ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

SUMMON STORM

MODO MULTIJUGADOR Y FUNCIONALIDAD DEL LADO DEL SERVIDOR

SUMMON STORM

MULTYPLAYER MODE AND SERVER-SIDE FUNCTIONALITIES

Realizado por
Manuel Francisco Álvarez Gaona
Tutorizado por
Gabriel Jesús Luque Polo
Coordinado por
Enrique Domínguez Merino
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Junio de 2017

Fecha defensa:
El Secretario del Tribunal

Resumen

En el presente Trabajo Fin de Grado en grupo se pretende crear un videojuego que permita realizar partidas entre varios usuarios utilizando diferentes equipos, o contra una inteligencia artificial. Cuyo objetivo final será el entretenimiento de los usuarios.

El juego resultante, simula un juego de mesa que enfrenta a los jugadores sobre un tablero de casillas hexagonales. Sobre estas casillas habrá unidades que se podrán mover y atacar unas a otras, con el objetivo de derrotar a la unidad principal del jugador rival.

Es este proyecto en concreto, se centra en ofrecer a los usuarios la capacidad de jugar partidas contra otros usuarios a través del envío de mensajes a diferentes servidores que a su vez se encargarán de realizar las gestiones/consultas necesarias en la base de datos. Estos mensajes se enviarán y recibirán de forma automática en función de las acciones realizadas por el usuario, para así facilitar la comunicación y evitar errores.

Palabras clave: videojuego, servidor, multijugador, base de datos, sockets, c#, mysql, unity3d

Abstract

The current final year project in group consists in developing a video game that allows to play matches against other users which are located on different computers, or play against an artificial intelligence. The final objective will be the entertainment of the users.

The resulting game simulates a board game that faces the players on a board of hexagonal sections. On these sections can have few units, that can be moved and attack each other, in order to defeat the main unit of the rival player.

This project specifically focuses on offering users the ability to play games against other users by sending messages by means of different servers that interact with the database, making the queries needed to manage all the game. These messages will be sent and received automatically based on the actions performed by the user,in order to make easy communication and avoid errors.

Keywords: video game, server, miltyplayer, data base, sockets, c#, mysql, unity3d

Índice general

Índice de figuras

Índice de códigos

Índice de cuadros

1

Introducción

1.1. Motivación

Actualmente el mundo de los videojuegos ofrece muchas posibilidades de estudio, trabajo y evolución tecnológica. Además, supone una gran fuente de entretenimiento para los usuarios que los juegan, así como también pueden ser una fuente de inspiración para futuros videojuegos.

Por ello, se ha decidido realizar un videojuego que permita a usuarios entretenerse y divertirse jugando partidas entre ellos desde distintos equipos, o contra una inteligencia artificial para esos momentos en los que no puedan jugar contra otros debido a que no deban abandonar la partida antes de tiempo, o cualquier otra razón.

1.2. Objetivos

Se ha decidido crear un videojuego, abarcando a la vez gran parte de las asignaturas del grado, y gran parte de las aptitudes necesarias para programar un juego completo. De forma que el trabajo será una aplicación de los conocimientos provenientes del grado a la implantación de un videojuego tanto para jugar en linea, para lo que se usarán bases de datos, conexiones cliente/servidor, etc., como para jugar contra el ordenador, aplicando conocimientos de Inteligencia Artificial, y aplicando los conceptos como la programación estructurada, el análisis de algoritmos o los patrones de diseño, a la programación del videojuego. El objetivo final es la programación del videojuego, con todo el aprendizaje sobre programación que ello conlleva, pero también existen subobjetivos en este Trabajo de Fin de Grado como el análisis de requisitos del producto a desarrollar, el uso de metodologías ágiles en grupo, o la evaluación de los resultados obtenidos, que han sido muy enriquecedores para nuestra formación.

Debido a la complejidad del objetivo considerado, se decidió realizar este Tabajo Fin de Grado en la modalidad de grupo. Entonces ese objetivo principal lo hemos dividido en tres partes, la común, la desarrollada en este proyecto, y la desarrollada por mi compañero Carlos de Miguel. De forma breve, los objetivos de este Trabajo Fin de Grado (la parte común y la mía) han sido: el análisis del producto a desarrollar, y el uso de un servidor centralizado en la que estarán todos los datos y mecanismos para permitir que dos usuarios (aunque estén en

equipos diferentes) puedan jugar entre sí.

1.3. Metodología de trabajo

Al ser un trabajo en grupo, ha sido necesaria la manipulación/modificación simultánea del proyecto, tanto para poder aportar bilateralmente nuevas funcionalidades al proyecto, como para que lo que cada alumno vaya haciendo esté disponible para el otro alumno de la mejor manera posible y más rápida. Unity5 proporciona una herramienta, Unity Collaborate [1], que permite a grupos, de forma gratuita, compartir y sincronizar proyectos, haciendo las veces de repositorio y de sistema de control de versiones. La parte de gestión tecnológica queda cubierta, por tanto, con esta herramienta. Para la parte de gestión del proyecto en cuestiones de planificación, división de tareas y retrospectiva, hemos usado una metodología ágil, Scrum, que hemos adaptado a nuestras necesidades, basándonos, eso sí, en los principios de la filosofía de trabajo ágil. Hemos realizado reuniones semanales en las que se descomponían los requisitos más prioritarios en tareas, y de las cuales decidíamos cual íbamos a desarrollar hasta la siguiente reunión, sin embargo no siempre han podido ser semanales, debido a nuestras necesidades de dedicar más o menos tiempo a otras cuestiones universitarias y de empleo. Pero no ha sido un problema, precisamente por la naturaleza flexible de la metodología Scrum.

1.4. Entorno tecnológico

1.4.1. Herramientas de desarrollo principales

Las herramientas de desarrollo que se han utilizado principalmente para desarrollar este trabajo han sido:

- **Visual Studio**: Puesto que el código del proyecto va a ser escrito con lenguaje **C#**, se ha decidido utilizar Visual Studio porque ofrece un entono de desarrollo amigable para este lenguaje.

- **Unity5**: Es un motor de videojuegos que ofrece un marco de trabajo potente para simular físicas, detectar colisiones, manejar modelos 3D y 2D, ejecutar animaciones y efectos gráficos y sonoros. Sin embargo, nuestro juego, por sus particularidades, no se ha visto demasiado apoyado en las funciones nativas de Unity5, sino que se ha tenido que programar tanto la lógica interna del juego, como las mecánicas de juego base, de forma casi completa, y, una vez implementadas, apoyarse en dichas funciones de nativas de Unity5 para mostrar por pantalla un entorno agradable en 3D que simule el estado de juego, y capte las interacciones con este.

- **MySQL**: Ofrece un sistema gestor de bases de datos muy completo y fácil de utilizar, además se ha utilizado **MySQL Workbench** como interfaz para trabajar con las bases de datos y no mediante consola, pues añade más facilidades a su uso y otras herramientas

como generados de Scripts de la base de datos completa además de permitir generar un diagrama de la misma..

Otra herramienta utilizada para el desarrollo del proyecto, más concretamente para el desarrollo del modelado inicial del proyecto y diagramas de secuencia, ha sido **Magic Draw**, no obstante, para los diagramas creados tras la finalización del código han sido realizados con la herramienta que proporciona **Visual Studio** para las clases ya creadas.

1.4.2. Herramientas secundarias

Además de las herramientas mencionadas anteriormente, se han utilizado **LaTeX** para el desarrollo de la memoria, **LogMeIn Hamachi** y **Evolve** para simular redes locales con equipos en distintas localizaciones para poder realizar pruebas del funcionamiento del proyecto y **Photoshop** para la edición de algunas imágenes.
Evolve fue el simulador de redes locales usado al principio, no obstante los desarrolladores de esta herramienta dejaron de ofrecer sus servicios por lo que se pasó a utilizar **LogMeIn Hamachi** que posee la misma funcionalidad pero más limitada tanto en número de usuarios como en velocidad.
El lenguaje de maquetación **LaTeX** ha añadido una necesidad de aprendizaje extra para poder utilizar la herramienta correctamente.

1.4.3. Organización de la memoria

En esta memoria, además del presente capítulo, consta de cuatro capítulos más y dos apéndices. El contenido de cada uno de ellos es el siguiente:

- En el capítulo 2 se explicará el producto a desarrollar, la base de datos necesaria, y la funcionalidad multijugador.
- En el capítulo 3 se explicará el desarrollo de los servidores, así como las pruebas realizadas.
- En el capítulo 4 se mencionarán y explicarán con menor detalle las implementaciones realizadas en teste Trabajo Fin de Grado, no contempladas en el anteproyecto.
- En el capítulo 5 se expondrán las conclusiones.

2

Análisis y modelado del juego

2.1. El producto a desarrollar

2.1.1. Definición de las reglas de juego y producto final

El primer proceso de la creación del juego se trata de definir de forma clara cuales son las reglas de este, cuantos jugadores participan, como lo hacen, y quien gana.
La redacción producto de este proceso, se encuentra en el **Anexo I: Reglas del juego**. No obstante en esta sección se pretende hacer un resumen que permita el entendimiento del funcionamiento del juego para facilitar la comprensión de las decisiones técnicas tomadas durante el desarrollo.

El juego simulará una batalla entre dos jugadores sobre un tablero de casillas hexagonales. En dicho tablero, los jugadores estarán representados con unas unidades principales llamadas Héroes, las cuales definen el objetivo de la partida: eliminar al Héroe rival para poder ganar la partida.
Mediante los héroes, ambos jugadores podrán invocar nuevas unidades sobre el tablero para que le ayuden a derrotar al rival. No obstante existen ciertas limitaciones para estas unidades invocadas: Puede haber un máximo de 6 unidades, además del héroe, a la vez en el tablero por cada jugador, así como también cada héroe tendrá un número limitado de unidades totales que podrá invocar (este límite puede variar dependiendo del héroe).
Estas unidades cuentan con las siguientes características:

-*Vida*: Al llegar a 0 la unidad muere.

-*Ataque*: Puntos de vida que restará a una unidad enemiga al atacar.

-*Defensa*: Daño que evita la unidad al recibir un ataque

-*Rango de Ataque*: Distancia a la cual una unidad puede realizar un ataque.

-*Rango de Movimiento*: Distancia por la que se podrá mover como máximo durante un turno.

Estas unidades, así como el héroe, pueden realizar una serie de acciones que vendrán en cierta medida delimitadas por algunas de las características mencionadas. Estas acciones serían:

- Movimiento: Mueve una unidad deseada desde la casilla en la que se encuentra, a otra casilla dentro de su rango de movimiento.

- Ataque: Inflige daño a un rival dentro de su rango de ataque para reducir su vida una cantidad igual a sus puntos de ataque. Si la unidad rival posee puntos de defensa, serán estos los que desciendan una cantidad igual al daño infligido. En caso de que reciba un daño superior a los puntos de defensa, la diferencia de puntos será obviada y dejará a la unidad sin puntos de defensa, pero sin reducir sus puntos de vida.

Por otro lado, estas unidades pueden hacerse más fuertes mediante evoluciones, al derrotar unidades enemigas. Es decir, cuando cualquier unidad propia derrota a una unidad rival, el héroe de dicho jugador obtiene una cantidad de puntos de evolución establecidos por la propia unidad derrotada. Estos puntos de evolución permiten evolucionar cualquier unidad propia sobre el tablero, mientras se tengan los puntos necesarios.
Es necesario indicar también, que cada jugador podrá realizar una única acción por turno, es decir, en cada turno un jugador únicamente podrá atacar con una unidad, mover una unidad, invocar una unidad o evolucionar una unidad, así como también podrá decidir no realizar ninguna acción y ceder el turno al rival. Ceder el turno al rival no tiene ningún beneficio para el jugador que lo realiza.

Con estas reglas ya se puede comprender el funcionamiento del juego. Por otro lado, como se ha mencionado al principio de esta sección, en el **Anexo I: Reglas del juego**, se encuentran las reglas del juego completamente detalladas.

2.1.2. Análisis de requisitos

El análisis de requisitos para este proyecto ha sido realizado en base a las reglas de juego mencionadas en la sección anterior y se encuentra reflejado en su completitud en el **Anexo II: Análisis de requisitos**.
El proceso de captación y análisis de requisitos, así como la estructuración del código y el modelado, se han hecho para el producto total, del que luego se han implementado las partes más prioritarias que funcionen como producto mínimo viable, siendo este el producto real del trabajo. Dichos requisitos que se proceden a implentar serían:
Requisitos funcionales

- El jugador puede:

1. Registrarse.
   1.1. Rellenará un formulario de datos obligatorios.
   1.2. El sistema guardará el nuevo usuario.

2. Identificarse.
   2.1. Introducirá los datos necesarios para identificarse.
   2.2. El sistema verificará los datos.
3. Iniciar una partida.
   3.1. Iniciar una partida en linea.
      3.1.1. El sistema emparejará a los jugadores.
   3.2. Iniciar una partida de un jugador.
      3.2.1. El sistema creará una partida local contra la IA.
   3.3. El sistema creará una partida nueva.
      3.3.1. El sistema posicionará el héroe elegido por cada jugador en su lado del tablero.
      3.3.2. Establecerá el turno inicial de forma aleatoria.
4. Invocar unidades de apoyo.
   4.1. El sistema proporcionará las unidades disponibles para invocar.
      4.1.1. El jugador puede elegir cuál de estas invoca.
   4.2. EL sistema establecerá las casillas disponibles para posicionar la unidad.
      4.2.1. El jugador puede elegir donde la posiciona.
5. Realizar acciones con una unidad durante la partida.
   5.1. Puede seleccionar una unidad propia.
      5.1.1. El sistema mostrará las acciones disponibles para dicha unidad.
   5.2. Mover una unidad.
   5.3. Atacar con la unidad seleccionada a una unidad enemiga.
      5.3.1. Puede eliminar una unidad enemiga.
   5.4. Evolucionar la unidad seleccionada.
      5.4.1. El sistema mostrará las posibles evoluciones.
         5.4.1.1. El jugador elegirá a que unidad evolucionará.
   5.5. Puede cancelar la acción.
6. Cambiar de turno.
7. Rendirse.
8. Ver información de la partida.
   8.1. Puede ver información detallada sobre las unidades en juego.
   8.2. Puede ver información sobre los jugadores.

- El sistema:

1. Detectará el final del turno.
   1.1. Detectará si el jugador quiere cambiar de turno.
   1.2. Detectará si al jugador no le quedan acciones disponibles.
   1.3. Detectará si se ha acabado el tiempo del turno.
   1.4. El sistema cambiará el turno al jugador contrario.
2. Detectará el final de la partida.
   2.1. Detectará si un jugador derrota al héroe rival.
   2.2. Detectará si un jugador decide rendirse.
   2.3. El sistema mostrará en pantalla el resultado de la partida.
   2.4. El jugador podrá volver al menú principal.

Requisitos No Funcionales

Para no extender demasiado el contenido de esta sección y, ya que los Requisitos no funcionales se encuentran incluidos en el **Anexo II: Análisis de requisitos**, se indicarán los apartados que se han implementado de dicha sección o en su defecto el contenido específico de los apartados de dicha sección si no han sido implementados por completo, sin profundizar en detalles:

- Requisitos de Aspecto: en los cuales se definen aspectos de la visualización de la interfaz.

- Requisitos de Facilidad de Uso y Aprendizaje: los cuales definen el tiempo que un usuario tardará en aprender a jugar, así como la facilidad de uso del mismo.

- Requisitos de Funcionamiento: Requisitos de Seguridad Crítica: revisión de las acciones enviadas al servidor.

2.2. La base de datos

Tras el análisis de los requisitos y con objetivo de mantener un esquema sencillo para facilitar futuras mejoras, se decidió incorporar en la base de datos, las menos tablas posibles. Como se observa en la Figura 2.1, principalmente existe una para la información del usuario y otra para los datos de las diferentes unidades. En los siguientes párrafos se detallarán cada una de ellas. Como sistema de gestión de bases de datos (SGBD) se decidió utilizar MySQL por su amplio uso.

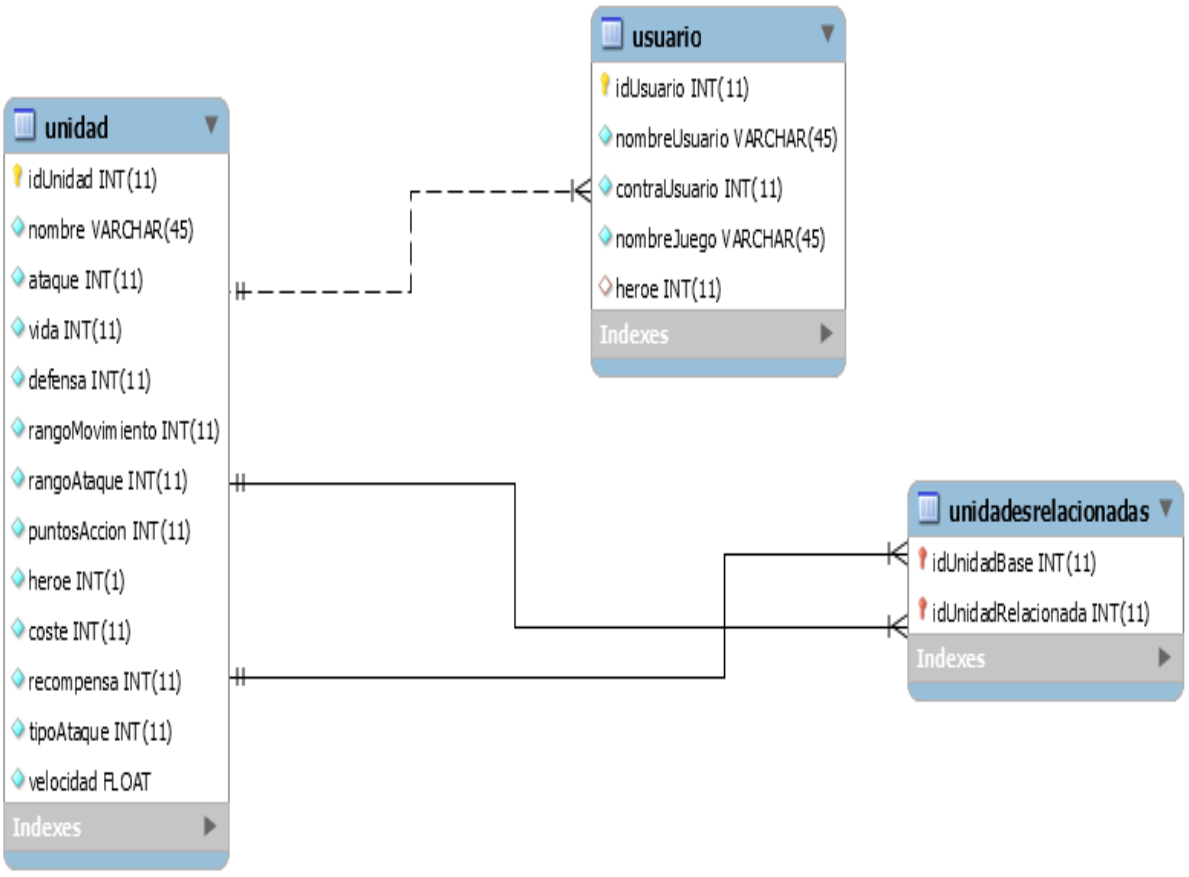


Figura 2.1: Diagrama de la base de datos

1. La tabla **usuario** contiene toda la información referida al usuario como su nombre de usuario (**nombreUsuario**) y contraseña (**contraUsuario**) utilizados en la identificación. Para no tener un acceso directo a la contraseña de los usuarios, esta es almacenada como un valor numérico que representa el *Hash* de la cadena de caracteres de la contraseña. La clave primaria de esta tabla es el valor numérico **idUsuario**, el cual es no nulo y autoincremental.
   Además contiene dos valores más que permitirían futuras ampliaciones:

   -**nombreJuego**: Representará el nombre que el usuario quiere mostrar para identificarse entre otros jugadores durante las partidas. La idea de este campo es poder otorgar un pequeño nivel de seguridad extra a los usuarios del juego, permitiendo mostrar un nombre distinto al nombre de usuario que será utilizado en la identificación de jugadores.

