Escuela Politécnica Superior

Diseño e implementación de "serious games" para entrenamiento de prótesis mioeléctricas

Máster Universitario en Automática y Robótica

Trabajo Fin de Máster

Autor:
Pablo Bey Cabrera
Tutor/es:
Andrés Úbeda Castellanos

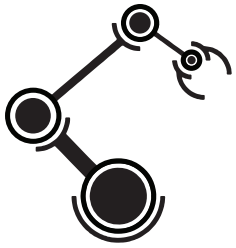
Julio 2019

Universitat d'Alacant
Universidad de Alicante

Diseño e implementación de "serious games" para entrenamiento de prótesis mioeléctricas

Autor
Pablo Bey Cabrera

Tutor
Andrés Úbeda Castellanos
Física, Ingeniería de Sistemas y Teoría de la Señal



Máster Universitario en Automática y Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2019

Preámbulo

Tenía muchas posibilidades a la hora de escoger el camino que iba a recorrer al acabar la carrera. Enfocarme en trabajar, empezar un máster de programación de aplicaciones móviles o pasarme un año con proyectos personales a la vez que estudiaba idiomas. Mi especialidad es la computación, en concreto, la gráfica. Sin embargo, quise hacer este máster porque en ese instante de mi vida necesitaba algo que me aportase nuevas visiones y entornos.

Al hacer este máster, encontré no solo profesionales muy buenos dándonos todos los conocimientos que necesitábamos, sino también lo increíble que es el mundo de la automática y la robótica. Todos los procesos industriales, todas las matemáticas que hacen posible estos mismos procesos y herramientas. Todo ello amplía nuestro paradigma y nos hace ver el mundo de otra manera.

La robótica y la computación gráfica no suelen tener tanto en común como uno esperaría, a pesar de que la robótica depende del uso de herramientas de computación gráfica para algoritmos, mostrar gráficas, realizar simulaciones, etc. Es por ello que quería hacer un trabajo que uniera los dos campos lo máximo posible. Cuando vi la propuesta de este proyecto, supe que era mi oportunidad. Y gracias a mi tutor, ha sido posible.

Agradecimientos

Para empezar, me gustaría agradecer a mi familia, que han estado en buenos y malos momentos, por haberme apoyado en todo lo que me proponía. A mi novia. Sin ella, hubiera terminado antes este proyecto, pero no hubiera disfrutado tanto el camino.

Javier Pascual Sarrazin, por ayudarme a revisar este texto y enseñarme a escribir mejor. A Diego Lezcano Reissner, el hombre, el mito, la leyenda; por sus consejos y acompañarme en esta aventura que es la vida. No hay suficientes trabajos en el mundo donde escribir agradecimientos para estos amigos.

Eblem Ramírez González, por el diseño y creación de todos los gráficos en *Balance Journey*. El trabajo de este profesional ha aportado al juego un pulido visual increíble.

Todos mis compañeros y profesores de máster por el tiempo juntos en clase, ayudándonos en las prácticas y compartiendo conocimientos que me han aportado mucho. En concreto, hacer mención a Jorge Pomares Baeza por su sentido del humor al permitirnos hacer la presentación de Faustino Fausto y *FAST*. A Andrés Márquez Ruiz y Cristian Neipp López por fascinarme con el mundo de la óptica.

Además, agradecer a uno de mis profesores del Grado en Ingeniería Informática, Francisco José Gallego Durán. Aún a día de hoy, sus lecciones, resuenan en mí.

Y, por último, pero no por ello menos importante, agradecer a mi tutor Andrés Úbeda Castellanos por apoyarme, guiarme en todas las ideas y, sobretodo, por darme la oportunidad de trabajar en un proyecto tan apasionante como este.

Dedicado a Francisco José Gallego Durán.
Como prueba de que tus lecciones han hecho de mí un mejor profesional.

Un viaje de mil millas
comienza con un primer paso

Lăo Zĭ

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1. Introducción

Actualmente las prótesis mioeléctricas permiten reemplazar una extremidad ausente por un sustituto robótico que se controla a voluntad. Sin embargo, el entrenamiento que lleva poder dominar la prótesis es un proceso difícil y, muchas veces, hasta repetitivo. Lo que puede llevar a que no se siga el entrenamiento o, incluso, que se abandone en la ausencia de un fisioterapeuta.

Este proceso puede amenizarse con el uso de videojuegos, los cuales pueden proporcionar un reto ajustado a sus dificultades, evitando caer en la repetitividad. Gracias a ello los usuarios se sentirán más incitados a entrenar con la ayuda de estos videojuegos.

Además, todos los datos que se obtengan al jugar con el videojuego de entrenamiento servirán para mejorar estos mismos videojuegos y controlar la mejora en habilidad del jugador tanto en el juego como con la prótesis. Estos datos también pueden ser usados en la mejora de las mismas prótesis e, incluso, como base para comprobar la viabilidad de nuevas arquitecturas.

1.1. Serious Games

Alrededor de toda la historia humana se han usado los juegos para entretener, comunicar y socializar. El juego como tal es una parte intrínseca de la naturaleza humana. No es extraño que haya juegos cuyo propósito no sea solo entretener sino educar, entrenar o, incluso, investigar.

Los videojuegos o juegos digitales no son una excepción en este campo. De hecho, son capaces de sumergir al jugador a en mundo y, además, le motivan a realizar repetidas sesiones de juego. Si el propósito principal es educar, cuanto más se entretenga el jugador, más aprenderá. Favoreciendo el objetivo del videojuego. Josef Wiemeyer (2016)

Esto es lo que se conoce como un *Serious Game*. Un juego digital creado con la intención de entretener y alcanzar un objetivo adicional.

Este concepto se utiliza en el proyecto debido a que se diseñan y desarrollan varios *Serious Games* para personas con una extremidad ausente que vayan a usar prótesis mioeléctricas, aunque también se podría usar para personas que padezcan algún tipo de neuropatía.Oskar C. Aszmann (2017) El objetivo es tanto entretener como entrenar el uso de esta prótesis. Por lo tanto, estos videojuegos son *Serious Games*.

1.2. Unión con la Robótica

A pesar de implicar únicamente desarrollo de videojuegos, la robótica esta intrínsecamente enlazada. Ya que no son simplemente una serie de juegos, sino una plataforma en la que desarrollar y comprobar la interpretación de señales que envíe el cerebro.

Cuanto mejor sea la interpretación de dichas señales, mayor será la facilidad de los usuarios para controlar el juego y, a su vez, la prótesis robótica, además de recibir más datos para investigar este campo.

1.3. Relevancia

Al menos un 50% de las personas con extremidades superiores ausentes tiene problemas con el control y funcionalidad de una prótesis mioeléctrica. Oskar C. Aszmann (2017) Estos problemas pueden ser achacados a la necesidad de más entrenamiento. Sin embargo, los entrenamientos tradicionales pueden llegar a ser repetitivos, provocando su abandono. Los *Serious Games*, como se ha comentado previamente, abren la posibilidad a solucionar este problema.

De todos estos entrenamientos se pueden extraer muchos tipos de métricas, como la mejoría del paciente, cómo se interpretan las señales, efectividad del videojuego y posibles mejoras de los mismos. Sin embargo, en la envergadura de datos que ofrece el *Big Data*, se podrían tener en cuenta más métricas, como la fatiga, el rendimiento estadístico de los usuarios, cómo cada sesión de entrenamiento/juego afecta a la mejoría y momentos concretos de la jugabilidad que no tengan una progresión de dificultad correcta para el usuario.

Tal cantidad de datos, se podría aprovechar con el uso de redes neuronales frente a algoritmos matemáticos de interpretación. Para poner estas redes en práctica, se necesita una arquitectura adecuada y mucha cantidad de datos, los cuales servirían para aprender de las señales de todos los individuos y proporcionar una interpretación a movimientos en el brazo más precisa. Incluso estas mismas redes podrían aprender de un único usuario a medida que lo use, acostumbrando la red neuronal a su señal mioeléctrica particular.

Además, las personas sin ningún miembro ausente podrían beneficiarse del control de una prótesis robótica para hacer tareas peligrosas a distancia de una forma mucho más natural e intuitiva.

2. Marco Teórico

2.1. Prótesis mioeléctricas

Para usar un músculo, el cerebro envía una serie de señales eléctricas mediante sistema nervioso central que son recibidas por dicho músculo, activándolo en mayor o menor grado en función de la intensidad. Éstas se denominan señales mioeléctricas y, como son corrientes eléctricas, provocan un campo electromagnético. Por lo que, con electrodos se puede averiguar cuánto se activa un músculo leyendo la intensidad del campo que genera.

Las prótesis mioeléctricas se benefician de ello, ya que, cuando se activa un músculo que debería mover una parte de la extremidad ausente, se lee la señal mioeléctrica y se puede realizar el movimiento pertinente mediante los servomotores de la prótesis. Aunque, también se puede configurar para que el uso de un músculo o varios al mismo tiempo realicen una acción personalizada.

La señal recibida puede ser interpretada de diversas maneras, entre ellas, con la binarización. A través de un umbral, todo dato de la señal que lo sobrepase se clasificará como la activación del músculo y en el caso contrario como la desactivación. Con esta interpretación, solo existen dos estados: activado o desactivado, como se puede observar en la Figura 2.1. Cómo se escoge este umbral puede ser clave, ya que, cada persona y músculo tiene una intensidad de señal base, requiriendo una calibración individual.

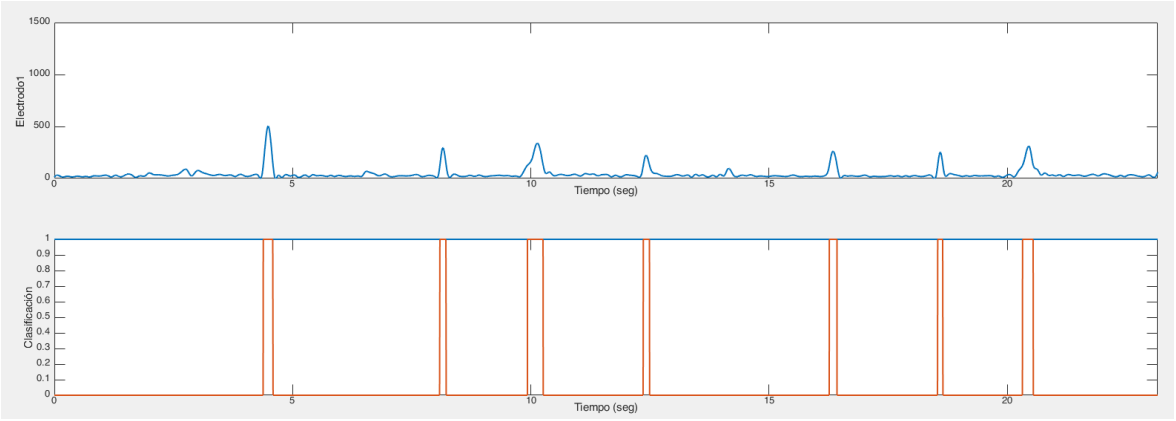


Figura 2.1: Ejemplo de una señal binarizada con un umbral de $\frac{1}{4}$ del valor máximo.

El otro tipo de interpretación es proporcional, la cual, permite observar el rango de fuerza que imprime el músculo en varios instantes de tiempo. Un ejemplo claro se encuentra en la Figura 2.2.

Se sitúa el enfoque en la prótesis que sustituye parte del brazo, en concreto, el izquierdo. Aunque, en teoría, el sistema podría recibir información de cualquier músculo. La plataforma

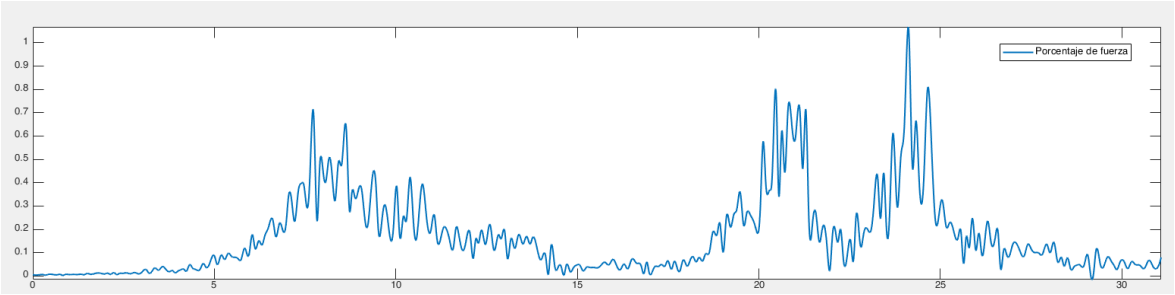
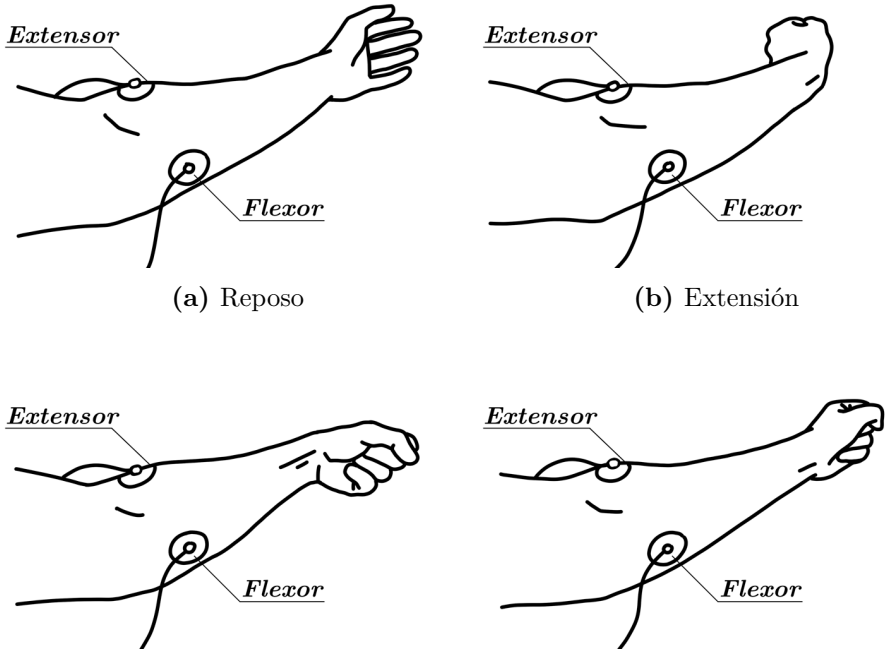


Figura 2.2: Ejemplo de una señal proporcional con porcentaje de fuerza.

se enfoca en dos músculos concretos: el extensor y flexor, donde se colocarán los electrodos para recibir las señales mioeléctricas. En la Figura 2.3 se puede observar las tres acciones que puede provocar el brazo: la extensión, flexión y cocontracción. La última de estas tres siendo provocada por accionar los dos músculos al mismo tiempo. Por lo tanto, la cocontracción es una unión entre la flexión y extensión.



(a) Reposo **(b)** Extensión

(c) Flexión **(d)** Cocontracción

Figura 2.3: Ejemplo de una señal proporcional con porcentaje de fuerza.

2.2. Game Feel

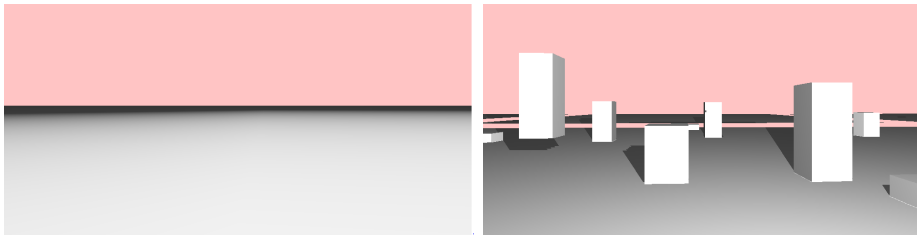
El *Game Feel* o sensación de juego se describe, en el libro de Swink (2009), como un tipo sensación virtual. Una unión entre lo visual, lo auditivo y lo táctil. Suele aparecer cuando la conexión entre el videojuego y la persona se fortalece hasta el punto que se puede sentir el movimiento, inercia y entorno de su propio avatar. Para poder comprender el concepto en su totalidad, se repasarán una serie de conceptos base que componen el *Game Feel* en pos de tenerlos en cuenta a la hora de desarrollar los videojuegos.

2.2.1. Conceptos Base

Interacción con el mundo virtual

Los usuarios, al jugar, crean un sistema de comunicación bidireccional entre la consola y ellos mismos. Mientras el aparato se comunica con el usuario a través de la pantalla con una imagen a la que reaccionar o a través de los controles con su vibración, el jugador se comunica con la consola a través de dichos controles que provocan cambios en el videojuego. La consola, entonces, retorna por sus periféricos los cambios estableciendo este bucle de retroalimentación.

Esta comunicación, como cualquier otra, tiene que ser bien transmitida para ser entendida. El entorno y los sistemas de referencia son algunos de los elementos de comunicación más importantes para generar sensación de movimiento y, por lo tanto, *Game Feel*. Sin un entendimiento espacial indicando dónde se encuentra el jugador o qué velocidad tiene, no se puede interactuar con el mundo correctamente. Por ejemplo, si se mueve en un mundo plano sin objetos ni otros elementos de referencia, es imposible apreciar el movimiento por ese mundo. Y si existen objetos de referencia, ¿cómo sabe si se esta moviendo él o el mundo? Esto se puede solventar añadiendo lineas de velocidad, una estela de viento que el objeto deje tras de sí, el sonido del propio viento al coger velocidad o un pequeño desenfoque de movimiento al mundo, pero no al personaje.



(a) **(b)**

Figura 2.4: Un mundo plano sin referencias (a) y uno con objetos de referencia (b).

Estos efectos no agregan funcionalidad al mundo ni cambian la lógica del juego de manera significativa, pero ayudan a convencer al usuario de que los objetos existen en ese espacio. Este tipo de efectos se denominan pulido. Para cada tipo de juego hay pulidos diferentes. No puede ser el mismo pulido que pueda tener un juego de rompecabezas, que un juego de acción o estrategia.

Cómo se interactúa con el mundo es relevante para generar una sensación de realidad al jugador e introducirle más en el entorno. Los objetos deben tener masa. Al chocar con

otros debería generar rebote en función de su elasticidad. Si las colisiones son siempre de un objeto inamovible frente a uno móvil que solamente para su movimiento al chocar, las interacciones son antinaturales y, por lo tanto, no mejoran la sensación de juego, incluso llegan a empeorarla.

Los componentes principales del *Game Feel* son el control a tiempo real, la simulación espacial y el pulido. Controlar a tiempo real implica que el videojuego y los periféricos reaccionen lo suficientemente rápido para que el jugador no sienta una desconexión entre sus decisiones inmediatas y la información que reciba del videojuego. La simulación espacial convence al jugador de que los objectos del mundo existen, creando unas físicas de mundo bien desarrolladas y definidas. Por último, el pulido que, como se ha comentado anteriormente, es todo aquel efecto que no añade funcionalidad, pero que convence al usuario de que los objetos existen en el mundo.

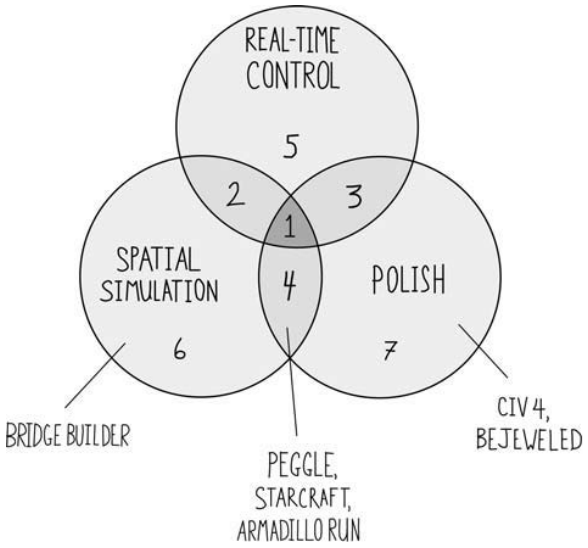


Figura 2.5: Diagrama de la sensación de juego del libro de Swink (2009).

Hay juegos que solo pueden tener dos de estos componentes, por ejemplo, los juegos de estrategia. Estos videojuegos carecen de control a tiempo real. Sin embargo, lo suplen enfocándose en pulido con todos sus efectos tanto visuales como auditivos. En la Figura 2.5 se puede observar que donde se cruzan todos los componentes se encuentra el *Game Feel* en su estado más puro.

