UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Arquitectura web para un simulador táctico de escaramuzas

Antonio Diego Barquero Cuadrado

Septiembre, 2015

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Arquitectura web para un simulador táctico de escaramuzas

Autor: Antonio Diego Barquero Cuadrado

Fdo.:

Director: Antonio Manuel Silva Luengo

Fdo.:

Tribunal Calificador

Presidente: José Moreno del Pozo

Fdo.:

Secretario: Pedro Miguel Núñez Trujillo

Fdo.:

Vocal: Pablo García Rodríguez

Fdo.:

CALIFICACIÓN:

FECHA: Septiembre, 2015

Agradecimientos

Este trabajo marca el fin de una época muy importante de mi vida. A mis padres por la confianza y el apoyo que me han dado.

Sumario

En sus inicios, los videojuegos eran creados para un sector de personas muy concreto y reducido. Desde la visión de la industria del entretenimiento, la concepción y el desarrollo de estos no se podían comparar a ningún otro producto como el cine o la televisión. En los últimos años hemos sido testigo de cómo se han ido incluyendo entre las alternativas de ocio y de cómo su desarrollo ha ido aumentando en complejidad debido a la gran variedad de aspectos que abarca.

La implantación de Internet y el aumento de uso de los dispositivos portátiles también han supuesto un gran cambio en la vida de todo el planeta. La comunicación entre las personas es cada día más inmediata y sencilla.

En algunas asignaturas del plan de estudios asociado a este trabajo, se ha podido enseñar a los alumnos algunas nociones sobre el desarrollo web y la computación gráfica. La finalidad de este es el crear una plataforma web que permita albergar un juego.

Abstract

In the beginning, video games were created for a tiny and specific amount of people. From the perpective of the entertainment industry, the design and development of video games can’t be compared to any other product like movies or television. In recent years, we have been witnees of how these products have been included in the leisure time of the society and how their development complexity have been increased.

The Internet penetration and the growing use of portable devices have also been a great change in the life of the entire planet. The communication between people is becoming more immediate and easy day by day.

In some subjects of this degree, the students can learn web development and graphic computing. The goal of this work is to create a web platform to host a game.

Contenido

Resumen

- **Capítulo 1: Introducción:** retos y objetivos del trabajo. Establece las líneas básicas que se han seguido desde el inicio y las tecnologías que se han utilizado.
- **Capítulo 2: Diseño de la aplicación:** principios lógicos y matemáticos de todos los elementos que forman parte de la aplicación. Se explica cómo se han utilizado los hexágonos para la creación del mapa, el comportamiento de las unidades, las batallas, todos los algoritmos asociados y los detalles sobre las estructuras de datos utilizadas tanto en servidor como en cliente para cada función.
- **Capítulo 3: Arquitectura del sistema:** explicación de la arquitectura utilizada, esquema de base de datos, optimizaciones y particularidades de la tecnología del servidor.
- **Capítulo 4: Pruebas:** pruebas gráficas realizadas sobre distintas librerías.
- **Capítulo 5: Software auxiliar:** se detalla el funcionamiento de los programas auxiliares desarrollados**.**
- **Capítulo 6: Manual de usuario:** explicación detallada de todas las funcionalidades de la aplicación.
- **Capítulo 7: Trabajos futuros:** posibilidades de mejora y ampliación de la aplicación.

Capítulo 1: Introducción

1.1 Objetivos

Antes de trazar ninguna idea concreta sobre la temática del desarrollo y antes de la primera reunión con el tutor, me establecí cuatro retos:

1. La aplicación tenía que ser web.
2. Las tecnologías utilizadas debían de ser desconocidas para mí en la medida de lo posible.
3. Tenía que tratarse de un juego.
4. Toda la aplicación debía ejecutar código nativo en cliente.

De forma opcional, también utilizar el lenguaje Python también fue propuesto.

1.2 Tecnología utilizada

Una alternativa para acometer el trabajo es el *stack* de desarrollo web Javascript conocido como MEAN (1) el cual incluye:

- MongoDB (2): motor de bases de datos no relacional.
- Express (3): *framework* de Node.js.
- AngularJS (4): *framework* web para desarrollo *front-end*.
- Node.js (5): servidor de aplicaciones web.

Está especialmente ideado para aplicaciones con necesidades de proceso concurrente bajo una plataforma escalable y ágil. Ya utilicé en el pasado AngularJS y en algunos casos puede tener problemas de incompatibilidad con librerías externas y era necesaria una gran libertad de prueba de diferentes librerías por el peso del cliente en la aplicación, es por esto que se desechó el uso de este *stack*.

1.2.1 *Back-end*

Al rechazar una solución con todos los componentes ya incluidos, se hizo necesario descomponer el desarrollo en partes fundamentales y elegir una tecnología concreta para cada una de ellas. En la parte servidor, se optó por Django (6), un *framework* en Python con muchas características listas para utilizar, entre otras:

- Sistema de plantillado dinámico con herencia.
- ORM (7) integrado para prácticamente todos los motores de bases de datos relacionales más frecuentemente utilizados.
- Posibilidad de utilizar un gran número de módulos ya desarrollados de forma cómoda.
- Servidor de aplicaciones local integrado.
- Documentación extensa y una gran comunidad de desarrollo detrás.

Por último, se eligió el motor de bases de datos relacional MySQL (8) para utilizar junto a Django.

1.2.2 *Front-end*

En el cliente es necesario establecer dos necesidades: gráficos e interfaz. Por el lado de la interfaz se optó por utilizar jQuery (9) y Underscore (10) para la lógica asociada y Gumby (11) para el estilo el cual utiliza SASS (12). Con esto se mantiene el carácter nativo de la aplicación y el diseño es adaptable a todas las resoluciones.

Para los gráficos, existen multitud de posibilidades de realizarlo de forma nativa. Se encontró una librería llamada jCanvas (13) la cual, bajo jQuery era capaz de mostrar gráficos y animaciones de forma ligera sin necesidad de crear recursos gráficos propios. Su documentación aunque escasa era suficiente y en un tiempo razonable obtuve resultados tangibles.

Tras desarrollar un prototipo de aplicación con el entorno gráfico integrado y realizar **pruebas**, se optó por cambiar de librería a una más eficiente llamada Phaser (14) la cual daba una tasa de refresco muy alta con muchos elementos en pantalla. Además ofrecía una documentación extensa y bien estructurada, ejemplos, tutoriales y foros de consulta. Como inconveniente, es necesario crear los elementos gráficos con un programa externo y se quería conservar

el aspecto gráfico logrado, por lo que con jCanvas se desarrolló un **creador de *sprites*** (15) para utilizarlos en Phaser**.**

1.3 Entorno de desarrollo

Las tecnologías y herramientas utilizadas para la realización del siguiente trabajo son las siguientes:

- Sistema operativo: Ubuntu 14.04 x86_64.
- Django: 1.7.4.
- Python: 2.7.6.
- MySQL: 5.5.41.
- jQuery: 2.0.2.
- Phaser: 2.3.0.
- Underscore: 1.8.3.
- GIMP: 2.8.10.
- IDE: PyCharm 4.0.5.

1.4 Concepto del juego

El concepto y la temática del juego es el siguiente: se trata de un simulador de batallas entre grupos de unidades de distintos tipos sobre un mapa de celdas hexagonales. Sería tarea del usuario la gestión, organización y mejora de las tropas. La interfaz se trataría en su mayoría de menús al uso de cualquier aplicación web, pero la visualización de los combates tenía que realizarse de forma gráfica.

Capítulo 2: Diseño de la aplicación

2.1 Mapas

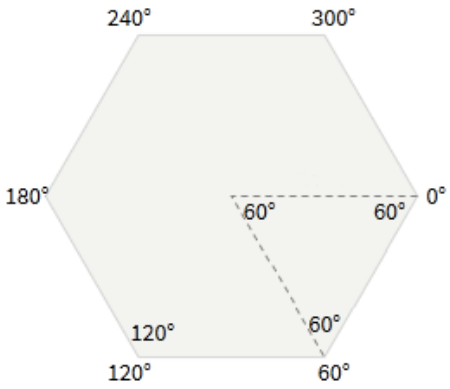
En la simulación de este trabajo y en los juegos de tipo RTS (16), el mapa es un elemento esencial ya que gracias a este la experiencia de usuario podrá variar en cada ejecución.

Para este trabajo se ha utilizado un mapa con posiciones hexagonales. Como base se ha utilizado este recurso (17) el cual recoge el trabajo de más de 20 años y ofrece una serie de alternativas y orientaciones útiles a la hora de trabajar con este tipo de mapas.

2.1.1 Hexágonos regulares

Un hexágono regular es un polígono de 6 lados. Todos tienen la misma longitud. Sus ángulos internos miden 120º.

Cada lado de los hexágonos del mapa creado para este trabajo tiene una longitud de 20 píxeles. En función de este tamaño se han **creado** todos los recursos gráficos.



2.1.2 Rotación de los hexágonos

A la hora de situar los hexágonos, existen dos tipos de orientaciones:

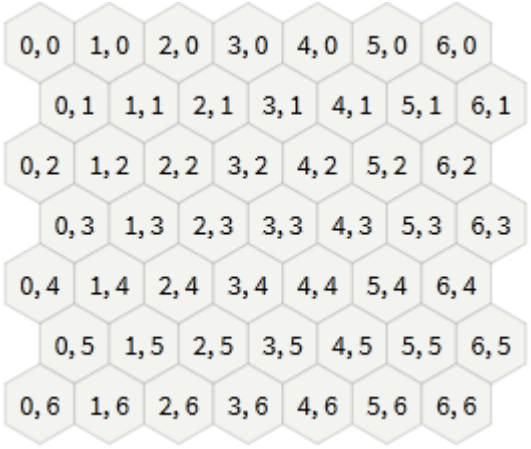
Para este trabajo se ha elegido la segunda opción ya que ofrece un resultado final más adaptable para el diseño de la aplicación. Se trata de una decisión fundamental ya que de esto depende todo el trabajo posterior.

2.1.3 Sistema de coordenadas

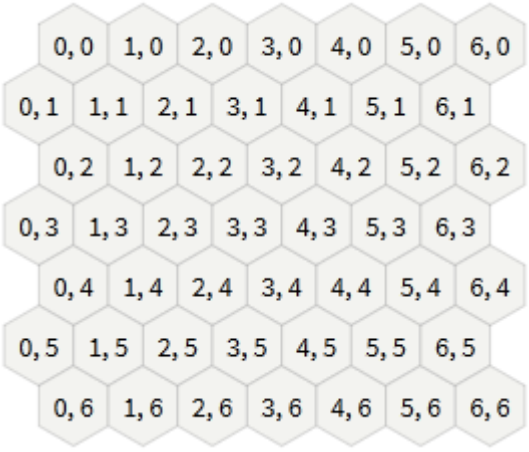
Para identificar las posiciones del mapa, se ha optado por un sistema en dos dimensiones con coordenadas que indican la fila y la columna. Se ha elegido esta aproximación de entre las que se proponen en la documentación sobre mapas hexagonales (17) por ser más natural a la hora de almacenar y procesar la información en una estructura bidimensional.

Con las dos posibles formas de rotar los hexágonos, dependiendo del desplazamiento bien de la fila o de la columna y con el sistema de coordenadas en 2 dimensiones, es posible posicionarlos e identificarlos para formar una rejilla de casillas hexagonales de cuatro formas distintas. Estas son:

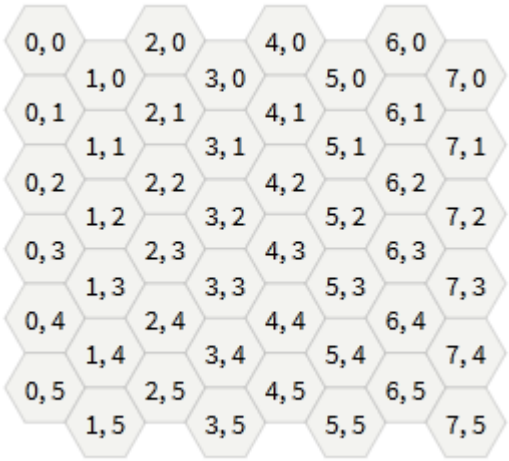
1. Parte superior en punta, filas pares desplazadas.



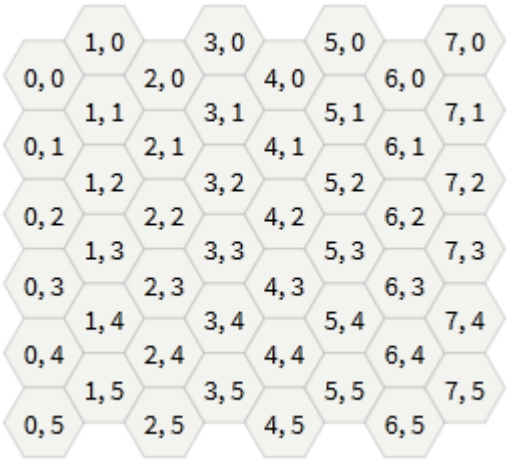
2. Parte superior en punta, filas impares desplazadas.



3. Parte superior plana, columnas pares desplazadas.



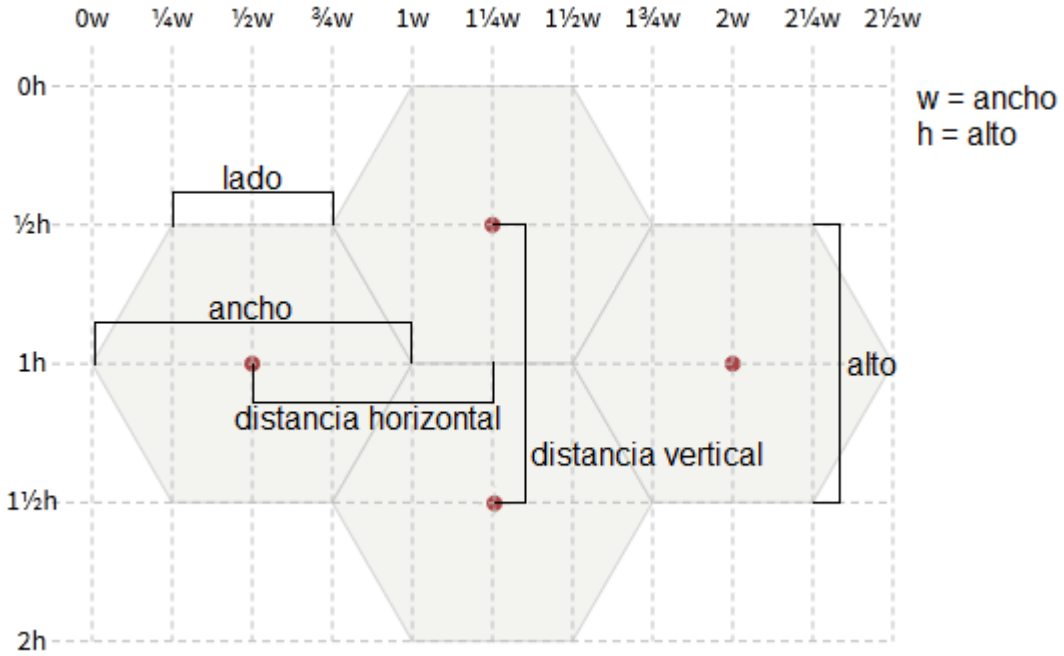
4. Parte superior plana, columnas impares desplazadas.



Para los mapas de la aplicación, se ha optado por la cuarta opción. De esta elección dependerá el posicionamiento (el cual se explica en el siguiente apartado) que es una decisión importante para el algoritmo de **generación del mapa**.

2.1.4 Posicionamiento de los hexágonos

Posicionar varios hexágonos regulares juntos para formar el mapa requiere calcular el espacio necesario entre ellos tanto en horizontal como en vertical.



Analizando los datos de la imagen anterior, pueden calcularse las siguientes medidas en función del tamaño del lado que se desee:

Sea:

- W = ancho.
- H = alto.
- h = distancia horizontal entre hexágonos.
- v = distancia vertical entre hexágonos.
- L = lado.

Entonces:

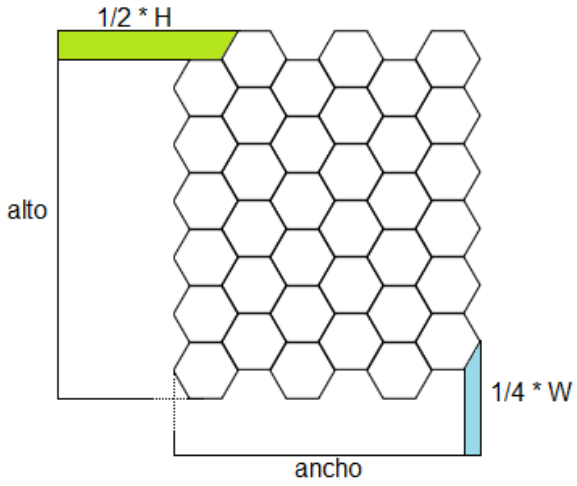
1. W = 2L.
2. H = $\sqrt{3}$ / 2 * W.
3. h = W * 3 / 4.
4. v = H.

2.1.5 Tamaño del *canvas*

El tamaño máximo del mapa en la visualización gráfica puede descomponerse en dos medidas: columnas y filas.

Para la simulación de la batalla y la creación de los mapas se ha establecido un tamaño máximo de 40 columnas por 20 filas para los combates. Por su parte la creación y modificación de formaciones utiliza la mitad de columnas y las mismas filas.

Con el fin de calcular el tamaño del elemento gráfico *canvas* (18) del DOM (19), se ha calculado el ancho y el alto en función de las filas y las columnas. Todos los mapas generados por la aplicación tendrán un tamaño par de filas y de columnas. Por ejemplo, para un mapa de 6 columnas y 6 filas:



Tomando como referencia los valores del **apartado anterior** para el ancho y el alto del hexágono, las medidas del *canvas* del DOM serán:

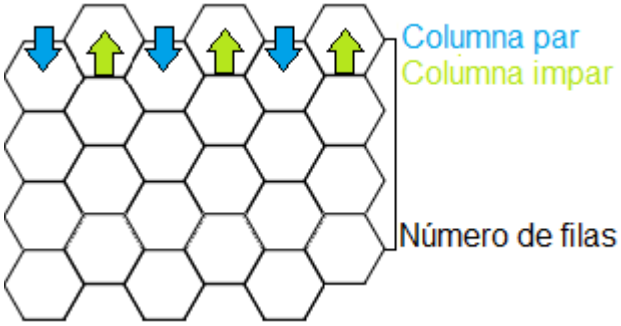
1. Alto: número de filas * H + 1 / 2 * H.
2. Ancho: número de columnas * (W + L) + 1 / 4 * W.

2.1.6 Creación de mapa

El algoritmo utiliza dos parámetros calculados en el apartado correspondiente al **posicionamiento de los hexágonos**:

1. La mitad del alto del hexágono.
2. Distancia horizontal.

En esencia, el funcionamiento se basa en desplazar la coordenada X la mitad del alto del hexágono hacia arriba o hacia abajo (en función de la paridad de la columna) y generar tantos hexágonos como filas se tengan en función de la distancia horizontal. Gráficamente el funcionamiento es el siguiente:



Se divide en dos etapas:

1. Para cada columna se desplaza el eje Y un medio del alto del hexágono dependiendo de si es par o impar y se multiplica el índice por la distancia horizontal para establecer el eje X.
2. Se generan tantos hexágonos como filas haya, multiplicando el índice por el alto para obtener la nueva coordenada Y de cada posición.

