



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN

MÁSTER EN INGENIERÍA DE LA TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

Torneos de programación de robots
en una plataforma online

Autor: Pablo Moreno Vera

Tutor: José María Cañas Plaza

Curso académico 2019/2020

Resumen

Gracias al auge de la robótica en la actualidad, cada vez encontramos más productos robóticos a nuestro alrededor, por ejemplo robots industriales, robots en logística automatizada, aspiradoras, coches con funciones automáticas, etc. Es por ello que se necesitan más especialistas en este campo. Debido a esto, surgen plataformas dedicadas a la formación y aprendizaje de personas de todas las edades. Una de ellas es *Kibotics*. Esta plataforma pone al alcance de los más pequeños un entorno seguro de aprendizaje, tanto con robots simulados como reales. Gracias a Kibotics, los niños pueden aprender a manejar robots desde el navegador y sin ningún tipo de requerimiento previo. El entorno facilitado a los niños ofrece un conjunto de ejercicios educativos de programación de robots en dos lenguajes, *Python* y *Scratch*.

En este *Trabajo de Fin de Máster*, se ha querido ir un poco más allá en la plataforma y, aprovechando la tecnología multirobot que ofrece, se han desarrollado prácticas multirobot y multiusuario. Gracias a la tecnología *WebRTC* se ha desarrollado un tipo de práctica donde dos usuarios pueden competir de manera simultánea, cada uno con su propio código. Además se ha incluido un chat en el que los contrincantes pueden hablar sobre *WebRTC*.

Además, sobre estos ejercicios multirobot y multiusuario, se han desarrollado torneos en los que se pueden diferenciar dos clases:

1. **Torneos de ranking.** En estos torneos el usuario desarrollado su código en un tipo específico de ejercicio y guarda su código para competir de manera individual y por puntos contra el resto.
2. **Torneos de enfrentamientos.** En estos torneos los usuarios compiten directamente contra otros usuarios uno contra uno multiusuario y el que pierda será eliminado, el ganador pasará a la siguiente ronda.

Los torneos se han hecho automatizables, de modo que se pueden lanzar y se realizan de modo desatendido. A lo largo de su ejecución utilizan una base de datos Elasticsearch y se refresca automáticamente en la página web correspondiente del torneo. Además, se ha automatizado su difusión en tiempo real en la plataforma de vídeos masivos *Twitch*.

Índice general

1. Introducción	1
1.1. Robótica	1
1.2. Software Robótico	6
1.2.1. Middlewares robóticos	6
1.2.2. Simuladores robóticos	7
1.3. Robótica educativa	8
1.3.1. Plataformas y entornos STEM	9
1.3.2. Campeonatos de robótica educativa	12
2. Objetivos	14
2.1. Objetivos	14
2.2. Requisitos	15
2.3. Metodología	15
2.4. Plan de trabajo	17
3. Infraestructura	18
3.1. Lenguaje JavaScript	18
3.2. Realidad virtual en el navegador web: A-Frame	19
3.3. Entorno Django para servidores web	20
3.4. Elasticsearch	21
3.5. Gestores de paquetes: NPM y Webpack	23
3.5.1. Node Package Manager (NPM)	24
3.6. WebRTC	24
3.6.1. Protocolos	25

3.6.2. Gestión de la conexión	26
3.6.2.1. Descriptores de sesión	27
3.6.2.2. Candidatos ICE	28
3.7. Retransmisión en directo	29
3.7.1. API de Youtube Live	30
3.7.2. API de Twitch	31
4. Juegos compartidos síncronos	32
4.1. Antecedentes	32
4.2. Uso de los juegos compartidos síncronos	33
4.3. Diseño	35
4.4. Elección de contrincante y chat	36
4.4.1. Barra de búsqueda de usuario	36
4.4.2. Sección de chat	37
4.5. Ejecución compartida y evaluadores	38
4.5.1. Botones de control de la ejecución	38
4.5.2. Indicadores del estado de la conexión	39
4.5.3. WebRTC	40
4.5.4. Carga flexible de código	41
4.5.5. Ejercicios	42
4.5.6. Evaluadores automáticos	43
5. Torneos	44
5.1. Antecedentes	44
5.2. Diseño	45
5.2.1. Torneos de ranking	46
5.2.2. Torneos de enfrentamiento	47
5.3. Torneos de ranking	48

5.3.1.	Plantilla de la página web del torneo	48
5.3.2.	Infraestructura del servidor	49
5.3.3.	Modelo de datos en Elasticsearch para los torneos	50
5.3.4.	Maestro de ceremonias	51
5.3.5.	Automatización	52
5.4.	Torneos de enfrentamientos	54
5.4.1.	Plantilla de la página web del torneo	55
5.4.2.	Infraestructura del servidor	56
5.4.3.	Modelo de Elasticsearch	56
5.4.4.	Maestro de ceremonias	57
5.4.5.	Automatización	58
5.4.5.1.	Automatización	59
5.5.	Retransmisión en vivo	61
5.5.1.	Retransmisión en Twitch	61
5.5.2.	Retransmisión en Youtube Live	62
6.	Conclusiones	65
6.1.	Conclusiones	65
6.2.	Trabajos futuros	66
A.	Refactorización en Kibotics	68
Bibliografía		72

Capítulo 1

Introducción

El TFM descrito a continuación se encuadra en la plataforma *Kibotics* de la asociación *JdeRobot*¹, para la enseñanza de la programación de robots. La intención principal de su desarrollo ha sido mejorar la plataforma incluyendo ejercicios que fomenten la interacción social y lúdica de sus usuarios.