Inmersión

La inmersión es un estado de alta concentración que sustituye la realidad por un mundo virtual. Es generada por un alto entretenimiento o proyección en el personaje principal, provocando que se actúe con él como una extensión del cuerpo. El avatar del jugador, cuanto más definido sea y más control tenga el usuario sobre él, más fácil es proyectarse. Además, si la cámara virtual que muestra todo lo que vemos en pantalla sigue los movimientos de usuario tal y como él desea, mayor es la inmersión. Pese a todo, existen elementos que pueden romperla y sacar al jugador de este estado. Si las físicas no son consistentes y llegan a ser impredecibles

para el jugador, puede dudar de su entorno y romper la ilusión de estar en ese mundo. Cuanto más se quiere simular el espacio de una manera más realista mayores son los problemas para mantener esta consistencia a nivel puramente técnico.

Propiocepción

La kinestesia o propiocepción es la capacidad de sentir la posición del cuerpo, su peso, el movimiento de los músculos, tendones y articulaciones. Esta sensación se puede experimentar si se cierran los ojos extiendes una mano y se juntan el dedo pulgar con el índice. Esta capacidad de poder conocer el estado del cuerpo pudiendo ejecutar una serie de pasos sin necesidad de ninguna retroalimentación visual o auditiva es la propiocepción.

En los videojuegos, la propiocepción virtual consiste en informar mediante referencias, la posición del jugador y todas sus partes. De esta forma, se genera la propiocepción virtual combinando todos los elementos visuales, auditivos, móviles y efectos táctiles para conseguir la sensación de pertenecer y estar en un mundo virtual. Como muestra, en el videojuego *Super Mario 64*, se informa constantemente al jugador de su posición con una sombra en el suelo del entorno de 3D, como se aprecia en la Figura 2.6(a). De hecho, al quitar la sombra se puede observar que es imposible conocer la posición exacta del fontanero. Ello genera al jugador confusión e incertidumbre sobre del espacio que le rodea y, en resumen, rompe la inmersión.

(a) **(b)**

Figura 2.6: Mario realizando un salto en el *Super Mario 64* con sombra (a) y sin ella (b).

Procesos cerebrales

Los tres procesos cerebrales implicados en la coordinación ojo-mano son el proceso perceptual, cognitivo y motor. En primer lugar, se reciben señales de los sentidos al cerebro y son interpretadas por el proceso perceptual. Éste, de todas las entradas, busca patrones, relaciones y generalizaciones, para entonces, enviar al siguiente proceso todos estos datos ya tratados. El cognitivo se encarga del pensamiento. Al recibir los datos del perceptual, compara el estado actual del sistema y el deseado para, entonces, decidir qué serie de acciones se deben realizar. El proceso motor, a su vez, recibe e interpreta del cognitivo, ordenando a los músculos que ejecuten esas mismas acciones deseadas. Estos tres procesos funcionan en paralelo, pero tienen un cierto retardo a la hora de procesar información nueva, como se ve en la Figura 2.7. Al menos, en un videojuego convencional, este planteamiento sería el de un sistema que convierte imágenes en acciones motoras.

Cuando estos tres procesos actúan al mismo tiempo, realizan constantes correcciones al movimiento en función de lo que se capte tras realizar alguna acción. Como en el caso de coger un vaso, la mente hace constantes correcciones a medida que el brazo se acerca al vaso para llegar al posición deseada y accionar los músculos adecuados para cogerlo. De hecho, este sistema es muy parecido al control de un brazo robótico.

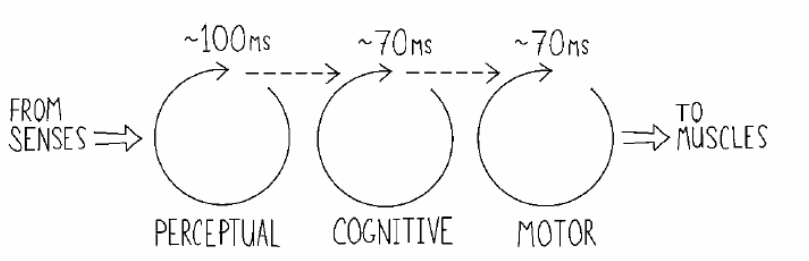


Figura 2.7: Diagrama de los tres procesos mentales del libro de Swink (2009).

El tiempo mínimo que tarda una persona en replanificar sus acciones se llama ciclo de corrección. Además, si el tiempo que tiene un usuario para poder reaccionar es demasiado grande, el control ya no es instantáneo y, por lo tanto, carece de un componente importante del *Game Feel*, el control a tiempo real. Normalmente, hasta los 150 ms de respuesta se siente un control instantáneo y hasta los 240 ms hay un cierto retraso que es perceptible, pero a partir de ese punto ya no se siente una respuesta a tiempo real. Sin embargo, la perdida de control puede ser ignorable si el jugador lo ha decidido, como es el caso de un personaje realizando un ataque fuerte, del cual no se puede recuperar rápido. Al ser consciente el usuario de la perdida de control y tomar esa decisión, no solo se está siendo coherente con el mundo, sino que, además no se confunde al usuario.

Los seres vivos, para poder percibir y entender nuestros alrededores, necesitamos poder interactuar en ellos. Swink (2009) Los videojuegos no son una excepción, ya que, para poder hacer que un mundo sea creíble y generar una correcta simulación espacial, se tiene que poder interactuar con el mundo y con sus elementos. También, para la simulación espacial, el mundo tiene que ser consistente. Aunque, en un videojuego no siempre es posible por los múltiples *bugs*, *glitches* y limitaciones técnicas que puedan surgir en el desarrollo. No por ello deja de ser uno de los principales elementos del *Game Feel*.

Retos y Metas

Por muy buenas mecánicas y *Game Feel* que tenga un videojuego, llevan al aburrimiento si no están acompañados por un reto. Se puede pecar de desarrollar videojuegos muy sencillos y accesibles, provocando que el entrenamiento se vuelva rutinario. Los retos deben servir, a su vez, para obligar al jugador a explorar todas las posibilidades mecánicas. Si un usuario puede completar el juego usando únicamente una sencilla mecánica, no explorará todas las posibilidades que le ofrezcan otras más complejas.

Los retos están compuestos de metas y limitaciones que ayudan a mantenerte atraído al propio juego. Las metas afectan a la sensación de juego, dándole al jugador la manera de comprobar su rendimiento en el juego. Si ha perdido o ganado, si ha fallado parcialmente y si lo ha hecho mejor que el último intento. Las limitaciones afectan al *Game Feel*, cambiando el

enfoque del juego al bloquear ciertos movimientos. En lugar de enfatizar un movimiento, una limitación quita algunos posibles movimientos del espacio simulado. En el caso de *Asteroids* al bloquearte el desplazamiento en horizontal y vertical, teniendo que rotar la propia nave para impulsarse, el jugador se enfoca más en los movimientos particulares de este sistema, lo que cambia la sensación del control.

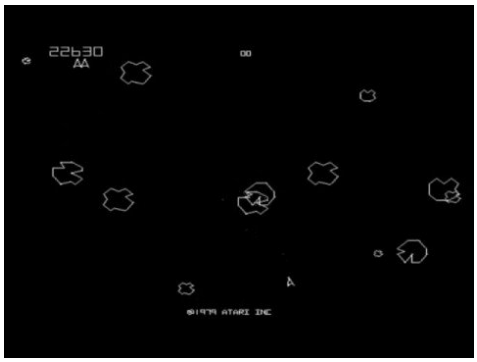


Figura 2.8: El videojuego de *Atari*, *Asteroids*.

Existen diferentes niveles de metas asociadas a las intenciones del usuario. Metas a corto, mediano y largo plazo. En *Super Mario 64*, una meta a largo plazo es derrotar a *Bowser* y liberar a la princesa secuestrada, la meta a medio plazo es conseguir la estrella del nivel y la meta a corto plazo es maniobrar para superar la plataforma que enfrente el jugador en ese instante. Todas ellas se pueden superar o fallar, y que las supere o no será un recordatorio del nivel de habilidad que tiene el jugador, permitiendo que, cuando consiga completar el juego, se hayan logrado todas las metas principales con la sensación de superación que ello conlleva. Aunque sea solamente a nivel narrativo, deben existir unas metas y objetivos que conseguir a la hora de mantener a los jugadores interesados al juego.

Métricas

Para examinar la efectividad del *Game Feel* se requieren métricas duras y blandas. Las métricas duras son medidas finitas y cuantificables que nos permiten estudiar a nivel científico cómo las personas juegan al juego. Por ejemplo, en qué partes se quedan atascados los jugadores, sus puntuaciones máximas, precisión con la que dominan los controles o rapidez de reacción.

Por otra parte, las métricas blandas contrastan los sentimientos provocados al jugar a un juego. Son métricas igual de relevantes que las duras, ya que, al final, nos indican lo cómodo que se encuentra el jugador. Estas métricas intentan registrar sentimientos como diversión, risa, peticiones de jugar más, si se habitúan a los controles, si no existe un reto real o confusión si algún concepto no ha sido bien explicado. Para poder obtener estas métricas, se tiene que desarrollar un informe ajustado al videojuego. Los jugadores deben rellenarlo al terminar una sesión de juego.

Los aspectos más importantes en el momento de crear las métricas blandas son la entrada, respuesta, contexto, pulido, metáforas y reglas. La entrada consiste en comprobar cómo las intenciones del jugador se traducen al sistema de juego y ver cómo afecta al *Game Feel*. La

respuesta indica si se cumple el control en tiempo real o no. El contexto es la interacción de los objectos entre sí en un espacio simulado y cómo ello afecta al control a tiempo real. El pulido comprueba si los efectos que no añaden funcionalidad sirven su propósito de crear un mundo creíble. La metáfora indica cómo el conocimiento preconcebido del jugador afecta a su relación con el mundo y sus interacciones. Las reglas miden la serie de variables arbitrarias del mundo que pueden cambiar la percepción del mismo al usuario. Con todas ellas se pueden realizar unas métricas completas.

2.2.2. Rendimiento y Hardware

Los controles intuitivos son una perfecta traducción de la intención del jugador a las acciones que se generan en el videojuego. Sin embargo, no siempre se puede alcanzar dicha traducción perfecta. Y se debe tener en cuenta que, si por una mala traducción se aumenta la dificultad, no es un reto, es **interferencia**.

Una mala traducción puede ocurrir fácilmente por limitaciones del hardware en el sistema que analiza las señales mioeléctricas. El juego tiene que responder en el menor tiempo posible. Sobre una media de 100 ms. Si la señal tarda en procesarse y enviarse, este retardo puede superar los 100 ms, aunque el límite crítico antes de que haya cierto retraso es 150 ms. Este problema técnico se enfoca más al hardware, el cual, a medida que avance la tecnología, puede acabar solventándose. Además, comprobar cómo este retardo afecta a la jugabilidad es una oportunidad para entender e investigar cómo estos retardos afectan a los usuarios de las prótesis.

3. Objetivos

El objetivo de este proyecto es crear una plataforma de *serious games*, orientada al entrenamiento de una mano protésica, con una serie de juegos y/o actividades sencillas, que recibirán como entrada las señales electromiográficas del usuario. Esta plataforma debe permitir futuras ampliaciones que expandan elementos como la cantidad de *serious games*, las maneras de control mioeléctrico o la base de datos.

En este listado se especifican cada uno de los objetivos inherentes al desarrollo de la plataforma de *serious games*:

Generales

- **Plataforma:** Diseñar e implementar la plataforma que dará acceso a los *serious games*, cuyo propósito será mejorar el control de los usuarios con manos prostéticas.
- **Controles:** Implementar el control en diversos juegos/actividades en función de las posibles entradas del control mioeléctrico.
- **Terapia:** La plataforma debe ofrecer un sistema complementario de terapia para personas que sufren discapacidades de tipo como motriz y cognitivo.
- **Moduralidad:** Modular la plataforma para que permita la introducción de nuevos juegos/actividades y controles en el futuro.

Específicos

- **Diseño de actividades/juegos:** Definir y diseñar los videojuegos a implementar en pos de evaluar su factibilidad dentro de la plataforma.
- **Desarrollo:** Desarrollar cada uno de los videojuegos en sus entornos 2D o 3D respectivos.
- **Tipos de actividad/juego:** Desarrollar actividades y/o juegos enfocados al entrenamiento de las manos prostéticas con diferentes controles.
- **Base de datos:** Añadir esta característica para monitorizar el progreso de todos los usuarios.
- **Evaluar:** Se realiza una evaluación de todas las características desarrolladas y se proponen mejoras y oportunidades de investigación.

4. Metodología

La metodología que usará el proyecto se desarrolla por iteraciones. A diferencia de otros procedimientos, se realizan las fases principales del desarrollo: planificación, análisis, diseño, implementación y pruebas; realizándose paralelamente en una iteración para después repetir el proceso con las mismas fases pero otros objetivos. Así sucesivamente hasta el objetivo final o hasta que se decida finalizar el proyecto.

4.1. Iteración

Las Iteraciones son periodos de desarrollo que se dividen en las fases anteriormente mencionadas, haciéndose simultáneamente a excepción de la planificación. En cada iteración se deben de tener claros los objetivos. Tanto los principales como secundarios. A partir de la planificación, cada fase de las iteraciones tiene asociada una serie de tareas que se coordinan para conseguir los objetivos antes mencionados, como pequeños hitos a realizar.

Planificación	Análisis
	Diseño
	Implementación
	Pruebas

<———— X semanas ————>

Tabla 4.1: Una iteración con una duración de X semanas.

4.2. Planificación

La planificación es una fase en la que se define todo el trabajo que se realizará en la iteración. En primer lugar, se definen las características a desarrollar dentro de los hitos planteados para la iteración. En segundo lugar, una característica será dividida en una *historia* por cada uno de los componentes a implementar de la característica. Por último, una historia se divide, a su vez, en tareas pequeñas con la cantidad de trabajo medida en horas y con cada una de las implementaciones necesarias para completar ese componente.

Característica	⟶	Historia	⟶	Tarea

Tabla 4.2: Modelo seguido en la planificación.

4.3. Análisis

Esta fase requiere analizar y comprender la tarea o problema, para extraer todas las posibles subtareas o nuevas tareas que requieran y, de esta manera, observar cómo se afectan unas a otras para actuar en consecuencia. La experiencia influye mucho a la hora de realizar con éxito esta fase, ya que, cuanta más experiencia, mayor será la facilidad de encontrar tareas adicionales, malas estimaciones en la cantidad de trabajo, errores que se hayan cometido en las especificaciones, tareas en una *historia* que no le corresponden y duplicación/solapamiento de tareas que pueden llevar a gastar los recursos de trabajo innecesariamente.

4.4. Diseño

El diseño comprende la resolución del problema, no implementándolo, sino con su funcionamiento. Tratando de valorar cuáles serán cada uno de los elementos que intervendrán en el proceso y compararlo con otros posibles diseños para entender por qué se ha escogido esa decisión y no otra. En primer lugar se dividen las tareas y subtareas para resolver los problemas de manera más sencilla, para, posteriormente, identificar la relación entre estas divisiones. Una vez han sido identificadas, se especifica la funcionalidad de las tareas y, también, las relaciones entre ellas y, por último, se definen cada uno de los componentes necesarios para la finalización de estas tareas.Sommerville (2005)

En el proceso de diseño es importante recalcar que siempre se debe abordar los problemas con enfoques diferentes, permitiendo escoger el mejor, apoyarse en diseños que ya se hayan hecho para encontrar la mejor manera de proceder, tener diseños versátiles a los que se puedan realizar cambios en pos de poder mejorar o cambiar el mismo, poder evitar errores a través de un diseño bien definido y, además, tener en cuenta que un diseño no es una referencia para el código, sino una serie de procesos y arquitecturas que ayudan a resolver un problema.

4.5. Implementación

La implementación como su mismo nombre indica, implica realizar la solución al problema o tarea. Se aplicará la solución diseñada previamente al proyecto y, entonces, se comprobará si la estimación de la cantidad de trabajo en horas ha sido correcta. Se registrará el margen de error con el objetivo de ser más precisos en futuras estimaciones. Si no se han realizado los diseños correctamente, es decir, no contemplan los suficientes casos o no especifican lo suficiente, se habrá arrastrado un error, proveniente del diseño o análisis, y se tendrá que solucionar en esta fase.

4.6. Pruebas

Al tener una implementación terminada, es inevitable que haya errores que puedan provocar el funcionamiento incorrecto del componente en ciertos casos específicos. Es por ello que las pruebas son una parte tan importante del desarrollo y se deben establecer un conjunto de comprobaciones para cada uno de los componentes, en pos de encontrar estos errores y solventarlos. Las pruebas se realizan a varios niveles: se parte de la tarea implementada

para comprobar el funcionamiento del componente, para, entonces, ir comprobando que su integración al resto de componentes relacionados no genera ningún problema y por último comprobar el funcionamiento en global con todo el proyecto.

Sin embargo, las pruebas se realizarán no solo a nivel de código, sino a nivel de integración. Cuando se haya generado el prototipo de un juego, este mismo se debe probar con los electrodos que recibirán las señales mioeléctricas. Si se usa el teclado como una simulación de señales, se pueden obtener muchas métricas, pero el ajuste de parámetros específicos a cada juego se debe realizar junto al control por señales.

Cada prueba o test podrán dar diferentes resultados, uno por cada persona que pruebe el sistema. Esto es debido a que cada persona tiene una coordinación visual-muscular, gustos y sensaciones diferentes. Además de que sus contextos también pueden serlo. Para valorar esta variedad de datos, se tienen que poner en perspectiva e intentar encontrar las razones que hayan llevado a esa persona en concreto a sentirse de esa manera y cómo se podría mejorar el juego en sí mismo.

Se deben tomar tanto métricas blandas como duras de un conjunto de personas, con el objetivo de encontrar fallos de diseño en el sistema y poder solventarlos en las siguientes iteraciones. Este proceso hará del proyecto una plataforma más completa y pulida.

4.6.1. Equipo de pruebas

El equipo son las herramientas que se poseen para poder hacer las pruebas. A continuación se lista una serie de las mismas:

- **DTS Lossless EMG Sensor:** Un par de electrodos, uno para el músculo extensor y otro para el flexor. Con ellos se pueden leer la señales electromiográficas y enviarlas a la estación receptora mediante ondas electromagnéticas. La batería, junto a la antena de estos dispositivos permite que no haya cables durante su uso.
- **TeleMyo Mini DTS:** Estación receptora de la cual pueden extraerse los datos de los electrodos a un computador para procesar las señales.

5. Desarrollo

5.1. Iteración 0: Puesta en marcha

La puesta en marcha empieza el desarrollo de la plataforma, cuyo nombre será *Huanghou*[1]. En esta iteración inicial se preparan todos los recursos necesarios en pos de poder empezar el desarrollo de los *serious games*.

5.1.1. Motor Base

Un motor gráfico es necesario para cualquier desarrollo de videojuegos. Su cometido es el de ahorrar al desarrollador la programación de código rutinario para mostrar elementos en pantalla, facilitando las herramientas con las que crear entornos y objetos en las dimensiones que permita el mismo motor.

Se usará un motor gráfico personalizado de tres dimensiones que ya se había desarrollado previamente, para, entonces, hacer pequeñas modificaciones como aceptar entornos en dos dimensiones. El motor tiene que ser capaz de aceptar estos entornos, porque, en los juegos de tres dimensiones, se suele requerir mucho más esfuerzo para conseguir un resultado satisfactorio que en los juegos de dos dimensiones. Una dimensión adicional puede complicar las lógicas de juego exponencialmente. Es por ello que la gran mayoría de juegos propuestos se enfocarán en entornos de dos dimensiones. Aún así, el motor ya tiene, de base, una serie de tecnologías para poder realizar todas las tareas requeridas, siendo estas tecnologías:

- **OpenGL:** Interfaz de programación de aplicaciones o API gráfica que permite renderización acelerada por hardware a través de la GPU. Para cagar las librerías de esta tecnología se utiliza GL3W.
- **GLFW:** Gestor de ventanas y entradas que permite escoger las características de la imagen que se muestra por pantalla y, además, gestionar la entrada por teclado, mando, etc.
- **GLM:** Una librería matemática de OpenGL para operaciones gráficas. Proporciona todo el código necesario para realizar diferentes tipos de operaciones matemáticas entre puntos, vectores y matrices de diferentes dimensiones.
- **Freetype:** Herramienta cuya función es cargar cualquier fuente de texto para renderizarla en un entorno.
- **SOIL:** Librería que permite la lectura de imágenes con diferentes formatos.
- **Dear imgui:** Un gestor de interfaces de usuario con ventanas, botones, campos de texto y todo lo que se necesite para una interfaz gráfica.

[1]Este nombre procede del chino, significa Emperatriz.

