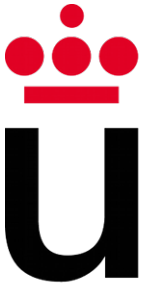

Universidad
Rey Juan Carlos

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS DE DECISIÓN

Curso Académico 2024/2025

Trabajo Fin de Máster

# DESARROLLO DE UN SISTEMA DE ANÁLISIS DE ACTIVIDAD Y PERFILADO DE JUGADORES EN UN VIDEOJUEGO

Autor : Roberto García Martín
Tutor : José Felipe Ortega Soto

# Trabajo Fin de Máster

Desarrollo de un Sistema de Análisis de Actividad y Perfilado de Jugadores en un Videojuego

**Autor :** Roberto García Martín
**Tutor :** José Felipe Ortega Soto

La defensa del presente Proyecto Fin de Máster se realizó el día de de 2025, siendo calificada por el siguiente tribunal:

**Presidente:**

**Secretario:**

**Vocal:**

y habiendo obtenido la siguiente calificación:

**Calificación:**

Móstoles, a de de 2025

*A mi madre Consuelo, a mi padre Carlos y a mi pareja Eva,*
*las alas de mis logros.*

*A los que no se rinden.*

# Agradecimientos

Empezar agradeciendo a mi madre Consuelo, por sugerir que sigua mi educación y recordarme que “sobretodo” es una prenda de vestir y no el conector. A mi padre Carlos por facilitarme el día a día, hacerme reír y recordarme la importancia de guardar los archivos.

A mi pareja Eva, por esucharme todos los días que estaba agobiado, por involucrarse en mí y abrazar lo que hago, insistiendo que yo podría acabar esto.

A los compañeros de clase por haberme abierto nuevos mundos. A mis compañeros de la carrera y de equipo por estar a mi lado.

Y a mi tutor, por confíar en mí, enseñarme tanto en estos meses y dedicar tiempo y paciencia a este proyecto.

# Resumen

El trabajo descrito en esta memoria trata sobre el uso de análisis de datos en el desarrollo de videojuegos. En medianas y grandes empresas esta práctica lleva en uso más de dos décadas. Sin embargo, en el desarrollo de videojuegos independiente es un campo poco común debido a la falta de recursos y herramientas accesibles.

El proyecto tiene como objetivo comprobar si es viable y útil aplicar sistemas de decisiones basados en datos en un desarrollo independiente de un videojuego. Para ello, se ha realizado un caso de estudio con el videojuego *Battlecore Robots*, un producto que empezó a crearse en la Universidad Rey Juan Carlos.

Al comenzar este trabajo, para financiar la producción del videojuego, se estaba desarrollando un perfil en Patreon, una página de micromecenazgo.

Se aplicarán tecnologías orientadas a la captación y tratamiento de datos, para pefilar usuarios y su actividad con el fin de mejorar el producto de cara a dicho objetivo. La metodología empleada se basa en definir arquitecturas capaces de procesar información recogida durante la ejecución del juego, almacenarla en una base de datos y posteriormente analizarla mediante herramientas técnicas de documentación. Estos datos permiten diseñar y visualizar sistemas de toma de decisiones para optimizar la experiencia de juego.

# Summary

The work described in this thesis deals with the use of data analysis in video game development. This practice has been in use for more than two decades in medium and large companies. However, it is uncommon in independent video game development due to a lack of accessible resources and tools.

The project aims to verify whether it is feasible and useful to apply data-driven decision-making systems in independent video game development. To this end, a case study was conducted with the video game *Battlecore Robots*, a product that began its development at the Rey Juan Carlos University.

At the beginning of this project, a profile was being developed on Patreon, a micropayment website, to finance the production of the video game.

Technologies aimed at data collection and processing will be applied to profile users and their activity in order to improve the product with this objective in mind. The methodology used is based on defining architectures capable of processing information collected during the execution of the game, storing it in a database and subsequently analysing it using technical documentation tools. This data allows decision-making systems to be designed and visualised in order to optimise the gaming experience.

# Índice general

# Índice de figuras

# Índice de tablas

# Índice de fragmentos de código

# Capítulo 1

# Introducción

Los videojuegos se han convertido en uno de los medios de entretenimiento más grande de la época. Se han estimando más de tres mil millones de jugadores [25] y un total de trescientos mil millones de dólares generados en ventas [52], tal como se recoge en la figura 1.1. Existen múltiples definiciones de qué es un videojuego, pero comúnmente un videojuego es un programa o un juego electrónico en un hardware con el fin de entretener, en el que se somete a un jugador a cumplir unas reglas y lograr uno o varios objetivos [21].

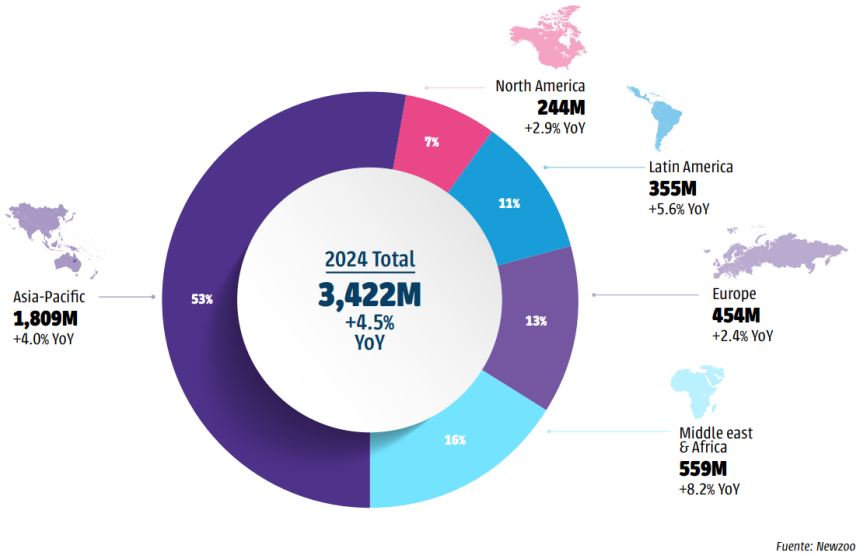


Figura 1.1: Número de jugadores de videojuegos por región [13].

El comienzo de su historia se remonta a la década de 1950 con el fin de la Segunda Guerra Mundial y el desarrollo de los primeros superordenadores programables. Aún sin estar claro cuál fue el primer videojuego debido a la interpretación de su definición, se puede considerar que el primer videojuego que se hizo fue "OXO" (figura 1.2), un tres en raya desarrollado en Reino Unido en el que un jugador podía enfrentarse a la máquina.

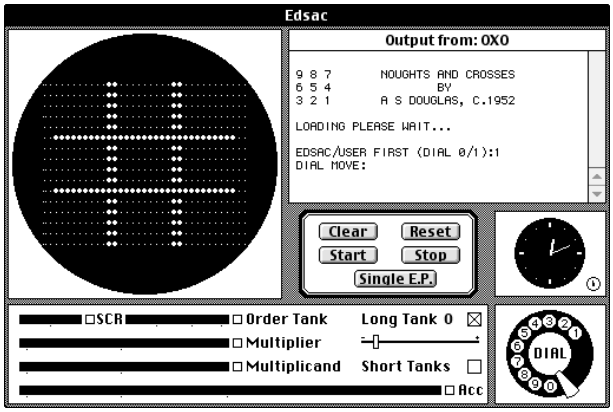


Figura 1.2: Pantalla del que se cree que es el primer videojuego de la historia, "OXO" [31].

Hoy en día, los videojuegos tienen un papel en la sociedad, no solo por el entretenimiento, si no porque también se utilizan con otros fines como son la educación y la medicina. En cuánto a la educación, existen diferentes aplicaciones y juegos para aprender de forma interactiva. Alguna de las más conocidas puede ser "Duolingo" [9], que se usa para aprender idiomas mientras el usuario sube de nivel, accediendo a desafíos más difíciles conforme más alto sea el nivel; o "Kahoot" [27] con el que se pueden crear preguntas para que sean respondidas por un grupo de personas, comúnmente utilizada en aulas de Colegios, Institutos y Centros de Eduación superior.

En la figura 1.3 se muestra el usuario final de este tipo de aplicaciones. Se puede observar que los clientes que utilizan más estas aplicaciones son usuarios individuales con el fin de mejorar una habilidad jugando.

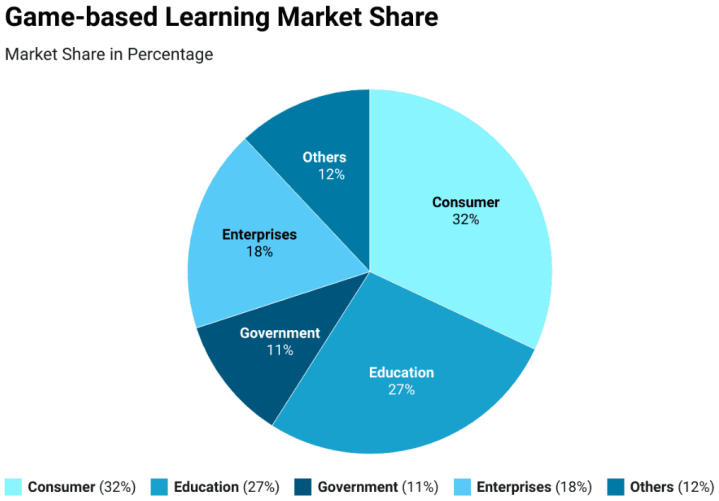


Figura 1.3: Mercado de aprendizaje basado en juegos [32].

En España, el sector del videojuego está en constante crecimiento, facturando en 2023 alrededor de mil cuatrocientos veinticinco millones de euros. Existen múltiples estudios con diferentes modelos de negocio. Los cinco modelos más empleados son: venta digital pre-

mium, venta de servicios, venta física premium, *outsourcing* (desarrollar para una empresa externa) y compras *in-game* (dentro del juego usando dinero real) [13]. Es una industria que genera empleo en el país. Se estima un crecimiento de 4 veces la faturación actual y de 2,3 del empleo [13] en 2030.

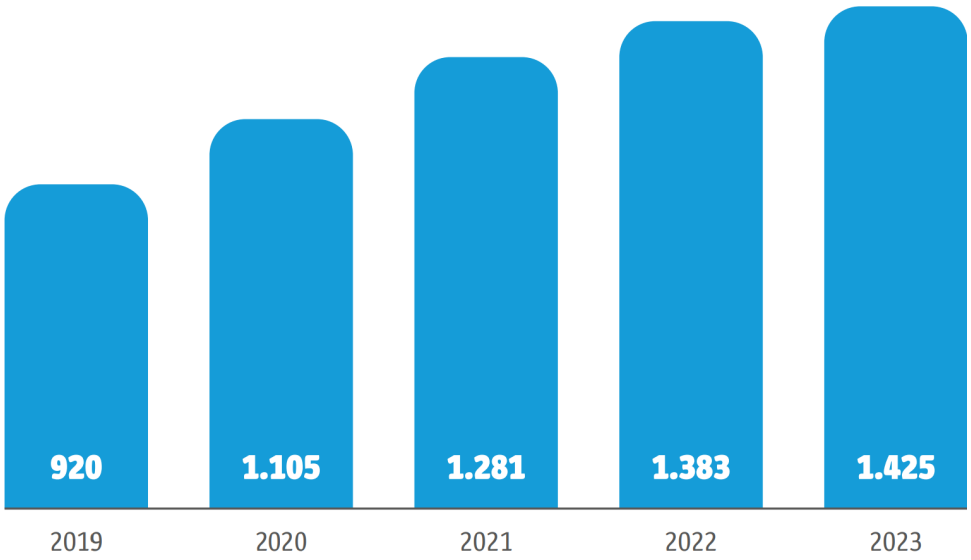


Figura 1.4: Evolución de la facturación de los estudios de videojuegos en España en millones de euros (2019-2023) [13].

Como se ha podido observar, el negocio de los videojuegos sigue en auge, habiendo mucha oferta de productos. Las plataformas para consumir videojuegos a día de hoy son:

- Steam: plataforma digital de Valve, para jugar juegos en ordenador. En el año 2024, se publicaron un total de 18.755 juegos [44].
- Play Station: consola de mesa (necesitan un televisor o monitor para ser usadas) de Sony. Sus modelos más recientes son PlayStation 4 y PlayStation 5 con unas ventas de más de 110 y 65 millones de unidades respectivamente [51].
- Xbox: consola de Microsoft. La familia Series X/S lleva alrededor de 30 millones de unidades vendidas y la plataforma Xbox también incluye servicios como Xbox Game Pass y Xbox Cloud Gaming, para jugar vía streaming [19].
- Nintendo: fabricante de consolas con una gran herencia. Ahora con la Switch 2, lanzada en junio de 2025, que vendió más de 3,5 millones de unidades en solo cuatro días; convirtiéndose en el lanzamiento más rápido de Nintendo [28].
- Dispositivos Móviles: engloban juegos en Android e iOS, siendo la plataforma que más genera con un 49% total del mercado [52].

Al tratarse de un modelo tan asentado, es común en las empresas aplicar análisis de datos para llevar a cabo este tipo de productos. Se suele emplear este tipo de procesos no solo

para analizar un juego a nivel de producto (como ventas, regiones con más usuarios...), si no también para realizar análisis de comportamiento de usuario con el fin de mejorar la experiencia lúdica [39].

A día de hoy, existen herramientas asentadas, como es el servicio web "GameAnalytics", que cuenta con múltiples soluciones para análisis, procesamiento de datos e inferencia en publicidad y mercado. Consultando el libro *Game Analytics: Maximizing the Value of Player Data* se pudo observar que el tratamiento de información con el fin de mejorar productos de gran escala se lleva aplicando años por grandes y medianas empresas. Fue en la década de 1990 y los inicios del 2000 cuando Sony y Electronic Arts empezaron a tratar estadísticas, con métricas como los usuarios activos mensuales.

El presente trabajo abordará el análisis de datos desde la perspectiva del desarrollo de videojuegos independientes, centrando el estudio en *Battlecore Robots*, un videojuego creado en 2021 por cinco estudiantes de la Universidad Rey Juan Carlos. El objetivo es evaluar si la toma de decisiones basada en datos puede influir significativamente en el producto y en la experiencia de juego de productos que no son de las mismas dimensiones que grandes y medianas empresas, si no también para equipos independientes.

## 1.1 Objetivos del proyecto

### 1.1.1 Objetivo general

Como se ha descrito en la introducción previa, el videojuego se encuentra en un punto en el que es posible recabar datos, tanto de los jugadores como de las diferentes plataformas utilizadas en el desarollo (véase la subsección 1.4.2). Sin embargo, actualmente el equipo no cuenta con sistemas de decisión propios que le permitan obtener métricas y analizar diferentes propiedades del videojuego, con el fin de mejorar la experienca del usuario.

Por ende, el presente Trabajo Fin de Máster tiene como objetivo comprender la función de la toma de decisiones basadas en datos e incluir sistemas de decisión en un entorno de desarrollo independiente basado en micromecenazgo, a través del análisis de la actividad de los usuarios a varios niveles, con el fin de mejorar la experienca de juego de los usuarios y apoyar las decisiones de negocio.

### 1.1.2 Objetivos específicos

Desglosando el objetivo general, el trabajo se puede dividir en cuatro metas principales:

1. Comprender métricas que describan la interacción de un usuario con el videojuego.
2. Definir sistemas de toma de decisiones con datos captados de usuarios reales.
3. Cómo utilizar los resultados de los dos puntos anteriores para la planificación del desarollo y la evolución del videojuego, siendo el propósito garantizar el beneficio y la satisfacción de los usuarios.
4. Definir una arquitectura que sirva de punto de partida para el análisis de comportamiento de los usuarios.

Cada punto gira en torno a la aplicación de la toma de decisiones en desarrollo de un videojuego independiente. A la finalización del proyecto se espera conseguir métricas que permitan conocer la interacción del usuario con el producto, de tal forma que se puedan obtener conclusiones para mejorar la experiencia de juego.

## 1.2 Planificación temporal

Para planificar el proyecto, se ha utilizado la herramienta Notion, con la que se pueden generar tareas para tener un seguimiento de lo que está hecho y lo que falta por hacer. La herramienta es tratada más adelante en la sección 2.8.

El diagrama 1.5 muestra gráficamente cuánto ha llevado la realización de este Trabajo de Fin de Máster. Ha tenido una duración de 10 meses, empleando 2 ó 3 horas de trabajo diarias.

Las tareas que más tiempo consumieron fueron aquellas teóricas ("Planteamiento y Objetivos", "Definición del Modelo Entidad Relacion (ER)", "Investigación de Análisis de Datos en los Videojuegos") debido a que era importante sentar las bases para llevar a cabo las tareas prácticas.

## 1.3 Estructura de la memoria

El presente trabajo de Fin de Máster ha seguido una estructura tradicional, a excepción de que se han añadido secciones en cada capítulo en torno al producto *Battlecore Robots*, y el impacto de este trabajo a nivel de negocio:

- El primer capítulo (1) engloba la introducción al proyecto, los objetivos del mismo, la planificación temporal y se da contexto al videojuego, objeto de este experimento.

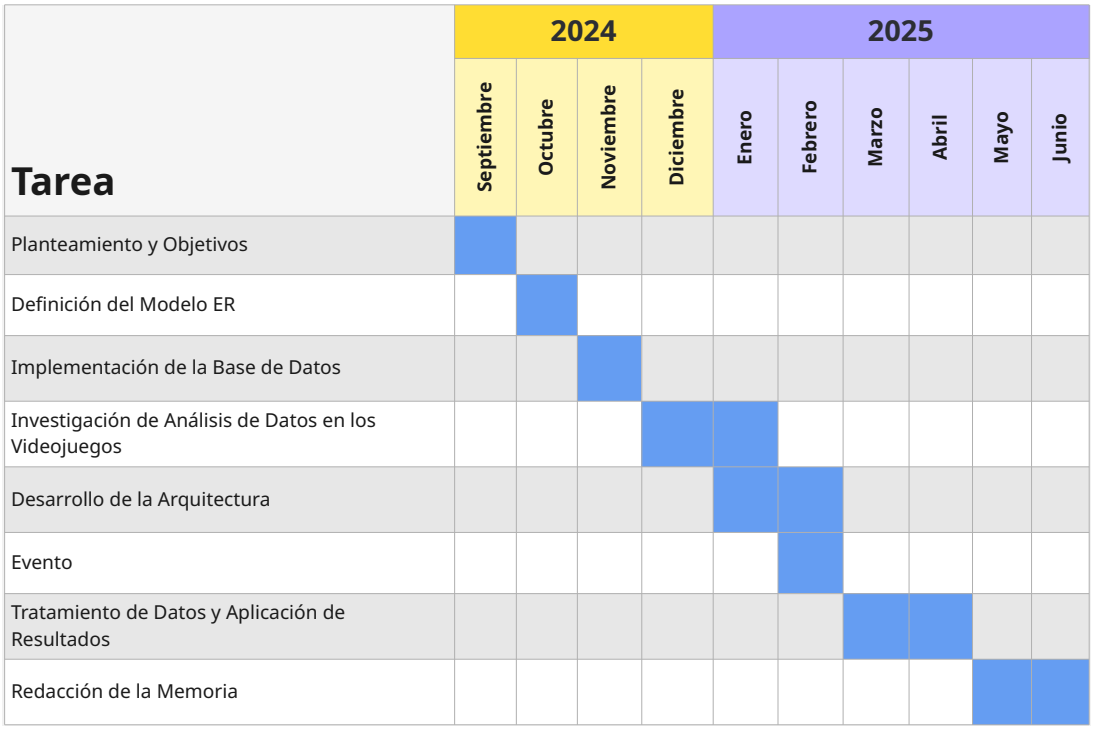


Figura 1.5: Diagrama GANTT de la duración de la realización del Trabajo de Fin de Máster.

- El siguiente capítulo (2) contiene las tecnologías utilizadas para el desarrollo del experimento y su posterior análisis.
- En el capítulo 3 se describe el estudio de métricas y el desarrollo técnico requerido para la realización del trabajo.
- En el capítulo 4 se recogen los resultados del experimento y los sistemas de decisión generados.
- Por último, en el capítulo 5 se presentan las conclusiones del proyecto, haciendo hincapié en las implicaciones que ha tenido a nivel de negocio; además de las lecciones aprendidas y trabajos futuros relacionados con el experimento.

## 1.4 Videojuego

*Battlecore Robots* es un videojuego de peleas 3D con robots personalizables desarrollado por estudiantes de la Universidad Rey Juan Carlos del grado Diseño y Desarrollo de Videojuegos. Los integrantes de dicho equipo son:

- Roberto García Martín.
- Manuel Mantecón Polo.

- Jacques David Meyns Villaldea.
- Jose Ignacio Pintado Murillo.
- Sofía de Vega Giménez.


Figura 1.6: Portada de *Battlecore Robots*.

El desarrollo comenzó en la asignatura de cuarto curso "Juegos para Web y Redes Sociales" en la que había que desarrollar un videojuego para la plataforma web "itch.io". La idea surge al identificar que había interés en Internet por la saga de videojuegos de los 2000, *Custom Robo* [12].

Se hizo un estudio de mercado y se detectó que muchos usuarios esperaban una nueva entrega de la saga, por lo que se optó por crear un videojuego con nuevas funcionalidades, manteniendo la estética de aquella época, y por tanto, apelando a la nostalgia de los usuarios. La figura 1.7 muestra la carátula del primer lanzamiento de dicha saga. El último lanzamiento tuvo lugar en 2006.

Se finalizó una primera versión de *Battlecore Robots* en diciembre de 2021. Fue evaluada por los profesores, y a los pocos días se quitó la página de "itch.io", para ocultar la idea de cara al público.

Desde la definición del juego en la asignatura, el objetivo era publicar un videojuego en Steam [43], la plataforma más grande de videojuegos de ordenador; junto a una campaña de Kickstarter, una plataforma de financiación colectiva donde publicar proyectos y solicitar apoyo financiero de otras personas a través de donaciones. También se quería realizar un "modo historia", con diferentes personajes y diálogos entre ellos, en los que el jugador puede envolverse y combatir para conseguir piezas y mejorar a su robot.


Figura 1.7: Portada del primer juego de la saga *Custom Robo* [12].


Figura 1.8: Pantalla de título de la primera versión del juego.

Al finalizar la asignatura, el juego contaba con robots y una lista de piezas para la personalización de los mismos. Después, se decidió enfocar los Trabajos de Fin de Grado de los integrantes del equipo en diferentes apartados del juego, para comenzar el desarrollo del mismo con unas bases definidas:

- Roberto García Martín: Herramientas con tecnologías en la nube.
- Manuel Mantecón Polo: Producción.

