THE REAL CODEWARS

Trabajo de Fin de Grado

Ingeniería Informática

Septiembre de 2023

Autor

Luis Prada Rodrigo

Tutores

Roberto Therón Sánchez

Luis Martín Liras

Nicolás García Martín

Certificado del tutor TFG

D. Roberto Therón Sánchez, profesor del Departamento de informática y automática de la Universidad de Salamanca,

HACE CONSTAR:

Que el trabajo titulado “TheRealCodeWars”, que se presenta, ha sido realizado por Luis Prada Rodrigo, con DNI ****0227Z y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Ingeniería Informática en esta Universidad.

Salamanca, 6 de septiembre de 2023.

THERON SANCHEZ ROBERTO - 07976246F

Digitally signed by THERON SANCHEZ ROBERTO - 07976246F
Date: 2023.09.05 13:35:26 +02'00'

Fdo.: Roberto Therón Sánchez.

Índice

Índice 3
Resumen 4
Palabras clave 5
Summary 6
Key words 7
1. Introducción 8
2. Objetivos del proyecto 10
  2.1 El juego 10
  2.2 La biblioteca 11
  2.3 Interfaz web 11
3. Conceptos teóricos 12
  3.1 Codewars 12
  3.2 Juego TBS 13
  3.3 Juego RTS 13
4. Técnicas y herramientas 14
5. Aspectos relevantes del desarrollo del proyecto 16
6. Trabajos relacionados 31
7. Líneas de trabajo futuras 31
Bibliografía 33

Resumen

En este proyecto se va a desarrollar una plataforma que permita la ejecución de competiciones entre distintos programadores usando la programación competitiva.

TheRealCodeWars será un juego por turnos en el que dos participantes competirán por ser el primero en destruir la base del contrario.

Antes de poder iniciar la partida, será necesario que cada equipo de programadores cree un bot, el cual definirá las acciones de los soldados. Posteriormente se subirá ese bot a la página y, finalmente, se creará la partida eligiendo el número de rondas, el mapa, los 2 participantes de la partida y los bots que usarán. Si al finalizar las rondas ninguna base ha sido destruida, el equipo ganador será aquel cuya base mantenga más vida.

Para ello, se desarrollará un Frontend en el que los usuarios puedan acceder a un tutorial, descargarse las herramientas necesarias para la creación de los bots y entender el funcionamiento del juego. Desde este Frontend, también se podrá subir, ver y eliminar el código personal y crear partidas entre los participantes y bots deseados. Una vez haya una partida en ejecución, cualquier usuario podrá acceder a la interfaz web en la que se verá como procede la partida en tiempo real.

Será necesario también un Backend que se encargue de implementar todo lo relacionado con el juego en sí (soldados, mapa, acciones posibles, etc.) y que gestione también la comunicación entre los bots y la partida.

Por último, se desarrollará una biblioteca que se pondrá a disposición de los usuarios para facilitar el desarrollo de los bots por su parte. Esta biblioteca se encargará de la comunicación con el servidor donde se ejecuta el juego y proporcionará funciones que ejecuten las acciones posibles de los soldados.

Palabras clave

Programación competitiva, concursos de programación, concurso de programación de IA.

Summary

In this project we are going to develop a platform that allows the execution of competitions between different programmers using competitive programming.

TheRealCodeWars will be a turn-based game in which two participants will compete to be the first to destroy the opponent's base.

Before the game can start, each team of programmers will have to create a bot, which will define the actions of the soldiers. Afterwards, this bot will be uploaded to the page and, finally, the game will be created by choosing the number of rounds, the map, the 2 participants of the game and the bots that will be used.

If at the end of the rounds, no base has been destroyed, the winning team will be the one whose base maintains more life.

For this, a Frontend will be developed in which users can access a tutorial, download the necessary tools for the creation of the bots and understand how the game works. In this Frontend, it will also be possible to upload, view and delete personal code and create games between participants and desired bots. Once a game is running, any user will be able to access the web interface where he/she will see how the game is proceeding in real time.

A Backend will also be needed to implement everything related to the game itself (soldiers, map, possible actions, etc.) and to manage the communication between the bots and the game.

Finally, a library will be developed and made available to the users to facilitate the development of the bots on their side. This library will be in charge of the communication with the server where the game is running and will provide functions that execute the possible actions of the soldiers.

Key words

Competitive programming, programming contests, AI programming competition.

1. Introducción

Según Halim et al. (2013), la programación competitiva consiste en que, dados unos problemas conocidos en la ciencia de la computación, estos sean resueltos lo más rápido posible.

El término “problemas conocidos en la ciencia de la computación” implica que en la programación competitiva nos enfrentamos a problemas de ciencias de la computación resueltos y no a problemas de investigación (cuyas soluciones aún se desconocen). “Lo más rápido posible” es el elemento competitivo, un comportamiento humano muy natural.

Cabe destacar que estar bien versado en programación competitiva no es el objetivo final, es sólo el medio. El verdadero objetivo final es formar programadores que estén mucho más preparados para producir mejor software o para enfrentarse a los problemas de investigación más complejos en el futuro.

HP Codewars es un ejemplo de programación competitiva, pero con un público principalmente novato en el ámbito de la programación, cuyo objetivo es más bien introducir a los estudiantes en el ámbito de la programación de una forma divertida.

TheRealCodeWars es propuesto por HP como uno de los TFG que propone la empresa en su colaboración con la Universidad de Salamanca. En este se propone crear un evento para HP Codewars más parecido al BattleCode desarrollado por el MIT.

BattleCode lo hace de una forma más visual y llamativa, lo que podría favorecer el evento de Codewars al ser para un público más joven.

Kar (s.f) define BattleCode como un juego de estrategia en tiempo real. En él los concursantes se enfrentan entre sí en un mundo virtual con un conjunto de robots como sus equipos, recogiendo recursos y diferentes tipos de armas. Los robots/agentes del juego utilizan radios para comunicarse y trabajar en equipo. La idea es ocupar el máximo terreno mientras se eliminan los robots del equipo contrario. Al final, quien tenga el máximo territorio bajo su control, es decir, el equipo que haya construido más edificios agrupados, es el ganador. En caso de una

ocupación territorial igual, gana el equipo con el mayor número de robots/agentes en pie. EL MIT organiza este torneo todos los años (Figura 1).