Aún así, estas tecnologías, por si solas, no forman nada de valor. Es la capa de código por encima la que define un motor gráfico, es decir, las características. Entre ellas se encuentran:

- **Cámaras:** Una cámara virtual capaz de mostrar el entorno que capta. Se pueden definir aspectos como su posición, orientación, campo de visión y alcance visual.
- **Textos:** Un texto plano predefinido y en dos dimensiones, que puede ser mostrado en cualquier posición de la pantalla.
- **Mallas:** Conjuntos de polígonos, en este caso triángulos, que se muestran en pantalla con una textura asociada. En la opción de dos dimensiones, las mallas son únicamente un plano formado por dos triángulos.
- **Ventanas:** Con ayuda de *Dear imgui*, se puede generar varias ventanas con sus propias características independientes y asociarlas con cualquier elemento del motor.
- **Transformaciones:** Al añadirse a cualquiera de los elementos comentados anteriormente, se provoca la transformación de posición, escalado y/o rotación deseada en ellos.
- **Abstracción:** Todos los elementos previamente mencionados se pueden abstraer como **Entidades**, pudiendo manejarlos con mayor facilidad.
- **Anclar:** Al anclarse una entidad a otra se aplican las transformaciones de la otra a la entidad anclada. De esta manera, se pueden asociar diferentes entidades a una misma transformación.
- **Texturas:** Únicamente, se cargan en la memoria de vídeo sin duplicados, usando la ruta como identificador. Si no existe la textura, se utiliza una por defecto.
- **Teclado:** Con GLFW se dan las herramientas para poder afectar a un elemento durante la pulsación o solamente al activarse inicialmente.
- ***Frames* variables:** Cada imagen es procesada en un tiempo variable, permitiendo ajustarse a la frecuencia del monitor, o menor, si el procesamiento gráfico es muy pesado para la máquina.

El motor funciona por ciclos de procesamiento. Cada vez que se procesa una imagen que mostrar por pantalla o *frame*, ésta se muestra y se vuelve a procesar con cierta frecuencia. La iteración de este bucle se conoce como ciclo de procesamiento. Para no confundir al monitor enviando varios *frames* por encima de su frecuencia, se realiza un proceso conocido como sincronización vertical que, valga la redundancia, sincroniza el ciclo de procesamiento para que funcione a la par que la frecuencia del monitor. Este ciclo de procesamiento tiene una serie de pasos definidos en la Figura 5.1.

Al ser un motor personalizado, tiene una serie de ventajas e inconvenientes. Siempre que haya un error, éste se puede arreglar sin muchos problemas. Si se usara un motor externo, al haber un error en el propio motor, sería necesario averiguar primero de dónde procede el error, si del motor o de un mal uso de él. Si no es de un mal uso, se tendría que invertir tiempo en evitar el error a lo largo del programa, ya que, muchos motores no tienen su código libre al público. En cambio, si el error es del motor personalizado, al conocerlo en profundidad,

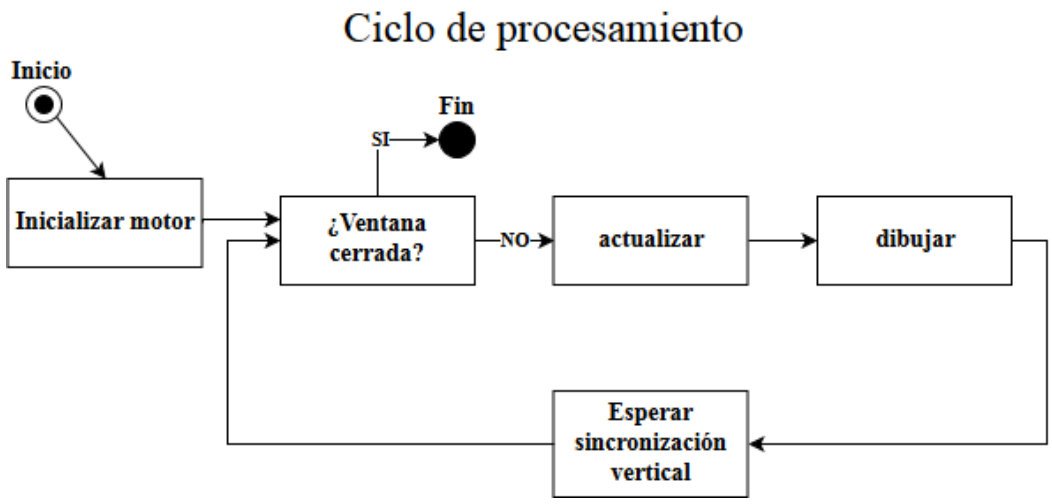


Figura 5.1: Ciclo de procesamiento del motor.

se reduce el tiempo de encontrar y solventar el error. Además, al tener este conocimiento profundo, se facilitaría la adición de módulos de funcionalidad como los controles, entre otras características. Sin embargo, una de las grandes desventajas es que, al ser personalizado, puede no tener todas las características que si tendría un motor externo de desarrollo longevo. Entonces, si no hay alguna característica, se tendrían que invertir recursos para desarrollarla, en lugar de utilizarlas directamente en los propios juegos.

Las ventajas predominantes que inclinan la balanza a favor del motor personalizado son la moduralidad y la posibilidad de añadir en el futuro nuevos elementos con facilidad, debido a que el motor no depende de ninguna versión en concreto para crear el programa, porque todas las tecnologías, con sus respectivas versiones, están integradas en el proyecto, a diferencia de otros motores gráficos externos. Entonces, al ser mínimas las dependencias del motor, se asegura la capacidad de añadir nuevas características, juegos o elementos durante un largo periodo de tiempo.

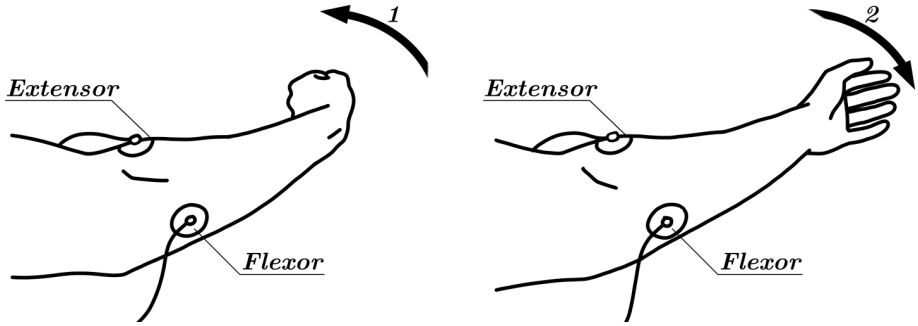
5.1.2. Control

El control consta de tres posibles acciones: extensión, flexión y cocontracción. Estas tres pueden ser interpretadas de diferentes maneras. Como se ha visto anteriormente, existen dos tipos de interpretación en las señales: el binario y el proporcional. Cada uno de ellos tiene sus propias salidas. El binario solamente tiene dos estados posibles: activado, que se representa con el 1, y desactivado, representado por el 0. En ese espacio entre 0 y 1 es donde trabaja el proporcional. Al tener como salida un porcentaje de fuerza en función del grado de activación del músculo, este tipo de señal es un eje con rango desde el reposo, cerca de 0.0 ó 0.0, hasta el máximo de fuerza del músculo, es decir, el 1.0. Esta interpretación permite los estados intermedios existentes en los números reales entre el 0.0 y 1.0.

Aún así, la plataforma no se encargará de la interpretación, sino del tipo de control y los datos ya interpretados. Para conseguirlo y hacer que sea lo más modular posible, siempre se enviarán los ejes de las tres acciones al programa como reales, de forma que, tanto si las interpretaciones son binarias como proporcionales, siempre se reciba un número entre 0.0 y

1.0. La cocontracción se recibe también como un real a parte, aunque sea el resultado del extensor y flexor, debido a que la plataforma no debe encargarse de interpretar las señales y, por lo tanto, tampoco de detectar la cocontracción.

Hay diversos tipos de control que se pueden implementar y, dependiendo la actividad o videojuego, puede ser más beneficioso usar uno u otro. En la Figura 5.2 se observa un movimiento percutante con dos pasos: extensión y reposo. Con este ejemplo, comprobaremos cómo afectan los datos recibidos al tipo de control en un solo músculo. Suponiendo que la señal interpretada es la misma en cada control.



(a) 1 **(b)** 2

Figura 5.2: Pasos de un movimiento percutante usando el flexor.

Control Proporcional

El control proporcional capta un dato en el rango de 0.0 y 1.0, provocando que el personaje pueda ser manejado con un eje como si de un *joystick* se tratase. Los datos que se reciben a lo largo del tiempo son lo más parecido a la señal interpretada. De hecho, en el binario serían estos datos aplicando un umbral.

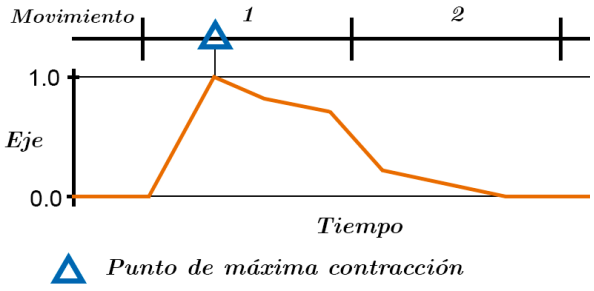


Figura 5.3: Ejemplo de control proporcional.

Características:

- Solo puede ser usado en juegos que puedan aprovechar los ejes.
- Alta precisión.

- Fuerza al usuario a usar con cuidado los músculos.
- Usa menos la fuerza máxima, minorando la fatiga.
- Difícil de usar inicialmente, requiere haber usado otros controles más sencillos previamente.

Control Binario

El control binario capta un 0.0 o cualquier otro dato, normalmente 1.0. Para poder representar desactivado y activado, se usan estos dos números. Cuando el control binario está activo, se esperan que los datos ya estén binarizados bajo un umbral, aunque, si no lo están, cualquier número distinto de 0.0 será convertido a 1.0.

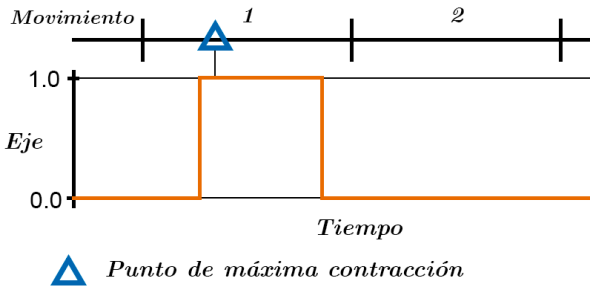


Figura 5.4: Ejemplo de control binario.

Características:

- Adaptable a muchos tipos de juegos.
- No es recomendable usarlo para mantener una pulsación. El usuario se fatigaría con facilidad.
- Movimiento percutante necesario, juegos de reflejos y reacciones rápidas constantes fatigan al jugador.
- Juegos de ritmo pausado pero constante pueden beneficiarse de este control.
- No requiere gran precisión.

Control por Flanco de Subida

Este control deriva del binario y se hará por *software* en la plataforma. Solamente modificará el comportamiento del control binario. El Flanco de Subida, como su mismo nombre indica, solo se activa en el cambio de estado desde 0.0 hacia el 1.0. Aunque se reciban por el canal de transmisión más 1.0, si el anterior dato recibido no es un 0.0, el eje no es activado.

Características:

- Se activa únicamente en un instante de tiempo.

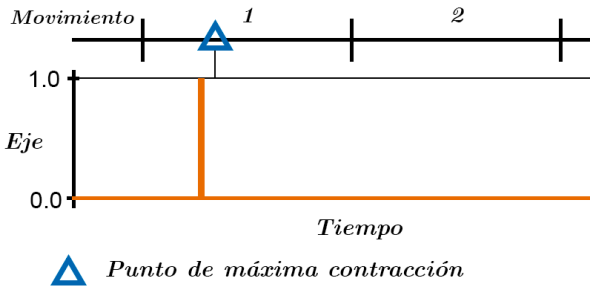


Figura 5.5: Ejemplo de control por flanco.

- Lo deben utilizar juegos que requieran una entrada fija y muy controlada.
- Fácil de utilizar ya que evita posibles errores.
- Los juegos que requieran reflejos o una entrada continua no se benefician de este control.

Control por Flanco de Subida T

Basado en los biestables tipo T, este control derivado del Flanco de Subida estándar cada vez que el programa recibe un flanco de subida se realiza un intercambio de la anterior salida, como se puede observar en la Figura 5.6. Sin embargo, este control se analiza de manera anecdótica porque es poco adaptable.

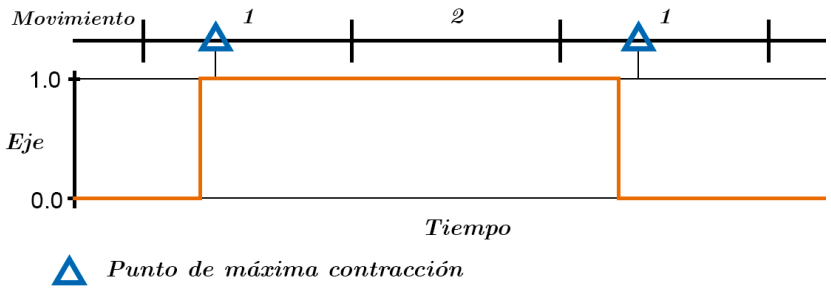


Figura 5.6: Ejemplo de control por flanco tipo T.

Características:

- Para juegos que requieran entrada continua pero no reflejos muy rápidos.
- Fácil de usar, pero puede llevar a movimientos indeseados. Ello se tiene que tener en cuenta en el juego que lo implemente.
- La fatiga es mínima con este control.

Protocolo de transmisión

La comunicación entre el intérprete de señales y la plataforma de *serious games* tiene que ser modular, compatible con cualquier tipo de sistema y rápido. Para ello, se usa el protocolo de

comunicación *UDP*, el cual permite enviar y recibir datos a cualquier máquina rápidamente, con el riesgo de perder los datos con mayor facilidad. Para solventar el problema de la pérdida, el intérprete de señales y la plataforma deben estar en la misma máquina, haciendo imposible dicha pérdida.

Al motor entonces se le incluye no solo una nueva característica, un servidor *UDP*, sino también un hilo de procesamiento añadido para poder procesar paralelamente todos los datos que envíe el interprete de señales.

De carácter más técnico, se explica que el protocolo de transmisión usará una ventana limitada a 12 bytes, habiendo en su interior tres números reales *IEEE-754* de 32 bits. Cada uno de ellos representando un eje de las posibles acciones. En este orden: la extensión, flexión y cocontracción.

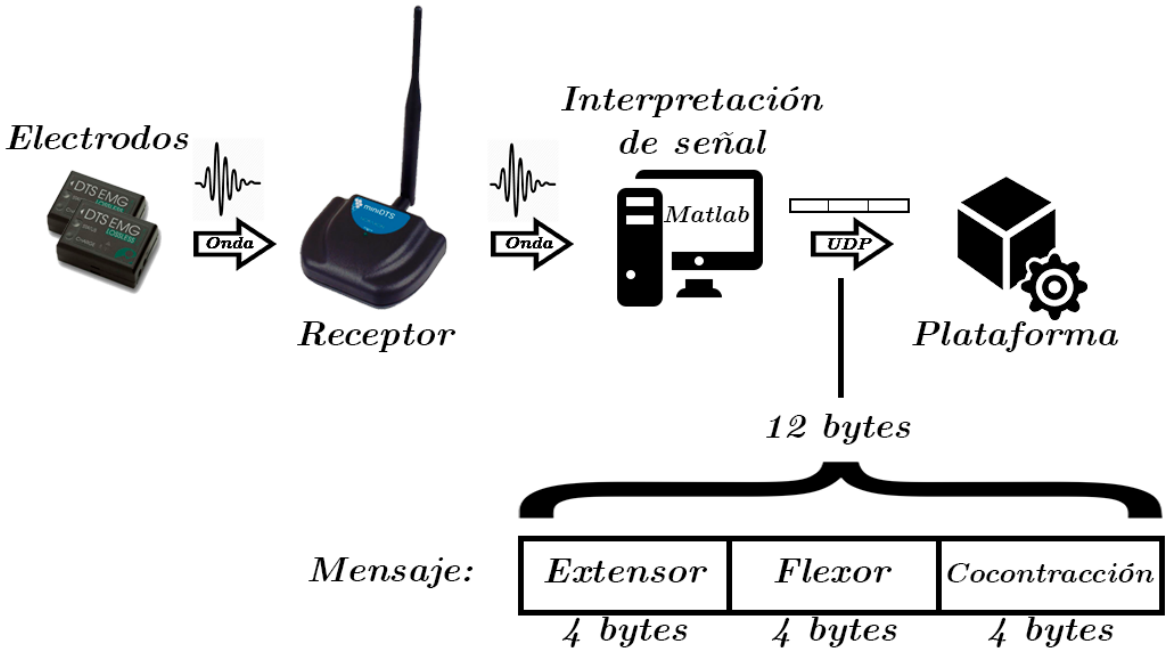


Figura 5.7: Arquitectura y protocolo de la transmisión de datos.

El intérprete de señales, en un inicio, se planteó que sería un algoritmo realizado en *Matlab*. Aunque, mientras se mantenga este protocolo de transmisión, se puede usar cualquier tipo de herramienta. Se escogió *Matlab* por su uso extendido en las comunidades científicas y su versatilidad. Incluso, para facilitar su uso, se realizó una función que se encarga de conectar y enviar los tres datos a la plataforma. Adicionalmente, no hay que preocuparse de enviar demasiados datos con *Matlab* y saturar la plataforma. La misma está preparada para leer únicamente el conjunto de tres reales más reciente. Se pueden encontrar más detalles en el manual de *Matlab* en el Anexo del apartado A.2.

5.1.3. Análisis y diseño de juegos

Antes de poder empezar a implementar, es necesario analizar los requerimientos de cada idea para diseñarlos. A la hora de diseñar se tiene que plantear el argumento del juego, las

metas con los diferentes niveles de intención, las mecánicas que lo compondrán, las características en relación al control y el desarrollo de la viabilidad con los diferentes tipos de control.

Todos los juegos se plantean con un objetivo en mente. Todos y cada uno de ellos deben servir a varios propósitos, ya sea una base de pruebas para un tipo de control, como una manera de entrenar ciertas acciones.

En este análisis y diseño se desarrollan las ideas de 4 juegos completos con todas sus mecánicas. Sin embargo, inicialmente se desarrollarán las características esenciales de los juegos. Lo que se conoce como una versión *MVP* o *Minimal Valuable Product*, no son juegos completos, pero tienen lo esencial para ser jugables. A medida que el tiempo e iteraciones del proyecto lo permitan, se agregarán las características necesarias para convertirlos en juegos completos. Inicialmente, se plantean 4 videojuegos de diferentes tipos y géneros para desarrollarlos, aunque solamente se realizan 2 y el resto se plantea para próximas iteraciones del proyecto. Todos los apartados de mejoras no se implementarán para la versión MVP de los juegos. Aún así, todas las características que se vayan a implementar se analizarán al principio de cada iteración.

El *Game Feel* de estos juegos se centra en el control a tiempo real y un espacio simulado. Sin embargo, el pulido, por la propia naturaleza del mismo, se tendrá que desarrollar en último lugar y, por ello, quitando algunas excepciones, no entraría dentro de la versión *MVP*.

Arkanoid

- **Descripción:** Eres una nave espacial que ataca a los ladrillos de una nave enemiga con una esfera de la cual solo tienes unas pocas cargas.[2] Si pierdes cada una de las esferas se acaba la partida.
- **Metas:**
  - **Largo plazo:** Llegar al máximo nivel destruyendo a la madre nodriza.
  - **Medio plazo:** Romper todos los bloques.
  - **Corto plazo:**
    - Conseguir mejoras.
    - Aumentar tu puntuación.
    - No perder la esfera.
- **Mecánicas *MVP*:**
  - Los bloques, cuando cualquier tipo de proyectil impacta contra ellos, se destruyen poco a poco, a no ser que sea un bloque indestructible. Cuando el proyectil es la esfera, ésta rebota.
  - Dependiendo en qué parte de la nave choque la esfera, el ángulo de la misma cambia. Se hace producto vectorial del movimiento inicial de la esfera y el punto de choque con la nave. Cuanto más cerca esté el choque de los extremos de la barra, provocará un ángulo de rebote más pronunciado.

[2] Basado en el argumento original del *Arkanoid* de la consola *NES*.

  - Se tienen 3 vidas al inicio pudiendo llegar a un máximo de 5 con mejoras. Pero si la esfera cae al fondo bajo, se pierde una vida.