- Jacques David Meyns Villaldea: Inteligencia Artificial.
- Jose Ignacio Pintado Murillo: Mecánicas de juego.
- Sofía de Vega Giménez: Diseño 3D y modularización de piezas.

Desde entonces, el proyecto ha pasado por diferentes fases, cada una con un objetivo concreto, mostrados en la figura 1.9:


Figura 1.9: Línea temporal del desarrollo de *Battlecore Robots*.

1. **Cambio visual y de mecánicas de juego**: se cambió el estilo de los robots y se redefinieron los tipos de piezas.
2. **Demo pública en "itch.io"**: se publicitó el juego en redes sociales con el fin de lanzar la primera versión pública gratis y que fuese probada por clientes reales.
3. **Primeros eventos, "Guerrilla" y "Gamegen"**: el juego apareció por primera vez en eventos presenciales. Se llevó a los eventos "Guerrilla", organizado por la Universidad Complutense de Madrid, y "Gamegen", organizado por la Universidad Rey Juan Carlos.
4. **Se crea la página de Steam y se publica el primer trailer**: el juego aterrizó en la plataforma, con una buena acogida de los usuarios, teniendo la primera semana 2000 "avísames".
5. **Redefinición del proyecto**: se optó por contactar a *Publishers* (empresas encargadas de publicar, promocionar y distribuir videojuegos a cambio de financiación). Para ello, se planteó un cambio de la estructura de juego con fin de presentar una propuesta única y llamativa a las empresas. En vez del "modo historia" se planteó que el juego contaría con un modo "Roguelite", un modo de juego en el que tienes que derrotar a enemigos por rondas continuamente, mejorando al robot entre rondas.

6. **Patreon**: debido a que no se consiguió financiación por la crisis de 2024, se plantea hacer una campaña de Patreon (tratada en detalle la subsección 1.4.2).

A fecha de la redacción de este trabajo, cuenta con las siguientes estadísticas:

1. **Twitter**: La cuenta oficial del proyecto cuenta con aproximadamente 3300 seguidores.
2. **Steam**: Más de 10.000 usuarios tienen el juego en sus listas de "avísames" (*wishlist*). Una *wishlist* es un dato que sirve para identificar el interés del juego en su lanzamiento, ya que la plataforma Steam manda un correo a los clientes que tengan un producto en su lista de avísame cuando sale a la venta.
3. **Discord**: Discord es una plataforma gratuita de mensajes; destaca por los canales, espacios de varios usuarios para comunicarse a tavés de texto o voz. El servidor del juego tiene 512 usuarios. (*Por razones de privacidad, no se muestra el número exacto*).

A fecha del inicio de Trabajo de Fin de Máster (TFM), el contexto de *Battlecore Robots* es el siguiente:

- El equipo estaba planteando crear la página de Patreon para financiar el videojuego.
- Se estaban creando exclusivamente piezas, para acabar el contenido horizontal y mínimo de un juego de peleas 3D.
- Existe una comunidad activa en Discord con la que se mantiene interacción regular. Los usuarios más participativos podrían considerarse como *clientes clave*, dado su nivel de implicación e interés en el desarrollo del proyecto.
- Se publica en redes sociales contenido, para así captar y atraer a nuevos usuarios y conseguir avísames en Steam.
- Se participa en eventos, consiguiendo así feedback y contactos.
- Los usuarios interesados son de múltiples países, especialmente de Japón y Estados Unidos.

El diagrama de dominio de la figura 1.10 muestra gráficamente en qué medios y actividades se encuentra el desarrollo del videojuego.

A continuación, se profundizará en las siguientes subsecciones acerca del videojuego, con el objetivo de contextualizar el proyecto y explicar sus diferentes componentes, facilitando así la comprensión de este trabajo:

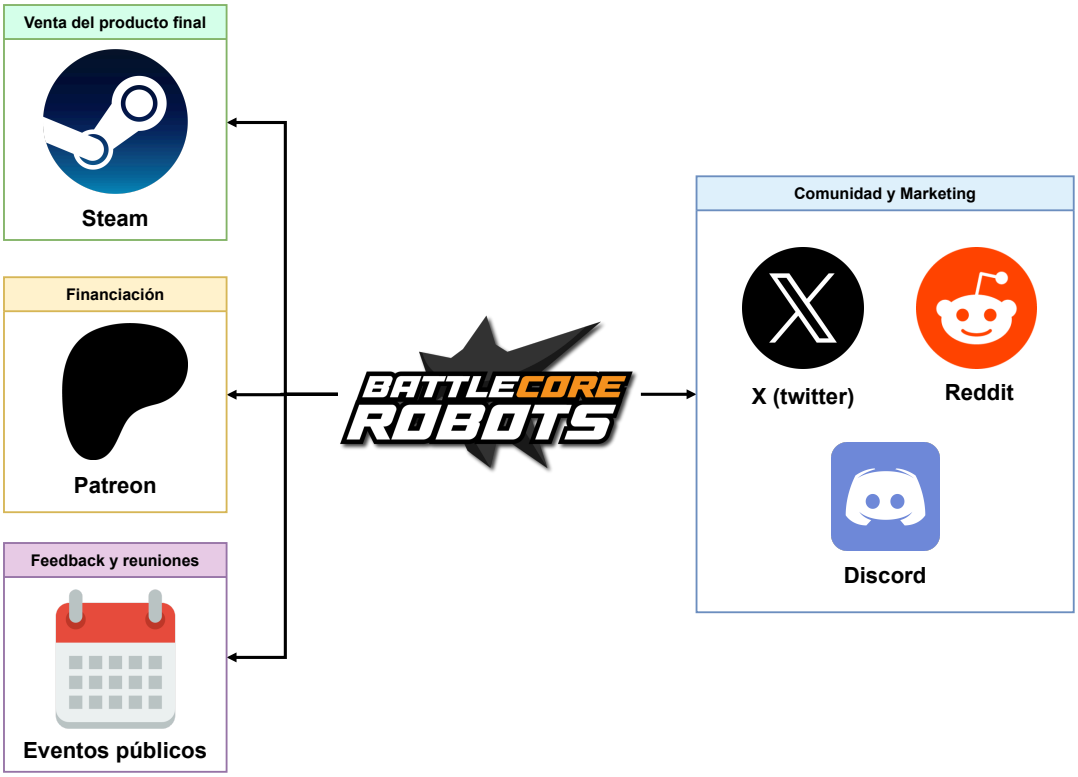


Figura 1.10: Diagrama dominio del contexto actual del desarrollo del videojuego.

- Eventos.
- Comunidad y Patreon.
- Características del videojuego.

### 1.4.1 Eventos

Una práctica común en el desarrollo de videojuegos es presentar un producto en un evento con varios fines, entre ellos:

1. Dar a conocer el videojuego: mediante personas visitantes, entrevistas de la televisión, redes sociales del evento, entre otras.
2. Conseguir opiniones de usuarios: para así mejorar la experiencia de juego, descubrir errores, escuchar ideas...
3. Entrevistas con inversores: los eventos suelen aunar tanto desarrolladores como empresas interesadas en invertir.

La figura 1.11 muestra los eventos con mayor número de asistentes según los estudios entrevistados en el "Libro Blanco de Desarrollo de Videojuegos Español" [13].

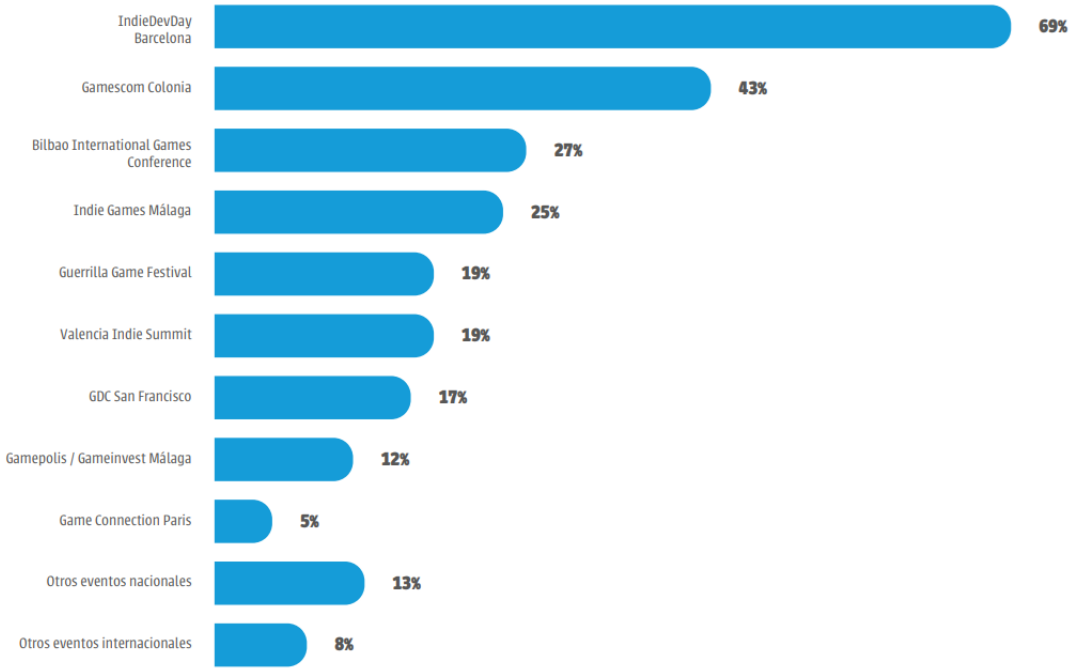


Figura 1.11: Eventos a los que han asistido los estudios españoles en 2023 [13].

Respecto a *Battlecore Robots*, el equipo ha asistido a múltiples eventos, con el fin de mejorar el producto y conseguir contactos:

1. Guerrilla Game Festival 2022.
2. Gamegen 2023.
3. Gamegen 2024.
4. Guadalindie 2024.
5. Indie Dev Day Madrid 2024.

A lo largo del desarrollo de este Trabajo de Fin de Máster, tuvo lugar en el campus de Móstoles de la Universidad Rey Juan Carlos el evento "Gamegen" [20]. Se trata de un evento en el que profesionales del sector realizan ponencias acerca del estado de la industria, tecnologías y clases magistrales.

También, cuenta con un espacio en el que se exponen videojuegos, dando oportunidad a los oyentes de las ponencias a que prueben nuevos productos mientras que los desarrolladores recogen opiniones de los mismos.

Este evento representa una oportunidad para el desarrollo de este proyecto, ya que se abre la posibilidad de obtener muestras de usuarios reales y su interacción con el videojuego, creando así métricas con datos reales para generar los sistemas de decisión.

### 1.4.2 Comunidad y Patreon

El videojuego cuenta con una comunidad involucrada, tal como se ha introducido anteriormente, porque no existe un producto parecido desde el año 2006.

De tal manera, los usuarios interesados se involucran y prestan atención en el desarrollo. En múltiples ocasiones, han sido los propios interesados los que han propuesto financiar económicamente el videojuego.

Este factor abría una oportunidad para realizar un Kickstarter, como se ha mencionado al principio de esta sección. Sin embargo, se analizó qué cosas serían necesarias y el equipo acordó paralizar dicho plan. Entonces, aprovechando la comunidad y el compromiso de los usuarios, se planteó realizar el Patreon.

Patreon [5] es una plataforma de micromecenazgo en la que hay "creadores", que son personas que ofrecen un servicio o producto con una suscripción mensual, y "miembros". Un miembro es alguien suscrito a uno de los servicios. Las suscripciones pueden ser de diferentes cargos y son un medio para que los miembros apoyen a los creadores.

A fecha de este proyecto, la plataforma cuenta con más de 8 millones de miembros y más de 250.000 creadores [33]. El 6% de los creadores de Patreon se dedican al desarrollo de videojuegos [46].

Se decidió crear cuatro tipos diferentes de suscripción. Una suscripción de más coste cuenta con más ventajas que aquellas con menos coste. En términos generales, cada suscripción se basa en lo siguiente:

1. Suscripción de apoyo: poca cantidad monetaria, para aquellos que quieran apoyar la idea pero no involucrarse.

2. Suscripción con el juego: teniendo acceso a parte del contenido hasta la fecha del videojuego.

3. Suscripción para involucrarse en el desarrollo: teniendo acceso a contenidos multimedia del trabajo en proceso de cada mes.

4. Suscripción de mucho apoyo económico: una suscripción para los que buscar donar más dinero.

La segunda suscripción cuenta con una versión del juego disponible para usuarios, lo que implica realizar una selección de todo el contenido generado hasta la fecha. En caso de incluir todo el contenido, el equipo se quedaría sin margen de error para generar contenido en los próximos meses.

En vista de que ya se contaba con una comunidad y de que, en plena fase de desarrollo de la página de Patreon, se iba a asistir al evento "Gamegen" para mostrar el videojuego, se decidió enfocar este trabajo en el uso de sistemas de decisión para apoyar al equipo en la planificación de los contenidos iniciales de la versión del juego de Patreon.

### 1.4.3 Características del videojuego

A continuación, se definirá en líneas generales cómo es el videojuego *Battlecore Robots* para facilitar la comprensión de la memoria. Como se ha mencionado, se trata de un juego de combates en un espacio 3D en el que participan entre dos y cuatro robots.

Los combates tienen un tiempo establecido a elegir por el usuario. Actualmente, la duración puede ser de: un minuto, minuto y medio, dos minutos, dos minutos y medio, tres minutos o sin límite de tiempo. La condición para ganar un combate es derrotar a los oponentes (bajar sus puntos de vida a cero) o que se acabe el tiempo y ser el jugador con más vida restante. Cada robot puede ser controlado por una persona o el ordenador. La captura de pantalla de la figura 1.12 muestra un combate entre dos jugadores.

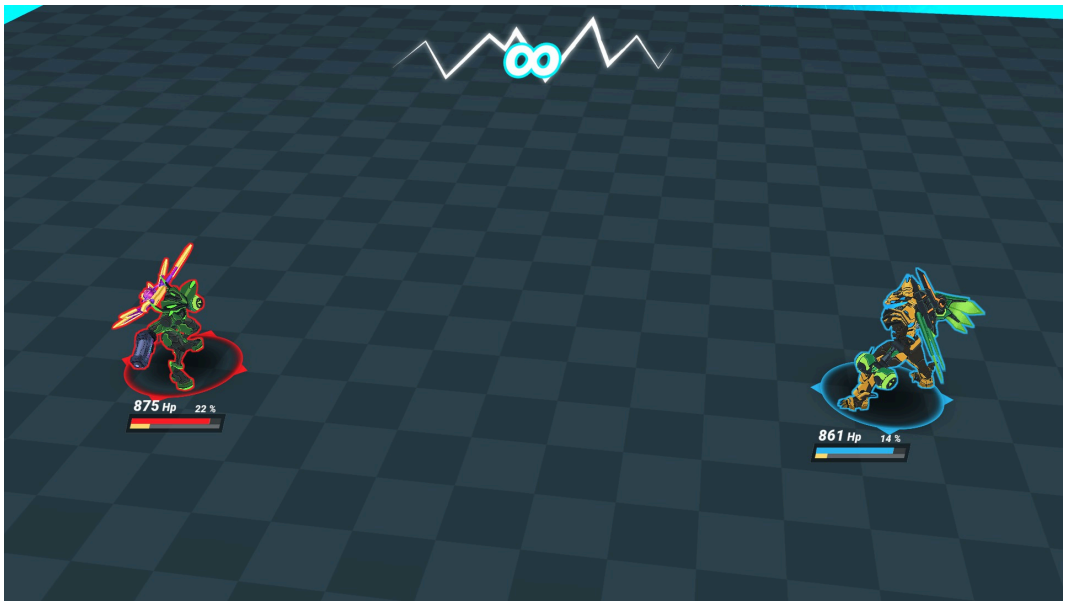


Figura 1.12: Combate entre dos jugadores.

Los robots son personalizables y antes de un combate hay que elegir con qué piezas combatir y en qué arena pelear. Hay seis partes que el jugador puede cambiar del robot;

mostradas visualmente en la figura 1.13:

1. **Robot**: define las características principales para el combate. Cada robot tiene una habilidad especial única conocida como *Overdrive*. Hay siete clases de robots según su tamaño y son *Mini*, *Normal*, *Goliath*, *Duelist*, *Star Glider*, *Blitzer*, *Mirage*.
2. **Pistola** (***Gun***): dispara balas para atacar al enemigo a distancia.
3. **Bomba** (***Bomb***): crea una explosión en el mapa. Cada bomba tiene un tamaño de explosión diferente, algunas incluso con efectos únicos.
4. **Cuerpo a cuerpo** (***Melee***): para atacar al enemigo a corta distancia.
5. **Vaina** (***Pod***): lanza un dron que reacciona y hace daño si el oponente está cerca.
6. **Chip**: modifica ligeramente propiedades del robot, se inserta en la nuca de los robots.

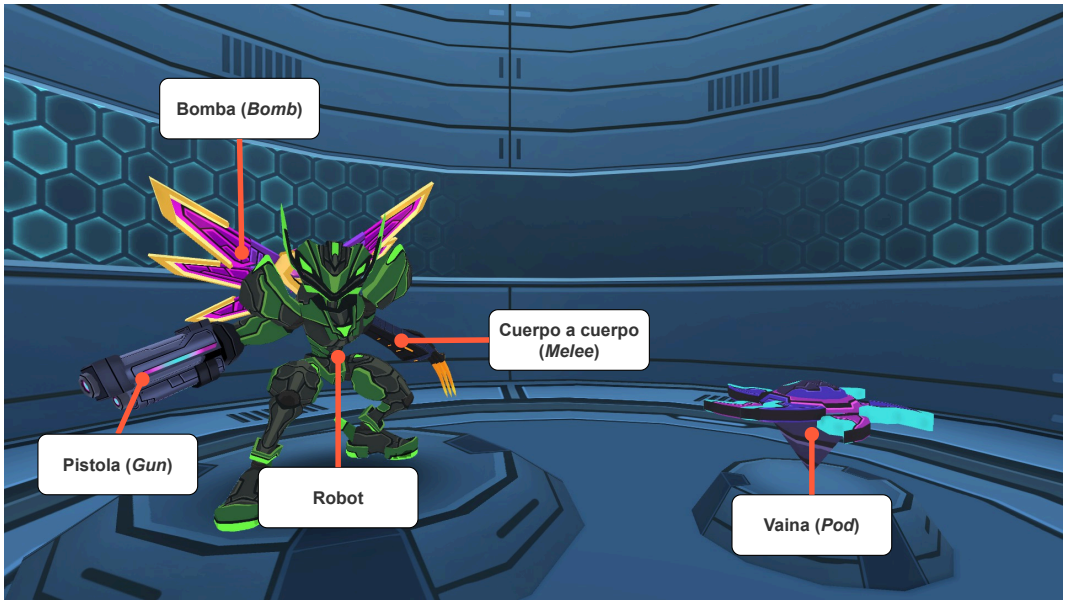


Figura 1.13: Robot y sus piezas.

El juego tiene como objetivo proporcionar dos formas de jugar al usuario. La primera es combates contra otras personas o el ordenador, donde el objetivo del jugador es mejorar su habilidad y conocimiento del juego para ser mejor que el resto.

La segunda manera que tendrá un cliente final de interactuar con el juego es a través de un "modo historia", actualmente en desarrollo. La premisa de este contenido es dar un espacio al usuario en el que vaya obteniendo las diferentes piezas disponibles a través de combates y retos dentro del mundo de *Battlecore Robots*.

# Capítulo 2

# Estado del arte

Las tecnologías empleadas abarcan tres campos:

1. Desarrollo técnico.
2. Análisis y visualización de datos.
3. Organización y gestión del proyecto.

Para cada una, se tratará por qué se eligió y su papel en el proyecto. Se han optado por tecnologías ampliamente utilizadas en la industria o recomendadas por el tutor.

## 2.1 Unity y C#

Unity [34] es el programa utilizado para desarrollar *Battlecore Robots*. Es un motor de videojuegos multiplataforma (permite compilar los archivos para que funcionen en diferentes plataformas, como Nintendo Switch, Play Station, ordenador) 2D y 3D lanzado en 2005. Fue un programa creado por 2 personas en un foro de internet hablando sobre la librería gráfica OpenGL [30].

Actualmente, Unity está apostando por tecnologías del momento como XR (realidad mixta), multijugador y *Liveops* (gestión continua de operaciones, muy común en juegos en línea) [42].

Unity emplea la plataforma de desarrollo .NET.

.NET es una plataforma de Código Abierto para desarrollar aplicaciones que se pueden ejecutar en cualquier sistema operativo.

Unity utiliza C# para compilar el código. C# es un leguaje fuertemente tipado [45] y es parte de la familia de lenguajes C, con sintaxis muy parecida. El lenguaje destaca por la administración automática de memoria, evitando así tener que liberar memoria de manera manual en el código.

Además, C# cuenta con NuGet [29], que es un gestor de paquetes .NET, con el que se podrán añadir funciones al proyecto de Unity que estén basadas en .NET.

## 2.2 Visualización de métricas

Para visualizar métricas, se va a utilizar el sistema de creación de documentos Quarto [35], utilizando bloques de código Python [55]. Se tratará cada tecnología por separado.

### 2.2.1 Python y Astral UV

Python es un lenguaje de programación interpretado (se ejecuta línea a línea por la máquina, no hay compilación de código) de alto nivel, con el que se puede desarrollar una variedad de aplicaciones, desde análisis de datos hasta automatización y desarrollo web.

Se utilizó debido a la gran variedad de librerías que ofrece. Una librería (o paquete en terminología de Python) es una colección de códigos con métodos que amplían la funcionalidad del lenguaje.

El papel de Python en el proyecto es la lectura de datos y la creación de métricas.

Para ello, se utilizaron las siguientes librerías:

1. `psycopg2`: Permite la conexión y ejecución de consultas en bases de datos PostgreSQL desde Python.
2. `pandas`: Librería para la manipulación y análisis de datos estructurados en forma de tablas (*DataFrames*).
3. `matplotlib.pyplot` y `matplotlib.colors`: Utilizadas para la creación de gráficos y personalización de colores.
4. `seaborn`: Biblioteca basada en `matplotlib` para visualizar estadísticas más atractivas y fáciles de definir.

5. `itables`: Permite mostrar tablas interactivas en entornos interactivos, como `HTML` que es el caso de este proyecto.

6. `numpy`: Librería para operaciones matemáticas.

7. `enum`: Proporciona soporte para crear enumeraciones, es decir, grupos de nombres simbólicos vinculados a valores constantes.