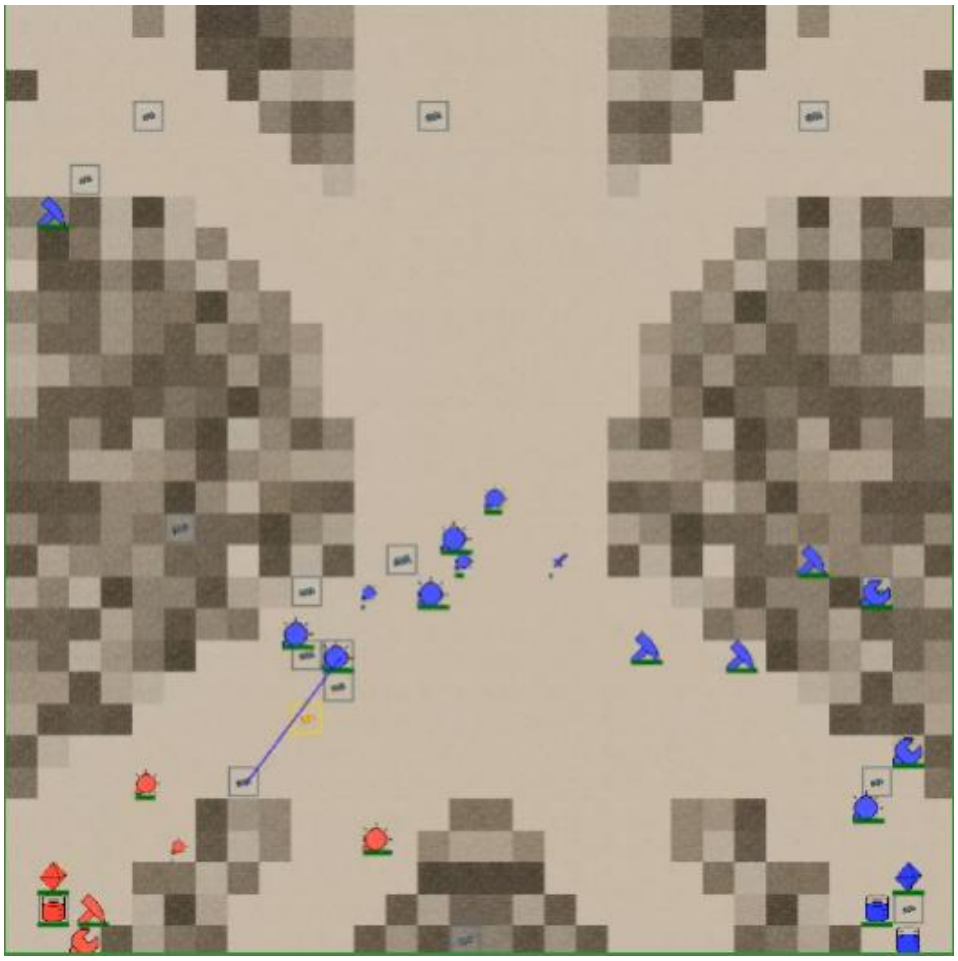


Figura 1. Ejemplo de partida de BattleCode.

El evento de Codewars se está quedando estancado con ejercicios de desarrollo muy estáticos, poco visuales y poco interactivos, que no animan a los estudiantes a iniciarse en el mundo del desarrollo. Por esto, se pretende unir la idea de BattleCode con el evento de Codewars, creando un juego parecido, pero más simple y que permita a los usuarios crear bots de una manera relativamente sencilla. De esta forma, los participantes, con conocimientos informáticos no muy profundos, pueden competir entre ellos de manera más interesante y divertida que resolviendo problemas.

2. Objetivos del proyecto

2.1 El juego

Se quiere hacer un juego TBS (Turn Based Strategy) en el que habrá distintos tipos de unidades:

- Base: La única acción que puede realizar es crear soldados en sus 4 casillas contiguas. Es la única unidad capaz de crear unidades.
- Atacante: Tiene unos puntos de ataque muy altos y poca vida.
- Tanque: Tiene mucha vida, pero poco ataque.
- Helicóptero: Tiene un ataque y vida intermedios, pero puede volar.
- Sanador: No puede atacar, pero puede curar a sus aliados. También puede nadar.

Habrá distintos tipos de casillas:

- Agua: solo admitirán soldados que vuelen o naden.
- Lava: Solo soldados que vuelen podrán estar en ellas.
- Tierra: Cualquier soldado puede estar en ella.

Las unidades o soldados podrán realizar un determinado número de acciones cada turno. Estas serán:

- Mover: La unidad se moverá en la dirección indicada, comprobando previamente si puede estar en la casilla destino (si no está ocupada o no lo impide el tipo de casilla.
- Atacar: La unidad atacará la posición indicada, restando a la vida del soldado que haya en esa posición los puntos de ataque del atacante.
- Curar: La unidad curará la posición indicada, sumando a la vida del soldado que haya en esa posición (excepto a las Bases) sus puntos de curación.
- Crear: Solo las Bases lo pueden hacer. Creará un soldado del tipo indicado en una posición libre adyacente.
- Ver la información de las casillas en su rango: Esta acción no disminuirá el número de acciones por turno del soldado. Devolverá un array de casillas que contendrá las casillas en rango del soldado con información de cada casilla (ocupada, enemigo o aliado, tipo de casilla, posición).

Se generará una base por equipo en una posición concreta del mapa al empezar la partida y se enviarán sus coordenadas al equipo contrario. El objetivo será destruir la base enemiga. Ganará el equipo que la destruya primero o el que su base tenga más vida si se acaban los turnos.

Todo esto formará parte del servidor backend, el cual podrá también, mediante un servicio REST, atender las peticiones a la interfaz web.

La información de la partida se guardará y actualizará en una base de datos SQLite.

2.2 La biblioteca

Se requerirá una biblioteca en Python que oculte a los participantes todo lo relacionado con la conexión con el servidor backend para controlar a los soldados.

La biblioteca deberá contener una clase Soldier y otra Tile con los atributos que se consideren necesarios.

También deberá tener funciones que permitan realizar las acciones que sean posibles (moverse, atacar, curar, crear soldados o ver la información de las casillas que están en el rango del soldado).

2.3 Interfaz web

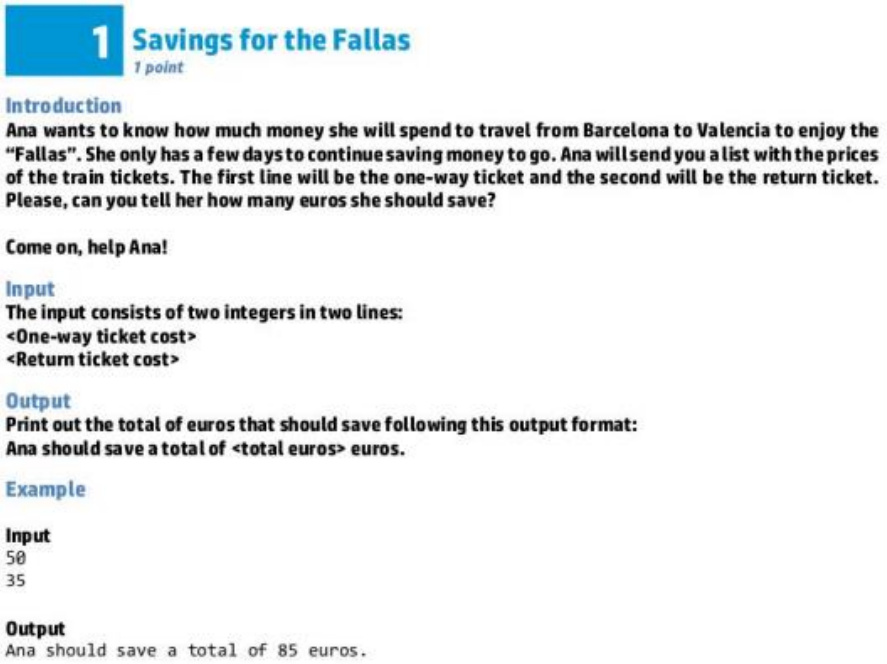
Se quiere crear una interfaz web que permita

- La gestión de usuarios (registrarse y acceder a la página).
- Subir código, verlo una vez subido y eliminarlo.
- Crear partidas pudiendo elegir los usuarios, el código del usuario si tiene más de uno subido, el mapa y las rondas.
- Visualizar una partida en tiempo real.

3. Conceptos teóricos

3.1 Codewars

Mañà Marín (2019) define HP Codewars como un evento celebrado en Barcelona desde 2015 por la división española de HP, con el objetivo de despertar el interés por las carreras STEM (Science, Technology, Engineering and Mathematics) en los estudiantes. Consiste en una competición en la que equipos de tres miembros reciben una lista de 30 problemas que deben resolver utilizando un lenguaje de programación como Java, Python o C++. Cada problema se especifica con un texto explicativo que le da un contexto, una especificación de la entrada que escribirán los jueces, una especificación de la salida que se espera y, a continuación, un ejemplo de una posible entrada y su salida correspondiente (Figura 2).



1 Savings for the Fallas

1 point

Introduction

Ana wants to know how much money she will spend to travel from Barcelona to Valencia to enjoy the "Fallas". She only has a few days to continue saving money to go. Ana will send you a list with the prices of the train tickets. The first line will be the one-way ticket and the second will be the return ticket. Please, can you tell her how many euros she should save?