En este capítulo se introducirá el contexto en el que se sitúa este proyecto y la motivación que ha llevado a su desarrollo. Para ello, es preciso comenzar con una explicación a grandes rasgos sobre qué es la robótica y sus aplicaciones para la sociedad.

Dentro del heterogéneo campo que es la robótica, este TFM se enmarca en la robótica educativa, destinada al aprendizaje en este campo.

1.1. Robótica

Durante toda la evolución de la humanidad el principal factor que la distingue del resto de seres vivos es su capacidad de pensamiento. Una de las principales muestras de inteligencia es el desarrollo de herramientas que facilitan el trabajo. La robótica nace en el momento en que el ser humano utiliza las computadoras y las máquinas para reducir el esfuerzo empleado. Así, la robótica es la rama de la tecnología basada en la utilización de la informática para el diseño y desarrollo de sistemas automáticos que faciliten la vida al ser humano e, incluso, llegando a sustituirle en algunos de ellos. La robótica está muy relacionada con campos tan distintos como la física, las matemáticas,

¹<https://jderobot.github.io/>

CAPÍTULO 1. INTRODUCCIÓN

la electrónica, la mecánica, la inteligencia artificial, la ingeniería de control, etc. Gracias a todas estas disciplinas, englobadas correctamente, se pueden diseñar máquinas que ejecuten comportamientos autónomos para el propósito que han sido desarrollados. Estas máquinas se denominan robots.

Desde 1950, estos sistemas autónomos han desarrollado un crecimiento exponencial en cuanto a su complejidad, versatilidad, autonomía y, sobre todo, su inclusión en una gran variedad de ámbitos. En parte, esto es debido al gran desarrollo en cuanto a potencia y complejidad computacional que han experimentado los ordenadores en los últimos años, que han permitido el desarrollo de comportamiento más complejos en los robots. Tal es el grado de desarrollo alcanzado que la mayoría de sistemas operados por el ser humano comienzan a incorporar un sistema de control específico programable que permiten el desarrollo de tareas repetitivas con un gran riesgo para las personas, englobando tareas básicas pero de difícil realización.

El principal ejemplo del desarrollo de la robótica es su inclusión en la industria. Este ha sido el principal ámbito que ha impulsado este campo debido a la repetitividad de sus acciones y ha su peligrosidad. Un ejemplo claro son robots de control de calidad mediante visión artificial (Figura 1.1) o robots con brazos articulados que pueden manejar grandes pesos (Figura 1.2).

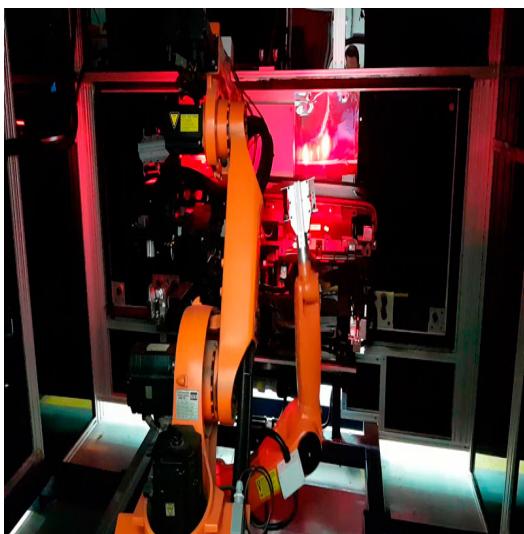


Figura 1.1: Robot de control de calidad



Figura 1.2: Brazo robótico en la industria

En la actualidad, y gracias al abaratamiento de construcción de un robot debido a la gran demanda, la robótica está presente en multitud de campos.

Otro claro ejemplo son los coches autónomos. Se trata de un campo en gran desarrollo y que cuenta con distintos horizontes. Algunos de ellos son el asistente de aparcamiento (Figura 1.3) o aparcamiento automático o la conducción autónoma de *Tesla* y *Audi* (Figura 1.4).



Figura 1.3: Conducción autónoma



Figura 1.4: Asistente de aparcado

Derivados de estos dos campos de aplicación, podemos encontrar la robótica también en sectores como la logística, como la flota de robots de *Amazon* (Figura 1.5) o en el sector doméstico con el robot aspiradora de *Roomba* (Figura 1.6). En estos sectores ha tenido un gran impacto el abaratamiento de la tecnología robótico, puesto que, en caso contrario, no habrían tenido nicho de mercado.



Figura 1.5: Flota de robot de Amazon



Figura 1.6: Aspiradora robótica

Además, existen otros campos de aplicación más novedosos, aunque no por ello menos importantes. Por ejemplo, la aplicación de la robótica en el sector militar (Figura 1.7). Otro campo es la agricultura, con robots drones que permiten fumigar plantaciones o robots con sensores térmicos para predecir incendios o la necesidad de regado o, incluso, robots cosechadora (Figura 1.8).