8. `yaml`: Permite leer y escribir archivos de configuración en formato YAML. Se usará para guardar secretos con los que acceder a la base de datos.

9. `os`: Se usará para acceder a archivos, en concreto el documento YAML anteriormente mencionado.

10. `math`: Biblioteca estándar para operaciones matemáticas básicas como raíces, logaritmos o funciones trigonométricas.

Para ejecutar Python, se ha utilizado Astral UV.

Astral UV [50] es un gestor de proyectos de Python e instalador de paquetes del lenguaje de la compañía Astral. Astral es el creador de Rust, un lenguaje de programación compilado que destaca por su alto rendimiento. Astral UV está escrito en Rust.

La razón por la que se escribió en Astral UV es que el proyecto de redacción de datos trabaja con bloques de Python. Al utilizar Astral UV, la previsualización de los cambios ha sido más rápida que si se hubiesen usado otros gestores de Python.

### 2.2.2 Quarto

Quarto es una herramienta de código abierto diseñada para redactar documentos científicos y técnicos. Está orientada a crear artículos de análisis de datos que se puedan compartir y reproducir.

Permite integrar código ejecutable de diferentes lenguajes de código (como Python o R [36]), lo que facilita combinar resultados y visualizaciones junto con explicaciones en un mismo documento.

Se utilizó debido a la flexibilidad a la hora de configurar el documento. En el contexto del proyecto, Quarto es la plataforma de documentación para analizar los datos extraídos y tratar los sistemas de decisión obtenidos.

## 2.3 Entorno de desarrollo videojuego: Jetbrains Rider

Jetbrains Rider [38] es un Integrated Development Enviroment (Entorno de Desarrollo Integrado) (IDE) dedicado al desarrollo en .NET y creado por la compañía checa Jetbrains.

De cara a este proyecto académico, es el IDE adecuado ya que proporciona análisis de código, herramientas de depuración para programas de .NET, inclusión de paquetes NuGet y cuenta con herramientas propias para desarrollar en Unity.

Su rol en el proyecto es desarrollar las diferentes herramientas presentes en el juego para poder mandar datos desde el juego a una base de datos, además de para crear herramientas internas en el motor cuando sea necesario.

## 2.4 Entorno de desarrollo datos: Visual Studio Code

Visual Studio Code [53] es un IDE desarrollado por Microsoft y lanzado en 2015.

Destaca debido a que es muy personalizable gracias a las extensiones. Además, cuenta con depuración de aplicaciones y scripts, integración con control de versiones y permite programar con diferentes lenguajes.

Se ha utilizado en el proyecto para redactar este documento, así como para crear documentos de análisis de datos con Python y Quarto, ya que Visual Studio Code permite trabajar con varios lenguajes en un mismo proyecto.

## 2.5 Base de datos: PostgreSQL y pgModeler

PostgreSQL es una base de datos relacional [23] de código abierto que extiende del lenguaje SQL [54]. Fue desarrollado en la Universidad de California en 1986.

Se ha escogido PostgreSQL entre otras bases de datos por su arquitectura y porque es una base de datos funcional en la gran mayoría de sistemas operativos, lo cual supone una ventaja para desarrollar tanto en entornos de prueba (un ordenador doméstico) como en entornos de producción (alojar una base de datos en la nube).

Junto a PostgreSQL se ha utilizado la herramienta pgModeler [40] para definir y modelar la base de datos. También permite exportar esas bases como código SQL, administrar bases de datos y cuenta con una consola para realizar operaciones en una base de datos.

## 2.6 RabbitMQ

RabbitMQ [37] es un sistema de mensajería de código abierto que implementa el protocolo AMQP (*Advanced Message Queuing Protocol*) que lanzó su primera versión estable en 2005. Está diseñado para permitir la comunicación entre procesos, servicios o aplicaciones de forma asíncrona, desacoplando los componentes del sistema mediante el intercambio de mensajes a través de colas.

Se ha implementado en el proyecto para distribuir las tareas de captación de datos. Cumple un papel esencial en el sistema: conecta el videojuego con un programa que inserta los datos recibidos a través de una cola de mensajes, lo que permite que las operaciones de mayor duración se realicen en segundo plano, evitando que el usuario tenga que esperar a que finalicen.

## 2.7 Docker Engine

Docker Engine [15] o Docker es una plataforma de código abierto lanzada en 2013 con la que se puede crear, implementar, ejecutar y gestionar aplicaciones en "contenedores". Un contenedor en Docker es un espacio virtual de ejecución de procesos (un entorno informático que actúa como un ordenador independiente dentro de otro ordenador), modular y flexible [2]. De esta manera, con Docker puedes lanzar, pausar y trasladar aplicaciones entre entornos de forma sencilla. Actualmente, es un estándar para la ejecución de servicios en nube.

Para este proyecto, se encapsulará la herramienta RabbitMQ en un contenedor ejecutado en un ordenador de sobremesa. Se ha elegido esta tecnología porque permite capturar datos mediante colas de mensajes en un entorno local, con la posibilidad de migrar fácilmente a un servicio en la nube en el futuro.

## 2.8 Organización: Miro y Notion

Para organizar el proyecto se han utilizado las aplicaciones Miro y Notion.

Miro [4] es una herramienta SaaS (*Software as a Service*, Software como Servicio) online para desarrollar flujos con el fin de gestionar proyectos. Anteriormente era conocida como RealtimeBoard y fue fundada en Rusia en 2011.

Miro se ha utilizado para crear esquemas con los que definir y comunicar diseño y

definiciones del experimento entre el tutor y el alumno periódicamente.

Notion [47] es un software de gestión de proyectos para tomar notas y generar documentación orientada a la eficiencia y productividad. Notion pertenece a la Startup Notion Labs Inc y fue lanzado en 2018.

Mediante Notion se ha gestionado el seguimiento de las tareas del proyecto, lecturas recomendadas por el tutor y las dudas sobre el proyecto. Concretamente, se partió de una plantilla para escrituras de tesis [57].

## 2.9 Git y GitHub: Control de versiones

Git [22] es un sistema de control de versiones diseñado para gestionar los cambios en archivos de manera eficiente. Lanzado en 2005, permite registrar el historial de modificaciones realizadas, creando versiones del contenido a lo largo del tiempo.

Git no guarda archivos completos cada vez que se realiza un cambio, sino que almacena snapshots inteligentes del estado del proyecto. Esto facilita volver a versiones anteriores, comparar cambios o colaborar con otras personas sin riesgo de perder información.

Un conjunto de archivos sobre los que se controlan las versiones se conoce como repositorio. En el caso del proyecto, existen dos repositorios:

1. El repositorio del videojuego.
2. El repositorio del proyecto, que contiene el documento de la memoria y los archivos para el tratamiento de datos.

Para almacenar los repositorios en la nube y asegurar que existan copias de seguridad, se utilizó GitHub [7]. Su desarrollo empezó en 2008, y fue ese mismo año cuando se publicó el servicio. A fecha de este documento, es la plataforma de alojamiento de código más grande del mundo y una subsidiaria de Microsoft.

GitHub es una plataforma donde se pueden almacenar y compartir repositorios de forma remota. Cuenta con múltiples funciones para administrar cambios en el código, como la revisión de cambios, control de versiones, colaboración entre desarrolladores y gestión de incidencias.

Ambos repositorios mencionados anteriormente se encuentran alojados en GitHub. El repositorio del videojuego es privado, ya que se trata de un producto comercial.

Se ha utilizado Git y GitHub para poder mantener el código y trabajo a lo largo de los meses que ha durado su desarrollo.

## 2.10 Redacción de la memoria: LaTeX

LaTeX es un sistema de composición tipográfica de código abierto de alta calidad que incluye características especialmente diseñadas para la producción de documentación técnica y científica. Estas características, entre las que se encuentra la posibilidad de incluir expresiones matemáticas, fragmentos de código, tablas y referencias, han hecho que LaTeX se convierta en el estándar *de facto* para la redacción y publicación de artículos académicos, tesis y todo tipo de documentos científico-técnicos.

Se instaló la extensión Latex Workshop [26], que ofrece muchas herramientas para trabajar con LaTeX y para personalizar la manera con la que trabajar en LaTeX.

Se optó por una estructura de multidocumento en la memoria, con la que cada capítulo de la memoria está representado en un archivo independiente, y todos los capítulos cargados en un documento "padre".

Para redactar la memoria, se creó un repositorio en GitHub para añadir control de versiones al documento.

Finalmente, la figura 2.1 engloba las herramientas usadas agrupadas por su papel en el proyecto.

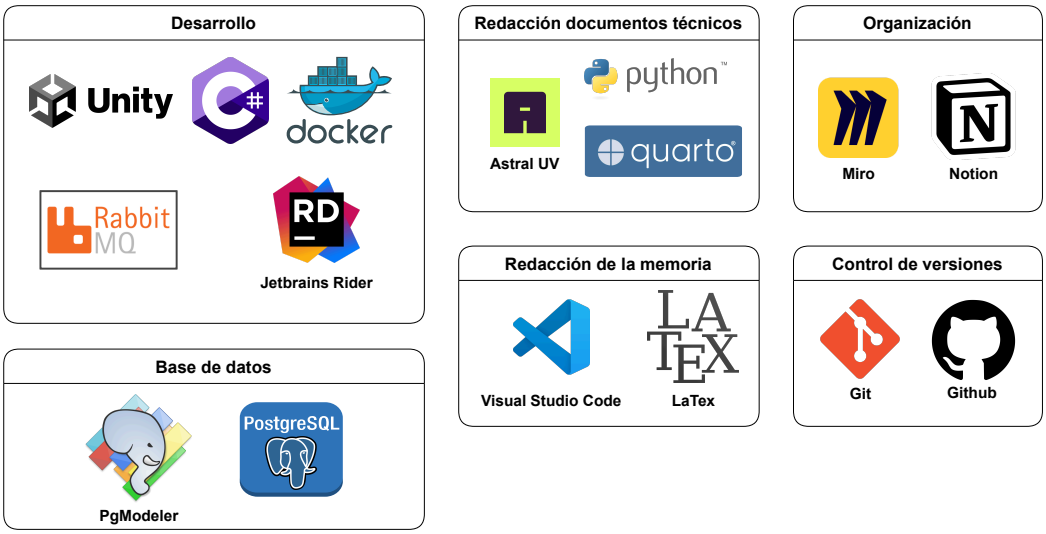


Figura 2.1: Esquema de las herramientas usadas agrupadas por rol para la realización del Trabajo de Fin de Máster.

# Capítulo 3

# Diseño e implementación

En el tiempo en el que se ha desarrollado el Trabajo de Fin de Máster, la decisión de negocio que surgió en el equipo fue abrir el juego en la plataforma de Patreon tal y como se explica en la sección 1.4.

Así, el diseño del experimento tiene como objetivo incorporar la toma de decisiones basada en datos en el proyecto.

El experimento se ha orientado hacia la plataforma Patreon, y sus resultados se utilizarán para definir el contenido del juego en la versión del primer mes. Además, la arquitectura obtenida servirá como base para trabajos futuros, centrados en el análisis de comportamiento de los usuarios y la toma de decisiones en fases posteriores del desarrollo.

Para tener conocimiento de los usos en la industria y conseguir una base de conocimiento con la que poder empezar a diseñar el experimento, se utilizó el libro *Game Analytics: Maximizing the Value of Player Data* [39], un compendio de artículos sobre análisis de datos de videojuegos desde el punto de vista empresarial y técnico por varias empresas del sector, entre ellas EA y Microsoft.

De los conceptos clave obtenidos de la lectura, los más importantes para el experimento han sido:

- A la hora de tratar datos, hay que diferenciar entre el juego como experiencia jugable (*game as a product*) y el juego como producto rentable (*game as a project*).
- Qué es el flujo del desarrollo del conocimiento y cómo afecta al desarrollo de videojuegos.
- Las métricas de usuario se dividen en tres: a nivel de consumidor, de comunidad y de jugabilidad (*Customer, Community, Gameplay Metrics*).

- La analítica de videojuegos tiene dos propósitos: asegurar una buena experiencia de usuario para mantener jugadores y asegurar que el ciclo de monetización genera beneficio.
- La selección de atributos a tratar ha de tener en cuenta los puntos de vista de todo lo que envuelve el desarrollo.

De los puntos anteriores, resulta especialmente relevante el primero, ya que es fundamental en este contexto. Dado que el juego se publicará en Patreon y estará en constante actualización, además de ser probado mensualmente por los usuarios, la experiencia y el producto van de la mano. Esto será relevante, ya que afectará a cómo será el procedimiento de trabajo mes a mes para poder asegurar una buena experiencia de juego y aumentar las posibilidades de que el producto sea rentable.

Como resultado, se definió un experimento de recogida y análisis de datos con el fin de cumplir los siguientes propósitos:

- Decidir contenido de piezas a incluir en el primer mes del lanzamiento en Patreon.
- Poder analizar la interacción del jugador con el juego.
- Crear sistemas de decisión a incluir en el flujo de desarrollo.
- Crear una la arquitectura de recogida de datos. Esa arquitectura será la base para integrarla en un futuro con servicios en la nube y analizar datos de los suscriptores de Patreon.

El experimento se probó en un evento en la Universidad Rey Juan Carlos, la "Gamegen" [20] (véase la subsección 1.4.1). Se decidió participar en dicho evento, ya que podría ser una oportunidad para crear una arquitectura capaz de obtener métricas del videojuego que apoye a la decisión del contenido inicial de la primera versión del videojuego para los suscriptores de Patreon.

## 3.1 Marco teórico: Métricas

Antes de seleccionar las métricas a tratar, se estudió el papel que tendrían dichas métricas en el desarrollo. Para ello, se estudió el Proceso de Descrubrimiento de Conocimiento (*Knowledge Discovery Process*) de Berry y Linoff [10], en el que se propone que la recogida y analítica de datos ha de tener un rol importante en las decisiones del negocio. El flujo trata de incluir en el proceso iterativo el análisis de datos y entender (ganar conocimiento) de nuevos aspectos del producto.

La imagen 3.1 muestra dicho proceso, orientado a la telemetría de usuarios [8]. Dicho proceso tiene cabida en el desarrollo del juego, ya que se persigue crear un sistema que apoye a la toma de decisión mes a mes.


Figura 3.1: Proceso del Descubrimiento de Conocimiento [8].

Así pues, para elegir las métricas, se estableció el siguiente criterio: la métrica ha de poder integrarse en un flujo de Descubrimiento del Conocimiento del desarrollo y aportar conocimiento de la experiencia del usuario o del apartado monetario del producto mes a mes.

Así, se definió un flujo de trabajo que incorpora el tratamiento de datos como apoyo al desarrollo mensual de nuevas versiones del videojuego para los miembros de Patreon (figura 3.2). La figura tiene las siguientes partes:

- Plantear y definir objetivos: tanto a nivel de producto (por ejemplo, una nueva mecánica incluida en el juego), como de empresa. Esto recoge metas fuera del videojuego en sí, como por ejemplo análisis de datos, marketing o trabajar en la retención de usuarios, entre otras.

- Acción: aquello que lleva a cumplir el objetivo. Por ejemplo: programar, organizar un evento para probar una nueva característica del juego y recoger datos concretos.
- Suscriptor prueba la nueva versión: en esta etapa del desarrollo, ya existe un actuable y un objetivo con una nueva versión del juego o actuable que probará un suscriptor de Patreon.
- Estudiar satisfacción y retención: aquí se utilizarán los sistemas de decisión planteados para evaluar los objetivos del mes y sacar conclusiones que apoyen tanto al diseño del producto como jugable y rentable.

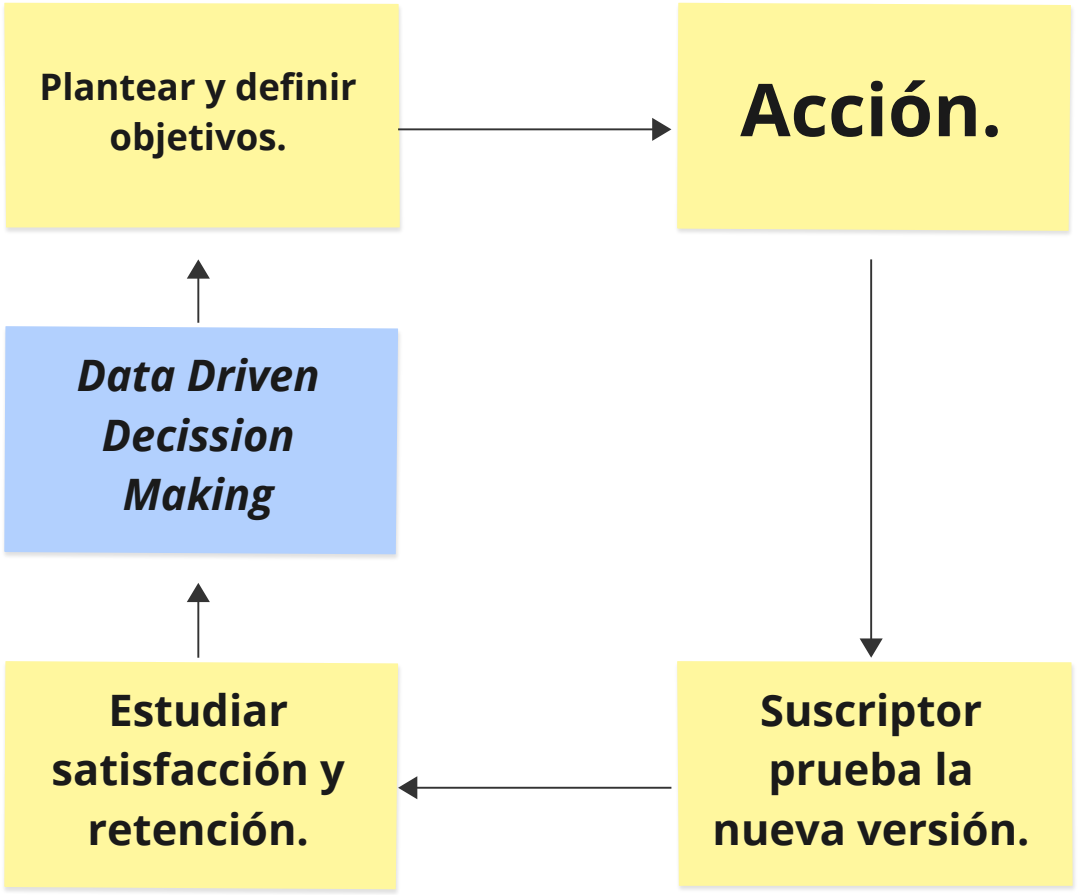


Figura 3.2: Flujo de desarrollo con Descubrimiento de Conocimiento integrado para el desarrollo con Patreon.

Teniendo en cuenta ya el papel de las métricas se realizó una primera definición recogida en la tabla 3.1:

Tabla 3.1: Relación de métricas con sus objetivos y su rol en el flujo de descubrimiento del conocimiento.

| **Métrica** | **Objetivo** | **Rol en el flujo** |
|---|---|---|
| ¿Qué regiones tienen más jugadores? ¿Qué regiones tienen mayor tiempo medio de juego? | Conocer qué regiones están más interesadas. | Generar planes de acción por región. Conseguir evidencia para saber dónde invertir en localización. Si hay regiones con alto tiempo medio pero pocos jugadores, pueden representar una oportunidad. |
| ¿Cuánto tiempo pasa cada jugador en cada pantalla? | Saber cuánto tiempo pasa cada jugador en cada pantalla. | Demasiado tiempo puede indicar que la pantalla no se entiende bien. Permite comparar con el tiempo deseado y tomar decisiones de diseño. |
| Uso de mecánicas por combate. | Saber qué controles son los más usados. | Identificar mecánicas poco utilizadas (posiblemente mal explicadas o poco atractivas) o mecánicas sobreutilizadas (posible desbalance o ventaja percibida). |
| ¿Qué piezas se escogen más y cuáles menos? | Saber si todas las piezas se usan. Identificar las más utilizadas. Analizar si hay correlación entre el uso de piezas y jugadores con más experiencia. | Ver qué piezas resultan más atractivas y decidir dónde enfocar la creación de nuevas piezas. |
| ¿Qué fallos han ocurrido en una partida? ¿Dónde han ocurrido y por qué? | Conocer en qué partes del juego fallan y por qué. | Definir fallos a tratar en la iteración del desarollo. |
| ¿Tiempo de los combates? | Ganar conocimiento sobre la duración de los combates. | Comprobar si es necesario ajustar propiedades de los robots para que los combates tengan una duración más adecuada. |

Continúa en la siguiente página

**Tabla 3.1 – continuación**

| Métrica | Objetivo | Rol en el flujo |
|---|---|---|
| ¿Qué zonas de una arena son más usadas? ¿Qué zonas tienen más tasa de victoria? ¿Qué zonas tienen más tasa de victoria por robot? | Comprender la interacción del usuario con las arenas. | Analizar puntos aislados de las arenas y establecer si es necesario revisar los resultados obtenidos. |
| ¿Dónde muere más un jugador? ¿Dónde se usa más cada mecánica? | Detectar abusos de mecánicas o zonas que sean perjudiquen a un jugador. | Aporta conocimiento sobre el equilibrio del juego y apoya a la decisión de revisar arenas. |
| ¿Qué tipo de jugador hay en el juego? | Conocer preferencias del usuario respecto a los elementos del juego (como: piezas, personajes, arenas). | Analizar si hay alguna patrón en la experiencia de juego que guste y que pueda aprovecharse para hacer marketing. |

## 3.2 Métricas empíricas sobre el comportamiento de usuarios

Una vez claras las métricas y su fin, se planteó el primer experimento. Como se ha mencionado en la introducción de este capítulo, el experimento fue llevado al evento "Gamegen" de la Universidad Rey Juan Carlos. De esa manera, se podía tener una primera prueba de los sistemas de decisión que se querían conseguir. Por ello, de entre todas las métricas planteadas, se seleccionaron aquellas que podían obtenerse durante el evento y que fuesen relevantes en el contexto actual del desarrollo. Finalmente, las métricas elegidas fueron:

- **Uso de mecánicas**: Con esta métrica, podemos visualizar cómo se está entendiendo el juego. Una mecánica poco usada puede implicar que no se ha explicado bien cierta parte del juego o que la usabilidad es pobre.
- **Tiempo medio de un combate**: Este dato es importante porque puede ayudarnos a medir cuánto dura el juego. Con una gráfica de densidad o un diagrama de barras podemos conocer cuanto duran los combates y si el ritmo del juego es adecuado. Una partida demasiado larga puede implicar que la experiencia de juego no es agradable.
- **Piezas elegidas**: El juego gira en torno a los robots y los poderes de cada pieza. Saber qué piezas se han escogido más y utilizar el conocimiento del equipo para compren-

der si es por su estilo visual o por su papel en el juego puede ser clave para el primer objetivo del experimento, elegir el contenido inicial para la versión del juego en Patreon.