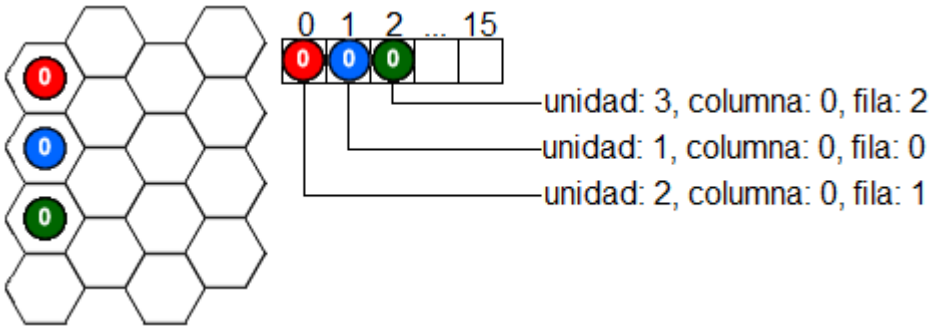
2.1.7 Procesamiento del mapa en el cliente

En el cliente, se utiliza el **algoritmo anterior** para generar los mapas según las dimensiones que se establezcan. Para mantener en todo momento la información del mapa y de las unidades que hay en cada una de las posiciones, se ha creado un vector con los siguientes datos:

1. Hexágono: referencia al objeto *sprite* del hexágono. Se trata de un objeto que retorna el *framework* gráfico el cual contiene entre otros: las coordenadas en el *canvas* y los eventos asociados. Se almacena para controlar las acciones del usuario en el **despliegue** de formaciones en el cliente.
2. Unidad: identificador de la unidad en la **capa de datos**.
3. Columna: columna que se corresponde con las coordenadas de la simulación
4. Fila: fila que se corresponde con las coordenadas de la simulación.

Al generar el mapa, a la vez se van guardando los *sprites* de los hexágonos, las coordenadas de la simulación y se inicializa el atributo unidad con un 0. Este vector se utiliza en la creación de formaciones: una vez que el usuario decida mandar la información al servidor, se filtran las posiciones mayores que 0 y se envía al servidor asíncronamente.

Por ejemplo, para un mapa de cuatro filas por cuatro columnas:



Las posiciones desde la 3 hasta la 15 conservan los valores de fila y columna pero el de unidad es 0.

2.1.8 Procesamiento del mapa en el servidor

En el servidor el mapa sólo es procesado a la hora de realizar la simulación de la batalla. Al haberse elegido el **sistema de coordenadas** basado en dos dimensiones, basta con crear un vector bidimensional con el tamaño de fila y columna máximo.

Como se van a tener distintos mapas, es posible que algunas posiciones no se encuentren disponibles. Estas posiciones no válidas se representan con un -1, el resto de posiciones libres serán 0. Es por ello que antes de cargar ninguna formación en la estructura, primero sea necesario cargar el mapa asociado.

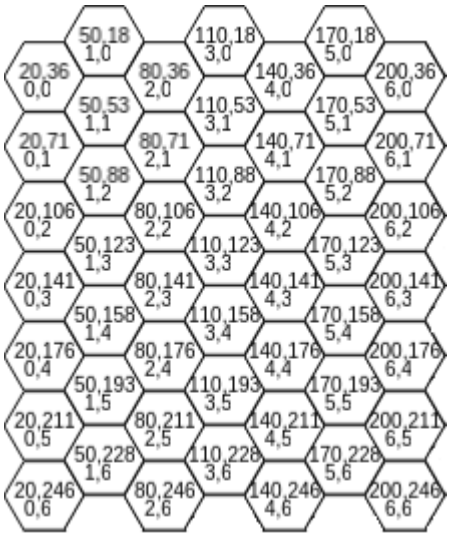
2.1.9 Transformación de coordenadas

Con todo lo anterior expuesto, hay que transformar las coordenadas que se utilizan en el servidor en las que utiliza el cliente.

Con la ayuda del **creador de mapas** y utilizando la **estructura** que procesa el mapa en cliente es posible realizar una asociación porque el objeto *sprite* que almacena, contiene las coordenadas del *canvas* que se necesitan y estas siempre tienen los mismos valores el todos los casos.

Para acceder a ellas cuando se necesiten basta con almacenarlas y para ello se tiene en la **base de datos** una tabla con toda esta información**.** Por último, para realizar estas transformaciones se ha utilizado una estructura bidimensional identificada por fila y columna que almacena estos valores. Cuando se desee realizar la conversión el acceso es directo.

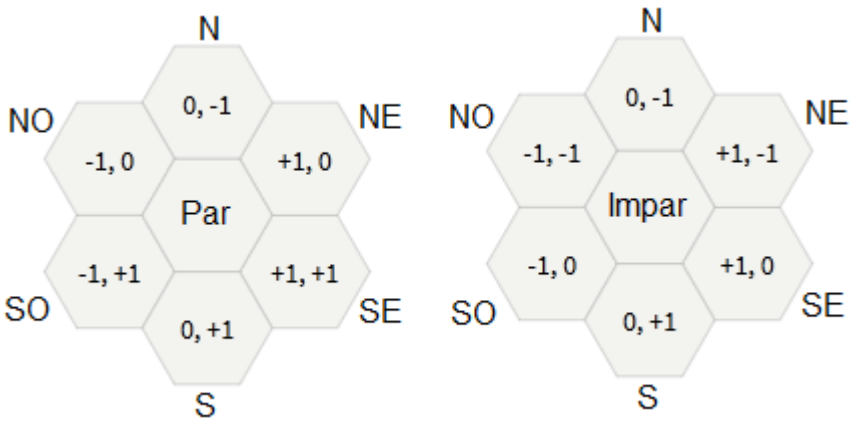
Un fragmento de un mapa generado con esta información es el siguiente:



Para cada posición del mapa se muestran las coordenadas de la vista en la parte superior y las de la simulación en la inferior.

2.1.10 Vecinos

Dependiendo de la paridad de la columna el desplazamiento utilizado en las diferentes direcciones pueden variar de la siguiente forma:



Para resolver el problema del cálculo de los vecinos en este tipo de mapas es necesario crear una estructura auxiliar que almacene las direcciones en las que se encuentran los vecinos. Basta con crear un vector de 6 enteros asociado a las direcciones. Para este trabajo se ha optado por un rotación comienza en la posición SE y recorre el resto en el sentido contrario a las agujas del reloj.

Por último se crea un vector con dos posiciones y una lista en cada una de ellas: una para las direcciones pares y otra para las impares. El orden de estas listas tendrá que estar en función del orden de búsqueda de vecinos que se desea seguir. Dada una fila y una columna el proceso consta de dos pasos:

1. Comprobar la paridad de la columna y acceder a la lista de posiciones de esta.
2. Para cada dirección, sumar o restar los valores.

Con este sistema se obtendrán todos los vecinos que se encuentren adyacentes a una posición dada. Si se quieren obtener todos los vecinos dado un rango, lo más sencillo es realizar las acciones anteriores **almacenando** todos los valores en una lista de elementos no repetidos y recorriéndola tantas veces como indique el rango.

2.1.11 Distancias y caminos

Para medir las distancias en un sistema de coordenadas se puede optar por dos métodos: distancia Manhattan (20), menos costosa y precisa o la distancia euclidiana (21). En este trabajo se ha realizado una versión de la primera ya que no es necesario tener una medida exacta.

La resolución del problema de los caminos no es trivial y se trataría de una opción demasiado pesada para mantener un tiempo de ejecución de la simulación razonable. Se ha optado por utilizar el método propuesto (17). La explicación que ofrece el sitio es muy detallada, se basa en el algoritmo DDA (22). En teoría no se trata de caminos sino de líneas entre posiciones.

2.2 Unidades

Las unidades en el juego serán las encargadas de llevar a cabo las batallas. Dependiendo de lo maximizado que tengan sus atributos, el mapa, la posición del resto de las unidades y un factor aleatorio de ataque o movimiento tendrán un comportamiento distinto.

2.2.1 Atributos

Cada unidad dispone de cuatro atributos:

1. Vida: cantidad de daño que puede recibir antes de morir.
2. Ataque: cantidad de daño que puede infligir.
3. Alcance: distancia a la que pueden atacar.
4. Defensa: cantidad de daño que pueden recibir sin perder vida.

Existen dos tipos de atributos, el común (todos los tipos pueden modificarlo) que es la vida y el propio de clase independiente para cada tipo que son el resto. Al crearse una unidad, tendrá el atributo común y propio inicializado a 0 y los demás a 1.

Será tarea del jugador la mejora de los atributos de todas las unidades (aunque puede no hacerlo si así lo desea) utilizando una serie de créditos que se podrán conseguir realizando combates. El común no tiene límite de mejora pero el propio del tipo tiene como máximo un valor de 40 puntos.

2.2.2 Tipos

Existen tres tipos:

1. Ataque cuerpo a cuerpo: su atributo propio de clase es el ataque. Sólo puede dañar a las unidades que se encuentren adyacentes.
2. Defensivas: su atributo propio de clase es la defensa. Sólo puede atacar a las unidades que se encuentren adyacentes.
3. Ataque a distancia: su atributo propio de clase es el alcance. Puede atacar a cualquier enemigo que se encuentre en su rango.

2.2.3 Nivel

Independientemente de los atributos que se modifiquen de cada unidad, cada 10 puntos de mejora utilizados (independientemente del tipo de atributo) sube de nivel. El nivel máximo que puede alcanzarse es el 20, por lo que es posible utilizar 200 puntos como máximo en cada una.

2.2.4 Reclutamiento

Para poder adquirir más unidades, es posible reclutarlas accediendo a la sección correspondiente de la aplicación. Independientemente del tipo de unidad que se desee reclutar, este tendrá el coste de un crédito. Una vez que el jugador disponga de unidades, estas se mostrarán en una serie de tablas (una por tipo) que muestra el nivel y el valor de los atributos de cada una.

Para seleccionar las unidades, se han creado una serie de tablas que muestran el nivel, la vida y el atributo propio de cada una. Al haber tres tipos de unidades, habrá tres tablas. Para seleccionar a la unidad basta con pulsar sobre la fila correspondiente de la tabla la cual se sombreará para indicar que ha sido seleccionada. También es posible seleccionar o deseleccionar todas las unidades disponibles.

Una vez seleccionadas las unidades, es posible eliminarlas también recuperando un crédito por cada una independientemente de los puntos de mejora empleados en ella.

Para almacenar las unidades seleccionadas en el cliente, se ha optado por una estructura secuencial que contendrá los siguientes datos: identificador de la capa de datos, nivel y tipo. Se han seleccionado estos tres datos de cada unidad porque a la hora de modificar y acceder a los atributos de esta se hace por identificador y a la hora de mostrar gráficamente las unidades en el mapa de formaciones, el nivel se muestra como número en cada una de las unidades, además dependiendo del tipo el color también cambia.

2.2.5 Atributos de batalla

Para la realización de las batallas entre formaciones ha sido necesario incluir, además de los atributos ya **comentados** otros de diverso tipo para acometer con mayor comodidad las acciones necesarias. Estos son:

- Camino: lista de posiciones que la unidad ha calculado.
- Índice de camino: posición del camino en la cual se encuentra la unidad.
- Alineamiento: indica si se trata de un aliado o un enemigo.
- Columna: columna en la que se encuentra la unidad
- Fila: fila en la que se encuentra la unidad.
- Tipo: tipo de la unidad.
- Nivel: nivel de la unidad.
- Etiqueta: identificador de la unidad para las animaciones de la vista.

2.2.6 Ataque

Cuando una unidad ataca a un enemigo se realiza una generación de un número entre 0 y 1. Para ello se tienen en cuenta dos factores:

1. Generación de un número aleatorio entre 0 y 0.5.
2. Obtención de un número en base al nivel: se divide el nivel actual de la unidad por el máximo y a su vez este resultado se divide entre 2. Con esto se obtendrá un número de cómo máximo 0.5.

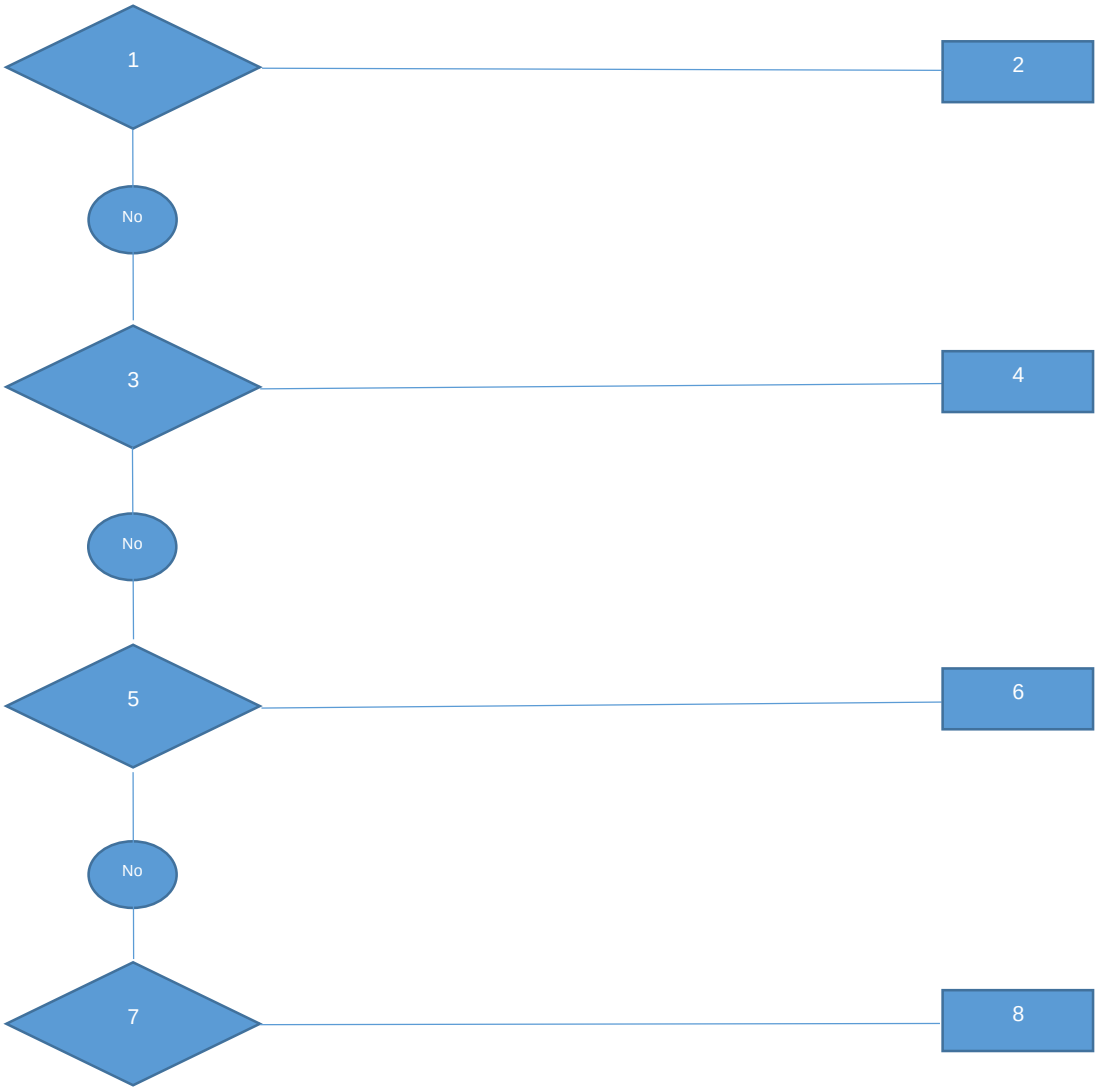
Por tanto cuanto mayor nivel tenga la unidad, mayor probabilidad de éxito tendrá a la hora de realizar el ataque. En caso de que la unidad atacante obtenga un resultado mayor que la atacada, se procede de la siguiente forma: si la que recibe el ataque tiene defensa mayor que 0, esta se disminuirá tanto como el ataque, en caso contrario el atributo que disminuirá será la vida. Si tras esto la vida de la unidad atacada es negativa, esta morirá y ya no formará parte del combate.

2.2.7 Algoritmo de movimiento aleatorio

En primer lugar se calculan las **posiciones adyacentes** de la fila y la columna de la unidad. A continuación se obtendrá una posición aleatoria de la lista de posiciones y si es una posición de movimiento válida (no está ocupada por ninguna otra unidad y forma parte del mapa) la unidad se moverá a la nueva posición. Esto se repetirá tantas veces como posiciones adyacentes se hayan obtenido. Si tras todos los intentos no se ha obtenido un resultado positivo, la unidad no se moverá.

2.2.8 Algoritmo de proceso de camino

Una vez se ha calculado el **camino** entre dos posiciones, este se guardará en una lista de la unidad. Llegado el momento la unidad tendrá que procesar esta lista de y para cada posición (si no se ha llegado al final) realizará lo siguiente:



1. En la posición hay un enemigo.
2. Se realiza un **ataque.**
3. La posición se encuentra vacía y es válida.
4. La unidad se mueve, el índice del camino aumenta.
5. La posición es inválida (no forma parte del mapa de la batalla).
6. Se anula el camino reseteando a los valores por defecto.
7. Hay un aliado en la posición del camino.
8. Se realiza un **movimiento aleatorio** pero se conserva el camino.

La situación descrita por el punto 5 se da porque los caminos se han calculado suponiendo que todas las posiciones del mapa se encuentran disponibles.

Las situaciones que pueden darse en la siguiente iteración si se ejecutan las acciones descritas por el punto 8 son las siguientes:

- La posición del camino está ocupada por un enemigo por lo que se realiza un ataque sea cual sea el tipo de unidad y la distancia a la que se encuentre.
- La posición del camino se encuentra libre, por lo que se realiza un salto hacia ella sea cual sea la distancia a la que se encuentre.

Con esta decisión aleatoria se hace que las simulaciones tengan menos determinismo y que el movimiento de las unidades cambie en cada ejecución. En caso de llegar al final se volverá a inicializar el camino.