   -**heroe**: Este campo almacena el identificador de la unidad Heroe que será utilizada por el jugador. Como actualmente solo se dispone de un Heroe, este campo se genera automáticamente con su identificador.

2. La tabla **unidad**, aquí se incluyen las características de las unidades que mencionamos en la sección previa como vida, ataque, defensa, rango de movimiento, etc. La clave primaria

de esta tabla es el valor numérico **idUnidad**, y al igual que con la tabla **usuario**, se trata de un valor no nulo y autoincremental.

3. La tabla **unidadesrelacionadas** representa la relación muchos a muchos de la tabla **unidad**. Esta relación es la encargada de definir a que unidades puede evolucionar una unidad:

   -**idUnidadBase**: Corresponde con el identificador de la unidad que puede ser evolucionada.

   -**idUnidadRelacionada**: Corresponde con el identificador de la unidad a la que puede ser evolucionada la unidad base.

   A diferencia de las tablas anteriores, la clave primaria de esta tabla está compuesta por sus 2 únicos valores **idUnidadBase** e **idUnidadRelacionada** para así evitar duplicaciones y campos vacíos.

La información de las tablas **unidad** y **unidadesRelacionadas** ha sido prefijada en la base de datos, mientras que la información de la tabla **usuario** se irá incluyendo cuando un nuevo usuario quiera registrarse en el juego (a excepción de los campos auto-generados).

2.3. Funcionalidad multijugador

Como vimos antes, uno de los requisitos importantes que se plantéan en este proyecto, consiste en permitir jugar de forma remota contra otro jugador. En concreto el presente Trabajo Fin de Grado se centra en este aspecto. Recordemos que esta parte debe encargarse de las siguientes actividades:

- Identificación de usuarios.

- Registro de nuevos usuarios.

- Búsqueda de partida.

- Jugar partida de forma remota.

Para llevar a cabo dichas tareas se consideraron dos posibles alternativas:

- Crear un único servidor que gestione todas las necesidades del proyecto.

- Crear varios servidores encargados de gestionar tareas específicas, pero que en conjunto contemplen todas las necesidades del proyecto.

Como se puede observar en el diagrama de la Figura 2.2, con el uso de un único servidor para el abanico de posibles acciones permitidas para el usuario, todas convergerían en un único punto, lo que facilitaría la aparición del fenómeno del cuello de botella en el servidor para un número elevado de solicitudes simultaneas aunque no sean todas iguales. No obstante, con un

único servidor se podría evitar también la duplicación de código y funcionalidades que pueden aparecer en varios servidores.
Por otro lado, si se observa el diagrama de la Figura 2.3, con varios servidores solo convergerían en el mismo punto las solicitudes que necesiten acceder al mismo servidor, evitando así un cuello de botella que ralentice todo el servicio, pero pudiendo aparecer un cuello de botella para cada uno de los servidores en el caso de recibir grandes cantidades de solicitudes en un servidor, no obstante, esto no obstaculizaría el funcionamiento de los demás servidores.

Además, en caso de que ocurriera un error que pudiese bloquear o hacer funcionar de forma incorrecta el servidor, podría ser más difícil encontrar el error mientras que en varios servidores se podría observar más fácilmente que servidor ha fallado y localizar el error de forma más concreta.Así pues, como uno de los objetivos establecidos para este proyecto (aunque sea más secundario) es la distribución de la carga y reducir el efecto del cuello de botella para numerosas solicitudes, se ha optado por la opción de crear varios servidores específicos que engloben todas las funcionalidades necesarias para completar el proyecto. En concreto, se han planteado realizar los siguientes servidores:

- Servidor de Identificación: será el encargado de identificar a los usuarios que quieran acceder al juego, y almacenar la información de los nuevos usuarios en la base de datos.

- Servidor de Juego: su función principal será controlar las acciones de los jugadores y de notificar las mismas a los jugadores rivales (siempre y cuando dichas acciones estén permitidas).

- Servidor de Emparejamiento: será utilizado para emparejar a los jugadores para que se enfrenten en una partida.

- Servidor de Consulta de Unidades: será el encargado de enviar toda la información de unidades a los jugadores.

En el Capítulo 3 serán detallados cada uno de ellos.

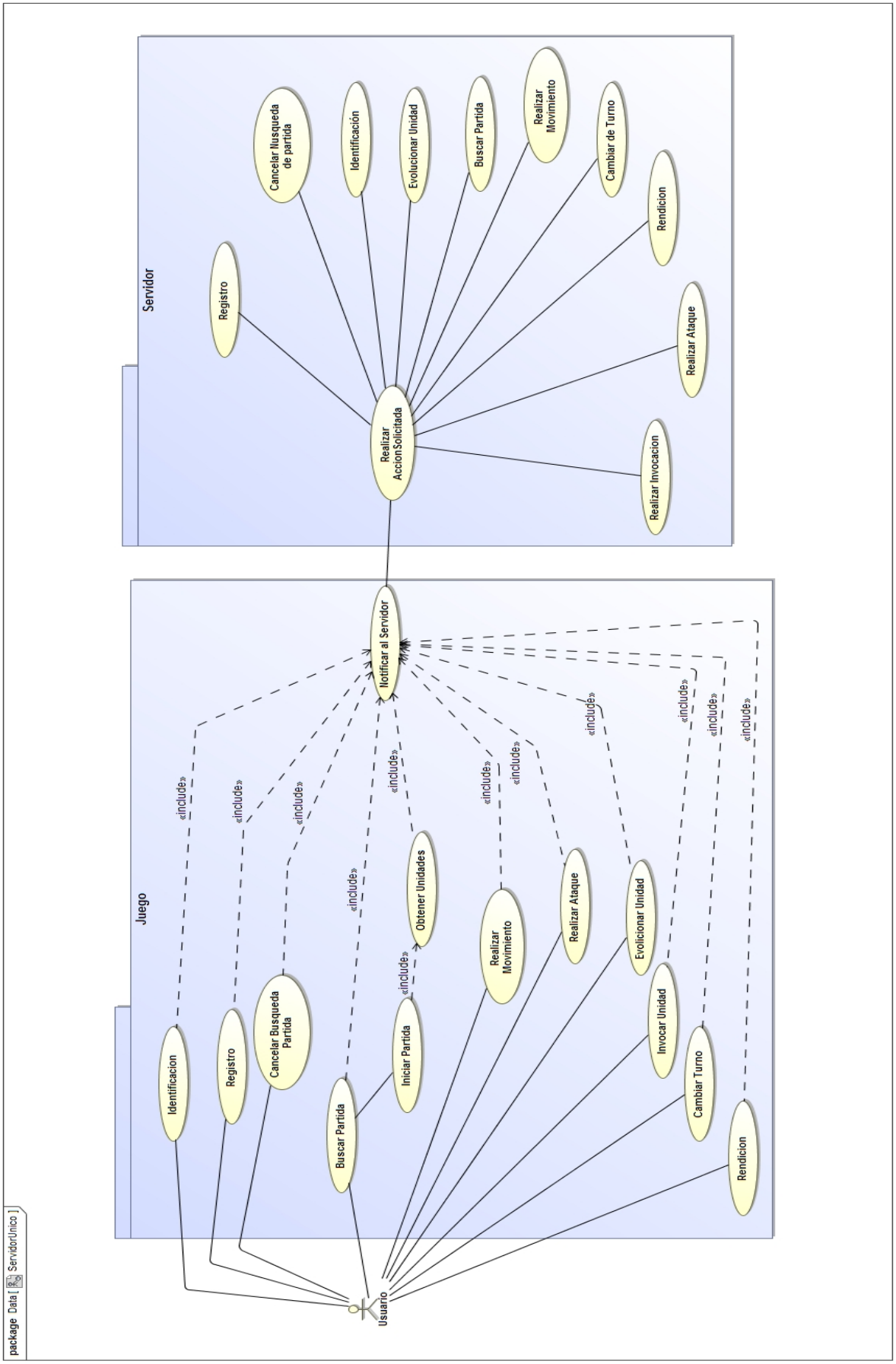


Figura 2.2: Caso de uso para un único Servidor

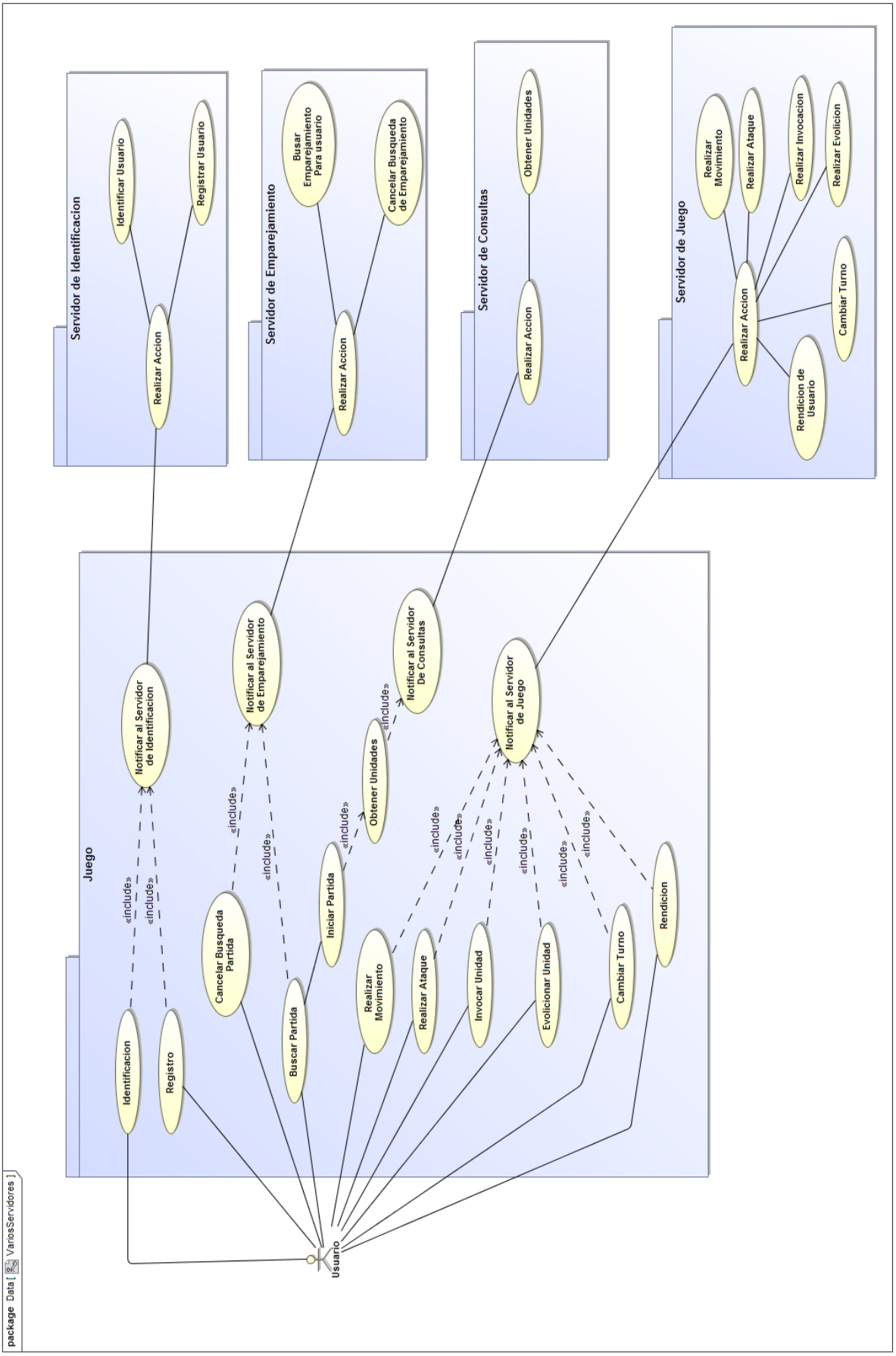


Figura 2.3: Caso de uso para varios Servidores

3

Implementación y pruebas

Como se mencionó en el capítulo anterior, en este se pretende explicar la funcionalidad de cada uno de los servidores así como la implantación realizada en Unity3D para permitir que varios jugadores jueguen entre sí en línea y el proceso de pruebas que se ha seguido.

Pero antes de comenzar a explicar cada servidor por separado, con el diagrama de secuencia de la Figura 3.1 se pretende explicar a alto nivel como será el proceso de interacción del usuario con los servidores y de los servidores entre si:

1. El usuario introduce los datos de identificación en el juego, el cual enviará un mensaje al **servidor de identificación**, quien verificará la solicitud y enviará la respuesta de vuelta al juego. Esto daría acceso al usuario al menú de juego.

2. El usuario pulsará el botón correspondiente para jugar una partida multijugador, esto implicará que el juego envíe un mensaje al **servidor de emparejamientos** y esperará la respuesta.

3. Una vez el **servidor de emparejamiento** ha encontrado un rival para el usuario, enviará un mensaje al **servidor de juego** para que cree la partida que jugarán ambos usuarios, y responderá con la información de la partida creada. Esta información, será a su vez remitida por el **servidor de emparejamientos** al juego de cada usuario.

4. Cuando el juego recibe la información necesaria de la partida en la que va a jugar, solicita al **servidor de consulta** la información de las distintas unidades y sus evoluciones. Tras finalizar este paso, el juego dará acceso al usuario a la partida.

5. Mientras no termine la partida, el jugador podrá realizar acciones, que serán notificadas al servidor para verificar que se trata de una acción disponible, tras lo cual, el juego realizará la acción y el usuario podrá visualizarla. El **servidor de juego** también enviará la acción al usuario rival para mantener el mismo estado de juego.

6. Una vez finalizado el juego, el usuario tendrá la opción de volver al menú de juego.

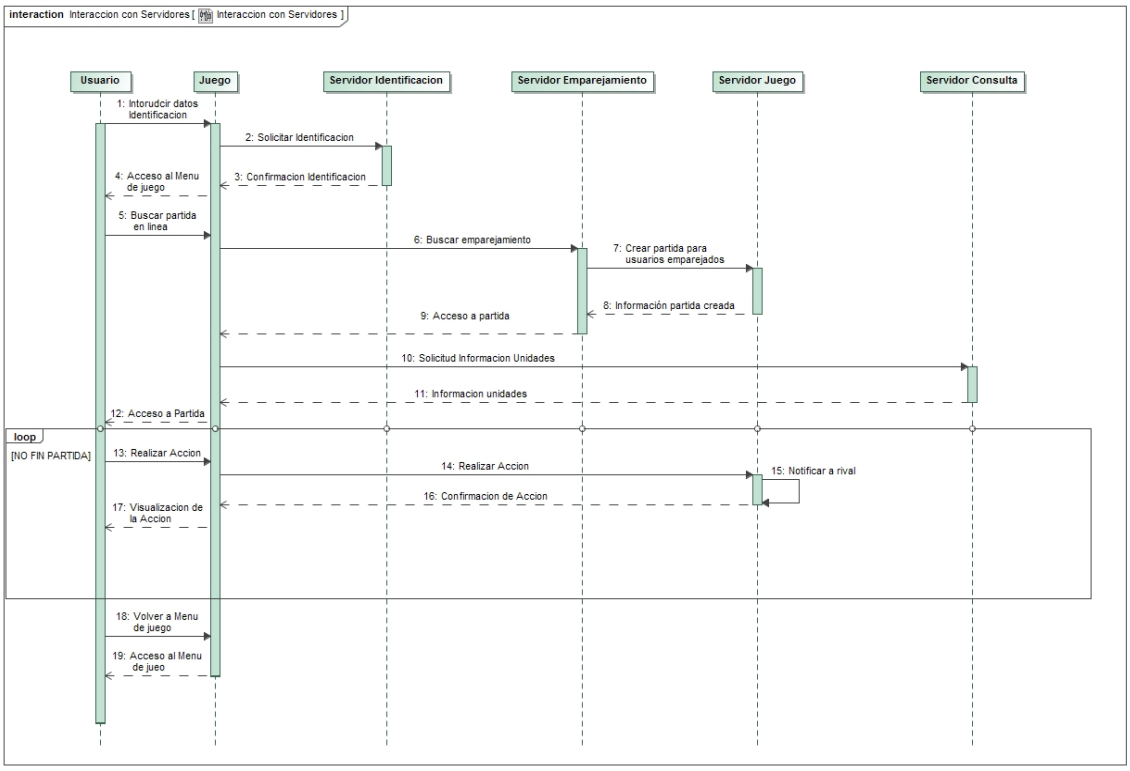


Figura 3.1: Diagrama de Secuencia de la Interacción con los Servidores

3.1. Servidor de identificación

Con este servidor se pretende controlar tanto la identificación de los usuarios como el registro de nuevos usuarios.