- **Piezas con más victorias**: Al ser un juego de lucha, se espera que sea posible ganar con todas las piezas, a la vez que haya piezas mejores que otras para que exista un progreso. Con esta métrica podemos generar un sistema de decisión con el que evaluar el rendimiento de las piezas y decidir si alguna pieza tiene que balancearse (cambiar sus estadísticas para que sea más justa o por el lado contrario menos poderosa).

Al no contar con identificación unívoca de usuarios, se centró el estudio en la interacción del jugador con el videojuego, en lugar de analizar los tipos de jugadores. Además, las personas que iban a probar el juego en el evento no corresponden con el público objetivo, ya que los jugadores no han encontrado el juego por interés propio o a través de internet, si no que se han topado con el juego para probarlo.

Por ende, se decició excluir las métricas centradas en analizar el tipo de jugador. En la sección 5.4 se tratarán trabajos futuros sobre el análisis de usuarios.

A continuación, se definirá la parte técnica para recoger los datos y obtener las métricas planteadas.

### 3.2.1 Diagrama entidad-relacion (ER)

Para definir un diagrama ER con el que almacenar datos (*Data-Driven Decission Making*) se ha tenido en cuenta:

- **Las métricas a recoger**: tener claro las búsquedas que hay que realizar ha facilitado la definición de la base de datos.
- **Futuros usos**: aunque para el evento no se realizara un análisis individual por usuario, la base se ha diseñado considerando que, en el futuro, se recogerán datos de múltiples usuarios únicos.

Como resultado, se necesita una base de datos que represente:

- Información de usuario.
- Información de sesión de juego.
- Información de un combate (duración, mapa seleccionado...).

- Piezas escogidas por los jugadores.
- Interacción del usuario con las mecánicas del juego.

Una vez claros los objetivos, se empleó PostgreSQL junto a PgModeler (veáse la sección 2.5) para diseñar e implementar la base de datos.

A continuación, se mostrará en la figura 3.3 el modelo completo con las entidades definidas y después se tratarán la figuras por partes dado su tamaño.

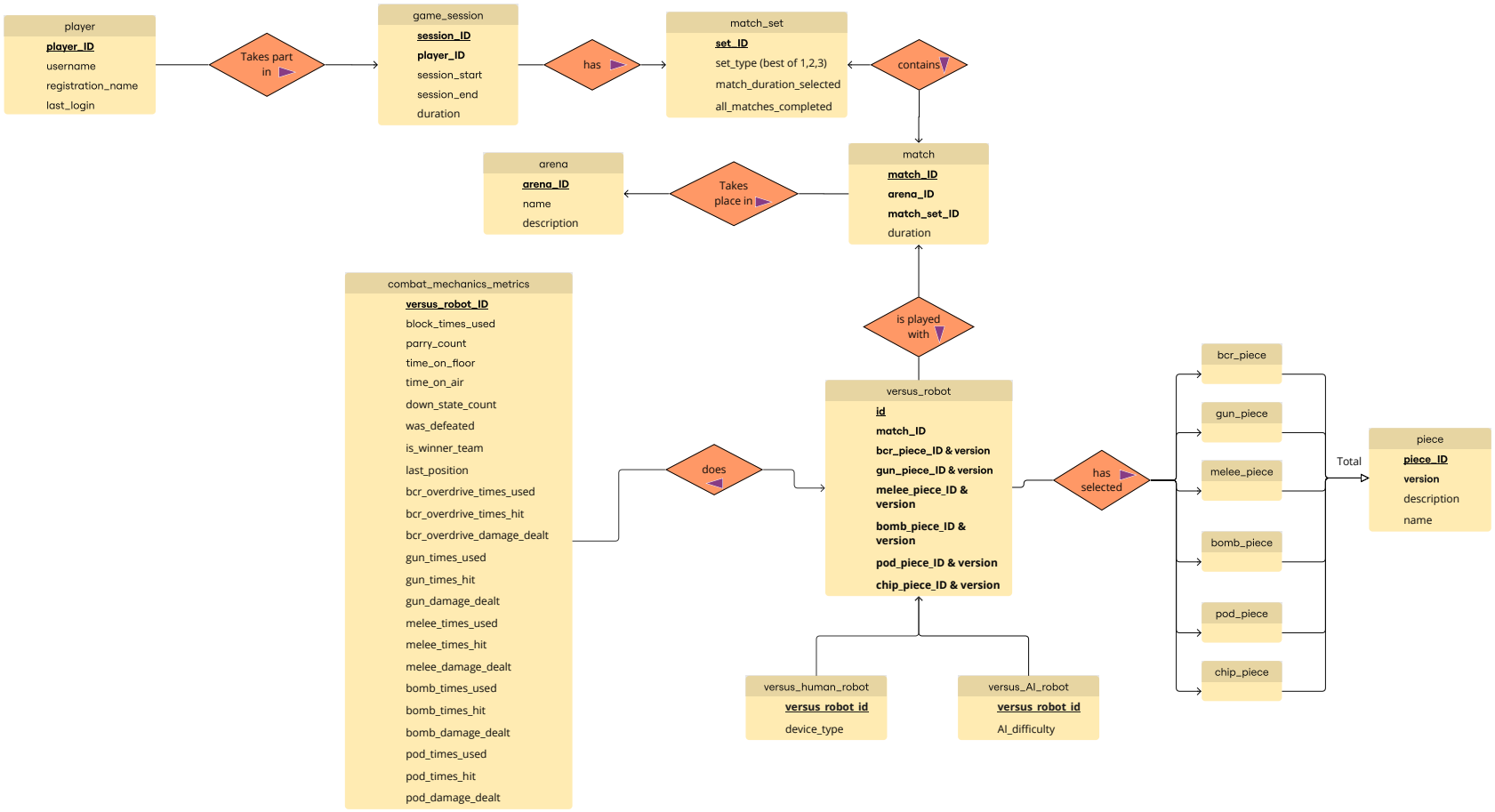


Figura 3.3: Modelo ER que define la interacción de los jugadores con el juego completo.

La figura 3.4 contiene las entidades necesarias para representar un usuario empezando una sesión de juego. Estas entidades a la hora del experimento, no serán clave. Sin embargo, para futuros trabajos serán determinantes porque dan pie a hacer diferentes tipos de análisis, orientados a usuarios y a análisis de supervivencia, con los que se pueden obtener estimaciones de qué es lo que lleva a un cliente a dejar de jugar.

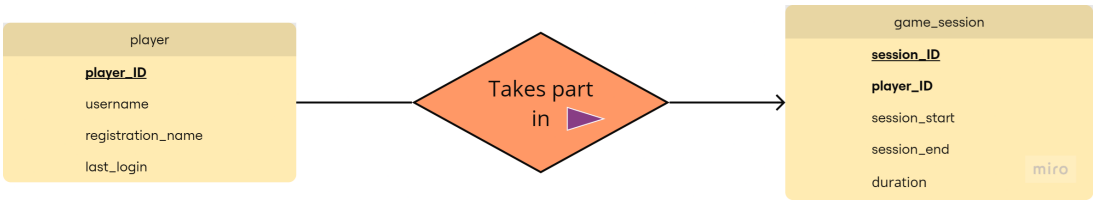


Figura 3.4: Modelo ER 1: Entidades usuario y sesión de juego.

Las entidades de la figura 3.5 recogen toda la información de un combate que no tiene que ver con los robots y mecánicas de juego. Con ello, podemos acceder a las métricas de duración de partida y mapas más seleccionados por los jugadores.

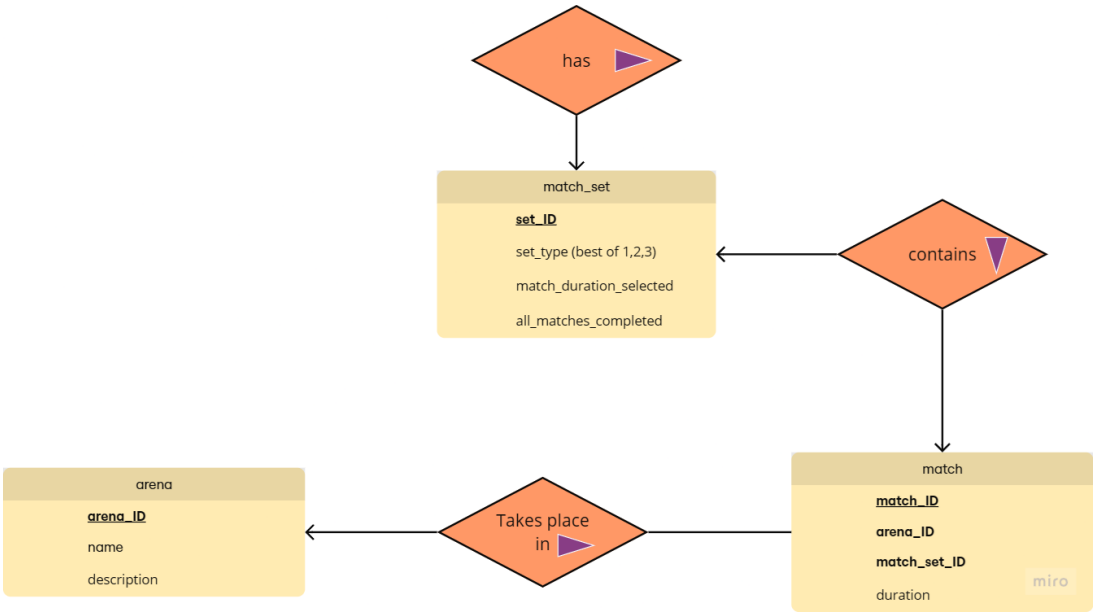


Figura 3.5: Modelo ER 2: Entidades para representar set de combates, combates y mapa.

En la figura 3.6 se representa la información de las piezas elegidas por un jugador. A la derecha de la figura está representada la entidad pieza con una Generalización Disjunta Total [17].

Esto significa que una entidad en la superclase (`piece`) solo puede ser de uno de los tipos de subclase (`bcr_piece`, `gun_piece`, `melee_piece`, `bomb_piece`, `pod_piece`, `chip_piece`) y tiene que pertenecer siempre a una de las subclases definidas. Por ejemplo, no puede existir una entidad `piece` sin que esta exista en una de las subclases.

PostgreSQL tiene soporte para herencia [6], por lo que se puede representar dicha relación y puede ser útil para realizar búsquedas de todas las piezas o de un tipo de pieza en concreto.

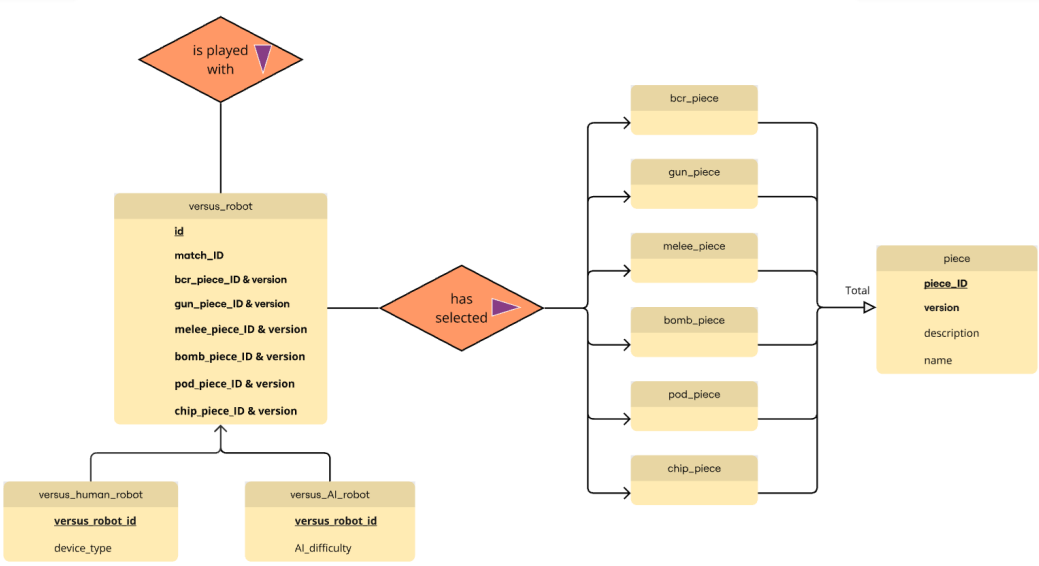


Figura 3.6: Modelo ER 3: Entidades de los robots con sus piezas

Por último, queda por representar la interacción del usuario con el robot. Como se puede ver en la figura 3.7, se va a recoger todo aquello relacionado con usar los controles del juego y el rendimiento con cada acción.

Algunos atributos a destacar son:

- **`is_winner_team`**: permitirá cruzar esta entidad con `versus_robot` y obtener que piezas tienen mayor tasa de victoria o derrota.
- Atributos que acaban con **`_times_used`**: se utilizará para evaluar cuanto uso se ha hecho de las mecánicas.

### 3.2.2 Arquitectura

Con el modelo ER definido, se planteó la arquitectura para poder recoger datos, y después leerlos y realizar su análisis.

Se planteó una arquitectura con los siguientes tres puntos:

1. Un sistema dentro del juego para almacenar datos, desacoplado del resto de sistemas ya que puede que no todas las versiones del juego cuenten con recogida de datos.

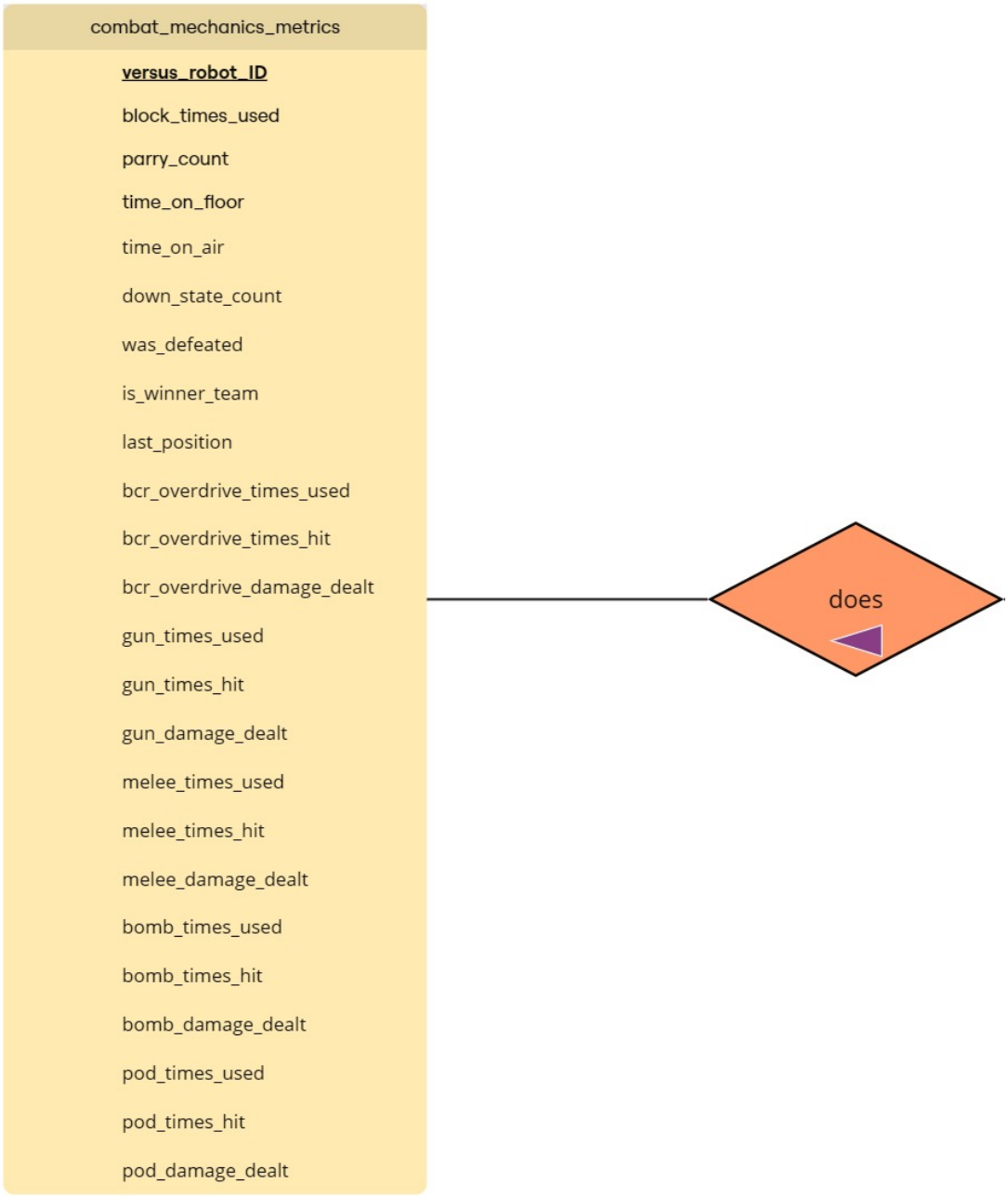


Figura 3.7: Modelo ER 4: Entidad mecánicas de combate.

2. Al tratarse de un videojuego, el sistema para almacenar datos no debe de interrumpir la experiencia del jugador, es decir, la lógica del videojuego no debe esperar a que los datos sean mandados para seguir avanzando por las diferentes pantallas, ya que dicha espera puede perjudicar la experiencia del jugador.

3. Garantizar que los datos son accesibles para realizar su documentación.

Se ha utilizado RabbitMQ como *middleware* (programa con el que las diferentes aplicaciones se comunican entre sí [3]) de cola de mensajes, así consiguiendo una arquitectura

en la que el videojuego puede mandar un mensaje con los datos y otro programa será el encargado de procesar el mensaje y almacenar los datos en la base de datos, cumpliendo así el punto de tener una lógica de guardado de datos paralela al programa principal (el videojuego). En los sistemas de cola de menajes hay un "productor" de mensajes y un "consumidor" que los recibe y trabaja con la información.

De tal modo, la arquitectura se divide en cuatro bloques principales:

1. El sistema dentro del videojuego para recoger datos, el productor de mensajes con datos.
2. La cola de mensajes en RabbitMQ.
3. El programa que recibe mensajes y manda los datos a la base de datos en PostgreSQL, el consumidor.
4. Un documento en Quarto con el que generar una página con los datos que facilite la lectura y decisión.

El figura 3.8 contiene el diagrama de la arquitectura final del Experimento I. Su funcionamiento es:

1º En Unity, hay un código que registra los datos de las partidas en estructuras de datos instanciadas dinámicamente en el *Heap* de Memoria del programa.

2º El código del juego se integra con distintos eventos asociados a los estados del jugador [18], como por ejemplo la carga de una nueva pantalla o la finalización de un combate. En Unity y .NET, estos eventos pueden representarse mediante `UnityEvent` [48] o mediante "delegados" [16]. El código responsable del envío de mensajes se suscribe a estos eventos para enviar los datos a RabbitMQ a través de su Interfaz de Programación de Aplicaciones (*Application Programming Interface*) (API).

3º La API dentro de un Contenedor de Docker hace que el mensaje se mande a RabbitMQ, llegando así a la cola de mensajes.

4º Rabbit MQ procesa el mensaje y espera a los consumidores.

5º El programa externo para guardar la información está conectado a través de la API, esperando a recibir mensajes para procesarlos.

6º Cuando un mensaje llega, se almacena en una cola dentro del programa, evitando incurrir en penalizaciones de rendimiento a la hora de insertar datos. El programa espera a que una entrada se añada para insertar otra nueva.

7º Para conectarse con la base de datos, el programa utiliza el proveedor de datos *Npsql*, con el que se establece una conexión y se inserta la información del mensaje.

8º Para redactar el documento, utilizando Quarto, se lanza un cliente de PostgreSQL desde Python.

9º El documento se renderiza a formato HTML para que el equipo pueda leer y tratar los datos captados.

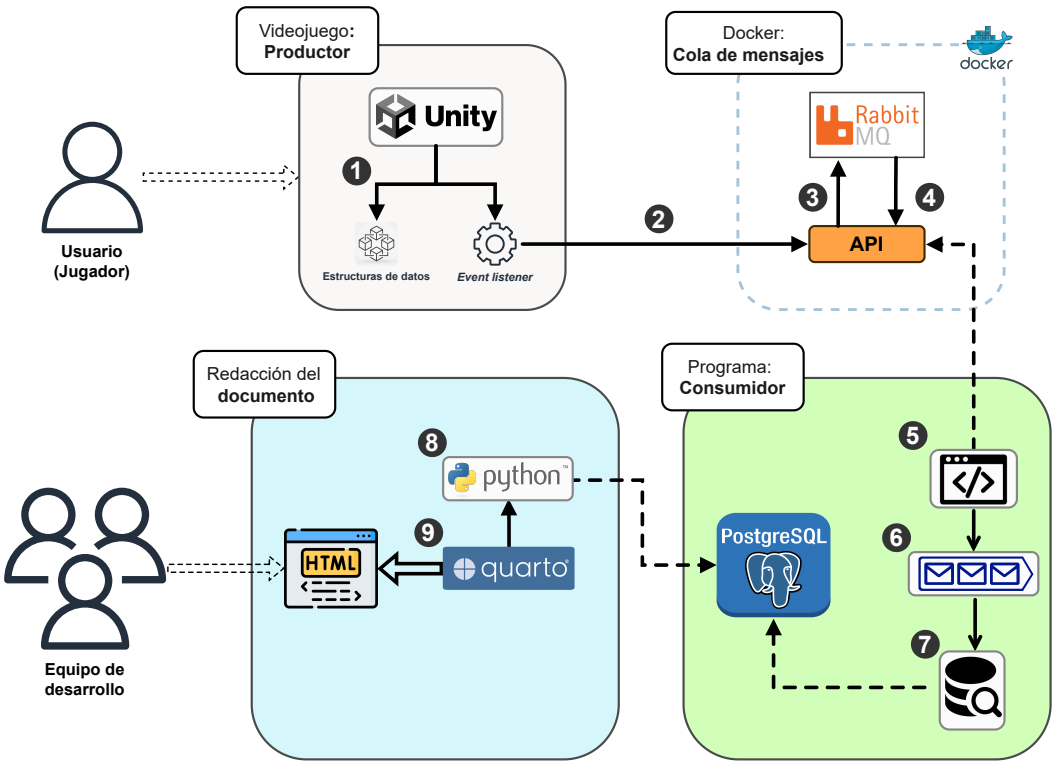


Figura 3.8: Arquitectura de la cola de mensajes para almacenar datos.