Come on, help Ana!

Input

The input consists of two integers in two lines:
<One-way ticket cost>
<Return ticket cost>

Output

Print out the total of euros that should save following this output format:
Ana should save a total of <total euros> euros.

Example

Input

```
50
35
```

Output

```
Ana should save a total of 85 euros.
```

Figura 2. Primer problema propuesto en una de las ediciones

En este concurso se permite la participación de alumnos desde 3º de la ESO hasta 2º de Bachillerato o un Ciclo formativo de Grado Medio en equipos de 3 personas pertenecientes al mismo centro educativo.

El evento actualmente consiste en lo siguiente:

- Los equipos participantes deben resolver problemas utilizando uno de los lenguajes de programación propuestos.

- Se propone un listado de problemas con diferentes puntuaciones según su dificultad.
- Los equipos deberán seleccionar los problemas que mejor se adapten a sus conocimientos y resolverlos durante las 3 horas de competición.
- Para conseguir los puntos de un problema, el ejercicio deberá estar correctamente resuelto.

3.2 Juego TBS

Un juego TBS o Turn Based Strategy es un juego en el cual el flujo se particiona en partes bien definidas y visibles llamadas turnos o rondas ("Videojuego de estrategia por turnos", 2022).

En estos el tiempo del juego no transcurre de forma continua (como sucede en un juego de estrategia en tiempo real), si no por turnos, de forma similar al ajedrez, damas u otros juegos de mesa.

Cada jugador tendrá su turno, en el que pensará sus acciones y moverá a sus unidades. Tras finalizar el turno se pasa al siguiente jugador.

En este tipo de juegos se dispone de un periodo de análisis antes de perpetrar la acción del juego, asegurando una separación entre el flujo del juego y el proceso de pensamiento, de manera que nos lleva presumiblemente a mejores soluciones.

3.3 Juego RTS

Un juego RTS o Real Time Strategy es un juego en el que no hay turnos, sino que el tiempo transcurre de forma continua para los jugadores ("Videojuego de estrategia en tiempo real", 2023).

Los RTS están pensados para ser jugados de forma muy dinámica y rápida. A diferencia de los basados en turnos, no precisan un planteamiento tan pausado de las decisiones y se centran muy a menudo en la acción.

4. Técnicas y herramientas

- Los lenguajes que se utilizarán serán C++ para el servidor y Python para los bots.

  La ventaja por la que se ha elegido C++ frente a otros lenguajes orientados a objetos como Java o Python es que es muy rápido en ejecución. Esto es un punto importante para el proyecto, ya que al ser un juego con muchos soldados realizando acciones cada turno, será importante la velocidad de ejecución.

  Se ha elegido Python para el código de los bots por su sencillez, dado que ese código lo escribirán alumnos con poca experiencia en programación.

  Otra de las ventajas de Python es que al ser un lenguaje interpretado no hay que compilarlo para ejecutarlo, con lo que es muy sencillo ejecutar el código de los alumnos, solo es necesario tener instalado Python en el servidor y cuando se cree una partida, el servidor creará un hilo por bot que ejecutará el comando:

  python {path_to_bot}/bot1.py

- Para hacer el servidor, se utilizará un servicio REST mediante la biblioteca cpp-httplib.

  Otras opciones eran drogon y cpprestsdk, pero después de ser instaladas en Windows daban errores de compilación por abandonos o no mantenimiento de las bibliotecas, con lo que se eligió cpp-httplib, la cual es muy intuitiva, sigue recibiendo actualizaciones, funciona en todas las plataformas y no es necesario instalarla para su funcionamiento, con incluirla en el Cmake funciona (esto facilitará la instalación del proyecto desde GitLab para quien lo quiera usar).

- Para la comunicación entre el servidor y peticiones que se hagan por JavaScript para la interfaz, se utilizará la biblioteca JSON for Modern C++ para enviar y recibir los datos en formato JSON. Esta biblioteca es muy sencilla de usar, tiene buena documentación, es fácil de integrar en el proyecto y es eficiente.

- Para la interfaz web del juego se necesitarán imágenes que representen cada soldado y cada casilla. Para ello utilizaré Gencraft, una IA capaz de generar imágenes a partir de frases. Hay varias IAs similares, pero esta es la que mejor funciona y más imágenes permite generar de forma gratuita.

- Para los diseños de las páginas web, se utilizará el framework Bootstrap, el cual permite simplificar mucho todo este apartado. Este framework permite crear páginas responsivas de manera sencilla, tiene muy buena documentación, es compatible con la mayoría de los navegadores, recibe actualizaciones regularmente y es gratuito y de código abierto.

- Se usará una base de datos SQLite para almacenar la información de la partida.

  Se ha decidido utilizar SQLite frente a otras como MySQL porque para las necesidades del proyecto no precisaba algo demasiado potente.

  Algunas ventajas de SQLite son que tiene un tamaño muy inferior al servidor MySQL, almacena directamente la información en un único archivo, facilitando su copia y no se requiere de ninguna configuración.

5. Aspectos relevantes del desarrollo del proyecto

A lo largo este apartado se mencionarán varias veces las clases que se han creado para el desarrollo del proyecto. En el Anexo III se encuentra un diagrama de clases a disposición del lector para facilitar la comprensión de las clases y sus relaciones.

El proyecto se ha desarrollado siguiendo una metodología ágil por Sprints de 2 semanas. Con un backlog de tareas y requisitos del proyecto y las reuniones cada 2 semanas con los tutores de HP y de la universidad, donde se iban seleccionando tareas en el orden que se creía adecuado.

El primer obstáculo al empezar el desarrollo fue el lenguaje de programación. Se había elegido C++ y no lo había utilizado nunca, con lo que el **primer Sprint** fue dedicado al aprendizaje de dicho lenguaje, a la preparación del entorno de desarrollo (Visual Studio) y al aprendizaje de CMake para la compilación del proyecto de forma más sencilla que con un make.

El **segundo Sprint** consistió en buscar una biblioteca para crear un servicio REST e implementar una sencilla prueba de funcionamiento:

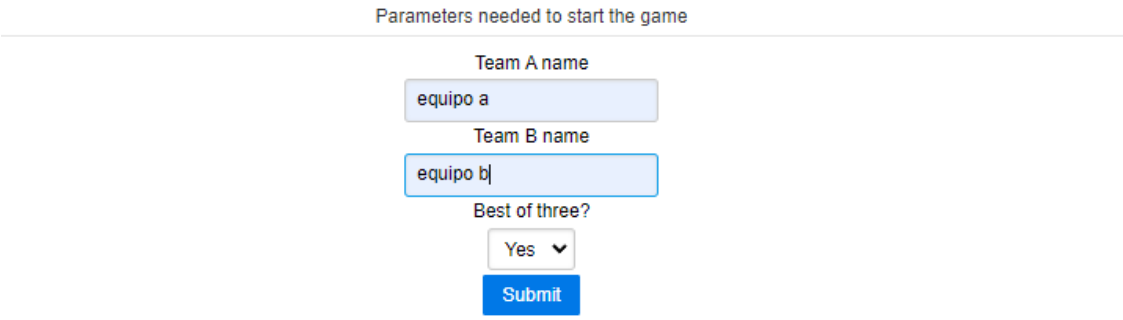


Figura 3. Prueba inicial

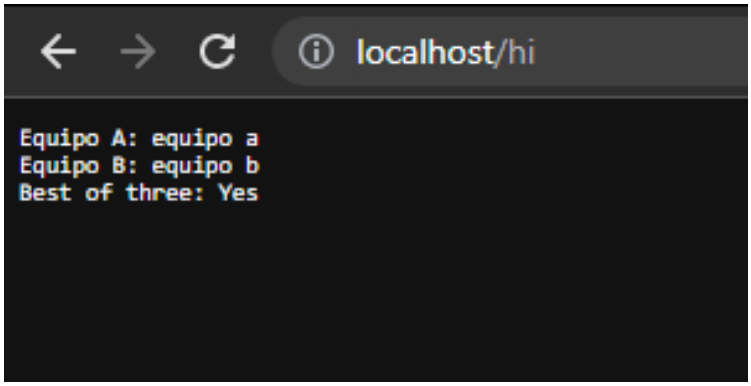


Figura 4. Respuesta Prueba inicial

Como se ve en las figuras 3 y 4, en esta prueba solo se quería comprobar el funcionamiento del servicio REST, mediante un formulario que enviara información al servidor y una respuesta del servidor con la información recibida.

