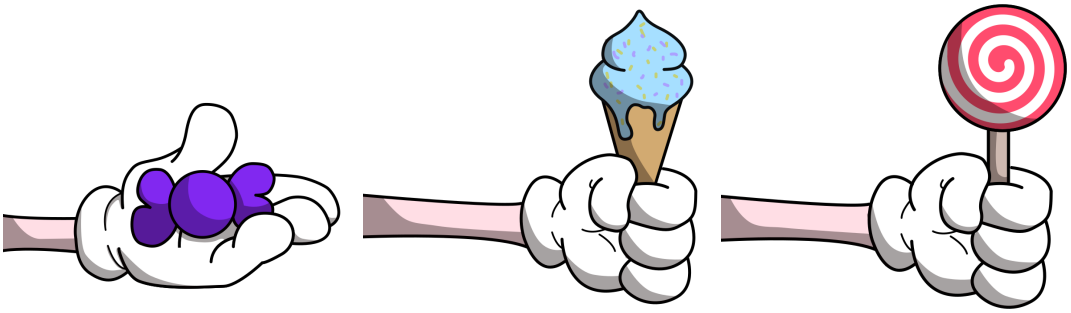


Figura 5.9.14 Obstáculos caramelo izquierda

5.9.2.3 CASTILLO ENCANTADO

El fondo del castillo está representado por la *Figura 5.9.15 Fondo castillo* y las paredes por la *Figura 5.9.16 Paredes castillo*.

Por otra parte, el conjunto de las imágenes anteriores queda reflejado en la *Figura 5.9.17 Escenario castillo*.

Figura 5.9.15 Fondo castillo

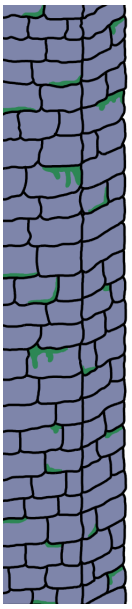


Figura 5.9.16 Paredes castillo

Figura 5.9.17 Escenario castillo

Los obstáculos de este ambiente son los que pueden verse en las Figuras *5.9.18* y *5.9.19*.

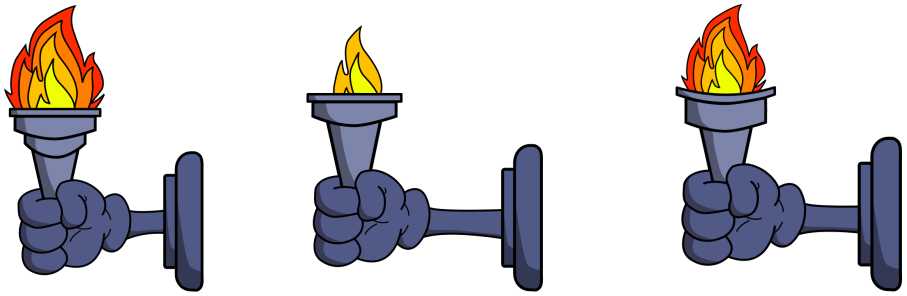


Figura 5.9.18 Obstáculos castillo derecha

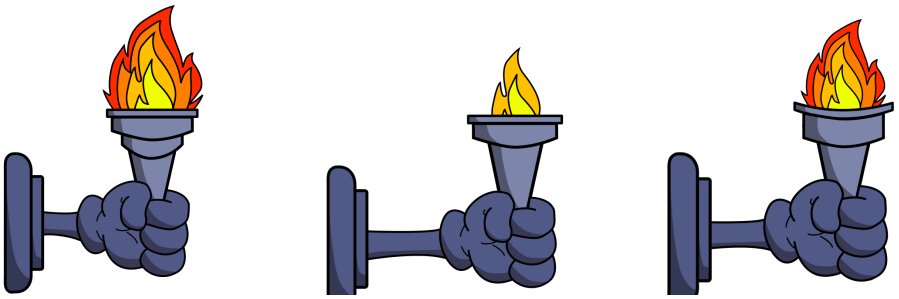


Figura 5.9.19 Obstáculos castillo izquierda

5.9.2.4 CIUDAD NOCTURNA

El fondo de la ciudad queda reflejado en la *Figura 5.9.20 Fondo ciudad* y sus paredes en la *Figura 5.9.21 Paredes ciudad*.

Por otra parte, el conjunto de las imágenes anteriores queda representado por la *Figura 5.9.22 Escenario ciudad*.

Figura 5.9.20 Fondo ciudad

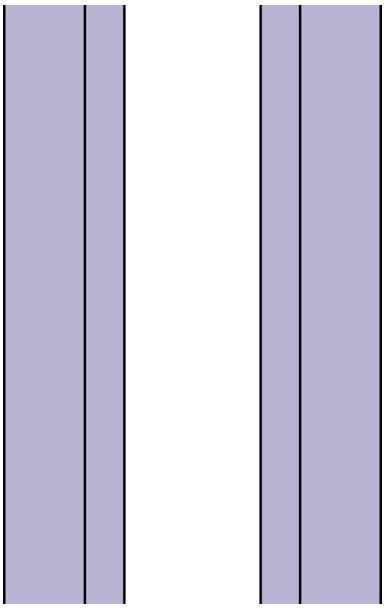


Figura 5.9.21 Paredes ciudad

Figura 5.9.22 Escenario ciudad

Los obstáculos de este escenario son los que pueden verse en las Figuras *5.9.25* y *5.9.26*.

Figura 5.9.25 Obstáculos ciudad derecha

Figura 5.9.26 Obstáculos ciudad izquierda

Por otra parte, las paredes de este escenario contienen ventanas, generadas aleatoriamente mediante código. Sus diseños son los mostrados en las Figuras *5.9.23* y *5.9.24*.

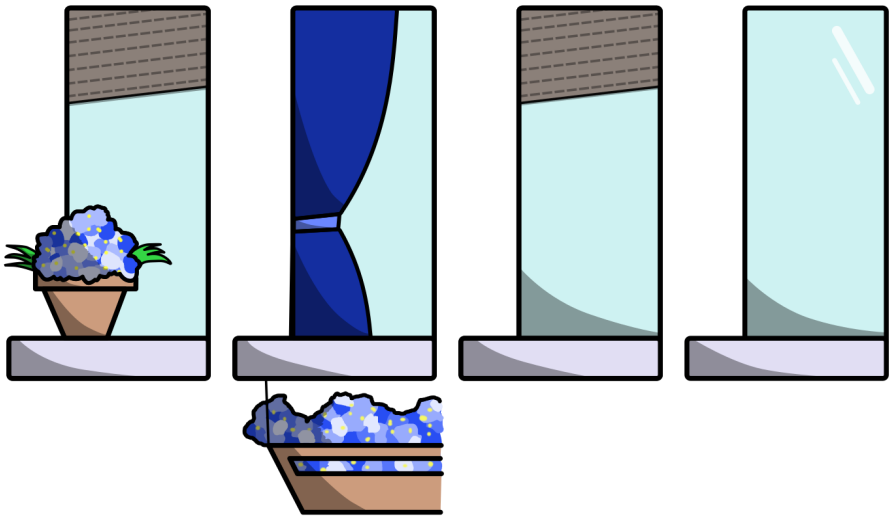


Figura 5.9.23 Ventanas derecha

Figura 5.9.24 Ventanas izquierda

5.9.3 PERSONAJES

En *Up, up! Nyumi!* tenemos cinco personajes distintos que comparten una misma base en el diseño. En este capítulo veremos los bocetos originales y el concept art, además de todos los sprites vectorizados definitivos.

5.9.3.1 ORÍGENES

El diseño básico de Nyumi está basado en un pequeño “garabato” que hizo uno de los integrantes del grupo meses atrás (*Figura 5.9.25 Dibujo original*). Este dibujo nos pareció

ideal para el diseño de nuestro protagonista ya que era simple, amigable, con un toque *cartoon* y era mucho más fácil de animar que un personaje antropomorfo[8].

Figura 5.9.25 Dibujo original

5.9.3.2 NYUMI BÁSICO

En las siguientes imágenes (*Figura 5.9.26 Bocetos Nyumi básicos*) podemos ver los bocetos originales de Nyumi en su forma más básica, en la cual se basan los otros cuatro personajes implementados.

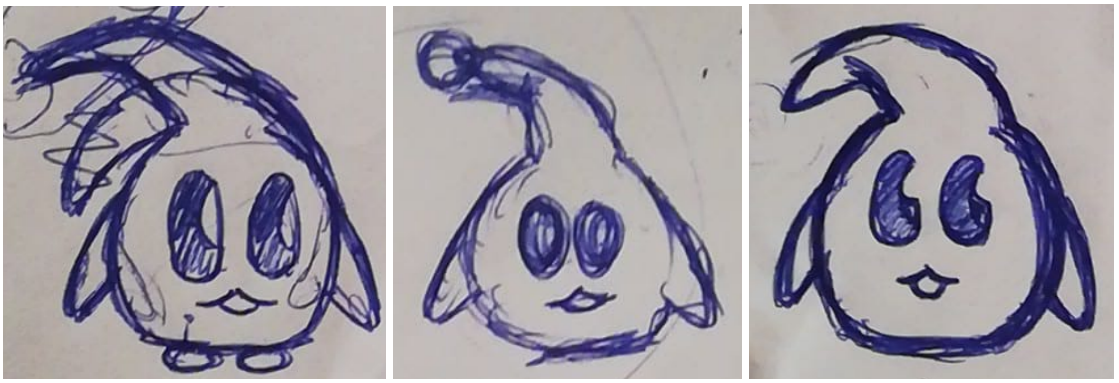


Figura 5.9.26 Bocetos Nyumi básico

La *Figura 5.9.27 Concept art Nyumi default* muestra la digitalización del diseño de este personaje, mientras que a su lado (*Figura 5.9.28 Nyumi default vectorizado*) podemos ver el diseño ya vectorizado.

Figura 5.9.27 Concept art Nyumi default

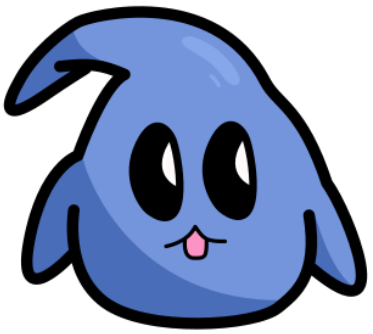


Figura 5.9.28 Nyumi default vectorizado

[8] Punto importante, debido a que todo el arte (diseño y vectorización) iba a ser realizado por una única persona.

Por último, la *Figura 5.9.29 Spritesheet Nyumi default* muestra todos los sprites vectorizados que se usaron para el juego.

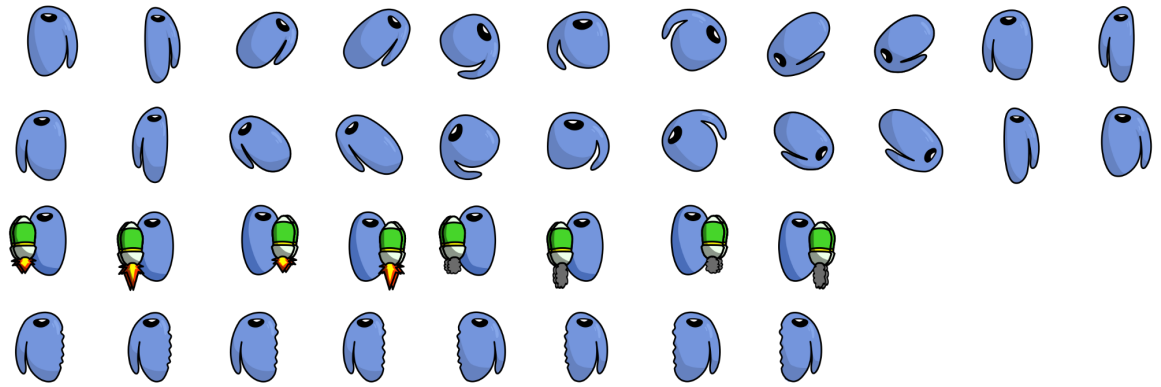


Figura 5.9.29 *Spritesheet* Nyumi default

5.9.3.3 NYUMI BRUJA

En las siguientes tres imágenes (Figuras *5.9.30*, *5.9.31* y *5.9.32*) podemos ver, por orden: el boceto, el concept art y el diseño final vectorizado.

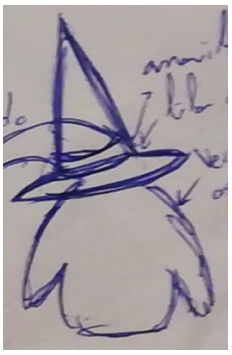


Figura 5.9.30 Boceto bruja

Figura 5.9.31 Concept art bruja

Figura 5.9.32 Bruja vectorizada

El *spritesheet* utilizado para este personaje es el que se muestra en la *Figura 5.9.33 Spritesheet bruja*.