2.2.9 Unidad más cercana

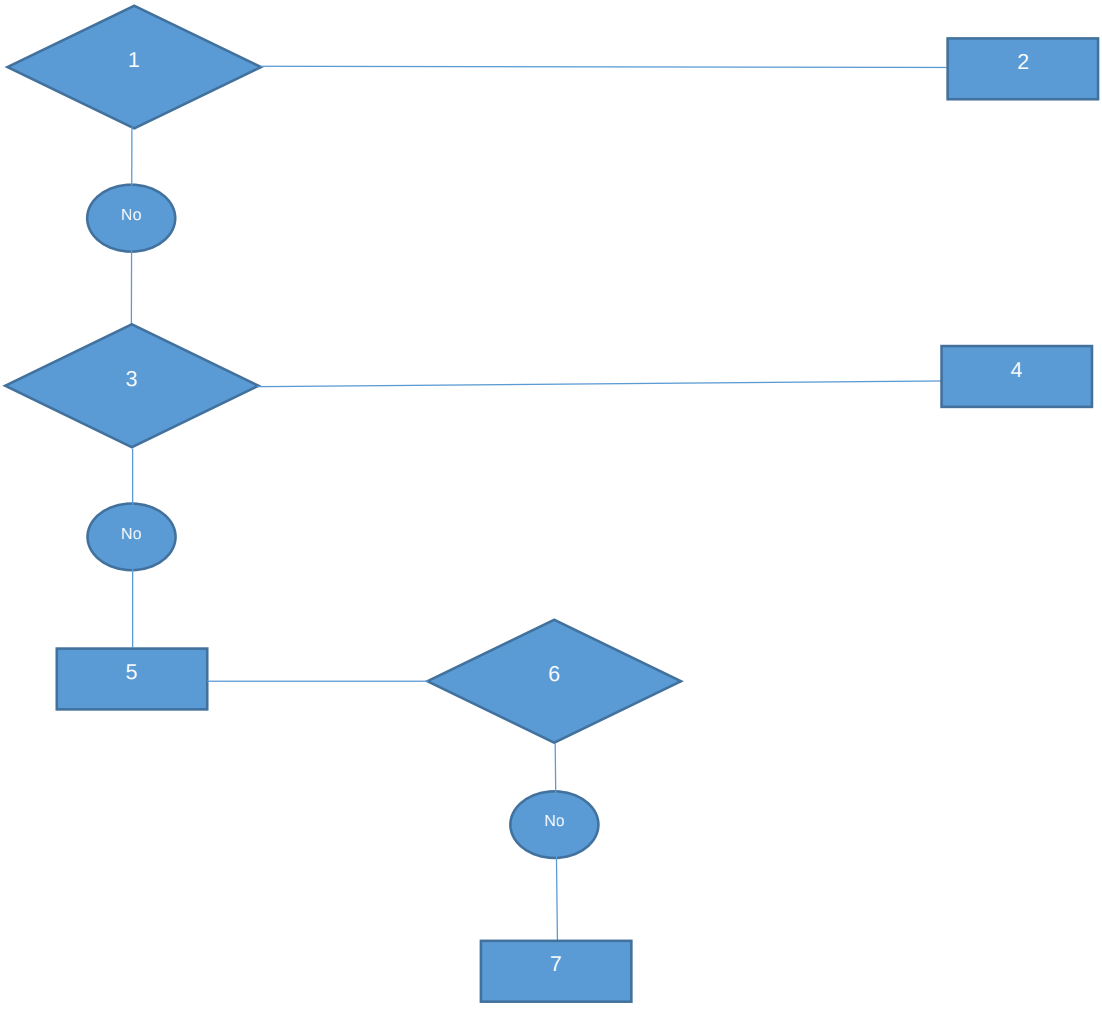
Se ha establecido un rango de visión del mapa para cada unidad. Este es la mitad del número de columnas. En este algoritmo, también se establece el alineamiento de las unidades que se quieren buscar.

Para ello simplemente se calculan los **vecinos** con este rango y se calculan las **distancias**. Se retornará la posición de la unidad más cercana con el alineamiento buscado.

2.2.10 Algoritmo de ataque cuerpo a cuerpo

El comportamiento de este tipo de unidades se basa en buscar a los enemigos en rango de visión, avanzar hacia ellos y atacar hasta eliminarlos. Como la unidad no puede mejorar sus defensas, es recomendable aumentar la vida lo máximo posible (con un balance adecuado para el ataque) para que así pueda soportar más daño y permanecer en el combate más tiempo.

Es una buena idea posicionarlas siempre lo más adelantadas posibles en la formación para que tarden lo menos posible en alcanzar al objetivo. Su algoritmo es el siguiente:

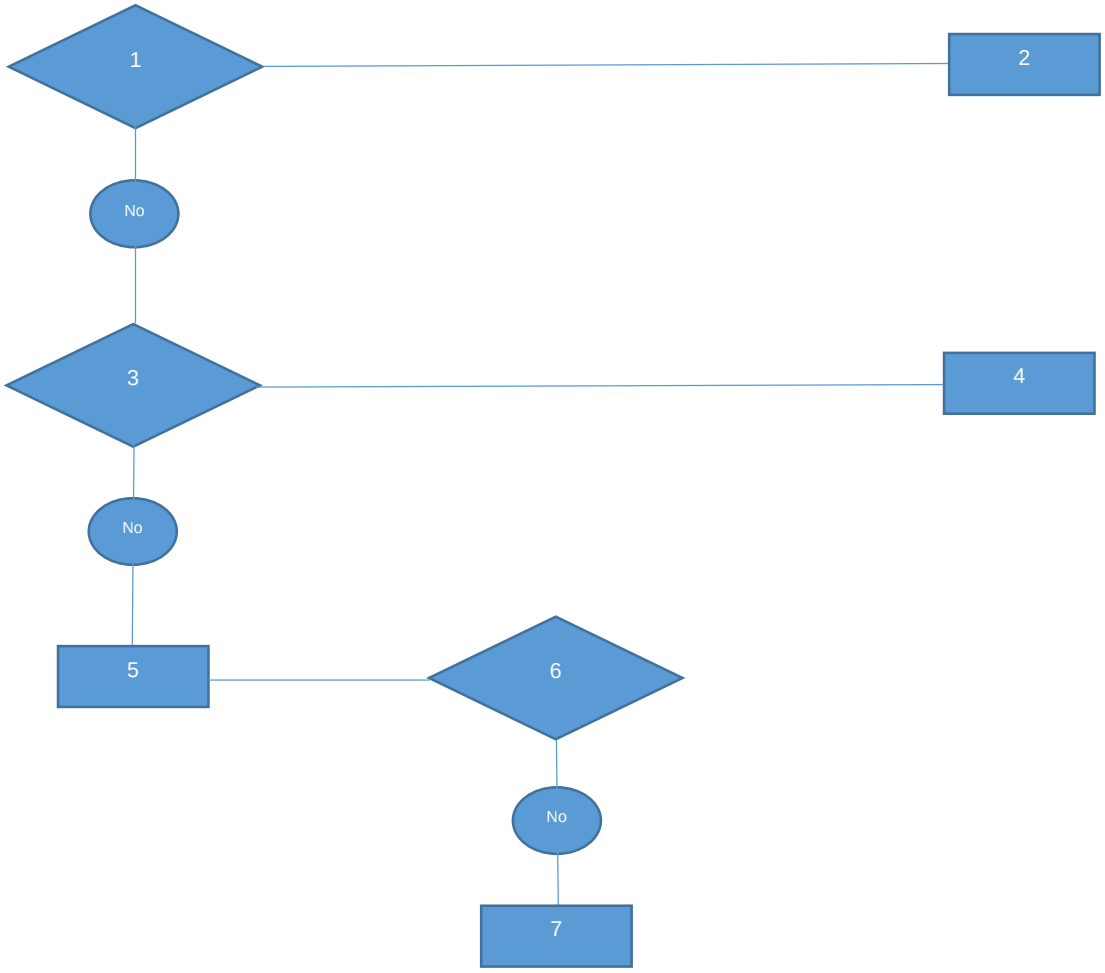


1. Hay un enemigo adyacente.
2. Realiza un **ataque**.
3. La unidad tiene un camino calculado.
4. Procesa la **siguiente posición** del camino.
5. Calcula el camino hacia el enemigo más cercano.
6. El camino se ha calculado de forma exitosa.
7. Se mueve hacia una **posición aleatoria**.

En caso de no encontrar un camino hacia una unidad enemiga cercana en el rango de visión, la unidad no realizará ninguna acción en ese turno.

2.2.11 Algoritmo defensivo

Las unidades defensivas realizan una labor de protección contra ataques a los aliados posicionándose en lugares próximos para atraer a los enemigos y absorber todo el daño posible mientras el resto de aliados ataca. Realiza una buena combinación con las unidades de ataque a distancia. Su comportamiento es el siguiente:

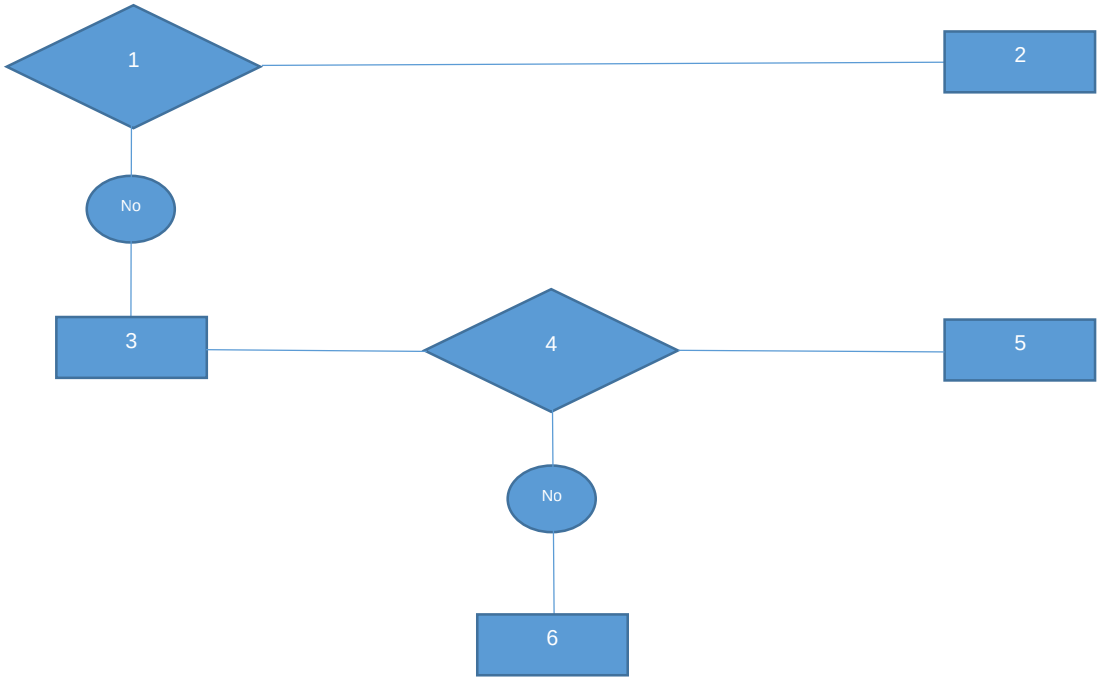


1. Hay un enemigo adyacente.
2. Realiza un **ataque**.
3. La unidad tiene un camino calculado.
4. Procesa la **siguiente posición** del camino.
5. Calcula el camino hacia el aliado más cercano.
6. El camino se ha calculado de forma exitosa.
7. Se mueve hacia una **posición aleatoria**.

En caso de no encontrar un camino hacia una unidad enemiga cercana en el rango de visión, la unidad no realizará ninguna acción en ese turno.

2.2.12 Algoritmo de ataque a distancia

Las unidades de ataque a distancia, como su propio nombre indica, buscan al más cercano de entre todos los enemigos alcanzables y le atacan. No utiliza el algoritmo de camino hacia enemigos, por lo que es recomendable maximizar su alcance y rodearlas de unidades defensivas siempre que se pueda. Las acciones que realiza son las siguientes:



1. Hay un enemigo adyacente.
2. Se mueve hacia una **posición aleatoria**.
3. Busca un enemigo que esté al alcance.
4. Ha encontrado un objetivo.
5. Realiza un **ataque**.
6. Se mueve hacia una **posición aleatoria**.

2.3 Formaciones

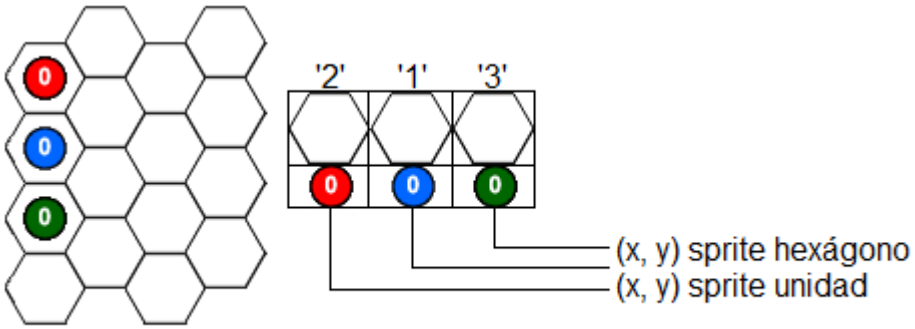
En el contexto de esta aplicación, se define formación como manera de posicionar las unidades en el mapa. Cada jugador podrá almacenarlas y modificarlas en la manera que así lo desee.

2.3.1 Despliegue de unidades

Las unidades del jugador siempre se van a posicionar en la mitad izquierda del mapa, por lo que podrá utilizar tantas unidades como posiciones disponibles tenga. Para otorgar a la aplicación de una mayor interactividad, se ha optado por desarrollar un sistema gráfico de *drag and drop* (23).

Una vez se han seleccionado las unidades de la misma forma que en el apartado de **reclutamiento**, es posible desplegarlas gráficamente en la parte que le corresponde al jugador en el mapa. Consideraremos unidad desplegada a mostrar gráficamente el *sprite* asociado en el mapa.

Para esta labor se ha creado un vector asociativo que contiene una referencia al hexágono que se ha creado en la **generación del mapa** y una referencia al *sprite* de la unidad. Las unidades se seleccionan en función de su identificador el cual se almacena tanto en el vector anterior como en el de la selección de unidades utilizado en el reclutamiento.



Para gestionar correctamente esta funcionalidad, hay que resolver dos problemas:

1. Identificar unidades desplegadas: al crear formaciones, es posible volver al menú textual en forma de tablas para modificar las unidades seleccionadas, esto conlleva el borrado de las unidades del entorno gráfico y la creación o reutilización de *sprites*.
2. Eventos del ratón: desde el entorno gráfico es posible realizar dos acciones: arrastrar la unidad para moverla a una posición concreta y pulsar sobre ella para modificar sus atributos. Para diferenciar estas acciones es necesario conocer las coordenadas tanto del hexágono de la posición como de la unidad.
3. Cambio de mapa: es posible cargar distintos mapas, por lo que la interfaz gráfica deberá variar dependiendo de las posiciones válidas.

Para resolver el primer problema se realizan dos filtrados: uno buscando unidades en la estructura del mapa que no se encuentren en las seleccionadas (lo que conllevará que el *sprite* de la unidad deje de mostrarse) y otra buscando las unidades seleccionadas que no se encuentren en el mapa para crearlas o, si ya se crearon anteriormente se muestren. Con esto se gana en eficiencia ya que sólo se crean las unidades una sola vez y se reutiliza el código anterior desarrollado.

Para revolver el segundo problema, se realiza una comparación entre las coordenadas de ambos *sprites*: si coinciden entonces el evento ha sido una pulsación de ratón por lo que se realiza la petición correspondiente al servidor y si son diferentes es que se ha realizado un evento *drag and drop* que puede tener las siguientes alternativas dependiendo del estado de la posición destino:

- Se encuentra ocupada por una unidad: se intercambian las posiciones mostrando una animación y se actualiza la información de las posiciones del mapa.
- No existe en el mapa: se muestra una animación haciendo que la unidad vuelva a su posición origen.
- Se trata de la zona de descarte: se elimina la unidad del entorno gráfico, de la estructura que almacena el mapa y se modifica su fila de la tabla para que no muestre el sombreado de selección.

Por último la carga de mapas realiza una petición al servidor que devuelve una lista de posiciones con el formato de coordenadas **ya explicado**. Se realiza un filtrado y se ocultan los hexágonos que no formen parte del mapa. Con este sistema en caso de cargar varios mapas siempre se reutilizará la misma estructura.

2.3.2 Modificación

Para modificar una formación se realiza una petición al servidor la cual retorna todas las unidades que la forman. Se procesan los datos recibidos y se procede de la misma forma que en el apartado anterior reutilizando el trabajo y las estructuras ya creadas.

2.4 Batalla

Las batallas entre formaciones utilizan todo lo expuesto anteriormente. Se da la oportunidad al usuario de establecer una serie de parámetros iniciales:

- Multiplicador de enemigos: cantidad de enemigos que se desea tener en el bando contrario. Esto se multiplicará por el tamaño total de la formación del aliado para generar un número de enemigos normalizado entre 1 y el tamaño máximo de la mitad del mapa.
- Tipo de enemigos: el tipo de unidades contra las que se desea combatir.
- Nivel máximo: el nivel máximo que van a tener los enemigos generados.
- Modo espejo: se combatirá contra las mismas unidades pero en posiciones opuestas. Si se selecciona este modo, los anteriores parámetros se ignorarán.

2.4.1 Generación aleatoria de enemigos

Si no se selecciona el modo espejo, se generarán aleatoriamente una serie de valores para cada unidad enemiga:

- Posición: cualquier posición que sea válida y se encuentre libre en el lado que corresponde a los enemigos (mitad derecha del mapa).
- Tipo: un tipo aleatorio de entre los que se ha elegido al inicio de la simulación.
- Nivel: un nivel aleatorio desde el mínimo hasta el máximo seleccionado.
- Atributos: los atributos se obtienen en función del nivel, siendo los puntos de mejora que se van a utilizar el resultado de multiplicar el nivel por 10 y sumarle 9. Tras esto se generarán los atributos de acuerdo con el tipo y el valor máximo.

2.4.2 Procesamiento de la batalla

Para cada iteración se recuperan de la matriz en donde se almacena el mapa (explicada en **este apartado**) las posiciones en donde hay unidades y se cuentan por un lado las unidades enemigas y por otro las aliadas. A continuación, para cada una de las posiciones calculadas se realizan las acciones correspondientes y se guardan los resultados en una estructura **transformando** las coordenadas a las del *canvas* en el cliente. Cada vez que las unidades realicen sus acciones y no se haya dado la condición de fin, el turno avanzará.

La batalla no terminará a menos transcurran dos minutos desde su inicio o que alguno de los bandos no tenga más unidades. Al terminar, el usuario gana un crédito por cada unidad enemiga eliminada y 50 más en caso de victoria.

2.4.3 Estructura del resultado

Tras terminar la batalla, se envía al cliente una estructura de tipo JSON (24) con los siguientes apartados:

- Inicio: las posiciones iniciales de las unidades, su nivel, su tipo y una etiqueta por la cual van a ser identificadas en el entorno gráfico.
- Resultado: las unidades que ha eliminado, las que ha perdido y si ha ganado o no.
- Los créditos que ha ganado el usuario.
- El mensaje con el error en caso de fallo.
- El total de turnos.
- Las acciones que ha realizado cada unidad por turno.

Este último parámetro se trata de un vector asociativo identificado por el turno el cual alberga otro identificado por las etiquetas mencionadas anteriormente. Dentro de cada una de ellas se encuentra la acción que ha realizado esta unidad, que pueden ser:

- Las coordenadas de destino del movimiento.
- Las coordenadas de destino de un ataque exitoso.
- Las coordenadas de destino de un ataque fallido.
- -1 si la unidad ha sido eliminada.

2.5 Usuario

Las tareas que puede realizar el usuario con respecto a sus credenciales y cuentas son las que se cabría esperar: registro, identificación, modificación de parámetros y borrado.

2.5.1 Notificaciones por correo electrónico

Se ha incluido una funcionalidad que en caso de haber utilizado una cuenta de correo válida (es posible registrarse con cualquier dirección de correo) manda una serie de avisos en las siguientes situaciones: registro, desactivación de la cuenta, activación de la cuenta y borrado del perfil. Este método se encuentra en el núcleo de Django.