Un problema que surgió en este Sprint fue la utilización de imágenes en una página HTML, algo que debería ser sencillo pero que no reconocía la ruta.

En el **Sprint 3** se resolvió el problema de la imagen. Este se debía a que al usar un servicio REST, la ruta que se ponía en el HTML no era una ruta a un directorio, sino al endpoint correspondiente en el código, el cual no existía. Para resolver esto y que no tuviera que implementar un endpoint por cada imagen, ya que iba a necesitar varias imágenes, por lo menos para la interfaz web del juego, se hizo lo siguiente:

```
// Provides the image with the name writen after 'images/' (e.g images/logo.jpg)
svr.Get(R"(/images/(.+))", [dir](const httplib::Request &req, httplib::Response &res) {
    string image = req.matches[1];
    auto dir1 = dir + "\\..\\html\\images\\" + image;

    ifstream file (dir1, ios::binary);

    if (!file.is_open())
    {
        res.set_content("Internal Error", "text/plain");
    } else {
        stringstream htmlBuf;
        htmlBuf << file.rdbuf();
        string format = image.substr(image.find('.'));

        if (format == ".svg")
        {
            format = "image/svg+xml";
        } else if (format == ".png")
        {
            format = "image/png";
        } else {
            format = "image/jpeg";
        }
        file.close();
        res.set_content(htmlBuf.str(), format);
    }
});
```

Figura 5. Endpoint para imágenes

Este endpoint mostrado en la *Figura 5* recibirá cualquier petición GET dirigida a */images/* y devolverá el contenido del fichero. Al principio no funcionaba, pero era porque se tenía que abrir el fichero en binario (*ios:binary*).

En este Sprint también se quiso crear un bucle de prueba que ejecutase las rondas de una partida, el problema que apareció aquí fue que el único hilo del proceso estaba dedicado al servidor, con lo que se investigó cómo crear hilos y se puso uno para la escucha del servidor y otro para la prueba de ejecución de las rondas.

Para la prueba se creó una clase base Soldier y varias clases que heredan de esta: Tank, Helicopter, Attacker y Healer, cada uno con sus atributos.

```
Soldier(int attack, int health, int speed, Location l, bool swims_, bool flies_, std::string name_)
    : attackPoints(attack), healthPoints(health), moveSpeed(speed), pos(l), swims(swims_), flies(flies_), name(name_) {}
```

Figura 6. Constructor de la clase Soldier

```
class Tank : public Soldier
{
public:
    Tank(Location pos) : Soldier(10, 20, 1, pos, false, false, "Tank") {}
    Tank(int x, int y) : Soldier(10, 20, 1, Location {x, y}, false, false, "Tank") {}
};
```

Figura 7. Clase Tank

Las clases hijas no tienen distintos métodos ni atributos a los de la clase padre, solo son un constructor con los atributos predeterminados para un soldado de dicha clase (*Figuras 6 y 7*).

La prueba era sencilla, simplemente se ejecutaba un bucle en el que se generaban soldados aleatorios en posiciones aleatorias (Figura 8).

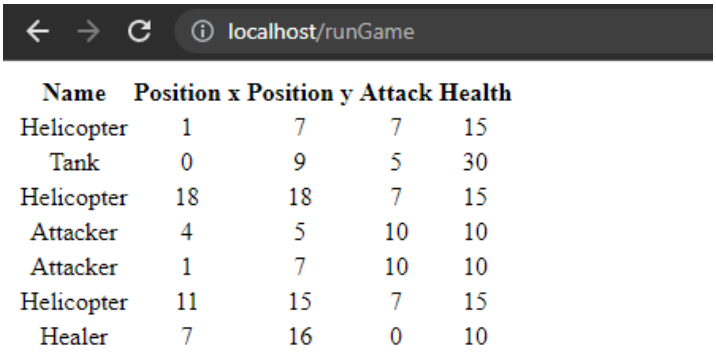


Figura 8. Prueba de rondas

En el **Sprint 4** se implementó la acción de move(), la cual hace que un soldado se mueva en la dirección indicada como parámetro.

Como todavía no se sabía cómo se iba a hacer la interacción con el código de los alumnos, se creó la función turn() que ejecutaría cada soldado cuando es su turno, donde se escribiría el código o bot que controlaría a los soldados para poder hacer pruebas de funcionamiento de las acciones que se iban implementando.

También se hicieron funciones para la serialización de la clase Dashboard, la cual contiene la información de las casillas del mapa, para poder guardar mapas en ficheros y elegir cargar el mapa que se quiera al iniciar una partida.

Por último, se decidió cambiar la interfaz para representar el mapa y los soldados de forma algo más visual, aunque todavía a modo de prueba.

Para ello mediante la clase canvas de HTML y funciones y cálculos que se hicieron en JavaScript se hizo la interfaz que se puede observar en la *Figura 9*. Los puntos negros representan a los Tanks, los rojos a los Attackers, los verdes a los Healers y los azules a los Helicópteros.

Las casillas blancas eran el vacío, las azules agua y las marrones tierra. Como se puede comprobar, no se pusieron limitaciones a los tipos de casillas en los que puede estar cada tipo de soldado por el momento.

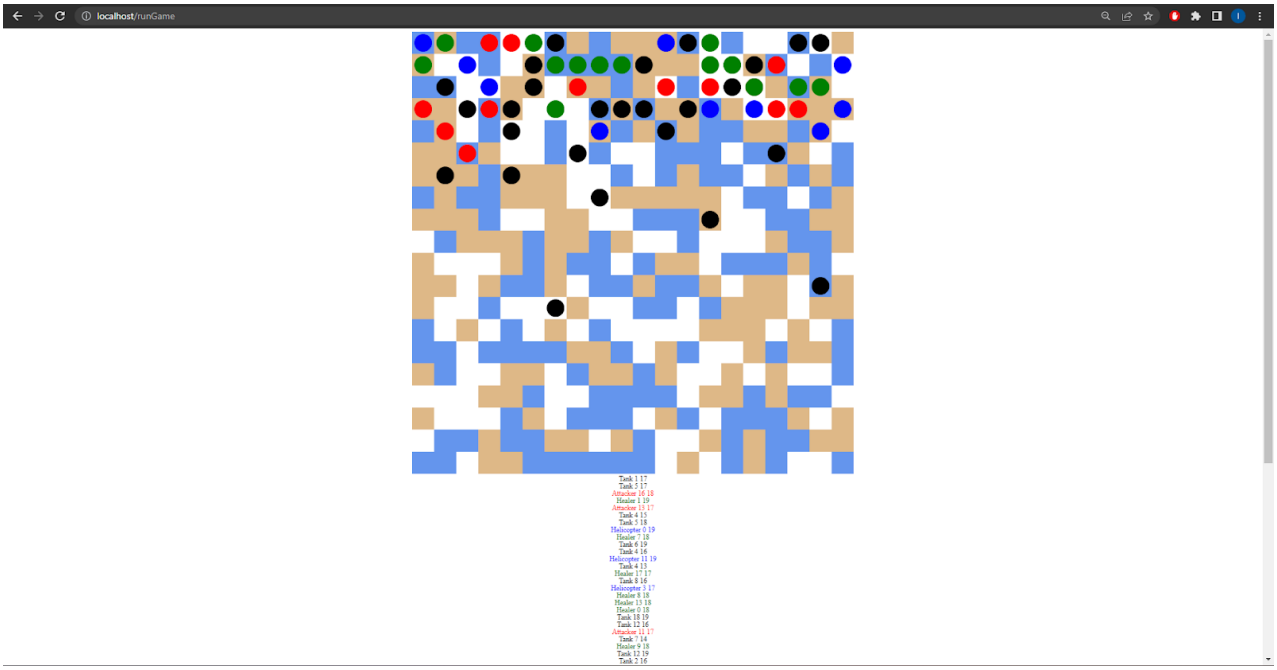


Figura 9. Primera interfaz

Para que la interfaz web actualice la información sobre la partida se utiliza la función setInterval() de JavaScript, haciendo peticiones al endpoint */gameStatus* para recibir información actualizada del estado del juego en formato JSON.