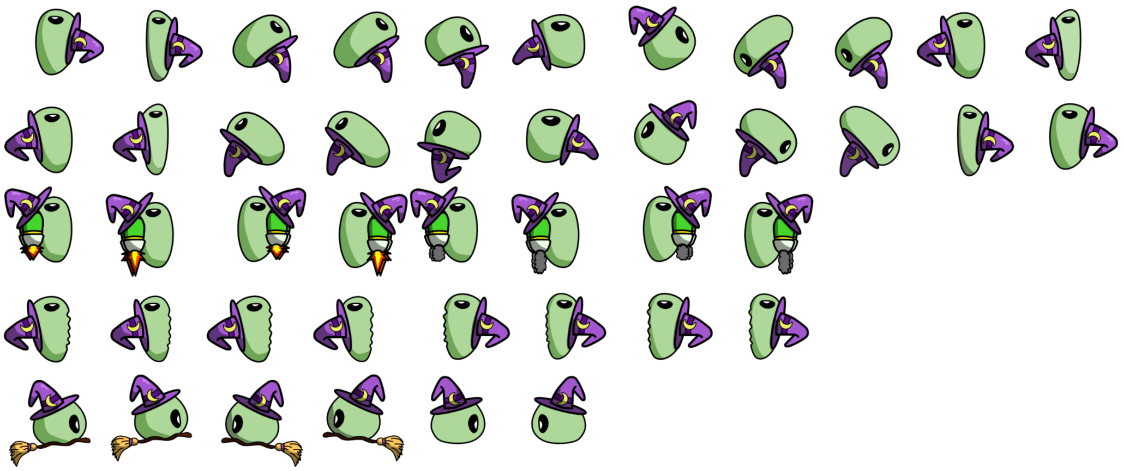


Figura 5.9.33 *Spritesheet* bruja

Como se puede observar en la figura anterior, este personaje tiene sprites exclusivos para animar el vuelo de la bruja, debido a que esta es su habilidad especial.

5.9.3.4 NYUMI CARAMELO

A continuación, vemos el boceto original, el diseño digitalizado y, a su derecha, el diseño vectorizado. Véanse las Figuras *5.9.34, 5.9.35* y *5.9.36*.

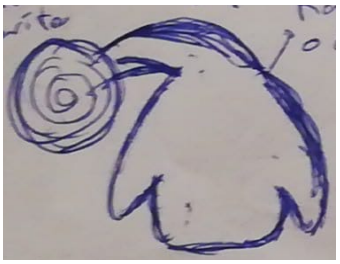


Figura 5.9.34 Boceto caramelo

Figura 5.9.35 Concept art caramelo

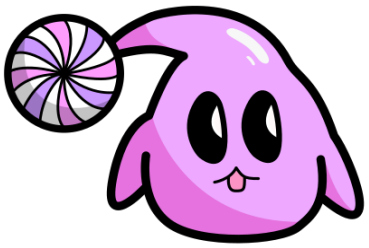


Figura 5.9.36 Caramelo vectorizado

Por otra parte, todos los sprites utilizados en el juego para este personaje son los mostrado en la *Figura 5.9.37 Spritesheet caramelo*.

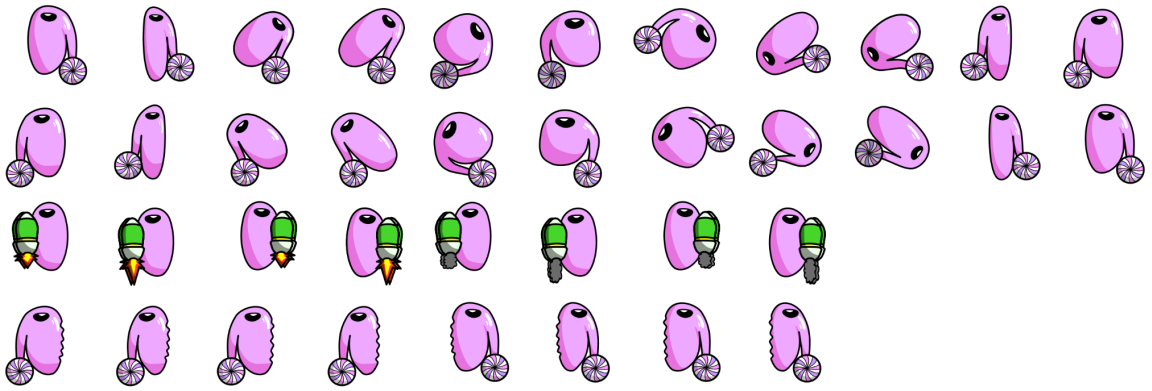


Figura 5.9.37 *Spritesheet* caramelo

5.9.3.5 NYUMI HONGO

A continuación, en las Figuras *5.9.38*, *5.9.39* y *5.9.40*, podemos ver el boceto esquemático, el concept art y el resultado vectorizado.

Figura 5.9.38 Boceto hongo

Figura 5.9.39 Concept art hongo

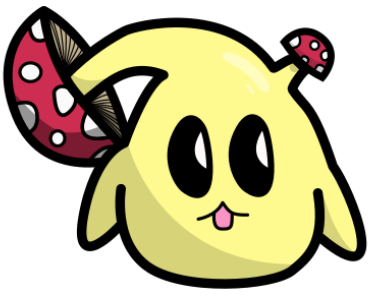


Figura 5.9.40 Hongo vectorizado

El *spritesheet* de este personaje es el representado en la *Figura 5.9.41 Spritesheet Hongo*.

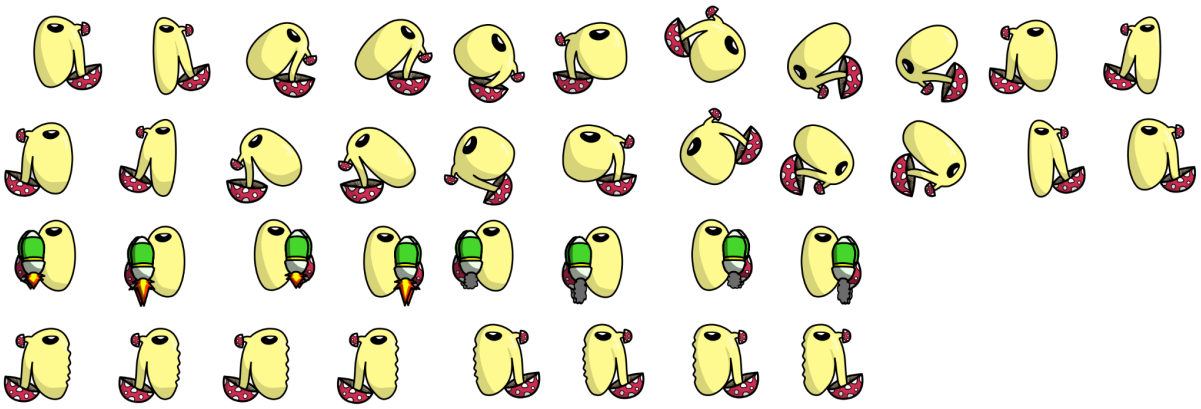


Figura 5.9.41 *Spritesheet* Hongo

5.9.3.6 NYUMI SERPIENTES

Por último pero no menos importante, tenemos el diseño de Nyumi con serpientes. En las figuras siguientes (*5.9.42*, *5.9.43* y *5.9.44)* podemos observar el boceto original, el diseño digitalizado y su vectorización.

Figura 5.9.42 Boceto serpientes

Figura 5.9.43 Concept art serpientes

Figura 5.9.44 Serpientes vectorizadas

El *spritesheet* de este personaje es el que se puede ver en la *Figura 5.9.45*.

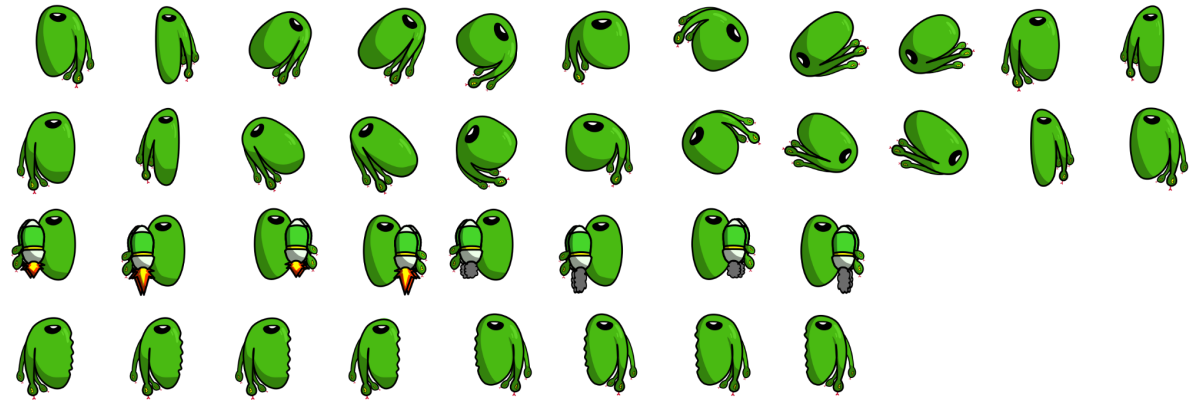


Figura 5.9.45 *Spritesheet* serpientes

5.9.4 POWER-UPS

Durante el juego el usuario puede recoger y usar distintos power-ups de dos tipos: beneficiosos y perjudiciales (solo en modo multijugador).

Los primeros pueden verse en la *Figura 5.9.46 Power-ups beneficiosos*, mientras que los segundos quedan reflejados en la *Figura 5.9.47 Power-ups perjudiciales*.

Figura 5.9.46 Power-ups beneficiosos

Figura 5.9.47 Power-ups perjudiciales

Para hacer más fácil la distinción de los tipos de power-ups durante la partida, decidimos que el color del borde fuera diferente, de manera que el jugador no tenga que observar siempre el dibujo interior.

Feedback

Para indicar que el power-up del escudo está activado, se coloca el *sprite* de la *Figura 5.9.48 Escudo* sobre el personaje. Por otra parte, en caso de colisión mientras el escudo está equipado, se ejecuta una animación de explosión, la cual usa el *spritesheet* de la *Figura 5.9.49 Spritesheet explosión*.

Figura 5.9.48 Escudo

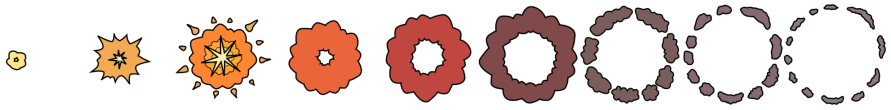


Figura 5.9.49 *Spritesheet* explosión

Para indicar que un jugador tiene su puntuación congelada porque usaron un power-up contra él, se coloca la imagen de la *Figura 5.9.50* (reescalada según convenga) sobre el texto que muestra los puntos.

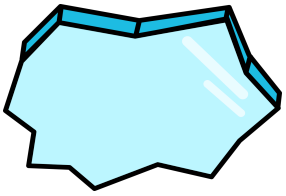


Figura 5.9.50 Hielo

Por último, cuando se activa el power-up para nublar la vista al oponente, se crea una combinación con el *sprite* de la *Figura 5.9.51 Nube*, la cual se anima posteriormente con Godot.

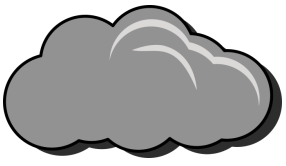


Figura 5.9.51 Nube

5.9.5 OTROS OBJETOS

Además de todo el arte anterior, también nos encontramos con otros elementos como son:

- **Monedas,** las cuales giran al caer. Ver *Figura 5.9.52 Spritesheet monedas*.

Figura 5.9.52 *Spritesheet* monedas

- **Medallas:** una para indicar qué jugador ganó en el modo multijugador (*Figura 5.9.53 Medalla ganador*) y la otra (*Figura 5.9.54 Medalla nuevo récord*) para informar de que se superó una de las tres mejores puntuaciones.

Figura 5.9.53 Medalla ganador

Figura 5.9.54 Medalla nuevo récord

- **Joystick**, para indicarle al jugador que debe mover el joystick a izquierda o derecha para realizar una acción. Ver *Figura 5.9.55 Spritesheet Joystick*

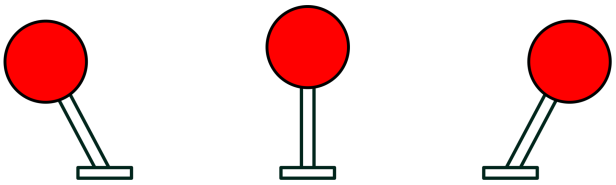


Figura 5.9.55 *Spritesheet* Joystick

5.10 SONIDOS

Para cualquier videojuego un punto crítico es la música y los efectos sonoros de este, ya que ayudan a intensificar el sentimiento de inmersión del jugador y, a su vez, sirven como feedback para el usuario.

En nuestro caso, ninguno de los dos teníamos experiencia en el sector musical, así que optamos por buscar sonidos de libre uso por internet y modificarlos a nuestro gusto.

Las páginas de las cuales descargamos todos los efectos y música fueron: ZapSplat, Sound Jay y Freesound.

