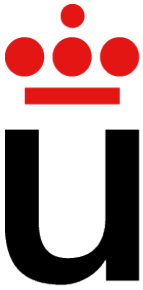

Universidad
Rey Juan Carlos

GRADO EN INGENIERÍA EN TELEMÁTICA

Curso Académico 2021/2022

Trabajo Fin de Grado

GAMIFICACIÓN DE PLATAFORMA UNIBOTICS

Autor : Daniel Hervás Rodao

Tutor : José María Cañas Plaza

Co-Tutor : David Roldán Álvarez

Trabajo Fin de Grado

Gamificación de la Plataforma Unibotics

Autor : Daniel Hervás Rodao

Tutor : José María Cañas Plaza **Co-Tutor :** David Roldán Álvarez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 202X, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 202X

Dedicado a

mi familia / mi abuelo / mi abuela

II

Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Índice general

1. Introducción **1**

1.1. Robótica 2

1.2. Componentes robóticas 4

1.2.1. Middlewares robóticos 5

1.2.2. Simuladores robóticos 5

1.3. Robótica educativa 6

1.3.1. Tecnologías Web 8

1.3.2. Estructura del documento 9

2. Objetivos **11**

2.1. Objetivos 11

2.2. Metodología 12

2.3. Plan de trabajo 12

3. Herramientas **15**

3.1. Lenguaje JavaScript 15

3.1.1. Caractrerísticas del lenguaje 16

3.1.2. Librería jQuery 16

3.2. Lenguaje HTML 17

3.3. Hojas de estilo CSS 18

3.4. Django para servidores web 19

3.5. WebRTC 20

3.5.1. Protocolos 21

3.5.2. Establecimiento de la conexión 22

3.5.3. Candidatos ICE 24
3.6. Control de versiones 24

4. Juegos Asíncronos **27**
4.1. Infraestructura 27
4.2. Follow Line Game 29
4.2.1. Diseño 29
4.2.2. Modo teleoperado 33
4.2.3. Evaluador automático 35
4.2.4. Carga de código 36
4.3. Drone Cat Mouse Game 37
4.3.1. Diseño 37
4.3.2. Evaluador automático 39
4.3.3. Carga de código 39

5. Juegos Compartidos Síncronos **41**
5.1. Objetivo 41
5.2. Follow Line Game Síncrono 41
5.2.1. Diseño 41
5.2.2. Elección de oponente 42
5.2.3. Carga flexible de código 44
5.2.4. Ejecución compartida 44
5.2.5. Transmisión de vídeo y datos vía WebRTC 45

6. Experimentos y validación **49**

7. Resultados **51**

8. Conclusiones **53**
8.1. Consecución de objetivos 53
8.2. Aplicación de lo aprendido 53
8.3. Lecciones aprendidas 54
8.4. Trabajos futuros 54

A. Manual de usuario **55**

Bibliografía **57**

Índice de figuras

1.1. Brazo robótico. 3
1.2. Coche autónomo. 3
1.3. Robot DaVinci. 4
1.4. Robot ATRIAS. 4
1.5. Robot militar. 4
1.6. Simulador Gazebo. 6

3.1. Documento básico de *HTML*. 17
3.2. Comparativa entre no usar CSS y usarlo. 19
3.3. Protocolo STUN. 22
3.4. Protocolo TURN. 22
3.5. Diagrama de intercambio de candidados ICE. 24
3.6. Gráfico de.actividad del repositorio *noVNC* de GitHub. 25

4.1. Arquitectura de Unibotics. 28
4.2. Barra conexión con RADI. 29
4.3. Barra de control de la simulación. 29
4.4. Plantilla del juego Follow Line. 30
4.5. Selector de circuitos. 32
4.6. Cambio de circuito. 33
4.7. Selector de dificultad. 33
4.8. Teleoperación apagada. 34
4.9. Teleoperación encendida. 34
4.10. Listener del botón de teleoperación. 34

4.11. Key Handler. 35
4.12. Evaluador automático. 36
4.13. Plantilla Drone Cat Mouse. 37
4.14. Selector de dificultad Drone Cat Mouse. 39
4.15. Evaluador Drone Cat Mouse. 39

5.1. Diagrama de conexión WebRTC establecida. 47

Capítulo 1

Introducción

El TFG que será descrito a continuación se ha desarrollado en la plataforma *Unibotics* de la asociación *JdeRobot*[1], orientado al aprendizaje de robótica para estudiantes universitarios. El principal motivo de este proyecto es la introducción de técnicas de gamificación para los diferentes ejercicios contenidos la plataforma, así como añadir nuevos.

En este capítulo se introducirá el contexto en el que se desarrolla el trabajo, así como los motivos relevantes para a llevarlo a cabo. Puesto que el campo de la robótica es muy amplio, en concreto, este TFG se centra en el marco de la robótica educativa, destinada a la enzeñanza de la misma, buscando fomentar el desarrollo, la independencia y la lógica a edades tempranas. Esta manera de enseñanza consiste en la metodología *learning-by-doing*, un sistema que facilita a los niños el aprendizaje aspectos científicos y matemáticos, y también ayuda en el incremento del pensamiento lógico y creativo de los mismos.

Existen diversas formas de iniciación a la robótica, estas son, mediante libros o juguetes. Relizando talleres de programación. También existen plataformas para la iniciación a la robótica, como por ejemplo, *Lightbot Jr* [2], *Scratch Jr* [3], *Kibotics* [4], etc.

[1] `https://jderobot.github.io`
[2] `https://lightbot.com/`
[3] `https://www.scratchjr.org/`
[4] `https://kibotics.org/`

1.1. Robótica

Los avances en computación de las últimas décadas han sido el impulso que ha permitido la creación de máquinas con comportamientos muy cercanos a los de los seres humanos. La robótica está muy relacionada no solo con la rama de la ingeniería, si no, que involucra conocimientos de matemáticas y física imprescindibles en el desarrollo de estas máquinas. Uno de los objetivos principales de la robótica es el de facilitar tareas de la vida diaria al ser humano, incluso, en algunas ocasiones, sustituir al ser humano.

En 1950 la robótica experimenta un gran desarrollo. Esto se debe a los grandes avances en relación a la potencia y complejidad computacional. El acoplamiento mecánico empezó a sustituirse por sistemas eléctricos. Tal es el grado de desarrollo que se empiezan a generar sistemas de control automático consistentes en máquinas de estado secuencial.

La motivación que existe en la robótica de facilitar algunas tareas que realizan los seres humanos en su día a día, viene dada por algunos casos en los que las personas realizan trabajos muy repetitivos (p.e. cadena de montaje), trabajos peligrosos (p.e. desactivación de una bomba). Entre estos, también surge con el motivo de mejorar la calidad de vida, como por ejemplo, con el objetivo de monitorizar personas mayores que viven solas, y si es necesario, avisar a un médico (*Misty II* [5]).

Estas máquinas, requieren de un *software* que proporcione la capacidad de aprender ciertas tareas previamente mencionadas, de manera que puedan ser más eficientes. Este *software* es conocido con el nombre de inteligencia artificial, muy ligada al campo de la robótica. Los avances en este campo permiten desarrollar sistemas capaces de tener una cierta memoria útil para realizar una serie de funciones.

Un claro ejemplo de este gran impluso es la implementación de robots en la industria automovilística, capaces de realizar tareas repetitivas, y que conllevan un gran riesgo para las personas (Figura 1.1).

[5] `https://isocial.cat/es/innovacio/misty-ii-robot-para-mejorar-la-calidad-de-vida-de-l`

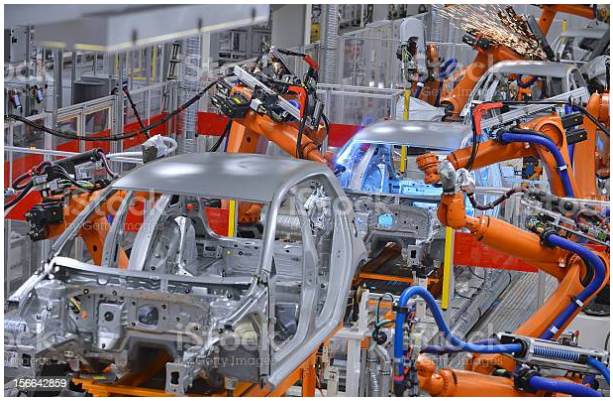

Figura 1.1: Brazo robótico.

En la actualidad, cada vez son más populares los coches autónomos. Es un campo muy amplio que cuenta con un gran número de posibilidades. Alguna de estas posibilidades serían los coches con conducción autónoma de *Tesla* (Figura 1.2) o los coches con aparcamiento autónomo que están desarrollando un gran número de compañías en la actualidad.

Figura 1.2: Coche autónomo.

Otro campo de la aplicación de la robótica es la medicina, donde existen robots capaces de filtrar las vibraciones naturales del humano para proporcionarle una gran precisión y seguridad, un claro ejemplo es el robot DaVinci (Figura 1.3). Adicionalmente, hay robots capaces de mantener una estabilidad la estabilidad necesaria para caminar sobre dos piernas robóticas, como es el robot ATRIAS (Figura 1.4).

Figura 1.3: Robot DaVinci.

Figura 1.4: Robot ATRIAS.

En el ámbito militar, existen robots capaces de sustituir a una persona en el a la hora de realizar tareas de gran peligro como la desactivación de bombas y la entrada en zonas contaminadas (Figura 1.5).

Figura 1.5: Robot militar.

1.2. Componentes robóticas

Todo robot está formado por dos componentes: el *software*, encargado de proporcionar la inteligencia al robot, el más importante, y, el *hardware* encargado de proporcionar la estructura física del robot.

Con al gran auge de la robótica han surgido numerosas plataformas que proporcionan herramientas que simplifican el desarrollo de software robótico, esto son los denominados *middlewares* robóticos.

Durante el desarrollo de software robótico es preciso realizar una serie de pruebas para

comprobar el funcionamiento del código y depurar errores, por lo que se necesitan simuladores que nos proporcionen un entorno cercano a la realidad previa al ensamblado del robot.

1.2.1. Middlewares robóticos

Un *middleware* robótico es un *framework* que proporciona una serie de herramientas que facilitan el desarrollo de software para robots. Proporciona los servicios necesarios para soportar y simplificar aplicaciones complejas y distribuidas. Para el control de los sensores y actuadores de los robots, los *middlewares* proporcionan *drivers*, APIs, etc.

OpenRDK es un proyecto de código abierto para el desarrollo de módulos poco acoplados. Proporciona una gestión de concurrencia, comunicación entre procesos y una técnica de enlace que permite el diseño de sistemas conceptuales de puertos de datos de entrada / salida. También proporciona módulos para realizar conexiones con simuladores robóticos y controladores de robots genéricos. También existen algunos otros *middlewares* como *Micro* y *Orca*.

El *middleware* robótico más generalizado es ROS[6] (*Robotics Operating System*). *Robotics Operating System* fue desarrollado en 2007 por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte a sus proyectos. A pesar de no ser un sistema operativo, ROS proporciona servicios como la abstracción *hardware*, mecanismos de comunicación entre prosesos, el control de dispositivos de bajo nivel y el mantenimiento de paquetes. *Robotics Operating System* fue desarrollado para sistemas UNIX, aunque en la actualidad está siendo adaptado para su funcionamiento en sistemas operativos como Fedora, Mac OS X, Arch, Gentoo, OpenSUSE, Slackware, Debian o Microsoft Windows.

1.2.2. Simuladores robóticos

Los simuladores robóticos surgen con la necesidad de realizar pruebas durante el desarrollo del *software* para la detección y depuración de posibles errores antes de llevarlo a un robot real debido al gran coste que suponen.

Uno de los simuladores más utilizados en la actualidad es *Gazebo*[7]. Su popularidad se debe a su robusto motor de físicas, sus gráficos de alta calidad y su amplio catálogo de robots y

[6] `https://www.ros.org/`
[7] `http://gazebosim.org/`

escenarios. Es una herramienta de código abierto integrada con ROS, por lo que permite ejecutar *software* robótico en un escenario simulado (Figura 1.6).