2.5.2 Activar o eliminar usuario

Para aprovechar la misma petición POST (25) se ha incluido un campo oculto en la vista de identificación que dado el caso en el que el usuario haya desactivado la cuenta y quiera volver a activarla, se incluya en la petición para indicar al servidor este hecho.

2.6 Gráficos

Una vez realizada la simulación, una parte imprescindible de la aplicación es la visualización de las batallas realizadas. Para ello se han realizado una serie de optimizaciones del motor gráfico para poder asegurar una tasa de refresco lo más alta posible y un funcionamiento que represente fielmente lo sucedido en la batalla.

2.6.1 Carga del mapa

Como no es necesario tener una interacción con los hexágonos del mapa en esta sección (no siendo igual en la **gestión de formaciones**) es innecesario además de ineficiente la carga de tantos hexágonos como posiciones tenga el mapa.

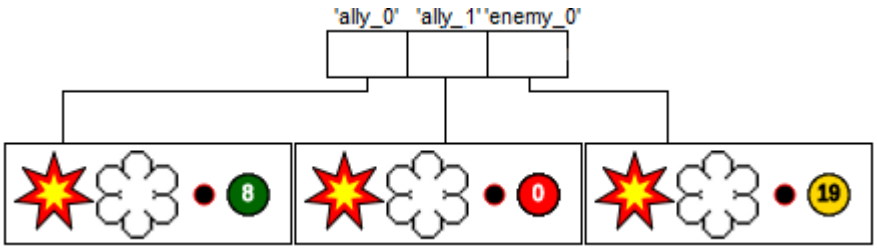
Es mucho más eficiente cargar en una sola imagen de fondo del *canvas* del mapa para así, en vez de procesar como máximo 800 objetos, se procese sólo uno. No existirá ningún problema con las coordenadas ya que estas imágenes han sido generadas con el propio **creador de mapas,** por lo que las dimensiones serán siempre las mismas.

2.6.2 Carga y almacenamiento de los *sprites*

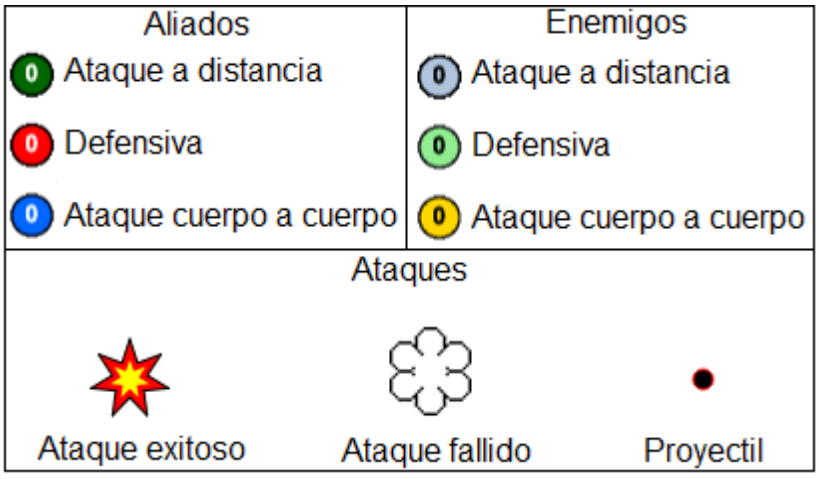
Para poder asegurar que dado cualquier número de unidades en pantalla estas puedan realizar al mismo tiempo y correctamente todas las animaciones asociadas a sus acciones, se ha creado un vector asociativo por identificador de unidad. Estos identificadores se extraen de la **estructura** que retorna el resultado de la batalla desde el servidor.

Para cada unidad aliada, se le asigna una cadena del tipo <*ally_x*> siendo "x" un número entre 0 y el total. Por su parte, las unidades enemigas también tendrán asignada una etiqueta del tipo <*enemy_y*> siendo "y" un número entre 0 y el total.

En cada celda del vector se almacenan todos los *sprites* que va a utilizar la unidad, estos son: unidad, proyectil, ataque exitoso y ataque fallido.



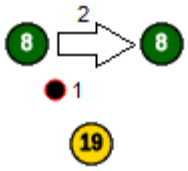
Por tanto cuando una unidad requiera de un *sprite* concreto para realizar una animación, accederá por clave a su posición en la estructura. Al ser de tipo clave – valor, este acceso tiene un coste constante. Los tipos de *sprites* utilizados son los siguientes:



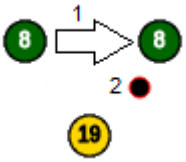
2.6.3 Sincronización de animaciones

Como todas las animaciones se realizan en cada turno al mismo tiempo y tienen la misma duración en la mayoría de los casos no se sincronizan porque unas dependen de otras. Pueden darse dos casos:

1. La unidad recibe un ataque al inicio del movimiento.



2. La unidad recibe un ataque al final del movimiento.



En el primer caso, será necesario introducir una espera igual que la velocidad de la animación para que la duración sea el doble y así la unidad realice la animación de movimiento después de recibir el ataque. En el segundo caso ocurre justo al contrario, será la animación de ataque la que recibirá el retardo para realizar la acción justo cuando se termine el movimiento.

Debido a esto es necesario tener un control de qué animaciones se van a mostrar por turno, por ello se han creado dos listas: una con movimientos y otra con ataques. Por tanto cuando se realice una animación de movimiento se buscará en la lista de ataques uno que tenga como destino la posición de inicio, en caso afirmativo se añadirá el retardo a la animación. En el caso de los ataques si existe un movimiento cuya posición de destino coincida con el ataque, se introducirá este retardo.

Para las muertes también se utiliza una lista la cual se procesa una vez finalizadas las demás acciones. Cada animación tiene un retardo igual a la velocidad de las animaciones, que es el máximo que puede durar cualquiera de ellas.

2.6.4 Animaciones

Las animaciones asociadas a las acciones de las unidades utilizan una técnica llamada *tweening* (26) que ya se encuentra implementada en Phaser. Este tipo de animaciones son poco costosas y la visualización de las batallas utiliza sólo tres por lo que se ha elegido esta alternativa, siendo innecesaria la utilización de un *spritesheet* (27) o técnicas similares. Tipos:

- Movimiento: posiciona el *sprite* de la unidad en la nueva posición.
- Ataque: esta animación se compone de dos partes, en la primera carga y mueve el proyectil al destino. En la segunda, dependiendo de si el ataque es exitoso o fallido, carga el *sprite* correspondiente.
- Muerte: difumina la unidad hasta que desaparece. Una vez termina elimina los *sprites* asociados de la simulación.

2.6.5 Procesamiento del combate

Una vez recibido el resultado del combate, se almacena en una estructura auxiliar, se cargan todas las unidades con sus correspondientes *sprites* de la forma descrita **anteriormente** y se muestra la interfaz.

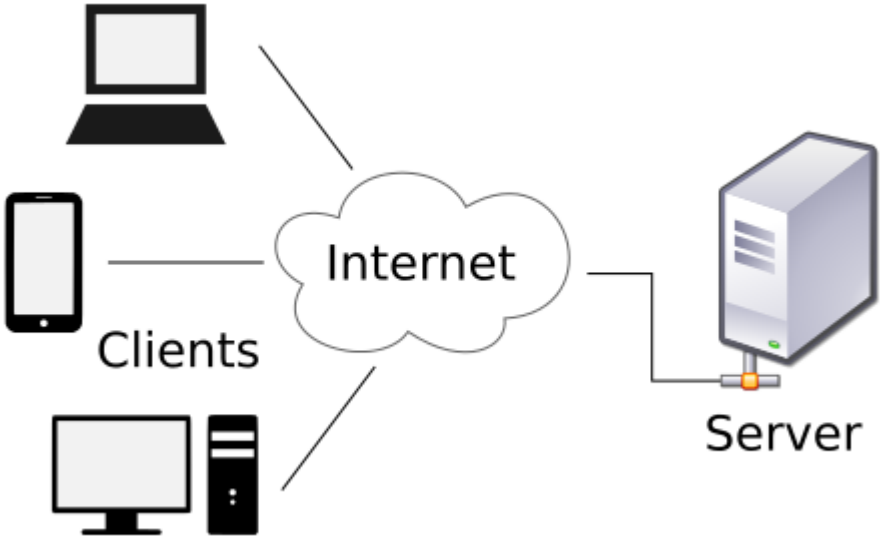
Una vez iniciada la batalla, para cada turno se procesa el resultado y se van guardando en las listas mencionadas en el apartado de **sincronización** de animaciones las acciones de las unidades. Una vez se ha almacenado la acción de la última unidad del turno, se procesan todas las animaciones y se resetean las listas.

El método que ejecuta la simulación repetidamente utiliza un evento de Javascript que repite la ejecución automáticamente transcurrido un tiempo (28). Este tiempo es el doble de la velocidad de animación por lo que para cada turno se asegura que todas las animaciones han terminado.

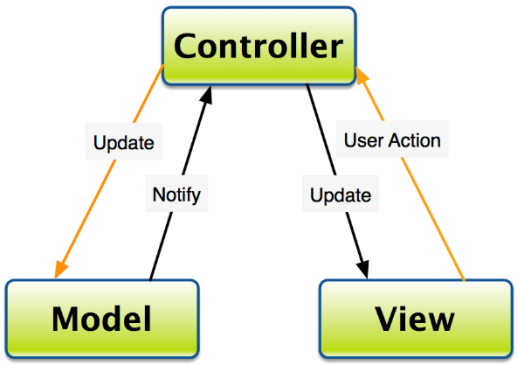
Capítulo 3: Arquitectura del sistema

3.1 Arquitectura software utilizada

Al tratarse de una aplicación web, esta utiliza una arquitectura del tipo cliente – servidor (29) la cual a grandes rasgos consiste en que el cliente realiza peticiones al servidor el cual responde a estas. Aunque esta arquitectura comúnmente es asociada a este tipo de aplicaciones, en la práctica puede aplicarse a cualquier tipo sea cual sea su naturaleza.



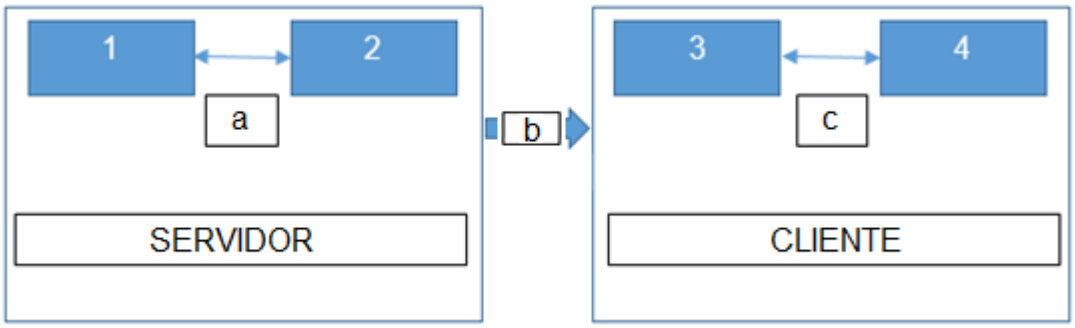
Un tipo concreto de este tipo de arquitectura es la llamada programación por capas (30), la cual separa la lógica del negocio con la de la presentación, disminuyendo el acoplamiento (31) y haciendo que el mantenimiento de la aplicación resulte más eficiente. En concreto se utiliza la arquitectura modelo – vista – controlador (MVC) (32). La parte del cliente es el *tipo single – page – application* (33) .



3.2 Servidor

Para la implementación de la parte servidor, como ya se comentó en la **introducción** se ha seleccionado el *framework* Django el cual aunque sigue la filosofía MVC tiene una particularidad: los controladores se denominan vistas y las vistas plantillas. Tiene como filosofía los principios de la reutilización, el desarrollo ágil y más que ajustarse a un tipo de arquitectura, su objetivo es ser eficiente y rápido.

El funcionamiento general de la aplicación en torno a las batallas utiliza el siguiente esquema:



1. **Procesamiento de la batalla**.
2. **Transformación de coordenadas**.
3. **Procesamiento del resultado**.
4. Animación de las acciones de la batalla **sincronizadas**.

a. Conversión para cada acción en cada turno.
b. Retorno del resultado de batalla con el **formato establecido**.
c. Se procesan todas las acciones.

3.2.1 Estructura de una aplicación en Django

Las aplicaciones desarrolladas con Django tienen una forma concreta de distribuir los ficheros y una serie de buenas prácticas para desarrollar un software de más calidad. El esquema que utiliza es el siguiente: un proyecto se compone a su vez de varias aplicaciones, cada una de estas realiza una función concreta y tiene su propio sistema de directorios. Dentro de la carpeta raíz existe un fichero de configuración en el cual se establecen, entre otros, qué aplicaciones se utilizan. Dentro de cada aplicación existen una serie de directorios y ficheros fijos:

- Directorios:
  - Plantillas (*templates*): dentro de este directorio se almacenarán los ficheros HTML.
  - Estático (*static*): dentro de este directorio se almacenarán los ficheros Javascript, CSS, las imágenes y cualquier otro recurso.
- Ficheros:
  - Administración (*admin.py*): su implementación es opcional, en este fichero se guarda la lógica que va a seguir la vista de administración del sitio en caso de utilizarla.
  - Modelos *(models.py*): fichero del modelo de datos, contiene las clases y las relaciones de claves externas entre ellos. El ORM mapeará los atributos de las clases, los transformará a los tipos de la capa de datos y añadirá las relaciones de claves externas.
  - Vistas (*views.py*): contiene los controladores de las vistas. Cada uno de ellos realiza la lógica asociada a una acción concreta y presenta los datos en las plantillas.
  - Direcciones (*urls.py*): asocia las direcciones de la aplicación a los controladores del fichero de vistas.
  - Pruebas (*test.py*): fichero que implementa las pruebas automatizadas.

3.2.2 Ficheros del trabajo

Para la implementación de este trabajo se han descompuesto en módulos las acciones y establecido los datos constantes a utilizar por cada una de las funcionalidades. Después, se han dividido en ficheros dentro del directorio de la aplicación.

En el servidor:

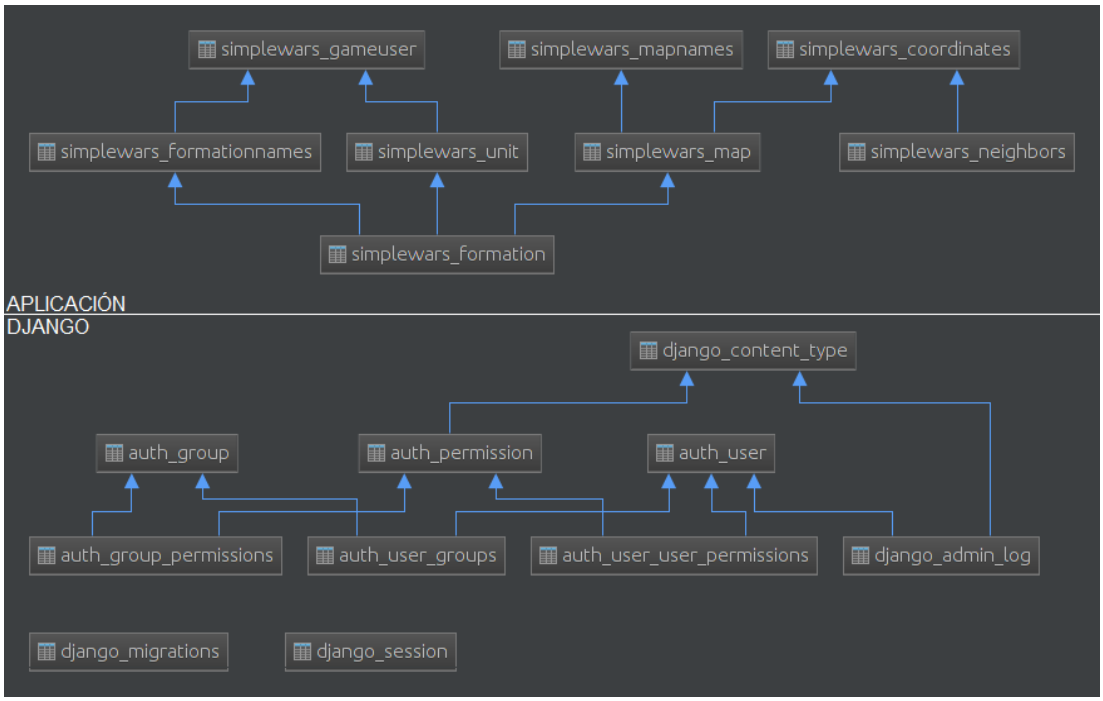
- Simulación (*simulation.py*): algoritmos utilizados en las batallas.
- Formularios (*forms.py*): formularios utilizados en los controladores.
- Constantes (*typification.py*): mensajes de error, direcciones de la aplicación, estilos de campos de los formularios, valores de la simulación… Son utilizados en toda la aplicación.
- Funciones de la vista (*util.py*): acciones que se llevan a cabo desde los controladores.

En el cliente:

- Formaciones (*formations.js*): implementación gráfica de la creación y modificación de formaciones.
- Batallas (*battle.js*): implementación de la visualización de las batallas.
- Creación de mapa (*map_creator.js*): implementación de la creación gráfica de mapas.
- Interfaz (*functions.js*): cambios en el DOM y peticiones al servidor.

3.2.3 Base de datos

El motor de base de datos que se ha utilizado, como ya se comentó en la **introducción** es MySQL. El esquema de la capa de datos se encuentra reflejado en la siguiente imagen:



Django genera su esquema propio cada vez que se crea una aplicación. En él recoge toda la gestión de usuarios, autenticación, grupos, seguridad, migraciones…

Todas las consultas utilizan sólo y exclusivamente los métodos que ofrece el ORM de Django (aunque es posible realizar consultas SQL propias) el cual:

1. Planifica y cachea las consultas para realizarlas de forma eficiente.
2. Es resistente a ataques de inyección SQL (34).

Otro aspecto que se ha tenido en cuenta es la eficiencia de la simulación y sus algoritmos asociados: el cálculo de **vecinos por rango** es un proceso costoso y los resultados siempre van a ser los mismos, por lo que se ha creado una tabla con todos los rangos ya generados para cada posición del mapa. Como el tamaño de los campos puede sobrepasar el tamaño máximo permitido (35), se ha modificado expresamente la tabla para evitar errores.

La velocidad de carga de la aplicación web es un aspecto a tener en cuenta, por ello se ha activado la caché (36) que ofrece MySQL la cual guarda un registro de las consultas realizadas en memoria: en caso de necesitar datos que ya se han leído previamente, el motor los retorna con menor coste.

También se han tenido en cuenta las peticiones que se realizan al servidor y la información que se muestra al usuario en las vistas:

1. En la vista del listado de formaciones, sólo se necesita el nombre de esta, el mapa asociado y el número total de unidades.
2. Cuando se crea una formación, el único dato necesario para el usuario es el nombre del mapa.
3. Se ha utilizado un paquete de Django (37) que analiza el rendimiento de cada vista.