Para seleccionar la música buscamos melodías cortas alegres, pegadizas y *cartoon*, que a su vez pudieran llegar a crear un sentimiento de estrés y/o nerviosismo al usuario. Una vez seleccionadas, las modificamos para que se pudieran reproducir en bucle.

6. IMPLEMENTACIÓN

En este capítulo no explicaremos únicamente la implementación de los elementos más destacables, sino que se mostrará también su estructura dentro del proyecto de Godot. Como podrá observarse, no se enseña todo el código, ya que este irá adjunto como anexo debido a que este apartado es para entender el diseño del *gameplay*.

6.1 PANTALLAS DE SELECCIÓN

La pantalla de selección de escenario y la de personaje comparten la misma estructura, cambiando únicamente las descripciones e imágenes que puedan contener. La *Figura 6.1.1* muestra la estructura de estas escenas, mientras que la *Figura 6.1.2* muestra la base de esta pantalla.

Figura 6.1.1 Estructura pantalla selección

Figura 6.1.2 Pantalla selección base

Esta pantalla está formada principalmente por sprites, textos y dos efectos sonoros. Estos elementos se detallarán a continuación.

Sprites

La estructura de este grupo es la mostrada por la *Figura 6.1.3*.

Figura 6.1.3 Estructura *Sprites*

Los elementos son los siguientes :

- ***Fondo***, imagen con el fondo del escenario actual
- ***Paredes***, imagen de las paredes del escenario actual
- ***OpacidadFondo***, imagen blanca con transparencia para reducir la saturación del fondo y las paredes.
- ***Recuadro***, es un spritesheet con las imagénes de la estructura principal de la pantalla. Ver Figura 6.1.4.

Figura 6.1.4 Spritesheet pantalla selección

- ***Imagen*****,** contiene un spritesheet con los elementos disponibles. Ver Figuras *6.1.5* y *6.1.6*.

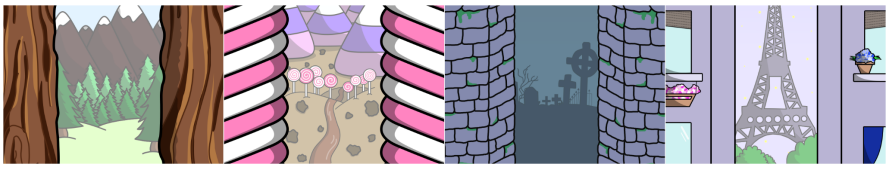


Figura 6.1.5 Spritesheet fondos

Figura 6.1.6 Spritesheet personajes

Description

Está formada únicamente con un elemento de tipo *Label*. Éste contiene el texto con la descripción del elemento actual.

Titulo

Compuesto por dos elementos tipo *Label*. Ambos muestran el mismo texto, pero su formato es diferente, de manera que uno de ellos simula la sombra del otro.

Tiempo

Ver *Figura 6.1.7* para la estructura de este grupo.

Figura 6.1.7 Estructura Tiempo

Este grupo contiene dos elementos:

- ***Texto***, es de tipo Label y muestra el tiempo restante que tiene el jugador para seleccionar un elemento.

- ***TimerTiempo***, es un contador que actualiza el texto del tiempo cada segundo. Cuando quedan cinco segundos cambia el color del texto a rojo y simula un parpadeo. Ver *Figura 6.1.8*.

```
func _on_TimerTiempo_timeout():
    if m_time <= 5:
        $SFX_Reloj.play()
        m_timeText.set("custom_colors/font_color",Color(1,0,0))
        var font = m_timeText.get("custom_fonts/font")
        font.set_size(font.get_size() + m_incTimeSize)
        m_incTimeSize *= -1

    if m_time < 0:
        m_timeText.set_text("0")
    else:
        m_timeText.set_text(String(m_time))
        m_time -= 1

    if (m_time == -1):
        yield(get_tree().create_timer(.6), "timeout")
        playSFX(m_acceptAudio, -6)
        selectObject()
```

Figura 6.1.8 Función *TimerTiempo*

Info

El grupo *Info* hace referencia a la animación de “OK” situada en la parte inferior derecha de la pantalla. Su estructura es la mostrada en la *Figura 6.1.9*.

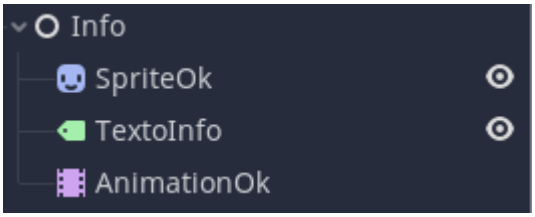


Figura 6.1.9 Estructura *Info*

- ***SpriteOk***, es el spritesheet del botón. Ver *Figura 6.1.10*.

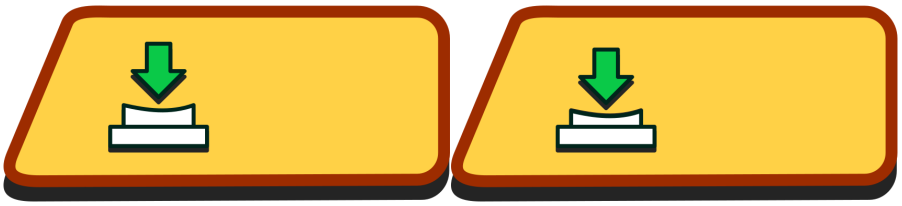


Figura 6.1.10 Spritesheet botón

- ***TextoInfo***, contiene el texto que se visualiza arriba de la imagen.

- ***AnimationOk***, es la animación en bucle de los sprites de la *Figura 6.1.10*.

SFX Y *SFX_Reloj*

Ambos son efectos de sonido. *SFX* se encarga de reproducir un audio al cambiar o confirmar la selección actual. Por otra parte, *SFX_Reloj* reproduce el sonido de un reloj durante los últimos cinco segundos, para indicarle al jugador que queda poco tiempo.

6.2 ESCENARIO

En este apartado veremos los componentes que forman el escenario de ambos modos de juego.

6.2.1 *SINGLEPLAYER*

La estructura de la escena básica para un solo jugador es la mostrada en la *Figura 6.2.1*.

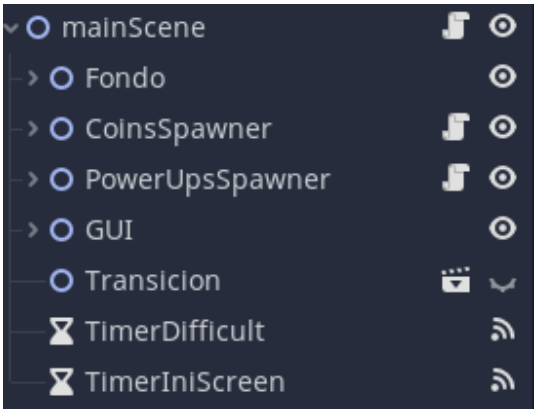


Figura 6.2.1 Estructura escena *singleplayer*

Como puede verse, está formada por: el fondo, la interfaz gráfica (GUI), varios generadores de objetos (*Spawners*) y algunos contadores.

A continuación, ahondaremos en estos elementos.

Fondo

La imágenes siguientes (Figuras *6.2.2* y *6.2.3*) muestran los componentes de este grupo:

Figura 6.2.2 Estructura *Fondo*

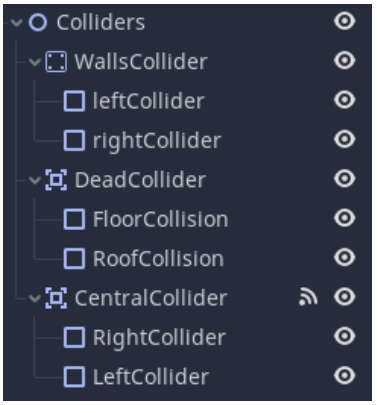


Figura 6.2.3 Componentes *Colliders*

- ***fondo*** es el sprite con el fondo del escenario escogido
- ***paredes1*** y ***paredes2***, son imágenes que contienen las paredes de la escena. Necesitamos dos sprites, ya que estos se mueven de manera circular para simular el movimiento constante del escenario.
- ***BgAuxIzq*** y ***BgAuxDer*** son sprites en negro que se colocan como fondo auxiliar a la izquierda y a la derecha de las paredes.
- ***Colliders*** contiene todas las formas de colisión que pueda haber en el escenario. Entre ellas encontramos:
    - ***WallsCollider***, que contienen los *colliders* de ambas paredes. Su funcionalidad es detener al personaje cuando salta, evitando que éste salga de la pantalla.
    - **DeadCollider**, marcan los límites superior e inferior de la pantalla. Si el personaje alcanza alguno de ellos, morirá de manera inmediata.
    - ***CentralCollider***, está formado por dos colliders que ayudan a ocultar los obstáculos de una pared u otra, según donde se encuentre el personaje.
- ***ObstacleSpawner***, puede resultar extraño que este elemento esté dentro de este grupo. Esto se debe al hecho que necesitamos que los obstáculos se sitúen por encima del fondo pero por detrás de las paredes[9]. Su funcionalidad es generar obstáculos de manera aleatoria y mostrarlos u ocultarlos según sea conveniente.

  El código para generar obstáculos es el que se muestra en la *Figura 6.2.4*. Su función es crear un objeto *Obstaculo* aleatorio (dentro de los candidatos a elegir) y le proporciona una posición, aleatoria en el eje vertical. Por último, lo oculta si se encuentra en la pared opuesta a la del personaje.

[9] A excepción de los escenarios del castillo y la ciudad, en los cuales este elemento se mueve para que quede también por encima de las paredes.

```gdscript
func spawn_obstacle(var esq, var posY = -1):
    # si esq = true, es col·loca l'obstacle a l'esquerra, altrament va a la dreta

    var obstacle = m_obstacle.instance()
    call_deferred("add_child", obstacle)

    var numObs = GlobalVariables.m_rand.randi_range(0, 2)
    obstacle.create(!esq, numObs)

    if esq:
        obstacle.position.x = 350
        if GlobalVariables.nameScene == "Default":
            obstacle.position.x -= 6
    else:
        obstacle.position.x = 586
        if GlobalVariables.nameScene == "Bosque":
            obstacle.position.x -= 12
        elif GlobalVariables.nameScene == "Castillo":
            obstacle.position.x -= 4

    if posY == -1:
        obstacle.position.y = -(GlobalVariables.m_rand.randi() % 40 + 80)
    else:
        obstacle.position.y = posY

    obstacle.setVisibility(GlobalVariables.m_dretaPl == (!esq)
```

Figura 6.2.4 Código generador de obstáculos

CoinsSpawner y PowerUpsSpawner

Ambos contienen un elemento del tipo *Timer*. *CoinsSpawner* se encarga de generar una moneda transcurridos unos segundos, mientras que *PowerUpsSpawner* instancia power-ups. Este lapso de tiempo se calcula de manera aleatoria, seleccionando un valor dentro de un rango preestablecido.

Por otra parte, *PowerUpsSpawner* se encarga de controlar si un power-up fue capturado por el jugador.

GUI

Contiene los elementos que pertenecen a la interfaz gráfica del usuario. Ver *Figura 6.2.5*.

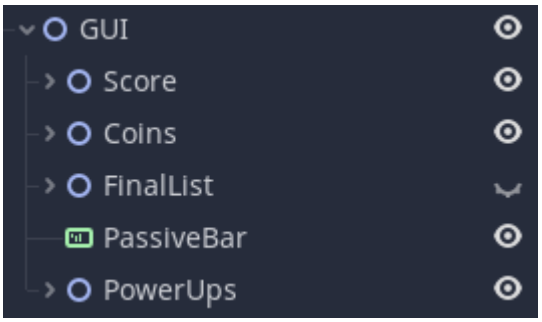


Figura 6.2.5 Estructura *GUI*

- ***Score***

  Contiene dos elementos: uno de tipo Label, para mostrar el texto de la puntuación del jugador, y el otro de tipo Timer, el cual se encarga de incrementar y actualizar la puntuación cada media décima de segundo.