- **Mecánicas a futuro:**
  - **Mejoras:**
    * **Láser** para romper bloques desde la nave.
    * **Alargar** la nave para que sea más fácil evitar perder la esfera.
    * **Capturar** la esfera cuando golpea con la nave.
    * **Duplicar** la cantidad de esferas en partida.
    * **Recuperar** una esfera perdida y, por lo tanto, una vida.
    * **Ralentizar** la velocidad de la esfera.
  - Solo se puede tener **Láser** o **Capturar**, pero nunca las dos mejoras al mismo tiempo.
  - La esfera aumenta de velocidad con cada bloque roto. Si se pierde una vida, ésta baja a la velocidad base.

- **Controles:**
  - **Características:**
    * Juego a tiempo real.
    * No requiere muchos reflejos hasta que aumente la velocidad de la esfera.
    * Se puede sincronizar el movimiento a los lados con el del brazo y apretar como la mejora del láser.
  - **Control binario:**
    * Impide movimientos muy rápidos y totalmente continuos. Habría que reducir considerablemente la velocidad de la esfera.
    * Fácil de desarrollar e intuitivo.
  - **Control proporcional:**
    * Se puede asociar los ejes del extensor y flexor con posiciones de la nave permitiendo movimientos rápidos.
  - **Control flanco de subida:**
    * Fatigaría mucho al usuario por los movimientos constantes al intentar reaccionar a la esfera.

Balance Journey

- **Descripción:** Eres un trapecista que quiere llegar lo más lejos posible en una barra interminable en el mar.

- **Metas:**
  - **Largo plazo:** Superar cualquier récord.

  - **Medio plazo:** Obtener recompensas de la rana.
  - **Corto plazo:** Avanzar lo más rápido y lejos posible evitando a los obstáculos.

- **Mecánicas *MVP*:**
  - Cuanto más equilibrado esté el personaje, más velocidad consigue.
  - Hay obstáculos que pueden tirar al personaje de la barra de equilibrio.
  - Si se está muy desequilibrado, la velocidad del jugador baja y, si se pierde completamente el equilibrio, cae al mar.

- **Mecánicas a futuro:**
  - **Mejoras:**
    * **Equilibrio** perfecto durante un tiempo limitado.
    * **Velocidad** aumentado temporalmente.
    * **Invencibilidad** momentánea.
    * **Multiplicador** de puntuación.
  - La cocontracción ayuda a estabilizarse pero solo se puede usar cada 30s.
  - Aparece la rana del *Qing Wa Jumper* y te da mejoras. Además, el color de la rana cambia en función de las mejoras que te traiga.

- **Controles:**
  - **Características:**
    * Juego a tiempo real.
    * Requiere constante control.
    * Puede fatigar mucho porque se necesita recalcular constantemente el movimiento.
  - **Control binario:**
    * Requiere movimientos percutantes constantes por la falta de precisión.
    * Fatiga mucho.
    * El control puede no responder correctamente a las intenciones del jugador.
  - **Control proporcional:**
    * En función de la colocación del brazo se puede responder al equilibrio.
    * Menor fatiga. No hace falta hacer movimientos percutantes.
    * El control puede no responder correctamente a las intenciones del jugador, pero lo puede percibir como la propia perdida de equilibrio.
  - **Control flanco de subida:**
    * Actúa solo en los movimientos percutantes de un instante.
    * La fatiga es mayor.
    * Puede no responder a la velocidad deseada del jugador. Se necesitarían ajustes *software* para que este control fuera viable.

Puzzle Scroll

- **Descripción:** Eres un explorador en una mazmorra cuya salida está bloqueada por diversos rompecabezas.
- **Metas:**
  - **Largo plazo:** Salir de la mazmorra.
  - **Medio plazo:** Conseguir todos los tesoros.
  - **Corto plazo:** Hacerte camino empujando bloques.
- **Mecánicas *MVP*:**
  - El movimiento es en retícula. Solo se puede estar en una, a no ser que se avance a otra retícula contigua.
  - Se pueden empujar y caer bloques móviles.
  - Escaleras que se recorren automáticamente al estar cerca de ellas y usar la cocontracción.
  - Se puede reiniciar el nivel en cualquier momento.
  - Si se cae el personaje en la lava, se perderá una vida.
  - Los bloques pueden estar en la lava para hacerte camino.
- **Mecánicas a futuro:**
  - Objetos recogibles que aumentan la puntuación.
  - Botón que genera un temblor y hace caer a los bloques sin gravedad que estén agrietados.
  - Plataformas móviles que permiten avanzar tanto en vertical como horizontal.
  - Hay un máximo de 10 vidas. Si el jugador pierde todas, vuelve al principio del juego. Se puede recuperar una vida por cada 10 tesoros que se recojan.
  - Puede haber enemigos.
  - Bloques que se destruyan al pasar por encima de ellos.
- **Controles:**
  - **Características:**
    - Juego pausado.
    - No requiere reflejos.
    - No fatiga con facilidad al jugador.
    - El movimiento puede ser por bloques.
  - **Control binario:**
    - Si el movimiento no fuera en retícula, sino a nivel de píxel, este control tendría sentido, pero el juego no se beneficia de ello.
    - Puede ser frustrante si no se empujan los bloques en el lugar exacto.

    - Puede fatigar al jugador.
  - **Control proporcional:**
    - Se podría hacer que se muevan lo bloques más rápido en función de la intensidad que se proporciona.
    - En algunos momentos el jugador puede estar más atento de moverse con precisión que del rompecabezas, añadiendo una limitación que cambia el enfoque del juego.
  - **Control flanco de subida:**
    - Se puede hacer un desplazamiento por retícula, permitiendo descansar entre cada movimiento.
    - Fatiga menos al solo necesitar un instante y poder descansar entre movimientos.
    - Intuitivo.

Qing Wa Jumper

- **Descripción:** Eres una rana que quiere volver a su casa y, para ello, debe saltar de apoyo en apoyo.
- **Metas:**
  - **Largo plazo:** Llegar a la casa de la rana.
  - **Medio plazo:** Conseguir la máxima puntuación.
  - **Corto plazo:** Saltar de apoyo en apoyo.
- **Mecánicas *MVP*:**
  - El salto es más grande en función de lo mucho que active el jugador la cocontracción.
  - Cuando se hace un salto perfecto(llegar al centro del siguiente apoyo) la puntuación sube más de lo normal. Cada salto perfecto encadenado multiplica la puntuación conseguida.
  - Diferentes modos y apoyos aleatorios para hacer cada partida única.
  - Al caer en el agua se pierde la partida y se empieza desde el principio.
- **Mecánicas a futuro:**
  - A partir de la parte más alta el jugador puede hacer un doble salto para solventar errores, aunque el uso del doble salto impide bonificación por salto perfecto.
  - Puede haber varios niveles de altura en los apoyos.
  - Los apoyos pueden ser móviles.
- **Controles:**
  - **Características:**

    - Juego pausado con elementos a tiempo real.
    - El doble salto permite corregir errores y fomenta usar más la cocontracción.
    - No fatiga al jugador, ya que puede descansar entre salto y salto.
    - El salto solamente se activa cuando el jugador quiere.
- **Control binario:**
    - Permite calcular lo potente que es el salto según una barra que indique la intensidad del mismo.
    - Para el salto doble puede ser confuso.
    - Fácil de desarrollar e intuitivo.
- **Control proporcional:**
    - Puede parecer arbitraria la fuerza que imprime la rana, habría que indicarle visualmente al usuario cuanto se está apretando y cuando suelte detectar la intensidad.
    - El doble salto puede ser complicado de detectar.
- **Control flanco de subida:**
    - Incapaz de usar las mecánicas de salto si no se usa con el binario en un control mixto.

Base de datos

La base de datos en una plataforma de estas características es imprescindible. Se necesita un lugar en el que almacenar masivamente todos los datos resultantes de cada sesión y mostrar estos datos para compararlos y poder estudiar a cada persona que juegue en la plataforma.

Al haber analizado en profundidad los diferentes juegos, todos tienen ciertos detalles en común. La puntuación, elemento con el que se podrá comprobar la mejoría del usuario a lo largo del tiempo. El control, cada juego puede usar diferentes tipos que pueden ser más efectivos en unos juegos u otros. Al final, los elementos principales son el usuario, los controles y el juego. La relación entre los tres, más el momento del día para almacenar el tiempo y la puntuación conseguida, provoca que se consiga una tabla de puntuaciones con persona, juego, tipo de control, tiempo y puntuación. De esta forma la Figura 5.8 presenta entonces la base de datos resultante en modelo entidad-relación de todos estos detalles en común.

5.1.4. Entrenamiento

La configuración del entrenamiento y cómo se entrena con la plataforma es un campo con muchas posibilidades. A medida que el jugador mejore en los juegos con controles sencillos, poco a poco se irían desbloqueando el resto de juegos con controles más complicados. De esta manera, el usuario sentiría una sensación de progresión que le llevaría a practicar más con la curiosidad y emoción de descubrir el resto de juegos.

Hacer un plan de entrenamiento con diversos niveles de todos los juegos. Por ejemplo, empezar con los primeros niveles de dos juegos que requieran reflejos, como *Arkanoid* o *Balance Journey*, y, para que el usuario no se fatigue, escoger, entonces, el primer nivel de un

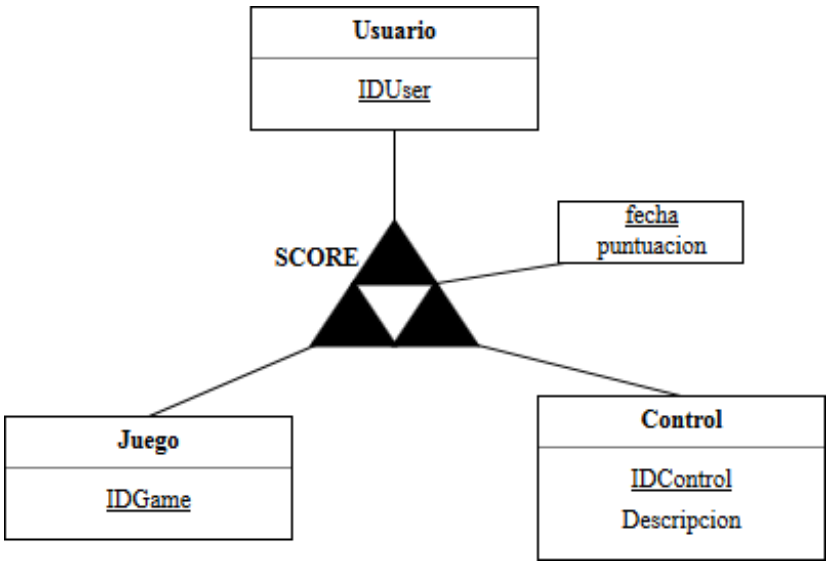


Figura 5.8: Modelo entidad-relación de la base de datos.

juego con ritmo pausado como *Puzzle Scroll* y entonces ir calentando para volver a los dos juegos con reflejos usando un juego con una mezcla de ritmo pausado y reflejos como *Qing Wa Jumper*. El plan de entrenamiento acabaría cuando se perdieran todas las vidas en todos los juegos. La variedad evitará que el usuario se acostumbre y, además, le haría más versátil. La cantidad de ciclos que pueda completar es una métrica dura que puede dar una medida general de su mejoría.

5.2. Iteración 1: Arkanoid MVP

El videojuego *Arkanoid*, basado en el también conocido *Breakout*, ha sido un videojuego muy clonado, mejorado e innovado a lo largo de los años. Tanto, que sus mecánicas están muy bien definidas. Tener un videojuego, el cual se ha perfeccionado a lo largo de los años, asegura una buena base para hacer las pruebas de los controles. Estando las mecánicas tan definidas, en el caso de que las métricas indiquen que los usuarios no asimilan bien el juego, no se entra en la duda de si la responsabilidad está en la mecánica o en el control. Por estas razones, se ha escogido un tipo de juego tan conocido.

Habiendo analizado cada elemento básico de un juego tipo *Arkanoid*, se vislumbra la serie de características que van a componer la versión *MVP*. Los elementos y comportamientos que se implementarán en esta iteración son:

- **Proyectil:** Empieza en la parte superior de la nave y hasta que no se active la cocontracción no empieza a moverse. Cuando choca con una superficie vertical, se invierte la velocidad en el eje de abscisas y, con una horizontal, el eje de ordenadas.
- **Bloque:** Al ser impactado por el proyectil, se destruye un poco cambiando de color, representando su entereza, y al estar totalmente destruido, desaparece. Si es un bloque indestructible, no cambia de color ni desaparece al chocar.

- **Nave:** Se mueve lateralmente e impide que el proyectil salga fuera de la pantalla perdiendo una vida. La velocidad del proyectil no se ve modificada, pero sí el ángulo de la misma. Cuanto más cerca de uno de los extremos esté el proyectil, se obtiene un ángulo más pronunciado en dirección contraria al mismo.

- **Muro:** Sirve para impedir que tanto la nave como el proyectil se salgan de los límites del juego.

- **Niveles:** Una matriz de bloques definidos con cada una de sus enterezas. Al destruir todos los bloques rompibles, se cargaría el siguiente nivel. Si no se ha ganado el juego.

- **Fondo inferior:** Cuando el proyectil se pierda en el fondo inferior, se perderá una vida y el proyectil volverá a la parte superior de la nave. Si se pierden todas las vidas, el juego acaba y, al activar la cocontracción, se reinicia desde el primer nivel.

- **Pantallas:** Cuando se pierda o gane mostrar la pantalla correspondiente en el juego, dando la posibilidad de reiniciar.

- **Sistema de puntuaciones:** Diseñar e implementar una manera de medir el rendimiento del usuario a través de puntos.

5.2.1. Características

Objectos de juego

Las entidades únicamente sirven para mostrar, no realizar, comportamientos a cada ciclo de procesamiento. Es por ello que se añade una derivación de las entidades llamada *Objeto de Juego* o *Game Object* con la habilidad de añadir comportamientos a cada tipo de objeto creado. Como, normalmente, los objetos de un juego, al actualizarse, pueden colisionar e interactuar entre ellos, la adición de este elemento provoca cambios en el ciclo de procesamiento como se puede observar en la Figura 5.9.

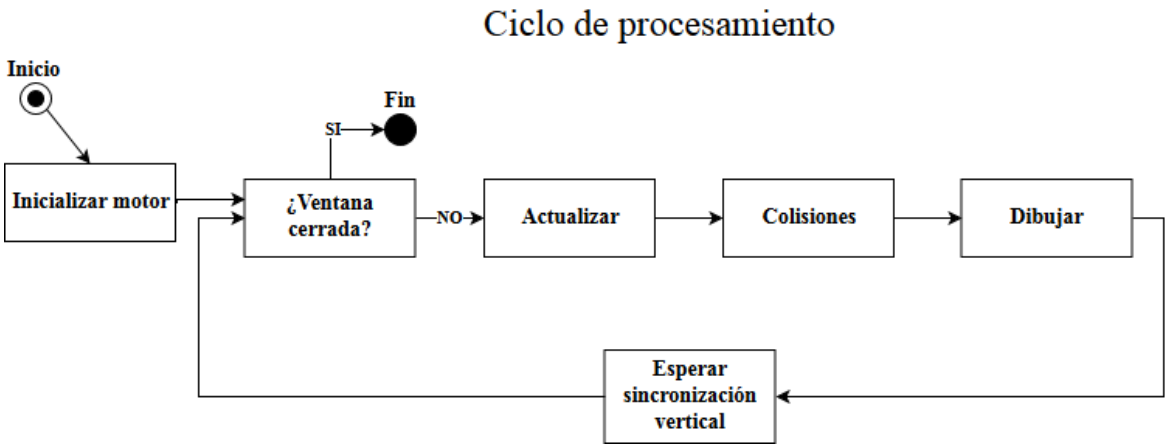


Figura 5.9: Ciclo de procesamiento del motor revisado.

Entonces, un *Game Object* no sólo puede actualizarse en función de su comportamiento, sino también, al colisionar con otros objetos, accionando algún proceso en concreto. Este podría ser el ejemplo del proyectil que, al actualizar, solo avanza según una velocidad en los dos ejes del espacio, pero, al colisionar, cambia el ángulo de la velocidad en función del objeto colisionado. Dos comportamientos diferentes para dos situaciones diferentes.

Sin embargo, para poder realizarlo, se debe definir qué provoca una colisión y con qué. En todos los juegos vamos a tener dos tipos de colisiones: entre rectángulos y círculo-rectángulo. Todas las entidades y sus derivados tienen un tamaño y posición que pueden ser definidos por un rectángulo. Debido a esto, en primer lugar, se realizarán las colisiones con rectángulos, formas comunes en todos los objetos, para, entonces, si alguno de ellos es un círculo, internamente se profundiza en la colisión círculo-rectángulo, porque éste es más costoso computacionalmente.

Al observar la Figura 5.10, el caso concreto de una colisión en un eje ocurre cuando la posición de mayor valor del primer objeto es a su vez mayor que la menor del segundo objeto, y viceversa. Como también se vislumbra en la Figura 5.11, para que exista la colisión, este mismo algoritmo se tiene que aplicar a los dos ejes, si en los dos ejes ha coincidido el caso, ha habido colisión.

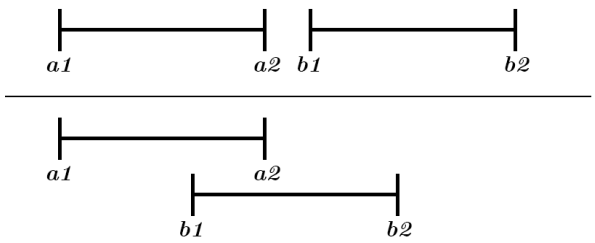


Figura 5.10: Colisiones en un eje.

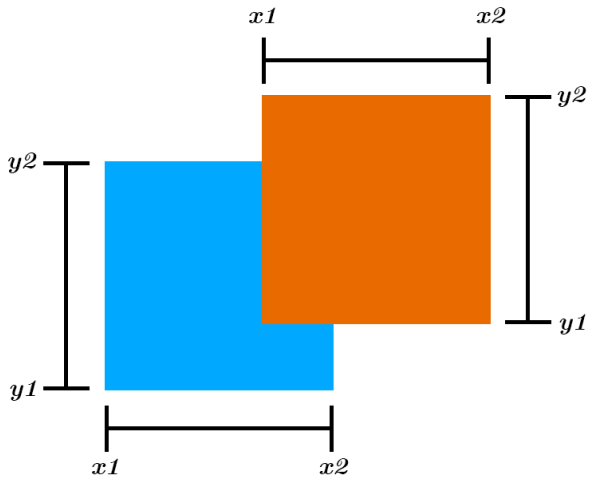


Figura 5.11: Colisión de dos cuadrados en los dos ejes.

Cuando se produce una colisión círculo-rectángulo el procesamiento se complica más. En

la Figura 5.12 tenemos una referencia del proceso. En primer lugar, se encuentra el punto P, el cual se halla juntando los centros de las dos figuras y escalándolo con el vector del centro a una esquina del rectángulo. Una vez con el punto P, se hace la diferencia con el centro de la circunferencia en los dos ejes. Si en esta diferencia alguno de los dos ejes es menor que el radio de la circunferencia, ha ocurrido una colisión.

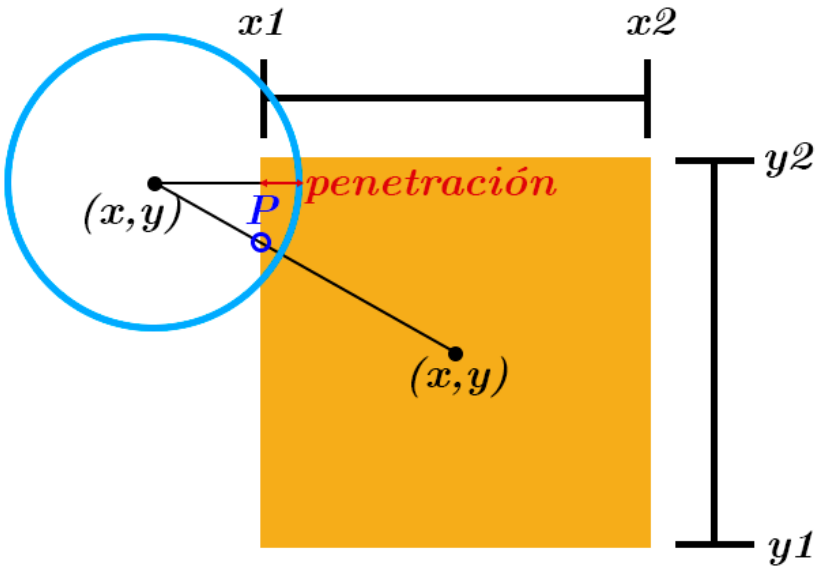


Figura 5.12: Colisión círculo-rectángulo.