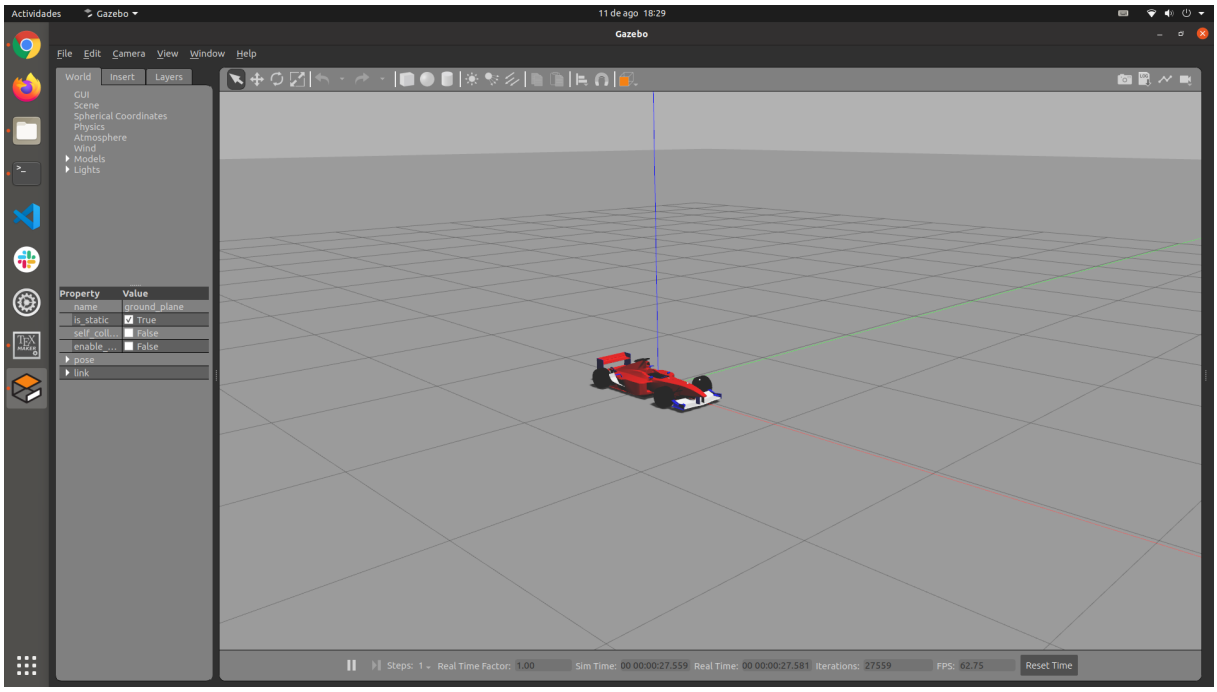

Figura 1.6: Simulador Gazebo.

1.3. Robótica educativa

La robótica educativa proporciona a los estudiantes la infraestructura para la construcción y programación de un robot, pero, además de la enseñanza robótica, estos entornos van más allá, ofreciendo la capacidad para el alumno de adquirir un pensamiento lógico. También, contribuye en la adquisición de una mentalidad resolutiva y al enriquecimiento de la cultura científica de los alumnos. Este método de educación con la robótica como objeto de enseñanza se denomina el método STEAM (Science, Technology, Engineering, Arts and Matemathics).

Como se ha comentado, para llevar a cabo este método de educación es necesaria una infraestructura, como alguna plataforma cuyo objetivo sea el de la enseñanza de la robótica, algunas de estas plataformas son las siguientes:

- **Robocode:** es un juego de programación en *Java* o *.NET*, cuyo objetivo es programar la lógica de un robot que es un tanque de batalla, para combatir contra otros tanques. Esta

plataforma ofrece una comunidad que incluye uuna Wiki y un grupo de google. Además, ha realizado una competición propia [8].

- **CodeCombat:** ofrece un aprendizaje basado en juegos, tiene el objetivo de que los estudiantes aprendan mientras juegan. Los estudiantes aprenderán *JavaScript*, *Python*, *HTML* y *CoffeeScript*, además de aprender los fundamentos de la informática. El contenido se divide en 11 unidades: tres unidades de desarrollo de juegos, dos unidades de desarrollo web y seis unidades de informática.

- **Gearsbot:** un entorno de programación de robots mediante bloques, que además también ofrece la posibilidad de programar la lógica en *Python*, usa un simulador web que emplea la tecnología *WebGL* [9] para renderizar gráficos.

- **Riders.ai:** una plataforma de programación de robots que ofrece la posibilidad de realizar competiciones idividuales como en grupos. Es accesible desde cualquier lado del mundo. También ofrece al usuario una capa de abstracción sobre los sensores, localización y diferentes partes del robot.

- **TheConstruct:** ofrece un entorno de programación de robots que pueden moverse por la Luna, Marte y más planetas. No requiere la instalación de ROS. Ofrece un gran extra que le diferencia de los demás, esto es, brinda la posibilidad al usuario de poder ejecutar su código en robots reales de forma remota, y, también proveen certificados al completar diferentes cursos.

Dentro de la robótica educativa, es preciso enfatizar la plataforma *Unibotics* en la que se desarrolla el presente proyecto. Esta plataforma es un proyecto internacional que ofrece material para la enseñanza de robótica en las aulas. *Unibotics* proporciona una infraestructura software en conjunto a una colección de ejercicios, cada uno con el material teórico correspondiente para su resolución.

[8] `https://gamesfleadh.ie/robocode/`

[9] `https://es.wikipedia.org/wiki/WebGL`

1.3.1. Tecnologías Web

Las diferentes tecnologías web permiten crear interfaces de usuario y establecer las comunicaciones con el servidor, además de implementar comportamientos de la web en el servidor. Se pueden distinguir en tecnologías *backend* (del lado del servidor) y tecnologías *frontend* (del lado del usuario).

Una gran ventaja de estas tecnologías es que ofrecen la ausencia de la necesidad de instalar componentes de software, configuración y dependencias adicionales. Únicamente es necesario disponer de un navegador de internet y una conexión a internet, pues todos los archivos necesarios se servirán desde el lado del servidor.

Estas tecnologías facilitan el uso al cliente, pues evitan que el usuario tenga que intalar actualizaciones, pues las actualizaciones se realizan en el lado del servidor. Esta ventaja elimina posibles fallos debidos a incompatibilidades entre las versiones, pues el servidor distribuye una única version para todos. También ofrecen la ventaja de aplicaciones web multiplataforma, únicamente es necesario el desarrollo en un único entorno (*Windows*, *Ubuntu*, *MacOS*, etc), pues estas tecnologías son comunes a los navegadores de todos los sistemas. Únicamente es necesario conocer *HTML*, *CSS* y *JavaScript*, el navegador será el encargado de interpretar dichos lenguajes.

Las aplicaciones web dependen de una conexión entre el usuario y el servidor, por lo que, una de las desventajas que estas es la latencia que esta conexión pueda generar. Estas tecnologías siempre son algo más lentas que otros lenguajes compilados como *C#* puesto que el navegador no realiza ninguna compilación, si no que interpreta el código línea a línea.

En el lado del servidor, existen diversos *frameworks* que facilitan la vida al desarrollador. Algunos ejemplos de ellos serían, *Express* para el lenguaje *JavaScript* y *Django* escrito en *Python*.

El desarrollo de tecnologías web está sujeto a un gran crecimiento. Cada vez las aplicaciones web son más potentes y eficientes, de manera que cada vez son más los desarrolladores que optan por este modelo de desarrollo.

Las tecnologías web ofrecen la posibilidad de que varios usuarios puedan programar robots y ejecutarlos en el mismo escenario, hacer competiciones tanto síncronas como asíncronas, algo que resultaría muy complejo si se realizase en local. *WebRTC* nos ofrece esta posibilidad, pues es una herramienta de código abierto que proporciona conexiones en tiempo real entre *peers*

mediante un API.

1.3.2. Estructura del documento

En esta sección se va a hacer un breve resúmen de los contenidos de cada capítulo, también se mencionará como están estructurados.

En el capítulo 2, Objetivos, se explica cuál es el propósito o los propósitos de este trabajo. También se mencionará la planificación temporal que se ha llevado a cabo durante el trabajo. Y finalmente, se mencionará la heramienta empleada para realizar un control de versiones.

Posteriormente, en el capítulo 3, Herramientas, se habla exclusivamente sobre las herramientas y tecnologías utilizadas. Serán descritas indicando sus ventajas, inconvenientes y áreas de uso.

A continuación, en el capítulo 4, Juegos Asíncronos, se explicará cómo se ha realizado el diseño de los mismos, tanto para la IU como para la infraestructura interna.

Seguidamente, en el capítulo 5, Juegos Síncronos, de igual manera que en la sección anterior, se explicará como se ha desarrollado la infraestructura necesaria para elaborar el ejercicio.

Finalmente, en el capítulo 6, tenemos los experimentos realizados y sus resultados, además, en el capítulo 7, se encuentran las conclusiones obtenidas de haber realizado este trabajo, y, en último lugar la bibliografía que contiene todos los contenidos empleados para el desarrollo del trabajo.

Capítulo 2

Objetivos

En este capítulo se explicarán los objetivos que se pretenden alcanzar en este TFG, los requisitos para los problemas que se van a plantear, la planificación y metodología seguida durante todo el desarrollo.

2.1. Objetivos

El objetivo general de este trabajo es la introducción de juegos en la plataforma *Unibotics*, en los que varios usuarios puedan competir síncrona y asíncronamente para fomentar su aprendizaje. Este objetivo principal se ha dividido en dos subobjetivos.

- La introducción de **juegos competitivos asíncronos** que permitan a los usuarios competir contra una serie de dificultades proporcionadas por la plataforma, en el mismo escenario y en tiempo real.

- La introducción de **juegos competitivos síncronos** que proporcionan una visión más competitiva y social de la plataforma a los usuarios, que pueden conectarse simultáneamente con sus amigos para competir entre ellos.

Las soluciones proporcionadas deben cumplir los objetivos previamente mencionados, además de los siguientes requisitos para una correcta implementación:

- El código del lado del servidor debe ser desarrollado en *HTML*, *CSS3* y *JavaScript*.

- No debe requerir instalaciones de software adicionales, el usuario unicamente ha de tener el contenedor docker *RADI*, instalado en la máquina local.

2.2. Metodología

Para llevar a cabo este trabajo, se ha empleado una metodología basada en *sprints* [1]. Este modelo de desarrollo se corresponde con un modelo de desarrollo de software ágil, consiste en dividir un proyecto en tareas más sencillas y de menor duración, de forma que cada cierto período de tiempo (1-2 semanas) puedan presentarse resultados. Esta metodología es muy útil pues, es susceptible a cambios por parte del cliente.

Para la implantación de esta metodología se han establecido reuniones semanales. En dichas reuniones se ha comentado los cambios que se han introducido en el *sprint* previo para proponer cambios, mejoras y fijar una serie de objetivos para la siguiente semana.

2.3. Plan de trabajo

Para el seguimiento de los objetivos previamente mencionados, se ha seguido una serie de etapas que permiten el avance incremental en la implementación del trabajo.

- **Estudio de la plataforma Unibotics:** Como una primera toma de contacto es preciso el estudio de la infraestructura que se está empleando en la plataforma. Además, se ha realizado la puesta en funcionamiento del *despliegue local* [2], el contenedor docker (*RADI*) y la comprobación de que todos los ejercicios funcionan de manera correcta. También, se ha estudiadio el funcionamiento de las platillas empleadas por los ejercicios y la conexión con el *RADI*.

- **Desarrollo de los juegos asíncronos:** Una vez entendidos los conceptos pricipales de la plataforma, se ha procedido al desarrollo de los juegos asíncronos. Se han diseñado nuevas plantillas para los nuevos ejercicios asícronos, y, añadido nuevos aspectos, como evaluadores automáticos.