### 3.2.3 Arquitectura en Unity

En el lado del videojuego, hay que conseguir que el sistema que mande los datos a la cola de mensajes esté totalmente separado de la lógica del juego.

Para ello, se han empleado dos tecnologías, *Addressables* [14] de Unity e inyección de dependencias con C#.

*Addressables* es un sistema de Unity que permite organizar, incluir y excluir contenido de forma eficiente. Este sistema se basa en el uso de catálogos y Asset Bundles (grupos de assets gráficos y de código), lo que facilita la carga dinámica de recursos en tiempo de ejecución. Gracias a ello, es posible optimizar tanto el tamaño de la *Build* inicial como la gestión de contenidos, mejorando la modularidad y escalabilidad del proyecto.

Para comprenderlo se puede observar la figura 3.9, obtenida de la documentación oficial de Unity [14], que muestra como se han separado diferentes objetos de una playa. Las cajas de arriba representan los modelos 3D de la playa, y abajo los sistemas de código. Dentro del juego, se puede cargar los diferentes recursos dinámicamente cuando sean necesarios.

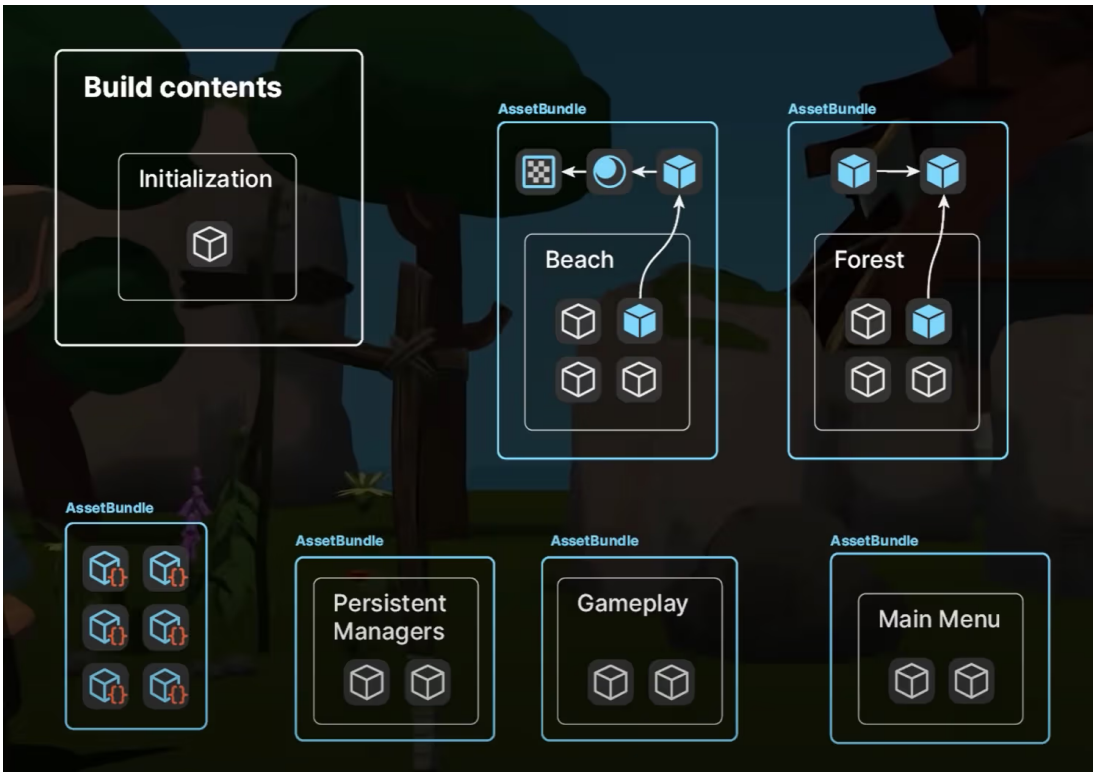


Figura 3.9: Ejemplo de los contenidos de un juego utilizando *Addressables* [14].

En el caso del videojuego, se utilizó para poder gestionar qué contenido es incluido en el juego y cual no. Para el experimento es clave, ya que es necesario un sistema que permita incluir o excluir el código y sistemas que permitan la lectura y escritura de datos, ya que no todas las versiones del juego publicadas contarán con recogida de datos.

*Addressables* se utilizó para incluir o excluir, y cargar dinámicamente escenas de Unity (grupo de objetos que representan un escenario del videojuego). Una práctica común es tener escenas en las que se incluye solo lógica del videojuego, creando así escenas reutilizables a lo largo del proyecto, además de tener un sistema robusto con código independiente.

De esta manera, se planteó tener escenas que se encarguen exclusivamente de mandar datos a la cola de mensajes. Así, se puede tener una con el contenido del videojuego y otra "superpuesta" con los sistemas necesarios nuevos para mandar los datos. La figura 3.10 muestra un esquema sobre está arquitectura.

De este modo, con incluir o excluir esa escena se podrá quitar o añadir la lógica de enviar datos. Además, para trabajos futuros en los que los datos se mandan a una nube, será suficiente con crear una escena concreta que se encargue de ello.

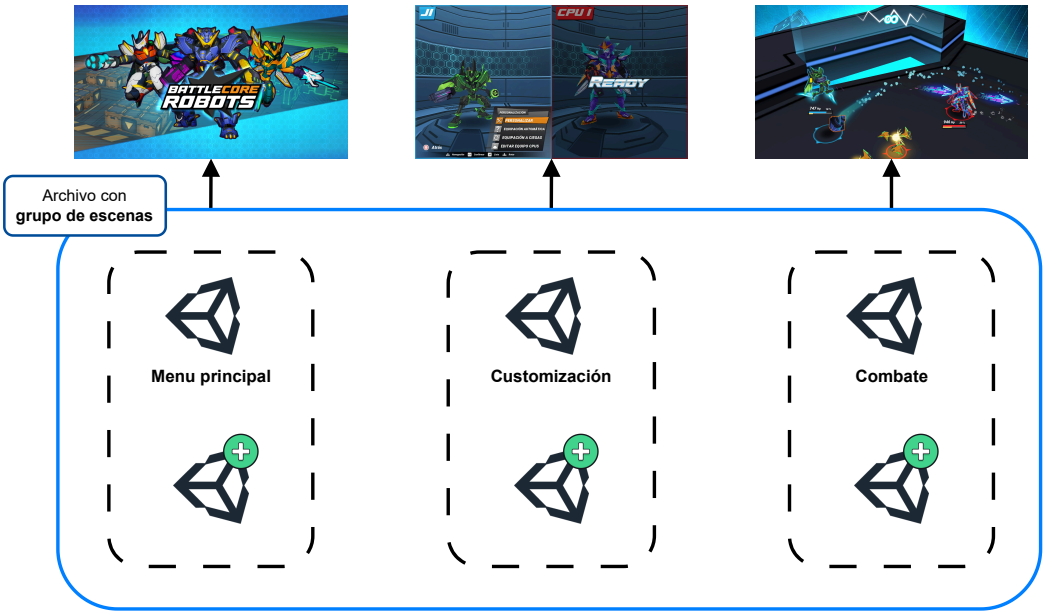


Figura 3.10: Representación de grupos de escenas con la escena principal y la escena superpuesta.

Por ello, se implementó un sistema de "grupos de escenas". Un grupo de escenas representa la escena principal y las escenas superpuestas que haya que cargar. Para ello, se utilizó un paquete de referencias de escenas [11], junto con una adaptación de repositorio sobre grupos de escenas [49] para poder crear grupos de escenas en el que compartir dependencias.

Sobre los sistemas se añadió:

- **Archivos de grupos de escenas**: así, podemos tener archivos para versiones concretas del juego.
- **Prioridad de carga**: así, las escenas que tengan que dar dependencias pueden proveerlas primero.
- **Finalizar la carga de escena manualmente**: en algunos casos, fue necesario para que se acabasen ciertas tareas de carga antes de mostrar la escena.

Lo último necesario es un sistema que referencia los objetos entre escenas ya que en Unity no se pueden referenciar objetos entre escenas diferentes, por lo que los sistemas superpuestos no pueden acceder a los objetos que necesitan.

Para solventarlo, se utilizó inyección de dependencias para poder desacoplar la lógica y poder compartir sistemas de una escena a otra.

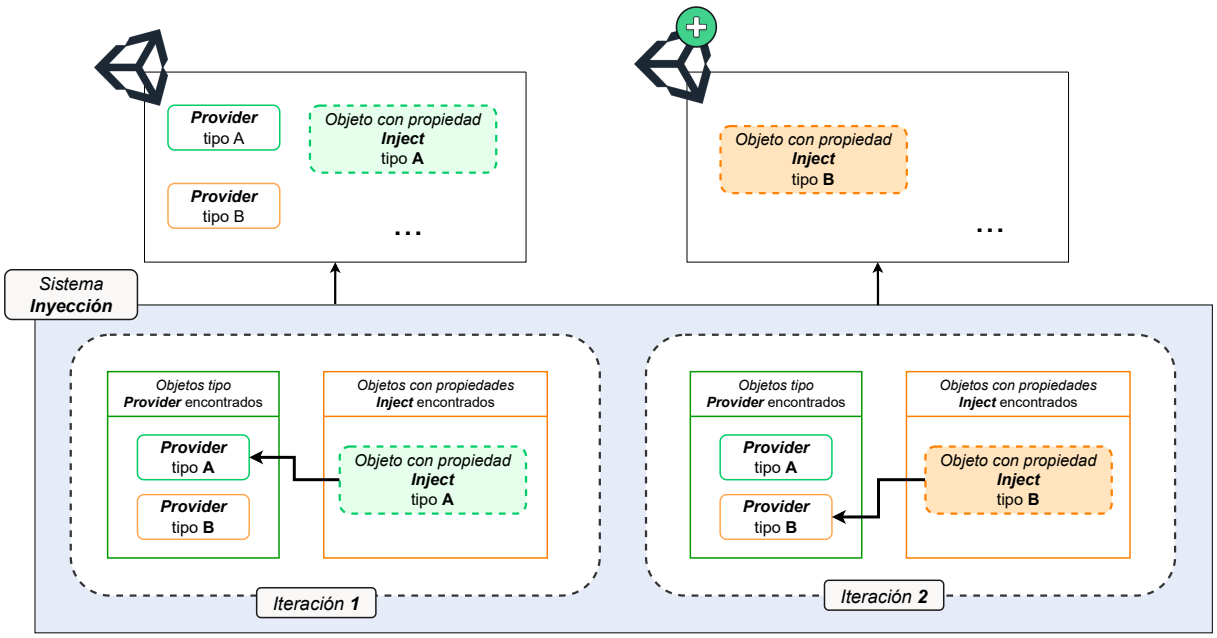


Figura 3.11: Ejemplo de la inyección con multiescena.

El diagrama de la figura 3.11 muestra como funciona la inyección de dependencias entre escenas:

- Las escenas tienen un orden de carga.
- Se recogen los sistemas marcados como "*Provider*" de la escena cargada.
- Se carga la siguiente escena.
- Una vez cargadas las escenas, se buscan las referencias marcadas con la etiqueta "*Inject*".
- Se otorgan las dependencias que hay que proveer a las que hay que inyectar.
- Se finaliza la carga de la escena y se muestra al jugador.

La escena implementada cuenta con el sistema que manda datos a la cola de RabbitMQ, haciendo así que el videojuego sea el *Sender*.

El mensaje mandado es la instrucción que tiene que realizarse en la base de datos para insertar el dato.

Antes de explicar la captación de datos, es importante entender un sistema de eventos en videojeugos. Una práctica común en videojuegos es contar con eventos que informen del estado en el que se encuentra el juego. Por ejemplo "Comenzar combate" o "Daño recibido". Estos eventos se utilizarán para crear y enviar las instrucciones SQL a la base de datos.

Se crearon los siguientes sistemas para la captación de datos, descritos en el apéndice A:

- **Un conversor de tipos de C# a tipos de PosgtreSQL**.
- **Uno para transformar instancias de C# comando SQL**: Se creó la clase abstracta genérica `A_BCR_Psql_InsertItemBuilder<TPsqlItem,TObject>`, que convierte la clase `TObject` al `TPsqlItem` definido. `TPsqlItem` cuenta con un método para crear el comando SQL.
- **Uno para crear los comandos y enviarlos**: Son las clases que heredan de `ADataCollector`, crean comandos de tipo `TPsqlItem` o la cadena de texto de las instrucciones para mandarlos a la base de datos.
- **Una clase *Singleton* [41] con la conexión a RabbitMQ**: Así, cualquier sistema de tipo `ADataCollector` puede acceder a ella sin importar la escena en la que se encuentre.

En el diagrama Lenguaje Unificado de Modelado (*Unified Modeling Language*) (UML) mostrado en la figura 3.12 contiene las relaciones entre de los sistemas para la captación de datos.

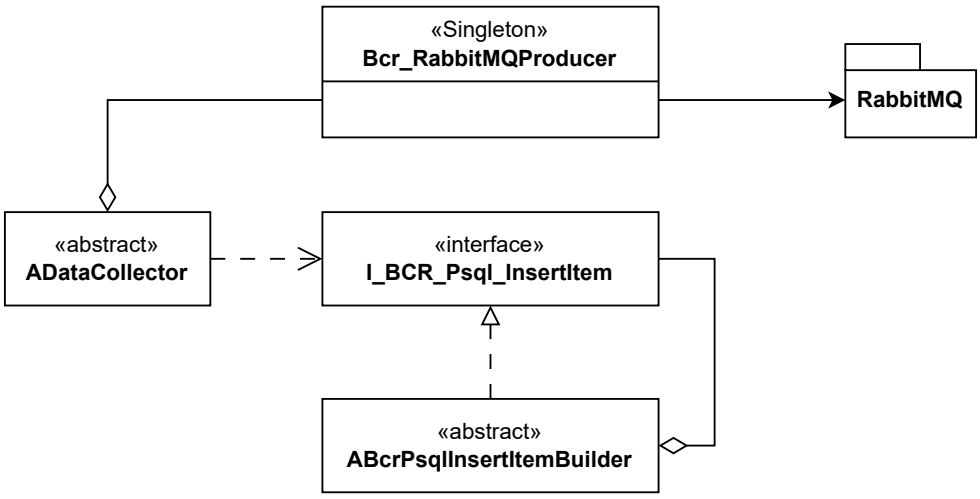


Figura 3.12: Diagrama UML del sistema que convierte información del videojuego a mensajes de la cola con los datos a insertar.

Un apartado importante es la gestión de claves foráneas. En el experimento, los datos tienen forma de jerarquía, en la que hay una cadena de dependencias a lo largo del ciclo de vida del programa. De esta manera, las estadísticas de combate están asociadas a un robot y este pertenece a un combate. Cada combate forma parte de un conjunto de combates (*set*) que ocurre dentro de una sesión de juego, la cual está vinculada a un jugador (*player*).

El programa contará con una memoria del dato actual de la sesión de juego, set de combates, combate y robots. El programa para insertar los datos en la base de datos (subsección 3.2.4) es el responsable de gestionar este aspecto.

Los mensajes incluyen etiquetas para indicar que la instrucción necesita una referencia. La etiqueta es "`*reader[READER_TARGET]*`", donde `READER_TARGET` es el atributo de la nueva fila a insertar que necesita el valor.

### 3.2.4 Arquitectura: Cola de mensajes e inserción en la base de datos

Se utilizó RabbitMQ como *broker* de mensajes para desacoplar la lógica del videojuego hecho en Unity del sistema encargado de escribir los datos en la base de datos de PostgreSQL.

El funcionamiento de RabbitMQ consiste en enviar mensajes y que otro sistema en paralelo los consuma. Así, el *Producer* envía mensajes que un *Consumer* procesa y realiza un trabajo concreto. Para el experimento, el proyecto en Unity será el *Producer*, y un programa hará de *Consumer* para recibir los mensajes, pre-procesar el mensaje (tratado más adelante) para ajustar la instrucción y enviar dicha orden a la base de datos.

Fueron necesarias las siguientes partes:

- **Sistema para mandar mensajes desde Unity (el *Producer*) a la cola**: publica instrucciones SQL a la cola de RabbitMQ.
- **Programa para gestionar el consumer y procesar las llamadas a la base de datos**: desarrollado en C#, con un cliente de RabbitMQ que escucha y recibe los mensajes de la cola.
- **Docker**: utilizado para tener un contenedor (veáse la sección 2.7) con el servicio de RabbitMQ.

A continuación, se va a tratar el programa consumidor.

La figura 3.13 muestra el flujo que sigue el programa para insertar datos. Se inicia un cliente en RabbitMQ para crear el *Consumer* de los mensajes para finalmente introducirlos en la base de datos.

Un paso a destacar es la caja con relleno azul, el procesamiento de la clave foránea (introducido en la subsección anterior). Antes de insertar los datos, se ha de comprobar si es una instrrucción que depende de una clave foránea de la jerarquía actual. En caso de

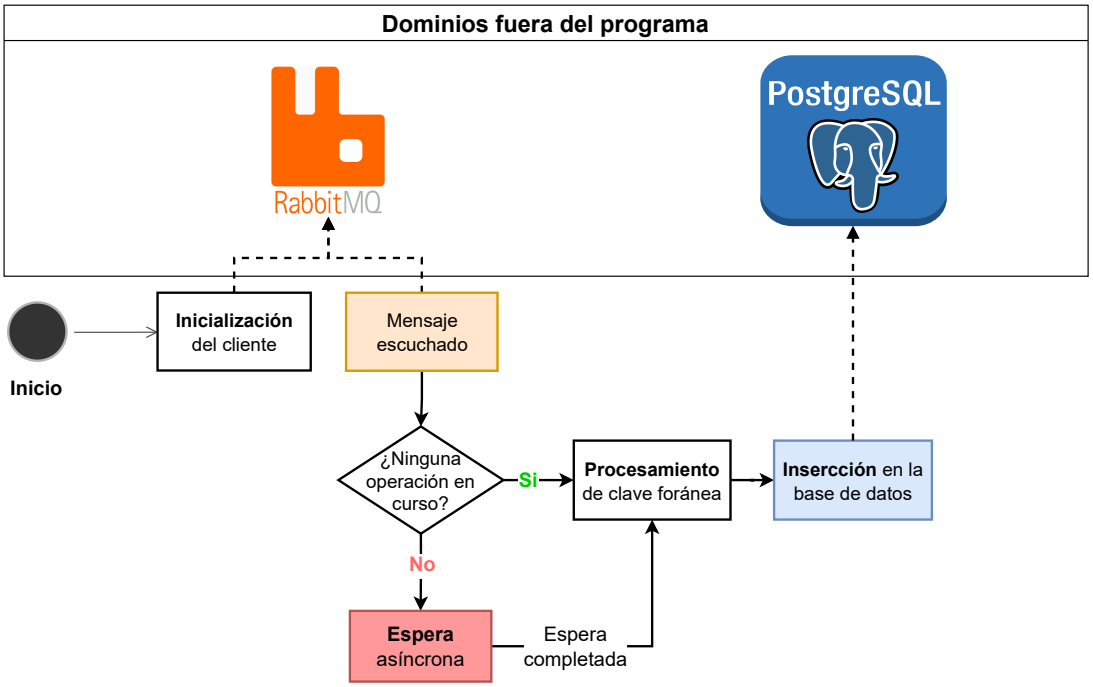


Figura 3.13: Diagrama de flujo del programa encargado de insertar los datos.

que así sea, la instrucción es procesada para tener los valores adecuados y ser insertada en la base de datos.

En el código 3.1 se muestran los métodos creados para dicha gestión de las claves foráneas.

Con el diseño definido hasta ahora se cuenta con todo lo necesario para llevar a cabo el experimento:

1. La base de datos en PostgreSQL.

2. El contenedor Docker con la cola de RabbitMQ.

3. Los sistemas en el videojuego para captar datos y enviarlos a la cola de RabbitMQ.

4. El sistema para recibir las instrucciones, procesarlas e introducirlas a la base de datos.

```
using System.Text.RegularExpressions;

namespace Consumer;

public class ConnectionForeignKeysManager
{
    private Dictionary<string, object> _foreignKeysHandledDictionary;
    public int _gameSessionID;
    public int _currentMatchSetID;
    public int _currentMatchID;

    private static readonly string[] ForeignKeysHandled =
      new[] { "game_session_id", "match_set_id", "match_id" };

    private String SubsituteForeignKeyPattern => @"'\[fk:(\w+)\]'";

    public ConnectionForeignKeysManager()
    {
        _foreignKeysHandledDictionary = new Dictionary<string, object>();
        foreach (var foreignKey in ForeignKeysHandled)
            _foreignKeysHandledDictionary[foreignKey] = -1;
    }

    public void StoreValue(string foreignKeyName, object value)
    {
        if (_foreignKeysHandledDictionary.ContainsKey(foreignKeyName))
            _foreignKeysHandledDictionary[foreignKeyName] = value;
    }

    public void TryHandleForeignKey(ref string command)
    {
        command = Regex.Replace(command, SubsituteForeignKeyPattern, match =>
        {
            string key = match.Groups[1].Value;
            Console.WriteLine($"foreign key found: {key}");
            return (_foreignKeysHandledDictionary.TryGetValue(key, out var value) ?
                 Extensions.FormatSqlValue(value) : "NULL")
                 ?? string.Empty; // Si no se encuentra, pone NULL
        });

    }
}
```

Código 3.1: Código de la clase del programa *Consumer* que gestiona la jerarquía de claves foráneas.

```
def execute_query(query) -> pd.DataFrame:
return pd.read_sql(query, conn)
```

Código 3.2: Método en Python para convertir una instrucción SQL en un *Dataframe* de pandas.

```
# Clase pieza
class Piece(Enum):
    bcr = 1
    gun = 2
    bomb = 3
    melee = 4
    pod = 5
    chip = 6
```

Código 3.3: Código Python del enumerador que representa un tipo de pieza.

El último requisito sería el entorno para analizar los datos con Quarto y Python.

### 3.2.5 Visualización de métricas y redacción con Quarto y Python

Se empleó Quarto (veáse la subsección 2.2.2) para crear un documento que permita mostrar las métricas junto con texto. El objetivo es crear una plantilla para poder tratar los datos y crear métricas con las que el equipo aplique los sistemas de decisión definidos.

El formato elegido para el documento ha sido `HTML`, junto con la opción de renderizado de Quarto `embed-resources: true` para tener el archivo con todos los gráficos embebidos. De esta manera se consigue un archivo que se puede almacenar en el repositorio del equipo.