Proyectil

Se ha detectado la colisión, pero el proyectil no puede funcionar únicamente con ello, ya que, como se describió previamente, en función de si colisiona en la parte vertical u horizontal se realizará una acción u otra. Para ello, cuando se realiza la diferencia entre el punto P y el centro del círculo, si la diferencia es mayor en el eje de abscisas, se detectará una colisión vertical. En caso contrario, horizontal. De esta manera, se cambia la velocidad en el eje adecuado en un choque.

Al colisionar con la nave, el comportamiento es ligeramente diferente. Se hace un porcentaje de lejanía desde el centro del proyectil respecto a los extremos de la nave y este porcentaje será lo que se aplique a la velocidad en el eje de abscisas dejando el resto del porcentaje para el eje de ordenadas. Aún así, se tiene limitado el porcentaje a 50%, para evitar un rebote eterno entre los muros verticales, si se colisiona justo en el extremo. Porque toda la velocidad iría a parar al eje de abscisas al ser un porcentaje muy grande.

Nivel

Un nivel se compone de una nave, un proyectil, muros y un conjunto de bloques. A nivel técnico, hacer que cada bloque individual este asociado a un nivel puede ser crítico para el rendimiento, por lo que todos los posibles bloques se precargarían en el motor y cada nivel

tendría asociado una matriz indicando la entereza de cada bloque. Si una posición de la matriz tiene una entereza nula, se toma como un bloque ya desaparecido. Por ahora, cada destrucción proporciona 100 puntos para añadir al sistema de puntuaciones.

El nivel puede requerir mucho ensayo y error para encontrar el tamaño adecuado. La anchura del nivel en su totalidad depende del tamaño de los bloques y de la cantidad que se vayan a colocar horizontalmente. De esta manera, se asegura la posibilidad de realizar cambios rápidos. Una vez comprobada la anchura con 5 bloques, se pudo observar que se debe ampliar a 10 bloques. Todo ello, porque, para generar una mayor simulación espacial, se requería más anchura, haciendo que el usuario pudiera mover la nave en un espacio más grande y manejar un número de bloques acorde a un juego tipo *Arkanoid*.

Cuando el conjunto de bloques destruibles del nivel baja a cero, se reinician todos los bloques para que sean invisibles y entonces se carga el siguiente nivel con las enterezas de cada bloque. Cuando ya no queden más niveles, se muestra la pantalla de victoria. Si se acciona la cocontracción, el juego vuelve a empezar. Como también se acciona en el caso de la derrota, cuando se pierden 3 vidas.

Cuando el proyectil llega al fondo inferior de la pantalla, debería perderse una vida. Aunque, para poder realizarlo, al motor se le ha agregado una característica llamada *Trigger* o disparador, que provoca una acción cuando colisiona con un objeto en concreto. Específicamente el proyectil. De forma que, el nivel contiene un objecto de juego adicional en el fondo inferior que al colisionar con el proyectil, activa la perdida de una vida.

Al ocurrir una derrota o la pérdida de una vida, el proyectil es privado de su velocidad y anclado a la nave hasta que se accione la cocontracción. Entonces, el juego transcurre de manera natural.

Salto entre *frames*

Al comprobar el buen funcionamiento del videojuego en diferentes tipos de máquinas, se ha comprobado que, en algunos casos, cuando no hay suficiente frecuencia de monitor, el proyectil no llega a chocar con los muros y sale disparado fuera de la pantalla. Después de depurar el código, se llegó a la conclusión de que ocurría debido a que la tasa de *frames* es variable y la velocidad del proyectil muy alta.

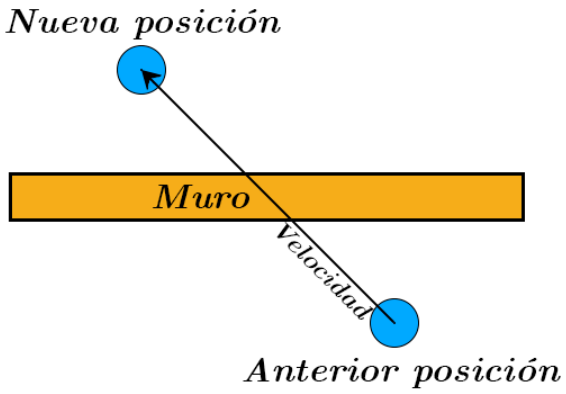


Figura 5.13: Salto entre *frames*.

El motor, para que funcione con tasas de *frames* variables, calcula el tiempo entre frames para, entonces, multiplicarlo a las velocidades. Así se consigue, no solo que siempre se ajusten los movimientos a cualquier tasa de *frames*, sino también, que las velocidades se representen en $\frac{unidades}{segundo}$. El problema suele aparecer en bajas tasas de *frames* y altas velocidades, ya que, en el momento de añadir una velocidad a la posición anterior, se puede evitar la colisión saltando objetos como se ve en la Figura 5.13.

La solución propuesta es realizar la etapa de actualización más veces en función de la tasa de *frames*. Cuanto menor sea la tasa, mayores serán los ciclos requeridos. Por ejemplo, si se requiere una frecuencia de actualización con 120 Hz y la tasa es 60 Hz, el periodo entre *frames* se divide entre 2 y se realizan dos ciclos de actualización. Otra posible solución sería, también, a cada *Game Object* que pueda dar problemas, aumentar sus ciclos de actualización en concreto. Aunque, se opta por la primera opción, al mantener la sincronía con el resto de objetos. Esta solución, provoca un cambio en el ciclo de procesamiento que se muestra en la Figura 5.14.

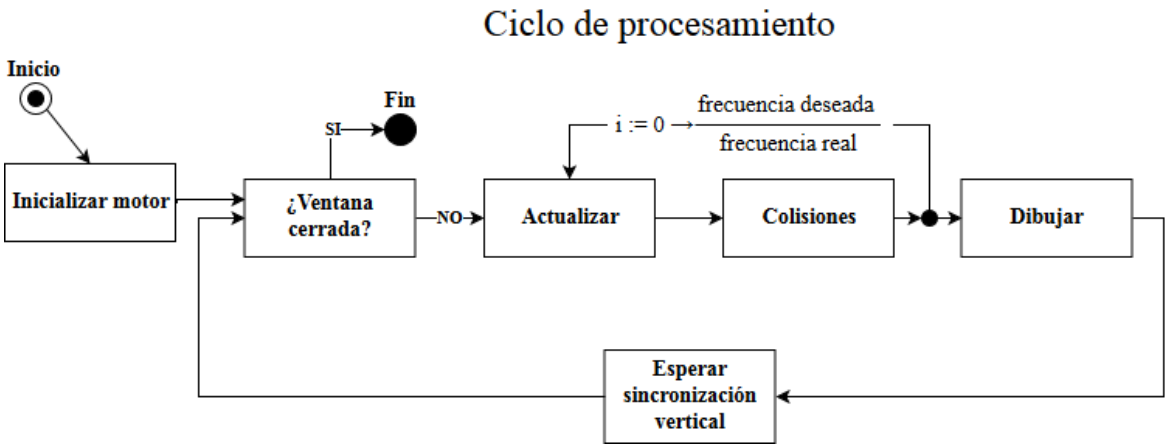


Figura 5.14: Ciclo de procesamiento del motor con ciclos de actualización.

Gestión de juegos

Esta plataforma tiene que permitir muchos juegos, por lo que no se pueden cargar los recursos de todos los juegos al inicio del programa. Con solo dos se puede permitir, pero, a medida que se añadan más juegos y aún más sofisticados, solo la carga y descarga de datos podrían suponer minutos. Para evitar eso, se desarrolla una pantalla de selección con todos los juegos y que requiera pocos recursos. Cuando se seleccione un juego, se procederá eliminar la pantalla de selección y entonces a inicializar todos los recursos necesarios para el juego elegido y, al volver desde el juego a la selección, se producirá el proceso inverso.

En pos de poner en marcha este sistema, se desarrolla un gestor de juegos que cargue y descargue los recursos de un juego al otro, evitando este problema.

5.2.2. Mejoras

Las posibles mejoras del sistema dan perspectiva al proyecto y ayudan a preparar la moduralidad del mismo. Es debido a esto que se plantean en este apartado:

- La adición de sonidos que le agreguen un pulido extra al sistema y enganchen al jugador.
- Añadir la puntuación más alta del usuario para que siempre tenga presente superar sus límites y, cuando lo haga, se sienta realizado.
- Una pantalla adicional en la derrota o victoria que recalque la superación de la puntuación más alta.
- Cuando la nave colisione con los muros añadir un pequeño rebote y sonido para dar más simulación espacial y pulido.
- Se podría mejorar el sistema de colisiones con el proyectil si se realiza trazado de rayos.
- La dificultad debería ir subiendo a medida que el jugador supere niveles. Aumentar la velocidad del proyectil o conjuntos de bloques más intrincados pueden ser una buena manera de conseguirlo.
- Cuando el proyectil consiga golpes encadenados a los bloques sin tocar la nave, añadir multiplicador a la puntuación. Ello fomentará que los primeros niveles no se hagan aburridos y sean oportunidades de conseguir mucha puntuación fácilmente, ya que tendrán menor dificultad.
- Añadir una estela al proyectil indicando la dirección de su velocidad en pos del pulido.
- Por supuesto, añadir todas las mecánicas restantes.

5.3. Iteración 2: Balance Journey MVP

Balance Journey es un videojuego original que consiste en mantener a un corredor equilibrado en una barra a lo largo de un mar interminable, mientras se esquivan los tiburones, enemigos que solamente podrán ser evitados aminorando la velocidad, que implica perder el equilibrio. Todos los juegos se plantean solamente con las 3 posibles acciones y este caso no es una excepción. Además, este juego se diseñó, exclusivamente, con el control proporcional en mente.

Los datos que se reciben de este control dependen de la activación muscular, pero los músculos no pueden estar constantemente activados al máximo. Se fatigan, baja el porcentaje de fuerza medio y la señal interpretada puede variar con mayor facilidad. Si se tradujeran los movimientos directamente a un juego como el tipo *Arkanoid*, se notaría, inicialmente, irregular y difícil de controlar, provocando frustración en el jugador y extrayéndolo del estado de inmersión constantemente. Obviamente, cuanto más mejore la interpretación proporcional de la señal, menores serán estos problemas, teniendo en cuenta elementos como la fatiga, el porcentaje de fuerza variable, etc. Sin embargo, se necesita una plataforma en la que comprobar el funcionamiento del control para poder mejorarlo.

Con *Balance Journey*, se propone una solución meta-jugable. La gran mayoría de personas ha perdido el equilibrio alguna vez en su vida o, al menos, tiene un concepto claro de ello. Al hacer que el personaje se esté constantemente equilibrando, permite que el jugador no se frustre, ya que, cualquier derivación provocada por la interpretación de las señales se achacaría a lo difícil que es para el personaje equilibrarse al mismo tiempo que correr. El control no ha cambiado, pero el contexto sí.

La versión *MVP* de *Balance Journey* establecerá la mayoría de las mecánicas principales del juego, porque, a diferencia del tipo *Arkanoid*, las mecánicas son más escuetas y requieren menor tiempo de desarrollo. Por lo tanto, las características del juego que se implementarán en esta iteración son:

- **Sistema de equilibrio:** Algoritmo que controlará el equilibrio del personaje y cuándo el jugador se caerá al mar.
- **Corredor:** Elemento estático en pantalla que representa el avatar del jugador. Cambiará su ángulo dependiendo del sistema de equilibrio.
- **Mundo:** Un mar eterno que desciende por la pantalla generando una ilusión de movimiento. Lo rápido que descienda dependerá de lo mucho que esté equilibrado el corredor.
- **Enemigos:** Tiburones que descienden en relación al mundo, porque, en teoría están dentro del mar. Además, se mueven hacia la barra de equilibrio para intentar atacar al jugador.

5.3.1. Características

Sistema de equilibrio

El sistema de equilibrio balanceo irá acompañado de una flecha que indica el equilibro del corredor y una barra que muestra los grados que debe tener la flecha antes de perder el equilibrio por alguno de los dos lados, como se ve en el ejemplo de la Figura 5.15. Cuando se llegan o superan estos grados de caída, se pierde la partida y se mostrará una pantalla de derrota.

La flecha, en un inicio, tiene movimiento continuo hacia uno de los extremos, representando la perdida de equilibrio. La dirección del movimiento de la flecha puede ser invertida realizando la acción muscular del movimiento contrario. Por ejemplo, si se desea cambiar el equilibrio hacia la derecha, con el brazo izquierdo se deberá activar el flexor. Pero realizar un cambio de dirección implica que la velocidad será modificada en proporción a lo cerca que se esté de la caída, es decir, cuanto más cerca esté la flecha del ángulo de caída, mayor será la velocidad hacia el otro extremo. Con este diseño se desarrolla más la simulación espacial del *Game Feel*, implementando un sistema equiparable a la vida real.

Este algoritmo puede provocar que los usuarios se mantengan constantemente en equilibrio, es por ello que existen los enemigos, como se analizará posteriormente.

El corredor, situado por encima del eje, en un inicio, se le asoció el mismo ángulo de la flecha, pero tanta graduación no resultaba realista, por lo que se le aplicó al corredor un 20% del ángulo de la flecha. Esta inclinación aunque solo es un efecto visual, lo hace mucho más natural.

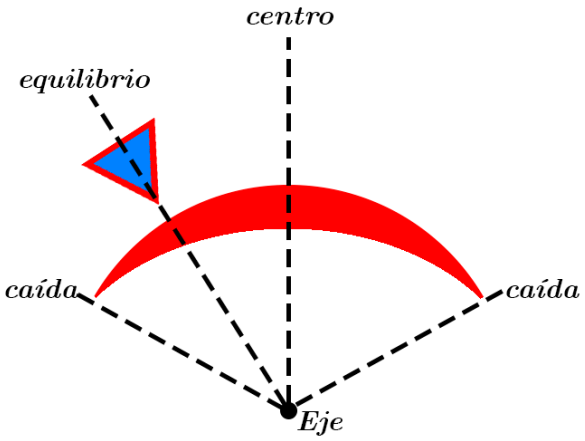


Figura 5.15: Componentes del sistema de equilibrio.

Enemigos y las animaciones

Los enemigos son tiburones que suponen un obstáculo para el corredor. Provocando que el jugador, para evitarlos, debe arriesgarse a perder el equilibrio debido a que, la perdida del mismo, implica reducir la velocidad de avance del corredor por el mar. Si quiere evitar a los tiburones, debe cambiar su velocidad y si quiere cambiar su velocidad, debe arriesgarse a perder el equilibrio. Este conjunto de decisiones genera una mecánica de riesgo-recompensa.

Los tiburones tienen un patrón definido, como se vislumbra en la Figura 5.16. En primer lugar, salen desde el lado izquierdo de la pantalla, escondidos y preparados para atacar en cuanto se acerquen a la barra. En segundo lugar, cuando están cerca de la barra, saltan para atacar al corredor, indiferentemente de si se encuentra en ese sitio o no. Y, por último, caen al mar y se dirigen al lado derecho de la pantalla hasta desaparecer.

Figura 5.16: Patrón de movimiento.

Este patrón implica una serie de pasos, puntos, movimientos y velocidades que, en su conjunto, se denomina animación. Las animaciones no son una característica del motor. Debido a ello, en esta iteración también se desarrollan, y servirán para agregar más pulido posteriormente.

Una animación se compone de un conjunto de tramos que, a su vez, están compuestos por un punto de destino, una velocidad y tipo de tramo, pudiendo éste ser lineal o curvo. La especial complicación con las animaciones son los *frames* variables, que ya se han tratado previamente. Al permitir esta variabilidad, pueden ocurrir instantes en los que se salten tramos si la velocidad es suficientemente grande, como se ejemplifica en las Figuras 5.17 y 5.18.

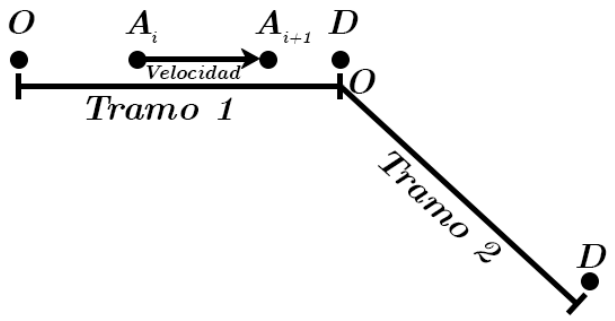


Figura 5.17: Proceso normal de una animación.

Para poder recalcular la posición cuando se sobrepasa el punto de destino, en primer lugar, se calculan los vectores $\overline{OD}$, origen a destino, y $\overline{A_{i+1}D}$, siguiente posición a destino. Si las direcciones de los dos vectores no coinciden, significa que es el caso de la Figura 5.17, pero si coinciden las direcciones, es el de la Figura 5.18. Si es el primer caso, el algoritmo termina sin mayor complicación, pero, si es el segundo, se usa la distancia escalar en el vector $\overline{A_{i+1}D}$ y se aplica en la dirección del segundo tramo, escalandola a la velocidad del mismo. Si aún así, se sigue superando al punto de destino, se repite el proceso. Aunque, en el caso de que no queden más tramos, se asigna la posición de destino del último tramo, finalizando el algoritmo y, por lo tanto, la animación.

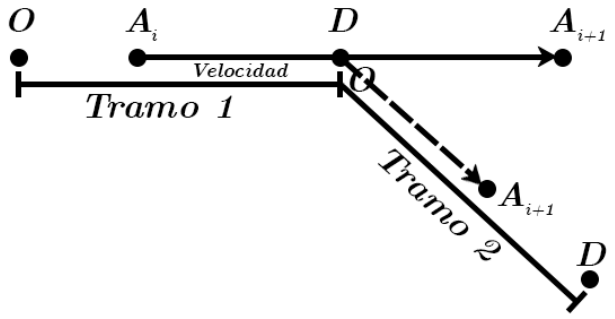


Figura 5.18: Proceso de recalculación de una animación.

Todo ello en una serie de tramos lineales, pero los curvos tienen una capa extra. Cuando se añade a la animación un tramo curvo, se generan interpolaciones de tramos lineales formando una semicircunferencia. Normalmente, los suficientes como para que en movimiento, la diferencia con una circunferencia real sea nimia. Un ejemplo con pocos intervalos se encuentra en la Figura 5.19.

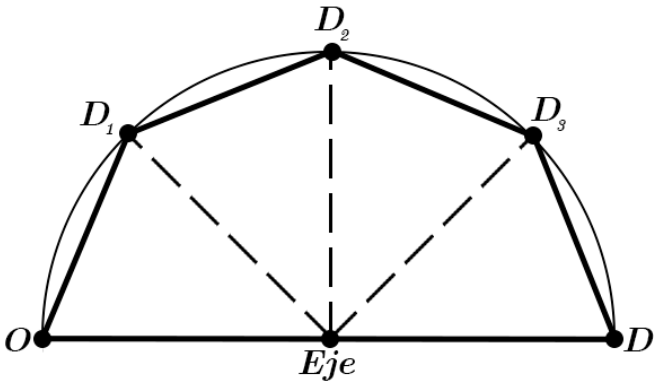


Figura 5.19: Cálculo de un tramo curvo interpolando tramos lineales.

Mundo

El mundo está compuesto de dos planos adyacentes, uno encima del otro, cuya velocidad descendente varía en función del porcentaje de estabilidad que proporcione el sistema de equilibrio. Cuando un plano llega al fondo inferior de la pantalla, se coloca encima del otro, creando la ilusión de movimiento interminable. Y, además, para que convenza más la simulación del espacio, los bordes de sus texturas empiezan de la misma manera que acaban, haciendo imposible diferenciar dónde empieza uno y termina el otro.

Los tiburones, al estar dentro del mar, también se tienen que mover al mismo ritmo que el mundo por lo que se anclan al mismo. En principio, habrá tres tiburones que empezarán en posiciones separadas entre sí y, al desaparecer de pantalla o terminar su patrón de animación, se volverán a colocar en la misma posición inicial. Sin embargo, esto hace el juego muy repetitivo, porque aparecen siempre de la misma manera y en las mismas posiciones. Para evitar esto, la velocidad inicial del primer paso de su patrón se le agrega un factor aleatorio, haciendo que cada partida sea diferente y proponiendo un ciclo de juego que no resulte tedioso después de muchas partidas.

Por último, este mundo, cada vez que desciende. indica cuantas unidades lo ha hecho, para añadir puntos que escalan con estas mismas unidades al sistema de puntuaciones. Cuanto más lejos llegue el usuario, mayor será su puntuación, motivando al jugador a superarse y, debido a ello, haciendo referencia a una de las metas planteadas en el diseño.