Figura 1.7: Robot militar



Figura 1.8: Robot cosechadora

Otro campo de aplicación muy importante es la medicina y un claro ejemplo es el robot DaVinci con el que los doctores pueden realizar operaciones de manera eficaz, segura y con una gran precisión (Figura 1.10). Existen también prototipos de robots de anda bípeda

CAPÍTULO 1. INTRODUCCIÓN

que son capaces de interactuar con humanos como el robot Atlas (Figura 1.9).

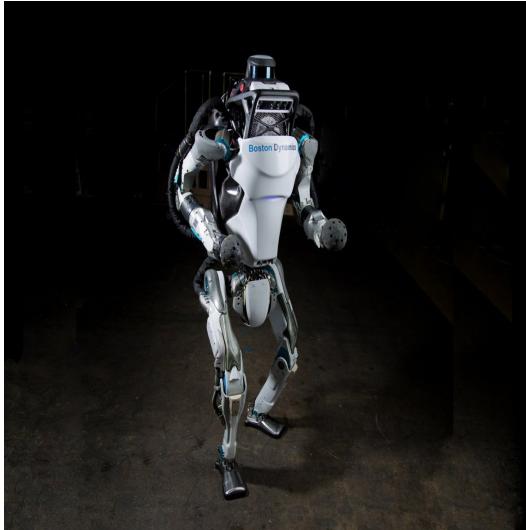


Figura 1.9: Robot Atlas

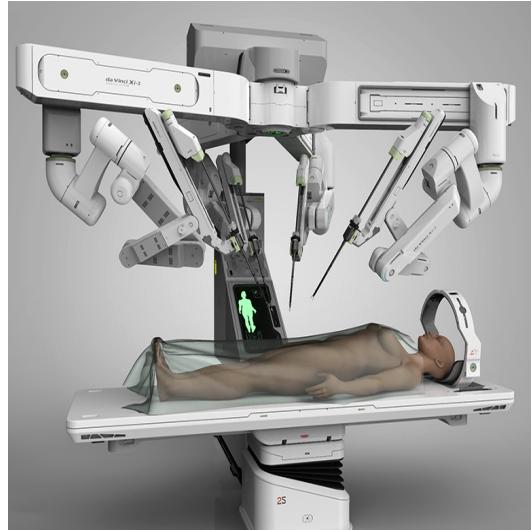


Figura 1.10: Robot DaVinci

Como hemos visto, la robótica está en muchos campos de aplicación en la actualidad y su importancia es creciente. Otro ejemplo claro ocurrió el 15 de Abril de 2019 en Notre Dame, París. Allí un grupo de robots fueron vitales para salvar la catedral a tiempo. Para ello, un equipo de bomberos especializados en la utilización del robot Colossus, consiguió acceder al interior de la catedral con ellos y extinguir así el incendio (Figura 1.11).



Figura 1.11: Robot Colossus

1.2. Software Robótico

Un robot está formado por dos componentes principales: el *hardware*, la infraestructura física del propio robot, y el *software* que se encarga de dotar de inteligencia al robot. Esta última es la parte más importante de los robots y podemos encontrar diferentes elementos como bibliotecas de código, *middlewares* robóticos o, incluso, simuladores.

Para que los robots puedan ser controlados de una manera efectiva, el comportamiento del software que los controla debe ser robusto. Típicamente, el software robótico se divide en tres capas: drivers, middleware y aplicaciones, cuya arquitectura será distinta en función de su finalidad.

Gracias al gran desarrollo de la robótica, los robots actuales ya no precisan del control del ser humano para su funcionamiento. En la actualidad tienen comportamientos autónomos que les permiten realizar las tareas sin la mediación de terceros. Esto ha sido posible gracias al minucioso desarrollo del software que compone sus sistemas, algo parecido a una inteligencia artificial. Aspectos importantes de este software robótico son los circuitos de realimentación, control, la planificación de caminos, la autolocalización o procesamiento de imágenes.

De la mano del auge de la robótica han aparecido multitud de plataformas que facilitan el desarrollo de software robótico, también llamados middlewares robóticos. Otra herramienta muy importante en la robótica son los simuladores dado que permiten realizar pruebas y depurar fallos para programar una versión funcional del robot antes de fabricarlo.

1.2.1. Middlewares robóticos

Los middlewares robóticos pueden definirse como entornos o *frameworks* para el desarrollo de software para robots. Se trata de software que conecta aplicaciones o componentes software para soportar aplicaciones complejas y distribuidas. Para poder controlar los sensores y actuadores de los robots, estos *middlewares* aportan *drivers*, arquitecturas *software*, bloques de funcionalidad, APIs, simuladores, visualizadores, etc. Es por todo esto que a los middlewares se les conoce como “pegamento para software”. Una de las tareas más importantes del middleware es conectar el hardware (real o simulado) con las aplicaciones (software).

El middleware más extendido es ROS. *Robotics Operating System*² es un sistema operativo para el desarrollo robótico libre que proporciona toda la funcionalidad necesaria de un sistema operativo como el control de dispositivos bajo nivel, mecanismos de intercambio de mensajes entre procesos y la abstracción hardware. Fue creado en 2007 por el Laboratorio de Inteligencia Artificial de *Stanford* para dar soporte a sus robots y, en la actualidad, está desarrollado por *Open Robotics*³. Aunque ROS fue diseñado para sistemas UNIX, ha sido adaptado para ser soportado en otros sistemas operativos como Fedora, Debian, Windows, MacOS-X, Arch, Slackware, Gento u OpenSUSE, llegando a permitir aplicaciones multiplataforma. Gracias a todo esto, ROS es el middleware más extendido en el mundo.