3.1.1. Descripción y funcionamiento

El Diagrama de clases de la Figura 3.2 es la base sobre la que se ha construido el servidor de identificación para este producto software. Como se puede observar, el servidor estará constituido por tres clases: **Servidor**, **DatosIdentificacion** y **ConexionBaseDatos**.
Las cuales se explican a continuación.

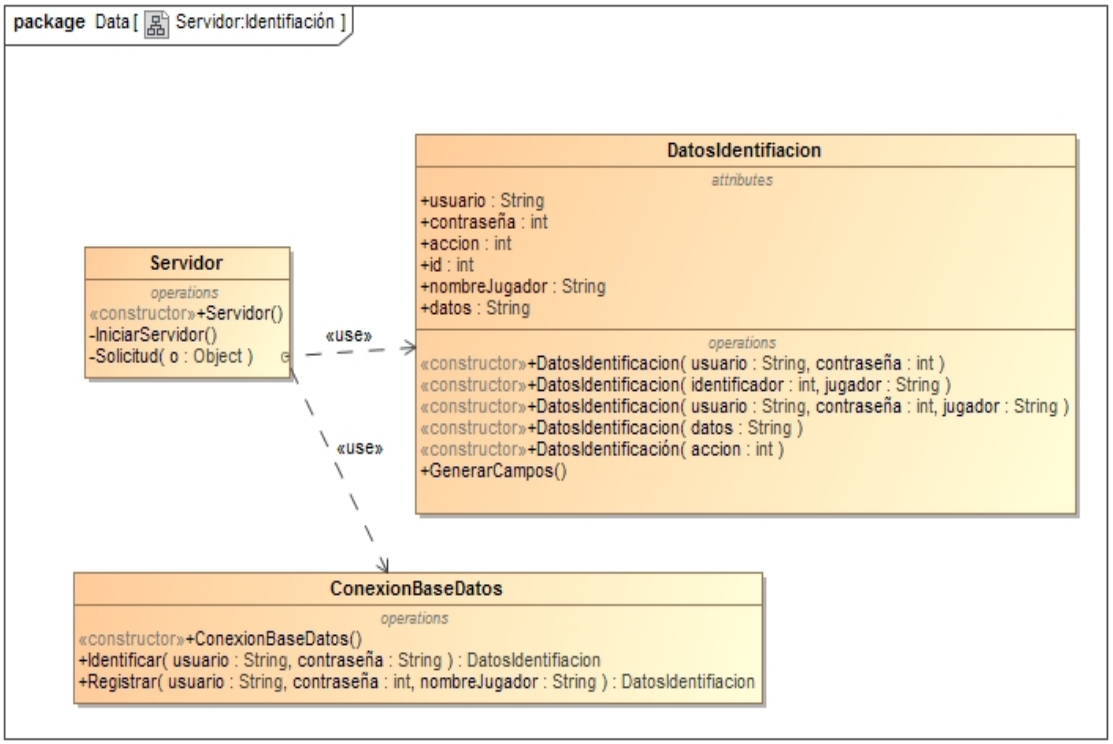


Figura 3.2: Diagrama de clases del Servidor de Identificación

Clase: Servidor

La clase **Servidor** será el núcleo de este servidor, pues será la encargada de gestionar todas las solicitudes de los usuarios e identificar si se trata de una solicitud de identificación o de registro. Además, esta clase se ha construido como un proceso multihebra para que pueda procesar las solicitudes de varios usuarios simultáneamente, como se puede observar en fragmento de código 3.1, correspondiente a la inizialización del servidor, el cual es llamado directamente desde el constructor.

Tanto en este servidor como en los demás, se utilizará la API de Sockets de C# para el envío de mensajes sobre TCP/IP. En concreto el servicio utiliza TCP como capa de transporte, para asegurar una transmisión fiable de los datos en el puerto correspondiente (configurable para cada servidor). A nivel de aplicación (mensajes, formato,...) se utiliza un protocolo desarrollado específicamente para este trabajo y será explicado en las siguientes secciones con forme se vaya necesitando.

```
private void IniciarServidor()
{
  Console.WriteLine("Servidor Escuchando!");
  while (true)
  {
    Socket socket = listener.AcceptSocket();
    Thread nuevaSolicitud = new Thread(new
       ParameterizedThreadStart(solicitud));
    nuevaSolicitud.Start(socket);
  }
}
```

Código 3.1: Inicio del Servidor de Identifiación

Clase: DatosIdentifiacion

La clase **DatosIdentifiacion** definirá el protocolo de los mensajes que serán enviados entre el usuario y el servidor y posee diferentes constructores para facilitar la creación y usos de las diferentes connotaciones de los mensajes. Estos mensajes serán textos que contenga los diferentes campos separados por un delimitador, en este caso ';'. No obstante el contenido de los mensajes, tanto el de este protocolo, como los protocolos de los demás servidores, serán transformados a bytes usando la encriptación ASCII para ser posteriormente enviados en este formato mediante los Sockets.
Este protocolo siempre tendrá el campo **acción**, los demás campos como usuario, contraseña, nombreJugador e id, serán incluidos en función a esta acción:

- Acción = *IDENTIFICAR*: Esta acción indica al servidor que un usuario quiere identificarse por lo que el mensaje irá acompañado de los valores correspondientes a usuario y contraseña.

-Acción = *CORRECTO*: Esta acción es la respuesta recibida ante una solicitud de identificación o registro e ira acompañada de los valores nombreJugador y su identificador.

-Acción = *REGISTRO*: Esta acción indica al servidor que un nuevo usuario quiere registrarse, por lo que vendrá acompañada con los valores usuario, contraseña y nombreJugador.

-Acción = *ERRORJUGADOR*: Esta acción es una respuesta del servidor ante una solicitud de registro de usuario, el cual sirve para indicar que el *nombre de jugador* indicado para el registro ya se encuentra en uso.

-Acción = *ERRORUSUARIO*: Esta acción es una respuesta del servidor ante una solicitud de registro de usuario, el cual sirve para indicar que el *nombre de usuario* indicado para el registro ya se encuentra en uso.

En caso de se produzca un error al solicitar al servidor la acción *IDENTIFICAR* el juego esperará cualquier acción distinta de *CORRECTO* para mostrar el mensaje 'Usuario o contraseña incorrectos'.

Clase: ConexionBaseDatos

Como su nombre indica, la clase **ConexionBaseDatos** se encarga de gestionar las solicitudes a la base de datos, para almacenar los datos de un nuevo usuario, o ver si coinciden los datos introducidos con alguno de los contenidos en la base de datos al realizar las identificaciones.

Funcionamiento conjunto

Estas tres clases funcionando en conjunto forman el servidor de identificación, cuyo funcionamiento se puede observar en el diagrama de la Figura 3.3.

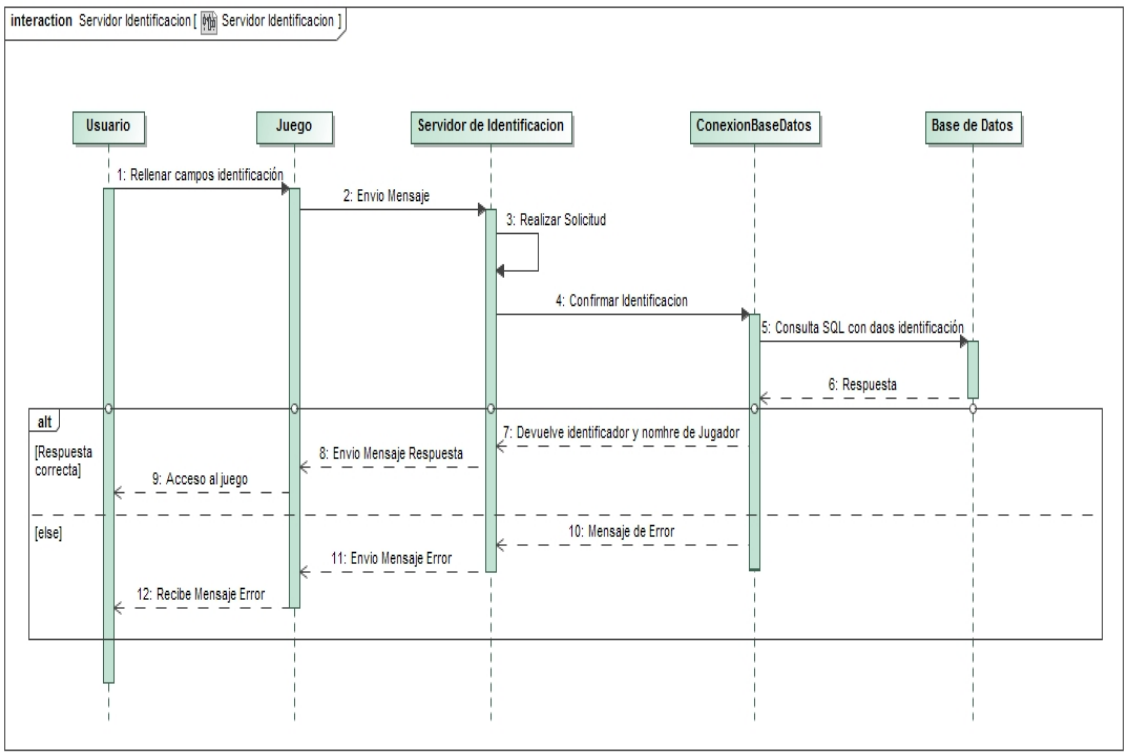


Figura 3.3: Diagrama de secuencia para el proceso de Identificación de un usuario

La Figura 3.3 consiste en un diagrama de secuencia en el que se puede observar todo el proceso que se ejecuta para que un usuario pueda ser identificado y acceder al juego, donde se puede observar tanto el comportamiento esperado (logra identificarse de forma apropiada) como el erróneo.

En el caso de registrarse como nuevo usuario, el proceso es prácticamente similar, con la diferencia de que el usuario no tendría acceso al juego en caso de recibir una respuesta

afirmativa del servidor, sino que tendría que identificarse con los datos que acaba de introducir para que el servidor ya le de acceso al juego.

3.2. Servidor de juego

El servidor de juego será el núcleo de este proyecto pues sobre él recaerá la mayor carga de trabajo y mensajes. El diagrama de la Figura 3.4 pretende mostrar la estructura básica sobre la que se ha construido este servidor.

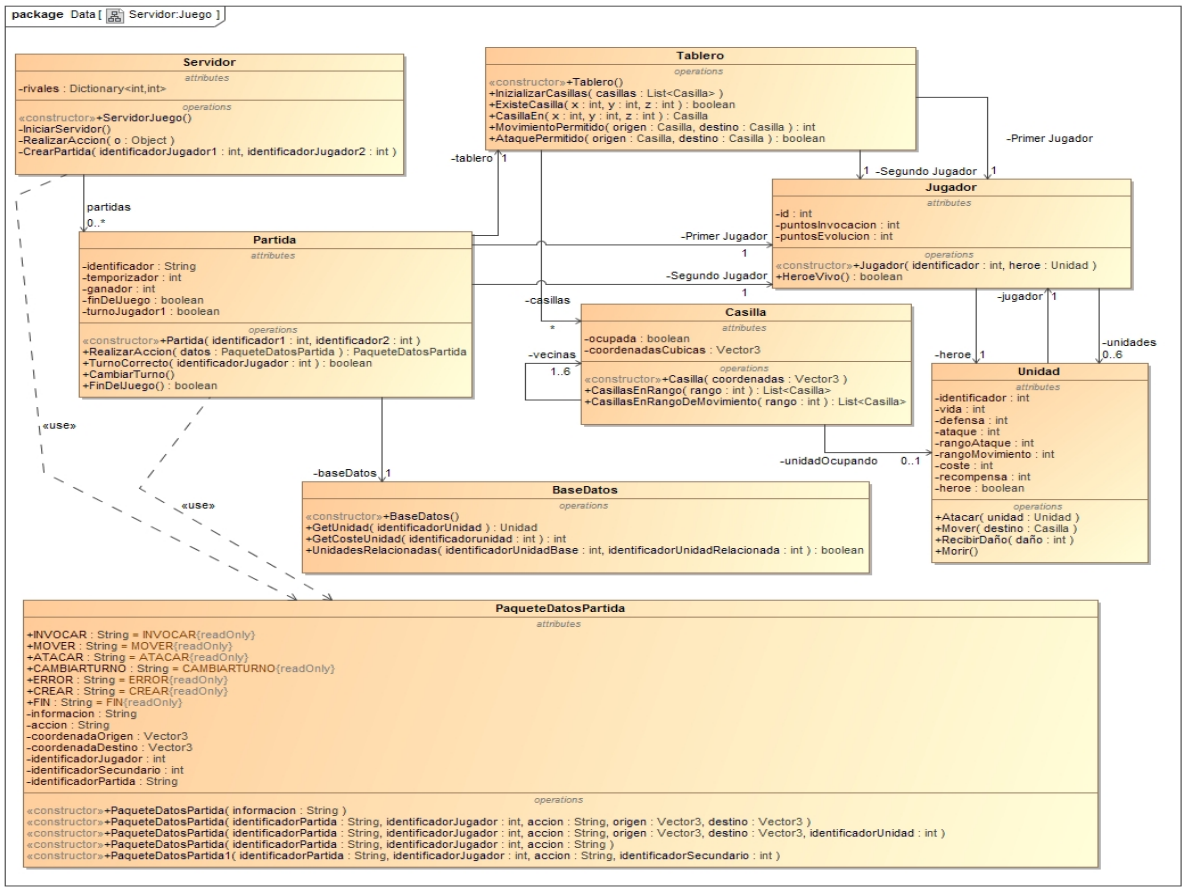


Figura 3.4: Diagrama de clases del Servidor de Juego

3.2.1. Definición y funcionamiento

A diferencia de los otros servidores, en esta sección se explicará el funcionamiento del servidor de juego en lugar de explicar la funcionalidad de cada clase, a excepción de la clase PaqueteDatosPartida que será explicada a continuación.

Clase: PaqueteDatosPartida

Esta clase define el protocolo para los mensajes que serán enviados al servidor de juego desde el servidor de emparejamiento y desde el juego de los usuarios.
Estos mensajes siempre tendrán el campo *acción*, *identificador de jugador*, e *identificador de partida*, mientras que el resto de campos serán incorporados al mensaje en función del tipo de acción a realizar, separados por ';'. También cabe mencionar que la utilidad delos campos identificador de jugador e identificador secundario. El identificador de jugador siempre hará referencia al jugador que está realizando la acción, mientras que el identificador secundario indicará, cuando sea necesario, el identificador de unidades que quieren ser invocadas o evolucionadas, así como para solicitar otra información según proceda que sea identificada por un valor numérico.

Funcionamiento

Una vez conocemos como están estructurados los mensajes que recibirá y enviará este servidor, podemos proceder a explicar su funcionamiento de forma secuencial, es decir comenzando por la creación de una partida y continuando hasta la finalización de la misma.

Creación de partida: El servidor de juego creará una partida cuando el servidor de emparejamiento le envié un mensaje con los identificadores de los jugadores que se enfrentarán. A lo que el servidor responderá con la misma información además del identificador de partida que ha creado.
Cuando se crea una partida, el servidor de juego además accede a la base de datos para posicionar las unidades heroicas de cada jugador sobre el tablero en la parte central superior o inferior según se haya decidido quien tendrá el primer turno, el cual ha sido decidido de forma aleatoria.

Iniciar partida: Cuando los el usuario del juego haya recibido la información necesaria, enviará un mensaje al servidor de juego con la acción INICIO, solicitando la posición de su héroe (el campo identificador secundario tendrá el mismo valor que el campo identificador de jugador) y después otro mensaje solicitando la posición del héroe rival (el campo identificador secundario tendrá el valor 0). A lo que el servidor responderá con mensajes con la acción *INVOCARHEROE* con el identificador del héroe y su posición. Además, cuando el servidor de juego ha detectado que el un usuario ya ha enviado ambos mensajes, sabrá que el jugador ya se encuentra preparado para comenzar la partida. Cuando ambos usuarios están preparados para iniciar la partida, el servidor enviará un mensaje de *CAMBIATURNO* a los jugadores (debido

a la implementación realizada en en Unity por el compañero para el inicio de partidas), lo cual les mostrará el tablero de juego con las dos unidades principales ya posicionadas.

Realizando acciones: Una vez los usuarios tienen acceso a la partida, pueden comenzar a realizar acciones de movimiento, ataque, invocación, evolución. Estas acciones enviadas por el jugador contienen además las coordenadas de origen y las coordenadas de destino (excepto evolución). En el caso de invocación y evolución se debe enviar el identificador de la unidad a posicionar sobre el tablero. Cuando el servidor recibe una de estas acciones, se encarga de comprobar que dicho usuario puede realizarla y en caso de que el resultado sea afirmativo, notificará al rival de que el usuario ha realizado esa acción, para que su juego se pueda actualizar en consecuencia. Además el propio servidor de juego actualiza la partida para que las siguiente acciones puedan ser comprobadas correctamente.
Además de dichas acciones el usuario también puede solicitar un cambio de turno o la rendición. El cambio de turno conlleva también una comprobación de acción, para evitar que se pida el cambio de turno en el turno del rival. Por el contrario rendición finaliza directamente la partida. Ambas acciones son notificadas a también a los rivales, excepto si el cambio de turno solicitado no es correcto.

Finalización de partida: Cuando una partida ha terminado por que un usuario ha derrotado al héroe rival o se ha rendido. La partida de dichos jugadores se actualizará para no aceptar más solicitudes, y se enviará un mensaje de *FIN* a ambos usuarios con el resultado de la partida. Además, se eliminará del servidor toda la información.

Otra información de interés

Además de lo ya descrito anteriormente, cabe destacar que cada partida posee un hilo de ejecución paralelo que se encarga de realizar cambios de turno si los usuarios no han realizado ninguna acción en el tiempo establecido. Esta funcionalidad está además controlada por un semáforo para evitar problemas si un usuario solicita el cambio de turno justo cuando el servidor realiza el cambio de turno automático. Por otro lado, este cambio de turno se realiza antes de enviar el mensaje a los usuarios, por lo que cualquier acción que llegase mientras se está enviando el mensaje sería rechazada por el servidor.