Hitbox

La textura de una persona equilibrándose tiene muchos espacios vacíos en un plano, por lo que, al colisionar con partes de la textura sin fondo con el plano de un tiburón, el jugador puede creer que no ha sido atacado por él y generar frustración. Para evitar este problema, se crea, a través de un *Trigger*, lo que se conoce como una *hitbox*, la zona generalmente invisible hecha específicamente para colisionar. Este disparador provocará que la partida se pierda cuando choque con el tiburón. Será más pequeña que el avatar del jugador, ocupando únicamente el cuerpo del mismo, lo que soluciona este problema. Aunque, lo mismo se realiza con el tiburón, al tener también, tantos espacios vacíos.

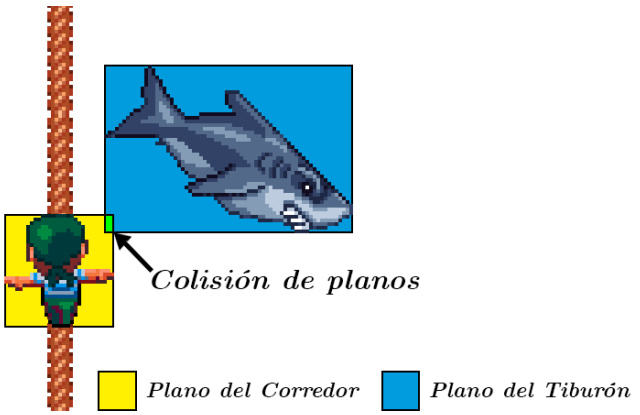


Figura 5.20: Situación visiblemente injusta al colisionar por planos cuando no coinciden con los sprites.

Derrota

Aprovechando que el motor tiene añadida la característica de las animaciones. Se le añadirá al personaje una animación de caída al mar cuando sea atacado por un tiburón, agregando al juego un toque de pulido. Adicionalmente, no solo se activará la animación de caída, sino también la pantalla de derrota, que solo permitirá reiniciar el juego cuando la animación de caída haya terminado. Se debería hacer la pantalla de victoria, pero, debido a que es un juego eterno, no tiene pantalla de victoria. Solo se consiguen puntuaciones mejores.

Pruebas

Al comprobar las reacciones de diferentes personas con el juego, se descubrió un comportamiento común: el control no era intuitivo. Estas personas necesitaron que se les explicara el juego para poder, si quiera, superar los primeros 6 tiburones. La conclusión que se extrajo es que existe una dificultad inicial provocada por no entender el juego y porque las distancias entre los tiburones son pequeñas, por lo que, estos espacios entre ellos se hicieron más grandes.

Ya realizadas las pruebas y unos cambios puntuales para arreglar leves errores de programación, se prepara la liberación de recursos al cambiar de juego y se añade el sistema al gestor de juegos, dando por acabada la iteración, para, entonces, analizar las posibles mejoras.

5.3.2. Mejoras

Las posibles mejoras extraídas de *Balance Journey* en esta iteración son:

- La adición de sonidos que le agreguen un pulido extra al sistema y enganchen al jugador.
- Añadir la puntuación más alta del usuario para que siempre tenga presente superar sus límites y, cuando lo logre, se sienta realizado.
- Una pantalla adicional en la derrota que recalque la superación de la puntuación más alta.

- Valorar la posibilidad de un tutorial para dejar clara las mecánicas del juego. Existen dos enfoques: o cuadros de texto explicando cada aspecto o introducir las mecánicas poco a poco de manera más natural. En este último caso, un ejemplo sería que, al principio, solo se aprenda a equilibrarse, introducir a un enemigo justo después y, por último, entrar ya en el juego real. Aún así, se puede realizar una combinación de ambas.
- A medida que crezca la puntuación, aumentar la velocidad base del jugador. Este cambio, que parece inocuo, le forzará a arriesgarse cada vez más, subiendo progresivamente la dificultad y evitando que el juego deje de ser un reto una vez domine las mecánicas.
- Por supuesto, añadir todas las mecánicas restantes.

5.4. Iteración 3: Selección de juegos e integración base de datos

5.4.1. Pantalla de selección

La pantalla de selección presenta todos los juegos que se hayan desarrollado, y cuando se escoja uno, cambiar los recursos de la pantalla al videojuego en cuestión. Cada juego tendrá un botón con imágenes descriptivas que, al pulsarlo, activará el cambio. Pero, además, cada juego se prepara para ser capaz de liberar los recursos en cualquier instante del mismo. Para poder hacer este cambio dinámico, el ciclo de procesamiento es modificado, como se observa en la Figura 5.21.

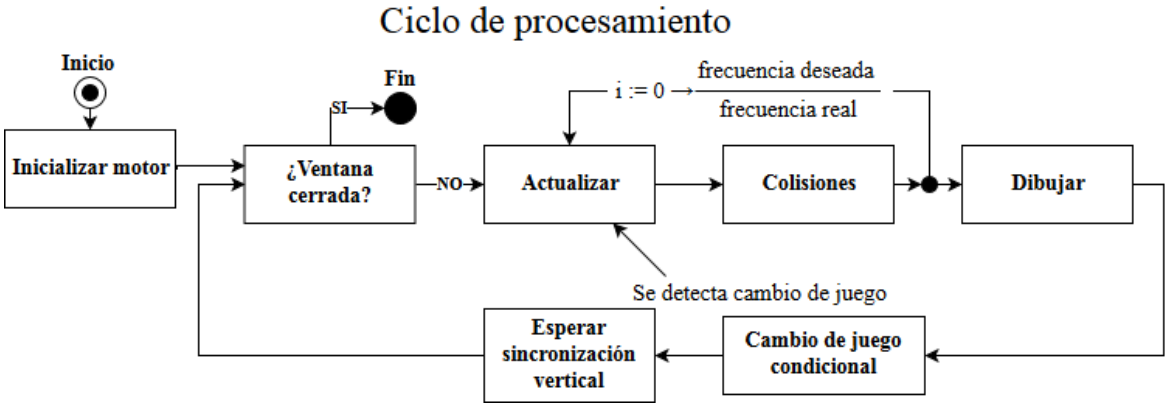


Figura 5.21: Ciclo de procesamiento del motor con apuntes de cambio dinámico.

Adicionalmente, dentro de todos los juegos cargados, siempre hay un botón en la esquina superior izquierda que permite volver a la pantalla de selección en cualquier momento.

5.4.2. Integración de la base de datos

La base de datos, previamente diseñada en la Figura 5.8, se traduce al esquema relacional en la Tabla 5.1. Aunque la traducción es solo un parte del trabajo. Se necesita integrar las herramientas de lectura y escritura de bases de datos para después traducir este esquema a código. La herramienta elegida es *SQLite3*, la cual, además de ser gratuita, tiene todo lo necesario y se puede integrar con el motor, ya que, pertenece al mismo lenguaje.

USUARIO (IDUSER)	**JUEGO** (IDGAME)	**CONTROL** (IDCONTROL, DESCRIPCION)
CP(IDUSER)	CP(IDGAME)	CP(IDCONTROL)
		VNN(DESCRIPCION)

SCORE (IDUSER,IDGAME,IDCONTROL,FECHA,PUNTUACION)
CP(IDUSER, IDGAME, IDCONTROL, FECHA)
CAj.(IDUSER) → USUARIO
CAj.(IDGAME) → JUEGO
CAj.(IDCONTROL) → CONTROL
VNN(PUNTUACION)

Tabla 5.1: Esquema relacional de la Base de Datos.

Para traducir este esquema a código se emplea el lenguaje usado por la herramienta, *mySQL*. Con éste, se realiza un *script* que crea la base de datos local con una serie de valores por defecto y, si ya existe, reiniciarla con esos mismos datos por defecto. Algunos de estos valores son el usuario llamado *ADMIN*, todos los tipos de control con sus descripciones y todos los juegos existentes.

Para dar el control de la base de datos local al usuario, hay una nueva ventana de opciones, la cual tiene datos para depurar y un botón que creará o sobrescribirá la base de datos ejecutando el *script*. Si la base de datos no existe, es intrínsecamente necesario que este botón sea pulsado antes de escoger un videojuego.

Ahora que la integración a nivel de código está terminada, se necesita pedir un nombre de usuario cada vez que se vaya a jugar. Para ello, se agrega un campo de texto en la pantalla de selección en el que el usuario debe introducir su nombre, apodo o lo que prefiera. Por defecto, en este campo se tiene el nombre *ADMIN* y solamente se puede cambiar desde la pantalla de selección. Por lo tanto, si se está en un juego, se deberá retornar a esta pantalla. Si el usuario no existe, se creará automáticamente en la base de datos sin mayores percances, aunque en el futuro, puede que esto cambie.

Este campo de usuario se almacenará siempre que un juego sea cargado, para tener todos los datos a introducir en la tabla *SCORE*: el nombre de usuario, el juego, tipo de control y, por último, la puntuación, introduciéndose en la base de datos en el mismo instante que el jugador pierda o gane. La fecha, por otra parte, se inicializa automáticamente con el tiempo exacto en el que se introducen el resto de datos.

Una mejora que se planteó en anteriores iteraciones fue el de introducir la máxima puntuación del usuario. Con la base de datos integrada, esto ya puede ser una realidad, mostrando la puntuación máxima al mismo nivel que la actual. Si no existe para ese usuario simplemente se mostrará un 0.

En el panel de control antes mencionado, se añade un botón que permite la posibilidad de ver la tabla *SCORE* a tiempo real en el propio programa y ordenada descendentemente por puntuación. Y, como última característica, para evitar que se pulse accidentalmente el botón de reinicio de la base de datos, se ha creado una ventana emergente que avisa al usuario de la eliminación de todos los datos almacenados y preguntando si realmente desea hacerlo.

5.4.3. Mejoras

Por último, se realiza el reiterado análisis de todos los sistemas creados al final de esta iteración y se valoran cuáles serían las posibles mejoras al mismo. Entonces, el progreso de esta plataforma se vería beneficiado por:

- **Ampliación de la base de datos:** El modelo actual puede variar con facilidad. Al ser cuatro tablas relacionadas entre sí, añadir un campo o quitar alguno no debe suponer un problema. Sin embargo, en el futuro se puede querer almacenar muchos más tipos de datos, lo que daría mayor riqueza a la base de datos, quitandole la capacidad de variar con tanta facilidad. Esta riqueza podría implicar nuevas tablas como **Patología**, que indicaría las discapacidades de cada usuario, **Dificultad**, que almacenaría los diferentes niveles de cada juego y **Dispositivos**, que guardarían la serie de máquinas donde los usuarios juegan. Esta ampliación supondría la base de datos del diagrama entidad-relación de la Figura 5.22.

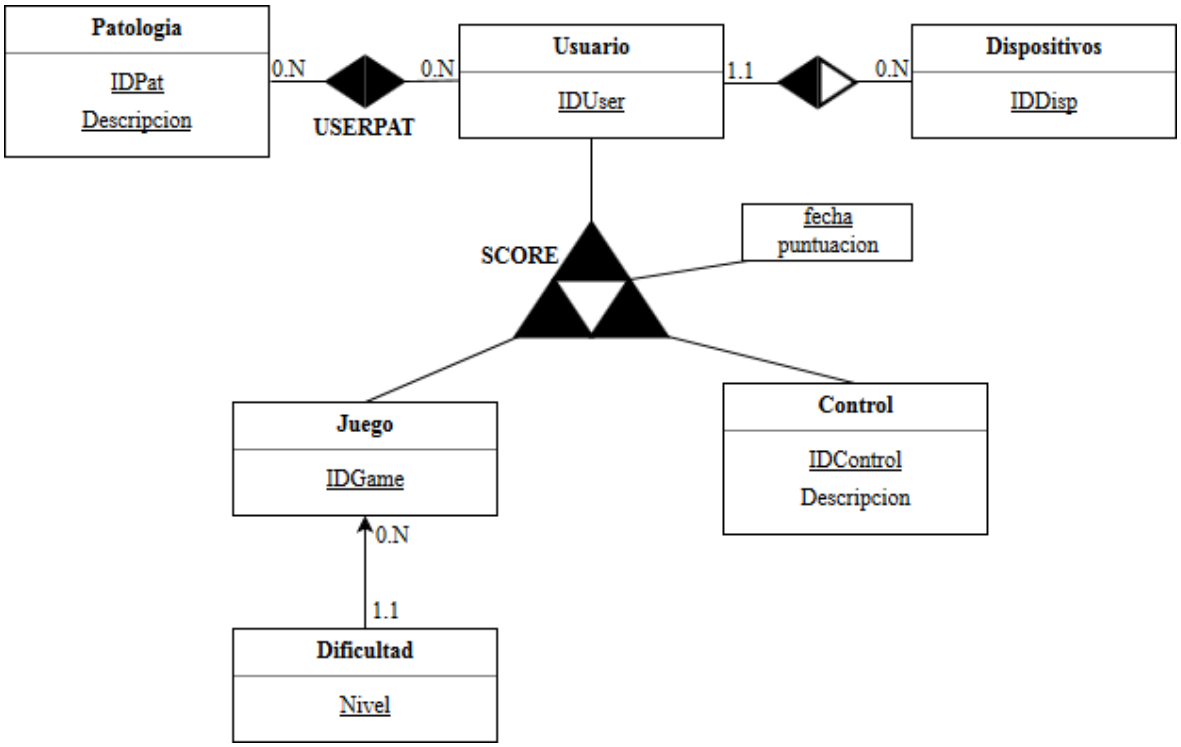


Figura 5.22: Modelo entidad-relación de la base de datos ampliada.

- **Base de datos no local:** Centralizarla a un servidor de manera que se pudiera tener la plataforma en cualquier máquina sin tener ningún dato dependiente.

- **Recordar usuario:** Tener en la plataforma un registro que mantenga el último usuario. De manera que, a pesar de cerrar el programa y reiniciarlo, su nombre se mantendría., facilitando el proceso de ejecutar la plataforma repetidas veces.

- **Seguridad:** Sistema de contraseñas que solamente permitan a los usuarios registrados acceder a la plataforma. Este sistema solamente sería contemplado si las anteriores dos mejoras fueran implementadas.
- **Navegador de base de datos:** Aumentar el alcance de muestra no solo a la tabla *SCORE*, sino a todas, y añadir la capacidad de realizar consultas personalizadas desde el propio programa.
- **Control de menús:** Actualmente, todos los botones y menús se controlan a través del ratón. Pero la plataforma sería más intuitiva si, además, se permitiera controlar todos los menús con las señales mioeléctricas.
- **Pulido visual:** Por ahora, en todo el programa hay imágenes por defecto y de prueba para cada elemento, fácilmente intercambiables. Uno de los principales elementos que se necesitan para que la plataforma sea más atractiva es el pulido visual. Un estilo definido en cada juego y menús que atraigan y enganchen a los jugadores.

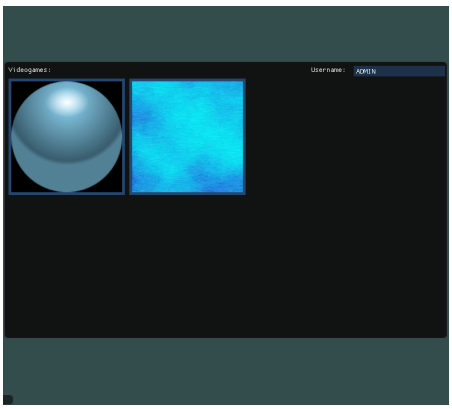
5.5. Iteración 4: Pulido visual y métricas

5.5.1. Pulido visual

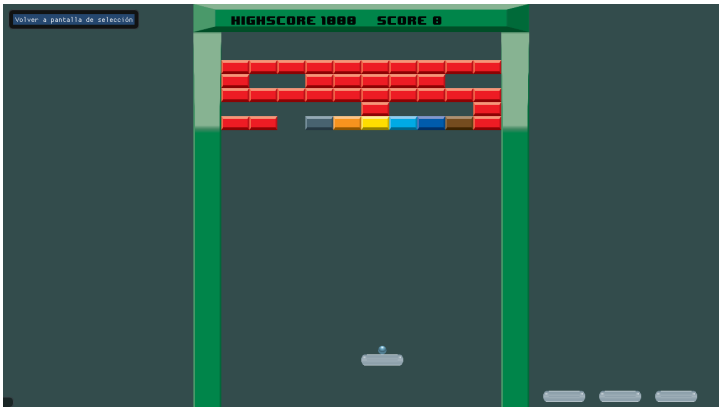
El pulido visual puede cambiar totalmente la sensación de juego, provocando el cambio de un videojuego inocuo a uno apasionante y rico en detalles. Si se consigue un buen pulido, el mundo convence más al jugador y genera más inmersión sin necesidad de cambiar ninguna lógica de juego.

Cambio visual

En un inicio, cada juego tenía una serie de imágenes por defecto, cuya única función era probar la lógica de juego. Sin embargo, estas imágenes reducen la calidad del producto, porque la impresión general del usuario será negativa, independientemente de lo bien que se hayan desarrollado las mecánicas. El ejemplo de estas imágenes por defecto en cada pantalla, se encuentra en la Figura 5.23.



(a) Pantalla de selección.



(b) *Arkanoid*

(c) (*Balance Journey*

Figura 5.23: Imágenes por defecto en las pantallas principales.

Como se puede observar, la plataforma, visualmente, no invita a jugar cada uno de los juegos. Realmente, da la sensación de ser más una prueba que videojuegos con sus mecánicas completas. En este apartado, se cambiarán cada una de estas imágenes por defecto para conseguir ese pulido visual.

En primer lugar, se cambian las imágenes del juego tipo *Arkanoid*, como puede observarse en la Figura 5.24. Pero, en el juego, las resoluciones y tamaños son diferentes. Por ello, se ha cambiado el tamaño de los *Game Objects*.

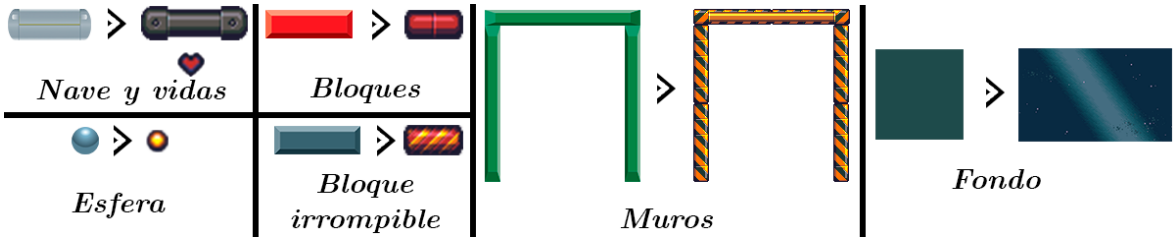


Figura 5.24: Gráficos totalmente renovados para el juego tipo *Arkanoid*. Autor prerenovación: ImagineLabs. Autor renovación de assets: Buch `https://opengameart.org/users/buch`. Autor renovación del fondo: Ansimuz `http://ansimuz.com/sites/`

La vidas, en el juego tipo *Arkanoid*, antes se representaban con repeticiones de la propia nave. Sin embargo, con los corazones queda más claro. El bloque irrompible y el muro comparten patrón para hacer entender al jugador con mayor facilidad que este tipo de bloque tiene las mismas características que el muro, es decir, son irrompibles. Además, el muro tiene una zona concreta en la que colocar la puntuación actual y máxima. El fondo plano se ha sustituido por uno que evoque al espacio para hacer referencia a la temática del juego.

Balance Journey también debe ser pulido visualmente, por lo que se cambia a una estética *pixel art*, haciendo al juego más atractivo, como se puede observar en la Figura 5.25.

Figura 5.25: Gráficos totalmente renovados para *Balance Journey*. Autor del fondo prerenovación: Aswin `https://opengameart.org/users/aswin909`. Autor de toda la renovación: Eblem Ramírez González

Al personaje principal se le dio una renovación total. Dándole un toque pirata, ayuda a que el contexto del juego se entienda mejor a primera vista. El tiburón consiguió una nuevas imágenes de salto y aleta, más características y pulidas. Además, el arco también se pule y contrasta con el mar azul de diferentes profundidades y con olas que le dan más riqueza. En lugar de una barra, ahora se usa una cuerda que introduce mejor al jugador el concepto de equilibrarse en el mar.

Se les proporcionó las nuevas versiones de los juegos a los mismos jugadores que probaron a la versión sin pulido. Todo ello para comprobar qué sensaciones provocaban en los usuario tras este pulido. Los jugadores coincidieron en una diferencia abismal en el control, a pesar de no haber cambiado nada de las mecánicas o lógica de juego. Ellos se sentían más inmersos en los juegos y el atractivo visual de los mismos les proporcionaba mayor concentración, pudiendo disfrutar del apartado visual mientras jugaban.