1.2.2. Simuladores robóticos

Debido al gran coste que supone la fabricación del hardware de los robot, es conveniente depurar el software al máximo antes de ejecutarlo en el robot físico, de esta manera se reducen los posibles daños de ejecuciones erróneas. Para ello existen herramientas especializadas en reproducir el comportamiento del robot en entornos controlados de manera virtual. Estas herramientas son los simuladores. Algunos de los más utilizados son:

*Gazebo*⁴ (Figura 1.12). Es el simulador más extendido de código abierto. Tiene gran importancia su motor de renderizado, sus motores de físicas y su soporte para plugins de sensores y actuadores, además de su amplio catálogo de robots. Otro hecho importante es su integración con ROS, por lo que permite probar el software desarrollado en un robot simulado.

²<http://www.ros.org/>

³<https://www.openrobotics.org/>

⁴<http://gazebosim.org/>

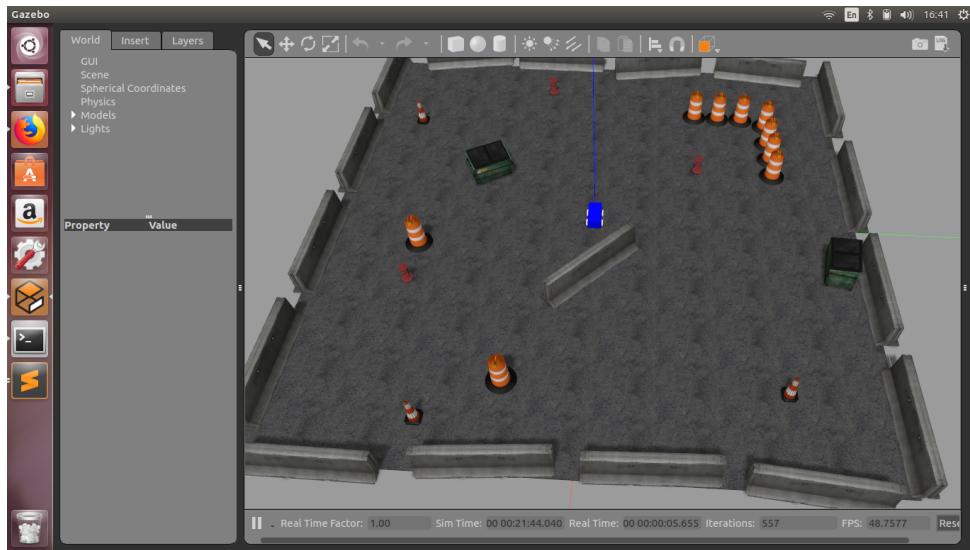


Figura 1.12: Simulador Gazebo

*Webots*⁵. Simulador de robótica avanzada en el que se pueden desarrollar modelos propios y su física, escribir controladores y hacer simulaciones a gran velocidad. Un ejemplo es su soporte para el humanoide Nao. Recientemente, se ha convertido a software libre.

*CoppeliaSim*⁶. Se trata de un simulador de robótica con un entorno de desarrollo integrado. Está basado en una arquitectura de control distribuido donde cada objeto puede ser controlado individualmente con un *script*, *plugin*, nodo *ROS*, API remota o solución personalizada, lo que lo hace muy versátil e ideal en aplicaciones multirobot. Los controladores pueden escribirse en C/C++, Python, Java, Lua, Matlab u Octave.

1.3. Robótica educativa

Como se ha comentado, la robótica educativa es un campo en auge debido al gran desarrollo de la robótica y la necesidad creciente de especialistas y profesionales en todo el mundo. En 2015, la *Comunidad de Madrid* introdujo la asignatura “Tecnología, Programación y Robótica” en el plan docente de Enseñanza Secundaria[1]. En el año 2020-2021 se prevé implantar la asignatura “Programación y Robótica” en el plan docente de

⁵<https://www.cyberbotics.com/>

⁶<https://www.coppeliarobotics.com/>

Enseñanza Primaria[2].

Alineado con la robótica se ha creado la educación STEM (Science, Technology, Engineering and Mathematics). Esta nueva forma de enseñanza promueve el pensamiento científico y la adquisición de conocimientos tecnológicos aplicables a situaciones reales permitiendo el desarrollo de competencias en la resolución de problemas.

Para poder impartir este tipo de asignaturas es necesaria la infraestructura adecuada. Las plataformas como la de LEGO o la de Arduino permiten una enseñanza y aprendizaje sencillos de la robótica resultando muy gratificante para el alumno por lo vistoso de los resultados y la obtención de una aplicación real inmediata.

1.3.1. Plataformas y entornos STEM

Es importante acercar este conocimiento tan complejo de manera adecuada a edades cada vez más tempranas para permitir un mayor desarrollo en el conocimiento de esta tecnología. Es por ello que cada vez hay más plataformas que desarrollan software orientado a niños.