Por último se han creado índices sobre los atributos que van a ser utilizados como filtro en las consultas. Estos son:

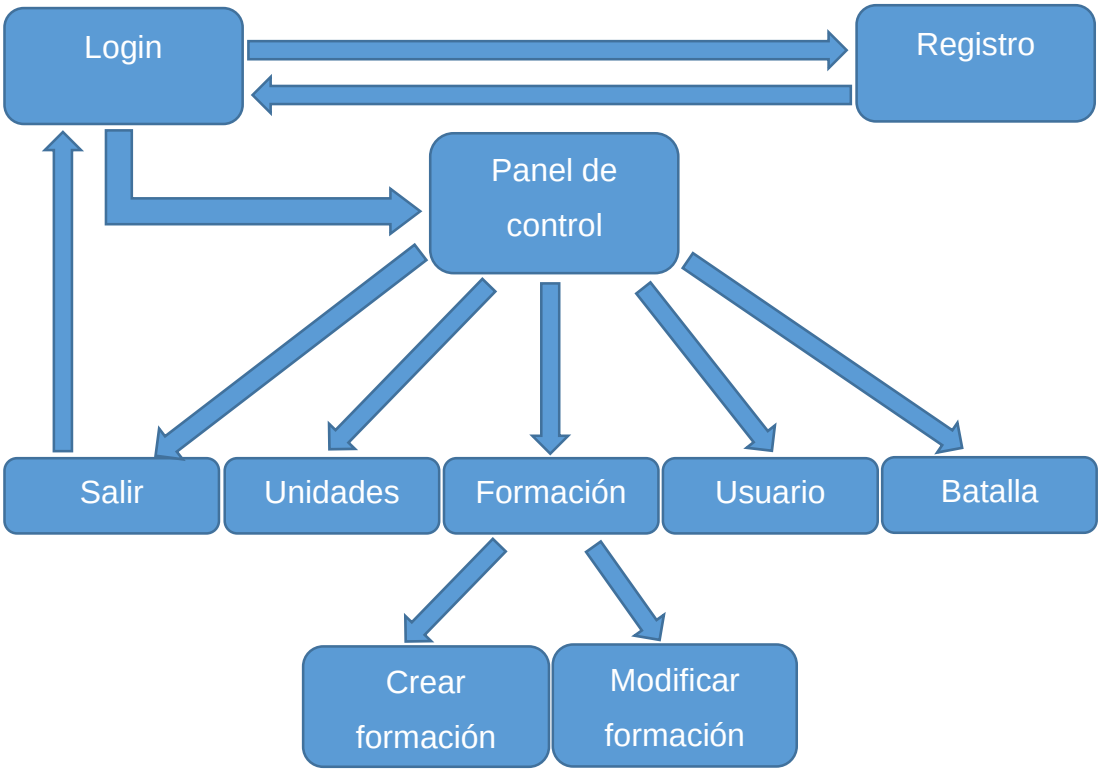
1. En la tabla de unidades, sobre los atributos tipo y el indicador de si la unidad se encuentra en formación.
2. En la tabla de coordenadas, sobre los atributos fila y columna de la simulación por una parte y las coordenadas de la vista por otra.

Las tablas creadas son las siguientes:

- *GameUser*: usuario del juego, contiene una clave externa uno a uno con respecto a la tabla de usuario de Django (*auth_user*):
    - Créditos: número total de créditos.
    - Unidades creadas.
    - Unidades borradas.
    - Unidades perdidas.
    - Victorias.
    - Derrotas.
    - Puntos de mejora utilizados en las unidades: suma total de todos los puntos de mejora que se han utilizado en las unidades.
- *Unit*: unidad, contiene una clave externa muchos a uno con respecto al usuario del juego:
    - Vida.
    - Ataque.
    - Alcance.
    - Tipo.
    - Defensa.
    - Puntos de mejora: puntos de mejora totales utilizados en la unidad.
    - Nivel.
    - En formación: bandera que indica si la unidad forma parte de una formación o no.
- *FormationNames*: nombres de formación, contiene una clave externa con respecto al usuario. Se ha creado esta tabla intermedia entre el usuario y las formaciones por eficiencia: la tabla de formaciones contiene un registro por unidad en formación por lo que puede alcanzar un tamaño considerable.
    - Nombre.
    - Total: unidades de la formación.
    - Mapa: mapa utilizado.

- *Formation*: registro de todas las unidades asociadas a formaciones. Contiene una clave externa a la tabla de nombres de formaciones, a la de unidades y a la de mapa. De esta forma no se tiene información repetida.
- *MapNames*: se ha considerado la creación de esta tabla por la misma razón que la de los nombres de las formaciones. Cada mapa del juego puede contener 800 posiciones y en caso de albergar varios, el número de registros puede alcanzar un nivel considerable.
    - Nombre del mapa.
    - Número total de posiciones.
- *Coordinates*: coordenadas de la simulación y de la vista generadas. Se almacena en la capa de datos para su carga en la estructura de **transformación de coordenadas** del mapa.
    - Fila: fila de la simulación
    - Columna: columna de la simulación.
    - Coordenada X: coordenada X del entorno gráfico.
    - Coordenada Y: coordenada Y del entorno gráfico.
- *Map*: posiciones del mapa, contiene una clave externa con respecto a la tabla de coordenadas y a la de nombres de mapa.
- *Neighbors*: cálculo de todos los vecinos por posición. Contiene una clave externa con respecto a la tabla de coordenadas. Para cada una de ellas, tiene en un campo una lista con todas las posiciones en las coordenadas de la simulación.

3.4 Mapa de la aplicación



- Login (“/login”): identificación del usuario. En esta sección los usuarios de la aplicación pueden introducir sus credenciales de acceso a la zona de usuarios.
- Registro (“/register”): registro del usuario. En esta sección los usuarios pueden registrarse en el sistema de la aplicación.
- Panel de control (“/dashboard”): ventana inicial de la zona de usuarios. Muestra los datos del usuario y los 5 primeros usuarios clasificados en función de las victorias.
- Unidades (“/army”): reclutamiento de unidades. Desde aquí se puede mejorar, reclutar y eliminar unidades.
- Formación (“/create-formation”, “/list-formation”): gestión de formaciones. Desde esta vista es posible crear y modificar formaciones de forma gráfica.
- Batalla (“/battle”): combates entre formaciones. El usuario puede modificar todos los parámetros de la simulación y elegir la formación que desea utilizar.

- Usuario (“/user”): gestión de los datos del usuario. En esta sección el usuario podrá cambiar la contraseña, el nombre y el correo electrónico. También tendrá las opciones de desactivar o eliminar la cuenta.
- Salir (“/logout”): sale de la zona de usuario y vuelve a la de identificación.

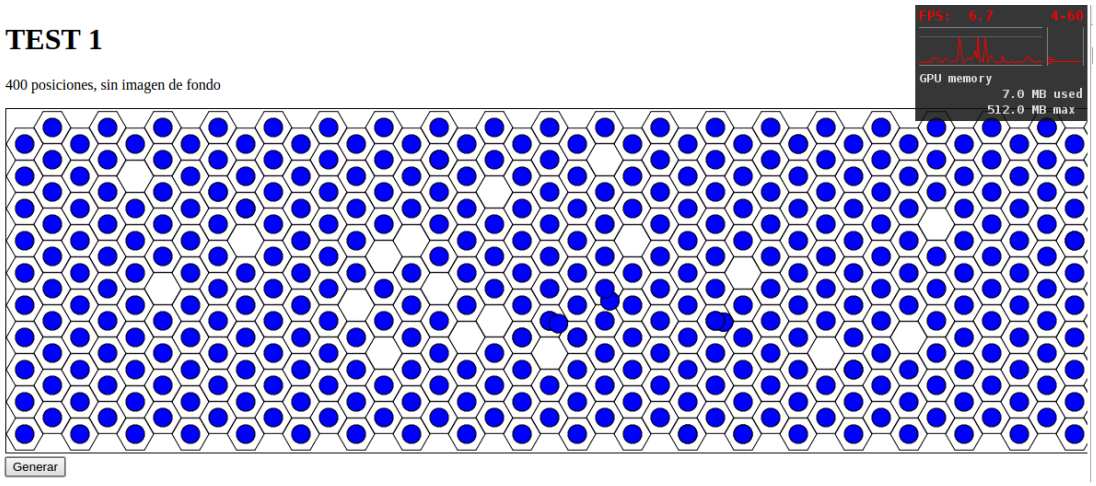
Capítulo 4: Pruebas

4.1 Gráficos

Para comprobar el rendimiento de las librerías que se han utilizado para mostrar gráficos en aplicaciones web, se ha implementado la siguiente prueba: dadas unas dimensiones de mapa, situar tantas unidades como el tamaño permita y cada vez que el puntero del ratón se posicione sobre una de ellas, realizar una animación de movimiento. El fin de esta prueba es medir las imágenes por segundo (38) que la librería puede manejar en el peor de los casos: todas las posiciones del mapa se encuentran ocupadas por unidades que disparan animaciones.

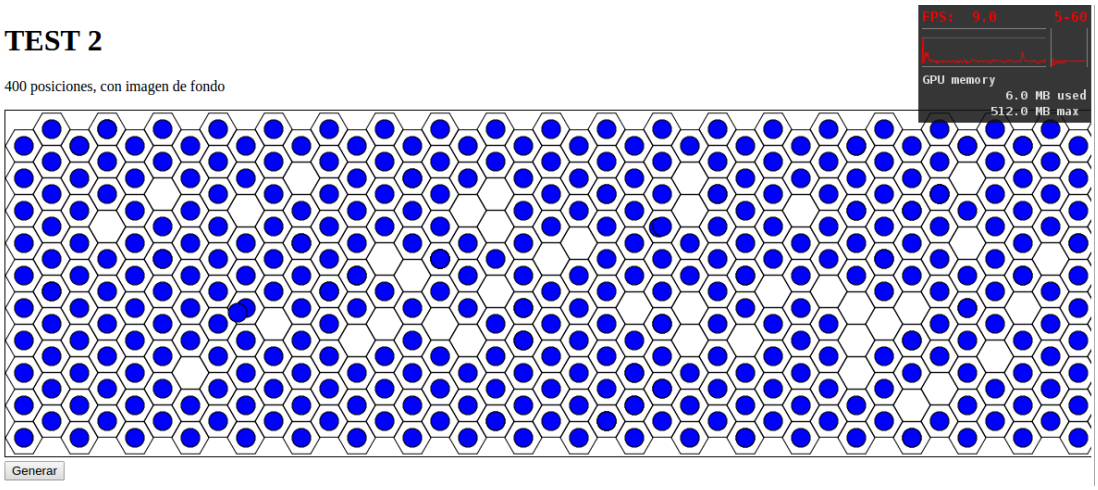
4.1.1 jCanvas

Para la primera se consideró un mapa de 400 posiciones sin imagen de fondo para los hexágonos, esto es, que la librería se encargará de mostrar todas las casillas en cada iteración. Aunque se trata de una aproximación ineficiente, resulta muy efectiva para medir el rendimiento. La meta es conseguir una tasa entorno a los 30, siendo 60 su valor ideal.



La tasa de refresco se sitúa entre 5 y 7 imágenes por segundo, el cual es un rango de valores insuficiente.

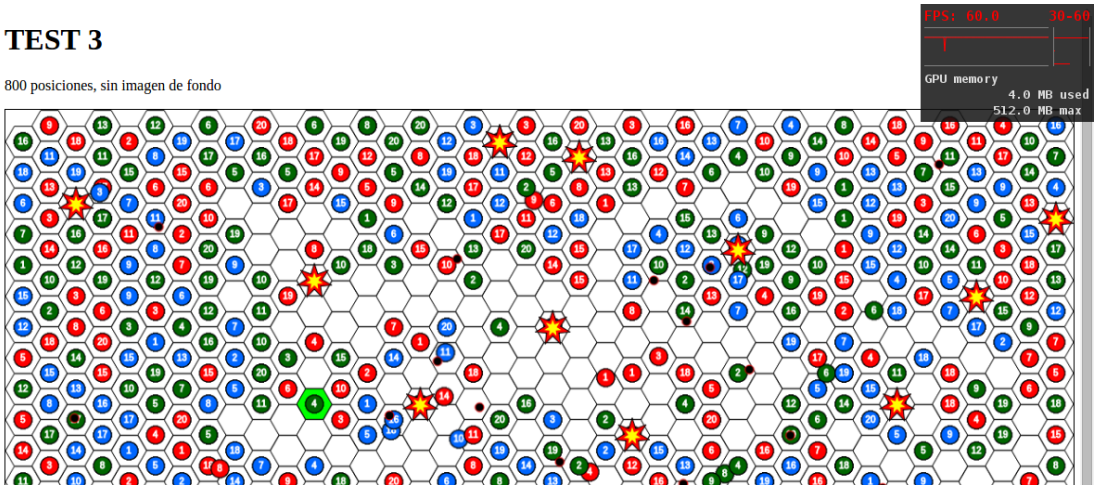
Para la segunda prueba, se utilizó una imagen de fondo para representar el mapa, de esta forma en lugar de procesar tantos hexágonos como posiciones, se procesaría sólo una imagen de fondo (las unidades seguirían siendo las mismas). En teoría el rendimiento debería de incrementar sustancialmente.



La tasa de refresco aunque mejora la anterior, se sitúa entre 10 y 12 imágenes por segundo, el cual es un rango de valores insuficiente. Como consecuencia de los resultados obtenidos, se abandonó el uso de esta librería para la visualización gráfica de los combates.

4.1.2 Phaser

Para probar esta librería, se creó un mapa capaz de albergar el doble de unidades que los anteriores y además no se utilizó imagen de fondo. Además se incluyó una animación de disparo y un resaltado en verde del hexágono sobre el que se encuentra el puntero del ratón.

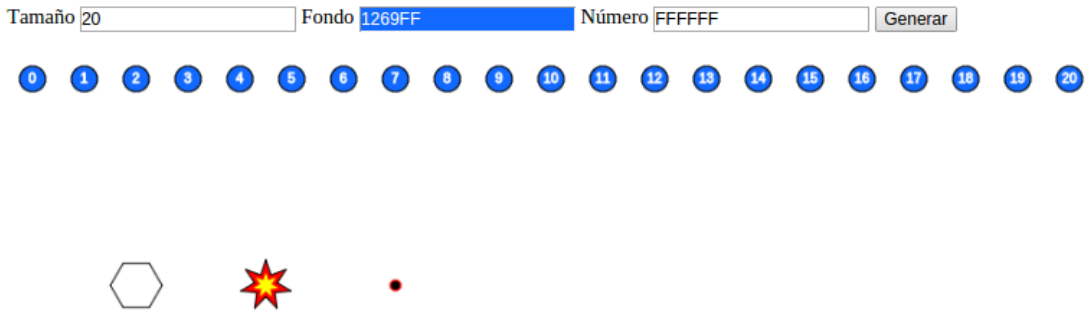


La tasa de refresco a duras penas baja de 60, por lo que se incluyó en el desarrollo de este trabajo.

Capítulo 5: Software auxiliar

5.1 Creador de *sprites*

Utilizando jCanvas de desarrolló una aplicación que dados unos colores, genera automáticamente en una imagen todos los *sprites* asociados a las animaciones y a las unidades. El color sólo afecta a estos últimos.



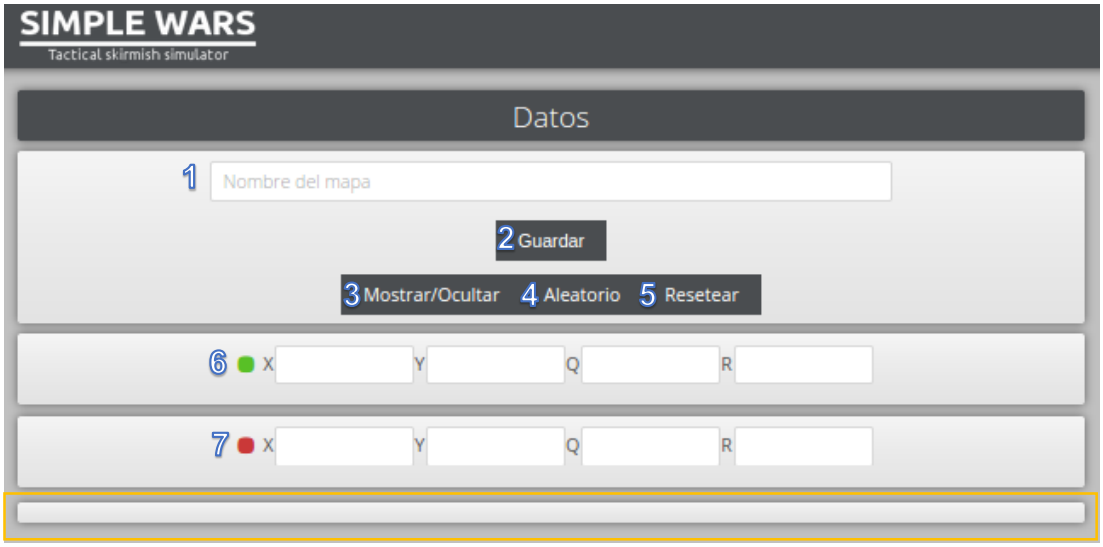
Tras generar los recursos gráficos, basta con guardar las imágenes y tratarlas con algún tipo de software de edición. En este trabajo se ha utilizado GIMP.

5.2 Creador de mapas

Con el fin de enriquecer el producto final y ayudar a la implementación, se desarrolló una aplicación sobre la cual se hicieron pruebas. Una vez se obtuvo una versión estable y sin errores, se añadió a la aplicación final aplicándole validación de datos y un estilo acorde al resto de vistas.

5.2.1 Versión de aplicación

Para esta versión del creador, se ha añadido una nueva dirección ("/map-creator") a la aplicación a la que sólo pueden acceder usuarios administradores registrados en Django (39). Todos los mapas generados conservarán en **tamaño** establecido por la aplicación.



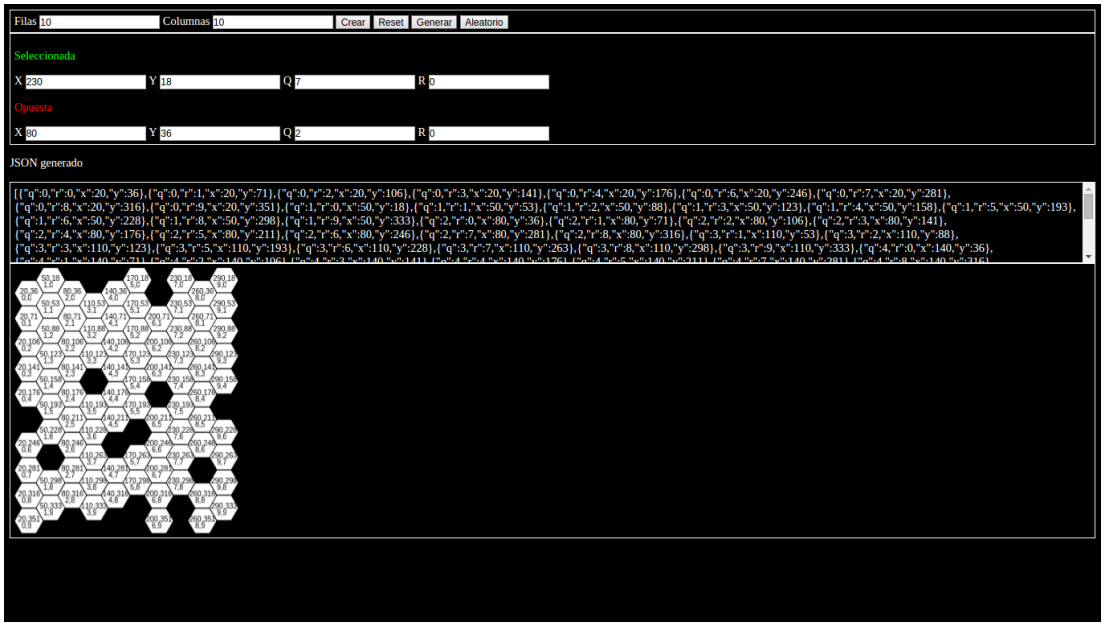
1. Nombre del mapa.
2. Guardar mapa en el servidor
3. Mostrar u ocultar la sección amarilla en donde se albergará el mapa.
4. Generación de mapa aleatorio.
5. Reinicio del mapa.
6. Coordenadas de la posición resaltada en verde. Las indicadas por (X, Y) hacen referencia a las del *canvas* y las indicadas por (Q, R) a la fila y columna de la simulación.
7. Coordenadas de la posición opuesta, resaltada en rojo.