[1] `https://www.bbva.com/es/metodologia-scrum-que-es-un-sprint/`

[2] Despliegue de un clon de la plataforma en un entorno local con el fin de realizar cambios y probar nuevas funcionalidades, previo a incorporar los cambios a la plataforma oficial

- **Estudio de la tecnología WebRTC:** Una vez completada la parte de los juegos asíncronos, se procede al estudio de estas tecnologías que nos permiten la retransmision de audio y vídeo entre dos navegadores web. Se han desarrollado algunos pequeños programas de prueba, como un chat de texto y vídeo, para asentar los conceptos.

- **Desarrollo del juego síncrono:** Con los conceptos de WebRTC asentados, se ha procedido a diseñar en el webserver de *Unibotics*, un sistema de señalización entre los usuarios, de manera que se pueda intercambiar la información necesaria para realizar la conexión WebRTC. Finalmente, en las plantillas del ejercicio, se incluye código *JavaScript* que se encarga de establecer las conexiones RTC para cada extremo.

Capítulo 3

Herramientas

En este capítulo se hará una breve presentación de todas las herramientas empleadas para el desarrollo del presente TFG. Estas tecnologías se pueden englobar en dos grupos, las tecnologías Front-End dedicadas a la presentación y a la interfaz de usuario (JavaScript, HTML y CSS), tecnologías Back-End dedicadas al servidor (Django), y, por último, tecnologías WebRTC que serán las encargadas de transmitir el vídeo en los juegos síncronos.

3.1. Lenguaje JavaScript

Se trata de un lenguaje de programación interpretado, su estándar es *ECMAScript*[1], basado en *Java* y *C*. Fue creado para aplicaciones web del lado del cliente. Es interpretado en el navegador web y permite mejoras en la interfaz de usuario, además de páginas web dinámicas. También puede usarse en el lado del servidor utilizando *Node.js*, un entorno de ejecucción de JavaScript construido con el motor *JavaScript V8*.

En este proyecto se ha empleado JavaScript en el lado del cliente. La lógica de las plantillas que usan los ejercicios ha sido programada usando *ECMAScript-6*, así como los evaluadores automáticos de los mismos, y, adicionalmente, para los *WebSockets* encargados de comunicarse con el servidor para realizar tareas de señalización, o de envío de mensajes. Sus principales características son:

[1]Especificación de lenguaje de programeción que define un leguaje de tipos dinámicos y soporta programación orientada a objetos basada en prototipos.

3.1.1. Caractrerísticas del lenguaje

- Se trata de un lenguaje del lado del cliente, es decir, ejecuta en la máquina del propio cliente a través de un navegador.
- Tipado débil, por lo que no es necesario especificar el tipo de dato al declarar una variable permitiendo que una misma variable pueda adquirir distintos tipos durante la ejecucción.
- De alto nivel, con lo que significa que su sintaxis es fácilmente comprensible. Esta sintaxis se encuentra alejada del lenguaje máquina.
- Es un lenguaje interpretado puesto que utiliza un intérprete que traduce las líneas de código a lenguaje máquina en tiempo de ejecucción.
- Es un leguaje orientado a objetos, ya que utiliza clases y objetos como estructuras.

Adicionalmente, junto con *ES-6* se ha empleado una librería llamada *jQuery*[2] que permite agregar dinamismo a un sitio web permitiendo interactuar con los elementos HTML, manipular el DOM, manejar eventos, diseñar animaciones y agregar interacción con la técnica *AJAX*[3] (*Asynchronous JavaScript and XML*).

3.1.2. Librería jQuery

jQuery es una librería multiplataforma desarrollada en JavaScript, inicalmente desarrollada por John Resig. Esta librería permite simplificar en gran medida la interacción con los documentos HTML y sus estilos, manipular el DOM, manejar eventos, crear animaciones, y agregar la integración de la técnica *AJAX* (*Asynchronous JavaScript and XML*).

En código, el constructor de jQuery puede llamarse empleando el nombre *jQuery*, o, también empleando el alias *$*.

Esta librería hace posible a desarrolladores que se inician en el mundo del desarrollo de interfaces de aplicación, puedan desarrollar una interfaz gráfica más elaborada sin la necesidad de tener amplios conocimientos de CSS.