Algunos ejemplos de plataformas educativas son:

- **Scratch**[3]. Es un proyecto utilizado en docencia y liderado por el MIT⁷ para programar animaciones, interacciones y juegos de manera sencilla por su interfaz visual. Toda la funcionalidad de un lenguaje de programación está embebida en bloques gráficos que agrupan funcionalidades específicas.
- **LEGO**[4]. Se trata de una plataforma que dispone de multitud de robots programables con su sistema gráfico.
- **Kodu**[5]. Es un sistema de programación visual para el desarrollo de videojuegos desarrollado por Microsoft⁸
- **Snap!**[6]. Se trata de un plataforma basada en *Scratch* pero con funcionalidad destinada a edades más avanzadas que permiten el desarrollo de funcionalidad más compleja.

⁷<http://www.mit.edu/>

⁸<https://www.microsoft.com/es-es>

- **OpenRoberta**⁹ [7]. Es una plataforma educativa en la nube desarrollada por el instituto alemán *Fraunhofer IAIS* para que los niños puedan programar robots utilizando *Lego Mindstorms* o robots programables de *LEGO* (Figura 1.13).



Figura 1.13: Plataforma Open Roberta

- **Vex**¹⁰. Plataforma orientada a la disciplina STEM de enseñanza que ofrece una gran cantidad de robots programables, así como tutoriales para aprender a montarlos y manejarlos.
- **AppInventor**¹¹. Es un entorno de desarrollo software creado por *Google Labs* y actualmente mantenido por el *MIT* que ofrece toda la infraestructura necesaria para crear aplicaciones en *Android*. De esta manera, y visualmente, el usuario puede desarrollar una aplicación con bloques de código visuales, parecidos a *Scratch*.
- **Arduino Web Editor**¹². Arduino Web Editor es una plataforma que proporciona al usuario todo lo necesario para poder programar su código en línea. De esta manera, no es necesario que el usuario instale ningún tipo de software en su sistema. Esto facilita mucho el aprendizaje dado que es inmediato (Figura 1.14).

⁹<https://lab.open-roberta.org/>

¹⁰<https://www.vexrobotics.com/>

¹¹<https://appinventor.mit.edu/>

¹²<https://create.arduino.cc/>

CAPÍTULO 1. INTRODUCCIÓN

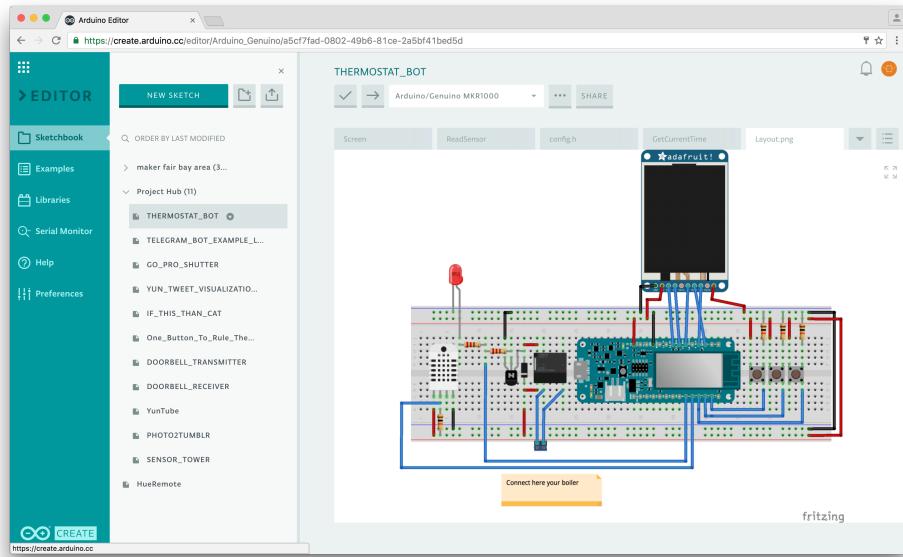


Figura 1.14: Plataforma Arduino Web Editor

Otra plataforma de robótica educativa, en la que se ha desarrollado este trabajo, es *Kibotics* (Figuras 1.15 y 1.16), que ofrece distintos entornos simulados y controlados con robots a los alumnos. Con esto, los alumnos pueden programar la inteligencia de robots autónomos para que realicen distintas pruebas sin ningún riesgo. De esta manera es posible aprender robótica en casi cualquier edad de manera sencilla y con la única necesidad de acceso a internet.



Figura 1.15: Plataforma Kibotics

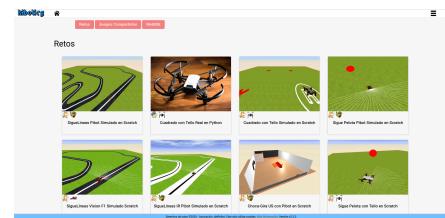


Figura 1.16: Página principal de Kibotics

Esta plataforma permite el aprendizaje de la robótica con un simulador en un entorno web. De esta manera, los usuarios sólo necesitan conexión a internet. Soporta distintos modelos de robots como el dron *Tello* o el *Mbot*, entre otros, tanto de manera simulada como en los robots reales. Además dispone de dos lenguajes de programación, *Scratch* (Figura 1.17) y *Python* (Figura 1.18) para llegar a usuarios de distintas edades.