Por último, la pantalla de selección también fue renovada con estética de ventanas diferentes y el añadido del nombre de la plataforma en *Hànzì*, como se ve en la Figura 5.26.

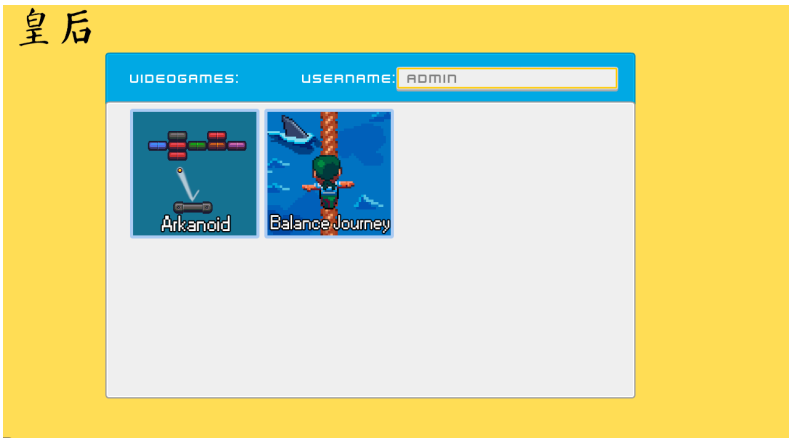


Figura 5.26: Pantalla de selección con los dos juegos y pulido visual.

Animación por frames

El juego *Balance Journey*, a pesar de tener ahora un pulido visual, se notaba poco fluido y muy estático. Un problema que no sufre el otro juego al solamente moverse una esfera y una barra/nave. Pero entonces, se obtuvo la posibilidad de realizar animaciones por frames, es decir, añadir diversas imágenes y cambiarlas a cada frame para generar la ilusión de movimiento. Ello evitaría que el juego fuera tan estático.

Aunque el motor no tiene la característica de procesar animaciones por frames, no es algo intrínsecamente difícil de implementar, mientras se tengan un conjunto de frames ordenados y un índice que indique cuál se muestra en pantalla. Cada vez que pase un determinado tiempo, se cambiaría el índice al siguiente frame. Cuando no queden más frames, si el desarrollador lo permite, se puede volver al principio del conjunto de frames, creando un bucle.

De esta manera, se puede proporcionar al personaje una animación caminando y, al mismo tiempo, otra inclinándose con lo que muestra el equilibrio, dándole aún más personalidad a este personaje y dinamizándolo. Al hacerlo por frames, el corredor ya no necesitaría asociar su ángulo con el de la flecha. En este caso, se asociarían las animaciones de inclinación pertinentes al ángulo de la fecha.

Las olas ahora pueden generar una animación, haciendo el mar mucho más interesante de ver, mientras que los tiburones consiguen aparecer y desaparecer con una animación personalizada. Además, cada paso del salto está animado, haciendo sentir incluso más amenazantes a los tiburones.

Mar aleatorio

El mundo son dos planos intercambiándose para generar la ilusión de mar infinito, como ya se vio en el Apartado 5.3.1. Sin embargo, al observarse constantemente el mismo fondo y las mismas olas, se llega a un punto en el que la ilusión es rota y, por lo tanto, también la inmersión.

Para evitar este caso, cada vez que se vaya a producir un intercambio, el plano que se posiciona encima del otro, generará aleatoriamente una serie de imágenes y olas diferentes a lo largo del mismo. A causa de esto, la ilusión ya no puede ser detectada y la inmersión queda intacta.

5.5.2. Métricas

Las métricas son la manera que se tiene de medir el rendimiento de los jugadores y su *Game Feel*. Gracias a estas, se pueden extraer un mayor número de mejoras del sistema y entender cuáles son los puntos fuertes y débiles de los juegos. Al conocer estos mismos puntos, se pueden reforzar los fuertes y paliar los débiles.

Se desarrollan métricas tanto duras como blandas, en pos de obtener todos los datos posibles a la hora de aplicarlas. Una vez diseñadas y estudiadas cada una de las métricas, cuyos datos se extraerán tanto de la plataforma como del informe, se tendrán que aplicar en próximas iteraciones a un conjunto variado y, a ser posible, vasto de personas. Cuando se han aplicado estas métricas y extraído los datos, estos mismos se deben analizar y sacar conclusiones de ellos, mejorando la plataforma y cada uno de los juegos con cada una de éstas.

Métricas duras

Las posibles métricas duras en la plataforma son limitadas. Sin embargo, gracias a la base de datos, la cantidad de métricas a extraer se amplia considerablemente:

- **Puntuación máxima:** Encontrar cuál fue el momento exacto de mayor puntuación del jugador. Medir si ha sido un golpe de suerte puntual o es un reflejo de su progresión teniendo en cuenta su puntuación media en un periodo de tiempo definido.
- **Mejoría del jugador:** Estudiar en una gráfica su progresión en puntuación para cada juego individualmente en relación a cada sesión de juego a lo largo del tiempo.
- **Fatiga:** Estudiar cada sesión en una gráfica con todas las puntuaciones y observar cuando se produce el efecto de la fatiga en una sesión de juego. En esta gráfica, cuando la puntuaciones bajan cada vez más, se podría concluir que es causa de la fatiga. También se estudiará cómo cada juego afecta en estos términos al jugador, además de si se produce menor fatiga a lo largo de varias sesiones.

Métricas blandas

Un buen diseñador puede encontrar muchos errores en el diseño por sí mismo, pero, aún así, algunos puede pasar desapercibidos, como enfoques que se han podido no tener en cuenta o, incluso, situaciones con el jugador no deseadas por el mismo diseñador.

Las métricas blandas crean una linea de comunicación entre diseñador y jugador para conseguir un producto más pulido y con un *Game Feel* elaborado. Estas métricas permiten visualizar y comprobar esos enfoques no planteados por el diseñador y errores que se pudieran haber cometido.

Los informes serán una serie de preguntas que conducirán al diseñador a entender cómo su producto es percibido por los jugadores y proponer nuevos diseños en función de estos. El conjunto de preguntas debe plantear al menos una vez los temas principales de las métricas, vistos en la sección 2.2.1. La entrada, respuesta, contexto, pulido, metáfora y reglas. Además, habrá preguntas generales y específicas para cada juego.

El pseudo-informe se compone de las siguientes preguntas generales y específicas:

1. **¿Cómo notas el control? ¿En qué momentos te ha costado más?**
   - ***Arkanoid:* ¿Cómo es llevar la nave de un lugar a otro de la pantalla?**
   - ***Balance Journey:* ¿Cómo notas la flecha al cambiar de sentido?**

   Pregunta de *Entrada*. Se quiere extraer puntos o situaciones que den problemas a los jugadores, que les suponga un reto personal o si no notan que sus acciones naturales se vean reflejadas en los juegos.

2. **¿Responde rápido el control? ¿Te ha costado en algún instante? ¿El juego va lento?**
   - ***Arkanoid:* ¿Podías alcanzar a la esfera?**
   - ***Balance Journey:* ¿Cómo es la velocidad de la flecha al otro sentido?**

   Pregunta de *Respuesta*. Al hacer estas preguntas, se hace hincapié en todos los elementos de los juegos que podrían no responder rápidamente. Al preguntar si el juego va lento, no se hace referencia a la cantidad de *frames* o rendimiento del programa, sino a la lentitud en el *Game Feel*, porque, en ocasiones la falta de *frames* puede parecer la causa.

3. **Explica de qué crees que trata el juego.**
   - ***Arkanoid:* ¿Dónde se dirige la esfera cuando se pierde?**
   - ***Balance Journey:* ¿Por qué al personaje le cuesta tanto equilibrarse?**

   Pregunta de *Contexto*. Se observa cómo los jugadores entienden el juego y qué contexto personal añaden al mismo. Con la pregunta de *Arkanoid* se espera una respuesta sobre el fondo del espacio y, con la de *Balance Journey*, se espera una referencia a equilibrarse en una cuerda. Aunque, si el jugador no responde exactamente lo esperado, puede ser una oportunidad para cambiar el contexto del juego o especificarlo más.

4. **¿Qué efectos visuales te han gustado más? ¿Y los que menos?**
   - ***Arkanoid:* ¿Qué animaciones añadirías en el juego?**
   - ***Balance Journey:* ¿Qué te ha parecido el mar? ¿Y los movimientos de los tiburones y el personaje?**

Pregunta de *Pulido*. Se plantean los efectos visuales y se quiere averiguar cuales son las imágenes más y menos convincentes. En *Arkanoid*, se desea que los jugadores piensen en qué le faltaría para que fuera un juego más completo y, en *Balance Journey*, se espera que el jugador piense en la animaciones del mar, personaje y tiburones, si son muy repetitivas esas animaciones o el escenario del mar en sí mismo.

5. **¿Cuáles son tus primeras impresiones al ver los juegos?**
   - ***Arkanoid:* ¿Ha cumplido tus expectativas?**
   - ***Balance Journey:* ¿Qué esperabas de este juego?**

   Pregunta de *metáfora*. Explorar como cualquier tipo de conocimiento previo ha afectado al juego y si han llevado a confusiones o descubrimientos interesantes. Con la de *Arkanoid*, se espera que el conocimiento de este juego tan conocido se compare con otras versiones del mismo y, con *Balance Journey*, comprobar si los jugadores tienen una idea adecuada del juego antes de empezar.

6. **¿Qué partes han sido las más divertidas? ¿Mecánicas que te hayan gustado más? ¿Y las que menos?**
   - ***Arkanoid:* ¿Cuál es tu sensación al chocar con la esfera?**
   - ***Balance Journey:* ¿Qué sensaciones te produce mantener el equilibrio hasta el último momento?**

   Pregunta de *Reglas*. Se quiere observar cómo las reglas de cada juego afectan al usuario y su *Game Feel*.

Una vez terminadas estas preguntas, el ejemplo de los dos informes completos se puede encontrar en el Anexo B.

5.5.3. Mejoras

Las posibles mejoras que llevar a cabo con lo realizado en esta iteración son:

- **Animaciones:** Más variedad y cantidad de frames para generar la sensación de un movimiento más fluido en *Balance Journey*. Crear animaciones de fondo en el espacio para el juego tipo *Arkanoid*, consiguiendo mucho más dinamismo y atractivo visual.
- **Aleatoriedad:** Hacer que las olas aparezcan aleatoriamente en los dos planos y ejecuten sus animaciones para, entonces, desaparecer. Ver constantemente las olas creándose y desapareciendo en el plano da una sensación más realista del mar.
- **Métricas:** Ponerlas en práctica y coordinarse con especialistas para mejorar tanto las propias métricas como los juegos, o bien desarrollar otros que coincidan mejor con el objetivo del especialista.

6. Resultados, Trabajos futuros y Conclusiones

6.1. Resultados

En todo este proyecto se han realizado una serie de características, las cuales, han contribuido a tener una plataforma de *serious games* con dos videojuegos, varios posibles controles y una base de datos completamente funcional.

Se partía de un motor base 3D con funcionalidades limitadas, para acabar consiguiendo un motor renovado para tanto 2D como 3D con sistemas de colisiones 2D, comunicaciones *UDP*, gestor de ventanas, objetos de juego con acciones de actualización y colisión, gestor de juegos y sus recursos, disparadores y animaciones.

Las colisiones 2D permiten detectar una superposición de dos rectángulos y de un círculo con un rectángulo, comprobándose justo después de haber realizado la actualización y, así, hacer los ajustes necesarios al escenario antes del dibujado.

Las comunicaciones *UDP* permiten conectar cualquier dispositivo a través de la red mientras se cumpla el protocolo de transmisión. Se usa para poder conectar cualquier tipo de control externo a la plataforma sin necesidad de instalaciones previas ni controladores adicionales, haciéndolo un sistema modular.

El gestor de ventanas que permite realizar botones, etiquetas, menús y todo tipo de interfaces como la pantalla de selección, el panel de control o la muestra de tablas.

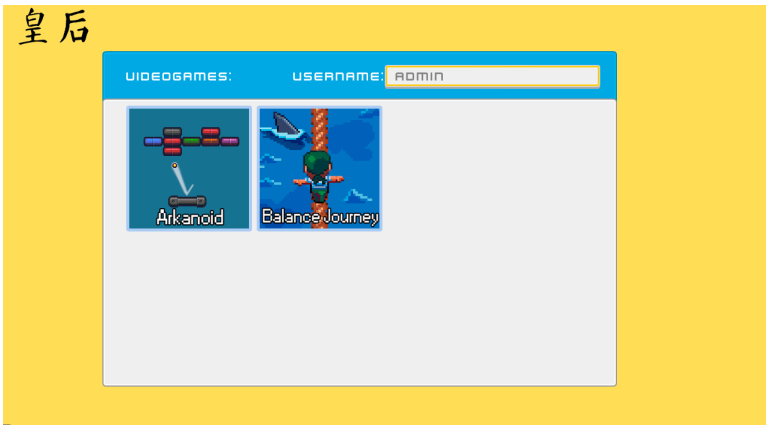


Figura 6.1: Pantalla de selección con los dos juegos y pulido visual.

GameObjects que permiten mostrar, actualizar y colisionar cualquier elemento en pantalla. Se le dan comportamientos y acciones específicas a este tipo de entidades que los convierten en elementos modulables. Siendo, además, un recurso de juego que maneja cada juego indi-

vidualmente y el gestor de juegos se encarga de que cada uno de ellos libere recursos cuando se cambie a otro. Adicionalmente, los disparadores son objetos de juego cuyo propósito no es mostrarse sino colisionar y actualizar en consecuencia.

Las animaciones permiten *frames* variables, asegurando que se muevan por la pantalla en relación a un origen y destino dependiendo de un conjunto de tramos previamente procesados con sus velocidades y destinos intermedios.

Los diferentes tipos de control principales desarrollados son el binario, que proporciona dos estados posibles; el proporcional, que recibe un porcentaje de fuerza; y el de flanco, que percibe un instante de activación. Estos tres controles se pueden aplicar a cada juego y el código está preparado para añadir más tipos sin necesidad de cambiar grandes cantidades de código, ya que las dependencias son mínimas.

El juego tipo *Arkanoid*, cuyas mecánicas tan bien definidas sirven para establecer un afianzado campo de pruebas a los controles.

Figura 6.2: Videojuego tipo *Arkanoid* con el pulido visual.

Balance Journey, juego diseñado con el objetivo de usarse con el control proporcional, supliendo las dificultades del mismo con su contexto y sensación de juego.

Figura 6.3: *Balance Journey*, el videojuego de equilibrio.

Una base de datos con todos los campos necesarios para monitorizar a los usuarios, viendo sus puntuaciones a lo largo del tiempo. Esta base permite pequeñas ampliaciones con facilidad que llevará a una gran riqueza de datos, pudiendo usarse en el campo del *Big Data* una vez recogidos los suficientes.

6.2. Trabajos Futuros

La plataforma tiene mucho rango de mejora y, como tal, está preparada para expandirse todo lo necesario. A lo largo de todo el proyecto, se han planteado muchas mejoras al sistema actual. Sin embargo, también hay otros trabajos que se plantearán para futuras iteraciones de este proyecto. Con cada iteración de trabajo se conseguirá una plataforma más rica y completa.

Los juegos, a pesar de ser versión *MVP* y, por lo tanto, completamente funcionales, aún no están terminados. Faltan las mecánicas en relación a las mejoras, algunas características de los juegos planteados anteriormente y la progresión de dificultad a medida que avance l partida. Aunque, antes de añadir las mecánicas restantes, hay que desarrollar las versiones *MVP* de los dos juegos restantes, *Qing Wa Jumper* y *Puzzle Scroll*.

La plataforma también se beneficiaría de un modo de juego que consistiera en un plan de entrenamiento con niveles intercalados de diferentes juegos. Este plan variaría entre videojuegos de ritmos pausados y de reflejos para no fatigar al jugador, dándole tiempo a descansar entre los juegos más exigentes. Además, se puede hacer que algunos de los juegos más difíciles de controlar, estén bloqueados de base, para que, cuando el jugador consiga más habilidad con el control mioeléctrico en los juegos más sencillos, se desbloqueen los difíciles y así el jugador sienta progresión y avance personal. Adicionalmente, se puede recalcar esta posible progresión y avance, indicando cuando el jugador se ha superado a sí mismo con una pantalla específica para cada caso y dándole multiplicadores a la puntuación cuando su rendimiento sea muy bueno. Todo ello para mantener su interés en la plataforma.

Pulido de la simulación espacial, el entorno auditivo y visual. El primer pulido se podría mejorar añadiendo elasticidad a los objetos, haciendo que al chocar con una pared haya un pequeño rebote, activando una animación de estiramiento y encogimiento a la hora de moverse por pantalla o estelas de velocidad indicando su movimiento. Mejorar las colisiones para que sean más precisas y actúen más naturales, convencen aún más al usuario del espacio de juego. El pulido visual se incluiría añadiendo animaciones más orgánicas, diferentes colores de personajes y enemigos. Un elemento visual que puede ayudar a las personas que usen prótesis, es mantener en pantalla una referencia visual de cómo sería el movimiento del brazo cuando se usa el control mioeléctrico. Por último, el pulido auditivo produciría sonidos que acompañen cualquier acción ocurrida en pantalla: rebotes, velocidades, animaciones etc.

Introducir tutoriales orgánicos para que el jugador se habitúe de manera natural a todos los juegos desarrollados, dándole información cuando la necesite e introduciéndole a las mecánicas una a una.

La base de datos se ampliará, añadiendo más información relevante a cada partida y persona. Se centralizará la base en un servidor para que la plataforma pueda usarse en diferentes dispositivos con un modelo usuario-contraseña, provocando que la recolección e interpretación de todos los datos se realice en un único archivo. Además, se podrá navegar por la base de datos y realizar peticiones a la misma sin necesidad de un programa externo. Esta cen-

tralización abre la posibilidad de tener la plataforma no solo en dispositivos de sobremesa o portátiles, sino también en aparatos móviles.

Y, por último, aplicar las métricas desarrolladas en un número amplio de personas en pos de encontrar aún más mejoras a futuro y hacer una plataforma más pulida a la par que completa.

6.3. Conclusiones

A lo largo de este proyecto han habido dificultades que se han ido superando una a una. *Bugs*, *glitches*, rediseños, pulido visual, características añadidas o controles, son ejemplos de los obstáculos que se han presentado en el desarrollo de la plataforma. Sin embargo, la misma está terminada y preparada para añadir más contenido. Las vicisitudes que han supuesto su desarrollo han servido para aprender, plantear mejoras para el futuro y sacar conclusiones.

Aún con todo lo aprendido, diseñar para estos controles es un nuevo paradigma a estudiar. De hecho, algunas empresas de plataformas de videojuegos hacen los controles en función de sus juegos estrella y no al revés. Este control mioeléctrico a nivel jugable es un campo sin explorar, esperando ser labrado y, aunque se ha tocado la superficie, en este proyecto aún hay mucho que realizar y comprobar. El ensayo y error, con unas buenas metodologías a la hora de analizar las métricas, son elementos importantes en la mejora del proyecto.

La moduralidad ha sido uno de los *leitmotive* del proyecto, siéndolo también, la posibilidad de ampliarlo. En pocas ocasiones, se querrá ampliar aún más la plataforma, llegando al punto de cambiar tanto el motor que algunas características dejen de funcionar o haya que realizar un parche para cada juego individualmente. Esto limita la moduralidad. Para evitar este caso, cuando se quieran rehacer características que puedan afectar a otras del motor, se realizará una versión derivada del mismo con los cambios añadidos para el juego en cuestión. Todas las características están desarrolladas con la posibilidad de generar estas versiones derivadas.

Aunque la verdadera moduralidad está en la transmisión mediante el protocolo *UDP*. Mientras el dispositivo ejecutando la plataforma tenga tarjeta de red y cumpla el protocolo de transmisión ya definido, siempre se podrá utilizar sin problemas la propia plataforma, independientemente de cómo y con qué se interpreten las señales.

En este proyecto, por cada juego e iteración realizados se ha tenido que desarrollar al menos una nueva característica al motor. A medida que las necesidades de los nuevos juegos lo requieran, se tendrá un motor más rico en características, ampliando su funcionalidad y posibilidades. Si los juegos futuros son cada vez más ambiciosos, nunca se dejarán de añadir características, pero si el alcance de estos juegos no supera el ya visto, llegará un punto en el que no se necesitarán más características, reduciendo el tiempo de desarrollo.

Los juegos son sistemas complejos llenos de variables y lógica matemática. Si hay demasiados de estos dos elementos sin una buena base modular, los resultados pueden ser catastróficos. Por ejemplo, recursos que no se liberan, bloqueos de programa, salidas no deseadas del mismo y comportamientos indefinidos de los objetos. La solución a este problema es tener una buena base en el motor, además, de conocer cuando y cómo se accionan los diversos objetos del programa en pos de encontrar fácilmente errores y no cometerlos.