[2] `https://jquery.com/`

[3] `https://developer.mozilla.org/es/docs/Web/Guide/AJAX`

3.2. Lenguaje HTML

HTML (*HyperText Markup Languaje*) es un lenguaje de marcado que permite indicar la estructura un documento utilizando etiquetas. Es utilizado para la creación de documentos electrónicos que se envían a través de la red. Los documentos pueden tener conexiones con otros a través de *hipervínculos*.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Hola mundo!</title>
    </head>
    <body>
        <p>Hola mundo!</p>
    </body>
</html>
```

Figura 3.1: Documento básico de *HTML*.

Primeramente se debe declarar el tipo del documento *HTML* mediante la línea *DOCTYPE html* que indica que es un documento *HTML-5*. El elemento *html* engloba el documento *HTML*, dentro de este elemento se encuentran dos etiquetas:

- *HEAD* es la cabecera del documento, que contiene la información general (metadatos) acerca del documento, incluyendo el título y los enlaces a scritps y hojas de estilos.

- *BODY* es el cuerpo del documento *HTML* donde se encuentran las etiquetas que dan formato al mismo. Puede contener imágenes, enlaces, vídeos, menús, formularios, botones, además de animaciones, que se pueden crear dentro de un elemento canvas.

Con la incorporación de *HTML-5* se han introducido diversas novedades y mejoras que son de interés para este trabajo:

- *WebSockets*, es una tecnología que hace posible abrir una comunicación entre el navegador y el servidor.

- *WebRTC* (*Web Real-Time Communications*) es una tecnología que permite a las aplicaciones web capturar y transmitir audio y vídeo entre navegadores sin necesidad de un intermediario.

- Se añade un mejor soporte de contenido multimedia sin la necesidad de instalar plugins adicionales. Mediante el elemento *video* la página reproducirá de manera nativa el contenido.

- Proporciona un elemento, *canvas* que permite renderizar escenas gráficas en la web mediante el uso de *JavaScript*.

3.3. Hojas de estilo CSS

CSS o *Cascading Stylesheet* es un lenguaje empleado para dar formato y estilo a un documento de texto, comunmente, las instrucciones se agrupan en archivos con extensión ***.css**, lo que permite que para una página web se puedan almacenar estilos por separados dependiendo del fin de cada uno, de esta manera, se podrán importar en cada documento únicamente las instrucciones de las que se va a hacer uso.

Con respecto a lo último mencionado, puesto que estas hojas almacenan el estilo, CSS nos ofrece una gran ventaja, la separación de la estructura del documento (HTML) de su representación. Aunque es posible añadir instrucciones CSS en el *head* de un documento HTML mediante la etiqueta *style*, esto no es una buena práctica.

Cascading Stylesheet nos permite utilizar *JavaScript* para si fin principal, el de dar dinamismo e inteligencia a una página web, etc. Por lo que mejora en gran manera el rendimiento de la página.

A continuación se muestra un ejemplo del uso de este lenguaje.

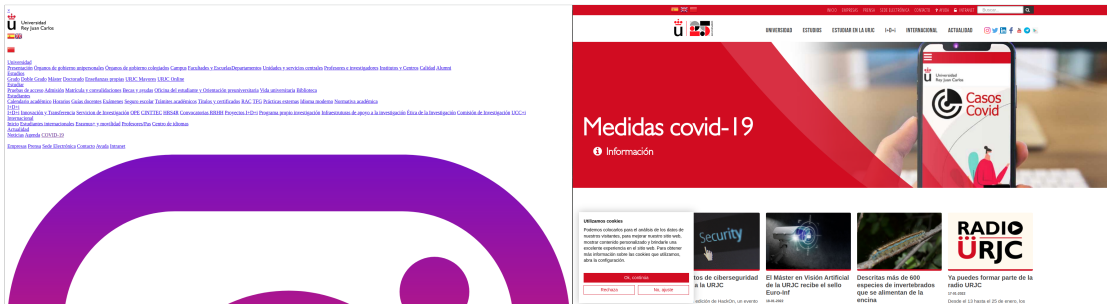

Figura 3.2: Comparativa entre no usar CSS y usarlo.

Como se puede apreciar en la imagen 3.2, hay un gran salto de no usar CSS a usarlo en los documentos HTML. Este lenguaje nos permite tamto crear un *Navbar*, menús desplegables, barras de búsqueda, botones personalizados, sub-apartados, es decir, aumenta en gran medida la experiencia del usuario.

CSS también permite la creación de páginas web responsivas, esto es, que la web se adapte al tamaño de la pantalla del usuario, redimensionando los elementos y recolocándolos. Esto es muy útil si se quiere desarrollar una aplicación multiplataforma.

3.4. Django para servidores web

Django [4] [3] es un framework de alto nivel que facilita el desarrollo de sitios web seguros y sostenibles. Fue lanzado como un framework web genérico en 2005, bajo una licencia de código abierto. Django se encarga de las complicaciones del desarrollo web, para que el usuario pueda centrarse en escribir su aplicación.

El objetivo de Django es la creación sencilla de sitios web complejos, poniendo énfasis en la reutilización, la conectividad y la extensibilidad de componentes, el desarrollo rápido y el principio de no repetir código (*Don't Repeat Yourself*).

Algunas de las carácterística de Django son:

- Un mapeador objeto-relacional, es capaz de convertir datos entre un sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional.

- Aplicaciones incorporables en cualquier página gestionada con Django.

[4] `https://es.wikipedia.org/wiki/Django_(framework)`

- Una API de base de datos robusta.
- Un sistema incorporado de ”vistas genéricas”que ahorra el tener que escribir la lógica de ciertas tareras comunes.
- Un sistema extensible de plantillas con herencia.
- Un despachador de URLs basado en expresiones regulares.
- Un sistema *middleware* para desarrollar caracteríscas adicionales.
- Soporte de internacionalización, incluyendo traducciones de la interfaz de administración.
- Documentación incorporada accesible a través de la aplicación administrativa.

3.5. WebRTC

WebRTC [5] [4] (Web Real-Time Communication) es un proyecto libre y de código abierto que proporciona una comunicación en tiempo real (RTC) a través de una serie de APIs. Permite que la comunicación de audio y vídeo funcione dentro de las páginas web al permitir la comunicación *Peer to Peer*, sin necesidad de instalación de plugins y sin necesidad de la intervención del servidor. WebRTC tiene soporte en los navegadores Safari, Chrome, Firefox, Mozilla y Opera, está estandarizado por el World Wide Web Consortium [6] y por el Internet Engineering Task Force [7].

El API de WebRTC tiene una serie de interfaces principales que son clave en el dearrollo del software para este proyecto:

- **RTCPeerConnection** [8] representa una conexiñon entre una máquina local y un par remoto. Esta interfaz provee métodos para: conectar un equipo remoto, mantener y monitorizar esa conexión y cerrar la conexión.

[5] https://es.wikipedia.org/wiki/WebRTC

[6] https://es.wikipedia.org/wiki/World_Wide_Web_Consortium

[7] https://es.wikipedia.org/wiki/Grupo_de_Trabajo_de_Ingenier\%C3\%ADa_de_Internet

[8] https://developer.mozilla.org/es/docs/Web/API/RTCPeerConnection

- **RTCDataChannel** [9] representa el canal que puede ser empleado para la transmisión de datos *Peer to Peer*.

- **MediaDevices** [10] proporciona el acceso a los dispositivos multimedia conectados, como webcams y micrófonos, y, también para compartir la pantalla. Ofrece un método *getUserMedia()*, que, con el permiso del usuario, enciende la cámara, obtiene la imagen de la pantalla, el audio del micrófono, y, proporciona un *MediaStream* que contiene pistas de vídeo y/o de audio del dispositivo.

- **MediaStream** [11] representa el flujo de contenido multimedia. Un flujo consiste de varias pistas, ya sean de audio o de vídeo.

3.5.1. Protocolos

- **ICE** [12] (*Interactive Conectivity Establishment*). Es una técnica empleada para permitir que un navegador web se conecte con otro/s navegador/es web mediante conexiones *Peer to Peer*. Debe de poder pasar por un *firewall*, dirección IP pública, y, transmitir los datos a través de otro servidor, si el *router* no permite las conexiones entre pares. Para lograr esta serie de casos, se emplean servidores STUN y TURN.

- **STUN** [13] (*Session Transvelsal Utilities for NAT*). Es un protocolo de descubrimiento de IP pública, para determinar cualquier restricción en el *enrutador* que impida una conexión directa con un par. El cliente enviará una solicitud STUN en Internet que responderá con la dirección pñublica del cliente y si el cliente esta accesible detrás del NAT del enrutador.

- **NAT** [14] (*Network Address Translation*). Es empleado por el *router* para asignar una dirección IP pública. El *router* tiene acceso a la dirección pública de cada dispositivo y cada dispositivo conectado a este, tendrá una dirección IP privada. Las solucitudes se traducen de la IP privada a la IP pública con un puerto único. Algunos *enrutadores* tienen restricciones sobre quien puede conectarse a su red. Esto puede dar lugar a que aunque

[9] `https://developer.mozilla.org/en-US/docs/Web/API/RTCDataChannel`
[10] `https://developer.mozilla.org/es/docs/Web/API/MediaDevices`
[11] `https://developer.mozilla.org/en-US/docs/Web/API/MediaStream`
[12] `https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment`
[13] `https://en.wikipedia.org/wiki/STUN`
[14] `https://en.wikipedia.org/wiki/Nat`

tengamos un única dirección IP pública encontrada por el servidor STUN, no se pueda establecer una conexión. Ante este caso, se debe recurrir a un servidor TURN.

- **TURN** [15] (*Transversal Using Relays around NAT*). Algunos router emplean una técnica llamada *NAT simétrica*. Esto es, el *enrutador* solo acepta conexiones de pares a los que se haya conectado previamente. TURN está destinado a aludir esta resticción de NAT simétrica. Consiste en establecer una conexión con un servidor TURN que retransmite todo el tráfico que se le envía, de una máquina a otra. Este intercambio es gestionado usando ICE.

  El proceso de *Offer/Answer* es realizado cuando se establece una nueva conexión, o bien, cuando debe cambiar un aspecto de una conexión ya establecida. A continuación se enumeran los pasos que ocurren durante el intercambio de la oferta y la respuesta:

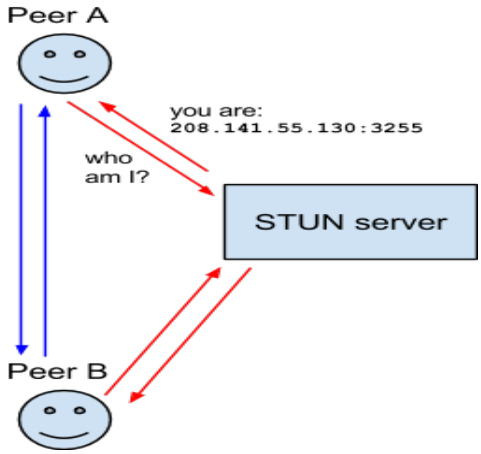

Figura 3.3: Protocolo STUN.

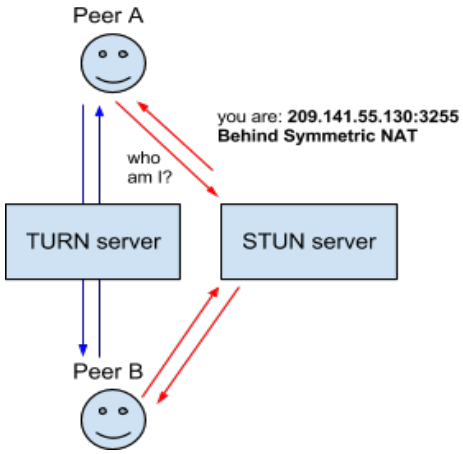

Figura 3.4: Protocolo TURN.

- **SDP** (*Session Description Protocol*) [16]. Es un protocolo empleado para describir el contenido multimedia de una conexión.

3.5.2. Establecimiento de la conexión

El protocolo WebRTC se encarga de establecer conexiones entre pares, pero desafortunadamente, una conexión WebRTC no se puede establecer sin un servidor intermedio. Este servidor

[15] https://en.wikipedia.org/wiki/Traversal_Using_Relays_around_NAT
[16] https://en.wikipedia.org/wiki/Session_Description_Protocol

podríamos llamarlo **servidor de señalización**, empleado en el intercambio de información previo al establecimiento de la conexión WebRTC.

La información que necesitamos intercambiar es una *Offer* y una *Answer* que contienen el *Session Description Protocol* mencionado anteriormente en 3.5.1.

El *Peer A* que inicia la conexión va a crear una oferta (*Offer*). Esta *Offer* será enviada al *Peer B* mediante el servidor intermedio seleccionado. *Peer B* recibirá esta *Offer* desde el servidor de señalización y creará una respuesta (*Answer*) que enviará haciendo uso del mismo intermediario al *Peer A*.

1. *Peer A* captura los medios locales mediante *MediaDevices.getUserMedia*
2. *Peer A* crea una conexión *RTCPeerConnection* y emplea el método *RTCPeerConnection.addTrack()* para añadir el flujo de datos a la comunicación.
3. *Peer A* crea la *Offer* mediante el método *RTCPeerConnection.createOffer()*.
4. *Peer A* establece el SDP de su *Offer* llamando al método *RTCPeerConnection.setLocalDescription()*.
5. *Peer A* solicita al servidor STUN que genere los candidatos ICE.
6. *Peer A* usa el servidor de señalización para transmitir su *Offer*.
7. *Peer B* recibe la oferta y emplea *RTCPeerConnection.setRemoteDescription()* para almacenar el SDP de *Peer A*.
8. *Peer B* hace la configuración necesaria relativa a su conexión, como obtener su flujo de datos multimedia.
9. *Peer B* crea una respuesta haciendo uso del método *RTCPeerConnection.createAnswer()*.
10. *Peer B* establece su descripción local llamando a *RTCPeerConnection.setLocalDescription()* y pasando la respuesta por parámetro.
11. *Peer B* envía la respuesta usando el servidor de señalización.
12. *Peer A* recibe *Answer*.

13. *Peer A* almacena el SDP de *Peer B* llamando al método *RTCPeerConnection.setRemoteDescription()*. Ahora *Peer A* y *Peer B* tienen la configuración de ambos y el flujo de medios comienza a transmitirse.

3.5.3. Candidatos ICE

Además de intercambiarse la información sobre los elementos multimedia, los *peers* deben intercambiar información sobre la conexión. Esto es conocido como *candidatos ICE*, que detallan los métodos disponibles con los que un extremo puede comunicarse (directamente o mediante un servidor TURN). Cada *peer* propondra una lista con sus mejores candidatos primero, estos candidatos son UDP (puesto que es más rápido, pues no hay retransmisiones, ni recuperación frente a pérdidas), aunque también permite emplear candidatos TCP. Esta comunicación es transparente para el usuario.

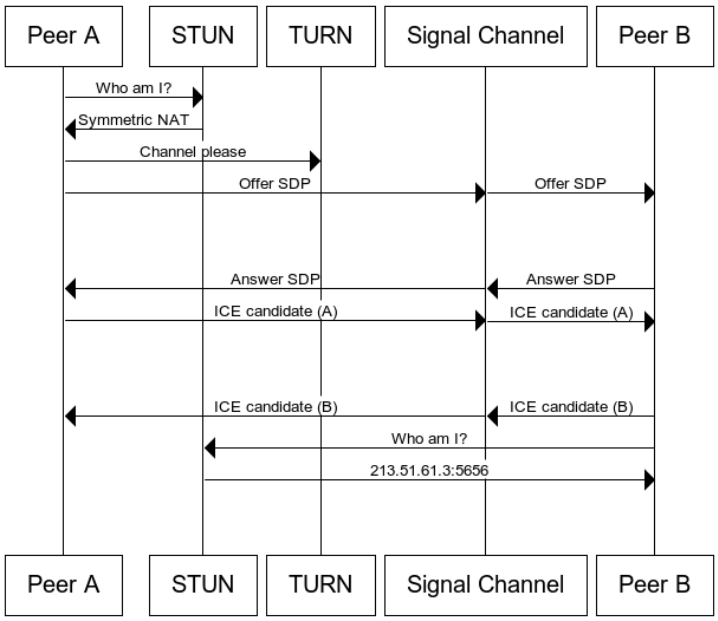

Figura 3.5: Diagrama de intercambio de candidados ICE.

3.6. Control de versiones

En todo desarrollo de software es necesaria una herramienta de control de versiones que facilite la gestión del código fuente. Estos softwares realizan un control de todas las modifi-

caciones, de manera que si se comete un error en un determinado instante, se puede volver a una versión anterior del código. La herramienta empleada para el control de versiones en este proyecto es *GitHub*.

Esta herramienta permite clonar cualquier repositorio en nuestra máquina local y mantenerlo actualizado frente a cambios que se hagan en el extremo remoto. También podemos contribuir en este código realizando *pull requests*. Es una herramienta muy potente, lo que le hace ser una de las herramientas de control de versiones más utilizadas por la comunidad de desarrolladores.

GitHub proporciona un interfaz gráfico que nos permite, de manera gráfica, ver la actividad de un repositorio, lo que nos permite saber si un proyecto tiene una gran cantidad de desarrolladores.

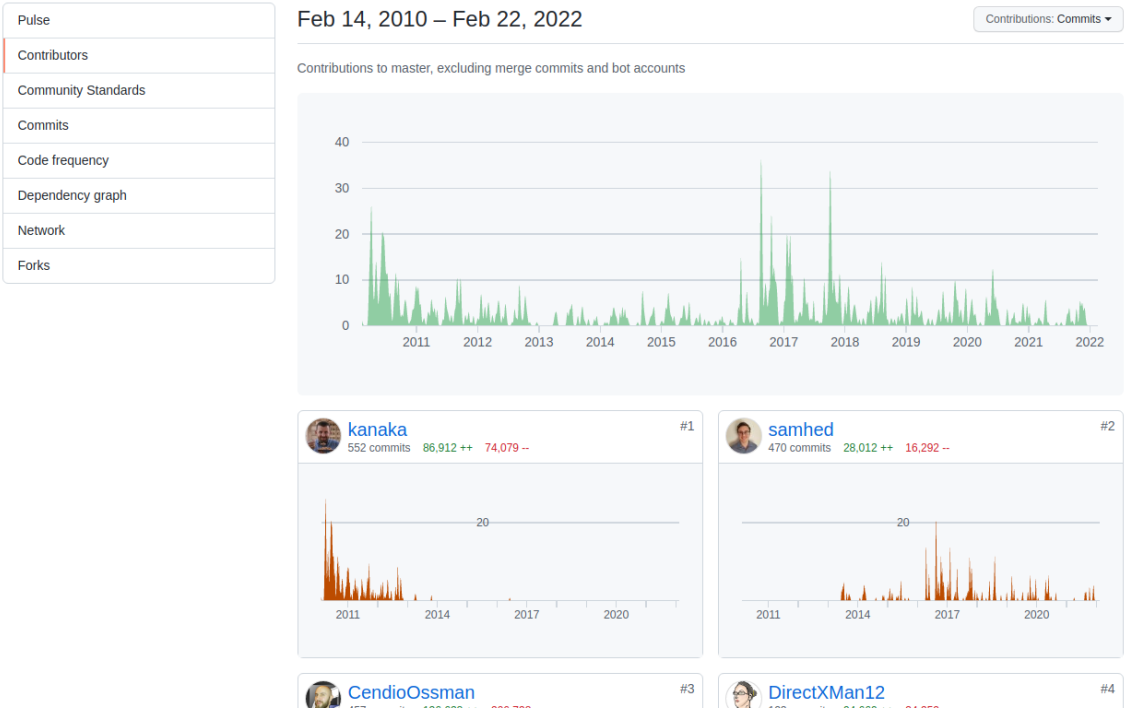

Figura 3.6: Gráfico de.actividad del repositorio *noVNC* de GitHub.

En este proyecto se ha trabajado en una serie de repositorios diferentes:

- **unibotics-webserver:** aquí se encuentran los recursos del webserver Django.
- **unibotics-exercises:** este repositorio contiene las platillas de los ejercicios de la plataforma.

- **RoboticsAcademy:** este repositorio es una colección *open-source* de algunos retos, que se han empleado para diseñar ejercicios para la plataforma de *Unibotics*.
- **CustomRobots:** este repositorio contiene todos los robots empleados en las simulaciones.
- **noVNC:** se ha realizado un *fork* del repositorio oficial puesto que para la versión del ejercició síncrono se necesitaba hacer algunas modificaciones para enviar el flujo de vídeo al extremo remoto.

La metodología que se ha seguido para incorporar nuevos cambios a los disferentes repositorios es la siguiente. En primer lugar se crea una *issue* el el repositorio correspondiente que describa el problema que se va a resolver. A continuación, se crea una rama a partir de la *master* con el nombre de *issue-XXX* donde se incorporarán todos los cambios mencionados en la *issue* que se creó. Finalmente, una vez se han realizado todos los cambios y se ha verificado que funciona de manera correcta, se realiza un *pull request*, esto es, una solicitud de incorporación de los cambios a la rama principal. Las *pull request* son revisadas por otro compañero, que se encargará de hacer la pruebas pertinentes e incorporarla a la rama *master*.

Capítulo 4

Juegos Asíncronos

En este capítulo se expondrán los *juegos asícronos* desarrollados para la plataforma de *Unibotics*. Se han desarrollado un total de dos juegos asíncronos, de los que se hablará desde el diseño de las plantillas de la interfaz de usuario, hasta la comunicación con el servidor y el contenedor docker donde es ejecutado el juego. Para llevar a cabo el desarrollo de estos juegos, se ha partido de las versiones ya existentes dentro de la plataforma llamados *retos*.

4.1. Infraestructura

Para la realización de los siguientes ejercicios, trabajado sobre la infraestructura inicial sobre la que opera Unibotics. Esta arquitectura es formada por tres componentes, un *Webserver*, el *navegador* del usuario y el *RADI* (contenedor Docker donde se ejecutan los ejercicios). A continuación, una descripción del trabajo que realiza cada componente:

- **Webserver:** este componente se encarga de proporcionar al navegador del usuario, los componentes del *Front End* que van a ser necesarios para llevar a cabo la ejecución del ejercicio en cuestión. Proporciona las plantillas **HTML** y el código **JavaScript**. También se encarga de almacenar y traer el código del usuario desde **AWS** (*Amazon Web Services* [1]), con el fin de tener una persistencia del código del mismo.

- **Navegador:** ejecuta en la máquina del usuario, es el encargado de comunicarse con el RADI y mostrar toda la información relacionada con la simulación al usuario. Este com-

[1] `https://es.wikipedia.org/wiki/Amazon_Web_Services`

ponente muestra las plantillas de los ejercicios recibidas desde el Webserver al usuario. Mediante *listeners* (escritos en **JavaScript**) sobre distintos elementos de la plantilla proporciona los controles sobre la ejecución del ejercicio (Play, Reset, Load Code, Gazebo, etc.). Además de proporcionar los controles sobre el ejercicio, también debe de transmitir estas órdenes al contenedor donde se aloja el ejercicio, de manera que durante el establecimiento de la conexión con este, en el caso de los juegos con dos vehículos, se establecen cinco WebSockets. Estos son, un máster, por el que se envían los comandos relacionados con el lanzamiento del ejercicio y Gazebo, y un dos WebSockets por cada vehículo, el primero *ws_code*, empleado para el envío del código al contenedor, se encarga de interactuar con el cerebro del robot, y, por útlimo, *ws_gui* cuya función es mostrar toda la información relativa a la interfaz de usuario (mapa al estilo f1, autoevaluador, imágenes de las cámaras), es decir, recibe toda la información del robot.

- **RADI:** es uno de los componentes con más peso, pues es donde se ejecuta la simulación. Este ofrece una serie de elementos, *ws_manager*, *ws_code* y *ws_gui* mencionados anteriormentes, y adicionalmente, ofrece la visualización de Gazebo mediante *GZClient* [2] y de la consola del contenedor, ambos por medio de un servidor *noVNC* [3] alojado localmente en el contenedor, al cual se conectará el navegador para mostrar dichos elementos.

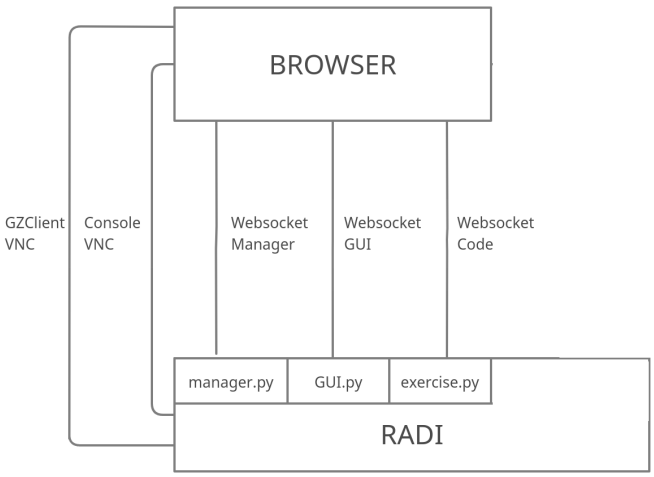

Figura 4.1: Arquitectura de Unibotics.

[2] http://manpages.ubuntu.com/manpages/bionic/man1/gzclient.1.html
[3] https://novnc.com/info.html

4.2. Follow Line Game

4.2.1. Diseño

Para desarrollar este ejercicio ha sido necesario implementar *software* en el lado del del cliente especialmente. Se ha diseñado un interfaz de usuario que radica en la simplicidad para el usuario, sin proporcionar un exceso de controles. La plantilla de los ejercicios es común a la mayoría de los ejercicios de la plataforma, pero mediante JavaScript se acondiciona la platilla a los requisitos del ejercicio. Todos los ejercicios tienen en común una barra en la parte superior que contiene una serie de botones para realizar la conexión con el contenedor Docker, que es donde se ejecutará todo lo relativo a la simulación del ejercicio.

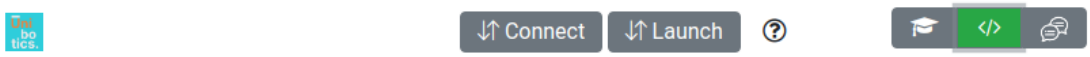

Figura 4.2: Barra conexión con RADI.

Volviendo a la parte específica para este ejercicio, tenemos una barra donde se encuentran los controles relativos al control de la simulación, visualización, carga de código e información adicional sobre la simulación.

Figura 4.3: Barra de control de la simulación.

Bajo las barras de control del ejercicio, se tiene una pantalla que está dividida en dos secciones. La primera sección contiene el selector de circuito, el selector de dificultad y el editor de código. En la segunda sección se encuentra toda la información que se recibe del escenario, esto es, las localizaciones de los coches y sus cámaras, el evaluador automático, el simulador Gazebo y la consola del contenedor.

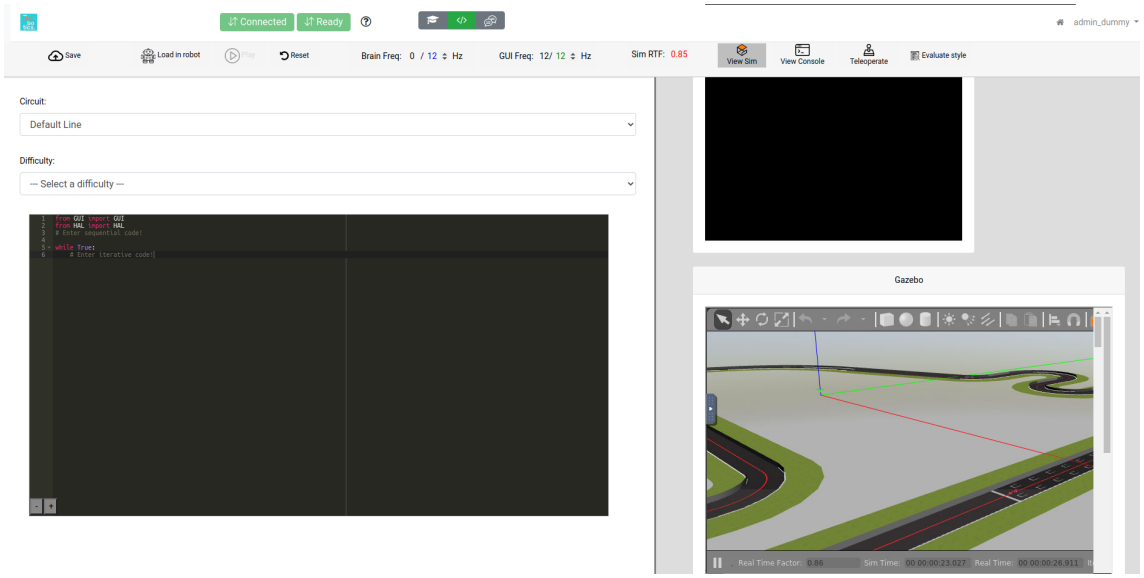

Figura 4.4: Plantilla del juego Follow Line.

Robot adversario

Ha sido necesario añadir un nuevo modelo de robot para que el usuario pueda diferenciar entre su robot y el adversiario. Con lo que se ha modificado el modelo del robot del usuario cambiándolo de color a verde. Esta modificación se ha llevado a cabo mediante *Blender* [4] y se ha llamado *f1_guest*.

Una vez creado el nuevo modelo, se ha añadido a los ficheros de declaración de los escenarios de Gazebo, incluyendo el nuevo modelo en una posición del circuito en la que ambos coches se encuentran a la misma distancia. Los ficheros empleados para la configfuración del escenario tienen la siguiente forma:

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
    <gui fullscreen=1></gui>
    <scene>
      <grid>false</grid>
      <sky>
        <clouds>
```