5.2.2 Añadir un nuevo mapa

Una vez se ha obtenido el resultado deseado, para añadir un nuevo mapa hace falta guardar la imagen desde el navegador y añadirla al directorio de ficheros estáticos con el mismo nombre con el que se registró en la aplicación. Una vez ahí, basta con acceder al fichero “battle.js” y en las primeras líneas, añadir al listado de mapas el nombre completo (con extensión) del mapa.

5.2.3 Versión de desarrollador

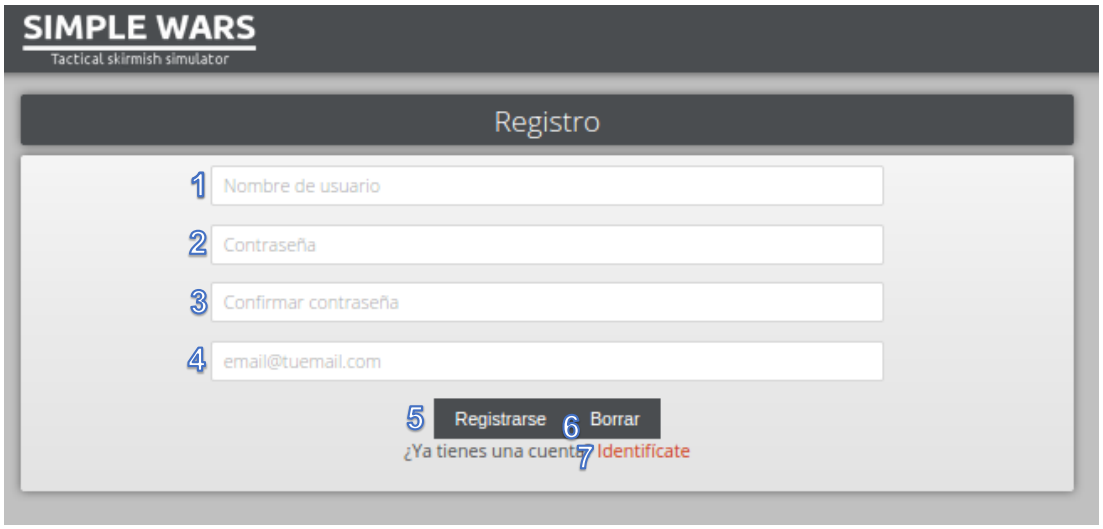
Versión en desarrollo que sirve como base. Es probable que contenga errores y no se asegura su correcto funcionamiento. Es posible generar un mapa de cualquier tipo de tamaño en el que se muestren las coordenadas de la vista y de la simulación en los hexágonos y una sección en donde puede observarse en JSON resultado que se enviará al servidor. Un ejemplo de utilización de esta versión puede ser el siguiente:



Cuando se pulsa sobre el botón “Generar”, en lugar de enviar el resultado al servidor, se muestra en la sección indicada el resultado en JSON.

Capítulo 6: Manual de usuario

6.1 Registro



1. Campo de nombre de usuario.
2. Campo de contraseña.
3. Campo de confirmación de contraseña.
4. Campo de correo electrónico.
5. Registro del usuario.
6. Borrado de los campos del formulario.
7. Enlace a la sección de **identificación**.

En caso de ocurrir algún error o que los campos de registro no tengan el formato correcto se informará al usuario debidamente. Para poder registrarse en la aplicación es necesario que el nombre de usuario no se encuentra ya registrado y que los campos de contraseña coincidan. La dirección de correo electrónico, en caso de ser válida (no es requisito indispensable) servirá para recibir notificaciones de registro, desactivación, activación, borrado o eliminación de la cuenta.

6.2 Identificación

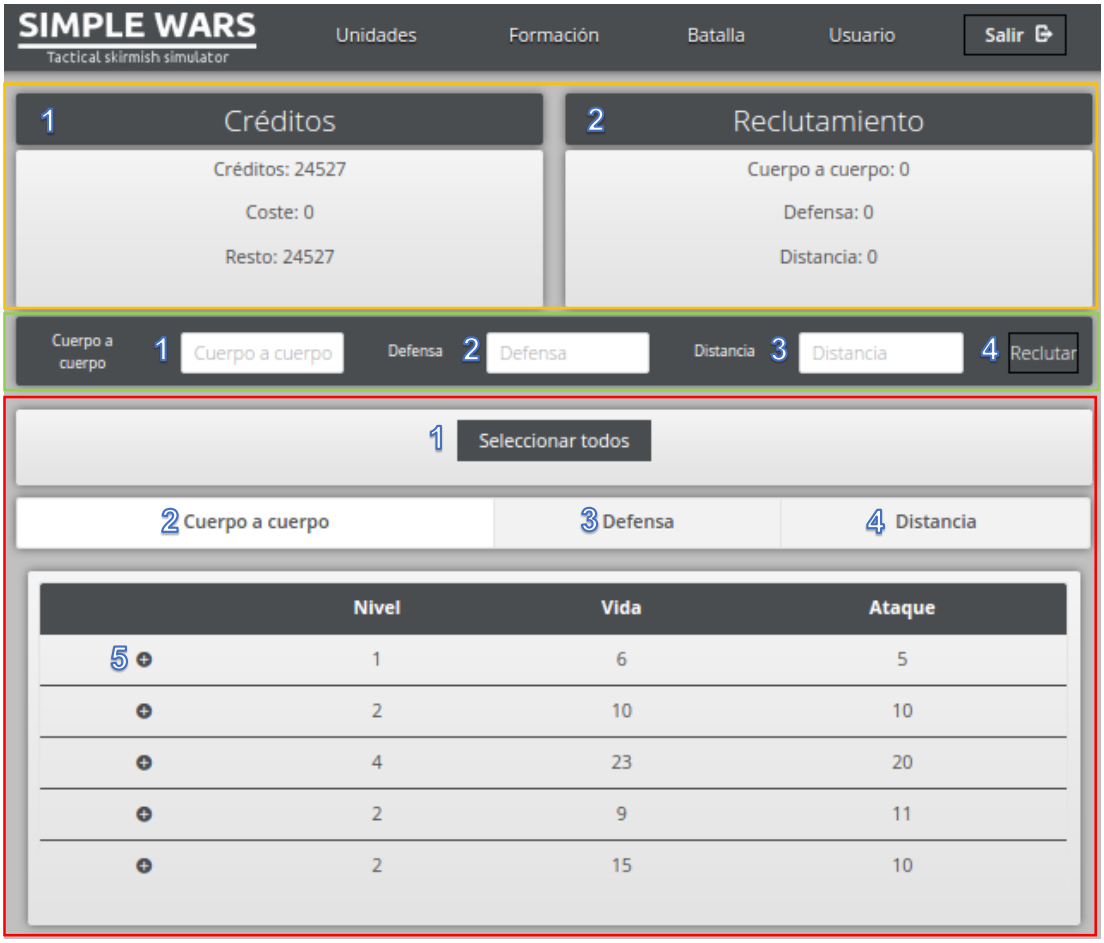
1. Campo de nombre de usuario.
2. Campo de contraseña.
3. Identificación del usuario.
4. Borrado de los campos del formulario.
5. Enlace a la sección de **registro**.

En caso de ocurrir algún error o que los campos de registro no tengan el formato correcto se informará al usuario debidamente. En caso de tener la cuenta desactivada, en lugar del botón de identificación se mostrará el de activación de la cuenta teniendo como resultado si se pulsa en el la reactivación del perfil del usuario.

6.3 Panel de usuario

- Sección amarilla: se trata de la barra de navegación de la aplicación.
    1. Enlace al panel de usuario.
    2. Enlace al **reclutamiento de unidades**.
    3. Enlaces a la **creación** y **modificación de formaciones**.
    4. Enlace a las **batallas**.
    5. Enlace a los **parámetros de usuario**.
    6. Enlace a la salida de la aplicación, lleva a la vista de identificación.
- Sección verde: en esta parte de la interfaz se muestran las estadísticas del usuario:
    - Unidades reclutadas: número total de unidades que se han reclutado.
    - Unidades perdidas: número total de unidades que se han perdido en combate
    - Derrotas: número total de derrotas.
    - Unidades eliminadas: número total de unidades eliminadas.
    - Victorias: número total de victorias.
    - Puntos de mejora usados: número total de puntos empleados en la mejora de unidades.
- Sección roja: en esta parte de la interfaz se muestra la clasificación de usuarios. En concreto siempre se muestran los 5 primeros.

6.4 Unidades


SIMPLE WARS
Tactical skirmish simulator
Unidades
Formación
Batalla
Usuario
Salir
1
Créditos
Créditos: 24527
Coste: 0
Resto: 24527
2
Reclutamiento
Cuerpo a cuerpo: 0
Defensa: 0
Distancia: 0
Cuerpo a cuerpo
1
Cuerpo a cuerpo
Defensa
2
Defensa
Distancia
3
Distancia
4
Reclutar
1
Seleccionar todos
2 Cuerpo a cuerpo
3 Defensa
4 Distancia
Nivel
Vida
Ataque
5
1
6
5
2
10
10
4
23
20
2
9
11
2
15
10

- Sección amarilla: información sobre los créditos del usuario.
    1. Créditos totales y gasto realizado.
    2. Gasto realizado descompuesto por tipos de unidades.
- Sección verde: barra de reclutamiento de unidades. Es posible introducir una cantidad entre 0 y 99.
    1. Cuerpo a cuerpo: unidades cuerpo a cuerpo a reclutar.
    2. Defensa: unidades defensivas a reclutar.
    3. Distancia: unidades de ataque a distancia a reclutar.
    4. Reclutar: hace efectivo el reclutamiento.
- Sección roja: muestra las unidades clasificadas por su tipo. Para cada uno, se mostrará una tabla distinta.
    1. Seleccionar todos: selecciona todas las unidades de la tabla activa.
    2. Muestra las unidades de ataque cuerpo a cuerpo.
    3. Muestra las unidades defensivas.
    4. Muestra las unidades de ataque a distancia.
    5. Abre la sección de modificación de atributos correspondiente para cada unidad.

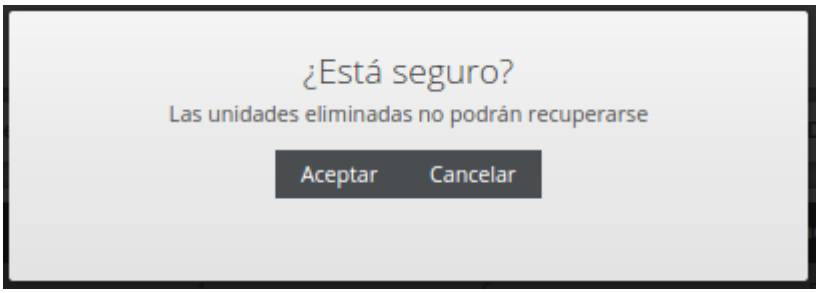
6.4.1 Modificar unidad

En caso de pulsar sobre alguno de los iconos indicado por el 5 de la sección roja, en la sección amarilla se cargará esta ventana cuyas acciones son:

1. Aumentar o disminuir el atributo común.
2. Aumentar o disminuir el atributo propio de la clase.
3. Hace efectiva la modificación y la sección vuelve al estado **anterior**.
4. Cancela modificación y la sección vuelve al estado **anterior**.

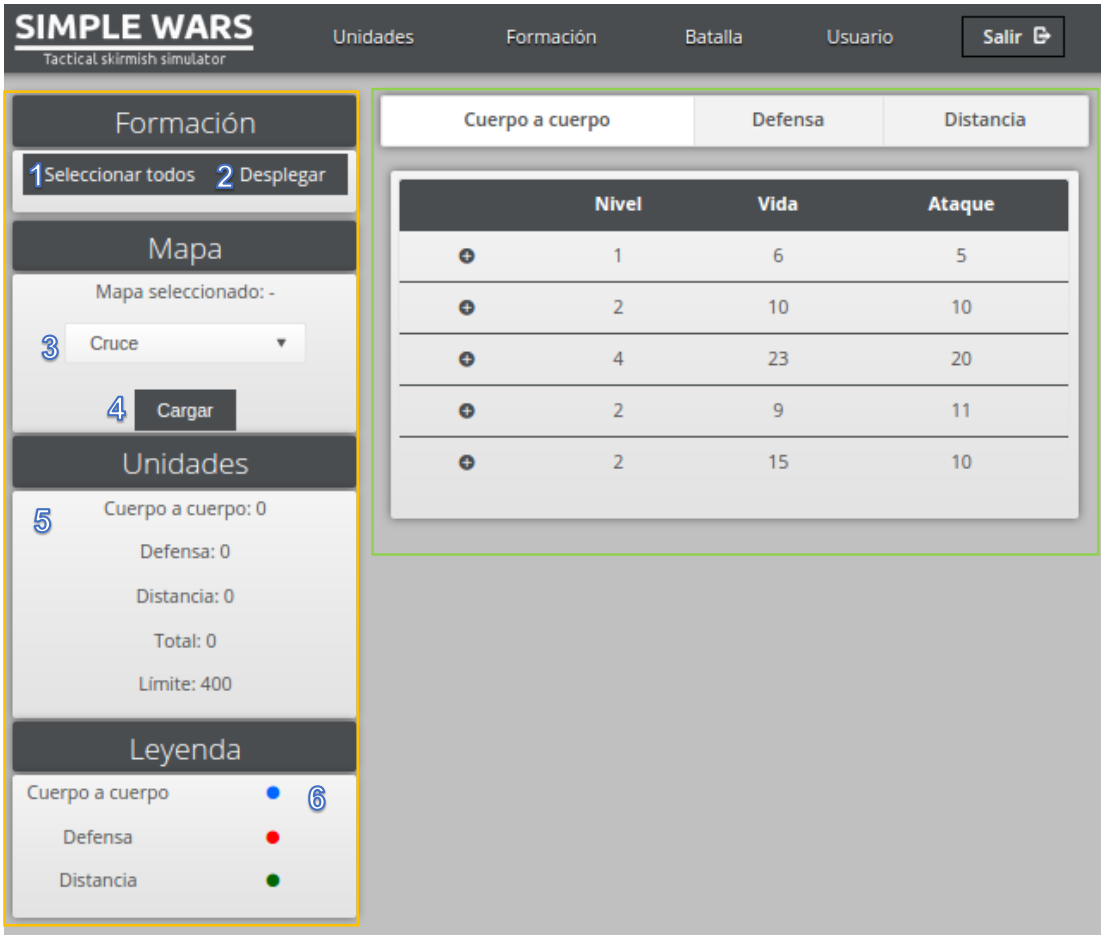
6.4.1 Eliminar unidades

1. Para eliminar las unidades hay que pulsar sobre la fila de la tabla correspondiente a la unidad. También pueden seleccionarse o descartarse todas las unidades pulsando sobre el botón correspondiente. Esto hará que en la parte superior de la sección roja aparezca un nuevo botón.
2. Si se pulsa sobre este, se pide confirmación para llevar a cabo la acción. Se recuperará un crédito por unidad eliminada



6.4 Crear formación

6.4.1 Selección de unidades y mapa

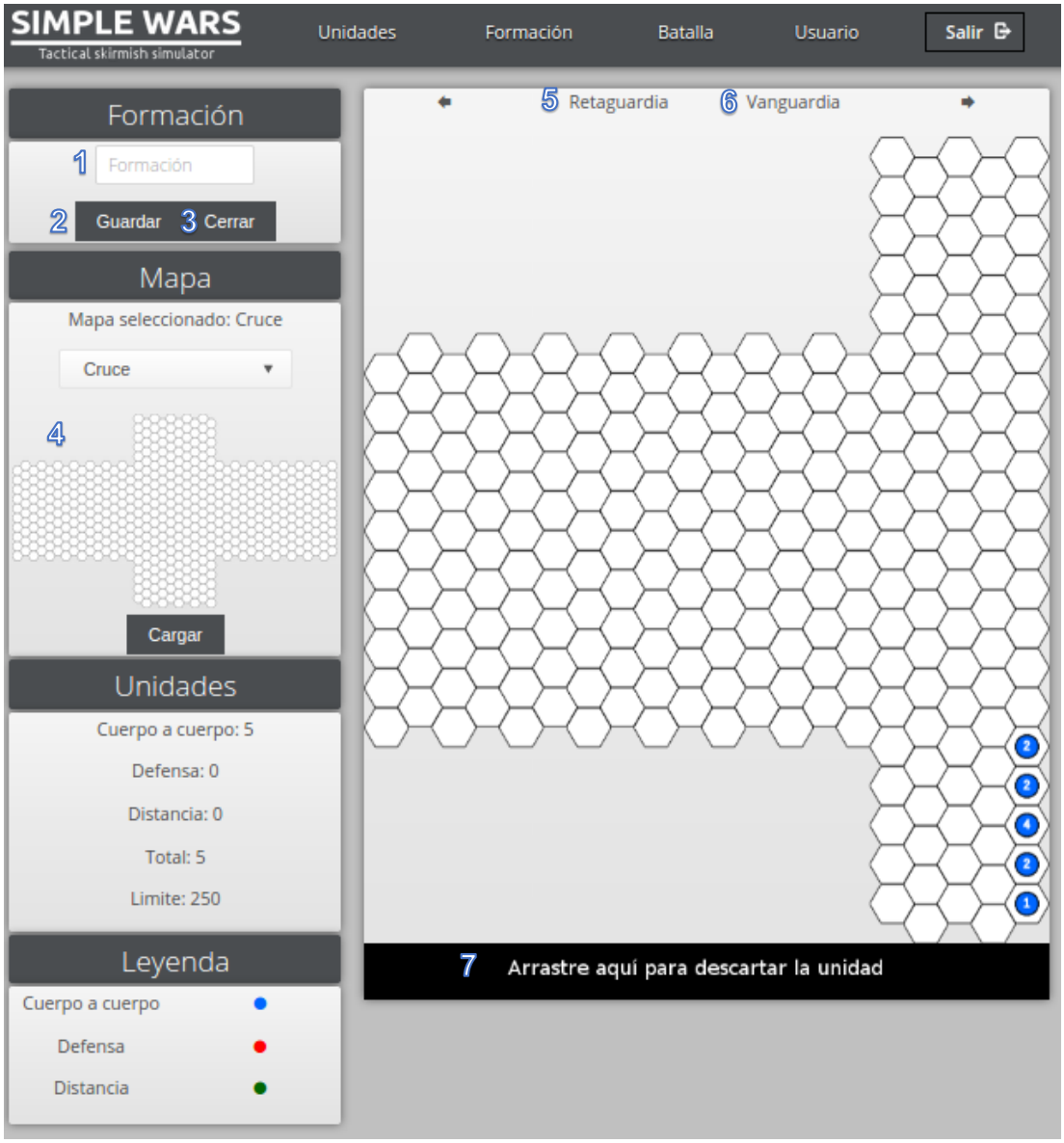


- Sección verde: selección de unidades para desplegar en la formación. También es posible **modificar los atributos** de las unidades desde esta parte de igual forma.
- Sección amarilla: muestra las acciones para seleccionar o descartar todas las unidades (tiene el mismo funcionamiento que en la **sección de reclutamiento**), desplegar en el mapa las unidades e información sobre los parámetros:
    1. Seleccionar todos: selecciona todas las unidades de la tabla activa.
    2. Despliega las unidades seleccionadas en el mapa.
    3. Selección del mapa.
    4. Carga del mapa.
    5. Unidades seleccionadas totales y por tipo. El límite hace referencia al número máximo de unidades que pueden desplegarse en el mapa.
    6. Identificación por colores de todos los tipos de unidades.