En el **Sprint 5** se ha continuado desarrollando la interfaz y se ha cambiado el canvas por una tabla de HTML, ya que se querían utilizar imágenes en vez de dibujar con el canvas cuadrados o círculos (*Figura 10*).

Ha habido bastantes problemas con esto, dado que al principio se creaba todo el HTML de las tablas de 0 en el JavaScript que actualiza la información de la página, pero esto generaba una tabla de 1 sola columna.

Al final se creó la tabla de 20x20 celdas en el HTML base, la cual se recorre con JavaScript para actualizar el contenido de cada celda.

Se ha intentado utilizar algún framework que facilitara la creación de dicha tabla como Bootstrap, pero, después de varios intentos, no se logró que la tabla quedase como se quería, así que se mantuvo sin framework.

Para las imágenes, las de los tipos de casilla se buscaron por Google, mientras que para los soldados se utilizó Gencraft.

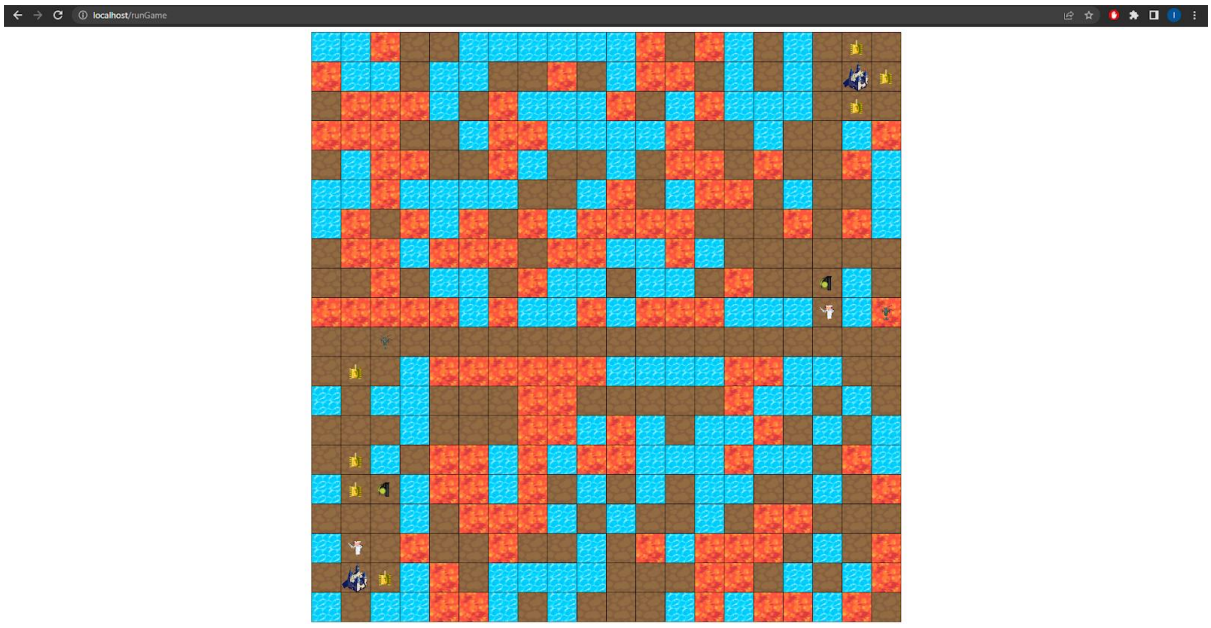


Figura 10. Interfaz web actualizada

Las casillas azules son agua, las rojas lava y las marrones tierra. Los dos castillos son las bases de cada equipo, los tanques amarillos los Tanks, los helicópteros son Helicopters, el médico (el personaje de blanco) es el Healer y el soldado verde con un lanzamisiles es un Attacker.

En cuanto al backend, se añadieron las acciones de atacar y la base, la cual es la unidad que generará soldados en sus 4 casillas adyacentes.

Para almacenar la información del mapa (un vector de casillas y otro de soldados), se incorporó una base de datos de SQLite. Así se tiene la misma información del mapa en todas las clases.

En el **Sprint 6** se incorporó un endpoint para iniciar una partida, pues hasta el momento se ejecutaba al iniciar el servidor. Cuando se acceda saldrá un formulario (*Figura 11*).

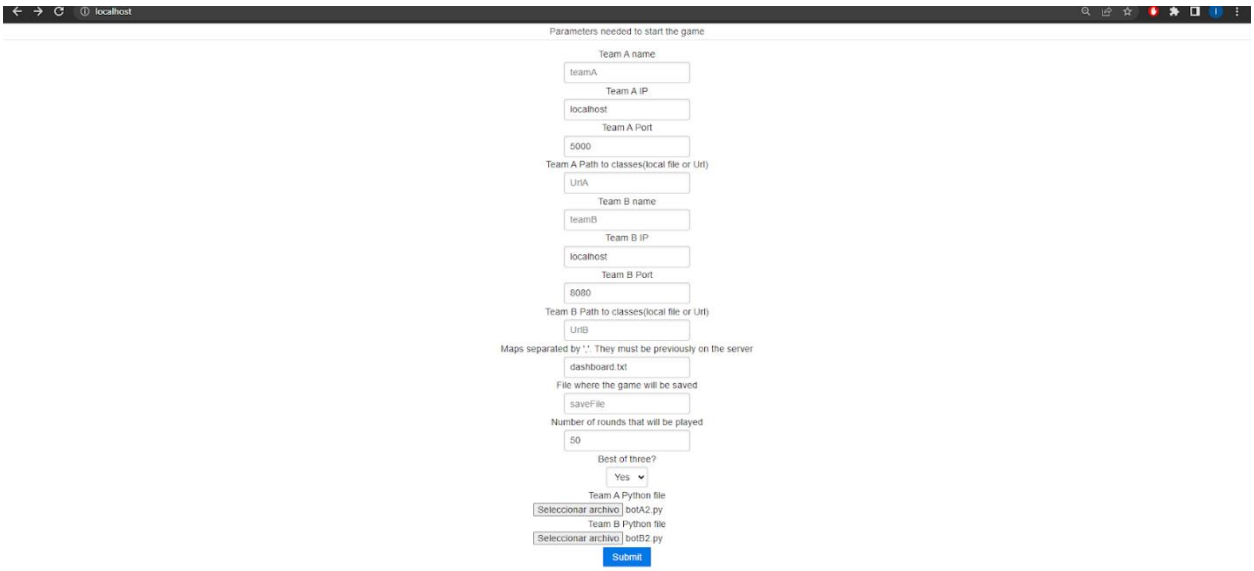


Figura 11. Formulario inicial para crear partida

Muchos de esos campos se crearon por si fueran necesarios, pero los campos de *Team A / Team B name, Team A / Team B path to classes, saveFile y Best of three* no hacían nada.