Por otro lado, las clases Tablero, Unidad y Casilla son las mismas que se encuentran en el juego en Unity (realizadas por el compañero), realizando algunas modificaciones necesarias para adaptarlas al funcionamiento del servidor de juego.

3.3. Servidor de emparejamientos

El servidor de emparejamientos será el encargado de establecer contra quien se enfrentará un jugador que quiera jugar una partida en linea.

3.3.1. Descripción y funcionamiento

El diagrama de la Figura 3.5 es el diagrama sobre el que se ha basado la creación del servidor de emparejamiento.

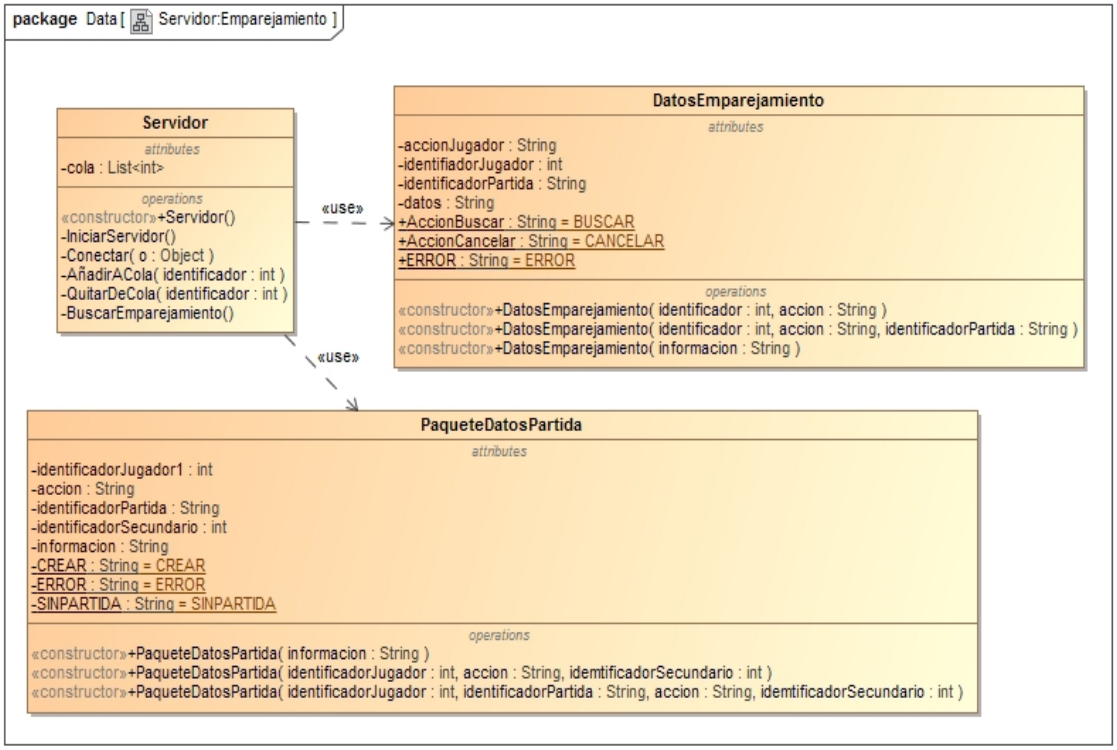


Figura 3.5: Diagrama de clases del Servidor de Emparejamiento

Como se puede observar en dicho diagrama, este servidor también se encuentra constituido por tres clases como el servidor de identificación. No obstante, dos de estas son definiciones de protocolos para los envíos de mensajes, el primero será para la comunicación entre los usuarios y el servidor de emparejamientos, y el segundo para la comunicación entre el servidor de emparejamientos y el servidor de juego.

Clase: Servidor

En esta clase, recae casi toda la funcionalidad del servidor de emparejamiento, que se encarga de gestionar todas las solicitudes de los usuarios y emparejarlos para que puedan jugar entre sí. Además, esta clase se ha construido como un proceso multihebra para que pueda procesar las solicitudes de varios usuarios simultáneamente además de gestionar los emparejamientos sin interrupciones por los envíos de mensajes.
Esto quiere decir que la función *BuscarEmparejamiento* es llamada en la inicialización del servidor como un hilo de ejecución paralelo.
La lista de enteros *cola* que posee esta clase, será la encargada de almacenar a todos los usuarios que quieren buscar partida y están esperando.

Clase: DatosEmparejamiento

La clase **DatosIdentifiacion** definirá el protocolo de los mensajes que serán enviados entre el usuario y el servidor de emparejamiento y posee diferentes constructores para facilitar la creación y usos de las diferentes connotaciones de los mensajes.
Este protocolo siempre tendrá el campo **acciónJugador** e **identificadorJugador**. El campo **identificadorPartida** se completará unicamente en los mensajes en los que el servidor de emparejamiento quiera notificar al jugador que ya puede empezar a jugar una partida, en cuyo caso, contendrá el identificador de partida que será utilizado por el juego del usuario para comunicarse con el servidor de juego.

Para comunicarse con este servidor, el usuario puede solicitar las acciones de *BUSCAR* o *CANCELAR*, con las cuales querrá buscar un rival con el que jugar una partida, o cancelar la busqueda de rivales. La acción *ERROR* será utilizada unicamente por el servidor para notificar al usuario de que se ha producido algún error.

Clase: PaqueteDatosPartida

Esta clase es una versión reducida de la clase **PaqueteDatosPartida** del *servidor de juego* (la cual se describe en la sección 3.2.1), que será utilizado para notificar al servidor de juego que se han encontrado a dos usuarios para que jueguen una partida entre ellos y debe crear dicha partida.
En este caso,los campos **identificadorJugador** e **identificadorSecundario** corresponde con los identificadores de los jugadores a los que el servidor ha emparejado para jugar una partida, por lo que la acción a enviarle al servidor será *CREAR*, acción que se recibirá como respuesta si se ha creado la partida correctamente, y se le asignará al campo **identificadorPartida** el valor *SINPARTIDA*. La acción *ERROR* será utilizada por el servidor de juego para notificar al servidor de emparejamiento que se ha producido un error al crear la partida.

Funcionamiento

En el diagrama de secuencia de la Figura 3.6, se puede observar el proceso que se sigue cuando dos jugadores quieren jugar una partida y el servidor de emparejamiento los empareja.

Si la respuesta recibida por el servidor de juego es un error, los jugadores siguen esperando una respuesta por lo que no se darían los pasos del 8 al 11 y el juego los volverá a emparejar.

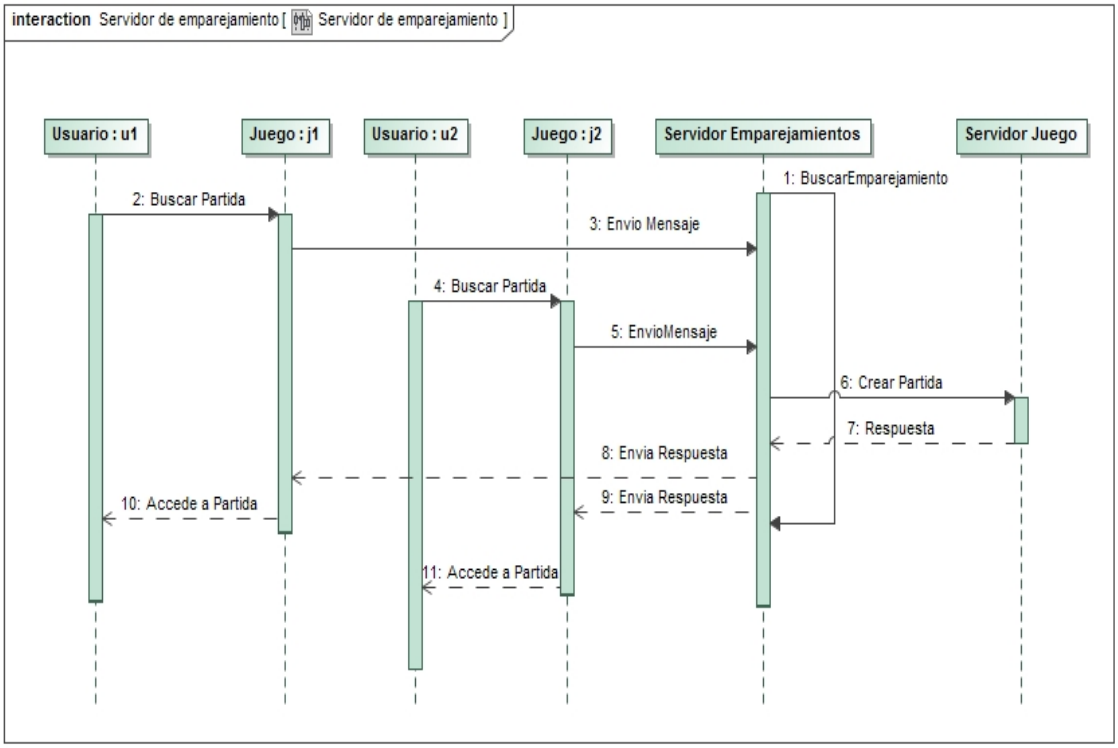


Figura 3.6: Diagrama de secuencia para el proceso de buscar partida

BuscarEmparejamiento

La mayor complejidad de este servidor recae sobre esta función, pues es la encargada de emparejar a los jugadores evitando que el servidor se quede bloqueado y los posibles errores a la hora de realizar los emparejamientos.

Esta función se ejecutará mientras el servidor esté activo, por lo que tendrá un bucle que englobe toda su funcionalidad. Además, esta función solo realiza las funciones de emparejamiento si la *cola* tiene al menos a dos usuarios buscando partida, en caso contrario la función esperará durante un segundo y volverá a realizar la comprobación.

Si posee a al menos dos usuarios buscando partida, la función obtendrá los identificadores de esos dos usuarios así como sus sockets de conexión y comprobará si siguen conectados. De ser así, la función generará un mensaje para el servidor de juego para que cree la partida para dichos usuarios.

Cuando ha recibido la respuesta del servidor con el identificador de partida correspondiente, envía un mensaje a cada usuario con el identificador de partida a la que han sido asignados.

Además si se ha producido algún error durante este proceso, se intentará re-colocar a ambos usuarios al principio de la cola mediante la función *RecolocarPrimero*, la cual se puede observar en el Código 3.2

```
private void RecolocarPrimero(Socket s, int idJugador)
{
  if(SigueConectado(s))
  {
    cola.Insert(0, idJugador);
  }
  else
  {
    conexiones.Remove(idJugador);
  }
}
```

Código 3.2: Funcion: RecolocarPrimero del servidor de emparejamiento

La función *SigueConectado*, es la encargada de comprobar si un usuario sigue esperando una respuesta del servidor. El funcionamiento de dicha función se puede observar en el Código 3.3

```
private bool SigueConectado(Socket s)
{
  return (!s.Poll(1000, SelectMode.SelectRead) || !(s.Available
     == 0));
}
```

Código 3.3: Funcion: SigueConectado del servidor de emparejamiento

3.4. Servidor de consultas de unidades

Este servidor ha sido creado para quitar que los usuarios obtengan la información de las unidades del juego sin añadir carga a los demás servidores.

3.4.1. Definición y funcionamiento

La Figura 3.7 representa el diagrama a partir del cual se ha construido el servidor de consulta de unidades.

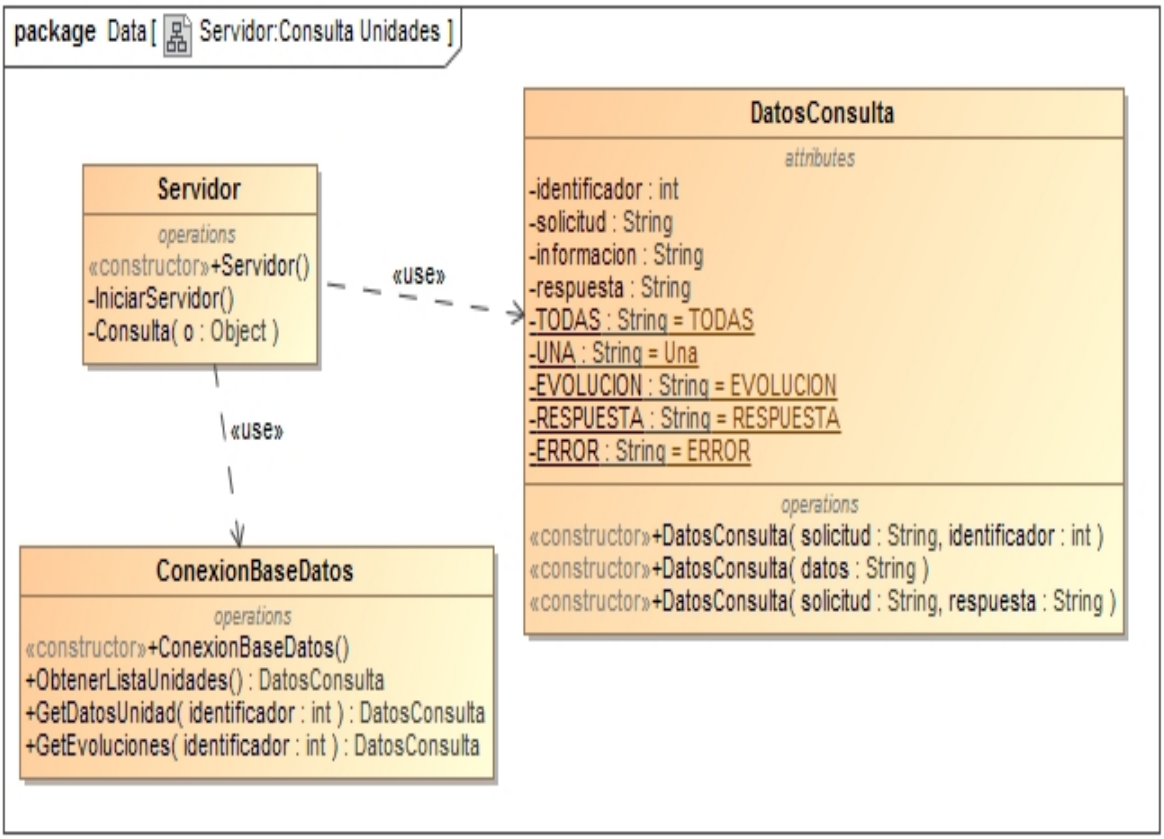


Figura 3.7: Diagrama de clases del Servidor de Consulta de Unidades

Como ocurre con el servidor de identificación, está constituido por tres clases: el servidor, un protocolo para los mensajes y la conexión con la base de datos, y su funcionamiento es similar:

1. El servidor recibe una solicitud desde el juego para obtener información sobre una o varias unidades, o cuales son las evoluciones de cierta unidad.

2. Se accede a la base de datos para obtener la información solicitada por el usuario.

3. El servidor envía un mensaje al usuario con la información que ha solicitado o un mensaje de ERROR si se ha producido alguno.

El único comportamiento que cabe destacar de este servidor respecto al de identificación, es la forma en la que se envían las respuestas. Como no se sabe a ciencia cierta como de grande puede ser el mensaje, ya que depende del número de unidades y de sus características, se procede a realizar la respuesta con dos mensajes, el primero indicando el tamaño del siguiente mensaje, para que el juego no falle al esperar un mensaje de menor tamaño, y en el segundo mensaje se envía la información solicitada.

No obstante, la funcionalidad de obtener la información de una única unidad no es usada actualmente por el proyecto. Aun así, se ha implementado pensando en posibles ampliaciones posteriores.

3.5. Implementación en Unity

Tras la realización de los distintos servidores, se ha procedido a realizar en Unity la creación del código necesario para permitir que un usuario pueda acceder al juego y jugar partidas. En este caso, se procederá a explicar la implementación del código según las escenas de Unity en las que han sido utilizadas: Identificación, Menú de Juego, Tablero de Juego.

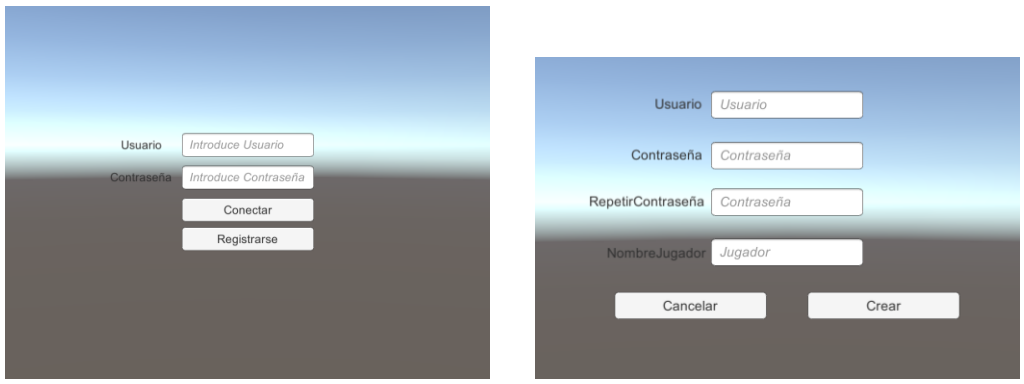
3.5.1. Identificación

En la escena de identificación se han creador dos "formularios" formados por campos de texto y botones:

- Formulario de identificación: El formulario de identificación consiste en dos campos de texto, el primero para el nombre de usuario, y el segundo para la contraseña. Además consta de dos botones, uno para proceder a la identificación y otro para acceder al formulario de registro.

- Formulario de registro: El formulario de registro consta de cuatro campos de texto, los cuales son para el nombre de usuario, un campo para la contraseña y otro para su verificación, y por último para el nombre de jugador. Además también tiene dos botones, uno para enviar el formulario y otro para cancelar y volver al formulario de identificación.

En la Figura 3.8 se pueden observar como han sido creados visualmente los formularios de identificación (Figura 3.8a) y de registro (Figura 3.8b).



(a) Formulario de Identificación (b) Formulario de Registro

Figura 3.8: Formularios

Para gestionar, controlar y enviar mensajes al servidor de identificación se han utilizado dos clases: **DatosIdentificacion** y **ControladorIdentifiación**. La clase **DatosIdentificación** es el protocolo para los menajes de identificación explicado en la Sección 3.1.1 y la otra clase se explica a continuación.