- ***Coins***

  Contiene tres tipos de elementos distintos:

  - *Label*. Para mostrar el texto con la cantidad de monedas que posee el jugador.
  - *Sprites*. Tiene dos: uno que se encuentra fijo para visualizar la moneda al lado de su texto, y otro que se utiliza para crear una animación para el final de la partida.
  - *AnimationPlayer*. Al final de la partida, cada moneda que tenga el jugador se le cambia por una cantidad de puntos. Para ello, mostramos una animación donde la moneda se traslada girando desde su posición hasta donde se encuentra el texto de la puntuación. Ver *Figura 6.2.6*.

```
func _on_Player_finishGame():
    ...

    for _i in range(GlobalVariables.m_coinsPl):
        MusicController.playCoin()
        $GUI/Coins/CoinsAnimation.play("puntosExtra")
        GlobalVariables.m_coinsPl -= 1
        m_textCoins.set_text(String(GlobalVariables.m_coinsPl))
        yield($GUI/Coins/CoinsAnimation, "animation_finished")
        GlobalVariables.scorePl += 6
        $GUI/Score/LabelScore.set_text(String(GlobalVariables.scorePl))

    ...
```

Figura 6.2.6 Animación final monedas

- ***FinalList*** (Ver *Figura 6.2.7* para la estructura)

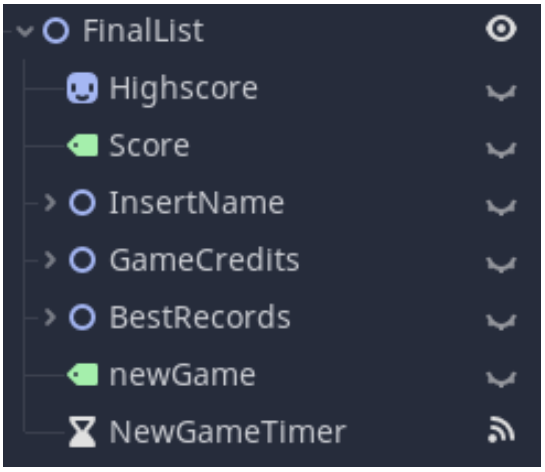


Figura 6.2.7 Estructura *FinalList*

- ***Highscore***

  Es la imagen de la medalla que indica que se superó una de las tres mejores puntuaciones.

- ***Score***

  Texto que muestra la puntuación final del jugador (después de haberse sumado todos los puntos por las monedas obtenidas).

- ***InsertName***

  Este elemento sólo se visualiza cuando el jugador superó un récord y debe registrar su nombre. Contiene tres elementos: un texto con las instrucciones a seguir, una imagen con el spritesheet de un joystick y una animación para hacer que el joystick se mueva de izquierda a derecha.

- ***GameCredits***

  Contiene tres etiquetas para informar de los créditos de los que dispone el jugador, el número de jugadores escogidos y, en caso de no disponer de ningún crédito, se visualiza el famoso texto “Insertar moneda”.

- ***BestRecords***

  Contiene diversas etiquetas para mostrar la tabla final con las tres mejores puntuaciones.

- ***newGame***

  Es el texto que se muestra en caso de tener créditos para jugar.

- ***NewGameTimer***

  Es un contador que se encarga de hacer que los textos informativos como “Insertar moneda” o “Enter para jugar” simulen un parpadeo, haciéndolos disminuir o aumentar su tamaño según la iteración.

- ***PassiveBar***

  Es un elemento del tipo *TextureProgress*, el cual se encarga de mostrar el estado de la barra de carga de la habilidad del personaje y, en caso de llenarse, activa la habilidad automáticamente.

- ***PowerUps***

  Contiene tres *sprites*, uno por cada power-up mostrado en pantalla. Estas imágenes se cambian mediante código durante el transcurso de la partida, haciendo que cuando un power-up es capturado, se muestre su imagen en la primera posición libre y, en caso de ser usado, mostrar la imagen original (power-up deshabilitado).

Transición

Contiene un *Sprite* y un *AnimationPlayer*. Su función es simular una animación de transición, haciendo que al inicio haya un círculo negro lo suficientemente grande para tapar toda la pantalla y, a medida que pasa el tiempo, el círculo va disminuyendo hasta desaparecer y dar lugar a la escena principal del juego.

TimerDifficult

Es el contador que cada pocas décimas de segundo aumenta la velocidad a la que se mueven las paredes del escenario y disminuye la distancia entre las apariciones de los obstáculos.

TimerIniScreen

Es un contador que se encarga de reproducir el vídeo inicial después de varios segundos de inactividad.

6.2.2 *MULTIPLAYER*

La estructura de la escena básica multijugador es la mostrada en la *Figura 6.2.8.*

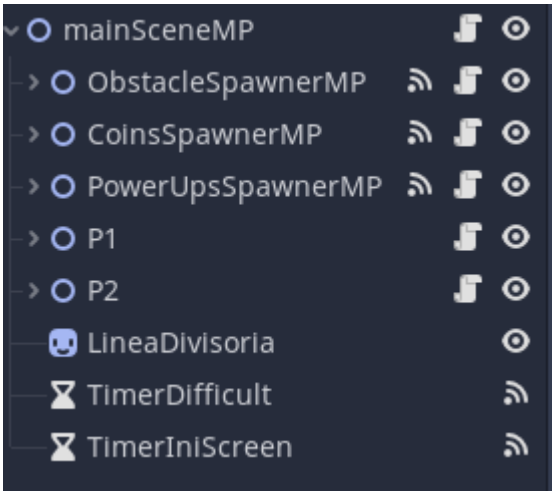


Figura 6.2.8 Estructura escena multi*player*

Esta escena está formada principalmente por generadores de objetos, contadores y dos instancias de escenas para un solo jugador (*P1* y *P2*).

El funcionamiento de este escenario es prácticamente idéntico al de un solo jugador. Por este motivo, no comentaremos todos sus elementos, sino que explicaremos aquellos que fueron añadidos o modificados.

Spawners

La diferencia principal con el escenario *singleplayer*, es que en este necesitamos generadores globales de objetos (obstáculos, monedas y power-ups). De esta manera, podemos colocar todos los objetos en las mismas posiciones de ambas pantallas y hacerlos aparecer con la misma frecuencia, equilibrando el juego para ambos usuarios. No obstante, cada escena tiene su propio generador de objetos; esto se debe al hecho que los dos jugadores no interactúan igual con el videojuego y van a realizar acciones de manera no simultánea que afectarán sólo a los objetos de una de las pantallas.

Lo que hacen los elementos *ObstacleSpawnerMP*, *CoinsSpawnerMP* y *PowerUpsSpawnerMP* es enviar una señal a las escenas *P1* y *P2* para que realicen una instancia del objeto correspondiente y lo integren en sendos escenarios. En el caso particular de *ObstacleSpawnerMP*, éste les pasa por parámetro las características del obstáculo a crear. Ver *Figura 6.2.9*.

```
func spawn_obstacle(var esq, var pos = -1):
    # si esq = true, es col·loca l'obstacle a l'esquerra, altrament a la dreta

    var numObs = GlobalVariables.m_rand.randi_range(0, 2)
    var posY = pos
    if pos == -1:
        posY = -(GlobalVariables.m_rand.randi() % 40 + 80) - 300

    emit_signal("spawn", !esq, numObs, posY)
```

Figura 6.2.9 Generador obstáculos *multiplayer*

Timers

Por otra parte, los contadores encargados de aumentar la dificultad del juego (*TimerDifficult*) y de volver a la pantalla inicial tras un tiempo de inactividad (*TimerIniScreen*), también pasan a ser globales, de manera que puedan controlar simultáneamente ambas pantallas de juego.

Escenarios *P1* y *P2*

Como comentamos anteriormente, ambas son copias del escenario para un solo jugador, modificadas a posteriori. Su estructura es la mostrada en la *Figura 6.2.10*.

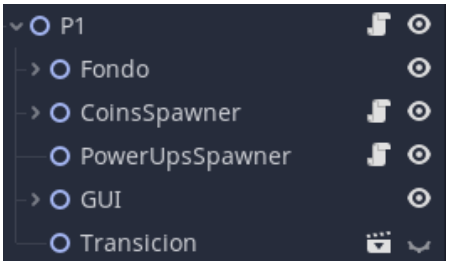


Figura 6.2.10 Estructura elementos *P1* y *P2*

Ninguno de los elementos que los componen cambian su funcionalidad (a excepción de que los generadores locales reciben una señal para instanciar los objetos correspondientes).

Los cambios más relevantes son a nivel de *Scripts*, ya que se debe comprobar continuamente sobre qué escenario estamos actuando y, se añaden funciones para activar los power-ups perjudiciales, las cuales se detallarán en el *Apartado 6.3.1 Personaje*.

Script mainMP.gd

Este *script* es exclusivo del modo multijugador. Se encarga de evitar los posibles problemas de concurrencia a la hora de comprobar si se superó alguno de los récords.

A nivel de código, espera a que ambos jugadores hayan finalizado la partida, en cuyo caso proclamará al vencedor (o vencedores en caso de empate) y, posteriormente, comprobará si batieron algún récord. En caso de que se tenga que registrar algún nombre, esperará hasta que se haya finalizado esta tarea e iniciará la cuenta atrás para volver a la pantalla inicial (la cual se resetea tras cada interacción con el juego). Ver Figura *6.2.11.*

```
func _process(_delta):
    if !m_comproved && GlobalVariables.m_isFinishedP1 && GlobalVariables.m_isFinishedP2:
        if (GlobalVariables.scoreP1 >= GlobalVariables.scoreP2):
            $P1/GUI/FinalList/Medalla.visible = true
        if GlobalVariables.scoreP2 >= GlobalVariables.scoreP1:
            $P2/GUI/FinalList/Medalla.visible = true

        searchWinners(GlobalVariables.scoreP1, GlobalVariables.scoreP2)

    if !GlobalVariables.m_finished && GlobalVariables.m_nameSelectedP1 && GlobalVariables.m_nameSelectedP2:
        GlobalVariables.m_finished = true
        $TimerIniScreen.start()
    elif GlobalVariables.m_finished && Input.is_action_just_pressed("anyButton"):
        $TimerIniScreen.start()
```

Figura 6.2.11 mainMP.gd

Para controlar si se superó algún récord, lo que hace es comprobar la puntuación del primer jugador, si es mayor que algunas de las tres máximas puntuaciones, actualiza el registro. A continuación, repite la acción con la puntuación del segundo jugador y la tabla ya actualizada. Por último, comprueba si el primer jugador sigue estando dentro de las tres mejores puntuaciones y se informa a ambos jugadores de su resultado. Ver *Figura 6.2.12*.

```gdscript
func searchWinners(score1, score2):
    m_comproved = true
    var m_posRecordP1 = 0
    for s in GlobalVariables.m_listScores:
        if score1 > s:
            break
        m_posRecordP1 += 1

    if m_posRecordP1 < 3: # nou récord
        insertarRecord(m_posRecordP1, score1)

    var m_posRecordP2 = 0
    for s in GlobalVariables.m_listScores:
        if score2 > s:
            break
        m_posRecordP2 += 1

    if m_posRecordP2 < 3: # nou récord
        $P2.newRecord(m_posRecordP2)
        insertarRecord(m_posRecordP2, score2)
    else:
        GlobalVariables.m_nameSelectedP2 = true

    if m_posRecordP2 == m_posRecordP1:
        m_posRecordP1 += 1

    yield(get_tree().create_timer(0.8), "timeout")
    if m_posRecordP1 < 3:
        $P1.newRecord(m_posRecordP1)
    else:
        GlobalVariables.m_nameSelectedP1 = true
```

Figura 6.2.12 Comprobar nuevos récords

6.3 ELEMENTOS PARTIDA

Una vez vistos los escenarios, explicaremos los elementos que forman parte de él.

6.3.1 PERSONAJE

Para implementarlo, decidimos hacer una escena base de la cual pudiera heredar cada uno de los personajes, de manera que cada uno tuviera *sprites* y *colliders* personalizados, a la vez de tener la capacidad de activar su habilidad. Ver *Figura 6.3.1*.

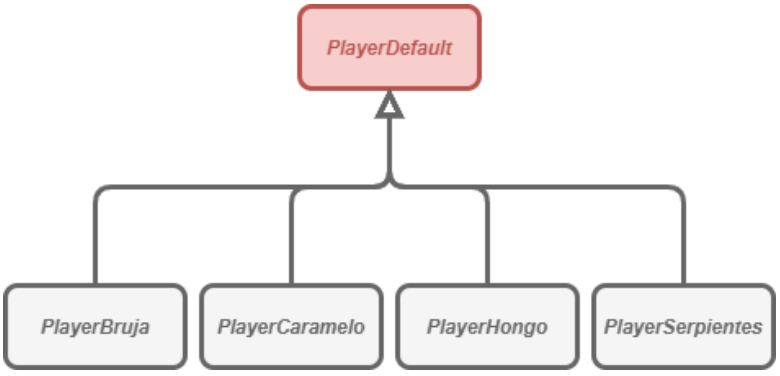


Figura 6.3.1 Diagrama herencia personajes

Los componentes de esta escena son los mostrados en la *Figura 6.3.2*.

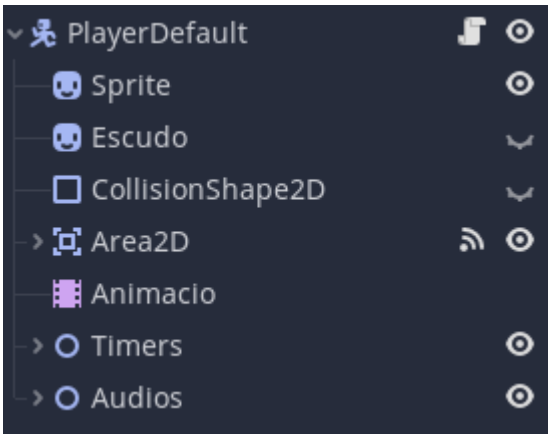


Figura 6.3.2 Estructura personaje

- ***Sprite***, contiene el spritesheet del personaje.
- ***Escudo***, contiene la imagen para indicar que el power-up del escudo está activado.
- ***CollisionShape2D***, es el collider que usamos para comprobar si el jugador chocó contra alguna de las paredes.
- ***Area2D***, contiene otro collider, mucho más preciso, para comprobar las colisiones con los obstáculos, power-ups y monedas.
- ***Animacio***, guarda todas las animaciones del personaje. Ver *Figura 6.3.3*.