Los mapas que se envíen deberán existir ya en el servidor, y se seleccionará uno aleatorio entre ellos para jugar la partida.

Los ficheros .py que se le pasen, se almacenarán en una carpeta del servidor y se ejecutarán cada uno en un hilo distinto mediante el uso de:

system("python " + nombreFichero)

El código en python se comunica con el servidor mediante endpoints en un servicio REST.

En cada turno, se enviará una petición a la IP y puerto correspondiente al endpoint */turn*, con la información del soldado. El servidor de python deberá llamar a los endpoints de */move, /attack, /create o /getNearTiles* para obtener información de su entorno y realizar acciones.

Para evitar que el código del equipo A pueda controlar soldados del equipo B y viceversa, se ha generado un token para cada equipo. Este token se envía al inicio de la partida al endpoint que deberán tener implementado en el servidor de Python llamado */setToken*. También, para poder hacer una buena finalización, y que al acabar una partida se finalicen los hilos correspondientes a los servidores en python, se deberá implementar un endpoint llamado */finish* con el código necesario para finalizar el servidor al ser llamado.

Se han implementado también endpoints para comprobar si el soldado puede moverse, atacar o crear unidades, pero está saltando un error que no está claro por qué es, pero parece ser por sobrecarga del servidor. Se ha decidido no usarlos de momento para intentar bajar la carga del servidor.

Para el servidor de Python se ha usado Tornado, ya que es el más potente e intuitivo a la vez, se ha tenido que añadir una línea para evitar que intente usar ipv6 y hasta que no dé timeout no use ipv4:

```
requests.packages.urllib3.util.connection.HAS_IPV6 = False
```

Al hacer esto, usa directamente ipv4 y se evitan alrededor de 2 segundos de retraso al hacer peticiones al servidor de C++.

La comunicación entre el código en Python y el servidor usando REST sobrecargaba demasiado al servidor y se necesitaba un control mediante algún token para que solo el bot correspondiente pudiera manejar sus soldados y no el bot del otro equipo, o cualquier persona haciendo la petición POST adecuada.

Frente a esa idea, los sockets son simplemente más sencillos, permitiendo una conexión entre el bot y el servidor, resolviendo con ello el problema del token y reduciendo la carga del servicio REST.

Esto hizo que se haya dedicado todo el **Sprint 7** a cambiar esa interacción por sockets TCP. Aquí los problemas que se han tenido fueron todos relacionados con intentar sincronizar bien mediante send() y recv() al bot y al servidor.

En el **Sprint 8** se ha creado una biblioteca en Python para ocultar toda la parte de comunicación con el servidor a los alumnos.

Los alumnos tendrán que implementar el código de la función RCW_turn() de la clase Bot de esta forma:

```
import RealCodeWars

1 usage  Luis Prada *
class Bot(RealCodeWars.RCW):
    Luis Prada *
    def RCW_turn(self, soldiers):
        for soldier in soldiers:
            # Controlar a cada soldado
```

Figura 12. Ejemplo de uso RCW_turn()

Y añadir estas líneas para que se ejecute el servidor:

```
if __name__ == "__main__":
    bot = Bot()
    bot.RCW_start()
```

Figura 13. Ejemplo de inicio

En la biblioteca hay funciones para mover, atacar, crear, curar y obtener las casillas que están en el rango de un soldado. Con estas funciones el alumno tendrá que controlar al bot y evitar que los soldados realicen acciones que le hagan perder el turno como, por ejemplo, intentar mover un soldado a una posición ocupada.

Para evitar que si uno de los bots se queda bloqueado, bloquee también al del equipo contrario, se ha hecho que cada turno se ejecute simultáneamente en 2 hilos distintos (1 por cada equipo) y se han creado 2 semáforos binarios de C++, de forma que se hace un release() de uno de ellos antes de iniciar una partida y cada equipo intenta hacer durante 2 segundos un acquire() del semáforo del equipo contrario, logrando así que se ejecute por turnos mientras ningún equipo sobrepase los 2 segundos. Si un equipo sobrepasa en algún momento los 2 segundos, pero no está bloqueado, los turnos empezarán a ejecutarse a la vez en los dos equipos. Si un equipo se queda bloqueado, el otro irá ejecutando sus turnos con 2 segundos de espera entre ellos.

También se ha desarrollado una landing page (Figura 14) para la web y un sistema inicial de crear usuarios (Figura 15) e iniciar sesión (Figura 16) en la página. Los

usuarios se almacenan en una tabla de SQLite que contendrá el nombre y el sha256 de la contraseña.

Para mantener la sesión se han utilizado cookies y un unordered_map de C++ que almacena el usuario con su cookie (user => session_id); de esta forma solo podrá haber una cookie de sesión a la vez para cada usuario. Esto se ha hecho así para evitar tener que implementar algo que limpiara ese unordered_map que mantiene las sesiones. De esta forma, el tamaño máximo sería el del número de usuarios, mientras que si hubiera sido (session_id => user), se podrían mantener varias sesiones a la vez para un usuario, pero el unordered_map podría crecer de forma descontrolada.

Como el servidor que se utiliza es http, para que la contraseña de los usuarios no se envíe en texto plano, se ha hecho que antes de enviarse se haga un sha256 mediante JavaScript y sea ese hash lo que se envíe al servidor.