El proceso de selección de unidades para la creación de formación es el siguiente:

1. Cargar el mapa: para ello se selecciona el mapa deseado desde el menú indicado por 5 de la sección amarilla y se pulsa sobre el botón 4. La carga se habrá hecho efectiva si el nombre del mapa aparece en la parte superior y se muestra una imagen con el mapa completo en la parte inferior.
2. Seleccionar unidades: de la misma forma que en el apartado correspondiente al **reclutamiento,** se seleccionan las unidades que se deseen utilizar en la formación.
3. Pulsa sobre el botón indicado por 2 en la sección amarilla.

6.4.2 Posicionamiento de unidades en el mapa



1. Formulario de nombre de la formación.
2. Guarda la formación actual con todas las posiciones en el sistema.
3. Cierra el mapa de despliegue de unidades y vuelve al **menú anterior**. Se conservan las posiciones de las unidades desplegadas.
4. Imagen del mapa completo que se desea utilizar.
5. Parte trasera de la formación.
6. Parte delantera de la formación
7. Zona de descarte de unidades.

Una vez que el usuario se encuentre en esta sección, es posible mover las unidades pulsando y arrastrando a las posiciones que se deseen del mapa.

La zona en la que es posible desplegarlas se corresponde con parte izquierda del mapa, estando la derecha destinada a las unidades enemigas en las batallas.

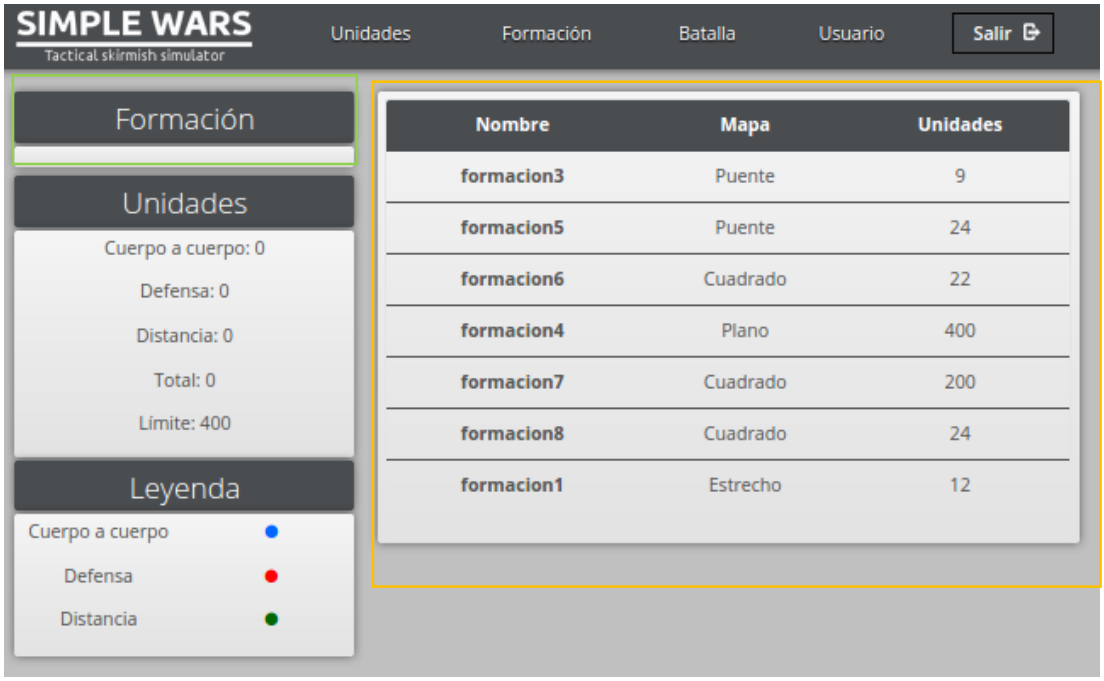
Sólo es posible posicionar unidades en las zonas permitidas del mapa, estas se encuentran representadas como hexágonos blancos. En caso de arrastrar una unidad hacia una posición ocupada por otra unidad, se intercambiarán.

En caso de querer quitar del despliegue alguna unidad, basta con arrastrarla hasta la zona indicada por 7.

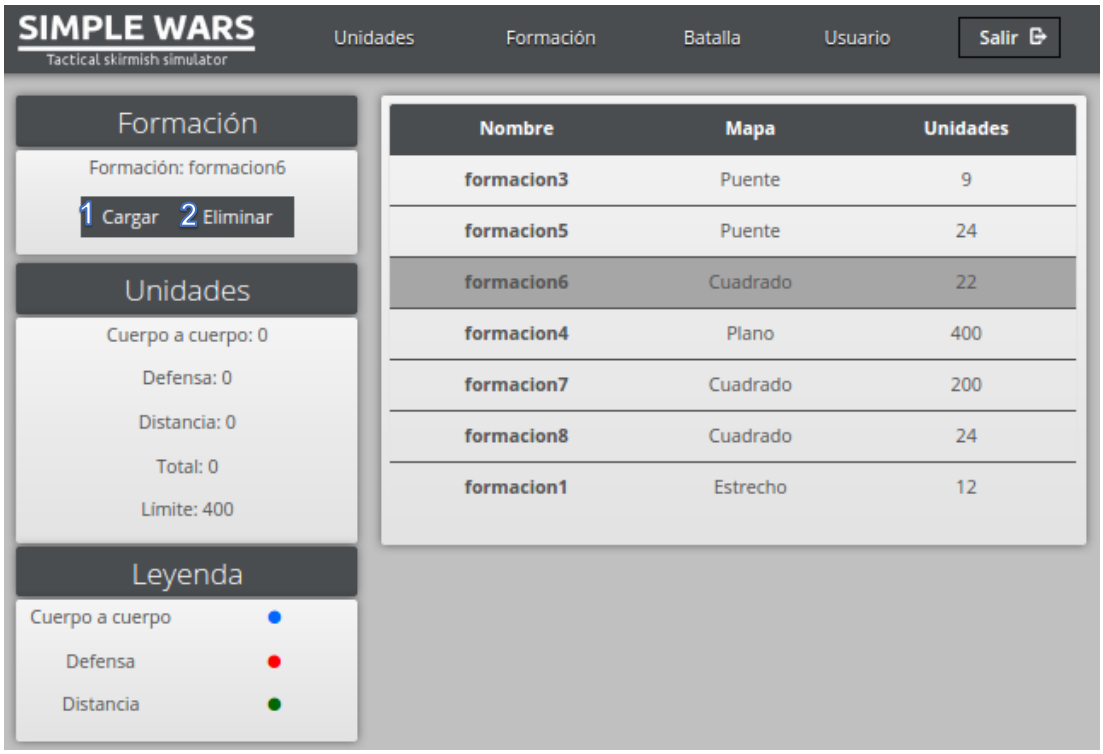
En caso de querer modificar los atributos de alguna de las unidades, basta con pulsar sobre cualquiera de ellas (sin arrastrar) y se abrirá en la parte superior una sección de modificación de atributos de igual forma que en la sección de **reclutamiento**. Si se hace efectiva la modificación de atributos, el hexágono correspondiente a la posición de la unidad se coloreará de verde durante un instante y se mostrará un mensaje indicando el resultado de la operación.

También es posible cambiar de mapa en esta parte de la misma forma que en la **sección anterior**. Las posiciones de las unidades no se conservarán.

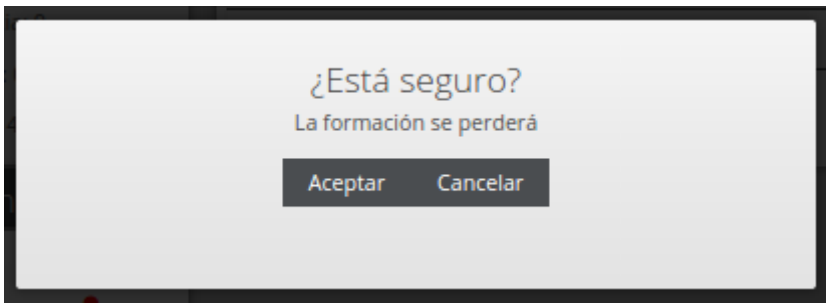
6.5 Modificación de formaciones



La sección amarilla muestra todas las formaciones que el usuario tiene guardadas junto con su mapa asociado y el número total de unidades que forman parte de ella. Si se pulsa sobre cualquiera de ellas, la parte que indica la sección verde cambia.

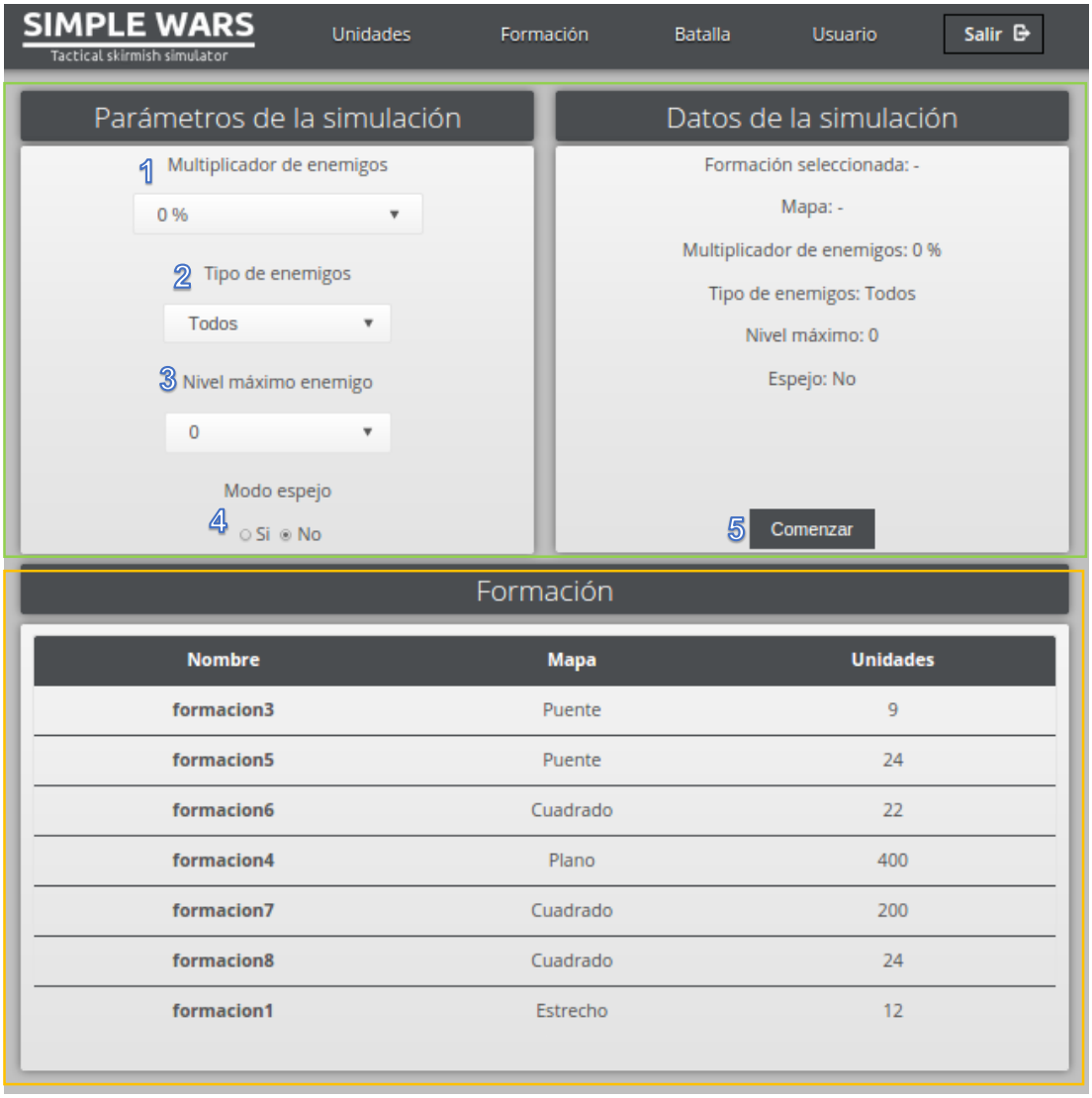


1. Carga la formación en el mapa. Puede ser modificada de la misma forma que en la **sección** correspondiente.
2. Elimina la formación. Muestra un mensaje de confirmación de al usuario. En caso de haber elegido la opción indicada por 1 y descartar todas las unidades, se muestra el mismo mensaje.



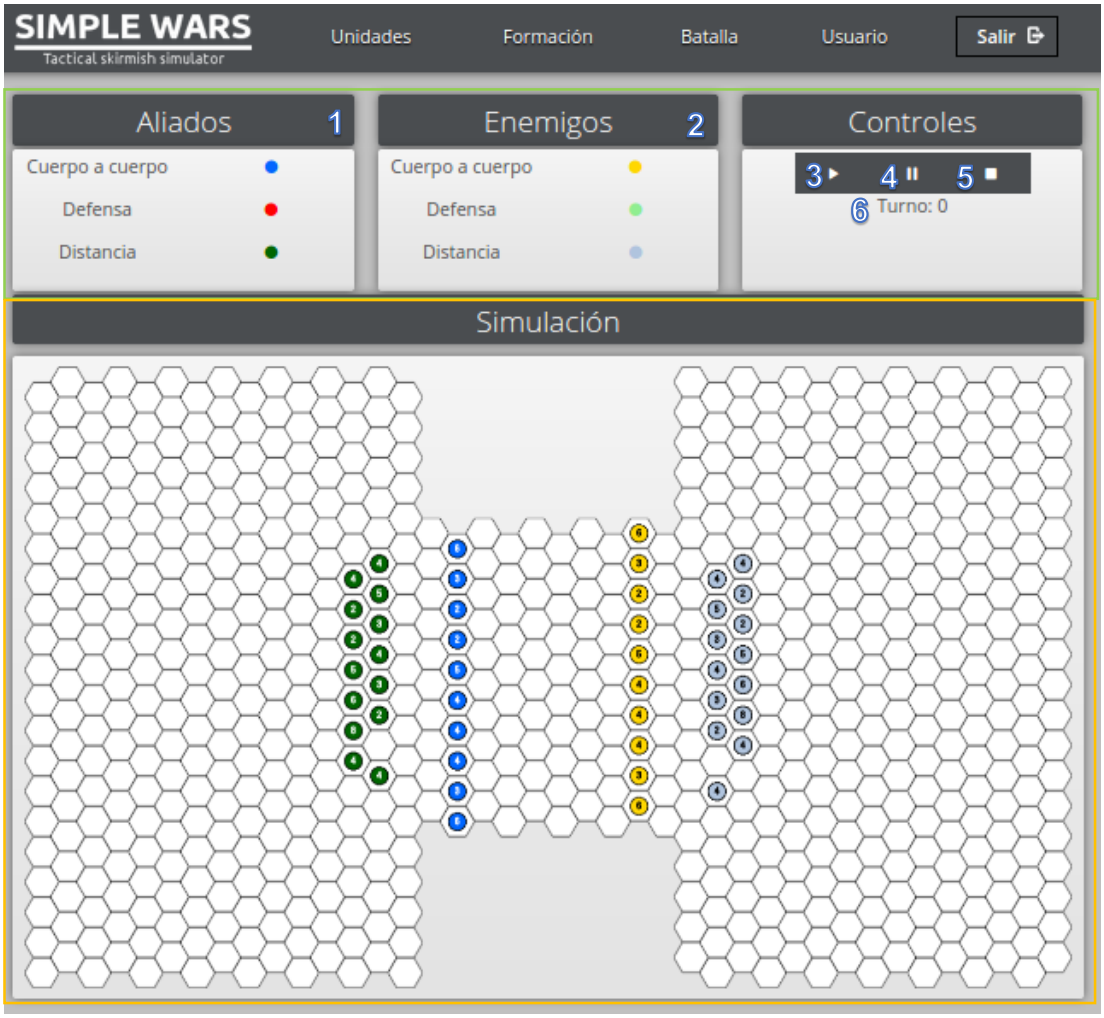
6.6 Batalla

6.6.1 Configuración de parámetros



- Sección verde: muestra los parámetros de inicio de la simulación. En la parte izquierda se seleccionan y en la derecha se muestra qué se ha seleccionado:
    1. Multiplicador de enemigos: cantidad de enemigos de más o de menos que van a combatir.
    2. Tipo de enemigos: contra qué tipo de enemigos se desea combatir.
    3. Nivel máximo enemigo: el nivel máximo que van a tener los enemigos en la batalla.
    4. Modo espejo: si se selecciona esta opción se anulan los demás parámetros ya que se combatirá contra la misma formación que la seleccionada por el usuario pero en posiciones opuestas.
    5. Inicio de la simulación.
- Sección amarilla: se selecciona la formación que se va a utilizar en la batalla de la misma forma que en la **sección** anterior.

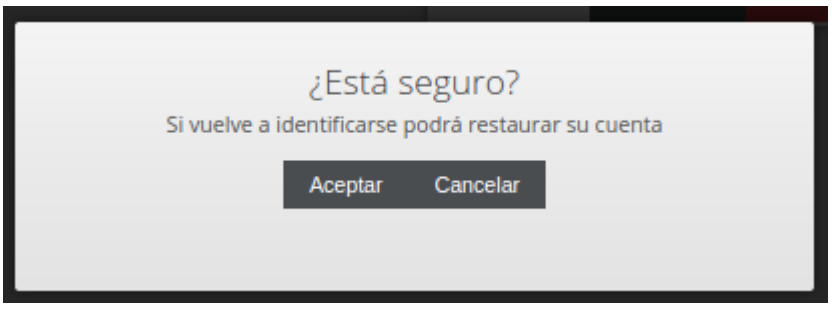
6.6.2 Visualización



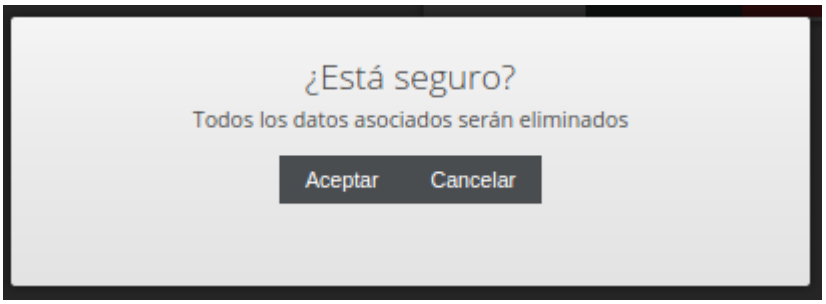
- Sección verde: muestra la información y controles de la visualización del combate:
    1. Código de colores para las unidades aliadas.
    2. Código de colores para las unidades enemigas.
    3. Inicia la visualización.
    4. Pausa la visualización.
    5. Detiene la visualización y muestra los resultados.
    6. Muestra el turno actual.
- Sección amarilla: muestra gráficamente el transcurso del combate.