En este proyecto se ha observado como cada juego es un mundo diferente, con sus requisitos y diseños definidos individualmente. Es por ello que el funcionamiento del control en cada uno debe ser específico. Aunque el control binario es altamente adaptable y hace las veces de

teclado, el control proporcional no debería funcionar igual en dos juegos diferentes. Si se usa el proporcional como una extensión más precisa del binario, se puede no estar aprovechando el control ni sus ventajas.

Los controles tienen solamente tres acciones y esto, a pesar de ser una limitación, es, además, una ventaja que evita hacer juegos desproporcionalmente ambiciosos y proporciona más tiempo a desarrollar unas mecánicas interesantes y útiles para entrenar con control mioeléctrico.

Todos estos juegos demuestran la cantidad de posibilidades que existen en el diseño de juegos con el propósito de ayudar en la neurorehabilitación e investigaciones para diferentes tipos de dolencias motrices y cognitivas. Aún con todo, solo se ha alcanzado la superficie del problema, habiendo aún más posibilidades que no han llegado a ser exploradas.

En lo personal, participar en este proyecto ha sido una experiencia enriquecedora al ponerme a prueba como diseñador, programador e ingeniero. Cada reto que aparecía frente a mí, me ayudaba a tener una idea más enfocada del proyecto en todo su alcance.

Al final, todo lo que existe en este proyecto tiene un propósito definido. Cada mejora planteada se acerca más a una plataforma completa y las posibilidades son interminables. Redes neuronales, *Big Data*, estudio del control, diseño de juegos en contextos no tradicionales, *Game Feel*, interpretación de señales y comprobaciones de nuevas arquitecturas robóticas de control mioeléctrico. Todo ello puede acercarse más, gracias a la plataforma, *Huanghou*.

Bibliografía

Josef Wiemeyer, R. S. W. (2016). *Serious games: Foundations, concepts and practice*. Springer International Publishing Switzerland.

Oskar C. Aszmann, C. I. F. P. (2017). Game-based rehabilitation for myoelectric prosthesis control. *US National Library of Medicine*. Descargado 2019-05-06, de `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5324011/`

Sommerville, I. (2005). *INGENIERÍA DEL SOFTWARE* (Seventh ed.). PEARSON EDUCACIÓN, S.A., Madrid.

Swink, S. (2009). *Game feel: A game designer's guide to virtual sensation*. Morgan Kaufmann Publishers.

West, M. (2008a). Measuring responsiveness in video games. *Cowboy Programming*, 1. Descargado 2019-05-06, de `http://cowboyprogramming.com/2008/05/30/measuring-responsiveness-in-video-games/`

West, M. (2008b). Pushing buttons. *Cowboy Programming*, 1. Descargado 2019-05-06, de `http://cowboyprogramming.com/2007/01/02/pushhing-buttons/`

A. Anexo I: Guía de usuario

En este anexo se procederá a explicar la plataforma con todos sus componentes y cómo el usuario puede instalarla y usarla. Existen dos partes diferenciadas en el manual: la plataforma como *software* y con qué partes se puede interactuar, el *script* en *Matlab* que permite enviar las señales ya procesadas a la propia plataforma mediante un protocolo definido.

A.1. Manual de plataforma

A.1.1. Instalación

La plataforma, *Huanghou*, es un *software* para *Windows 7* en adelante con un sistema de carpetas totalmente dinámico. Para poder ejecutar este programa no es necesaria una instalación engorrosa de varios pasos y elecciones. Directamente, con tener la carpeta del proyecto en cualquier directorio del computador y no modificar la estructura del mismo, definida en la Figura A.1, se puede ejecutar el *software*, estando localizado en el binario llamado *main_win.exe* en la carpeta *bin* del proyecto.

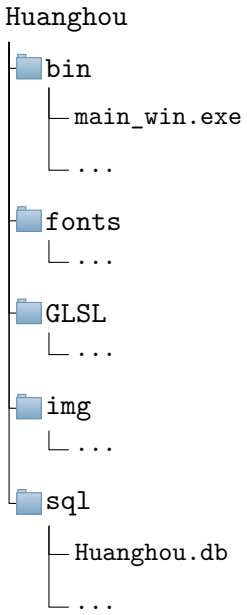


Figura A.1: Diagrama de carpetas del proyecto

A.1.2. Programa

El programa en sí mismo se compone de diversas características a nivel de usuario. Los menús se manejan con el ratón, pero los juegos con el control mioeléctrico.

Pantalla de selección

La pantalla de selección en la Figura A.2 permite escoger el juego deseado e introducir el nombre de la persona que jugará mediante ratón y teclado. Los componentes de esta pantalla son:

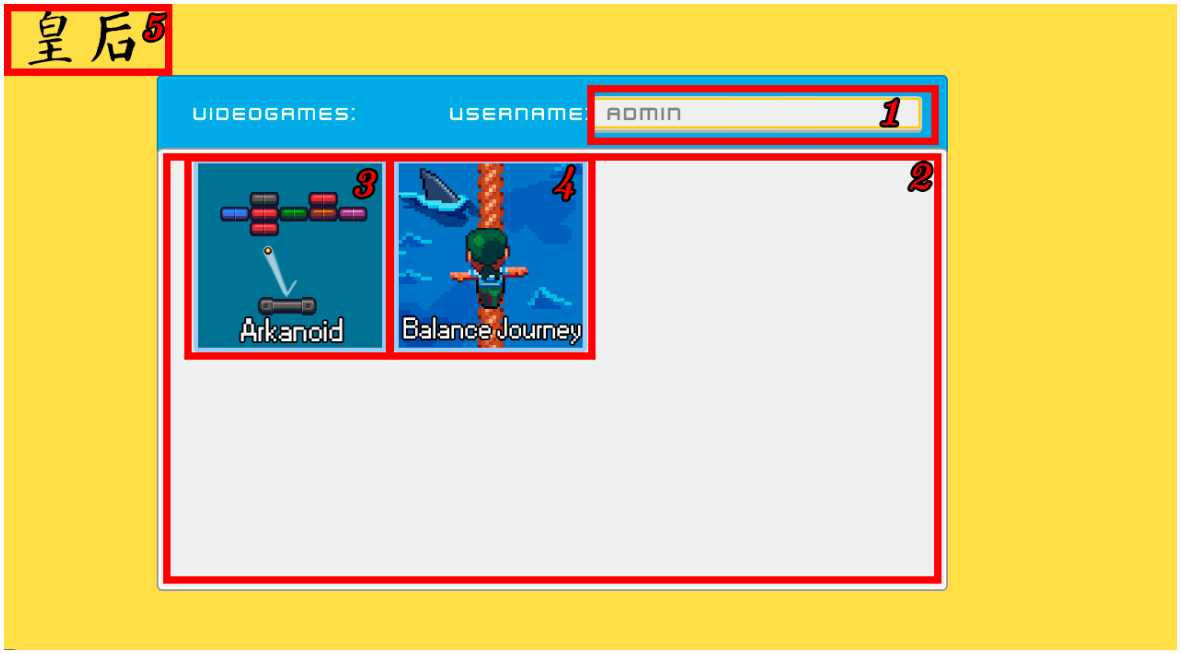


Figura A.2: Diagrama enumerado de la pantalla de selección con cada uno de sus componentes señalados.

1. Campo de texto a rellenar con el nombre del usuario que vaya a jugar. Éste solo se puede cambiar en esta pantalla, no durante la ejecución de cualquier juego.
2. Zona de la pantalla dedicada a los botones con imágenes de los juegos que se cargan al pulsarlos.
3. Botón con imagen descriptiva, cuya pulsación carga el juego tipo *Arkanoid*.
4. Al pulsarse este botón del *Balance Journey*, carga el juego en cuestión.
5. Logo de *Huanghou* sin ningún tipo de funcionalidad.

Panel de control

El panel de control se activa y desactiva con la tecla *F1*. Además, no depende de la pantalla de selección para activarse. Puede aparecer en cualquier momento a través del motor. A su

disposición hay muchos datos y elementos de control para comprobar o ajustar la funcionalidad del mismo. Tiene a su disposición tres categorías: comunicación, encargada de mostrar todas las variables relevantes en la comunicación de datos; juego, dedicada a la muestra de datos del juego seleccionado y a la selección de controles; y, por último, BD permite reiniciar la base de datos y mostrar la tabla *SCORE*. Si la base de datos no existe inicialmente, se tiene que crear pulsando el botón de reinicio. En el caso de no pulsarlo, pueden darse comportamientos indefinidos.

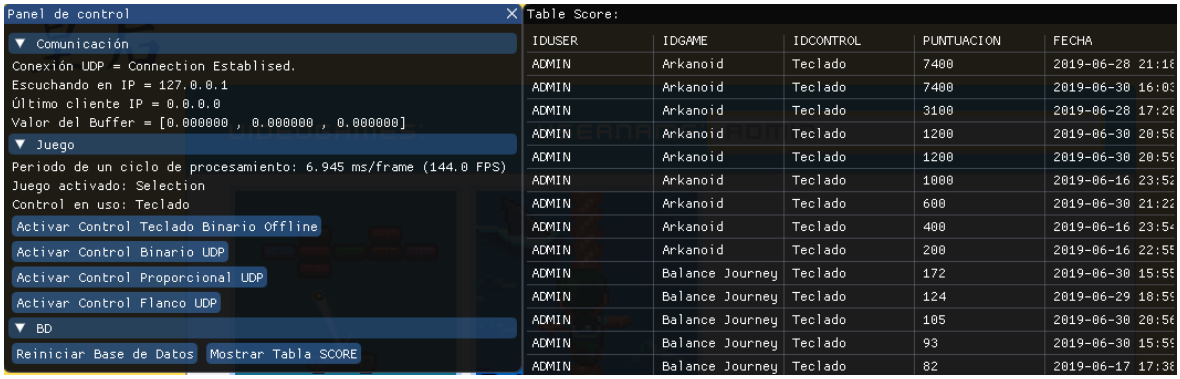


Figura A.3: Panel de control con todas las categorías desplegadas y la tabla *SCORE*.

Arkanoid

El videojuego tipo *Arkanoid* usa el control mioeléctrico y tiene estos componentes:

Figura A.4: Diagrama enumerado del juego tipo *Arkanoid* con cada uno de sus componentes señalados.

1. Muros que impiden a la nave y al proyectil salir de los límites del juego.
2. La nave que controla el usuario y cuyo movimiento es horizontal.
3. Proyectil esperando a ser accionado por la cocontracción.
4. Conjunto de bloques con todos los tipos de entereza.
5. Puntuación más alta del jugador y la actual.
6. Cantidad de vidas restantes en el juego.
7. Botón exclusivamente accionado por el ratón, provocando el retorno a la pantalla de selección.

Balance Journey

El videojuego *Balance Journey* usa el control mioeléctrico y tiene estos componentes:

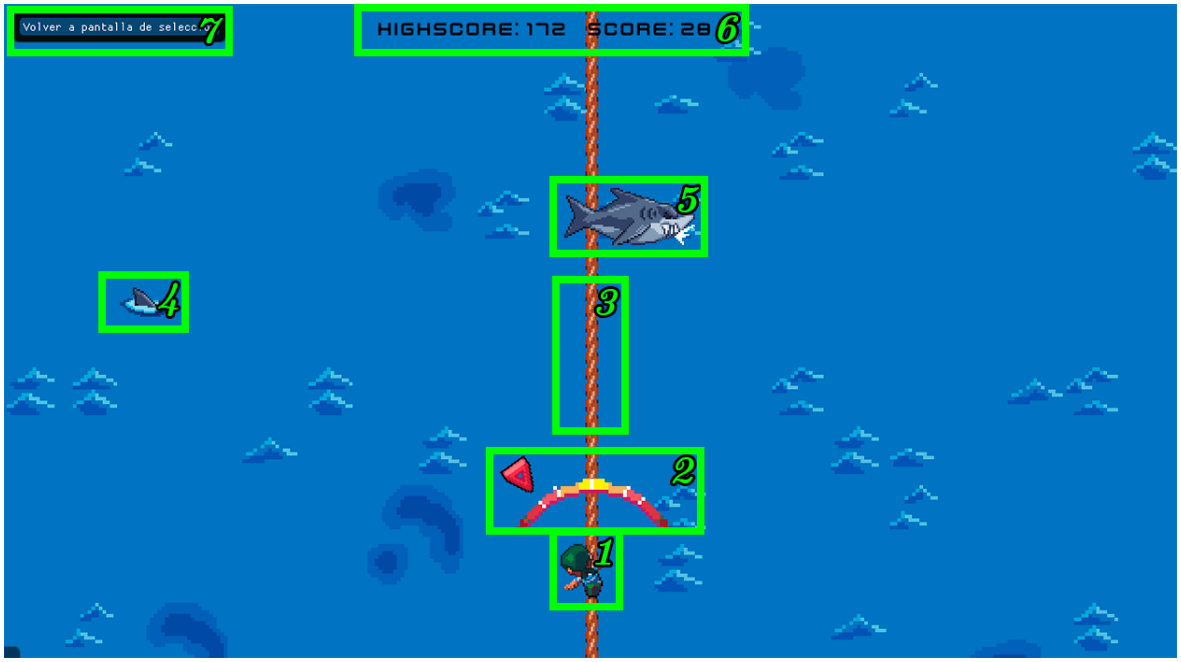


Figura A.5: Diagrama enumerado de *Balance Journey* con cada uno de sus componentes señalados.

1. El corredor, personaje principal del juego y avatar del jugador.
2. Arco con flecha que indica el equilibrio del personaje.
3. Cuerda por la que avanza y equilibra el jugador.
4. Tiburón escondido, acercándose para atacar.
5. Al saltar del agua asesta un ataque, pero falla al estar lejos del personaje.

6. Puntuación más alta del jugador y la actual.
7. Botón exclusivamente accionado por el ratón, provocando el retorno a la pantalla de selección.

A.1.3. Base de datos: Navegación

La plataforma solamente puede, por ahora, mostrar la tabla *SCORE*. Por lo que, si se quiere navegar por toda la base de datos y realizar consultas personalizadas, se recomienda usar el programa *DB Browser for SQLite*. La base de datos se encuentra en la carpeta *sql* del proyecto bajo el nombre *Huanghou.db*. Se pueden realizar cambios en la estructura de la base de datos, pero ello puede conllevar problemas de coherencia con la plataforma. Se debe realizar esta opción con cuidado. Si se genera un problema grave, siempre se puede reiniciar la base de datos a su estado original con el panel de control.

A.2. Manual Matlab

En este proyecto se usa *Matlab* por ser un entorno muy utilizado en el tratamiento de señales y, aunque no es el mejor en rendimiento, permite mucha versatilidad. Aún así, mientras se mantenga el protocolo de envío de datos, se puede usar cualquier herramienta que pueda enviar información mediante el protocolo *UDP*.

Esta parte del manual tratará cómo se ha realizado el envío de datos en un *script* de *Matlab* en pos de que, si se desea replicar en otra tecnología, se tengan los conocimientos para ello.

A.2.1. Envío de ejes

La Figura A.6 indica cómo tiene que ser el mensaje a enviar para que la plataforma sea capaz de captarlo y procesarlo correctamente. Son tres números reales entre 0.0 y 1.0 que siguen el estándar *IEEE-754*. Si se llega a enviar datos en otro estándar, la plataforma puede actuar con comportamientos indefinidos.

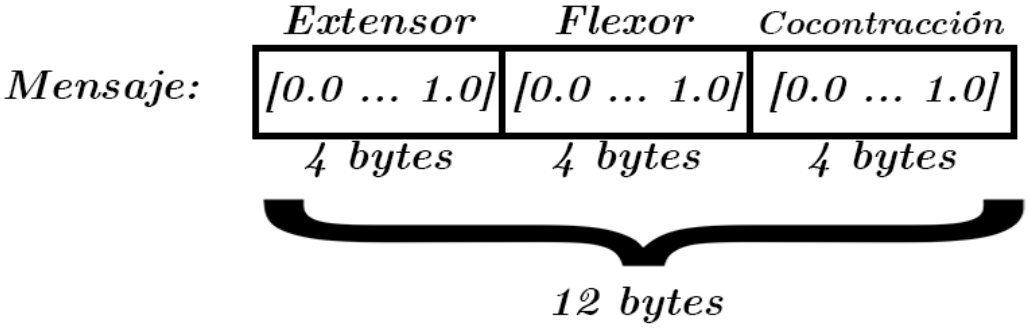


Figura A.6: Protocolo de envío de datos.

Además, se necesita que la plataforma envíe cada real en el formato *LittleEndian*. Esta manera de organizar los datos consiste en enviar un real con cada byte ordenado del menos

significativo al más significativo. En el caso de un número en hexadecimal como *0xFF884422* el envío será en este orden *[0x22 0x44 0x88 0xFF]*.

La plataforma está preparada para recibir estos 12 bytes y ninguno más en un instante de tiempo. Si se envían muchos datos en un intervalo pequeño de tiempo, la plataforma no se saturará porque el *buffer* de destino está limitado a 12 bytes. De esta manera, si se envían dos mensajes antes de que la plataforma sea capaz de procesarlos individualmente, se usará el mensaje más reciente.

A.2.2. Protocolo UDP

Para cumplir el formato de envío, un gestor del protocolo *UDP* en *Matlab* se tiene que ajustar a todas las opciones anteriormente dichas y, como se puede discernir en el Código A.1, se cumplen cada una de las condiciones.

Código A.1: *Script* con una función para envíar los mensajes.

```
function send3Floats(f)
    persistent u;
    if isempty(u)
        % Cantidad de números reales a enviar
        floatCount = 3;
        % Espacio ocupado por cada número real
        sizeBytes = 4;

        % IP y puertos
        u = udp('127.0.0.1', 2812);
        %Cantidad de datos a enviar
        u.OutputBufferSize = sizeBytes*floatCount;
        %Formato de envío
        u.ByteOrder='littleEndian';
        fopen(u);
    end
    fwrite(u,f,'single');
end
```

El gestor del protocolo *UDP* es una variable de tipo *persistent*, haciendo que exista la misma variable para cualquier llamada de la función. Inicialmente, estas variables son nulas, por lo que cuando lo es, se inicializa el gestor con todos los parámetros necesarios una única vez en toda la ejecución de un programa de *Matlab*.

Por ahora, la plataforma solo recibe mensajes que provengan de la misma máquina en la que se ejecuta. Entonces, si se cambia manualmente la ip para hacer el envío de mensajes en una máquina diferente, la recepcion de mensajes por parte de la plataforma no funcionará. Si, aún así, se desea hacerlo, el usuario puede desarrollar un *script* personalizado en *Matlab* que haga de intermediario entre las dos máquinas.

B. Anexo II: Métricas Blandas

B.1. Arkanoid

Informe de métricas: *Arkanoid*

---

1. **¿Cómo notas el control? ¿En qué momentos te ha costado más? ¿Cómo es llevar la nave de un lugar a otro de la pantalla?**

2. **¿Responde rápido el control? ¿Te ha costado en algún instante? ¿El juego va lento? ¿Podías alcanzar a la esfera?**

3. **Explica de qué crees que trata el juego. ¿Dónde se dirige la esfera cuando se pierde?**

4. **¿Qué efectos visuales te han gustado más? ¿Y los que menos? ¿Qué animaciones añadirías en el juego?**

5. **¿Cuáles son tus primeras impresiones al ver los juegos? ¿Ha cumplido tus expectativas?**

6. **¿Qué partes han sido las más divertidas? ¿Mecánicas que te hayan gustado más? ¿Y las que menos? ¿Cuál es tu sensación al chocar con la esfera?**

B.2. Balance Journey

Informe de métricas: *Balance Journey*

1. **¿Cómo notas el control? ¿En qué momentos te ha costado más? ¿Cómo notas la flecha al cambiar de sentido?**

2. **¿Responde rápido el control? ¿Te ha costado en algún instante? ¿El juego va lento? ¿Cómo es la velocidad de la flecha al otro sentido?**

3. **Explica de qué crees que trata el juego. ¿Por qué al personaje le cuesta tanto equilibrarse?**

4. **¿Qué efectos visuales te han gustado más? ¿Y los que menos? ¿Qué te ha parecido el mar? ¿Y los movimientos de los tiburones y el personaje?**

5. **¿Cuáles son tus primeras impresiones al ver los juegos? ¿Qué esperabas de este juego?**

6. **¿Qué partes han sido las más divertidas? ¿Mecánicas que te hayan gustado más? ¿Y las que menos? ¿Qué sensaciones te produce mantener el equilibrio hasta el último momento?**