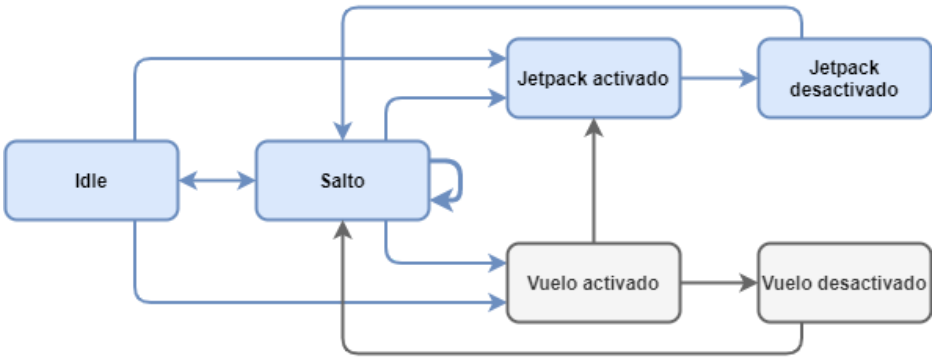


Figura 6.3.3 Diagrama animaciones personaje

Nota: para simplificar el esquema anterior no se especifica si el personaje está situado a la izquierda o a la derecha; por otra parte, los recuadros en gris están disponibles únicamente para el personaje Nyumi Bruja.

- ***Timers,*** el personaje tiene varios temporizadores:
    - Uno por cada power-up, para controlar cuándo finalizarlos. El hecho de tener cada uno su propio temporizador es por la capacidad que tiene el jugador de mantener activados varios power-ups de manera simultánea.

- Dos para controlar la habilidad del personaje, ya que uno se encarga del *cooldown* (cuenta atrás para activarla automáticamente), mientras que el otro se encarga de desactivarla después de unos segundos.
- Uno para cambiar los valores de algunos parámetros del personaje y así aumentar la dificultad. Véase la *Figura 6.3.4*.

```
func aumentarVelocidad():
    m_acceleracio = min(m_acceleracio + 0.0005, m_maxAcceleration)
    jumpSpeed = min(jumpSpeed + 0.21, m_maxJumpSpeed)
    jumpHeight = min(jumpHeight + 0.14, m_maxJumpHeight)
```

Figura 6.3.4 Código aumento dificultad personaje

- ***Audios,*** el personaje tiene tres pistas de audio diferentes: una para el sonido que hace al saltar, otra para indicar el inicio de su habilidad y, la última, para cuando se activa un power-up.

Las acciones principales que puede hacer el personaje son las siguientes:

Volar

Cuando la habilidad de la bruja está activada o cuando se está haciendo uso del jetpack, el código que gestiona el vuelo es el visualizado en la *Figura 6.3.5*.

```
func flyP1(delta):
    if Input.is_action_pressed("ui_right"): # movem cap a la dreta
        position.x = position.x + 8.0
        if GlobalVariables.m_usingJetpackP1:
            $Animacio.play("JetpackDerAnimation")
        else:
            $Animacio.play("VolarDer")
    elif Input.is_action_pressed("ui_left"): # movem cap a l'esquerra
        position.x = position.x - 8.0
        if GlobalVariables.m_usingJetpackP1:
            $Animacio.play("JetpackIzqAnimation")
        else:
            $Animacio.play("VolarIzq")
    elif Input.is_action_pressed("ui_down"): # abaix accelerem la caiguda
        position.y = position.y + 120 * delta * m_acceleracio
    elif Input.is_action_pressed("ui_up"): # accelerem la pujada
        position.y = position.y - 550 * delta * m_acceleracio

    if position.x <= 950.0:
        GlobalVariables.m_dretaP1 = false
    else:
        GlobalVariables.m_dretaP1 = true

    motion = move_and_slide(motion, Vector2.DOWN) # simulem el moviment
```

Figura 6.3.5 Código vuelo

Moverse

La *Figura 6.3.6* muestra el código del movimiento del personaje.

```
func moveP1(delta):
    if !$Animacio.is_playing() && m_idle:
        if GlobalVariables.m_dretaP1:
            $Animacio.play("idleDer")
        else:
            $Animacio.play("idleIzq")

    if (m_canJump && (Input.is_action_just_pressed("ui_right") || Input.is_action_just_pressed("ui_left"))) ||
        (m_jumps < GlobalVariables.m_maxJumpsP1 && ((!GlobalVariables.m_dretaP1 &&
        Input.is_action_just_pressed("ui_right")) || (GlobalVariables.m_dretaP1 &&
        Input.is_action_just_pressed("ui_left")))): # si vol i pot saltar

        m_jumps += 1 # comptabilitzem el nombre de salts fets

        $Audios/Salto.stream = m_audioSaltosList[GlobalVariables.m_rand.randi_range(0,
                               m_audioSaltosList.size() - 1)]
        $Audios/Salto.play()

        if Input.is_action_pressed("ui_right"): # movem cap a la dreta
            $Animacio.play("saltoDer")
            motion.x = jumpSpeed
            GlobalVariables.m_dretaP1 = true
        elif Input.is_action_pressed("ui_left"): # movem cap a l'esquerra
            $Animacio.play("saltoIzq")
            motion.x = -jumpSpeed
            GlobalVariables.m_dretaP1 = false
        motion.y = -jumpHeight # el personatge puja
    elif m_jumps == 0: # mentres està a la paret
        if Input.is_action_just_pressed("ui_down"): # el primer cop que s'apreta la tecla cap abaix
            m_lastMotion = motion
        elif Input.is_action_pressed("ui_down"): # accelerem la caiguda
            motion.y = min(motion.y * m_acceleracio * 1.2, m_maxSpeed)
        elif Input.is_action_just_released("ui_down"): # està a la paret i ha deixat d'accelerar la caiguda
            motion.y = m_lastMotion.y
    elif m_jumps != 0 && is_on_wall(): # ha tocat una paret
        motion.y = min((m_gravity * delta - motion.y) * m_acceleracio, m_maxSpeed)
        m_jumps = 0
        m_idle = true
        if GlobalVariables.m_dretaP1:
            $Animacio.play("stopDer")
        else:
            $Animacio.play("stopIzq")

    if motion.x == 0:
        motion.y = abs(motion.y)

    motion = move_and_slide(motion, Vector2.DOWN) # simulem el moviment
```

Figura 6.3.6 Código movimiento

Activar power-ups

Por otra parte, el usuario puede hacer uso de los power-ups que haya recogido. Nótese que ninguno de ellos es acumulativo, por lo que usar consecutivamente el mismo power-up sólo hace que se resetee su temporizador.

- **Escudo.** La *Figura 6.3.7* muestra el código de la activación y desactivación del escudo. Su función es guardar una variable indicando que el escudo está activado (para evitar que el jugador muera al colisionar por primera vez) y mostrar u ocultar el *sprite* del escudo.

```
func usarEscudo():
    m_teEscut = true
    $Escudo.visible = true
    $Timers/ShieldTimer.start()

func _on_ShieldTimer_timeout():
    m_teEscut = false
    $Escudo.visible = false
```

Figura 6.3.7 Código power-up escudo

- **Reloj ralentizador.** Hace que las variables de movimiento del jugador vuelvan a tener los valores iniciales. Una vez finalizado, vuelve a tener los valores que tenía el personaje la primera vez que se usó este objeto. Ver *Figura 6.3.8*.

```
func usarReloj():
    var firstTime = false

    if $Timers/RelojTimer.is_stopped() && $Timers/RayoTimer.is_stopped():
        m_lastAcceleration = m_acceleracio
        m_lastJumpHeight = jumpHeight
        m_lastJumpSpeed = jumpSpeed

        firstTime = true

    $Timers/RelojTimer.start()

    jumpHeight = 240.0
    jumpSpeed = 1100.0
    m_acceleracio = 1.0

    emit_signal("reduceVelocity", true, firstTime)

func _on_RelojTimer_timeout():
    m_acceleracio = m_lastAcceleration
    jumpHeight = m_lastJumpHeight
    jumpSpeed = m_lastJumpSpeed

    emit_signal("reduceVelocity", false, true)
```

Figura 6.3.8 Código power-up reloj ralentizador

- **Jetpack.** Su uso implica la capacidad de vuelo del personaje, la cual no puede desactivarse hasta que el power-up se acabe. En caso de controlar al personaje de Nyumi Bruja, si se activa el jetpack se pausa la carga de la habilidad, de manera que ésta no pueda detener el power-up. Ver Figuras *6.3.9* y *6.3.10*.

```
func usarJetpack():
    if GlobalVariables.characterSelected == "Bruja":
        if m_passiveCurrentTime > 0:
            stopPassive()
        $Timers/ActivePassiveTimer.stop()
        $Timers/CooldownTimer.stop()

    if m_numPlayer == 1:
        GlobalVariables.m_usingJetpackP1 = true
    else:
        GlobalVariables.m_usingJetpackP2 = true

    $Timers/JetpackTimer.start()
    activarVuelo()

func _on_JetpackTimer_timeout():
    $Timers/CooldownTimer.start()
    desactivarVuelo()
    GlobalVariables.m_usingJetpackP1 = false
```

Figura 6.3.9 Código power-up jetpack

```
func activarVuelo():
    m_isFlying = true
    m_lastMotion.y = motion.y
    motion.x = 0.0
    motion.y = m_maxSpeed / 5
    m_canJump = false
    if m_numPlayer == 1:
        if GlobalVariables.m_usingJetpackP1:
            if !GlobalVariables.m_dretaP1:
                $Animacio.play("JetpackIzqAnimation")
            else:
                $Animacio.play("JetpackDerAnimation")
    else:
        if GlobalVariables.m_usingJetpackP2:
            if !GlobalVariables.m_dretaP2:
                $Animacio.play("JetpackIzqAnimation")
            else:
                $Animacio.play("JetpackDerAnimation")

func desactivarVuelo():
    m_isFlying = false
    m_jumps = 0
    motion.y = m_lastMotion.y
    m_canJump = true

    if (m_numPlayer == 1 && GlobalVariables.m_usingJetpackP1) ||
        (m_numPlayer == 2 && GlobalVariables.m_usingJetpackP2):
        if $Animacio.current_animation == "JetpackIzqAnimation":
            $Animacio.play("JetpackIzqAnimation2")
        elif $Animacio.current_animation == "JetpackDerAnimation":
            $Animacio.play("JetpackDerAnimation2")
```

Figura 6.3.10 Código activación y desactivación vuelo

- **Hielo.** Cuando se usa este power-up, se detiene el contador que incrementa la puntuación del contrincante durante unos segundos, a la vez que se activa la visualización de una imagen de hielo encima del texto que muestra los puntos.

- **Rayo acelerador.** Este power-up es el opuesto al reloj ralentizador. Su función es hacer que las variables de movimiento del personaje contrario adopten los valores máximos que se pueden tener durante una partida. En las Figuras *6.3.11* y *6.3.12* podemos ver el código del usuario afectado.

```
func usarRayo():
    var firstTime = false

    if $Timers/RayoTimer.is_stopped() && $Timers/RelojTimer.is_stopped():
        m_lastAcceleration = m_acceleracio
        m_lastJumpHeight = jumpHeight
        m_lastJumpSpeed = jumpSpeed

        firstTime = true

    $Timers/RayoTimer.start()

    jumpHeight = m_maxJumpHeight
    jumpSpeed = m_maxJumpSpeed
    m_acceleracio = m_maxAcceleration

    motion.y = min(motion.y * m_acceleracio, m_maxSpeed)

    emit_signal("incVelocity", true, firstTime)
```

Figura 6.3.11 Código activación power-up rayo acelerador

```
func _on_RayoTimer_timeout():
    m_acceleracio = m_lastAcceleration
    jumpHeight = m_lastJumpHeight
    jumpSpeed = m_lastJumpSpeed

    emit_signal("incVelocity", false, true)
```

Figura 6.3.12 Código desactivación power-up rayo acelerador

- **Nubes.** Este power-up dificulta la visión del oponente. Para ello, se activa la animación de unas nubes que tapan parte de su pantalla y desaparecen a los pocos segundos. De la activación y desactivación de este power-up no se encarga el propio personaje, sino que lo hace el escenario. Ver *Figura 6.3.13*.

```
func _ready():
    ...
    m_nubes = load("res://Scenes/Nubes.tscn").instance()
    call_deferred("add_child", m_nubes)
    ...

func useClouds():
    m_nubes.get_node("AnimationClouds").play("nubes")
```

Figura 6.3.13 Código power-up nubes

6.3.2 OBJETOS

Además del personaje, tenemos tres objetos principales: obstáculos, monedas y power-ups.