ControladorIdentifiación

Esta clase se encarga de interactuar con los elementos gráficos de Unity para enviar los mensajes al servidor e informar al usuario de las respectivas respuestas y/o errores. Dichas funciones son:

- *Registro*: Se encarga de gestionar el envío de mensajes de registro al servidor de identificación e informar al usuario de que el registro se ha llevado a cabo con éxito, el nombre de usuario introducido ya se encuentra en uso o que el nombre de jugador ya se encuentra en uso.

- *Conectar*: Se encarga de gestionar el envío de mensajes de identificación al servidor de identificación, así como de mostrar un mensaje de error en un caso de identificación erróneo, o en caso afirmativo cambiar la escena a la correspondiente del menú de juego.

- *EnviarMensaje*: Se encarga de enviar los mensajes al servidor y recibir las respuestas correspondientes. Esta función es llamada desde las funciones *Registro* y *Conectar* que le pasan a su vez String con el mensaje que se quiere enviar, y devuelve un **DatosIdentificacion** como respuesta.

3.5.2. Menú de juego

Cuando aparece el menú de juego, lo único que aparece es el botón Jugar, como se muestra en la Figura 3.9a, el cual, tras pulsarlo, nos muestra las diferentes opciones de juego que encontramos, como muesta la Figura 3.9b.

(a) Menú de juego: Jugar

(b) Menú de juego: Botones

Figura 3.9: Menú de juego

En esta sección nos vamos a centrar únicamente en el botón "Multi jugador". Al pulsar dicho botón, el juego enviará un mensaje del tipo **DatosEmparejamiento** (explicado en la Sección 3.3.1) al servidor de emparejamiento y esperará a que el servidor responda al mensaje con la información necesaria para iniciar la partida. Mientras se espera para iniciar la partida, el juego ha sido preparado para no quedarse bloqueado y ofrecer la opción de cancelar la búsqueda.

Una vez recibida la respuesta desde el servidor de emparejamiento, el juego procederá a cambiar a la escena Tablero de Juego, que se procede a explicar a continuación.

3.5.3. Tablero de juego

Antes de aparecer esta escena, Unity comienza a cargar toda la información necesaria y a generar los elementos del tablero. Además, en este proceso, se enviarán mensajes al servidor de juego para obtener la ubicación del héroe del usuario y del rival, además de comenzar a obtener información del servidor de consulta de unidades. Esto evita tener que enviar solicitudes de información sobre las unidades cada vez que se quiera consultar las características de cualquier unidad cuando se quiera invocar o evolucionar. Además, durante este proceso, el juego mantendrá una linea de comunicación abierta con el servidor de juego, con la cual estará siempre a la espera de recibir mensajes del servidor. Dicha información serán siempre accione realizadas por el rival, ha movido una unidad, a atacado, etc. El motivo de implementarlo de esta forma es añadir otro pequeño nivel de seguridad al juego, es decir, todas las acciones no realizadas por el usuario, excepto las respuestas a las acciones de cambio de turno y rendición, provendrán de dicho canal comunicado con el servidor de juego. Además, así se evita que el juego permanezca bloqueado mientras se esperan mensajes del servidor, pues se realiza en un hilo de ejecución paralelo.

Todo esto, se realiza desde la clase **ModoOnline**, que hereda de la clase **ModoDeJuego**, para que la funcionalidad del juego siga siendo la misma que desde cualquier otro modo de juego y sin necesidad de estar comprobando continuamente si se está jugando en linea o en cualquier otro modo de juego.
La diferencia principal de este modo de juego con el resto de modos es que cuando se quiere realizar una acción de movimiento, ataque, invocación o evolución, se envía un mensaje al servidor de juego y se espera su respuesta. Si la respuesta es correcta, se llama a la función de la clase padre **ModoDeJuego** correspondiente, la cual permitirá visualizar la acción. Si es incorrecta, no se realiza ninguna acción y el jugador puede intentar realizar otra acción.
Como se ha mencionado anteriormente, las acciones "cambio de turno" y "rendición" no esperan un mensaje de respuesta por el mismo canal de comunicación. Dicho mensaje será recogido por el canal que se mantiene abierto con el servidor de juego.

La Figura 3.10 muestra la escena Tablero de Juego tras cargar la información necesaria y una vez se ha iniciado la partida.

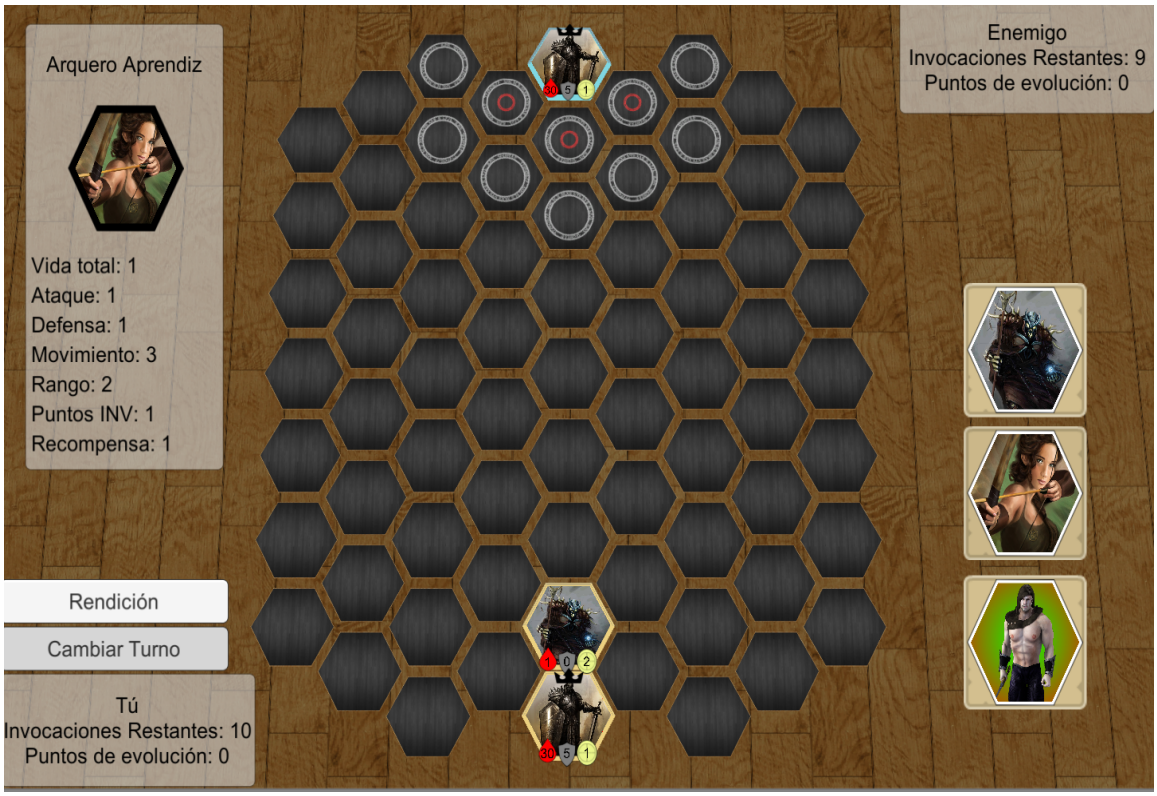


Figura 3.10: Escena tablero de Juego

3.6. Pruebas

Tras completar todo el código del proyecto, se han ido realizando pruebas manuales para comprobar el correcto funcionamiento del mismo con el siguiente procedimiento. Se han realizado este tipo de pruebas debido a que era necesario comprobar el comportamiento de Unity3d ante la comunicación con los servidores.

1. Probar el funcionamiento del servidor de identificación ejecutando el servidor y el juego en la misma máquina.

2. Probar el funcionamiento del servidor de identificación ejecutando el servidor y el juego en distintas máquinas dentro de la misma red local.

3. Probar el correcto funcionamiento del servidor de emparejamiento con dos usuarios distintos en equipos diferentes, y por consiguiente, probar el correcto funcionamiento de creación de partida y del servidor de consulta de unidades una vez realizado el emparejamiento.

4. Probar el correcto funcionamiento del servidor de juego al realizar las acciones de *Movimiento*, *Ataque*, *Invocación*, *Evolución*, *Cambio de Turno* y *Rendición*, así como del cambio de turno automático.

Además de el procedimiento de pruebas mencionado, se ha dado el juego a diferentes personas ajenas al proyecto para poder encontrar errores que se hayan pasado por alto al realizar las pruebas por los propios desarrolladores. Estas últimas pruebas han sido realizadas desde equipos remotos a la red local utilizando como intermediario el programa **LogMeIn Hamachi** que se encarga de simular una red local entre equipos de diferentes ubicaciones. Lo cual ha permitido a su vez probar la velocidad de respuesta de los servidores desde distintos puntos geográficos, aunque hayan podido, o no, tener un retraso extra por el intermediario **Hamachi**, entre ellos Galicia, Barcelona, Burgos, Tenerife y Málaga.

Durante el proceso de pruebas se han encontrado funcionalidades obviadas o no limitadas en el desarrollo del proyecto, así como varios errores (especialmente creando partidas para jugadores que ya se encuentran jugando una partida), haciendo posible su corrección.

4

Ampliaciones

Además de todas las funcionalidades mencionadas en los capítulos anteriores, se han desarrollado otras que no se habían contemplado inicialmente en anteproyecto pero que permiten mejorar la consecución de los objetivos y/o otorgan una mayor facilidad de visualización y gestión de los servidores y/o su información.

4.1. Interfaz gráfica para el servidor de juego

Como el servidor de juego es el servidor principal de este proyecto, se ha decidido crear una interfaz de usuario que permita visualizar información del mismo.

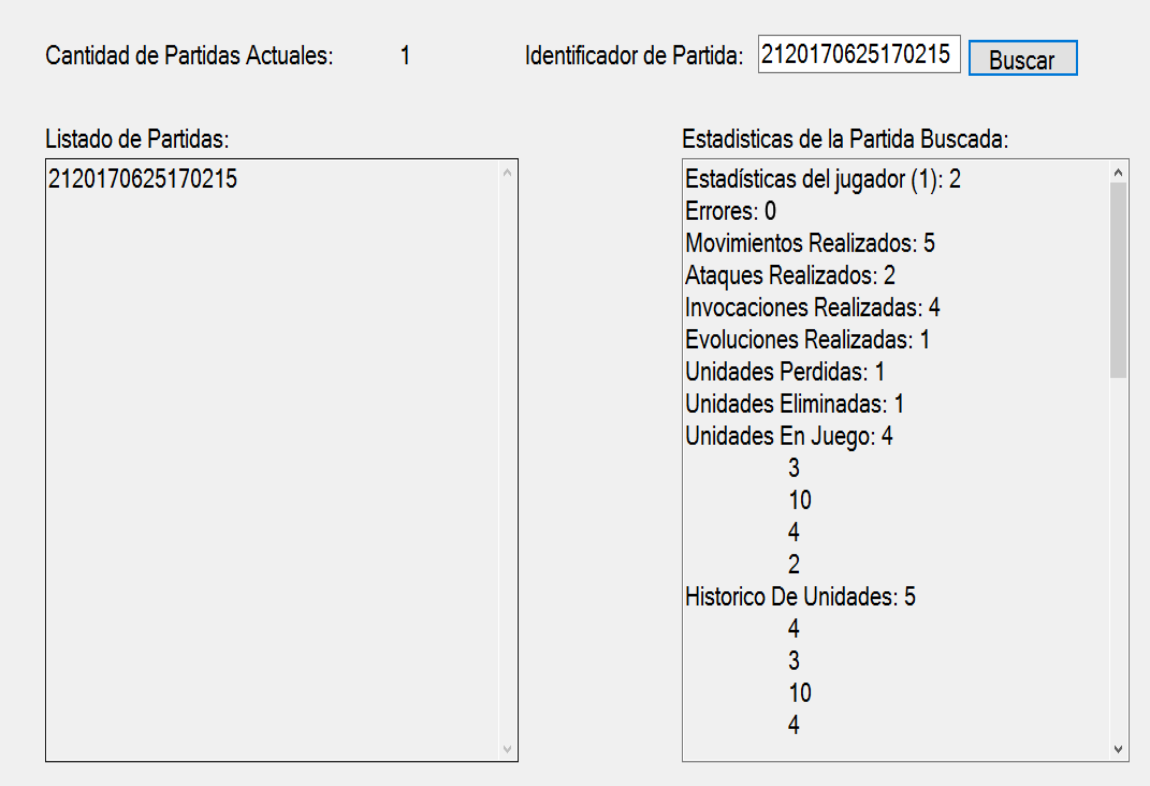


Figura 4.1: Interfaz Gráfica del Servidor de Juego

La Figura 4.1 representa el resultado final de esta interfaz gráfica y como se puede observar, ofrece información visual del número de partidas, el identificador de partidas y estadísticas una partida que está en juego actualmente y ha sido introducida en el buscador superior.

4.2. Registro de actividad

Para cada uno de los servidores se ha creado un registro de actividad que permite almacenar todos los mensajes y errores, recibidos y producidos respectivamente, en los servidores de forma automática. Es decir se ha creado la clase **EscritorDeLogs** que permite escribir en un fichero toda la información que se le indique sin entorpecer el funcionamiento de los servidores, ya que es un hilo de ejecución paralelo.
Esta clase es idéntica para todos los servidores, a excepción de la asignación del nombre de fichero que va a utilizar para escribir, en el cual se ha asignado el nombre del servidor seguido de la fecha en la que se ha recibido la información a guardar.
Un ejemplo de la información que almacena esta clase se puede ver en la Figura 4.2, el cual muestra un error que fue corregido durante la fase de pruebas del proyecto.

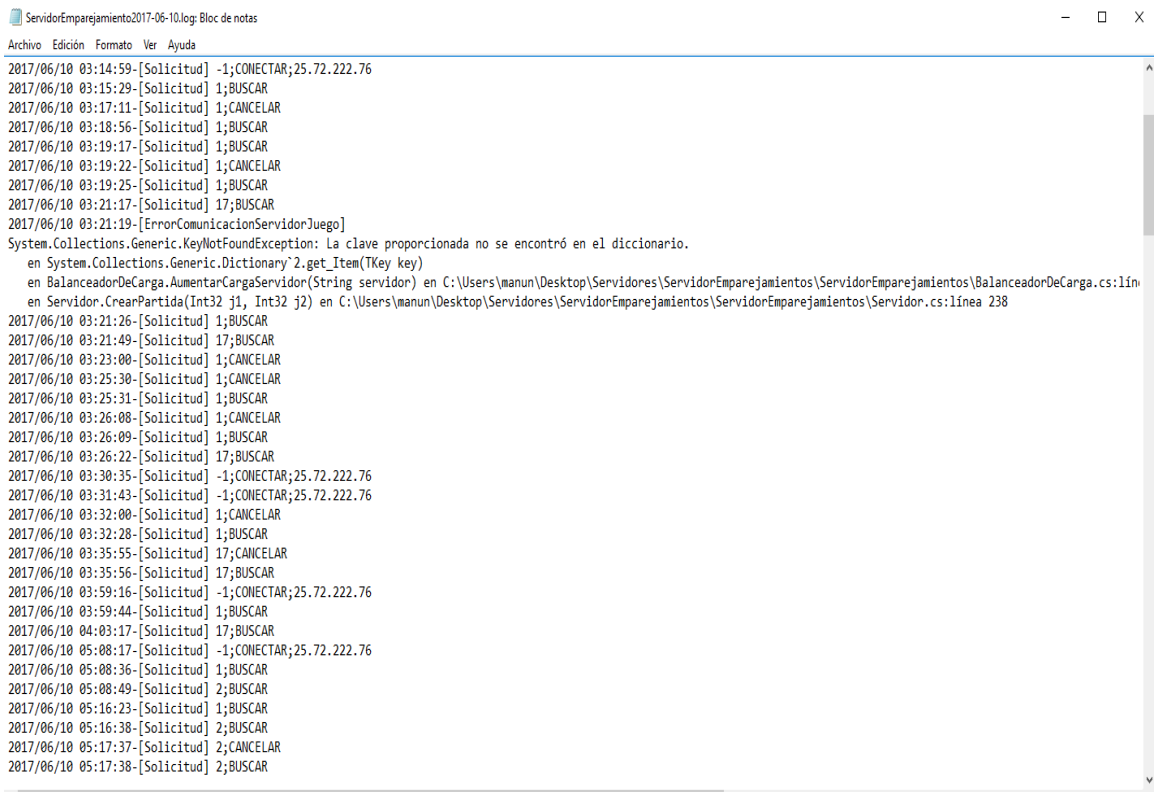


Figura 4.2: Ejemplo de Información de Salida del Escritor de Logs

4.3. Balanceador de carga

Como uno de los objetivos secundarios de este proyecto es evitar el cuello de botella para los servidores, y el servidor de juego es el servidor con más posibilidades de tener cuello de botella, pues recibirá un numero de mensajes mayor que el resto de servidores, se ha creado un balanceador de carga en el Servidor de Emparejamiento, el cual permite la utilización de varios servidores de juego y asignar partidas a uno u otro en función del número de partidas que se estén jugando actualmente en cada servidor. Este proceso de auto balanceo se realiza de forma automática y paralela a la ejecución del funcionamiento normal del servidor de emparejamiento. Para ello se ha creado una nueva clase **BalanceadorDeCarga** el cual se encarga de almacenar la información necesaria de los servidores de juego (su dirección ip y el número de partidas) y de proporcionar el servidor cuya carga de trabajo sea inferior. El Código 4.1 muestra el funcionamiento de está nueva funcionalidad, con la cual:

1. Se envía un mensaje a cada servidor de juego conectado solicitando el número de partidas activas que tiene actualmente. Esto ha supuesto una modificación también en el servidor de juego, en el cual se ha tenido que incorporar un sistema de comunicación extra con el servidor de emparejamiento con el cual, cuando el servidor de juego se inicia, envía un mensaje al servidor de emparejamiento para que lo tenga en cuenta en el balanceador de carga.Así como también se ha modificado el protocolo de mensajes del servidor de juego para que acepte esta nueva solicitud.