[4] https://www.blender.org/

```
        <speed>12</speed>
      </clouds>
    </sky>
  </scene>
  <!-- A global light source -->
  <include>
    <uri>model://sun</uri>
    <name>sun_1</name>
    <pose>0 0 1 0 0 0</pose>
  </include>
  <include>
     <uri>model://simple_circuit</uri>
     <pose>0 0 0 0 0 0</pose>
  </include>
  <include>
    <uri>model://f1</uri>
    <pose>53.462 -10.734 0.004 0 0 -1.57</pose>
  </include>
  <include>
    <uri>model://f1_guest</uri>
    <pose>-53.2 -6.8 0.004 0 0 -4.85</pose>
  </include>
 </world>
</sdf>
```

Inicio del juego

Para llevar a cabo la iniciación del juego es necesario realizar previamente la conexión con el contenedor Docker (*RADI*) donde se va a llevar a cabo toda la ejecución del ejercicio. Una vez realizada esta conexión, se deberá seleccionar el circuito en el que el usuario quiera jugar, seleccionar la dificultad del oponente, y, pulsar el botón *Load in robot*, una vez se hayan cargado los respectivos códigos, se podrá proceder pulsando el botón *Play* para comenzar la ejecución.

Con respecto a las versiones iniciales del reto *Follow Line* de **Unibotics**, se ha tenido que implementar un cambio dentro del RADI, pues para era necesario poder lanzar el escenario con el circuito seleccionado por el usuario.