Obstáculos

Estos tienen como base un elemento *Area2D* y están compuestos por:

- Dos *Sprites*: uno con la imagen del obstáculo y otro con el spritesheet de una explosión.
- Diversos *colliders*: personalizados según la forma del obstáculo. Por defecto se encuentran todos inhabilitados y se habilita el que corresponde en el momento de su creación.
- Un temporizador (*Timer*), para hacer que la velocidad de su caída acelere con el transcurso de la partida.
- Un *AnimationPlayer* con tres animaciones: una para simular la aparición por la izquierda, otra para simularla por la derecha y, la última, de una explosión, la cual se activa en caso de que el jugador choque con un obstáculo mientras tiene el escudo activado. En la *Figura 6.3.14* se muestra el código para mostrar y ocultar los obstáculos con sus respectivas animaciones, mientras que la *Figura 6.3.15* muestra el código que se llama cuando el jugador rompe un obstáculo.

```
func setVisibility(showObs):
    if showObs != visible:
        if showObs:
            if m_dreta:
                $Animacio.play_backwards("desaparicionDer")
            else:
                $Animacio.play_backwards("desaparicionIzq")
        else:
            if m_dreta:
                $Animacio.play("desaparicionDer")
            else:
                $Animacio.play("desaparicionIzq")
            yield($Animacio, "animation_finished")

        visible = showObs
```

Figura 6.3.14 Código visibilidad obstáculo

```
func explode():
    collider.disabled = true
    $Animacio.play("Explosion")
    yield($Animacio, "animation_finished")
    queue_free()
```

Figura 6.3.15 Código explosión obstáculo

Monedas

La estructura de una escena de una moneda es la mostrada en la *Figura 6.3.16*.

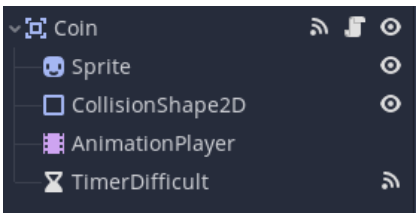


Figura 6.3.16 Estructura moneda

La utilidad de cada elemento es la siguiente:

- ***Sprite***. Contiene un spritesheet de giro de una moneda.
- ***CollisionShape2D***. Comprueba si colisionó con uno de los personajes, en cuyo caso se le sumará una moneda al jugador correspondiente (a excepción de que estén jugando con Nyumi Bruja y usando el jetpack simultáneamente) y se eliminará la instancia de la moneda. Ver *Figura 6.3.17*.

```
func _on_Coin_body_entered(body):
    if body.name.find("Player") != -1:
        MusicController.playCoin()
        if body.getNumPlayer() == 1:
            GlobalVariables.m_coinsP1 += 1
            if !(GlobalVariables.characterSelected == "Bruja" && GlobalVariables.m_usingJetpackP1):
                GlobalVariables.m_passiveProgressP1 += 0.6
        else:
            GlobalVariables.m_coinsP2 += 1
            if !(GlobalVariables.characterSelected == "Bruja" && GlobalVariables.m_usingJetpackP2):
                GlobalVariables.m_passiveProgressP2 += 0.6
        queue_free()
```

Figura 6.3.17 Código colisión moneda

- ***AnimationPlayer***. Contiene la animación de una moneda girando en bucle.
- ***TimerDifficult***. Es un temporizador para acelerar la caída de la moneda con el transcurso del tiempo.

Power-ups

Los power-ups siguen prácticamente la misma estructura que las monedas, a excepción de no tener un elemento *AnimationPlayer*. En cuanto a funcionalidad, la diferencia entre estos dos objetos es que, cuando un personaje colisiona con un power-up, se comprueba si tiene un hueco libre en el inventario, en cuyo caso se añadirá el objeto en la posición correspondiente. Ver *Figura 6.3.18*.

```
func _on_PowerUp_body_entered(body):
    if body.name.find("Player") != -1:
        if body.getNumPlayer() == 1:
            var pos = GlobalVariables.m_powerUpsP1.find("-1")
            if pos != -1:
                MusicController.playPowerUpAchieved()
                emit_signal("captured", m_name, pos)
                GlobalVariables.m_powerUpsP1[pos] = m_name
                queue_free()
        else:
            var pos = GlobalVariables.m_powerUpsP2.find("-1")
            if pos != -1:
                MusicController.playPowerUpAchieved()
                emit_signal("captured", m_name, pos)
                GlobalVariables.m_powerUpsP2[pos] = m_name
                queue_free()
```

Figura 6.3.18 Código colisión power-up

7. RESULTADOS

Realizamos todos los apartados planificados inicialmente. A continuación, se pueden ver algunas capturas del videojuego llevado a cabo.

7.1 PANTALLA INICIAL

La *Figura 7.1.1* muestra una parte del vídeo inicial que se reproduce cuando la máquina está inactiva.

Figura 7.1.1 Frame del vídeo inicial

En las Figuras *7.1.2* y *7.1.3* podemos ver dos estados diferentes de la pantalla inicial.

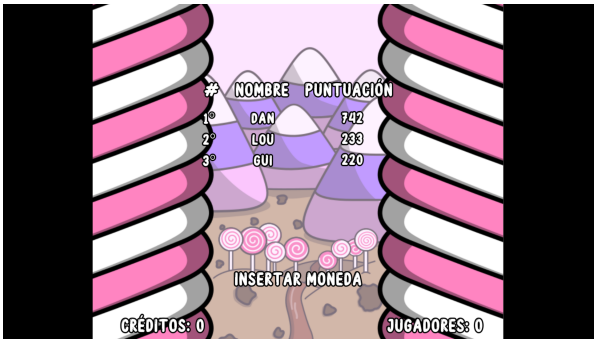


Figura 7.1.2 Pantalla inicial sin créditos

Figura 7.1.3 Pantalla inicial para multijugador

7.2 SELECCIÓN DE ESCENARIO Y PERSONAJE

En la *Figura 7.2.1* se muestra la pantalla de selección de escenario en sus cuatro estados.

Figura 7.2.1 Pantalla selección escenario

Una vez seleccionado el escenario, se debe escoger uno de los cinco personajes disponibles. Ver *Figura 7.2.2*. Como puede observarse, el fondo de esta pantalla muestra el escenario escogido anteriormente.

Figura 7.2.2 Pantalla selección de personaje

7.3 *IN-GAME*

En esta sección mostraremos capturas de diversas partidas en ambos modos de juego.

7.3.1 *SINGLEPLAYER*

Las imágenes siguientes son de partidas de un solo jugador.

En la *Figura 7.3.1* puede verse a Nyumi Hongo usando el power-up del jetpack en el escenario del bosque.

Figura 7.3.1 *In-game power-up jetpack*

En la *Figura 7.3.2* tenemos a Nyumi Bruja volando, debido a que su habilidad está activada, mientras tiene el power-up del escudo activo.

Figura 7.3.2 *In-game power-up escudo*

En la *Figura 7.3.3* se visualiza a Nyumi Default saltando entre las paredes del castillo. Nótese que la barra de carga de la pasiva está inhabilitada debido a que este personaje no posee ninguna habilidad especial.

Figura 7.3.3 *In-game escenario Castillo*

Por último, podemos ver el escenario de la ciudad junto a Nyumi Serpientes apoyada en la pared situada a la derecha. Nótese que no se pueden ver los obstáculos de la otra pared, debido a que la especie Nyumi tiene la visión limitada. Ver *Figura 7.3.4*.

Figura 7.3.4 *In-game escenario Ciudad*

7.3.2 *MULTIPLAYER*

En este apartado se verán las capturas realizadas en modo multijugador.

En la *Figura 7.3.5* tenemos una partida con Nyumi Caramelo en el escenario del bosque.

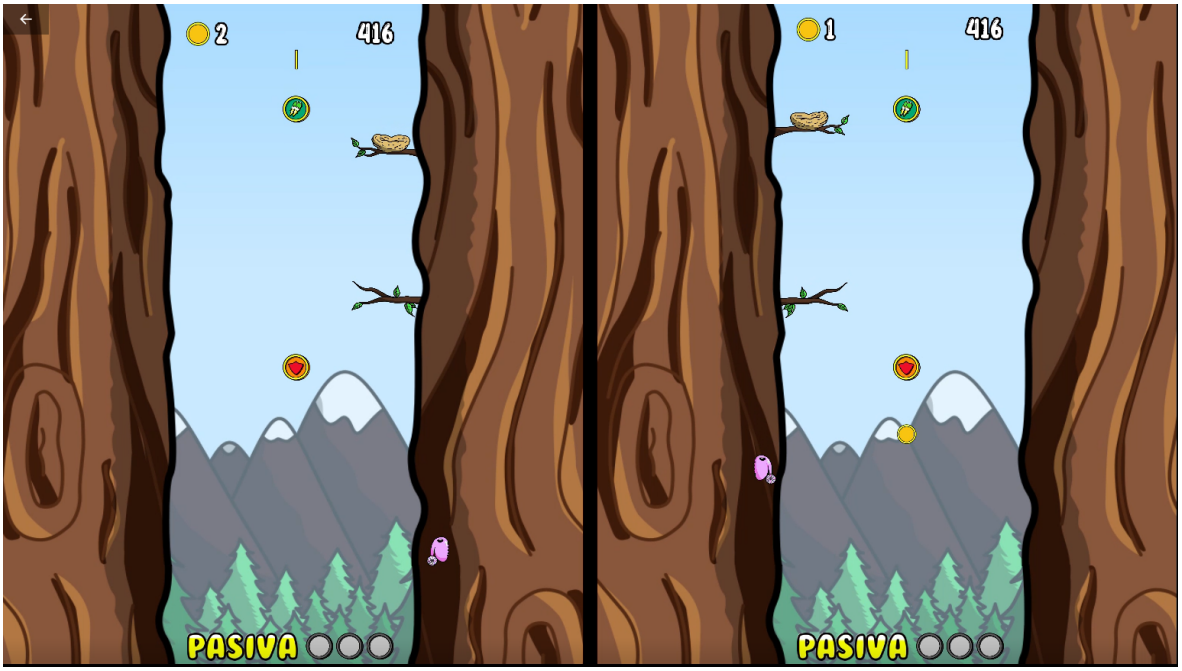


Figura 7.3.5 *In-game escenario Bosque*

Por otra parte, podemos ver a Nyumi Hongo en el escenario de caramelo con su habilidad activada, la cual le permite ver más monedas. Ver *Figura 7.3.6*.

Figura 7.3.6 *In-game pasiva Nyumi Hongo activada*

En la imagen siguiente (*Figura 7.3.7*) podemos ver cómo el jugador 2 perjudicó a su contrincante haciendo uso del power-up que nubla la visión.

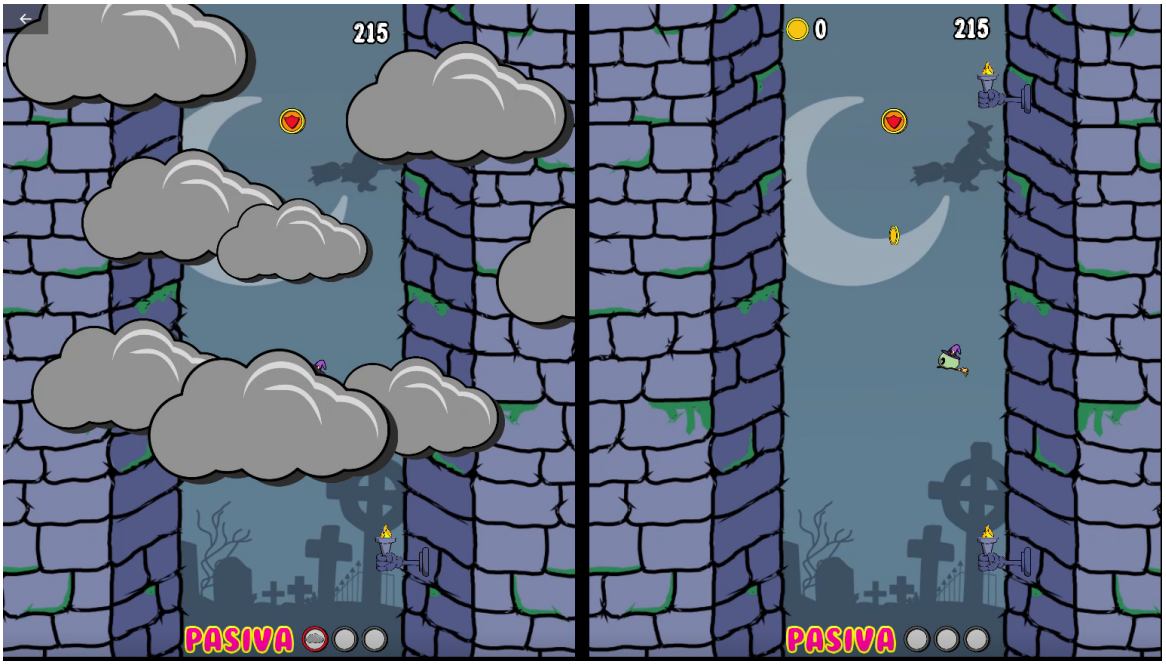


Figura 7.3.7 *In-game power-up nube*

Por otra parte, la *Figura 7.3.8* muestra cómo la puntuación del segundo jugador se encuentra congelada debido a que su contrincante usó un power-up contra él.