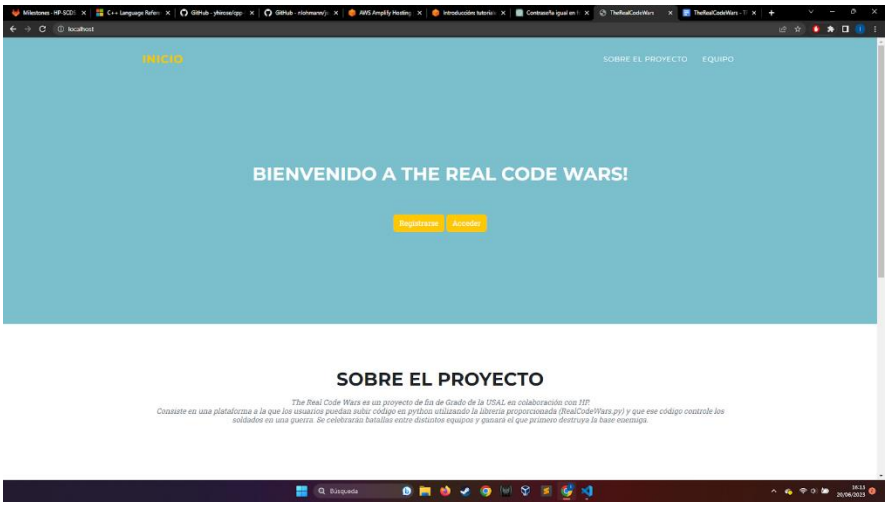


Figura 14. Landing page

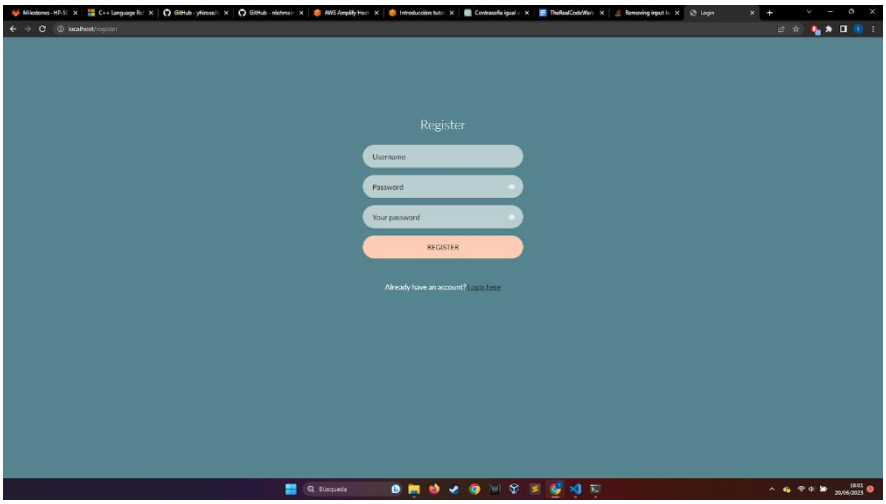


Figura 15. Página de Registro

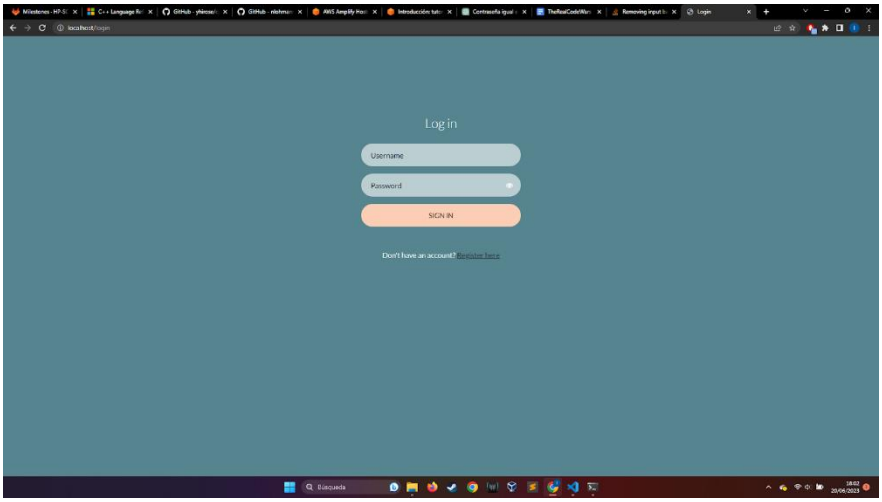


Figura 16. Página de iniciar sesión

En el **Sprint 9** se ha añadido la acción de curar y cambiado la interfaz del juego para que se vea la vida de los soldados en forma de barras de vida, haciendo así que se distingan mejor los soldados de un equipo y de otro, como se puede ver en la Figura 17.

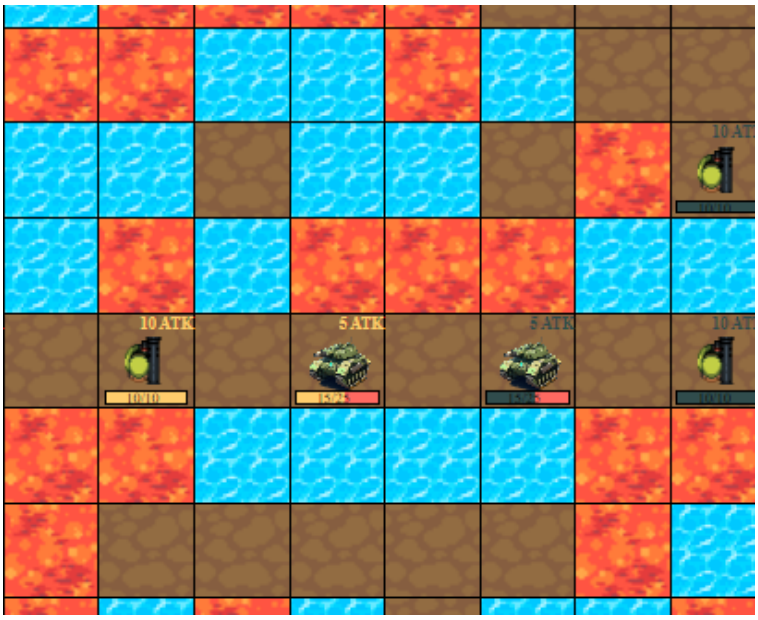


Figura 17. Interfaz mejorada

También se han añadido a la biblioteca de Python las clases Soldier y Tile, que antes eran un array, para que sea más intuitiva de usar dentro de un IDE, ya que este mostrará sus atributos.

En el **Sprint 10** se ha hecho una página principal para los usuarios (Figura 18), desde donde podrán ver si hay una partida en curso, y acceder a verla o acceder a una página para administrar su código (Figura 19) o a otra para crear una partida (Figura 20).

También se ha añadido en la misma una opción de Log out, ya que, si se accedía a la página de iniciar sesión y había cookies de sesión válidas, esta redirige directamente a la página principal del usuario y no permite iniciar sesión con otro usuario.

Figura 18. Página principal del usuario

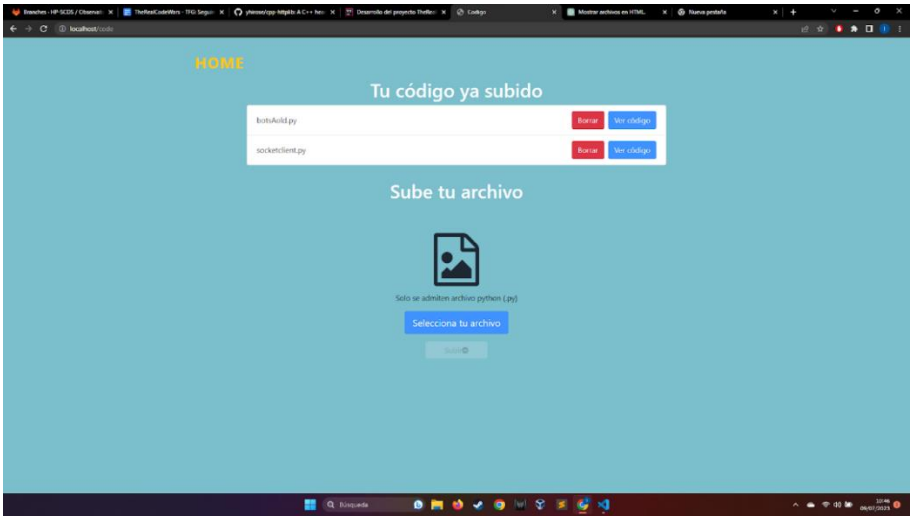


Figura 19. Pagina para administrar tu código

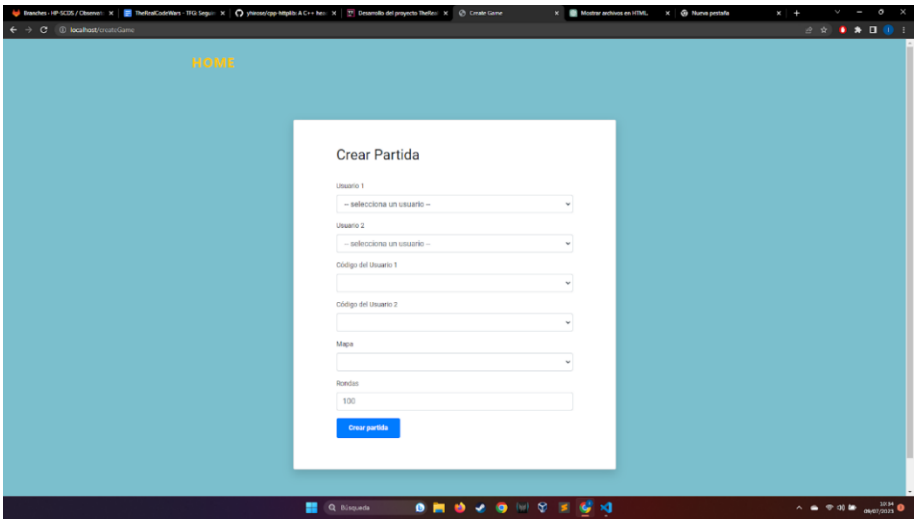


Figura 20. Página de crear partida

En todas las páginas se ha añadido un botón de Home para regresar a la página principal.