Para leer los datos, se utilizó Python junto con Astral UV (tratados en la subsección 2.2.1). En el apéndice B se describe el código para realizar el documento. Algunos códigos a destacar aplicando las liberías introducidas en la subsección 2.2.1 serían los siguientes:

El código 3.2 contiene el método para convertir una instrucción SQL a un *Dataframe* de Pandas. Esto aligerará el desarrollo del documento.

Después, se crearon métodos para agilizar la creación de gráficas. Para ello, se creó un enumerador con el que representar cada tipo de pieza para los apartados de análisis de piezas. El código 3.3 muestra la implementación.

Trabajando con el enumerador, se pueden generalizar métodos para evitar repetir tanto

```
1   def Piece_formater(piece_enum: Piece) -> str:
2       mapping = {
3           Piece.bcr: "Battlecore",
4           Piece.gun: "Pistola",
5           Piece.bomb: "Bomba",
6           Piece.melee: "Cuerpo a cuerpo",
7           Piece.pod: "Vaina",
8           Piece.chip: "Chip",
9       }
10      return mapping.get(piece_enum, piece_enum.name.capitalize())
```

Código 3.4: Código Python Para convertir un enumerador en texto.

código. Otro de los usos del enumerador es el mostrado en el código 3.4, con el que se puede convertir en texto el valor introducido, para una lectura menos técnica de los datos.

Finalmente, se cuenta además de la arquitectura de captación de datos, con una herramienta para tratar los datos. En el siguiente capítulo se abordan los resultados obtenidos.

# Capítulo 4

# Experimentos y validación

Se consiguió llevar a cabo el experimento descrito en el capítulo anterior para recoger datos de jugadores y realizar un análisis para tomar decisiones. Tras implementar la arquitectura definida, se asistió al evento "Gamegen" de la Universidad Rey Juan Carlos, llevando dos equipos informáticos para facilitar que más personas pudieran jugar sin esperas. En cada ordenador se instaló la versión del juego con la capacidad de enviar instrucciones SQL a la cola, junto con el contenedor de Docker que ejecutaba el servicio de RabbitMQ.

El evento tuvo una duración de 3 días. Una vez completado, se consolidaron los datos y se utilizó Quarto y Python para generar un documento web con las métricas y anotaciones de los resultados. Dicho archivo se trata en el apéndice A.

A continuación, se mostrarán las métricas y los resultados obtenidos.

## 4.1 Volumen de datos

Para evaluar el volumen de datos recogido, se mostrará la cantidad de tuplas de las entidades:

- `match_set`: Esta entidad representa el número total de sets de partidas. Sirve como estimación de la cantidad de sesiones de juego o grupos de jugadores que participaron en el experimento.
- `match`: Indica el número total de combates individuales que se han disputado.
- `versus_robot`: Para conocer la cantidad de datos de interacción relacionados con la selección de piezas y el uso de mecánicas de juego que se van a analizar.

El volumen de datos obtenido del experimento está mostrado en la tabla 4.1.

La cantidad de datos recopilados parece ser adecuada para llevar a cabo un análisis detallado y fundamentar conclusiones válidas.

| Sets de Combates | Combates | Robots |
|---|---|---|
| 150 | 205 | 453 |

Tabla 4.1: Volumen de Datos Obtenidos del Experimento.

Las posteriores secciones tratarán los análisis realizados para generar sistemas de decisión.

## 4.2 Análisis del tiempo de combates

Se va a comenzar tratando la duración de los combates. Dicho dato es importante ya que permite evaluar si la experiencia de juego tiene una duración adecuada. Para ello, se empleó un diagrama de densidad y otro de caja para así conocer frecuencias de la duración de partidas y tener un análisis descriptivo en una sola figura. La métrica se muestra en la figura 4.1.

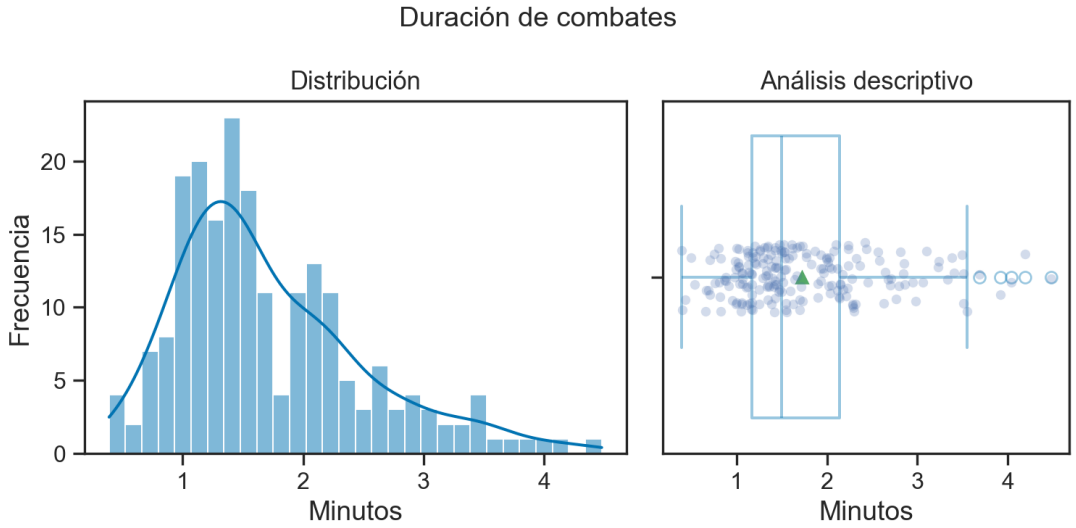


Figura 4.1: Diagrama de densidad de la duración de los combates.

De dicha gráfica, se pueden extraer varios puntos importantes:

- Los combates duran entre 30 segundos y 4 minutos y medio, como mucho.

- Lo habitual parece ser que dure entre 1 minuto y 2.
- La media está en casi dos minutos.
- La moda está en minuto y medio.

Estos datos permiten generar un sistema de decisión para analizar si tanto los combates como el juego tienen una duración adecuada.

La duración de los combates será útil para estimar si el juego tiene una acción y ritmo buenos (un combate largo puede aburrir al jugador). La duración deseada de una partida es aproximadamente de dos minutos, para que no sea tan corto como un minuto, pero no tan largo como tres. Observando el análisis descriptivo, la mediana se sitúa en el minuto y medio, por lo que se puede concluir que hay que profundizar en los diferentes aspectos que definen la duración de un combate (como por ejemplo valores de daño de las piezas, atributos de los robots...) para corregir la duración de los combates.

Como resultado, si incluimos esta misma reflexión en el flujo de trabajo, se consigue el primer sistema de decisión para responder la pregunta: ¿Los combates de los jugadores han tenido una duración adecuada?

El juego en su versión final contará con un modo historia (véase la subsección 1.4.3). Por lo que esta métrica sirve para poder establecer una estimación *a priori* de cuánto puede durar el juego. Un ejemplo de cómo puede ayudar es a través de la fórmula de la ecuación 4.1.

$$D_{\text{total}} = (D_{\text{combate}} \times N_{\text{combates}}) + (N_{\text{diálogos}} \times D_{\text{diálogo}}) + \left( \frac{N_{\text{combates}}}{N_{\text{diálogos}}} \times D_{\text{interacción}} \right) . \quad (4.1)$$

Donde:

- $\mathbf{D_{combate}}$: duración media de un combate (en segundos o minutos).
- $N_{\mathbf{combates}}$: número total estimado de combates que el jugador realizará a lo largo del juego.
- $N_{\mathbf{diálogos}}$: número total de diálogos presentes en el juego.
- $\mathbf{D_{diálogo}}$: duración media estimada para completar un diálogo. En un futuro, podría obtenerse a través de datos.
- $\mathbf{D_{interacción}}$: tiempo medio estimado que el jugador dedica a interactuar con el entorno entre combates y diálogos.

Como resultado, poder recoger esta información podrá ayudar al equipo a tomar decisiones que afecten significativamente al desarrollo del juego.

También es interesante realizar el estudio de la duración de combates por el número de jugadores. Como se ha mencionado en la subsección 1.4.3, las partidas pueden ser de 2 a 4 jugadores. La métrica representada en la figura 4.2 muestra dichos datos.

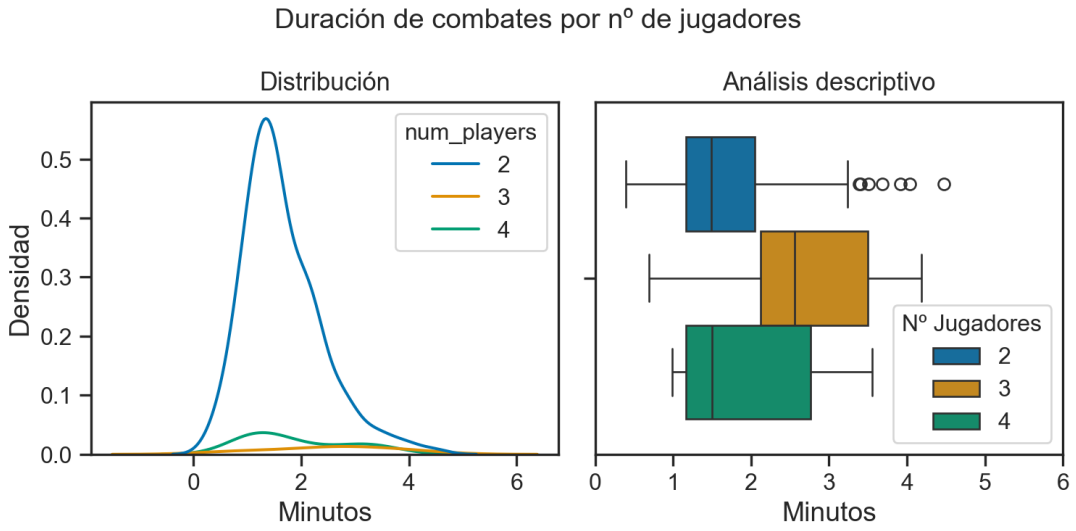


Figura 4.2: Duración de partidas por nº de jugadores.

Observando el análisis descriptivo, parece que las partidas con más jugadores tienden a durar más. La cantidad de datos afecta significativamente al resultado. Como se puede ver en la gráfica de distribución, hay más densidad de datos para combates de 2 jugadores que de 3 y 4.

En resumen, se ha conseguido leer datos de la duración de los combates, dando lugar a métricas y consiguiendo establecer sistemas de decisón que permiten apoyar a la toma de decisiones del equipo. El sistema concretamente tendría la siguiente forma:

1. Se procesan los datos.
2. Se recogen las métricas de duración de partidas.
3. Se realiza una evaluación. Para ello, se comprueba la mediana y la moda.
4. Si la mediana no está cerca de 2 minutos, surge una nueva tarea para estudiar por qué han durado tanto y hacer los ajustes necesarios.
5. Teniendo el modo historia ya en desarrollo, se comprueba la estimación *a priori* que tendrá el juego con la duración obtenida. Con un registro de las estimaciones, el equipo puede comparar los resultados mes a mes.

## 4.3 Análisis del uso de piezas

Esta sección se centra en el estudio de las piezas del robot, una fase crítica para la selección del contenido inicial disponible para los jugadores de Patreon. Hay dos objetivos principales:

1. Identificar qué piezas son más adecuadas para la primera versión del juego.
2. Generar un sistema de decisión que permita identificar si hay alguna pieza o categoría de piezas descompensada (se han ganado demasiados combates con esa pieza).

Como se ha mencionado en la sección 3.2, el objetivo es crear una versión para los futuros suscriptores de Patreon. Dicha versión tiene reducido el contenido, por lo que hay que elegir el contenido más adecuado para la primera versión.

Los criterios para decidir si una pieza es adecuada y cumplir el primer objetivo de este análisis son:

- **La cantidad de veces que se ha usado**: esta métrica puede mostrar que una pieza ha llamado la atención a los jugadores.
- **Su tasa de victoria**: la tasa de victorias es el resultado de las veces que se ha ganado un combate entre el total de veces que se ha utilizado la pieza. Una pieza que se ha seleccionado muchas veces, pero tiene baja tasa de victoria puede generar frustración (y como resultado una mala exepriencia de juego) si se incluye en la primera versión.
- **Acabado y estilo visual**: se contrastará el juicio del equipo con la cantidad de partidas para poder medir este factor.

Para realizar este análisis, se emplearán dos gráficos por cada categoría de piezas:

- **Un gráfico de barras agrupadas de la cantidad de partidas**: este gráfico se ordenará según el total de veces que se ha utilizado cada pieza. Adicionalmente, en lugar de mostrar únicamente el conteo de partidas, se visualizarán las victorias y derrotas de cada pieza.
- **Un gráfico de barras que muestre la tasa de victorias**: ordenado de mayor a menor y basado únicamente en combates de dos jugadores. Este último aspecto es debido a que la mayoría de la experiencia de juego se centra en combates 1 contra 1, por lo que se han descartado del análisis los resultados en combates de 3 ó 4 jugadores ya que el número de jugadores puede afectar al rendimiento de una pieza.

Para el segundo objetivo, se busca generar una métrica que muestra si una pieza o categoría de piezas está descompensada.

Al ser un juego de lucha, se espera que tanto las categorías de piezas como las piezas en sí tengan una tasa de victoria alrededor 50%. Para ello, al gráfico de la tasa de victoria de cada pieza de una categoría se añadirá una barra a la altura del 50% para poder identificar las piezas que están descompensadas. Se puede entender una pieza desequilibrada como aquella que dista más del 5% de dicha barra.

También, se hará una gráfica mostrando el error cuadrático medio de la tasa de victorias de las piezas, comprobando así cual se aleja más del 50%. Se puede presuponer que un tipo de pieza con mayor error cuadrático medio es más significativa para determinar el resultado de una partida.

A continuación, se hará el análisis para cada tipo de pieza.

### 4.3.1 Uso de *Battlecores*

En la figura 4.3 se muestra en ánalisis de los robots. De la gráfica izquierda, se puede extraer que *Sharky*, *Harleking* y *Taiga Oscuro* son los robots más escogidos.

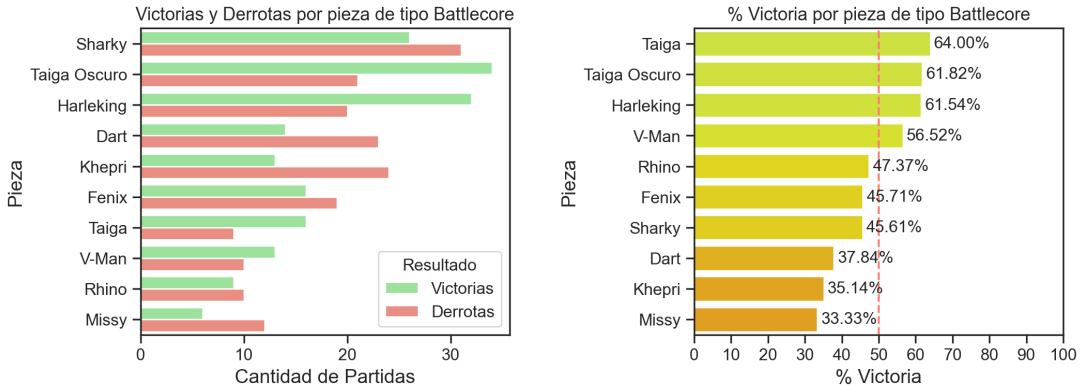


Figura 4.3: Cantidad de partidas y tasa de victoria para piezas de tipo *Battlecore*.

Contrastando con la gráfica de la derecha, se observa un ejemplo claro de éxito con el robot *Taiga Oscuro*. Este caso, no solo presenta una frecuencia de selección considerable, sino que también tiene el mayor número de victorias. Por lo que parece que *Taiga Oscuro* debería ser incluido en la versión de Patreon.

Por otro lado, el robot *Dart*, que es el Battlecore predeterminado del tutorial, muestra un rendimiento más bajo ya que su tasa de victoria no es elevada. En este caso concreto, es algo bueno, ya que comenzar con un robot de menor rendimiento y posteriormente adquirir otros más potentes puede generar en el jugador una sensación de progreso y mejora.

Se podría decir que el sistema de decisión planteado funciona, ya que la gráfica de la derecha revela la necesidad de revisar el balance de varios robots, como *Khepri* o *Missy*. Sus tasas de victoria se desvían significativamente del 50%, lo que implica un desequilibrio que será investigado para futuras iteraciones en el desarrollo del juego.

Para elegir los robots, se tienen que elegir al menos uno de cada clase (veáse la sección 1.4). Por lo que, sobre todo hay que elegir entre *Rhino* y *Khepri*, y elegir otro robot que descartar.

Finalmente, se decidió incluir a *Rhino*. A pesar de que no mucha gente lo probase respecto a *Khepri*, es más fácil y se estaba realizando un efecto visual para su habilidad especial más llamativa que la de *Khepri*. Por lo que el sistema sirvió para identificar que *Khepri* es más llamativo, sin embargo más frustante, así que se decidió marcar el robot como en estado de revisión para mejorarlo e incluirlo en una futura versión.

El otro robot descartado fue *Sharky*, el robot Dart es de la misma clase, y al tener tanto uso y debido a su acabado se archivo el robot para una futura versión, en la que lo destacado sea el robot.

Finalmente, los robots escogidos han sido: *Dart* (pieza inicial), *Taiga*, *Taiga Oscuro*, *Rhino*, *V-man*, *Harleking*, *Fénix*, *Missy*.

Con este primer caso, se puede comprobar que el sistema de decisión funciona, habiendo obtenido conclusiones de las elecciones de los jugadores y usándolo de apoyo para seleccionar aquellos robots que se van a incluir.

### 4.3.2 Uso de Pistolas (*Guns*)

La figura 4.4 muestra lás métricas para las piezas de tipo “Pistola”.

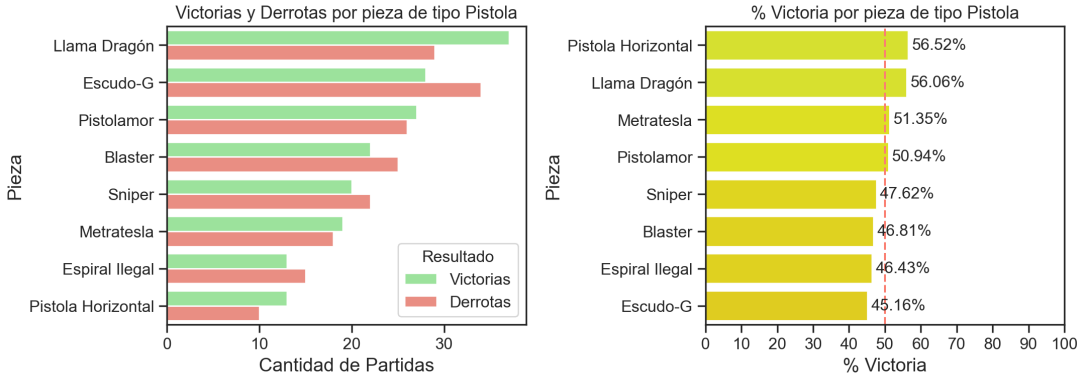


Figura 4.4: Cantidad de partidas y tasa de victoria para piezas de tipo “Pistola” (*Gun*).

En cuanto a las pistolas, la pieza *Escudo-G* es el ejemplo a remarcar de esta categoría de

piezas. Aunque ha sido frecuentemente seleccionada (lo cual podría atribuirse a su estilo visual), es la pieza con la menor tasa de victorias.

Un aspecto positivo observado en la categoría de las pistolas es que la tasa de victorias de todas las piezas se sitúa en torno al 50% deseado, lo que indica un buen balance general dentro de esta categoría.

Para la versión de Patreon, había que elegir seis piezas. Finalmente se descartaron *Pistola Horizontal* y *Metratesla*. La razón es su baja selección pero su alta tasa de victoria, serían piezas que podían hacer el juego aburrido porque son mucho mejores respecto las otras piezas. La razón por la que se eligió inlcuir *Llama Dragón* en vez de descartarla a pesar de ser una pieza poderosa es porque fue la pieza más seleccionada por los usuarios, probablemente debido a su estilo visual.

Las piezas que se inlcuyeron finalmente fueron:

1. *Llama Dragón*.
2. *Escudo-G*.
3. *Pistolamor*.
4. *Sniper*.
5. *Espiral Ilegal*.
6. *Blaster* (pieza inicial).

### 4.3.3 Uso de Bombas (*Bombs*)

En la figura 4.5 se muestra la interacción de los usuarios con las bombas.

Observando las gráficas, se puede concluir que todas las bombas están balanceadas, a excepción de *Brilli-Brilli*, *Terrormoto* e *Impacto Ácido*.

*Impacto Ácido* es la pieza inicial de la categoría Bombas, por lo que la tasa de victoria con la que cuenta no es correcta a nivel de experiencia de juego. Sin embargo, *Brilli-Brilli* y *Terrormoto* si son piezas que habría que revisar para futuras versiones porque se alejan más del 5% del 50% deseado.

En el caso de las bombas, se van a incluir seis, por lo que hay que descartar una.

Se decidió descartar *Terrormoto*, ya que era la que menos uso ha tenido y la que se aleja más 5% después de *Impacto Ácido*, que es la pieza inicial. Así pues, se incluyeron:

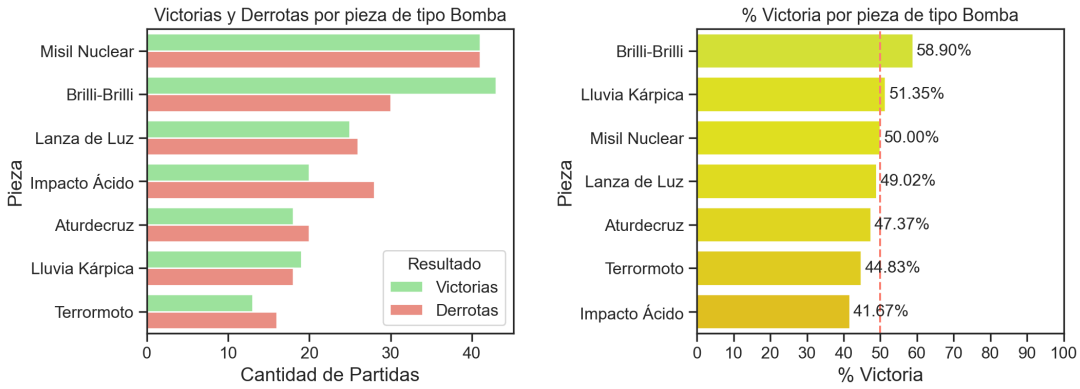


Figura 4.5: Cantidad de partidas y tasa de victoria para piezas de tipo "Bomba" (*Bomb*).