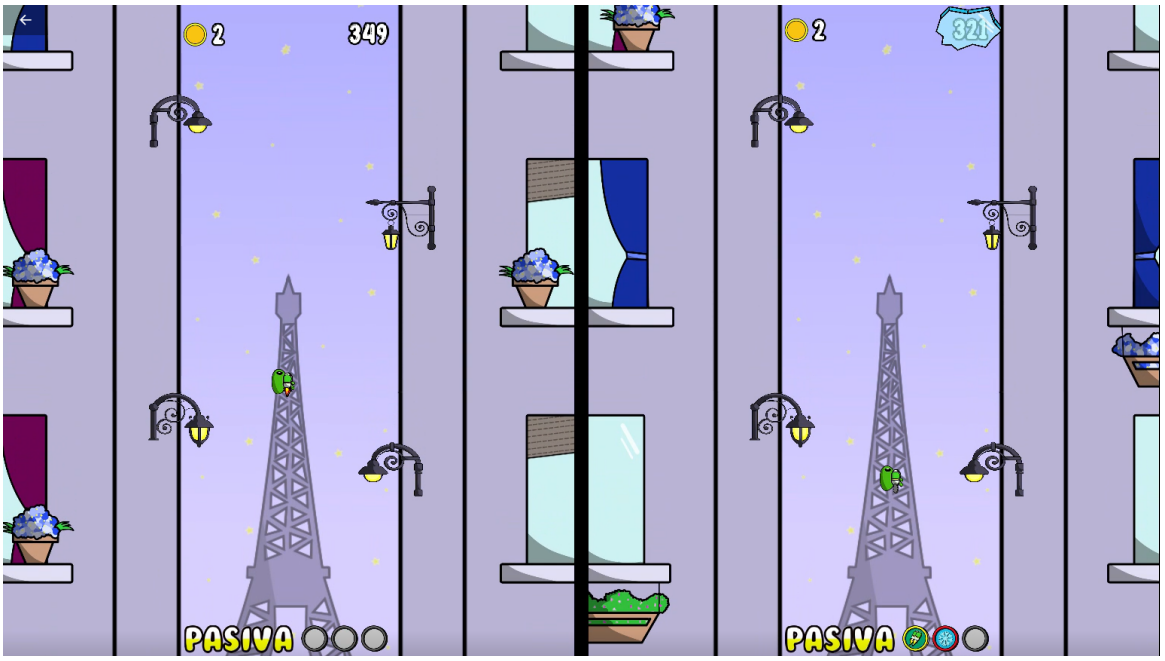


Figura 7.3.8 *In-game power-up hielo*

No obstante, no podemos olvidar que un jugador puede morir antes que el otro, en cuyo caso la pantalla del juego sería la representada por la *Figura 7.3.9*.

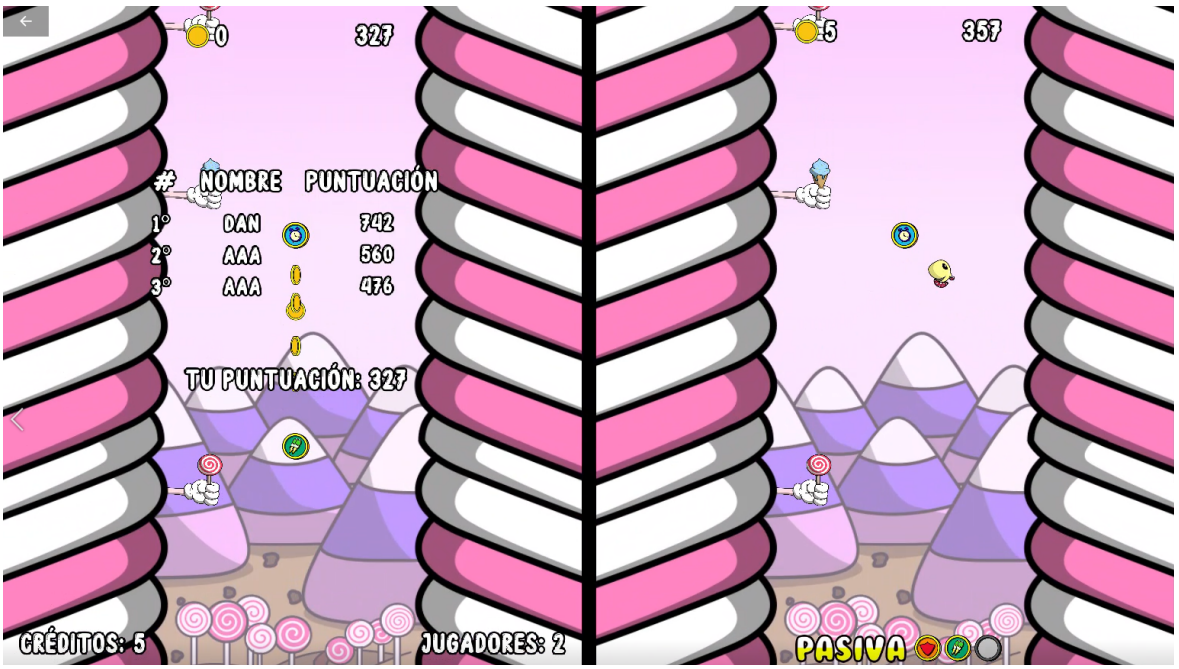


Figura 7.3.9 Primer jugador derrotado

7.4 PANTALLA FINAL

En este apartado mostraremos capturas de los distintos estados que puede tener la pantalla final en ambas modalidades de juego.

7.4.1 *SINGLEPLAYER*

Las siguientes imágenes son capturas de partidas de un solo jugador.

En caso de que el jugador haya superado un récord y deba introducir su nombre, la visualización es la mostrada por la *Figura 7.4.1*.

Figura 7.4.1 Nuevo récord *singleplayer*

Por otra parte, una vez introducido el nombre, la pantalla final sería la mostrada en la *Figura 7.4.2*.

Figura 7.4.2 Pantalla final *singleplayer*

En caso de no haber superado ningún récord, la pantalla final podría ser una de las representadas por las Figuras *7.4.3* o *7.4.4*.

Figura 7.4.3 Pantalla final sin créditos

Figura 7.4.4 Pantalla final con créditos

7.4.2 *MULTIPLAYER*

En caso de que la partida sea multijugador, pueden darse diversos casos: que ninguno supere un récord, que lo supere uno de ellos o que lo hagan ambos.

La *Figura 7.4.5* muestra una partida en la que ambos jugadores empataron, pero ninguno de ellos superó ninguna de las máximas puntuaciones.

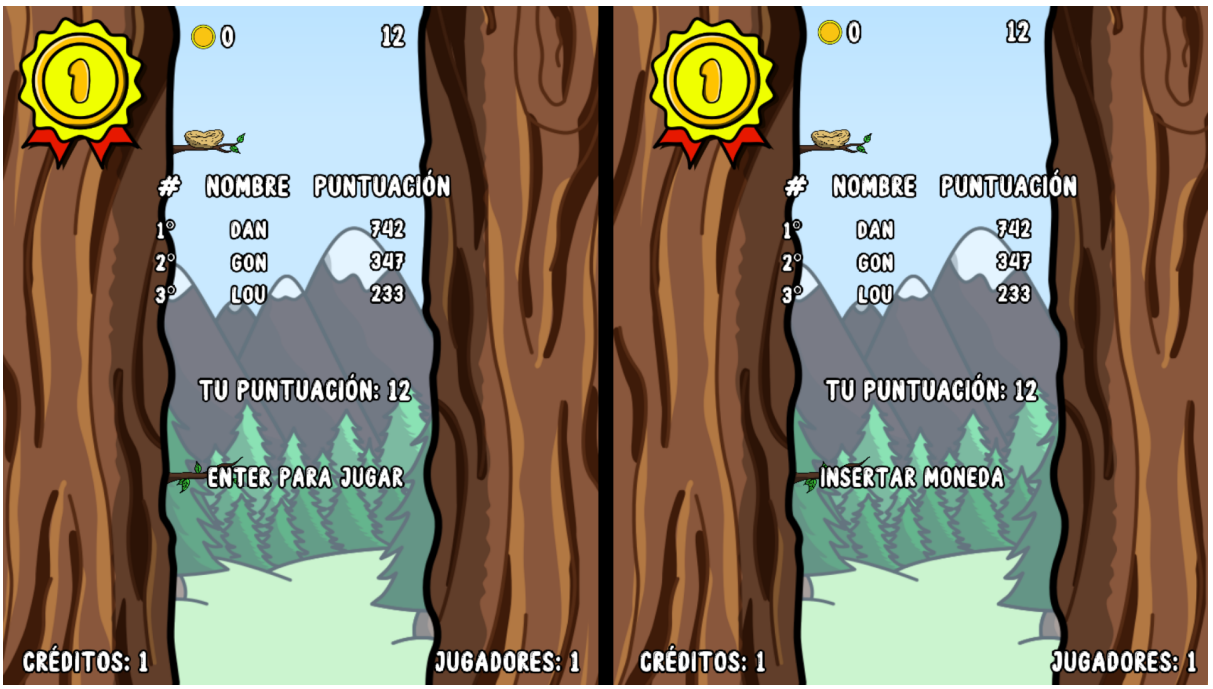


Figura 7.4.5 Empate sin récord

Por otra parte, la *Figura 7.4.6* muestra un final de partida en el que el primer jugador se proclama vencedor y, a su vez, supera una de las mejores puntuaciones registradas.

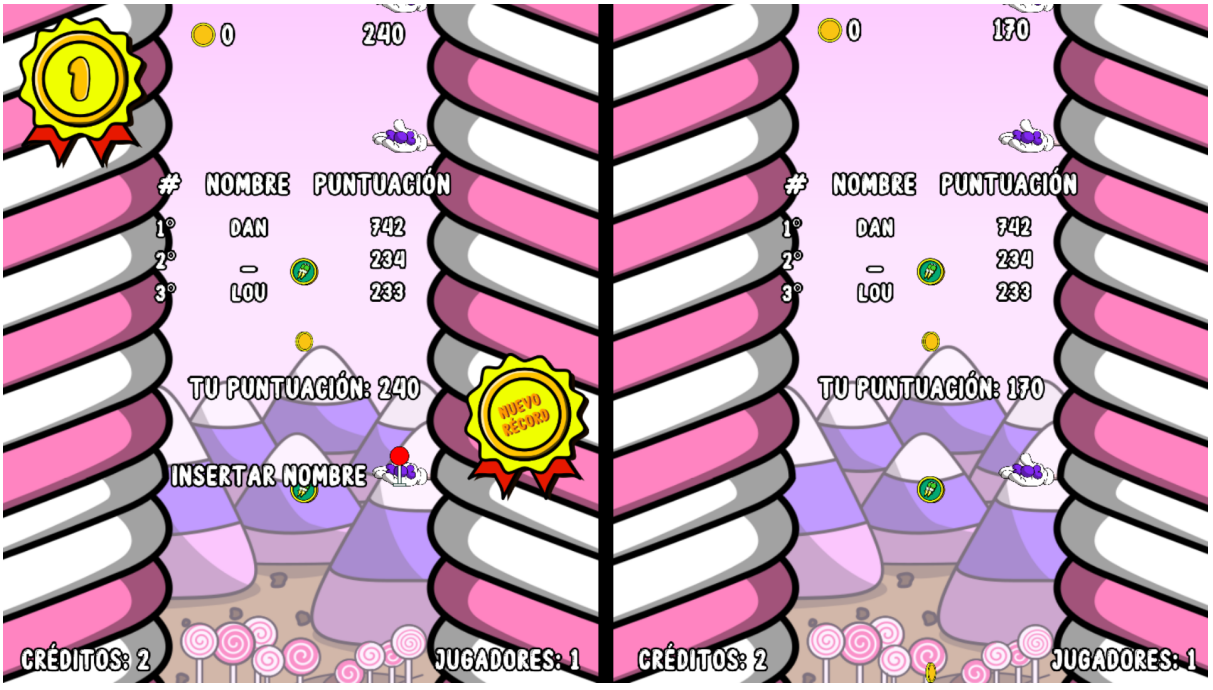


Figura 7.4.6 Récord superado por el primer jugador

Para acabar, la *Figura 7.4.7* representa la pantalla final en la que el segundo jugador es vencedor, pero aún así ambos quedan entre las mejores tres puntuaciones.