Para implementar esta característica, dentro del fichero *manager.py* (encargado del lanzamiento de los ejercicios, comprobar el código mediante PyLint, noVNC, etc), se ha añadido una variable con scope global que contiene un array con todos los ejercicios que requieren el cambio de circuito. De manera que, antes de lanzar un ejercicio se comprueba si es uno de estos, y, si es así, el circuito que se ha obtenido del comando de lanzamiento que llega vía *WebSockets* se emplea para formatear la cadena de texto que contiene la instrucción de lanzamiento del proceso. Esta instrucción de lanzamiento contendrá el nombre del fichero que se corresponde con el modelo del circuito deseado por el usuario, de manera que Gazebo ya sabe qué escenario cargar.

Finalmente, se lanzará este proceso (además de algunos otros como noVNC para mostrar GZClient y la consola), y se avisará al *Front End* que la conexión está lista.

Selección de circuito

El usuario podrá elegir mediante un desplegable uno de los circuitos disponibles para realizar la ejecución del juego, estos circuitos son *Default*, *Montmelo*, *Montreal* y *Nürbugring*.

Figura 4.5: Selector de circuitos.

El circuito *Default* será el escenario por defecto en el que comenzará la simulación, si el usuario selecciona otro, se le comunicará al contenedor Docker, y este se encargará de volver a lanzar la simulación con el escenario seleccionado. Esta comunicación se realiza por medio de WebSockets.

```
// Kill actual sim
startSim(2);

// Stop Code
stopCode();

// StartSim
startSim(1, circuit);
```

Figura 4.6: Cambio de circuito.

Como se puede apreciar en la figura anterior, el código del selector de circuitos se encarga de matar la simulación en curso e iniciar la nueva simulación con el circuito que se obtiene del selector.

Selección de dificultad

El selector de circuitos brinda al usuario un total de tres dificultades diferentes, *Easy*, *Medium* y *Hard*, adicionalmente, el usuario también podrá seleccionar una cuarta dificultad, llamada *Teleoperate* que consiste en que podrá teleoperar el coche del oponente con las teclas de dirección.

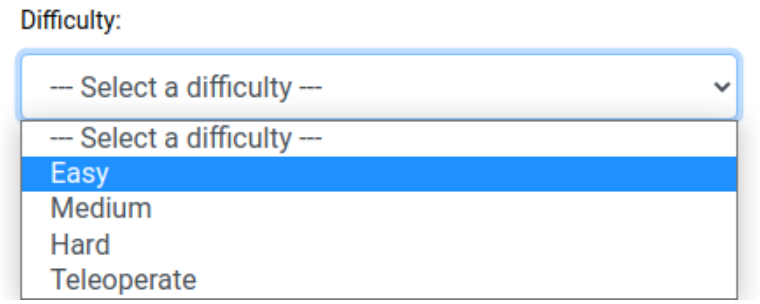

Figura 4.7: Selector de dificultad.

4.2.2. Modo teleoperado

En este juego se ha considerado el extra de implementar un modo de teleoperación para ambos vehículos, para brindar al usuario la posibilidad de probar su código ante diversas situaciones y pueda desarrollar un código más robusto, también con el objetivo de que pueda controlar alguno de los dos vehículos en el caso de que se haya quedado atascado. El usuario

podrá controlar su vehículo, mediante el botón de teleoperación situado en la barra superior, y el coche oponente seleccionando la dificultar *Teleoperate*, como se ha mencionado antes.

Figura 4.8: Teleoperación apagada.

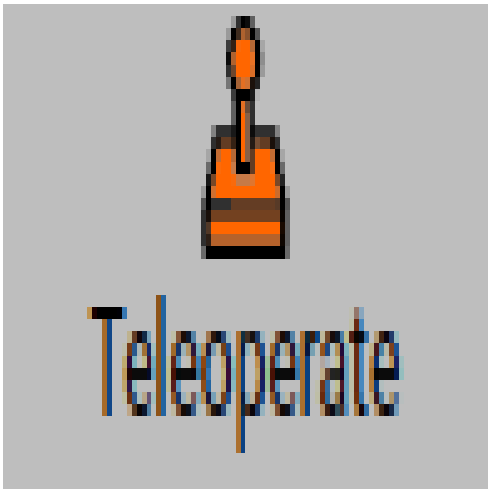

Figura 4.9: Teleoperación encendida.

Como se puede apreciar en las figuras, el botón de teleoperación del vehículo del usuario, cambiará de color en función de si la teleoperación está encendida o apagada.

Implementación del botón de teleoperación

Para llevar a cabo el control de la teleoperación se ha implementado mediante código JavaScript un *listener* para el botón de teleoperación, que se encarga de comprobar si se puede activar el modo, es decir, no se está teleoperando el otro vehículo, cambiar la imagen del botón de teleoperación y añadir el handler correspondiente a los eventos *keyup* y *keydown*.

```
teleop_btn.addEventListener('click', function(){
    // If variable is false, activate teleoperate mode
    if((!teleop_mode)&&(!teleop_mode_guest)){
        // Activate mode
        teleop_mode = true;
        teleop_img.src = "{% static 'unibotics-exercises/static/assets/exercises/follow_line_game/img/joystick_on.png' %}";
        document.addEventListener('keydown', keyHandler, false);
        document.addEventListener('keyup', keyHandler, false);
        return;
    }
    if((teleop_mode)&&(!teleop_mode_guest)){
        // Else, turn of teleop mode and remove event listeners
        teleop_mode = false;
        teleop_img.src = "{% static 'unibotics-exercises/static/assets/exercises/follow_line_game/img/joystick_off.png' %}";
        document.removeEventListener('keydown', keyHandler, false);
        document.removeEventListener('keyup', keyHandler, false);
        return;
    }
});
```

Figura 4.10: Listener del botón de teleoperación.

El handler que se le asigna a los eventos *keyup* y *keydown* se encarga de determinar cual es la tecla que se ha soltado o pulsado, y, si es una de las flechas de dirección establece un valor constante a la variable de velocidad o giro.

```
// Event Handlers for controls
function keyHandler(event) {
    // Right (39), Left (37), Down (40), Up (38)

    // First check if websocket_gui_guest and websocket_code_guest are defined
    if((typeof(websocket_gui) !== 'undefined' && (typeof(websocket_code) !== 'undefined'))){
        let cmd = "#tele";

        // Prevent using arrow keys to scroll page
        if([32, 37, 38, 39, 40].indexOf(event.keyCode) > -1) {
            event.preventDefault();
        }

        if(event.keyCode == 39) {
            //console.log('Right');
            w = (event.type == 'keydown')?-1:0;
        }else if(event.keyCode == 37) {
            //console.log('Left');
            w = (event.type == 'keydown')?1:0;
        }else if(event.keyCode == 40) {
            //console.log('Down');
            v = (event.type == 'keydown')?-2:0;
        }else if(event.keyCode == 38) {
            //console.log('Up');
            v = (event.type == 'keydown')?2:0;
        }
        console.log('v: ', v, 'w: ', w);
    }
}
```

Figura 4.11: Key Handler.

Como se puede apreciar en la figura 4.10,los valores de velocidad son asignados a las correspondientes variables de velocidad o de giro. Los valores de estas variables son enviados al contenedor Docker mediante el websocket por el que viaja el código desde el navegador al contenedor.

4.2.3. Evaluador automático

Este evaluador consiste en mostrar en tiempo real las distancias que hay entre los coches, con el fin de que el usuario pueda competir contra la plataforma. Para cada circuito se han establecido una serie de puntos (*checkpoints*) a lo largo del mismo, que sirven para saber en que lugar se encuentran los robots, y, con esta información poder calcular la distancia que le queda a cada robot para alcanzar al adversario. Estos *checkpoionts* son almacenados en una serie de *arrays*, y, en función del circuito que haya seleccionado el usuario se emplerá el correspondiente.

La información de posición se esta recibiendo con una determinada frecuencia en el tiempo a través del WebSocket *ws_gui*, este se encarga de actualizar las variables que contienen las posiciones de los robots. A continuación, se ha desarrollado código JavaScript que se encarga

de utilizar los valores que contienen estas variables de posición y calcula el checkpoint más cercano a la posición de cada robot y construye los caminos (*paths*) desde un robot hasta el otro. Una vez se obtienen los paths, se calcula la distancia total de estos y se muestra en la interfaz del usuario.

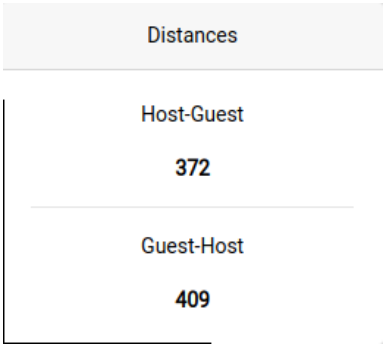

Figura 4.12: Evaluador automático.

4.2.4. Carga de código

Una vez se ha establecido la conexión con *RADI*, se puede proceder a la carga del código. Esta carga se realiza mediante un botón *Load in robot*, que una vez pulsado, se encarga de habilitar el botón *Play* después de haber realizado una comprobación exitosa del código del usuario.

Cuando el usuario procede a realizar la carga del código en los respectivos robots, el código que se ha desarrollado se obtiene del editor de código situado en la parte izquierda de la pantalla. Para realizar la carga de código de las distintas dificultades, se ha desarrollado un script en *JavaScript* que se encarga de realizar una solicitud al servidor web, pidiendo el código correspondiente con la dificultad seleccionada.

Para que el servidor web pueda proporcionar el código de cada dificultad al navegador web solicitante, se han creado una serie de usuarios con el prefijo de *bot*, que contienen el código correspondiente a cada dificultad. También, ha sido necesaria la creación de un nuevo *endpoint* en el servidor web.

Desde el lado del servidor web, cuando se recibe una solicitud en dicho *endpoint*, este se encarga de determinar cual es la dificultad y el ejercicio del que se está solicitando por medio de los datos aportados en el propio *endpoint*.

Finalmente, unz vez se tienen ambos códigos, estos se envían cada uno por el *WebSocket* que se corresponde con su robot.

4.3. Drone Cat Mouse Game

4.3.1. Diseño

Al igual que todos los juegos de la plataforma, estos comparten una plantilla *HTML* con los componentes básicos de un ejercicio, por lo que para el caso de este ejercicio, cabe mencionar que las barras superiores de conexión con *RADI* y de control de la simulacón son similares a las de **Follow Line Game**.

Bajo el navbar, a mano izquierda se encuentra el editor de código, y a mano derecha, los botones para el control de la ejecución del ratón, el evaluador automático, el simulador *Gazebo* y la consola del contenedor Docker.

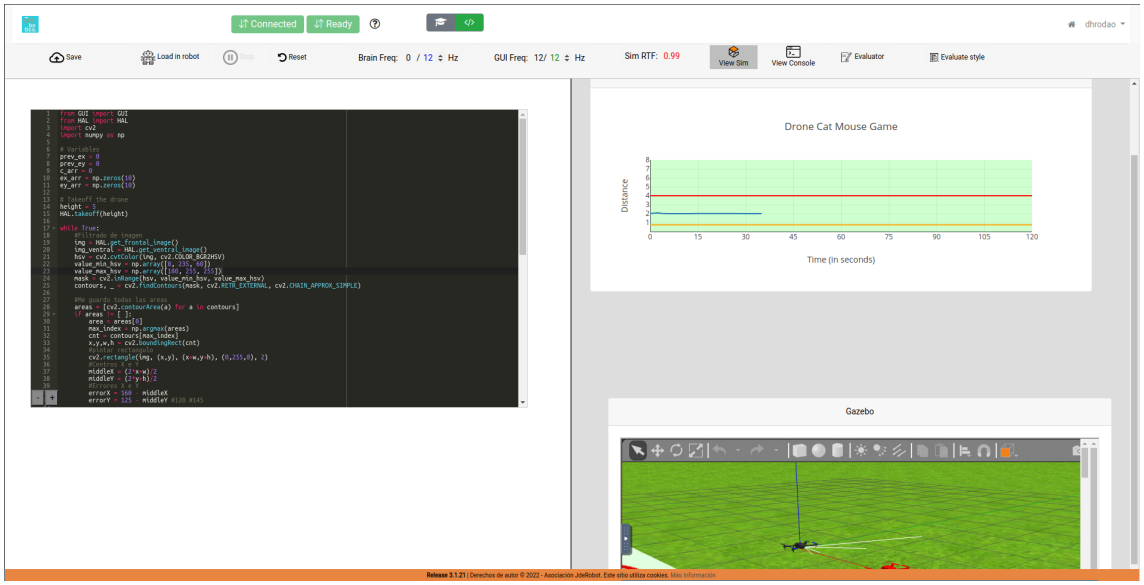

Figura 4.13: Plantilla Drone Cat Mouse.