2. Una vez se recibe la respuesta del servidor, se actualiza la clase **BalanceadorDeCarga** con dicha información.

También se ha visto la necesidad de modificar el proceso de emparejamiento, en el cual ha sido necesario incluir una llamada a la clase **BalanceadorDeCarga** para obtener el servidor al que se le enviará el mensaje de creación de partida.

```
private void ControlBalanceadorDeCarga(){
  while(true){
    foreach (string servidor in balanceador.GetServidores()){
      //Enviar mensaje y actualizar servidor con respuesta
      try{
        PaqueteDatosPartida p = new PaqueteDatosPartida(0,
           PaqueteDatosPartida.BALANCEADOR, -1);
        TcpClient tcpClient = new TcpClient();
        Stream stream = null;
        tcpClient.Connect(servidor, puertoServidorJuego);
        stream = tcpClient.GetStream();
        ASCIIEncoding ascii = new ASCIIEncoding();
        Console.WriteLine(p.GetInformacion());
        byte[] send = ascii.GetBytes(p.GetInformacion());
        stream.Write(send, 0, send.Length);
        byte[] bit = new byte[100];
        int i = stream.Read(bit, 0, bit.Length);
        PaqueteDatosPartida resultado = new PaqueteDatosPartida(
           ascii.GetString(bit, 0, i));
        Console.WriteLine(resultado.GetInformacion());
        if (resultado.GetAccion() == PaqueteDatosPartida.
           BALANCEADOR){//Actualizar la carga del valanceador
          balanceador.ActualizarServidor(servidor, resultado.
             GetIdJugador());
        }
      }
      catch (Exception e){
        EscritorDeLog.AddMensajeALista("[
           ErrorContorBAlanceadorDeCarga] ip_Servidor="+servidor+"
           \r\n" + e);
        Console.WriteLine("ErrorContorBAlanceadorDeCarga =" +
           servidor + ": \n" + e);
      }
    }
    balanceador.RecarcularCargaServidores();
    Thread.Sleep(60000);//1 min
  }
}
```

Código 4.1: Funcion: ControlBalanceadorCarga del servidor de emparejamiento

4.4. Resultados finales de los servidores

En este apartado se pretende mostrar una serie de figuras que muestran el resultado final de los diagramas de clase de los diferentes servidores, incluyendo la implementación de las ampliaciones mencionadas anteriormente.

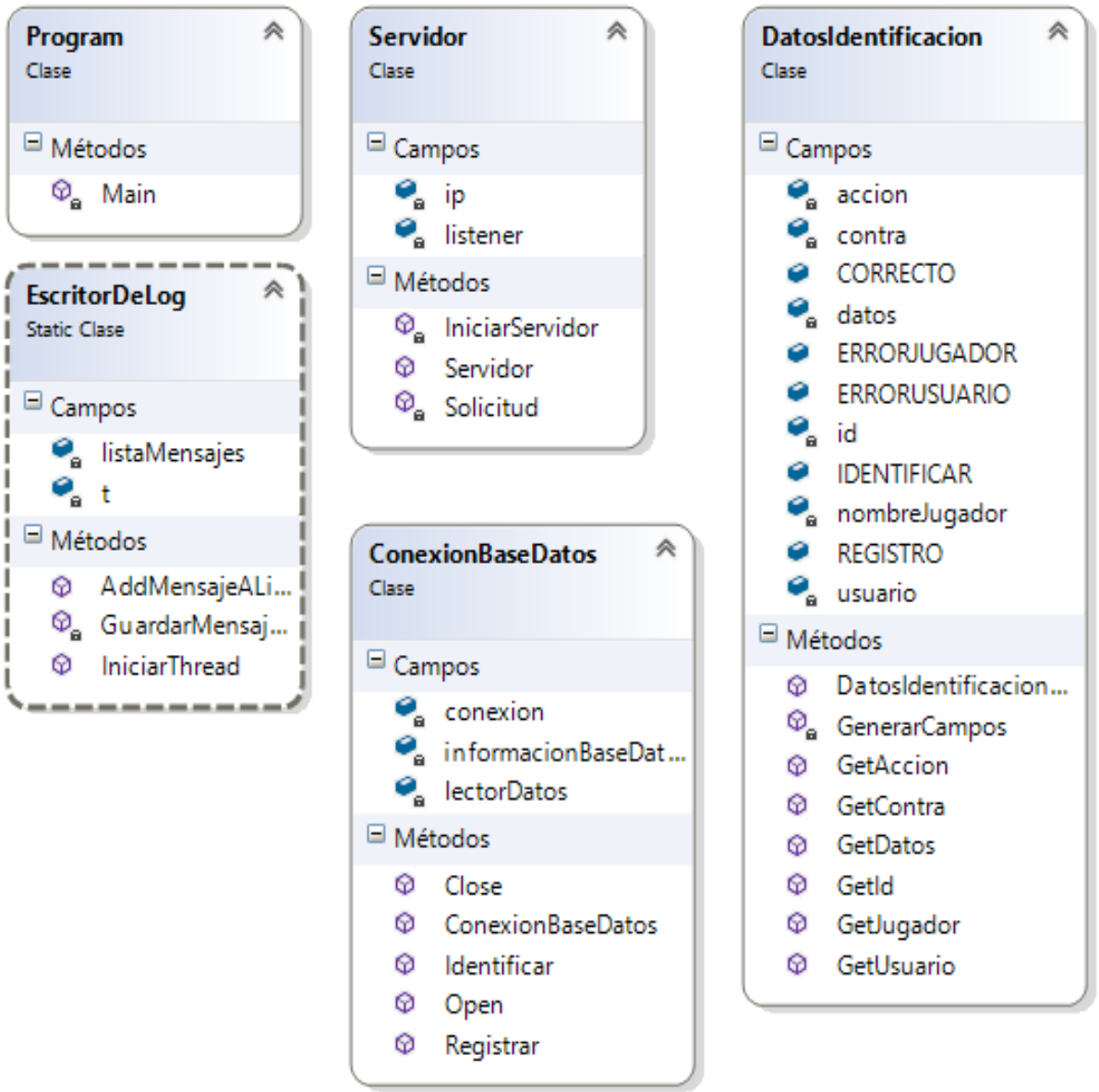


Figura 4.3: Diagrama de clases final del servidor de identificación

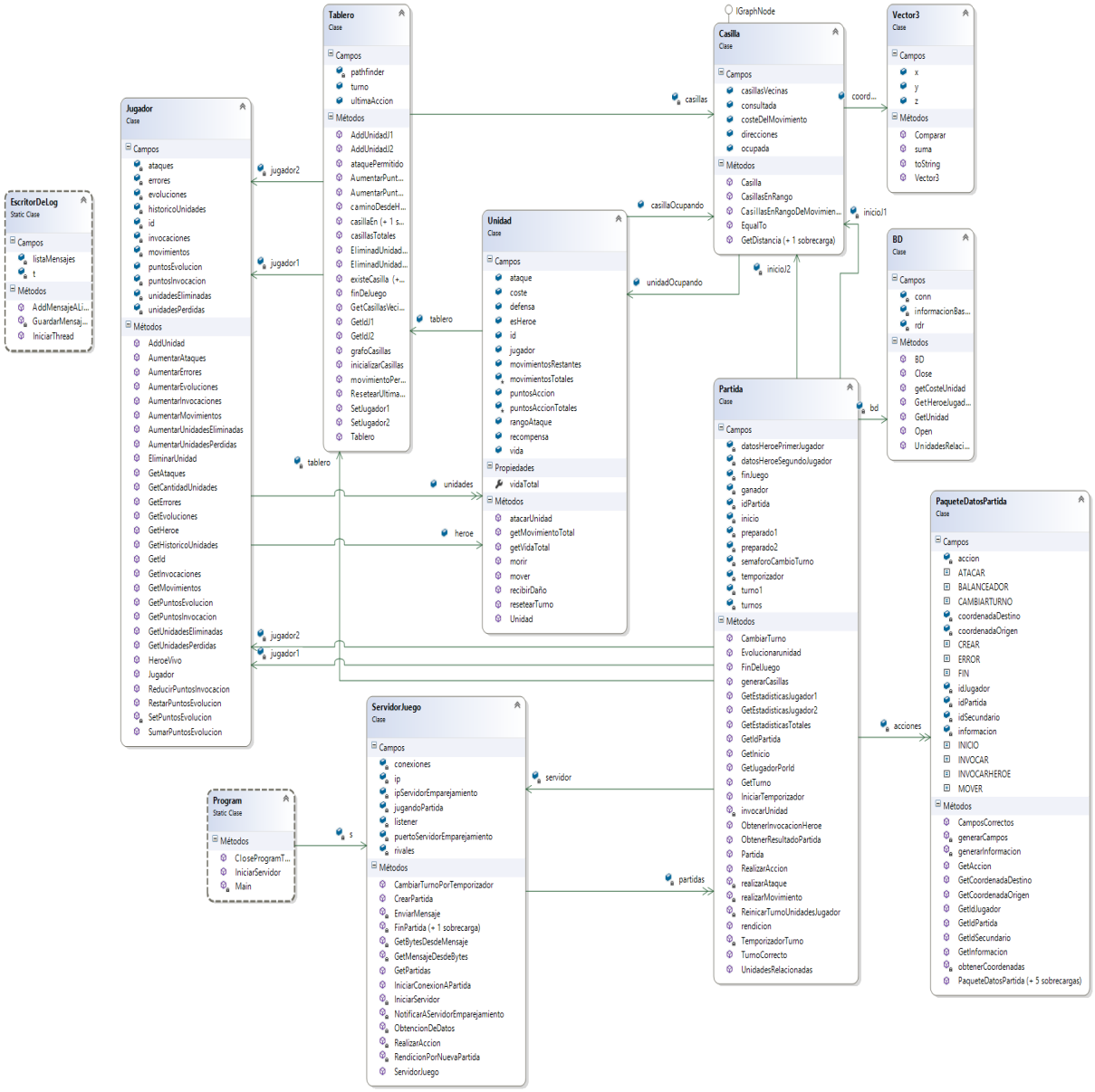


Figura 4.4: Diagrama de clases final del servidor de juego

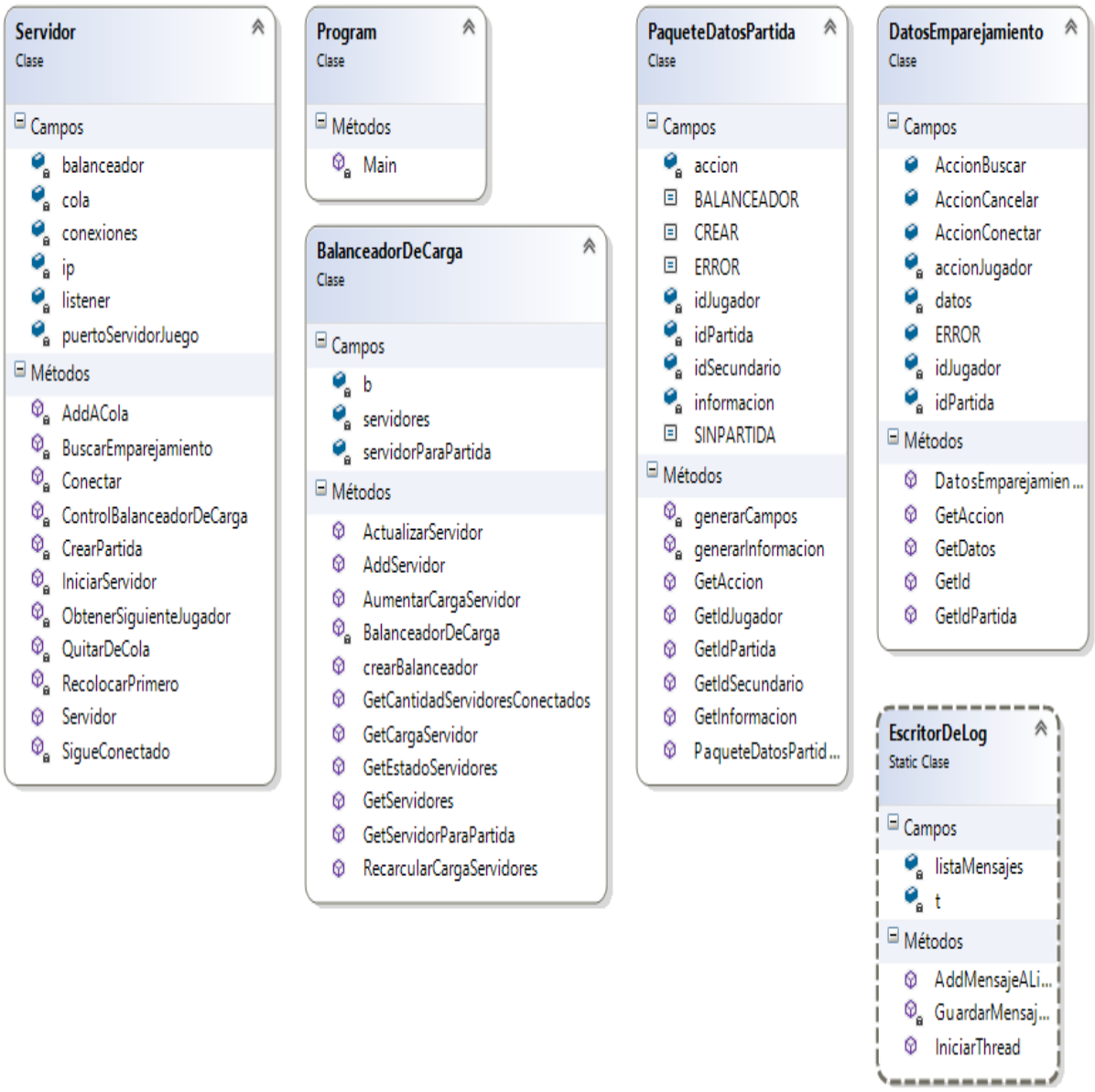


Figura 4.5: Diagrama de clases final del servidor de emparejamiento

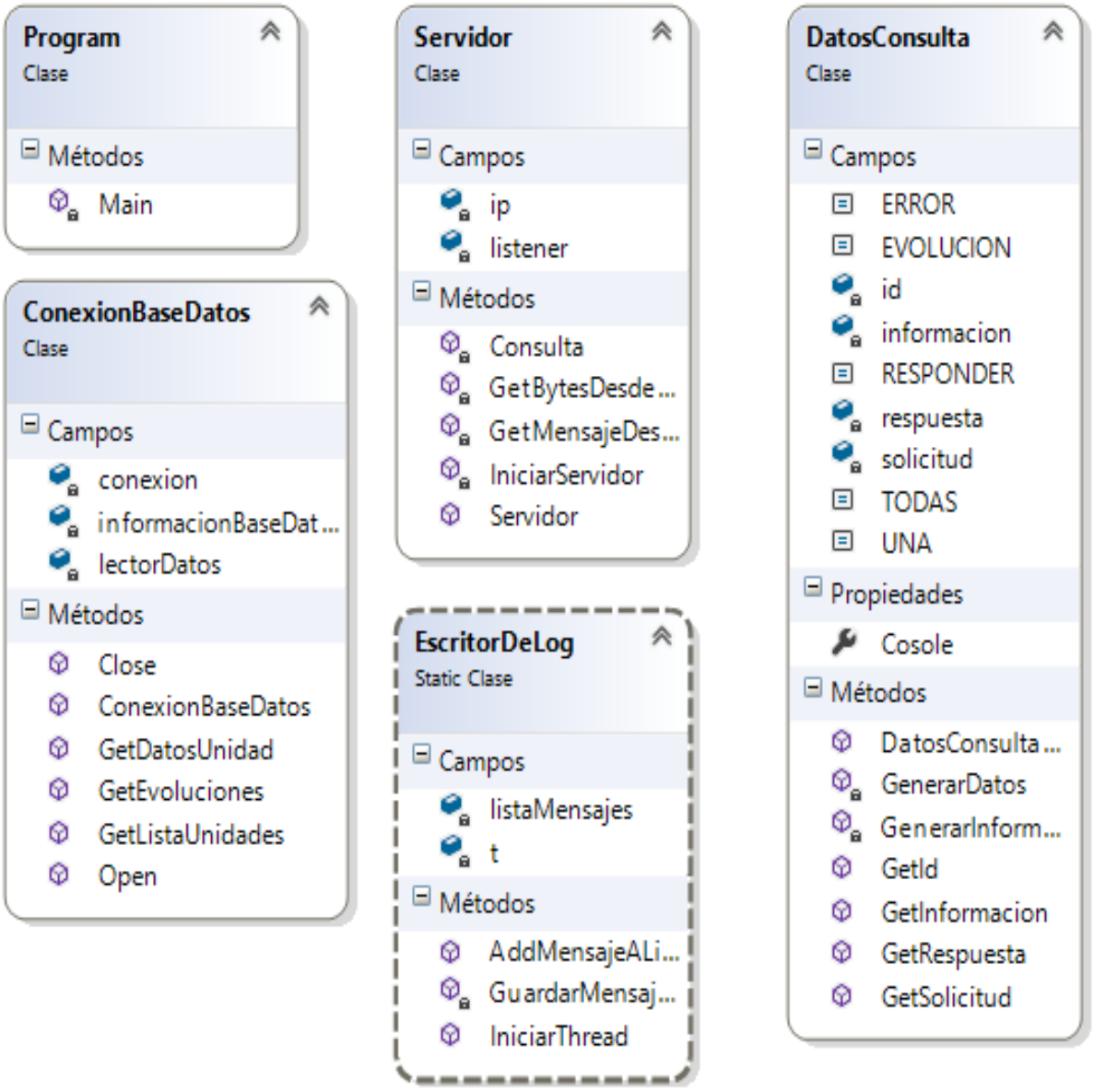


Figura 4.6: Diagrama de clases final del servidor de consultas

5

Conclusiones

Actualmente el mundo de los Videojuegos se encuentra en continua evolución, ello provoca que cada vez resulte un poco más fácil la realización de un videojuego debido a que ciertas empresas se dedican a crear motores que así lo permiten. No obstante, con el gran aumento de videojuegos en el mercado, cada vez se hace más difícil poder mantener un juego en el mismo, sobretodo un juego como el que se ha desarrollado en este proyecto, que depende por completo de los usuarios y sus gustos.