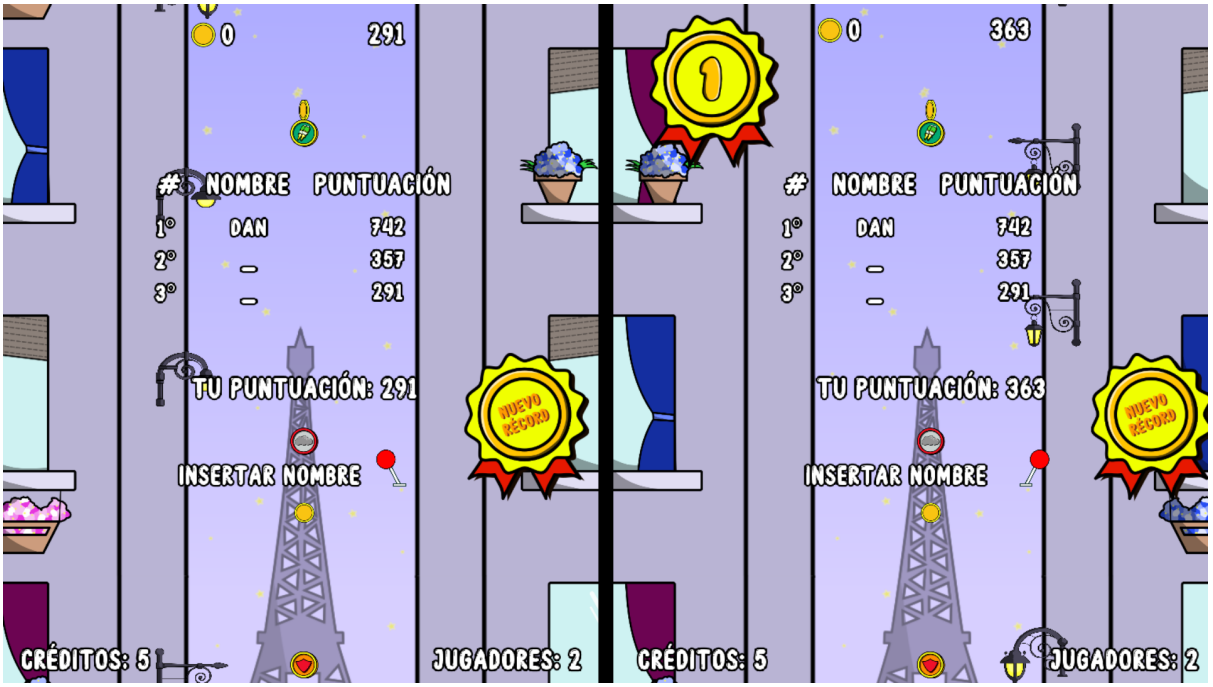


Figura 7.4.7 Récord superado por ambos jugadores

8. CONCLUSIONES

El objetivo principal de este proyecto era diseñar y construir una máquina arcade *low-cost* junto a un videojuego arcade y otro juego más simple, ambos de creación propia. A nuestro parecer, esta meta fue alcanzada con éxito, presentando como prueba esta documentación.

Tenemos que admitir que participar en este proyecto fue realmente emocionante y gratificante a nivel personal. El hecho de llevar a cabo un proyecto de principio a fin, pasando por todas sus fases de desarrollo, incluido su planteamiento, nos permitió dar lo mejor de nosotros y demostrar nuestras capacidades, incluido a nosotros mismos.

Este trabajo no sólo requería que pusiéramos en práctica todos los conocimientos obtenidos a lo largo de nuestra estancia en nuestros respectivos grados, sino que también era imprescindible un trabajo constante de investigación. Durante el camino hubo baches, agujeros y muros, los cuales pudimos atravesar haciendo uso del conjunto de nuestros conocimientos previos, nuestra innovación y los resultados de todas nuestras búsquedas de información.

Cabe decir que estamos satisfechos no sólo con el resultado, sino con la manera en la que transitamos este largo camino.

Conseguimos construir una máquina recreativa con un diseño único y acompañada por dos videojuegos funcionales diseñados e implementados por nosotros desde cero. Este logro nos complace enormemente y nos motiva a seguir adelante con nuevos trabajos, sabiendo que pudimos llevar a cabo un proyecto a nivel prácticamente profesional con un equipo de solo dós personas.

9. TRABAJO FUTURO

A pesar de nuestra satisfacción respecto al resultado final de este proyecto, nos gustaría mejorar algunos aspectos e implementar nuevas características, entre los cuales encontramos:

- **Incorporar un selector de monedas.** Nuestra máquina consta de dos botones para simular la introducción de dinero; no obstante, nos gustaría comprar un dispositivo real de manera que, si se diera el caso, pudiésemos comercializar nuestra recreativa.
- **Mejorar la interfaz del menú principal** desde el cual se pueden seleccionar los juegos y hacerla más atractiva visualmente.
- **Añadir nuevos juegos** implementados por nosotros**,** tanto de diseño propio como basados en otros videojuegos ya existentes.
- Respecto a *Up, up! Nyumi!* nos gustaría trabajar en lo siguiente:
    - **Nuevos power-ups** (de ambos tipos) para ampliar su variedad y evitar su excesiva repetición, sobretodo a nivel de un solo jugador.
    - **Diseñar nuevos personajes,** con sus respectivas habilidades y **escenarios**.
    - **Implementar obstáculos dinámicos** en las partidas, como por ejemplo: que el pato que está en el bosque sea capaz de volar y trasladarse de una rama a otra.
    - **Guardar un registro personalizado** de las mejores puntuaciones, de manera que cada personaje tenga sus propios récords a superar y no sea genérico para todos.
    - **Añadir parámetros extra** para configurar el juego a un nivel más interno. De manera que el propietario de la máquina pueda optar por desactivar los power-ups de las partidas o escoger entre una dificultad fácil (donde los obstáculos puedan verse en todo momento y las velocidades sean menores) o estándar (implementación actual).

Además de los puntos anteriores, nos gustaría refinar todos los detalles necesarios, tanto a nivel de hardware como de software, y solucionar cualquier *bug* que pueda surgir.

10. BIBLIOGRAFÍA

GitHub, 8 de febrero del 2008. GitHub, Inc.. 2020.
https://github.com/

Godot Engine - Q&A. Godot Technologies. 2020.
https://godotengine.org/qa/

Godot Engine documentation. Godot Technologies. 2020.
https://docs.godotengine.org/es/stable/index.html

Stack Overflow, 2008. Stack Exchange, Inc.. 2020.
https://stackoverflow.com/

Wikipedia, 2020. Jimmy Wales. 2020.
https://www.wikipedia.org

Diseño conceptual de videojuegos, 2019. Imma Boada y Núria Puig. 2020.
Universidad de Gerona.

GDQuest. Home [YouTube Channel]. 2021.
https://www.youtube.com/channel/UCxboW7x0jZqFdvMdCFKTMsQ

AwesomeTuts. Home [YouTube Channel]. 2021.
https://www.youtube.com/c/awesometuts/featured

11. ANEXOS

11.1 PROYECTO GODOT *UP, UP, NYUMI!*

Como uno de los anexos, adjuntamos el proyecto entero de Godot del videojuego *Up, up! Nyumi!*, su ejecutable y un vídeo de demostración.

Dentro de la carpeta *ProyectoTFG* pueden explorarse todos los assets creados, así como el código escrito. Los *sprites* se encuentran en la ruta *ProyectoTFG*/*Assets/Sprites*, en la cual se encontrarán los subdirectorios que contienen: los escenarios (*ProyectoTFG*/*Assets/Sprites/Fondos*), los personajes (*ProyectoTFG*/*Assets/Personaje*), los obstáculos (*ProyectoTFG*/*Assets/Obstaculos*) y otros objetos (*ProyectoTFG*/*Assets/Items*).

Para acceder al código hay que ir a la carpeta *ProyectoTFG/Assets/Scripts*.

Por otra parte, en la carpeta *EjecutableTFG*, encontraremos el archivo ejecutable (*upUpNyumi4.0.exe*) para poder probar el videojuego en un ordenador con un sistema operativo Windows y una pantalla con resolución mínima de 1920x1080.

Por último, el directorio *DemoArcade* contiene imágenes y vídeos de demostración del videojuego ejecutado sobre la máquina recreativa.

11.2 DISEÑO VINILOS

Como comentamos durante este documento, una parte de nuestro proyecto consistía en construir una máquina arcade, para la cual quisimos diseñar nuestros propios vinilos y así personalizarla.

En esta sección veremos los diseños descartados y el resultado final.

11.2.1 DISEÑOS DESCARTADOS

Como en todo proceso de creación, al diseñar los vinilos de la máquina hicimos una cantidad de pruebas y bocetos inimaginables. Aunque la mayoría fueron descartados, nos gustaría incorporarlos en esta sección para recordar que fueron ellos los que nos ayudaron a llegar al diseño definitivo.

En las siguientes imágenes (Figuras *11.2.1*, *11.2.2*, *11.2.3* y *11.2.4*) podemos ver los diseños que consideramos más relevantes.

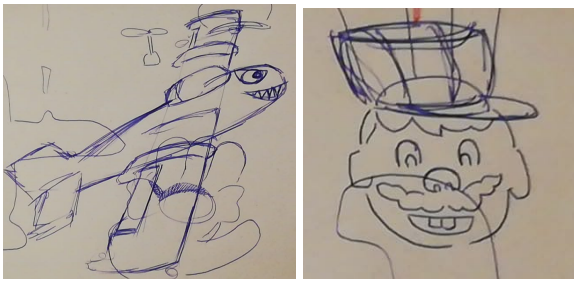


Figura 11.2.1 Bocetos descartados

Figura 11.2.2 Marquesinas descartadas

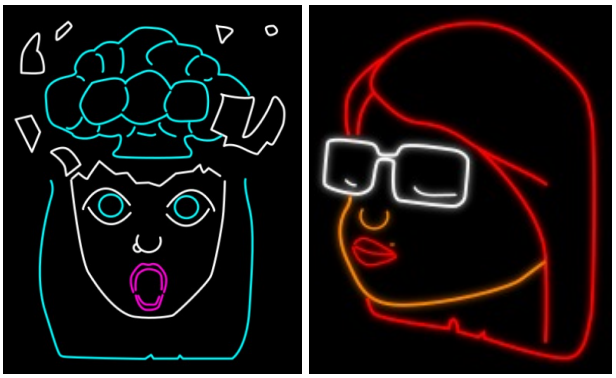


Figura 11.2.3 Elementos marquesina descartados

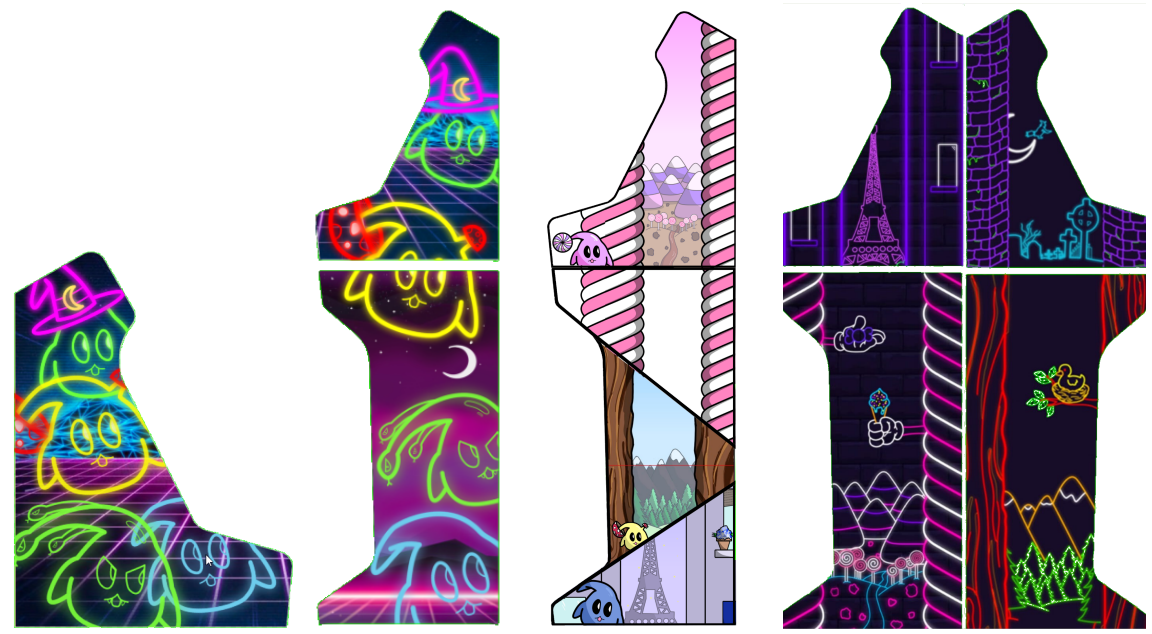


Figura 11.2.4 Diseños laterales descartados

11.2.2 DISEÑO FINAL

La *Figura 11.2.5 Diseño final* muestra un montaje con todas las piezas diseñadas para obtener una mejor visualización del resultado final en su conjunto.

Figura 11.2.7 Diseño final