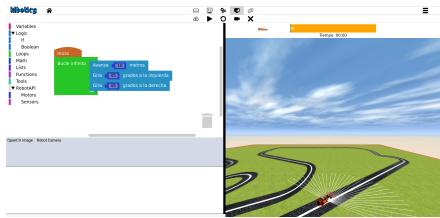


Figura 1.17: Editor y simulador de un ejercicio Scratch

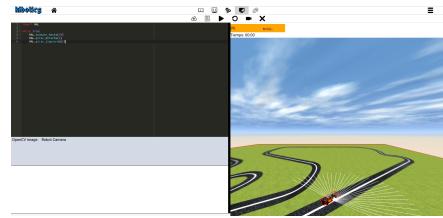


Figura 1.18: Editor y simulador de un ejercicio Python

1.3.2. Campeonatos de robótica educativa

Como acabamos de ver, la robótica educativa cada vez cuenta con un mayor soporte y tiene una aceptación mayor por parte de los usuarios. Es por ello que cada vez, surgen más campeonatos que desarrollan el aspecto social y lúdico para fomentar su uso y conseguir que el usuario adquiera un mayor conocimiento en la programación. Esto se consigue incitando al usuario a probar el código desarrollado contra el código de otros usuarios. es una manera muy divertida de depurar errores y perfeccionar el código. Algunos ejemplos de competiciones robóticas son:

- **First Lego League¹³**. Fue creado en 2007. En esta competición promocionada por LEGO, los participantes compiten en distintas franjas de edad en distintas disciplinas y retos. Se celebra en Agosto y la plataforma presenta una prueba a la que los participantes deberán hacer frente para encontrar una solución.
- **RoboCup Junior¹⁴**. Fue creada en 1997. Se trata de una entidad pionera a nivel de torneos que ofrece un torneo para desarrolladores junior y otro para senior. En torneo junior compiten usuarios de distintas franjas de edad en distintas tareas que deben completarse. Tiene tres pruebas diferentes: fútbol (2 contra 2), rescate y baile.
- **Torneo Nacional VEX Robotics IQ¹⁵**. Organizado por la *Fundación educaBOT* en el que los participantes deben fabricar y programar un robot para completar las distintas tareas que se proponen.

¹³<https://www.firstlegoleague.es/>

¹⁴<https://junior.robocup.org/>

¹⁵<https://vexspain.com/vex-iq/torneos/>

- **RoboCampeones¹⁶**. Es un campeonato orientado a institutos aunque también admite participantes libres en otras modalidades en los que los participantes compiten en pruebas como sumo con robots LEGO y Arduino. Nació en el *RoboticsLabURJC* de la Universidad Rey Juan Carlos y se celebra en Fuenlabrada en los últimos años. Cuenta con más de 2000 participantes.
- **Eurobot Junior¹⁷**. Fue creado en 1998 en París. Es un torneo para menores de 18 años en el que se debe programar un robot guiado por cable y otro autónomo para competir en las distintas pruebas.



Figura 1.19: Torneo de Robocampeones



Figura 1.20: Torneo Robocup Junior

Como puede verse, en la actualidad existe un auge en la creación de plataformas educativas y de torneos para niños en el ámbito de la robótica. Aprovechando esta circunstancia, se ha desarrollado en este trabajo la posibilidad de realizar torneos en línea dentro de la plataforma *Kibotics*. Con esto se elimina la barrera de la presencia en los torneos por parte de los niños que, muchas veces, es un impedimento para que puedan acceder a ellos.

¹⁶<http://robocampeones.org/>

¹⁷<https://www.eurobot.org/junior/eurobot-junior-2020>

Capítulo 2

Objetivos

Una vez introducido todo el contexto en el que se ha desarrollado este trabajo, es momento de profundizar en los objetivos que se quieren alcanzar, los requisitos para las soluciones que se puedan plantear y la metodología seguida durante todo el desarrollo de este TFM para conseguirlos.

2.1. Objetivos

La principal meta planteada en este TFM ha sido la mejora de la plataforma *Kibotics* con funcionalidades lúdicas y sociales que fomenten un mejor aprendizaje. Este objetivo principal se ha dividido en dos subobjetivos.

- La introducción del elemento social en la plataforma con ***Juegos compartidos síncronos*** que permitan a dos estudiantes competir con los robots que han programado en el mismo escenario y comentarlo en vivo en un chat compartido.
- El desarrollo de la infraestructura necesaria para realizar ***Torneos automáticos*** de programación de robots dentro de la plataforma, con múltiples participantes y su retransmisión a través de mecanismos de difusión masiva de vídeos.

2.2. Requisitos

Las soluciones a desarrollar para alcanzar los objetivos descritos en la sección anterior deben cumplir además, los requisitos siguientes:

- Para el desarrollo de código en la plataforma en el cliente, se va a utilizar *HTML-5, CSS-3, JavaScript*.
- Debe funcionar dentro de Kibotes v2.0 o superior.
- Los torneos deben ser guionizables (scriptables), de modo que se puedan lanzar de manera completamente desatendida.
- No debe requerir la instalación de software adicional o extensiones en el navegador web.

2.3. Metodología

El desarrollo de este TFM puede descomponerse en un conjunto de iteraciones con distintas fases. A este modelo de desarrollo se le conoce como *Modelo de desarrollo en espiral*. Este modelo es típico en la *Metodología de desarrollo ágil*, muy frecuentes en desarrollo software.