En la página de crear partida, mediante JavaScript se hacen peticiones al cargar la página al servidor para obtener el listado de usuarios y que se pongan como opciones en Usuario 1 y Usuario 2.

Se hace lo mismo para obtener el listado de mapas.

Luego, con el evento change de JavaScript, cuando cambie el valor de Usuario 1, se lanzará una petición al servidor que devolverá un JSON con los nombres de los ficheros que tenga en el directorio correspondiente al usuario seleccionado (sus bots subidos al servidor) y se pondrán como opciones en Código de Usuario 1. Esto se hace igual con el Usuario 2.

Aquí ha aparecido el siguiente problema, al cargar la página, se autoseleccionaba un usuario, pero no lo detectaba como evento change, con lo que no se generaban las opciones de bots correspondientes. Para solucionarlo, se ha hecho que los listados de Usuario 1 y 2 tengan la opción por defecto de *–selecciona un usuario–*, la cual una vez se cambie, no puede volver a ser seleccionada. Con dicha opción se fuerza a que el usuario cambie el valor al menos una vez, con lo que se solventa el problema.

Para ejecutar la partida, se copia el código seleccionado en el formulario del directorio del usuario al directorio de runningGame, en el cual está la biblioteca de TheRealCodeWars.py necesaria para la ejecución.

Por último, en el **Sprint 11** se ha añadido un tutorial de uso a la página web (*Figura 21*). Es bastante largo, con lo que no se puede poner una imagen de toda la página. En él se explica el juego, los tipos de soldados y de casillas y el objetivo. Luego se explica un poco la biblioteca, las clases que tiene, sus atributos y métodos y se proporciona un botón de descarga para la misma y para un bot sencillo de ejemplo. Por último, se explica cómo subir código y cómo crear una partida.

Figura 21. Tutorial en la web.

También se ha cambiado un poco la interfaz del juego (*Figura 22*), se ha añadido información de qué usuario es cada equipo y el bot que está usando, y cambiado algunos colores para su mejor visualización.

Para terminar, se han cambiado los atributos de las clases Soldier y Tile en la biblioteca de Python para que no se puedan modificar sus valores.

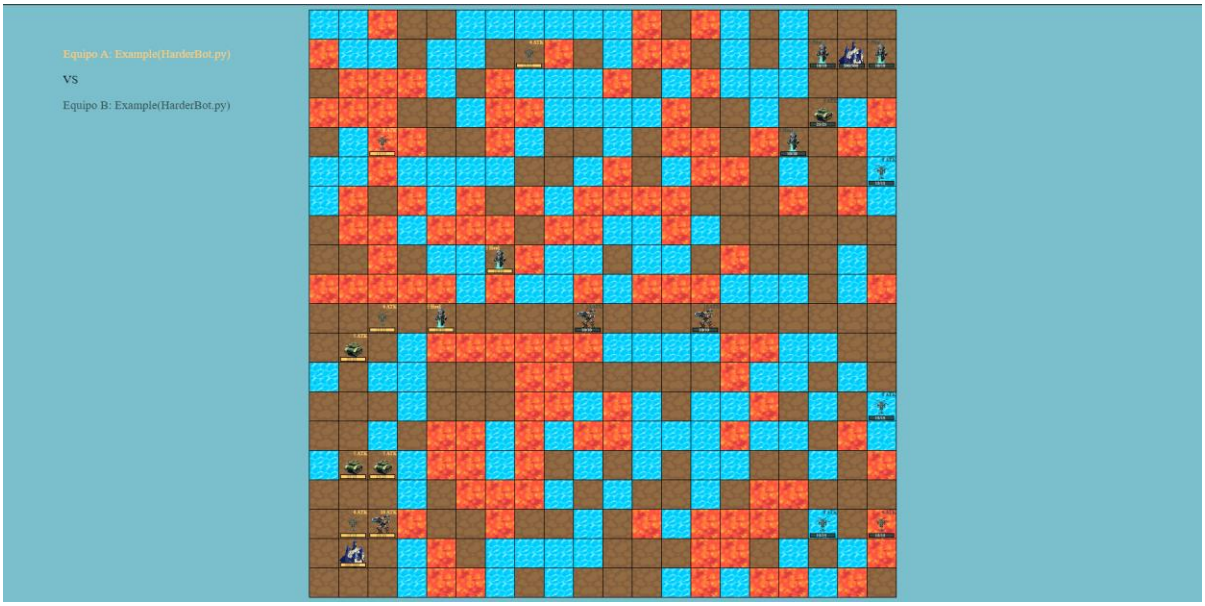


Figura 22. Interfaz final.

6. Trabajos relacionados

Como ya se ha mencionado, este proyecto coge inspiración del battlecode del MIT, el cual está escrito en Java y se rehace cada año.

Este es bastante más complejo que lo que se ha llegado a desarrollar, ya que en él hay también distintos recursos que controlar y usar.

En Steam (plataforma de videojuegos para ordenadores) salió el 26 de junio Code Rivals, el cual tiene características comunes, pero este es en un juego en 3D y se programa con bloques, una forma más fácil y visual para que las personas sin conocimientos previos puedan utilizarlo también.

También existió otro juego parecido, Bot Land, en el que se programaban los bots de forma similar al Code Rivals ya mencionado, pero el proyecto fue abandonado y cerrado en 2020.

7. Líneas de trabajo futuras

En cuanto a trabajo futuro, la interfaz web es muy mejorable, se ha hecho lo más rápido y funcional que se ha podido con los pocos conocimientos y tiempo de los que se disponía, pero la parte de visualización del juego es complicada de entender si nadie explica lo que está pasando. Sería muy aconsejable añadir estadísticas del número de soldados vivos de cada equipo, la ronda en ejecución y las rondas totales para facilitar la comprensión de lo que sucede en la partida. Además, como el objetivo es hacer torneos entre los participantes, se podría añadir una opción que permitiera la creación automática de torneos.

También el backend del juego es quizás demasiado simple, solo hay 4 acciones: mover, atacar, curar y crear. El único límite existente para crear soldados es que las bases solo puedan hacer una acción por turno. Podrían incorporarse recursos que recoger de alguna forma y que sean necesarios para crear unidades. Esto sin embargo podría ser contraproducente, ya que, al ser el público objetivo jóvenes con pocos conocimientos, puede que sea mejor mantener el juego simple. Otra opción para limitar los soldados podría ser poner un límite de soldados por equipo, ya sean vivos o totales.

Si se llegase a publicar en un servidor, la seguridad sería un tema muy preocupante, ya que la única comprobación que se hace del fichero que suben los usuarios y que se va a ejecutar en el servidor es que sea un *.py* (formato de Python). Esto significa que cualquier fichero de Python se podrá ejecutar en el servidor si no utiliza ningún paquete que requiera su instalación.

Bibliografía

Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). 1.1 Competitive Programming. En *Competitive programming 3* (p. 1). Lulu Independent Publish.

Kar, S. (s. f.). AI and Gaming: A new level of gaming; a new interest in AI. *Computer Science/Information Systems, The College of St. Scholastica, Duluth*, *55811*.

Mañà Marín, F. (2019). *Jupyter notebooks as a development and documentation tool for supporting computer programming learning among adolescents: A case study in a k-18 school*.

Videojuego de estrategia por turnos. (2022, 30 de noviembre). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 02:14, noviembre 30, 2022 desde https://es.wikipedia.org/w/index.php?title=Videojuego_de_estrategia_por_turnos&oldid=147639435.

Videojuego de estrategia en tiempo real. (2023, 19 de agosto). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 10:49, agosto 19, 2023 desde https://es.wikipedia.org/w/index.php?title=Videojuego_de_estrategia_en_tiempo_real&oldid=153176571.