6.7 Usuario

1. Sección de modificación de nombre de usuario.
2. Sección de modificación de contraseña. Si se cambia, el usuario deberá volver a identificarse en la aplicación.
3. Sección de modificación de correo electrónico.
4. Hace efectivo el cambio en la sección que se encuentre activa.
5. Borra los datos del formulario de la sección que se encuentre activa.
6. Muestra la información actual del usuario.
7. Desactiva el perfil. Muestra un mensaje de confirmación al usuario.



8. Elimina el perfil. Muestra un mensaje de confirmación al usuario.



Capítulo 7: Trabajos futuros

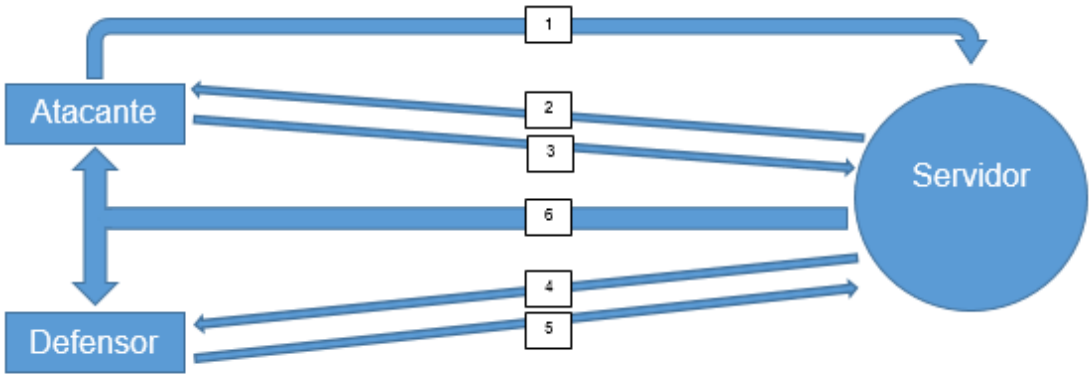
7.1 Conexión entre usuarios

Uno de los objetivos que se establecieron al inicio de este trabajo fue la posibilidad de que los usuarios de la aplicación pudieran realizar batallas entre sus formaciones. Para ello se requiere de la sincronización de conexiones entre los clientes y el servidor para gestionar eventos en tiempo real. Una posible solución pasa por dos objetivos: mantener en memoria una lista de usuarios que se encuentran utilizando la aplicación y gestionar las conexiones entre los clientes y el servidor con algún tipo de tecnología.

Para mantener en el servidor una estructura en memoria que almacene todos los usuarios que en ese momento se encuentran utilizando la ejecución pueden filtrarse las *cookies* (40) que todavía no han expirado igualando el tiempo de validez de estas al tiempo que el navegador permanece abierto (41) y obtener del registro de usuario todos los datos necesarios.

Gestionar conexiones en tiempo real entre el servidor y los clientes no es un problema trivial. Existen diferentes alternativas pero una a tener en cuenta es la utilización de WebSockets (42) e implementar una propia librería o utilizar alguna solución ya implementada para Django (43) en el servidor. Para la parte cliente habría que incluir todo el código Javascript necesario y formatear la información para que no existan problemas de incompatibilidades de formatos entre las partes.

Una vez que se consiga una comunicación correcta en tiempo real entre los clientes y el servidor, el proceso de batallas entre jugadores podría ser el siguiente:



1. Petición del listado de usuarios actuales: para obtener los usuarios que se encuentran utilizando la aplicación en el momento puede realizarse un filtrado entre la tabla de usuarios con la de sesiones, retornando sólo los que no tienen su *cookie* expirada.
2. Retorno al atacante del listado de nombres de usuario.
3. Envío del usuario al que se desea atacar.
4. Notificación de combate al usuario atacado.
5. Confirmación o denegación del combate por parte del usuario atacado.
6. Respuesta del servidor con el resultado final a ambos.

A parte de esta funcionalidad, se podrían utilizar los WebSockets para realizar peticiones asíncronas al servidor sin tener que utilizar AJAX (44). Como punto de partida para este fin, en el siguiente enlace (45) se muestra una posible implementación utilizando AngularJS.

7.2 Migración del cliente a AngularJS

Implementar toda la interfaz utilizando sólo y exclusivamente jQuery aunque fue un reto que se estableció para sí mismo el autor del trabajo (con el fin de aprender y conseguir soltura en Javascript) no es necesario y no es una buena práctica a la hora de facilitar el mantenimiento de una aplicación web.

Para ello puede utilizarse este *framework* el cual gestiona de forma automática las peticiones asíncronas al servidor y los cambios en el DOM, con el consiguiente ahorro de líneas de código.

Como AngularJS consume un servicio REST (46) el primer paso sería adaptar el modelo de datos y desarrollar este servicio. En Django existe un módulo para desarrollar un servicio REST (47). Para poder utilizar AngularJS con Django puede utilizarse una solución ya implementada (48) o desarrollar una propia (49).

7.3 Utilización de un motor de bases de datos NoSQL

Unas de las necesidades de la aplicación web es que las transacciones a la capa de datos sean lo más rápidas posibles. Además, la consistencia de los datos no es un requisito indispensable, ya que no se va a almacenar información importante o protegida.

El soporte oficial de Django no incluye este tipo de motores de bases de datos pero existen multitud de implementaciones de terceros, en este enlace (50) se proponen diversas alternativas.

7.4 Juego

La inteligencia artificial desarrollada para este trabajo aunque sencilla, cumple con su objetivo y se ejecuta en un tiempo razonable. Como el código asociado a esta se encuentra totalmente desacoplado con el del cliente, es posible realizar cambios en ella independientemente de la implementación del cliente siempre que se respete el formato de la **estructura resultado** y que no se requieran nuevos elementos gráficos.

7.4.1 Distintos tipos de terreno

Los mapas pueden enriquecerse con distintos tipos de terreno y penalizaciones al movimiento de las unidades. Sería necesario modificar toda la estructura que almacena los mapas, modificar la inteligencia artificial de la simulación y crear nuevos *sprites* para su la representación gráfica.

7.4.2 Búsqueda de caminos

La búsqueda de caminos óptimos en un mapa generado con situaciones aleatorias no es una tarea sencilla. Una posible solución sería guardar el mapa en una estructura de tipo grafo (51) y realizar búsquedas utilizando algún tipo de algoritmo, por ejemplo A* (52). En este enlace (53) se ofrece una introducción muy detallada sobre cómo funciona y una posible forma de implementarlo.

7.4.3 Orientaciones, incluir más tipos de unidades y más atributos

En este trabajo no se tiene en cuenta la última orientación de la unidad para la búsqueda de posiciones adyacentes por lo que una ampliación sencilla sería modificar el código correspondiente para que se guarde la última orientación utilizada y comience la búsqueda desde ahí.

Los tipos de unidades de este trabajo al ser sólo 3 es posible nivelar los atributos de forma sencilla para que no existan desventajas entre combinaciones de unidades. Si se desean añadir más, sería necesario desarrollar algún tipo de fórmula matemática que controle los atributos de las unidades, añadir más si corresponde y normalizar los resultados como convenga.

7.4.4 Creación de una fortaleza base para cada jugador

Se trata de una posible ampliación que combinaría muy bien con **incluir** distintos tipos de terreno y los **combates** entre jugadores. Para ello cada jugador podrá crear su fortaleza con una serie de unidades (que podrían tener un comportamiento diferente al que tienen en una batalla normal) que defendiesen al jugador en caso de recibir un ataque.

7.4.5 Enriquecer la creación de formaciones y la modificación de unidades

Desarrollar en el entorno gráfico la selección múltiple de unidades para posicionar varias al mismo tiempo en una sola acción e incluir una funcionalidad que permita modificar los parámetros de varias unidades a la vez.

7.4.6 Mejorar la inteligencia artificial de las unidades en los combates

Desarrollar otro tipo de algoritmos de inteligencia artificial para conseguir un comportamiento más complejo. Este objetivo es muy amplio, depende del diseño y las posibilidades del desarrollador. Como documentación, este informe (54) analiza una posible aproximación apoyando su estudio en juegos comerciales.

7.4.7 Crear *sprites* adaptables a cualquier resolución

Las imágenes que se han creado para la realización de este trabajo no se redimensionan correctamente para cada resolución mostrando un aspecto *pixelado* en algunos momentos. Para evitar esto es posible utilizar otro formato llamado SVG (55) el cual sea cual sea la resolución escala los gráficos convenientemente de forma automática.

7.4.8 Torneos

En el caso de implementar un sistema de **conexión entre usuarios**, sería posible organizar torneos entre ellos. Un posible sistema de torneos podría ser el sistema suizo (56).

7.5 Seguridad y rendimiento

7.5.1 Auditoría de seguridad

Desplegar la aplicación y realizar una auditoría de seguridad que encuentre todas las vulnerabilidades e implante una serie de medidas preventivas ante un posible ataque. Un buen recurso sobre amenazas de seguridad en las aplicaciones web es el siguiente (57) el cual analiza las amenazas más peligrosas y ofrece mucha información para paliar sus posibles consecuencias.

7.5.2 Aumentar el rendimiento

Analizar el rendimiento de la aplicación: tiempo de proceso, consultas a la capa de datos, consumo de memoria, pruebas en los navegadores más utilizados... Con los resultados del análisis, modificar lo necesario para disminuir el tiempo de respuesta y mejorar el rendimiento general de la aplicación. Podría ser necesario modificar o cambiar el esquema de la **base de datos** e implementar algún sistema de cacheo (58).

Bibliografía

1. **Linnovate.** MEAN. [En línea] 22 de julio de 2015. http://mean.io/#!/.

2. **MongoDB.** MongoDB. [En línea] 22 de julio de 2015. https://www.mongodb.org/.

3. **StrongLoop.** Express. [En línea] 22 de julio de 2015. http://expressjs.com/es/.

4. **Google.** AngularJS. [En línea] 22 de julio de 2015. https://angularjs.org/.

5. **Joyent.** Node.js. [En línea] 22 de julio de 2015. https://nodejs.org/.

6. **Django Software Foundation.** Django. [En línea] 22 de julio de 2015. https://www.djangoproject.com/.

7. **Wikipedia.** ORM. [En línea] 27 de agosto de 2015. https://es.wikipedia.org/wiki/Mapeo_objeto-relacional.

8. **Oracle Corporation.** MySQL. [En línea] 22 de julio de 2015. https://www.mysql.com/.

9. **The jQuery Foundation.** jQuery. [En línea] 22 de julio de 2015. https://jquery.com/.

10. **DocumentCloud.** Underscore. [En línea] 22 de julio de 2015. https://github.com/jashkenas/underscore/.

11. **Digital Surgeons.** Gumby CSS. [En línea] 22 de julio de 2015. http://www.gumbyframework.com/.

12. **Hampton Catlin, Natalie Weizenbaum, Chris Eppstein, and numerous contributors.** SASS. [En línea] 6 de agosto de 2015. http://sass-lang.com/.

13. **Evans, Caleb.** jCanvas. [En línea] 22 de julio de 2015. https://github.com/caleb531/jcanvas.

14. **Photon Storm.** Phaser. [En línea] 22 de julio de 2015. http://phaser.io/.

15. **Wikipedia.** *Sprite***.** [En línea] 26 de agosto de 2015. https://es.wikipedia.org/wiki/Sprite_%28videojuegos%29**.**

16. **Wikipedia.** RTS. [En línea] 6 de agosto de 2015. https://es.wikipedia.org/wiki/Videojuego_de_estrategia_en_tiempo_real**.**

17. **Games, Red Blob. Red Blob Games.** *Hexagon grids*. [En línea] 6 de agosto de 2015. http://www.redblobgames.com/grids/hexagons/**.**

18. **W3Schools.** *Canvas*. [En línea] 7 de agosto de 2015. http://www.w3schools.com/html/html5_canvas.asp.

19. **Wikipedia.** DOM. [En línea] 26 de agosto de 2015. https://es.wikipedia.org/wiki/Document_Object_Model**.**

20. **Wikipedia.** Distancia Manhattan. [En línea] 26 de agosto de 2015. https://es.wikipedia.org/wiki/Geometr%C3%ADa_del_taxista**.**

21. **Wikipedia.** Distancia euclidiana. [En línea] 26 de agosto de 2015. https://es.wikipedia.org/wiki/Distancia_euclidiana**.**

22. **Wikipedia.** DDA**.** [En línea] 26 de agosto de 2015. https://es.wikipedia.org/wiki/Analizador_Diferencial_Digital_%28algoritmo_gr%C3%A1fico%29**.**

23. **W3Schools.** *Drag and drop*. [En línea] 25 de agosto de 2015. http://www.w3schools.com/html/html5_draganddrop.asp**.**

24. **Wikipedia.** JSON. [En línea] 26 de agosto de 2015. https://es.wikipedia.org/wiki/JSON**.**

25. **Franco, Jose Manuel Lopez.** HTTP. [En línea] 26 de agosto de 2015. http://trevinca.ei.uvigo.es/~txapi/espanol/proyecto/superior/memoria/node46.html.

26. **Wikipedia.** *Tweening*. [En línea] 26 de agosto de 2015. https://es.wikipedia.org/wiki/Tweening**.**

27. **GmbH, CodeAndWeb. CodeAndWeb.** *Spritesheet*. [En línea] 26 de agosto de 2015. https://www.codeandweb.com/what-is-a-sprite-sheet**.**

28. **W3Schools.** *setInterval*. [En línea] 26 de agosto de 2015. http://www.w3schools.com/jsref/met_win_setinterval.asp.

29. **Wikipedia.** Cliente - servidor. [En línea] 27 de agosto de 2015. https://es.wikipedia.org/wiki/Cliente-servidor**.**

30. **Wikipedia.** Programación por capas. [En línea] 27 de agosto de 2015. https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas.

31. **Wikipedia.** Acoplamiento. [En línea] 27 de agosto de 2015. https://es.wikipedia.org/wiki/Acoplamiento_inform%C3%A1tico.

32. **Wikipedia.** MVC. [En línea] 27 de agosto de 2015. https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador.

33. **Wikipedia.** *Single page application*. [En línea] 27 de agosto de 2015. https://es.wikipedia.org/wiki/Single-page_application**.**

34. **Wikipedia.** Inyección SQL. [En línea] 27 de agosto de 2015. https://es.wikipedia.org/wiki/Inyecci%C3%B3n_SQL.

35. **Stackoverflow.** MySQL *row limit*. [En línea] 27 de agosto de 2015. http://stackoverflow.com/questions/15585602/change-limit-for-mysql-row-size-too-large.

36. **How-To Geek, LLC. How-To Geek.** MySQL *query caching* [En línea] 27 de agosto de 2015. http://www.howtogeek.com/howto/programming/speed-up-your-web-site-with-mysql-query-caching/.

37. **Augustin, Aymeric.** Django debug toolbar. [En línea] 27 de agosto de 2015. https://github.com/django-debug-toolbar/django-debug-toolbar.

38. **Wikipedia.** FPS. [En línea] 29 de agosto de 2015. https://es.wikipedia.org/wiki/Im%C3%A1genes_por_segundo**.**

39. **Django Software Foundation.** Crear superusuario en Django. [En línea] 28 de agosto de 2015. https://docs.djangoproject.com/en/1.8/intro/tutorial02/#creating-an-admin-user.

40. **Wikipedia.** *Cookie*. [En línea] 28 de agosto de 2015. https://es.wikipedia.org/wiki/Cookie_%28inform%C3%A1tica%29**.**

41. **Stackoverflow.** *Authenticated users list*. [En línea] 28 de agosto de 2015. http://stackoverflow.com/questions/2723052/how-to-get-the-list-of-the-authenticated-users.

42. **Wikipedia.** WebSocket. [En línea] 28 de agosto de 2015. https://es.wikipedia.org/wiki/WebSocket**.**

43. **Daniel Greenfeld, Audrey Roy & Two Scoops Press.** Django Packages. [En línea] 28 de agosto de 2015. https://www.djangopackages.com/grids/g/websockets/.

44. **Wikipedia.** AJAX. [En línea] 29 de agosto de 2015. https://es.wikipedia.org/wiki/AJAX.

45. **Berry, Clint. Clint Berry.** AngularJS WebSockets. [En línea] 29 de agosto de 2015. http://clintberry.com/2013/angular-js-websocket-service/.

46. **Wikipedia.** REST. [En línea] 28 de agosto de 2015. https://en.wikipedia.org/wiki/Representational_state_transfer.

47. **Christie, Tom.** Django REST *framework*. [En línea] 28 de agosto de 2015. https://github.com/tomchristie/django-rest-framework/tree/master**.**

48. **Rief, Jacob.** Django + AngularJS. [En línea] 28 de agosto de 2015. https://github.com/jrief/django-angular.

49. **Stone, Kevin. kevinastone.** Django REST *framework*. [En línea] 28 de agosto de 2015. http://blog.kevinastone.com/getting-started-with-django-rest-framework-and-angularjs.html**.**

50. **Foundation, Django Software.** Django NoSQL. [En línea] 28 de agosto de 2015. https://code.djangoproject.com/wiki/NoSqlSupport.

51. **Wikipedia.** Grafo. [En línea] 28 de agosto de 2015. https://es.wikipedia.org/wiki/Grafo_%28estructura_de_datos%29.

52. **Wikipedia.** A*. [En línea] 28 de agosto de 2015. https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*.

53. **Games, Red Blob. Red Blob Games.** A* *path finding*.[En línea] 28 de agosto de 2015. http://www.redblobgames.com/pathfinding/a-star/introduction.html.

54. **David Churchill, Abdallah Saffidine y Michael Buro. Skatgame.** *Fast Heuristic search*. [En línea] 29 de agosto de 2015. https://skatgame.net/mburo/ps/aiide12-combat.pdf.

55. **Wikipedia.** SVG. [En línea] 28 de agosto de 2015. https://es.wikipedia.org/wiki/Scalable_Vector_Graphics**.**

56. **Wikipedia.** Sistema suizo. [En línea] 29 de agosto de 2015. https://es.wikipedia.org/wiki/Sistema_suizo.

57. **Project, Open Web Application Security.** OWASP. [En línea] 29 de agosto de 2015. https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013.

58. **Wikipedia.** Caché web. [En línea] 29 de agosto de 2015. https://es.wikipedia.org/wiki/Cach%C3%A9_web.