1. *Misil Nuclear*.
2. *Brilli-Brilli*.
3. *Lanza de Luz*.
4. *Impacto Ácido* (pieza inicial).
5. *Aturdecruz*.
6. *Lluvia Kárpica*.

### 4.3.4 Uso de Cuerpo a Cuerpo (*melee*)

La figura 4.6 muestra las piezas cuerpo a cuerpo. Es la categoría con menos piezas desarrolladas a fecha de la versón del evento.

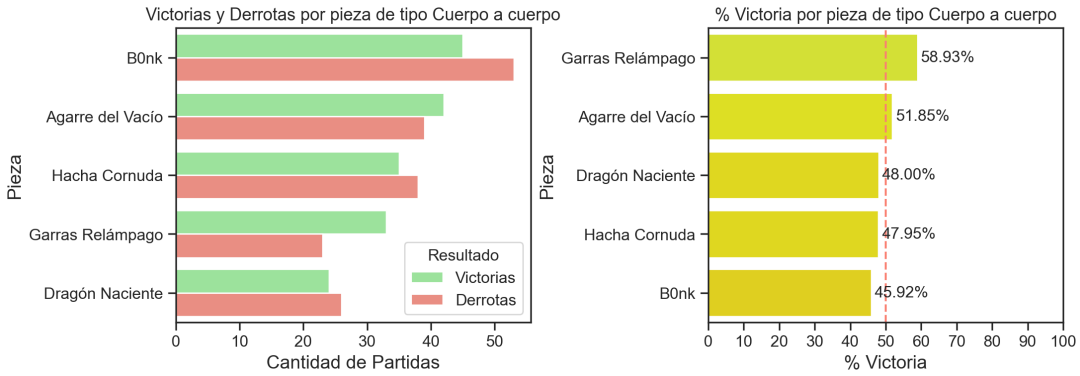


Figura 4.6: Cantidad de partidas y tasa de victoria para piezas de tipo "Cuerpo a cuerpo" (*Melee*).

La pieza inicial es *B0nk*, observando la gráfica derecha, se puede apreciar que ha sido la más seleccionada, pero la que menos ratio de victoria ha tenido. La gráfica izquierda

sugiere que todas las piezas, a excepción de *Garras Relámpago*, entran dentro de la franja de error del 5%.

Finalmente, se decidió excluir *Hacha Cornuda*. No es la pieza que menos veces se ha usado, pero es la segunda con menos tasa de victoria seguida de la pieza inical. Esto puede deberse a que es más difícil de usar en comparación con las otras piezas.

Como resultado, en la primera versión de Patreon se ha incluido:

1. *B0nk* (pieza inicial).
2. *Agarre del Vacío*.
3. *Garras Relámpago*.
4. *Dragón Naciente*.

### 4.3.5 Uso de Vainas (*Pods*)

El uso de piezas de tipo Vaina está mostrado en la gráfica 4.7.

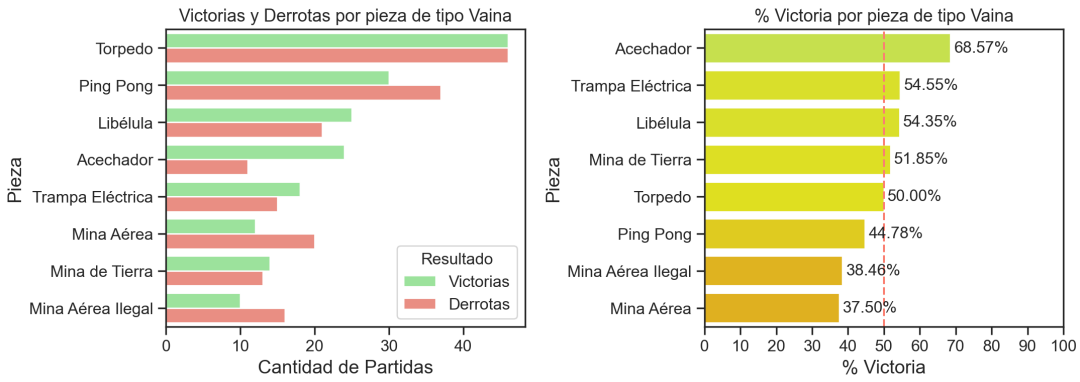


Figura 4.7: Cantidad de partidas y tasa de victoria para piezas de tipo "Vaina" (*Pod*).

*A priori*, esta figura indica que es la categoría de piezas con más variación de tasa de victoria. La pieza con más tasa de victoria, pero en el 4º puesto de uso es *Acechador*. Con este dato, gracias al sistema de decisión se puede identificar una pieza desequilibrada de forma clara.

También se muestran piezas que claramente no han llamado la atención ni han sido efectivas a la hora de jugar, como es el caso de *Mina Aérea* y *Mina Aérea Ilegal*. Estas piezas tendrán que ser revisadas por el equipo.

Se utilizó la desproporción de la tasa de victorias como oportunidad a nivel de la experiencia que podía surgir de esta categoría de piezas. Al estar tan marcados los resultados, se generan piezas mejores que otras. Esto es una excepción, puesto que en el futuro flujo de desarrollo se trabajará en que las piezas tengan resultados más cercanos al 50%.

De tal manera, observando la gráfica lo correcto parece ser excluir *Torpedo* y *Mina de Tierra*, ya que son las dos piezas con resultados más cercanos al 50%.

Las seis piezas de tipo Vaina incluidas en la primera versión fueron:

1. *Ping Pong* (pieza inicial).
2. *Libélula*.
3. *Acechador*.
4. *Trampa Eléctrica*.
5. *Mina Aérea*.
6. *Mina Aérea Ilegal*.

### 4.3.6 Uso de Chips

La última métrica de las categorías de pieza que queda por tratar son los Chips, mostrados en la figura 4.8.

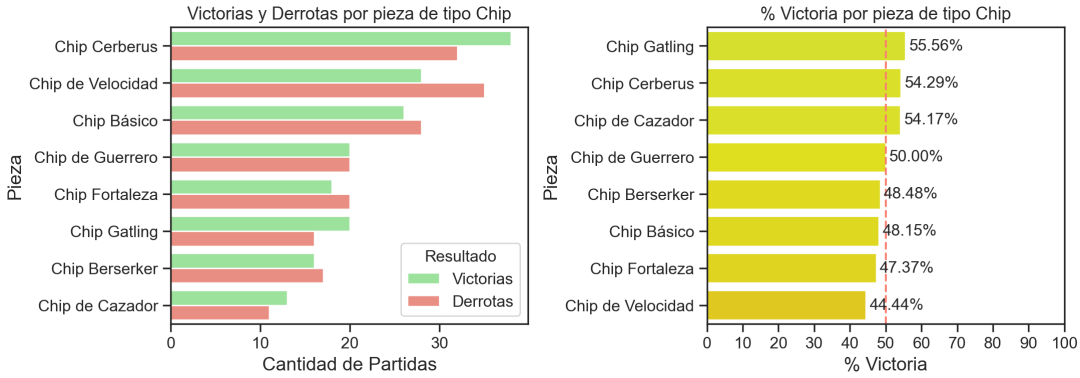


Figura 4.8: Cantidad de partidas y tasa de victoria para piezas de tipo “Chip”.

Observando la gráfica de las tasas de victoria, parece que los chips sí son un tipo de pieza que está equilibrada, a excepción de *Chip Gatling* y *Chip de Velocidad*, pero difieren menos del 1% del margen definido, que no es significativo.

El más usado ha sido el *Chip Cerberus*, probablemente por su estética. Hay tres chips con tasa de victoria positiva: *Chip Cerberus*, *Chip Gatling* y *Chip de Cazador*. Estos dos últimos no han sido tan escogidos como la primera opción. Esto implica que el sistema de decisión ha permitido identificar posibles piezas que potencialmente están desequilibradas.

Respecto el contenido para Patreon, se decidió incluir todos los Chips. Esto es debido a que es una pieza que simplemente modifica características de las piezas seleccionadas del robot (veáse la subsección 1.4.3) y el hecho de que haya más abundancia mejora la experiencia de usuario.

Una vez analizada la interacción de los jugadores con la selección de piezas, se puede proceder al análisis de tasa de victorias.

## 4.4 Análsis de la tasa de victorias por tipo de pieza.

Es interesante realizar una comparativa sobre la tasa de victoria, en lugar de por pieza, entre categoría de piezas. De esta manera, se puede conocer qué sección de piezás es más significativa para ganar una partida.

Esto apoya al sistema de decisión del equilibrio de piezas, teniendo una vista general de todas las piezas que permita decidir, sin tener que entrar a analizar casos aislados para comprobar si una categoría de piezas está equilibrada.

Para ello, se ha utilizado la raíz cuadrada del error medio cuadrático (RMSE) entre la tasa de victoria de las piezas respecto del 50%, que es valor esperado para contar una pieza como equilibrada. La figura 4.9 plasma esta métrica.

Observando la gráfica, hay dos categorías que parecen significativas a la hora de determinar el resultado de una partida, los Robots (*Battlecores*, barra "*Bcr*") y las Vainas (barra "*pod*").

De los resultados presentados en las gráficas en las subsecciones 4.3.1 y 4.3.5, se puede corroborar que dichos resultados son correctos, teniendo piezas muy alejadas del 50% esperado.

Como consecuencia, se ha evaluado el impacto de la categoría de piezas en el resultado de los combates, obteniendo una métrica que permite determinar si es necesario inferir en los resultados de una categoría concreta para decidir si ajustar determinadas piezas. Esta métrica ha permitido desarrollar un sistema que apoye a la toma de decisiones con el que el equipo puede disponer de una estimación del rendimiento de las piezas dentro del juego.

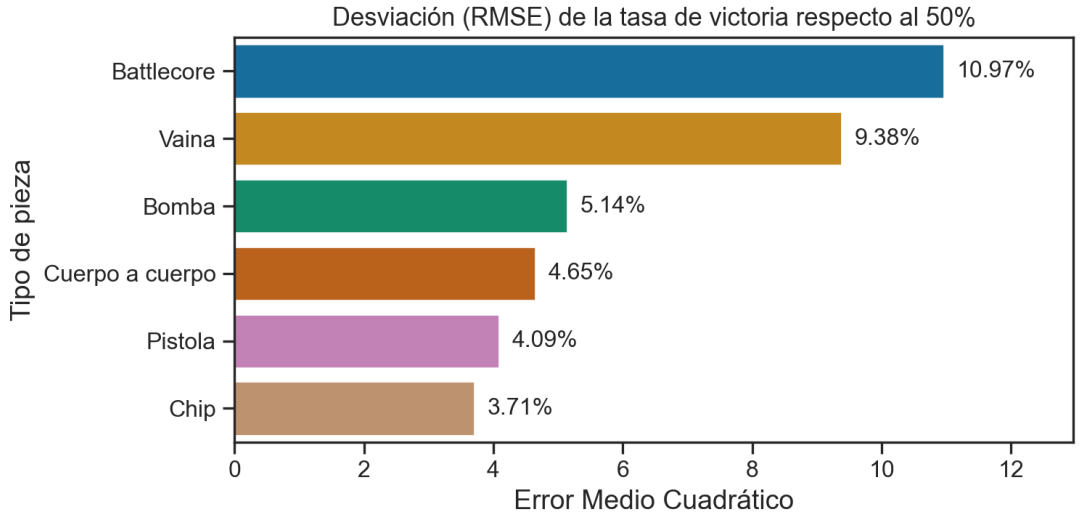


Figura 4.9: Raíz cuadrada del Error Medio Cuadrático de las tasa de victoria de las piezas respecto al 50%.

## 4.5 Análisis de mecánicas

El análisis de la interacción del jugador con las diferentes mecánicas de combate también resulta de sumo interés. El objetivo es conseguir un proceso de creación de métricas con el que detectar qué mecánicas han sido pasadas por alto. De esa manera, se puede conseguir conocimiento sobre las funciones que convendría enfatizar o resaltar en los tutoriales del juego.

Para mostrarlo, se creará un diagrama de violín para hacer un análisis descriptivo de cada atributo que refleja una mecánica en la entidad que representa, a su vez, la interacción del usuario con el robot. Las mecánicas disponibles están descritas en la tabla 4.2.

El análisis descriptivo que representa el uso de dichas mecánicas está plasmado en la figura 4.10. Se puede apreciar una diferencia de magnitud grande de cada violín, debido a que lo esperado es que ciertas mecánicas se usen muchas menos veces que otras, por ejemplo la mecánica de *overdrive* se usa entre una o dos veces por partida como mucho. Esto genera un efecto de dispersión de los datos.

La figura 4.11 muestra los datos pasados por la función logarítmo para reducir dicha dispersión y poder leer mejor la información. Esta gráfica facilita la lectura de los datos y con ello, la aplicación del sistema de decisión que apoye a la identificación de mecánicas poco usadas. Se puede observar que las mecánicas que se usan menos son la de *Parry* y Habilidad Especial (*Overdirve*), ya que sus medianas son muy cercanas a cero.

El resultado de esta última métrica puede ser motivo de preocupación, debido a que la habilidad especial representa una de las funciones más llamativas del juego, tanto por su

| Mecánica | Descripción |
|---|---|
| Disparo con pistola | Permite disparar ráfagas de balas utilizando la pieza de tipo Pistola. |
| Lanzamiento de bomba | Lanza el misil de la pieza de tipo Bomba para generar una explosión en el suelo. |
| Cuerpo a cuerpo | Permite golpear al oponente a corta distancia. |
| Vainas (Pod) | Despliega drones teledirigidos automáticamente para acorralar al oponente. |
| Bloqueo | Permite cubrirse de ataques y bloquear el daño recibido. |
| Desvío | Si se bloquea justo antes de recibir un ataque, se reduce completamente el daño. |
| Habilidad especial (Overdrive) | Cada robot dispone de una habilidad especial con la que desatar su poder temporalmente. |

Tabla 4.2: Resumen de las mecánicas implementadas en el juego.

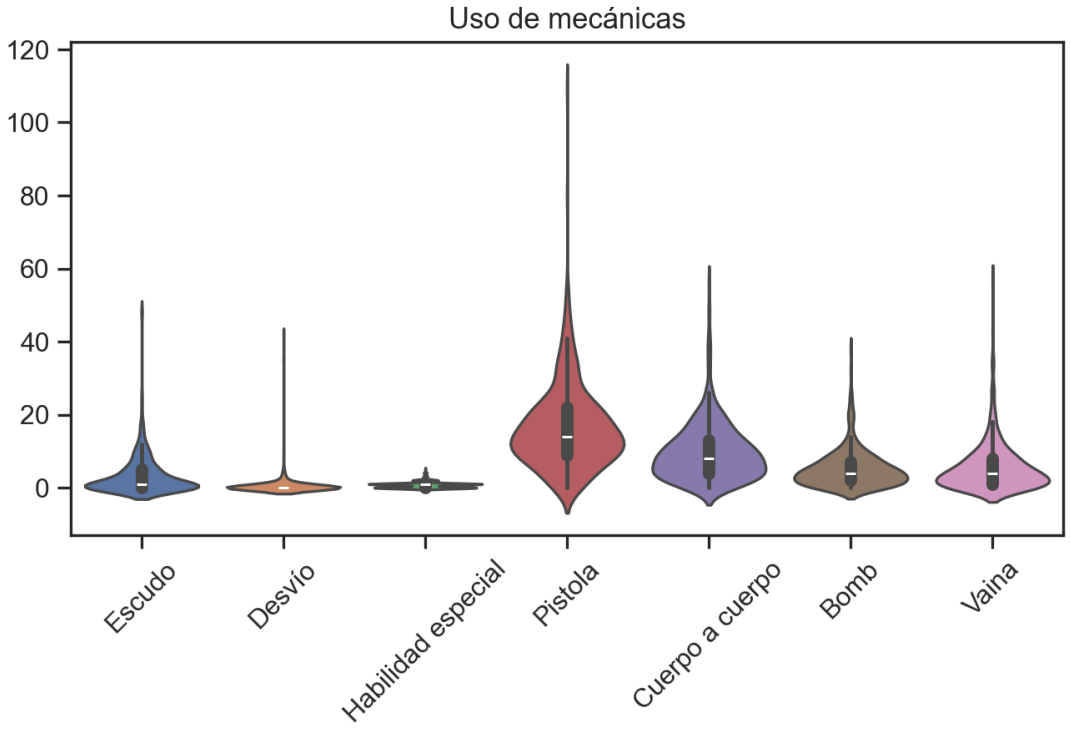


Figura 4.10: Análisis descriptivo del uso de mecánicas del juego.

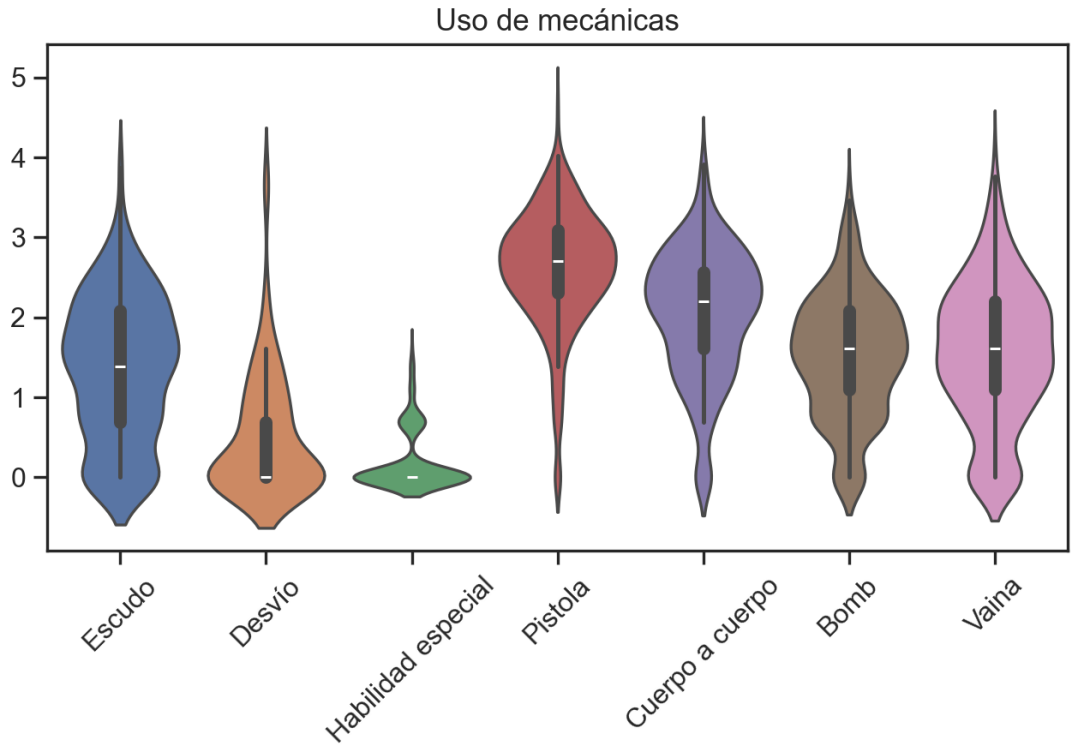


Figura 4.11: Análisis descriptivo del uso de mecánicas del juego con reducción de dispersión.

espectacularidad visual, como por sus implicaciones a nivel de juego. Tan bajo uso sugiere que una parte que puede destacar de la experiencia de juego no está siendo aprovechada.

Por ello, es interesante usar el sistema para estudiar mécanicas de forma aislada. La figura 4.12 muestra el diagrama de violín, sin aplicar la transformación logarítimica, de la mecánica *Overdrive*.

La mediana está en 1, lo que significa que en la mitad de partidas llega a usarse al menos una vez la habilidad especial. Sin embargo, la comparación entre las anchuras 0 y 1 es preocupante. De modo que, la métrica ha permitido identificar que se tiene que plantear algún cambio a nivel de juego para que la habilidad especial se utilice más veces.

En la siguiente capítulo, se tratarán las conclusiones que se han podido obtener gracias al experimento.

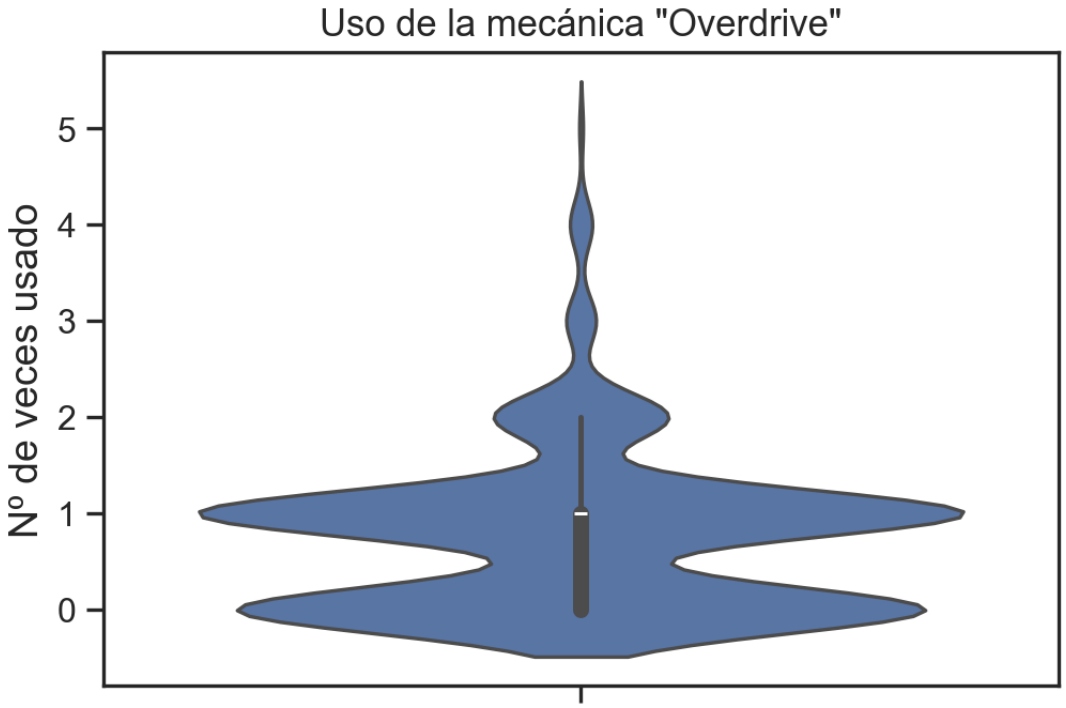


Figura 4.12: Uso de la mecánica *Overdrive*, la habilidad especial de los robots.