El videojuego desarrollado con este proyecto podría no atraer a muchos usuarios debido a que se trata de un producto mínimo viable, si se finalizara podría comenzar a atraer a más usuarios. No obstante, siempre se debería estar pendiente de las solicitudes y opiniones de los usuarios, pues podrían tener varias sugerencias que podrían ayudar a mejorar el juego, además de evitar que los mismos lo abandonaran e incluso atrajesen a más usuarios.

No obstante, este Trabajo Fin de Grado ha cumplido con los objetivos establecidos pues permite que varios usuarios puedan jugar partidas entre ellos estando en diferentes equipos, sin tener que preocuparse más que por introducir los datos correctos y realizar jugadas válidas. Además se han incorporado nuevas funcionalidades que ayudan a cumplir el objetivo secundario de evitar el cuello de botella, y otras mejoras que pueden ayudar a la detección de errores y al seguimiento del funcionamiento de los servidores.

La mayor dificultad a la hora de desarrollar este Trabajo Fin de Grado, se ha encontrado a la hora desarrollar el servidor de juego, pues requería de una lógica para iniciar partidas y controlar el cambio de turno automático que no se habían contemplado en profundidad al principio del desarrollo, por lo que, tras varios intentos de definir una forma eficaz para el iniciar una partida, se ha necesitado incorporar nuevos tipos de acción como INICIAR e INVOCARHEROE, además del comportamiento que debe seguir tanto el servidor como el juego, que han permitido establecer de forma más precisa el inicio de la partida y del temporizador.

Por otro lado, este Trabajo Fin de Grado también podría ser mejorado implementando nuevas funcionalidades o medidas de seguridad, como podría ser incluir una capa de cifrado

extra en el envió de mensajes entre servidores y usuarios para evitar una fácil obtención de la información que se podría producir en un ataque man-in-the-middle.

Bibliografía

[1] Unity, *Unity Collaborate*, https://unity3d.com/es/services/collaborate, Recuperado en 06/2017

[2] Kenney, *Galería de uso público*, https://kenney.nl/assets, Recuperado en 06/2017

[3] BOE, *Ley 13/2011, de 27 de mayo, de regulación del juego*, https://www.boe.es/boe/dias/2011/05/28/pdfs/BOE-A-2011-9280.pdf, Recuperado en 06/2017

[4] Unity, *Unity Manual*, https://docs.unity3d.com/Manual/index.html, Recuperado en 06/2017

[5] Microsoft, *Referencia de C#*, https://msdn.microsoft.com/es-es/library/618ayhy6.aspx, Recuperado en 06/2017

[6] Microsoft, *Biblioteca de clases de .Net Framework*, https://msdn.microsoft.com/es-es/library/mt472912(v=vs.110).aspx, Recuperado en 06/2017

[7] MySQL, *MySQL 5.7 Reference Manual*, https://dev.mysql.com/doc/refman/5.7/en/introduction.html, Recuperado en 06/2017

[8] darksouls1, *Viking Guerrero Masculina Muscular Encapuchado*, https://pixabay.com/es/viking-guerrero-masculina-muscular-2009503/, Recuperado en 06/2017

[9] Jade lilly, *Fan Art of Katniss*, https://www.flickr.com/photos/jade_lilly/8267232152, Recuperado en 06/2017

[10] Jerad S Marantz, *SUCKER PUNCH DESIGNS*, http://jeradsmarantz.blogspot.com.es/2011/02/sucker-punch-designs.html Recuperado en 06/2017

[11] kirill777, *Necromancer*, https://opengameart.org/content/necromancer Recuperado en 06/2017

[12] Crooked Head, *Turn Based Strategy Framework*, https://www.assetstore.unity3d.com/en/#!/content/50282 Recuperado en 06/2017

Anexos

A

Anexo I: Reglas del juego

A.1. Elementos del Juego

A.1.1. El tablero

El tablero está formado por 73 casillas hexagonales distribuidas como se muestra en la Figura A.1

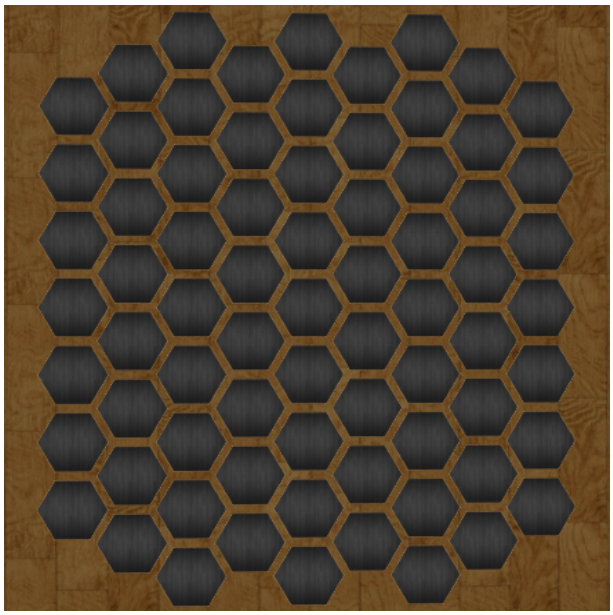


Figura A.1: Tablero

Cada casilla puede, tener una única unidad asociada, o no tener unidad asociada. Cada unidad deberá tener en todo momento, una única casilla asociada obligatoriamente. A las casillas se les puede establecer si se puede pasar, o no, a través de ella, y cuantos puntos de movimiento cuesta hacerlo.

A.1.2. Los jugadores

En cada partida participarán 2 jugadores, numerados como Jugador 1 y Jugador 2. A los cuales se les asignará los siguientes colores, azul para el 1 y naranja para el 2, y se usarán durante el juego para distinguir a cada oponente.

Héroe del jugador

Ambos tendrán una unidad especial (Héroe) sobre el tablero de forma inicial, en la columna central del tablero.El objetivo, por tanto, de cada jugador, es el de eliminar a la unidad especial, que llamaremos Héroe, del jugador enemigo.

Unidades del jugador

Además del héroe, los jugadores podrán tener hasta 6 unidades extra en el tablero.

A.1.3. Las unidades

Valores de las unidades

Cada unidad contará con una serie de valores que determinarán su jugabilidad durante la partida, tales como:

-Vida: Al llegar a 0 la unidad muere.

-Ataque: Puntos de vida que restará a una unidad enemiga al atacar.

-Defensa: Daño que evita la unidad al recibir un ataque

-Rango de ataque: Distancia a la cual una unidad puede realizar un ataque.

-Rango de Movimiento: Distancia por la que se podrá mover como máximo durante un turno.

-Habilidades: Habilidades (activas o pasivas) que tiene una unidad.

-Puntos de acción: Número de veces que puede hacer una acción la unidad por turno, como atacar, o usar una habilidad.

-Puntos de invocación: Puntos de invocación necesarios para ponerla en juego.

Movimiento

Cada unidad cuenta con unos puntos de movimiento y de acción máximos por turno. Cada movimiento a una casilla adyacente, reduce el número de puntos de movimientos el coste de movimiento de dicha casilla, en la mayoría de casos, las casillas tendrán un coste 1, con lo que los puntos de movimiento muestran cuántas casillas se puede mover la unidad.

Para moverse a una casilla, por lo tanto, es necesario que la cantidad de puntos de movimiento sea igual o mayor al coste de dicha casilla.

Ataque y habilidades

Al atacar o usar una habilidad, automáticamente los puntos de movimiento pasan a 0, ya que no podrá moverse más en ese turno.
Los puntos de acción son la cantidad de ataques o habilidades que la unidad puede usar en el turno. En la mayoría de casos será 1, con lo que la unidad no podrá hacer ninguna acción de ningún tipo tras atacar o usar una habilidad.

A.2. El juego

A.2.1. Los turnos

Acciones del jugador en cada turno

En cada turno, el jugador podrá, realizar varias acciones, que se pueden clasificar de la siguiente forma:

- Posicionar una nueva unidad en el tablero:
  Cada jugador tendrá, como se muestra en la Figura A.2, su zona de posicionamiento (ZP), que corresponde con las seis casillas adyacentes al Héroe.

  Para posicionar una nueva unidad será necesario elegir una casilla vacía de la zona para poner ahí la unidad, que no se podrá usar hasta el turno siguiente.
  Las condiciones de posicionamiento de unidad, así como el poder sustituir las unidades por evoluciones de estas, están indicadas en la Sección A.2.4

- Mover una unidad disponible en el tablero:
  Cada unidad se podrá mover por el tablero, dependiendo de movimiento que tenga, para ello elegiremos la unidad, y se nos mostrarán las casillas disponibles a las que la podemos mover. Al hacer clic en una de esas casillas, moveremos allí la unidad.
  En cada turno, cada jugador podrá mover únicamente una unidad. Un ejemplo de la previsualización de movimiento se puede ver en la Figura A.3

- Ejecutar una acción de ataque de una unidad:
  Cada unidad podrá a atacar a otra enemiga, dependiendo de si está dentro del rango de ataque de la unidad, que puede variar, de la misma forma que el movimiento, se nos mostrarán las unidades enemigas atacables y podremos hacer clic en ellas para ejecutar el ataque. Un ejemplo de la previsualización de ataque se puede ver en la Figura A.4

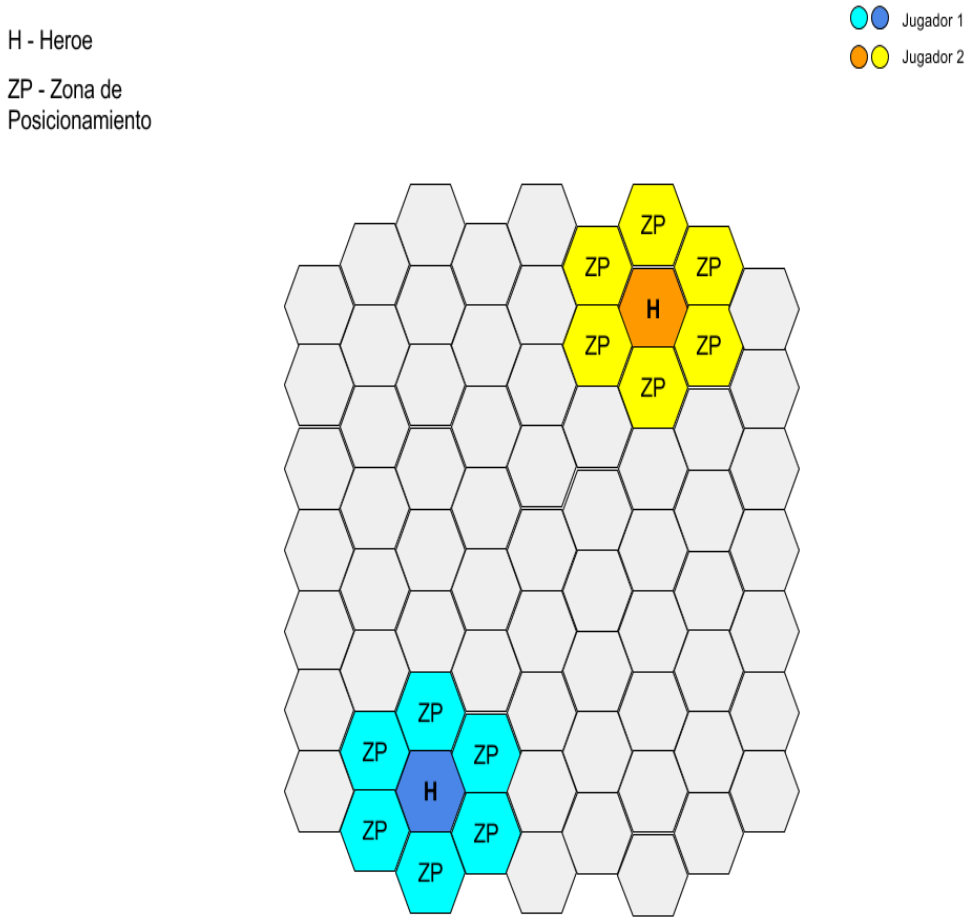


Figura A.2: Zonas Invocación

Duración de un turno

El turno termina cuando elegimos terminar el turno, cuando se acabe el tiempo, o cuando terminamos nuestras acciones.
El tiempo máximo de duración de un turno es de 30 segundos.

A.2.2. Eliminación de unidades y ataque

Cuando una unidad ataca a otra, la unidad atacada recibe puntos de daño de la siguiente forma:
Al recibir un daño X, si la unidad tiene 1 de defensa o más, el daño restará X puntos de defensa, hasta llegar a 0.

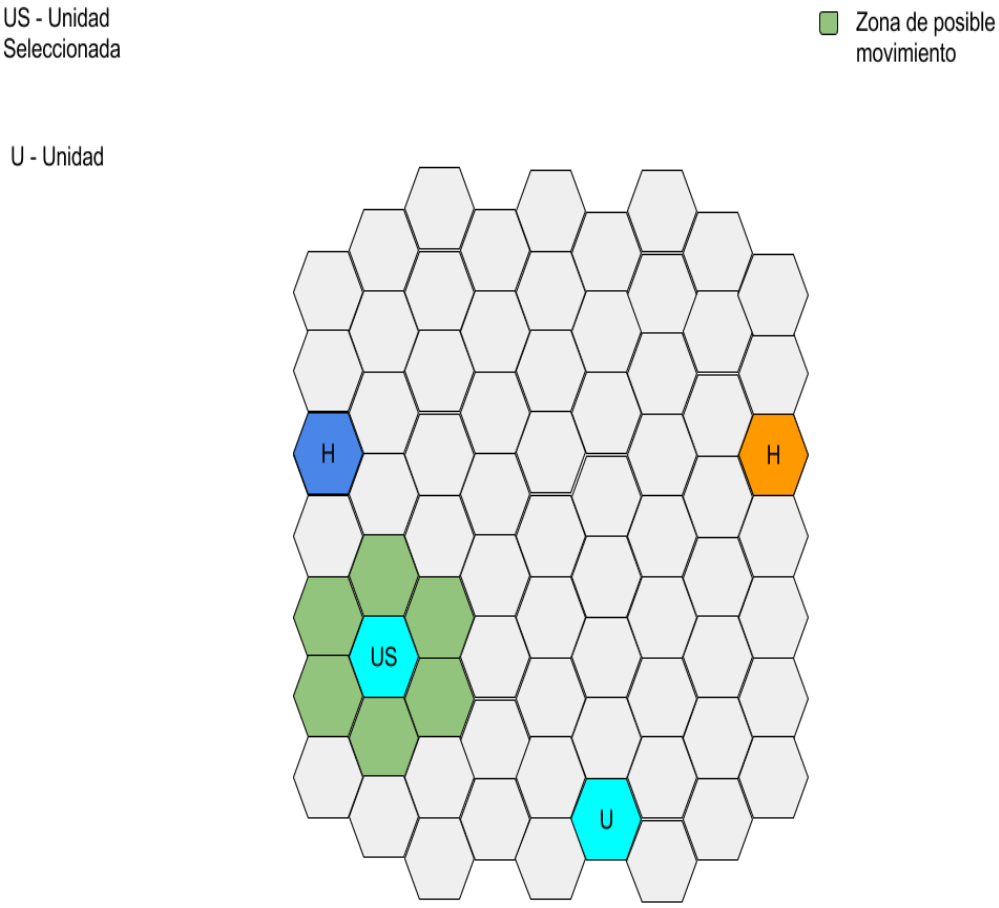


Figura A.3: Zonas de Movimiento

Al recibir ese daño X, si la unidad tiene una defensa igual a 0, se reducirán X puntos de vida a la unidad, hasta llegar a 0.

Si los puntos de vida de una unidad llegan a 0, dicha unidad es eliminada.

Ejemplos:

- Una unidad con vida 4, defensa 4, recibe un daño 3 – termina con vida 4 defensa 1.
- Una unidad con vida 4, defensa 1, recibe un daño 3 – termina con vida 4 defensa 0.
- Una unidad con vida 4, defensa 0, recibe un daño 3 – termina con vida 1 defensa 0.
- Una unidad con vida 1, defensa 0, recibe un daño 3 – es eliminada.

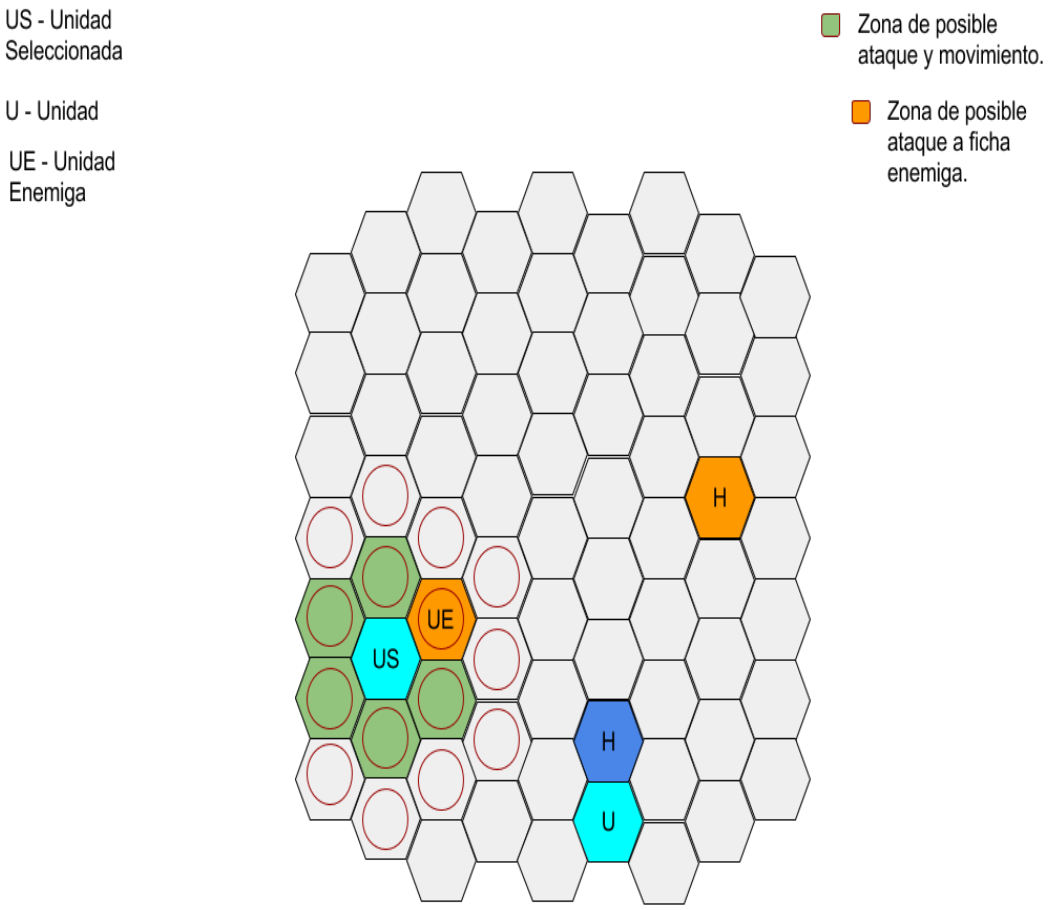


Figura A.4: Zonas de Ataque

A.2.3. Jugador ganador de la partida

Cuando un jugador consigue eliminar al Héroe enemigo, dicho jugador es el ganador de la partida.