Por otro lado, la plataforma Unibotics contaba con una versión del reto *Drone Cat Mouse* con una arquitectura en la que, internamente, el *ratón* era instanciado dentro del controlador del robot que se correspondía con el *gato*, y este, se controlaba desde el propio objeto del *gato*.

En la version desarrollada para este trabajo, se han introducido algunos cambios. En primer lugar, se han desacoplado los controladores para los robots, de manera que ahora cada uno es

independiente del otro. Esto da lugar a la necesidad de tener dos *WebSockets* para cada robot (inicialmente se tenían únicamente dos *WebSockets* para comunicarse con el *gato*, y este, se encargaba de trasladar la información al *ratón*), *ws_code* y *ws_gui* para el *gato* y el *ratón*.

En la versión inicial del ejercicio, el código del ratón, se encontraba almacenado dentro del *RADI*. Actualmente esto se ha cambiado, de manera que el código del *ratón* se pide al *webserver* especificando el ejercicio y la dificultad seleccionada. Este cambio se ha realizado con el fin de seguir el mismo paradigma que con los usuarios, el código de los mismos se recibe desde el *webserver*. Una vez recibido el código, este es enviado al *ratón* por medio del *WebSocket* correspondiente.

Inicio del juego

De igual manera que en el juego *Follow Line*, primeramente hay que realizar la conexión con *RADI*, que se encargará de lanzar el proceso del juego, y, una vez se haya establecido la conexión con los *WebSockets*, se avisa al *Front End* de manera que el usuario quede notificado. Posteriormente, puede elegirse la dificultad del ratón, cargar el código del usuario, mostrar *Gazebo* y la consola, y reanudar la simulación.

Selección de dificultad

El selector de dificultad del *ratón* está situado en la mitad suyperior derecha de la interfaz del usuario junto a los botones de control del mismo. El usuario debe seleccionar primeramente la dificultad deseada mediante el desplegable situado en la mitad derecha de la pantalla. Y, a continuación, pulsar el botón *Load in robot* para proceder a la carga de los códigos.

Para realizar la carga de códigos, primero se obtiene el código del editor de texto situado en la mitad izquierda de la platilla y se envía por el *WebSocket* empleado para comunicarse con el *gato*. A continuación, mediante una peticón HTTP al *WebServer*, se solicita el código del ratón (en función de la dificultad), y finalmente se envía por el *WebSocket* definido para la comunicación *RADI-Ratón*.

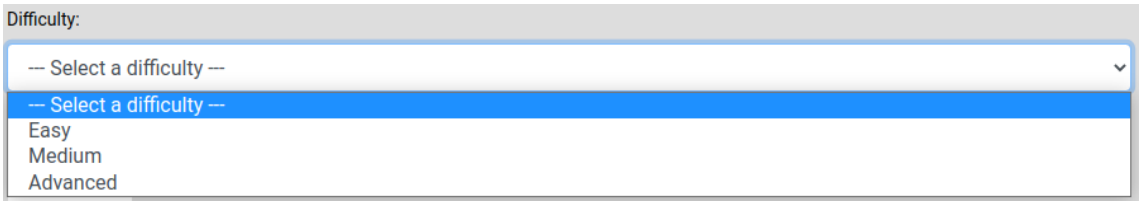

Figura 4.14: Selector de dificultad Drone Cat Mouse.

4.3.2. Evaluador automático

En una versión inicial, la plataforma no disponía del *Juego Drone Cat Mouse*, sino, del *Reto Drone Cat Mouse*, es decir, este ejercicio inicialmente no incorporaba un evaluador automático que proporcionase un desafío para el usuario, por lo que se decidió incluir este evaluador para dar lugar al *Juego Drone Cat Mouse*.

Este evaluador automático consiste en una gráfica que muestra la distancia *Gato-Ratón* en el tiempo. La gráfica se genera con una librería de *JavaScript* llamada *Plotly* [5], en la cual se muestra la distancia en el eje vertical y el tiempo en el eje horizontal.

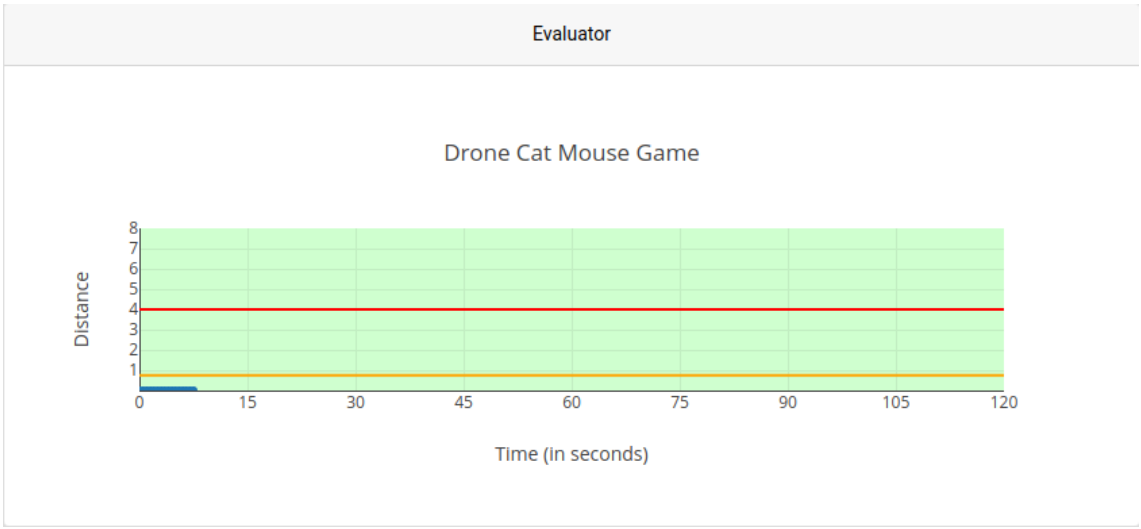

Figura 4.15: Evaluador Drone Cat Mouse.

4.3.3. Carga de código

Una vez se ha establecido la conexión con *RADI*, se puede proceder a la carga del código. Esta carga se realiza mediante un botón *Load in robot*, que una vez pulsado, se encarga de

[5] `https://plotly.com/javascript/`

habilitar el botón *Play* después de haber realizado una comprobación exitosa del código del usuario.

Cuando el usuario procede a realizar la carga del código en los respectivos robots, el código que se ha desarrollado se obtiene del editor de código situado en la parte izquierda de la pantalla. Para realizar la carga de código de las distintas dificultades, se ha desarrollado un script en *JavaScript* que se encarga de realizar una solicitud al servidor web, pidiendo el código correspondiente con la dificultad seleccionada.

Para que el servidor web pueda proporcionar el código de cada dificultad al navegador web solicitante, se han creado una serie de usuarios con el prefijo de *bot*, que contienen el código correspondiente a cada dificultad. También, ha sido necesaria la creación de un nuevo *endpoint* en el servidor web.

Desde el lado del servidor web, cuando se recibe una solicitud en dicho *endpoint*, este se encarga de determinar cual es la dificultad y el ejercicio del que se está solicitando por medio de los datos aportados en el propio *endpoint*.

Finalmente, unz vez se tienen ambos códigos, estos se envían cada uno por el *WebSocket* que se corresponde con su robot.

Capítulo 5

Juegos Compartidos Síncronos

En este capítulo se abardarán todos los detalles de los *juegos síncronos*. En una primera parte hablando del diseño e implementación de las plantillas del usuario, para porsteriormente, dirigirnos a los aspectos más técnicos relacionados con las comunicaciones *WebRTC* y el servidor web como servidor de señalización entre los usuarios.

5.1. Objetivo

En *Unibotics*, se parte de los *ejercicios asíncronos* en lo que únicamente pueden elegirse una serie de dificultades proporcionadas por la plataforma. Con el desarrollo los *ejercicios síncronos* se han querido orientar los juegos hacia un modo más competitivo, de manera que el usuario pueda conectarse con sus amigos y realizar competiciones poniendo en ejecucción sus mejores códigos.

5.2. Follow Line Game Síncrono

5.2.1. Diseño

El juego *Follow Line Síncrono* parte de la plantilla básica que comparten todos los ejercicios. A esta plantilla se le han añadido una serie de extras, como son el chat de texto, la barra de búsqueda de usuarios y los botones de carga de código, del anfitrión y del invitado. Adicionalmente, se ha añadido un botón para ocultar el editor de código con el fin del que el usuario

tenga únicamente la vista del chat de texto.

Con respecto la pantalla del usuario invitado, una vez este acepta la invitación proveniente del usuario anfitrión, mediante código *JavaScript*, se ha implementado un sistema que se encarga de ocultar todos los controles y elementos irrelevantes para este extremo de la conexión.

Para desarrollar este ejercicio, ha sido necesario implementar *software* tanto en el lado del cliente como en el lado del servidor. Pues los usuarios participantes deben usar el servidor web de *Unibotics* como servidor de señalización para el protocolo *WebRTC*. Una vez ambos extremos se conocen entre sí, se inicia el tráfico *WebRTC*.

Una vez el usuario entra en el ejercicio, automáticamente el navegador se conecta a una ruta del servidor web por medio de *WebSockets* generando una sala de juego. Esta ruta será por medio de la que se van a comunicar los extremos con el servidor de señalización.

Debido a la complejidad de las subredes, ha sido necesaria la incorporación de un servidor *TURN* (Transversal Using Relay NAT) que se encargue de hacer de retransmisor entre los dos extremos una vez se inicia la comunicación *WebRTC*. En la mayoría de casos no es posible realizar una conexión *WebRTC* directa entre los dos extremos (a menos que residan en una red local), esto es debido a que *WebRTC* no es capaz de atravesar un NAT (Network Address Translation).

5.2.2. Elección de oponente

Para proporcionar al usuario la elección de un contrincante, se ha implementado una barra de búsqueda de usuarios que permite al anfitrión, escribir el nombre del usuario que desea invitar.

Una vez el usuario realiza la selección del contrincante, la consulta es enviada al servidor web por medio del *WebSocket* de la sala de juego, y, este la reenviará al usuario invitado en caso de que se encuentre en línea y dentro del ejercicio.

En el caso de que el invitado acepte la solicitud, el anfitrión comenzará con la negociación *WebRTC*.

Barra de búsqueda

Para poder realizar la conexión entre dos usuarios, el anfitrión debe escribir el login del usuario que quiere invitar en la barra de búsqueda, de manera que se pueda hacer la consulta al

servidor web. Se ha desarrollado código *JavaScript* que se encarga de esperar a que se inserte un login y se pulse el botón *"Enter"*, y además, que se encargue de enviar la inserción al servidor web vía el *WebSocket* establecido para la sala del juego.

Una vez el servidor recibe este mensaje por el *WebSocket*, este se encarga de extraer el login del usuario, y mediante el uso de un paquete instalado en el servidor *Django*, llamado *"Django-online-users"*, obtiene los usuarios que hay conectados actualmente, y comprueba si el usuario recibido se encuentra en línea. En el caso en que la búsqueda no sea satisfactoria, se envía un mensaje al usuario anfitrión indicando la imposibilidad de realizar la invitación. Si la búsqueda se realiza con éxito, se envía la invitación a dicho usuario por *WebSocket*.