# Capítulo 5

# Conclusiones y trabajos futuros

Gracias a los diferentes análisis, se pudo cumplir el objetivo de este Trabajo de Fin de Máster, que era incluir tratamiento de datos en decisiones estratégicas de un estudio independiente. A pesar de no contar con tantos recursos ni fuentes de datos como pueden tener medianas y grandes empresas para tratar datos, se ha podido conseguir conocimiento de la experiencia de juego de los usuarios para así apoyar a la decisión estratégica del diseño de juego.

Con las gráficas tratadas en el capítulo 4, se han obtenido los siguientes sistemas de decisión:

- Un sistema para ayudar a decidir las piezas incluidas el primer mes de Patreon.
- Un sistema para evaluar el uso de las mecánicas, lo cual permite ayudar a las decisiones de diseño de juego para enfatizar mecánicas.
- Un sistema para tratar cómo de determinante es una categoría de piezas para ganar o perder una partida.
- Un sistema para considerar si las partidas tienen una duración deseada. La duración de las partidas puede definir la duración del juego.

## 5.1 Conclusiones principales

Se han podido tomar decisiones basadas en datos a nivel táctico y a nivel estratégico.

A nivel táctico, se han podido emplear los resultados obtenidos para comprender la interacción del usuario con el juego y definir el contenido de la versión del juego para el

lanzamiento del Patreon. Se ha podido evaluar el rendimiento de piezas obteniendo conclusiones de cuáles son las más escogidas y con qué piezas se tiende a ganar más combates. También se ha adquirido conocimiento sobre lo equilibradas que están las piezas, permitiendo saber cómo de justo puede sentir un jugador un combate. Además, se ha adquirido conocimiento sobre el uso de las mecánicas, lo que ha permitido al equipo evaluar si el jugador aprovecha todas las opciones disponibles y disfruta plenamente del contenido del juego.

En relación al aspecto estratégico, se ha comprobado que las métricas pueden apoyar al desarrollo del proyecto aprovechando los usuarios implicados en la comunidad y los eventos presenciales. Se ha cumplido el objetivo de corroborar que aplicar analítica de datos, siendo un estudio de desarrollo independiente, tiene un efecto positivo en los resultados obtenidos. Además, se ha incluido en la forma de trabajar del equipo la toma de decisiones basada en datos. En la sección 5.4 se tratarán trabajos a realizar en el horizonte a medio y largo plazo.

## 5.2 Implicaciones para operaciones y plan de negocio

Se ha conseguido que los resultados obtenidos sean significativos para las operaciones y el plan de negocio del producto.

A nivel operacional:

- Se han utilizado los datos para decidir qué contenido debería incluir la primera versión del juego disponible para los miembros de Patreon que tenga la segunda membresía o más (veáse la subsección 1.4.2).
- Se ha generado un sistema capaz de identificar piezas que tienen que ser revisadas en base a su variación en la tasa de victoria. Por lo que se han identificado piezas que tienen que ser revisadas.
- Se han descubierto qué mecánicas de juego son las más usadas y se ha llegado a la conclusión de que la habilidad especial se ha pasado por alto, con la mitad de casos en un solo uso o por debajo. Por lo que se ha identificado la necesidad de explicar cómo usar esa mecánica y mejorar a nivel visual de qué modo podría entenderse mejor.
- Se han empleado tecnologías e interconectado diferentes herramientas con el videojuego.

Respecto al plan de negocio:

- La arquitectura del sistema ha sido diseñada para que sea escalable y modular, lo que garantiza flexibilidad para abordar futuras fases del proyecto.
- Se ha conseguido un sistema para estimar la duración del juego, apoyando una fórmula en datos de usuarios. La duración del juego es un factor significativo para el precio del videojuego, por lo que dicha métrica es relevante para el plan de negocio.
- Se ha definido un flujo de trabajo en el que el equipo utiliza las conclusiones obtenidas a partir de los datos recogidos durante el mes, para decidir los pasos a seguir y las tareas a realizar de cara al mes siguiente.

## 5.3 Lecciones aprendidas

Con este Trabajo de Fin de Máster se han aprendido muchas lecciones, mayoritariamente sobre análisis de datos, centrando su aplicación al videojuego. Las lecciones aprendidas se detallan a continuación:

1. A nivel de desarrollo de videojuegos, se aprendió como gestionar dependencias entre escenas y a acoplar y desacoplar sistemas de una arquitectura. Además, se ha conseguido incluir y excluir contenido para diferentes versiones de un mismo proyecto.
2. En cuanto a bases de datos, se ha conseguido definir el modelo con el que representar la interacción de un usuario con el videojuego *Battlecore Robots*.
3. Respecto al análisis de datos, se ha aprendido bastante de las técnicas y tecnologías comúnmente aplicadas, sobre todo enfocando en la industria de los videojuegos.
4. Se ha aprendido a definir métricas que resulten útiles. No solo para uso personal, si no para ser comprendidas por un grupo de personas y contar una historia sobre unos datos.
5. Se ha conseguido crear documentación reproducible e interactiva para mostrar métricas y conclusiones obtenidas a partir de datos captados.

Lo aprendido en el Máster ha sido crucial para la realización del proyecto. Concretando asignaturas, la estadística de "Modelización y Tratamiento de la Incertidumbre" ha sido imprescindible para comprender las métricas. De "Análisis de Big Data" se ha extraído la lectura y procesamiento de datos con Quarto y Python.

Extrapolando el contenido de todas las asignaturas como un conjunto global, puede afirmarse que todo lo aprendido a lo largo del Máster ha resultado relevante para la realización de este proyecto, especialmente en lo relativo a la teoría de la decisión y al valor del trabajo con datos como apoyo para la toma de decisiones.

## 5.4 Trabajos futuros

Por último, lo conseguido y aprendido a través de este Trabajo de Fin de Máster abre muchas puertas para abordar posibles trabajos futuros. Además, la arquitectura y el modelo de la base de datos se ha planteado en torno a futuros usos.

Un primer paso sería implementar la captación de datos con tecnologías que permitan hacerlo en línea, como pueden ser tecnologías en nube. En el Trabajo de Fin de Grado se aplicaron tecnologías de la proovedora de nube de Amazon, AWS [1], por lo que puede ser un punto de partida para incorporar recogida de datos remotamente.

Una limitación del proyecto era el tiempo. Con las lecciones aprendidas y pudiendo enfocar el esfuerzo en desarrollar, se pueden aplicar nuevas técnicas de análisis en el proyecto. A fecha de la finalización de la memoria, hay una base suficiente de miembros suscritos a la membresía de nivel 2 o más y que prueban la nueva versión mes a mes. Por ello, a corto plazo, se pueden utilizar técnicas de análisis de supervivencia [24], para comprender qué es lo que lleva a un jugador a salir del juego o no volver a probarlo.

Como se mencionó en la subsección 1.4.3, se desarrollará un "modo historia" para que el jugador consiga el contenido del juego mientras conoce los personajes del mundo de *Battlecore Robots*. El tratamiento de datos presenta una gran oportunidad, puesto que se podrían evaluar características como dificultad percibida de un combate o satisfacción de la recompensa mediante encuestas dentro del videojuego, aprovechando la arquitecutra definida.

Se quiere también centrar en el análisis de comportamiento de usuarios, creando sistemas de decisión que apoyen al videojuego como producto y maximizando los beneficios posibles.

A nivel personal, se va a investigar sobre análisis de comportamiento de usuarios y estadística aplicada a la toma de decisión, basada en datos en la industria de los videojuegos.

# Siglas

**API** Interfaz de Programación de Aplicaciones (*Application Programming Interface*). 37

**ER** Entidad Relacion. 5

**IDE** Integrated Development Enviroment (Entorno de Desarrollo Integrado). 20

**TFM** Trabajo de Fin de Máster. 10

**UML** Lenguaje Unificado de Modelado (*Unified Modeling Language*). 42

# Glosario

***Build*** Versión de un videojuego compilada en un momento determinado del desarrollo.. 38

***Data-Driven Decission Making*** Uso de hechos, métricas y datos para guiar decisiones de negocios estratégicas que se alineen con las metas, los objetivos y las iniciativas de una organización.. 31

***Heap* de Memoria** Región de memoria utilizada por los programas para almacenar datos durante la ejecución del programa.. 37

**Código Abierto** Software cuyo código fuente es accesible y modificable por cualquier persona.. 18

**Jugabilidad** Forma en que un jugador interactúa con un juego, ya sea un videojuego o un juego de mesa.. 25

**Mecánicas** Reglas, acciones y sistemas que definen cómo interactúa el jugador con el juego y cómo se desarrolla la experiencia de juego.. 9

**SQL** Structured Query Language, traducido en español como lenguaje de consulta estructurada, es un lenguaje de dominio específico utilizado en programación, diseñado para administrar y recuperar información de sistemas de gestión de bases de datos relacionales.. 20

**Tuplas** Secuencias ordenada de elementos, que pueden ser de diferentes tipos de datos. En bases de datos, una tupla representa un registro o fila en una tabla.. 49

# Apéndice A

# Sistema de captación de datos en Unity y .NET

Se detallará el funcionamiento de las clases de código y sistemas creados para la captación de datos dentro del videojuego, en Unity. El código fuente está accesible en el repositorio: https://github.com/drowlerd/Master-Thesis-Roberto-Garcia.git.

El código A.5 muestra el código para transformar datos de los objetos instanciados en C# e Unity a PostgreSql. Después de entender cómo había que tratar los valores para que fuesen válidos en C#, se crearon las clases para crear toda la estructura necesaria del sistema de recogida de datos.

```
public static string FormatSqlValue(object value)
    {
        if (value == null)
            return "NULL";

        return value switch
        {
            string s => $"'{s.Replace("'", "''")}'", // Ajustar comillas en strings
            bool b => b ? "TRUE" : "FALSE", // PostgreSQL usa TRUE/FALSE para booleanos
            float f => $"'{f.ToString("0.00", System.Globalization.CultureInfo.InvariantCulture)}'",
            DateTimeOffset dt => $"'{dt:yyyy-MM-dd HH:mm:ss}'",// Formato SQL estandar
            float[] fArray=> "'{"+$"{String.Join(',',fArray.Select(f=>f.ToString("0.00", System.Globali
            TimeSpan t => $"'{t.ToString("c")}'",
            _ => value.ToString() // Para numeros y otros tipos
        };
    }
```

Código A.5: Código de la métodos para transformar valores de C# a PostgreSql.

Se recomienda leer los archivos de código que se han incluidos en el repositorio. Para facilitar su lectura, las clases se dividen del siguiente modo:

- Carpeta `BCR_PsqlItems`: contiene las entidades traducidas a objetos de C#.
- Carpeta `BCR_PsqlItemBuilders`: contiene clases para convertir una instancia de C# al mensaje para mandarlo a la cola de RabbitMQ.
- Carpeta `BcrRabbitMQ`: contiene las clases que se han usado para trabajar con RabbitMQ. Contiene únicamente la clase `Bcr_RabbitMQProducer` un *Singleton* para que las clases de la carpeta del punto anterior puedan mandar mensajes a la cola.
- Carpeta `DataCollectors`: son los sistemas en las escenas superpuestas que, apóyandose de los eventos del juego, determinar en qué momento se tiene que mandar qué mensaje.

# Apéndice B

# Desarrollo del documento Quarto con Python

Por motivos de extensión y legibilidad, no se incluirá todo el código fuente en el apéndice, se presentarán fragmentos representativos. Tanto el código del documento como el archivo web renderizado se encuentran disponible en el repositorio: `https://github.com/drowlerd/Master-Thesis-Roberto-Garcia.git`.

Se desarrolló el documento Quarto utilizando python. Se definierón métodos modulables para tratar las piezas sin repetir código, las funciones están dentro del capítulo "Definición de métodos y ajustes del documento" del documento Quarto.

Para guardar los secretos de la base de datos, se utilizó un archivo YAML [56], que es un formato de archivos de configuración más fácil de leer y comprender que otras alternativas actuales.

La cliente del paquete `psycopg2` utiliza los valores obtenidos del archivo para establecer la conexión (código B.6).

Por último, indicar las opciones de Quarto para renderizar el documento, contenidas en el código B.7.

```
with open("secrets.yml", "r") as f:
    secrets_file = yaml.safe_load(f)["experiment_I"]
# Conexión a PostgreSQL obteniendo datos del archivo YAML
conn = psycopg2.connect(
    dbname=secrets_file["DB_NAME"],
    user=secrets_file["DB_USER"],
    password=secrets_file["DB_PASSWORD"],
    host=secrets_file["DB_HOST"],
    port=secrets_file["DB_PORT"]
)

```

Código B.6: Código para establecer la conexión con la base de datos mediante los secretos del archivo YAML.

```
title: "Experimento I: Gamegen - Gráficos y análisis de los datos recogidos"
execute:
  # cache: true
  warning: false
  fig-align: center
format:
  html:
    fig-width: 10                          # Tamaños de las figuras
    fig-height: 5
    page-layout: full                      # Indica que el documento tiene que ocupar toda la pági
    grid:                                  # Para definir el tamaño de las areas del documento
      sidebar-width: 300px
      body-width: 1200px
      margin-width: 200px
      gutter-width: 1.5rem
    table-of-contents: true                # Para mostrar el índice a la derecha
    toc-location: left
    cap-location: margin                   # Para poner el pie de las figuras en el margen de la p
    reference-location: margin             # Para mostrar las referencias y citaciones a la derech
    citation-location: margin
    csl: vancouver.csl                     # Para mostrar las referencias y citaciones a la derech
    toc: true                              # Incluye el indice
    embed-resources: true                  # Para que se renderice tanto el texto como las métrica
    code-fold: true                        # Para que se muestre el código en cajas pegables, así

bibliography: experimento_I_bibliografia.bib    # La bibliografía del documento.


```

Código B.7: Ajustes para el renderizado del documento de Quarto.

# Referencias


[1] *¿Qué es AWS? - Computación en la nube con Amazon Web Services*. https://aws.amazon.com/es/what-is-aws/. (Visitado 17-06-2025).

[2] *¿Qué es Docker? | IBM*. https://www.ibm.com/es-es/think/topics/docker. Jun. de 2024. (Visitado 18-06-2025).

[3] *¿Qué es el middleware? - Explicación del software middleware - AWS*. https://aws.amazon.com/es/what-is/middleware/. (Visitado 19-06-2025).

[4] *¿Qué es Miro?* https://help.miro.com/hc/es/articles/360017730533. Mayo de 2025. (Visitado 14-06-2025).

[5] *¿Qué es Patreon?* https://support.patreon.com/hc/es-es/articles/204606315–Qu%C3%A9-es-Patreon. Jun. de 2023. (Visitado 16-06-2025).

[6] *3.6. Inheritance*. https://www.postgresql.org/docs/17/tutorial-inheritance.html. Mayo de 2025. (Visitado 10-06-2025).

[7] *Acerca de GitHub y Git - Documentación de GitHub*. https://docs-internal.github.com/es/get-started/start-your-journey/about-github-and-git. (Visitado 14-06-2025).

[8] Anders Drachen, Magy Seif El-Nasr y Alessandro Canossa. «Game Analytics – The Basics». En: *Game Analytics: Maximizing the Value of Player Data*. London: Springer, 2013, págs. 13-40. ISBN: 978-1-4471-4768-8.

[9] *Aprende un idioma completamente gratis*. https://es.duolingo.com/. (Visitado 18-06-2025).

[10] Michael J. A. Berry y Gordon Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*. 2nd ed. Indianapolis, Ind: Wiley Pub, 2004. ISBN: 978-0-471-47064-9.

[11] S. Tarık Çetin. *Starikcetin/Eflatun.SceneReference*. Jun. de 2025. (Visitado 09-06-2025).

[12] «Custom Robo (serie)». En: *Wikipedia, la enciclopedia libre* (ene. de 2024). (Visitado 14-06-2025).

[13] DEV (Desarrollo Español de Videojuegos). *Libro Blanco DEV 2024*. Mayo de 2025.

[14] *Direccionables: Planificación y buenas prácticas*. https://unity.com/blog/engine-platform/addressables-planning-and-best-practices. (Visitado 09-06-2025).

[15] *Docker: Accelerated Container Application Development*. Abr. de 2025. (Visitado 18-06-2025).

[16] dotnet-bot. *Action<T> Delegado (System)*. https://learn.microsoft.com/es-es/dotnet/api/system.action-1. (Visitado 17-06-2025).

[17] Ramez Elmasri y Sham Navathe. *Fundamentals of Database Systems*. 6. ed. Boston: Addison-Wesley, 2011. ISBN: 978-0-13-608620-8.

[18] *Event Queue · Decoupling Patterns · Game Programming Patterns*. https://gameprogrammingpatterns.com/event-queue.html. (Visitado 17-06-2025).

[19] Sarah Fields. *Estimated Xbox Series X and S Console Sales Numbers Revealed*. https://gamerant.com/xbox-series-x-s-console-sales/. Feb. de 2025. (Visitado 18-06-2025).

[20] *GameGen X (2025) | Virtual Soul*. https://virtualsoul.es/gamegenX.html. (Visitado 01-05-2025).

[21] Jordi Duch Gavaldà y Heliodoro Tejedor Navarro. «Introducción a los videojuegos». En: ().

[22] *Git - Fundamentos de Git*. https://git-scm.com/book/es/v2/Inicio—Sobre-el-Control-de-Versiones-Fundamentos-de-Git. (Visitado 14-06-2025).

[23] PostgreSQL Global Development Group. *PostgreSQL*. https://www.postgresql.org/. Feb. de 2025. (Visitado 14-02-2025).

[24] *Introduction to Survival Analysis with Scikit-Survival — Scikit-Survival 0.24.1*. https://scikit-survival.readthedocs.io/en/stable/user_guide/00-introduction.html. (Visitado 17-06-2025).

[25] Wajeeh Khan. *42 Video Game Industry Statistics 2023*. Mayo de 2023. (Visitado 18-06-2025).

[26] *LaTeX Workshop - Visual Studio Marketplace*. https://marketplace.visualstudio.com/items?itemName=James-Yu.latex-workshop. (Visitado 25-03-2025).

[27] Guillermo O. Lavagnino. *Kahoot! en las Aulas*. https://www.educrear.com.ar/es/recursos/kahoot-educacion-gamificacion-juegos/. Service. (Visitado 18-06-2025).

[28] «Nintendo Switch 2». En: *Wikipedia* (jun. de 2025). (Visitado 18-06-2025).

[29] *NuGet Gallery | Home*. https://www.nuget.org/. (Visitado 14-06-2025).

[30] *OpenGL - The Industry Standard for High Performance Graphics*. https://www.opengl.org/. (Visitado 14-06-2025).

[31] «*OXO* (Video Game)». En: *Wikipedia* (jun. de 2025). (Visitado 18-06-2025).

[32] Tajammul Pangarkar. *Game-Based Learning Statistics and Facts (2025)*. Ene. de 2025. (Visitado 18-06-2025).

[33] *Patreon: Subscriber and Creator Statistics for 2024*. https://backlinko.com/patreon-users. Mar. de 2023. (Visitado 16-06-2025).

[34] *Plataforma y editor de desarrollo 3D en tiempo real*. https://unity.com/products/unity-engine. (Visitado 14-06-2025).

[35] *Quarto*. https://quarto.org/. (Visitado 13-06-2025).

[36] *R: The R Project for Statistical Computing*. https://www.r-project.org/. (Visitado 14-06-2025).

[37] *RabbitMQ: One Broker to Queue Them All | RabbitMQ*. https://www.rabbitmq.com/. (Visitado 14-02-2025).

[38] *Rider: El IDE .NET multiplataforma de JetBrains*. https://www.jetbrains.com/es-es/rider/. (Visitado 25-03-2025).

[39] Magy Seif El-Nasr, Anders Drachen y Alessandro Canossa, eds. *Game Analytics: Maximizing the Value of Player Data*. London: Springer, 2013. ISBN: 978-1-4471-4768-8 978-1-4471-4769-5. DOI: 10.1007/978-1-4471-4769-5. (Visitado 14-02-2025).

[40] Raphael Araújo e Silva. *pgModeler - PostgreSQL Database Modeler*. https://pgmodeler.io. (Visitado 25-03-2025).

[41] *Singleton · Design Patterns Revisited · Game Programming Patterns*. https://gameprogrammingpatterns.com/singleton.html. (Visitado 18-06-2025).

[42] *Software de desarrollo de juegos: Crea juegos 2D y 3D*. https://unity.com/games. (Visitado 14-06-2025).

[43] «Steam». En: *Wikipedia, la enciclopedia libre* (feb. de 2025). (Visitado 14-02-2025).

[44] *Steam Game Release Summary by Year*. https://steamdb.info/stats/releases/. (Visitado 18-06-2025).

[45] stevewhims. *Escritura fuerte - Win32 apps*. https://learn.microsoft.com/es-es/windows/win32/rpc/strong-typing. (Visitado 14-06-2025).

[46] *The First-Ever Patreon Creator Census*. https://news.patreon.com/articles/the-first-ever-patreon-creator-census. (Visitado 16-06-2025).

[47] *Un espacio de trabajo que conecta tus notas, documentos y tareas*. https://www.notion.com/es-es/personal. (Visitado 14-06-2025).

[48] *Unity - Manual: UnityEvents* (*Eventos de Unity*). https://docs.unity3d.com/es/530/Manual/UnityEvents.html. (Visitado 17-06-2025).

[49] *Unity-Inventory-System/Assets/_Project/Scripts/SceneManagement at Master · Adammyhre/Unity-Inventory-System*. https://github.com/adammyhre/Unity-Inventory-System/tree/master/Assets/_Project/Scripts/SceneManagement. (Visitado 09-06-2025).

[50] *Uv*. https://docs.astral.sh/uv/. (Visitado 13-06-2025).

[51] *Ventas totales de todas las consolas vendidas de PlayStation de la historia*. Nov. de 2024. (Visitado 18-06-2025).

[52] *Video Game Market Size, Share And Growth Report, 2030*. https://www.grandviewresearch.com/industry-analysis/video-game-market. (Visitado 18-06-2025).

[53] *Visual Studio Code - Code Editing. Redefined*. https://code.visualstudio.com/. (Visitado 14-06-2025).

[54] *Welcome - SQL Database Reference Material - Learn Sql, Read an Sql Manual, Follow an Sql Tutorial, or Learn How to Structure an SQL Query!* https://www.sql.org/. (Visitado 18-06-2025).

[55] *Welcome to Python.Org*. https://www.python.org/. Jun. de 2025. (Visitado 13-06-2025).

[56] *YAML: qué es, usos, sintaxis y ejemplos*. https://www.redhat.com/es/topics/automation/what-is-yaml. (Visitado 18-06-2025).

[57] *Your connected workspace for wiki, docs & projects*. https://www.notion.so. (Visitado 25-03-2025).