El modelo de desarrollo en espiral escogido ha sido el creado por *Barry Boehm* (Figura 2.1). Al tratarse de un modelo en espiral, se adapta a la perfección a nuestras necesidades, permitiendo disponer de flexibilidad ante cambios en los requisitos semanales, algo común mientras se avanzaba en este desarrollo. Con esto, se ha conseguido una mitigación del riesgo en el proyecto gracias a la temprana identificación de potenciales riesgos en el mismo, consiguiendo así, una mejor calidad final.

CAPÍTULO 2. OBJETIVOS

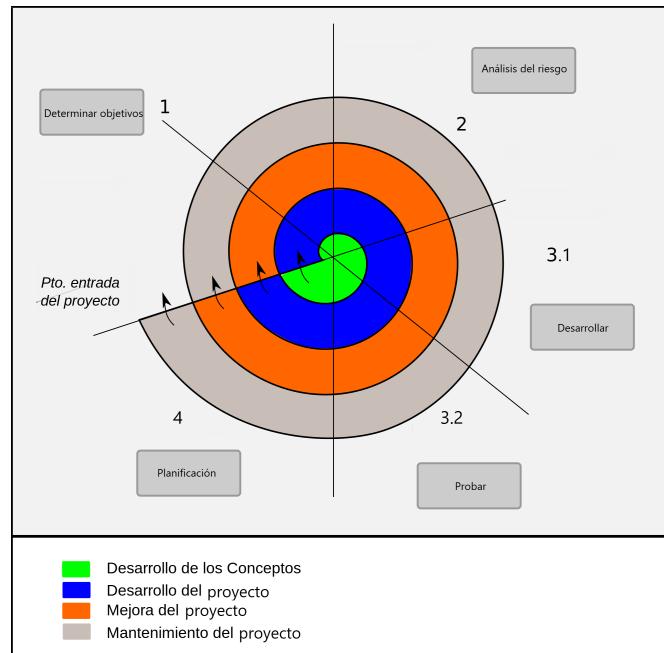


Figura 2.1: Modelo de desarrollo en espiral

La ventaja con este ciclo de vida es que permite la obtención temprana de prototipos funcionales, la optimización progresiva de estos prototipos y, en última instancia, pulir los detalles para abarcar la totalidad de los objetivos especificados. De esta manera, el trabajo se desarrolla de manera incremental con cuatro fases bien definidas:

- **Determinación de objetivos:** Esta primera fase del ciclo está formada por la definición de las metas para esa etapa.
- **Análisis del riesgo:** En esta etapa se evalúan los posibles problemas iniciales al desarrollo y las soluciones a los mismos.
- **Desarrollar y probar:** En esta tercera fase se procede al desarrollo del trabajo propiamente dicho, junto con una serie de pruebas para verificar su funcionamiento.
- **Planificación:** En esta última fase del ciclo de vida se valoran los resultados obtenidos y se planifican las siguientes etapas del proyecto.

Según este modelo, se han realizado reuniones con el tutor cada 3 días en las que se determinaban las tareas esenciales e indivisibles en los que se podían subdividir los subobjetivos y que se abordaban para la siguiente reunión.

Además de las reuniones semanales, se ha realizado un seguimiento de las primeras fases del desarrollo con herramientas externas como la bitácora semanal¹ en el que se redactaban los avances principales. Se ha contado con un repositorio en *GitHub* de apoyo para almacenar esa bitácora en forma de *GitHub Pages*². Igualmente, el desarrollo de software de este TFM se ha realizado en *GitHub*, utilizando el mecanismo de incidencias (*issues*), parches (*pull-request*) y revisión por expertos (*commiters*).

2.4. Plan de trabajo

Para la consecución de los objetivos descritos, se han seguido estas etapas de trabajo:

1. **Estudio de la plataforma Kibotics:** En esta primera fase de toma de contacto se han modificado las plantillas de la plataforma con una primera versión. Además, la puesta en funcionamiento en local del servidor, el simulador y los ejercicios. En cuanto al simulador, se han realizado pequeñas modificaciones en su funcionalidad.
2. **Estudio de la tecnología WebRTC:** En esta fase se ha estudiado la tecnología web de retransmisión de vídeo/audio.
3. **Diseño y desarrollo de los *juegos compartidos síncronos*:** Una vez adquiridos los conocimientos necesarios, se ha realizado una refactorización completa de las plantillas el servidor y se ha desarrollado toda la infraestructura para los *juegos compartidos*. Éstos se apoyan en los ejercicios competitivos en *Kibotics*.
4. **Diseño y desarrollo de los *torneos automáticos*:** Tras completar el desarrollo de los *Juegos-Compartidos*, se va a crear la infraestructura para los torneos. Éstos se apoyan en los *juegos compartidos síncronos*, con las adaptaciones necesarias para automatizarlos.
5. **Integración:** Una vez completadas las etapas anteriores, es necesario integrarlas en la plataforma en producción, resolviendo las cuestiones de despliegue necesarias.

¹<https://roboticslaburjc.github.io/2019-tfm-pablo-moreno/logbook/>

²<https://github.com/RoboticsLabURJC/2019-tfm-pablo-moreno>

Capítulo 3

Infraestructura

En este capítulo se presentan todos los componentes y software que han servido de apoyo en el desarrollo del TFM. Se pueden organizar en tres grupos: las tecnologías usadas dentro de la plataforma *Kibotics* (JS, A-Frame, Django, Elasticsearch), las tecnologías *WebRTC* usadas para desarrollar los juegos compartidos síncronos en este trabajo y las tecnologías de difusión de vídeo usadas para los torneos automáticos.