En el lado del usuario receptor, el navegador recibirá la petición y la mostrará al usuario por medio de un menú de diálogo que le da la posibilidad de aceptar o rechazar la invitación.

Una vez aceptada, se iniciará la conexión *WebRTC*. Y el usuario anfitrión será el encargado de iniciar la simulación y compartir la retransmisión.

Sección de chat

Como se ha mencionado antes, la plantilla del juego contiene un chat por el que ambos contrincantes pueden comunicarse. Este chat se ha implementando usando una conexión *WebRTC* y *DataChannels* entre los navegadores, de manera que se reduce la carga de tráfico que atraviesa el servidor web, pues estos mensajes son enviados por medio del canal de datos *peer-to-peer*.

Este sistema se ha desarrollado en la parte del navegador, mediante un *script* en *JavaScript* que contiene toda la configuración para realizar la creación de los *DataChannels*. Además, se ha implementado una función de *callback* (*onDataChannelMessage*) que es ejecutada cuando se recibe un mensaje por el *DataChannel*, esta se encarga de añadir el mensaje al chat.

Transmisión de datos entre plantillas

Este *DataChannel* es aprovechado para el paso de más información además de para el envío de los mensajes del chat. De manera que, una vez el anfitrión conecta el *RADI* en su máquina local, por este data channel son enviados todos los datos relativos a la interfaz del usuario, es decir, la vista de pájaro, y el evaluador automático hacia el otro extremo. Adicionalmente, cuando el usuario invitado presione el botón *Play* o el botón de carga de su código, estas órdenes serán enviadas al anfitrión por medio del canal de datos, que será quien las complete.

5.2.3. Carga flexible de código

Para proporcionar la libertad de la carga del código, se ha desarrollado un módulo *software* en *JavaScript* que se encarga de soportar la carga flexible del código.

De esta manera, cada usuario debe cargar su código antes de iniciar la simulación. Con respecto al usuario invitado, se ha desarrollado un sistema que se encarga de avisar al usuario host de que este quiere cargar su código, mediante el *DataChannel* establecido entre ambos. Una vez el host recibe este mensaje, se encarga de pedir el código al *WebServer* y cargarlo en el coche correspondiente. En el lado de la carga de código del host, simplemente, al pulsar el botón de carga de código, se pedirá el código al servidor web y se cargará en el robot correspondiente.

Una vez se carga un código, el botón de carga correspondiente se colorea de verde, de manera que para el usuario es más intuitivo a la hora de comprobar el estado de su código.

Ambos navegadores sincronizan sus interfaces por medio del canal de datos establecido entre los navegadores, enviando mensajes de control que una vez son interpretados, se aplican los cambios a la IU.

5.2.4. Ejecución compartida

Una vez se ha establecido la conexión *WebRTC* entre los usuarios, es necesario realizar la carga de código en los robotos, así como mostrar el estado de carga de los códigos y proporcionar una serie de controles de la ejecución de la sumilación a los usuarios.

Botones de control de la ejecución

En ambas interfaces, se encuentran serie de elementos de control de la simulación en común. Estos son:

- **Botón *Play*:** ambos usuarios tendrán la posibilidad de iniciar la ejercución del ejercicio una vez se han cargado los códigos en los robots.
- **Botón *Reset*:** botón que proporciona a ambos extremos realizar un reseteo del escenario.

Una vez cargados ambos códigos, se habilitará el botón *Play* en ambos extremos. Desde el lado del usuario *invitado*, se ha elaborado un sistema que se encarga de enviar las órdenes al usuario *anfitrión* cuando algún botón es pulsado. Una vez el *emisor* recibe una órden, este se

encarga de procesarla y ejecutar las órdenes. De esta manera, ambos usuarios pueden controlar la ejecución de la simulación de manera individual.

Botones de carga de código

De manera que los usuarios puedan conocer el estado de carga de sus códigos, se ha implementado un sistema que guarda el estado de carga y realiza una serie de ajustes visuales. Los indicadores de carga son los siguientes:

- **Botón de carga de código del anfitrión:** se encuentra únicamente en la plantilla del usuario *emisor*, este elemento es un botón que inicialmente tiene un color rojo, mostrando un estado que representa que no hay ningún código cargado, una vez se carga el código, este botón cambia a un color verde.
- **Botón de carga de código del invitado:** se encuentra ambas interfaces, este elemento es un botón que inicialmente tiene un color rojo, mostrando un estado que representa que no hay ningún código cargado, una vez se carga el código, este botón cambia a un color verde en ambos extremos. Este botón no puede ser pulsado en el lado del *emisor*.

El establecimiento de los estados se realiza mediante mensajes vía *WebSockets* entre ambos usuarios. En la plantilla del *anfitrión* es donde se almacenan todas las variables relativas al estado de los códigos, el extremo *invitado* únicamente recibe órdenes vía *WebSockets* para ajustar su interfaz.

5.2.5. Transmisión de vídeo y datos vía WebRTC

Una vez elegido el oponente, y aceptada la invitación, ha sido necesario el desarrollo de las fases descritas en la sección 3.5.2 de manera que se pueda negociar una conexión entre los dos extremos.

Dependiendo del extremo de la conexión que sea el usuario (*caller* o *callee*), necesita realizar unos pasos u otros, por lo que para cada extremo se necesita un código diferente, este código se ha recogido en un fichero llamado *rtc_stream.js"*.

En primer lugar, se ha creado una función llamada *startLocalStream*, que se encarga de realiazar la conexión desde el lado el anfritrión. Estos son los primeros 6 pasos detallados en la sección 3.5.2.

Desde el lado del usuario invitado, se ejecutará una función llamada *startRemoteStream*, que, una vez se reciba desde el servidor web la oferta enviada por el anfitrión mediante *WebSockets*, se encargará de realizar los pasos 7-11 descritos en la sección 3.5.2.

Finalmente, en el lado el anfitrión se recibirá desde el servidor web, la respuesta del otro extremo. Una vez recibida la respuesta, se almacena el *SDP* [1] (Session Decription Protocol) del extremo remoto. Una vez establecida la descripción del otro extremo, ambos *peers* tienen la identificación del otro, por lo que puede iniciarse una comunicación *WebRTC*.

Para realizar el envío de sus candidatos, cada *peer* debe enviarlos al servidor de señalización (haciendo uso de la función *onIceCandidate*, como respuesta al callback *icecandidate*) para que este los reenvíe al otro *peer*. En primer lugar, en el extremo remoto, se reciben los candidatos, este debe almacenar los candidatos (función *remotePeer.setRemoteDescription(offer)* que establece como descripción remota los candidatos recibidos por parámetro en la función *startRemoteStream*), y, responder con su oferta. Finalmente, en el extremo local, se reciben los candidatos del *peer* remoto, y se establece el SDP del usuario invitado, una vez realizado este paso, ambos extremos tienen el SDP del extremo remoto, por lo que pueden iniciar la elección del mejor candidato ICE, y, posteriormente el establecimiento de la conexión *WebRTC*.

De esta manera, gracias a este sistema, se permite que los usuarios tengan una conexión directa entre ellos, y únicamente un usuario ejecute el simulador, puesto que el elemento *HTML* que contiene el vídeo del simulador es retransmitido al otro extremo haciendo uso de la conexión entre *peers*.

Para soportar el intercambio de cantidatos ICE, entre los extremos, se ha implementado en el servidor web *Django* un sistema de señalización entre usuarios basado en salas usando *Django-channels* [2]. Ha sido necesario implementar un objeto (*StreamConsumer*) en el fichero *çonsumers.py"* que contiene la funcionalidad necesaria para el soporte de grupos, y el reenvío de mensajes al extremo remoto.

De igual manera, en el lado del canal de datos, la implementación del establecimiento de los candidatos ICE es similar al sistema implementado para la tranmisión de vídeo, únicamente ha sido necesario cambiar la parte del *streaming* por el establecimiento de un canal de datos, por el que viaja todo lo relativo a la IU y a mensajes del chat. En el lado del servidor, también ha sido

[1] `https://es.wikipedia.org/wiki/Session_Description_Protocol`

[2] `https://channels.readthedocs.io/en/stable/`

necesario implementar un consumidor espoecífico para la sala en el fichero *çonsumers.py"*.

Además, en ambos casos, en el lado del navegador, ha sido necesaria la implementación de un software para la gestión de los mensajes ente sockets, tanto para el *WebSocket* de vídeo, como para el de datos. Este sistema se encarga de ejecutar la distintas fases del establecimiento de candidatos, como iniciar la conexión, añadir la descripción SDP remota una vez recibida, y señalizar al extremo remoto. Esta lógica se agrupa en dos ficheros, *"ws_stream.js"* para el vídeo y *"ws_room.js"* para el canal de datos.

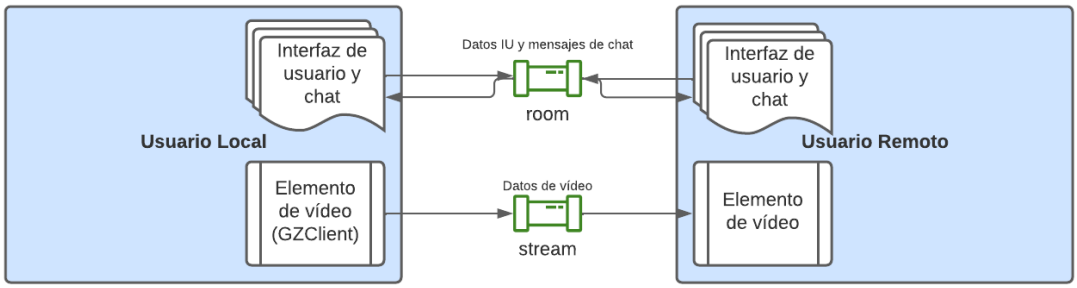

Figura 5.1: Diagrama de conexión WebRTC establecida.

Capítulo 6

Experimentos y validación

Este capítulo se introdujo como requisito en 2019. Describe los experimentos y casos de test que tuviste que implementar para validar tus resultados. Incluye también los resultados de validación que permiten afirmar que tus resultados son correctos.

Capítulo 7

Resultados

En este capítulo se incluyen los resultados de tu trabajo fin de grado.

Si es una herramienta de análisis lo que has realizado, aquí puedes poner ejemplos de haberla utilizado para que se vea su utilidad.

Capítulo 8

Conclusiones

8.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

Y si has llegado hasta aquí, siempre es bueno pasarle el corrector ortográfico, que las erratas quedan fatal en la memoria final. Para eso, en Linux tenemos aspell, que se ejecuta de la siguiente manera desde la línea de *shell*:

```
aspell --lang=es_ES -c memoria.tex
```

8.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a

2. b

8.3. Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. Aquí viene uno.
2. Aquí viene otro.

8.4. Trabajos futuros

Ningún proyecto ni software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

Apéndice A

Manual de usuario

Esto es un apéndice. Si has creado una aplicación, siempre viene bien tener un manual de usuario. Pues ponlo aquí.

Bibliografía

[1] HISTORIA DE LA ROBÓTICA. https://scielo.isciii.es/pdf/aue/v31n3/v31n3a02.pdf

[2] DOCUMENTACIÓN OFICIAL DE JAVASCRIPT https://developer.mozilla.org/es/docs/Web/JavaScript

[3] DOCUMENTACION OFICIAL DE DJANGO https://docs.djangoproject.com

[4] DOCUMENTACIÓN OFICIAL DE WEBRTC. https://webrtc.org/

[5] PABLO MORENO. TORNEOS DE PROGRAMACIÓN DE ROBOTS EN UNA PLATAFORMA ONLINE. (2020) https://gsyc.urjc.es/jmplaza/students/tfm-kibotics-torneos-pablo_moreno-2020.pdf

[6]