A.2.4. Las Invocaciones

Invocación de unidad

Las invocaciones son las habilidades principales de los héroes, como se especifica anteriormente, cada jugador solo puede tener 6 unidades invocadas a la vez.

Invocación de Unidad Inicial

Las invocaciones son las habilidades principales de los héroes. Una vez por turno se puede invocar una unidad a elegir de entre las 3 disponibles de nivel 1.
Además, durante la partida únicamente se pueden realizar una cantidad de 10 invocaciones iniciales
Las unidades únicamente podrán ser invocadas en las casillas circundantes al héroe.

Invocación por evolución

Se pueden usar puntos de evolución para sacrificar una unidad del tablero, eliminándola y sustituyendola por una evolución, (una unidad 1 nivel mayor a la sacrificada, que se encuentra en su árbol de evoluciones)
Cuando una unidad es invocada (inicial o evolución), no puede realizar ninguna acción durante el turno de su invocación (a menos que tenga una habilidad que se lo permita).

Obtención de puntos de evolución

Los Puntos de Evolución pueden ser obtenidos eliminando eliminando a las unidades enemigas del rival. Esta cantidad de puntos puede variar dependiendo de la unidad eliminada.

Árboles de Invocación

Cada unidad inicial contará con un árbol de evoluciones, el cual determinará el proximo tipo de invocación que se podrá realizar al evolucionar dicha unidad usando los puntos de evolución necesarios.
Cada nivel de profundidad del árbol indicará el nivel de las unidades, habiendo un máximo de profundidad 3, y por lo tanto nivel máximo 3.

El diagrama de la Figura A.5 podría representar el árbol de evoluciones disponible para la unidad 1. Además, podría darse el caso de que el árbol de invocación de varias unidades distintas de nivel 1 converjan llegado a cierto nivel.

A.3. Enumeración de unidades

A.3.1. Héroes

El héroe es la unidad más importante del jugador, ya que su muerte implica perder la partida. No obstante el héroe no es más que una unidad, que está situada en el tablero de forma inicial, con las siguientes particularidades:

- Durante la partida no se puede invocar ningún otro Héroe, es único.
- Tiene la habilidad "Invocar" que permite invocar una unidad de nivel 1.

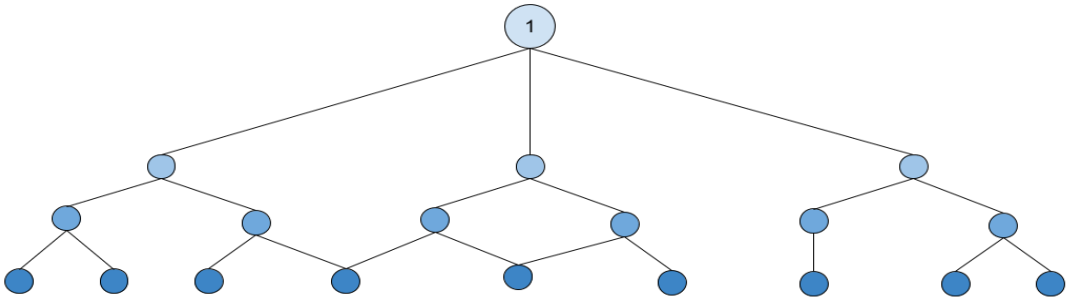


Figura A.5: Zonas de Ataque

- Inicialmente está en la columna central del tablero, de la fila superior o inferior, dependiendo del jugador al que pertenezca.

(En un futuro queremos que haya varios héroes, entre los cuales el jugador pueda elegir cuál quiere usar, y dependiendo del héroe podrá invocar distintas unidades de nivel 1, no obstante para el Trabajo de Fin de Grado hemos decidido que el héroe sea fijo, y el mismo para ambos jugadores, y por lo tanto tenga opciones de invocación fijas)
En el Cuadro A.1 se pretende mostrar las valores base que tendrá la unidad héroe.
Significado de las cabeceras:

-Base: Unidad desde la que es evolucionada.

-Nombre: Nombre de la unidad.

-A: Ataque de la unidad.

-V: Vida de la unidad.

-D: Defensa de la unidad.

-R.M: Rango de movimiento de la unidad.

-R.A: Rango de Ataque de la unidad.

-P.A: Puntos de acción de la unidad.

-P.I: Puntos de invocación necesarios para invocar a dicha unidad, en el caso del héroe indican la cantidad de unidades que podrá invocar el jugador durante toda la partida.

-Rec: Puntos de invocación que se obtienen como recompensa por elimiar a dicha unidad.

-T.A: Tipo de ataque de la unidad.

-Vel: Velocidad de la unidad.

Nombre	A.	V.	D.	R.M.	R.A.	P.A.	P.I.	Rec.	T.A.	Vel,
Héroe	1	30	5	2	1	2	10	0	0	3

Cuadro A.1: Héroe.

A.3.2. Unidades Iniciales

En el Cuadro A.2 se pretende mostrar las unidades que pueden ser invocadas por el jugador en las casillas adyacentes al héroe.

Nombre	A.	V.	D.	R.M.	R.A.	P.A.	P.I.	Rec.	T.A.	Vel.
Guerrero	1	2	2	2	1	2	1	1	0	3
Arquero Aprendiz	1	1	1	3	2	3	1	1	0	3
Mago Aprendiz	2	1	0	1	3	3	1	1	0	3

Cuadro A.2: Unidades Iniciales.

A.3.3. Evoluciones

En el Cuadro A.3 se pueden observar las unidades de nivel 2 a las que es posible evolucionar las unidades iniciales.
Así mismo, en los cuadros A.4, A.5 y A.6 se pueden observar las evoluciones de nivel 3 según la rama de guerrero, arquero y mago respectivamente.

Base	Nombre	A	V	D	R.M.	R.A	P.A.	P.I	Rec.	T.A.	Vel.
Guerrero	Guerrero n2 tanque	2	1	0	1	3	3	1	1	0	3
Guerrero	Guerrero n2 ataque	3	2	2	2	1	2	2	2	0	3
Guerrero	Guerrero n2 mov. ataque	3	2	1	3	1	3	1	1	0	5
Arquero Aprendiz	Arquero n2 armadura	1	3	1	1	3	1	2	2	1	4
Arquero Aprendiz	Arquero n2 ataque	3	2	0	1	3	1	2	2	1	4
Arquero Aprendiz	Arquero n2 mov.	1	2	0	2	3	2	1	1	1	4
Mago Aprendiz	Mago n2 armadura	2	2	2	1	2	1	2	2	2	2
Mago Aprendiz	Mago n2 ataque	4	2	0	1	2	1	2	2	2	2
Mago Aprendiz	Mago n2 armadura	2	2	2	1	3	1	2	2	2	2

Cuadro A.3: Unidades Nivel 2.

Base	Nombre	A	V	D	R.M.	R.A	P.A.	P.I	Rec.	T.A.	Vel.
Guerrero n2 tanque	Guerrero n3 tanqueta	1	5	8	1	1	1	4	3	0	2
	Guerrero n3 tanque daño	3	3	5	1	1	1	4	3	0	2
Guerrero n2 ataque	Guerrero n3 lancero	3	2	2	2	2	2	4	3	0	3
	Guerrero n3 tanque daño	3	3	5	1	1	1	4	3	0	2
Guerrero n2 mov. ataque	Guerrero n3 mov. ataque ataque	5	2	1	3	1	3	4	3	0	5

Cuadro A.4: Unidades Nivel 3 Rama Guerrero.

Base	Nombre	A	V	D	R.M.	R.A	P.A.	P.I	Rec.	T.A.	Vel.
Arquero n2 armadura	Arquero n3 armadura rango	1	3	1	1	4	1	4	3	1	4
	Arquero n3 armadura ataque	1	3	1	1	4	1	4	3	1	4
Arquero n2 ataque	Arquero n3 armadura ataque	1	3	1	1	4	1	4	3	1	4
	Arquero n3 ataquisimo	5	2	0	1	3	1	4	3	1	4
Arquero n2 mov.	Arquero n3 mov. ataque	2	2	0	2	3	2	4	3	1	4
	Arquero n3 mov. def.	2	2	2	2	3	2	4	3	1	4

Cuadro A.5: Unidades Nivel 3 Rama Arquero.

Base	Nombre	A	V	D	R.M.	R.A	P.A.	P.I	Rec.	T.A.	Vel.
Mago n2 armadura	Mago n3 armadura ataque	4	2	2	1	2	1	4	3	2	2
	Mago n3 armadurisima	2	3	6	1	2	1	4	3	2	2
Mago n2 ataque	Mago n3 armadura ataque	4	2	2	1	2	1	4	3	2	2
	Mago n3 ataquisimo	7	2	0	1	2	1	4	3	2	2
Mago n2 rango	Mago n3 rango ataque	4	2	0	1	3	1	4	3	2	2
	Mago n3 rango rango	2	2	0	1	5	1	5	3	2	2

Cuadro A.6: Unidades Nivel 3 Rama Mago.

B

Anexo II: Análisis de requisitos

B.1. Actor objetivo del software: Usuario Jugador

Perfil: Persona de cualquier edad que usa la aplicación con la finalidad de jugar una partida.
Objetivo: Obtener un entretenimiento con el juego.

B.2. Requisitos Funcionales

B.2.1. El jugador puede:

1. Registrarse.
   - 1.1. Rellenará un formulario de datos obligatorios.
   - 1.2. El sistema guardará el nuevo usuario.
2. Identificarse.
   - 2.1. Introducirá los datos necesarios para identificarse.
   - 2.2. El sistema verificará los datos.
3. Ver un listado de las unidades del juego.
   - 3.1. Puede ver la información detallada de cada una.
4. Seleccionar el héroe que quiere usar.
5. Iniciar una partida.
   - 5.1. Iniciar una partida en linea.
     - 5.1.1. El sistema emparejará a los jugadores.
   - 5.2. Iniciar una partida de un jugador.
     - 5.2.1. El sistema creará una partida local contra la IA.
   - 5.3. El sistema creará una partida nueva.

5.3.1. El sistema posicionará el héroe elegido por cada jugador en su lado del tablero.

5.3.2. Establecerá el turno inicial de forma aleatoria.

6. Invocar unidades de apoyo

6.1. El sistema proporcionará las unidades disponibles para invocar.

6.1.1. El jugador puede elegir cuál de estas invoca.

6.2. EL sistema establecerá las casillas disponibles para posicionar la unidad.

6.2.1. El jugador puede elegir donde la posiciona.

7. Realizar acciones con una unidad durante la partida.

7.1. Puede seleccionar una unidad propia.

7.1.1. El sistema mostrará las acciones disponibles para dicha unidad.

7.2. Mover una unidad.

7.3. Atacar con la unidad seleccionada a una unidad enemiga.

7.3.1. Puede eliminar una unidad enemiga.

7.4. Evolucionar la unidad seleccionada.

7.4.1. El sistema mostrará las posibles evoluciones.

7.4.1.1. El jugador elegirá a que unidad evolucionará.

7.5. Puede cancelar la acción.

8. Cambiar de turno.

9. Rendirse.

10. Ver información de la partida.

10.1. Puede ver información detallada sobre las unidades en juego.

10.2. Puede ver información sobre los jugadores.

B.2.2. El sistema:

1. Detectará el final del turno.

1.1. Detectará si el jugador quiere cambiar de turno.

1.2. Detectará si al jugador no le quedan acciones disponibles.

1.3. Detectará si se ha acabado el tiempo del turno.

1.4. El sistema cambiará el turno al jugador contrario.

2. Detectará el final de la partida.

2.1. Detectará si un jugador derrota al héroe rival.

2.2. Detectará si un jugador decide rendirse.

2.3. El sistema mostrará en pantalla el resultado de la partida.

2.4. El jugador podrá volver al menú principal.

B.3. Requisitos No Funcionales

B.3.1. Requisitos de Aspecto

Interfaz

1. - Se establecerán formularios para registrarse e identificarse.

2. - El usuario podrá ver una representación del juego sobre el tablero (unidades aliadas y enemigas).

   2.1. - También se deberá poder visualizar e identificar las acciones de las unidades.

3. En la partida, las casillas y unidades serán objetos seleccionables.

4. Cuando se seleccione una unidad debe ser visible la siguiente información:

   4.1. Toda la información de dicha unidad.

   4.2. Todas las casillas a las que se puede mover y todas las unidades y/o casillas en rango de ataque.

   4.3. Una visualización con la información básica de todas las unidades a las que se puede evolucionar dicha unidad (si tiene evoluciones) o en caso del héroe, de las unidades que puede invocar.

5. Todas las unidades (héroe incluido) sobre el tablero de juego deben tener visibles su vida, defensa y ataque en todo momento.

6. Durante la partida deberá ser visible los puntos de evolución de los jugadores así como sus puntos de invocación restantes.

El estilo del producto

1. La aplicación diferenciará a los jugadores durante una partida con colores Azul (jugador) y Naranja/Amarillo (rival).

2. La fuente utilizada será Arial.

3. Se emulará un juego de tablero de madera.

B.3.2. Requisitos de Facilidad de Uso y Aprendizaje

Facilidad de uso

1. La interfaz de usuario debe adaptarse a las acciones de la partida. Al pasar el ratón por encimad e los elementos de juego aparecerá información sobre ellos.

2. El Español será el idioma usado en el juego.

3. La aplicación debe verse correctamente en cualquier tamaño de pantalla y resolución.

4. EL sistema debe poseer interfaces gráficas dinámicas. Algunos botones cerrarán y abrirán paneles.

5. La distancia entre elementos de la interfaz deben tener una separación suficiente para que todo el conjunto se distinga con claridad.

Facilidad de aprendizaje

1. El sistema debe poder ser usado correctamente después de la primera partida.

2. El tiempo de aprendizaje básico del sistema por un usuario deberá ser menor a 2 horas.

B.3.3. Requisitos de Funcionamiento

Requisitos de Velocidad

1. Toda acción es respondida por el servidor en un tiempo máximo de 5s.

2. Las acciones de juego deben ser respondidas en un tiempo máximo de 2s.

3. La inteligencia artificial realizará las acciones en un tiempo máximo de 10s.

4. La rapidez de la búsqueda de partida multijugador en lina, dependerá de la cantidad de usuarios que quieran jugar esta modalidad.

Requisitos de Seguridad Crítica

1. El sistema revisará las acciones enviadas al servidor.

Requisitos de Fiablidad y Disponibilidad

1. El sistema debe estar disponible con un nivel de servicio temporal para los usuarios 7 días x 24 horas x 365 días al año.

2. En caso de enviar una acción no disponible al servidor, no la ejecuta.

Requisitos de Escalabilidad

1. El sistema debe usar de forma óptima los recursos de conexiones a la base de datos.

    1.1. La base de datos debe permitir ampliar su capacidad de almacenamiento.

2. El sistema debe tener una clara partición entre datos, recursos y aplicaciones.

3. El sistema debe poder ampliar su capacidad para la gestión de un incremento en la cantidad de usuarios e información.

B.3.4. Requisitos Operacionales

Entorno Físico

1. La aplicación será usada en ordenadores y no es portable, por lo cual será un entorno amigable.

Entorno Tecnológico

1. El sistema será usado por el jugador a través de la aplicación Unity.

2. El hardware necesario para ejecutarlo es de minimos requerimientos.

3. La aplicación podrá ser instalada en cualquier Sistema Operativo de Windows actual.

Soporte

1. Introducir nuevas unidades implica añadirlas a la Base de datos.

2. Introducir nuevas mecánicas de juego implica modificar el código del juego y del servidor.

B.3.5. Requisitos de Mantenimiento y Portabilidad

¿Cuál es la dificultad de mantenimiento de este producto?

1. Al tratarse de un juego, es muy recomendable introducir nuevas unidades y héroes, así como ir comprobando la experiecia de juego de nuestros usuarios.

¿Existen condiciones especiales aplicables al mantenimiento de este producto?

1. Las actualizaciones del juego requerirán descargarse la nueva aplicación completamente.

Requisitos de portabilidad

1. La aplicación solo será utilizable desde un orndenador con SO Windows.

B.3.6. Requisitos de Seguridad

¿El sistema es fiable?

1. Todos los usuarios deben iniciar sesión en el sistema para poder utilizarlo.

    1.1. Los usuarios deben ser identificados y autenticados con los datos de usuarios en la Base de Datos.

2. El sistema debe desarrollarse aplicando patrones y recomendaciones de programación que incrementen la seguridad de datos.

B.3.7. Requisitos Culturales y Políticos

¿Existe algún factor especial sobre el producto que lo pudiera hacer inaceptable por motivos políticos?

El juego puede ser considerado como bélico, a pesar de no tener imágenes de violencia y ser apto para todos los públicos. Sin embargo, pertenece más al género de estrategia, o juego de mesa.

B.3.8. Requisitos legales

El sistema no almacenará ningún dato personal de los usuarios como nombre, edad, sexo...

Jurisdicción

1. El Artículo 18.4 de la constitución establece: "La ley limitará el uso de la informática para garantizar el honor y la intimidad personal y familiar de los ciudadanos y el pleno ejercicio de sus derechos.

2. LA ley 11/1998, «Ley General de Telecomunicaciones» que regula la privacidad y servicios en la red.

3. La Ley 13/2011, de 27 de mayo, de regulación de juego.

Estándares a cumplir

1. El estándar ISO 8402-94 que esablece la calidad de data a auditorías.

2. El estándar ISO 9126 de IEEE que establece la calidad y mantenibilidad de un producto software.