3.1. Lenguaje JavaScript

Se trata de un lenguaje interpretado de alto nivel bajo el estándar *ECMAScript*¹ y basado en *Java* y *C*. Fue creado como lenguaje de aplicaciones web en el cliente. Se interpreta en el navegador web para mejorar su interfaz y obtener páginas web dinámicas. En la actualidad se ha extendido al lado del servidor con *Node.js* y se ha convertido en el lenguaje de desarrollo web más extendido.

En este proyecto se ha utilizado *ECMAScript-6* para el desarrollo del lado del cliente. De esta manera, se ha programado la inteligencia de las plantillas que corre en el navegador. Las principales características de ES-6 son:

- Es un lenguaje estructurado similar a C (comparte gran parte de su estructura). Le diferencia con *JavaScript* en el alcance de las variables definidas, ya que incorpora la palabra reservada “*let*” para tener compatibilidad *block-scoping*.

¹Especificación de lenguaje de programación que define tipos dinámicos y soporta programación orientada a objetos

- Tiene tipado débil, por lo que los datos se asocian al valor en lugar de la variable. Esto permite que la misma variable sea de distintos tipos en distintas partes del código.
- Se trata de un lenguaje interpretado, por lo que no requiere compilación ni fichero binario de código y cada navegador tiene su intérprete para ejecutarlo.
- Está formado por objetos en su totalidad en los que los nombres de sus propiedades son claves de tipo cadena.
- Tiene evaluación en tiempo de ejecución con la función “eval” que evalúa un código en *JavaScript* representado como una cadena de caracteres.

3.2. Realidad virtual en el navegador web: A-Frame

Es un entorno de código abierto para crear entornos de realidad virtual a partir de HTML para que sea sencillo de leer y comprender². Se usa dentro de *Kibotics* como base para el simulador robótico *WebSim*, que corre dentro del navegador web. La versión en *Kibotics* es la última estable (v0.9.2), aunque está en constante desarrollo. Además, tiene compatibilidad con otros entornos como *Vive*, *Rift*, *Windows Mixed Reality*, *Daydream* o *GearVR* y soporte tanto para ordenadores como para *smartphones*.

A-Frame se basa en HTML y DOM con *polyfill*³. Para la creación de una escena solo es necesario el fichero HTML con su descripción. Además, la mayoría de las herramientas de HTML (como *React*, *Vue.js* o *JQuery*) tienen soporte para *A-Frame*.

Tanto HTML como DOM forman la capa más externa del entorno, por debajo está el componente *Three.js* gracias al cual un componente puede ser utilizado en otras entidades. Con esta característica no es necesario repetir código, ya que el elemento puede ser referenciado las veces que sea necesario.

Además, *A-Frame* proporciona primitivas para la declaración de los objetos en el escenario. Éstas no son más que los distintos elementos de los que va a constar el escenario. También permite primitivas complejas para la inclusión de elementos como robots.

²<https://aframe.io/>

³Fragmento de código en *JavaScript* para dar soporte a funcionalidad moderna en navegadores antiguos.

A-Frame sigue una arquitectura Entidad, Component y Sistema (ECS), que es muy utilizada en entornos de desarrollo de videojuegos y en 3D. Se trata de una jerarquía caracterizada por el principio de herencia, con lo cual los elementos de más bajo nivel heredan las características de los elementos de mayor nivel de los que dependen. Esto aporta flexibilidad al definir objetos, gran escalabilidad o eliminación de largas cadenas de herencia. El API para definir cada tipo es el siguiente:

- Entidad: Se representa con *a-entity*.
- Componente: Se representa como un atributo HTML y está formado por esquema, manejadores y métodos. Para declarar un componente se utiliza el método de *A-Frame registerComponent*.
- Sistema: Se representa con atributos HTML mediante la etiqueta *a-scene* y se registran con el método de *A-Frame registerSystem*.

3.3. Entorno Django para servidores web

Django[8][9] es un entorno de alto nivel en Python que fomenta el desarrollo rápido y limpio y un diseño pragmático de servidores web. Se utilizó en producción y, en 2005, se liberó bajo una “licencia BSD”. Se ocupa de lo relacionado con el desarrollo web, para que puedas centrarte en escribir tu aplicación. Es el entorno en el que se ha desarrollado el servidor web principal de la plataforma *Kibotics*.

Se fundamenta en la creación sencilla de sitios web, poniendo énfasis en la conectividad y extensibilidad de sus componentes, la reutilización, el desarrollo dinámico y en principio “Don’t Repeat Yourself”. También soporta el uso de bases de datos como *MySQL*, *PostgreSQL* y *SQLite-3*. En cuanto a servidores web en producción soportados, aunque se recomienda *Apache*, soporta la especificación *WSGI*. El único requerimiento de Django es Python 2.5 como mínimo.

Algunas características de Django son:

- Mapeador objeto-relacional. Es capaz de crear una BD virtual orientada a objetos sobre la BD relacional utilizada.
- Soporta aplicaciones instalables en cualquier página gestionada con Django.

- API de BD robusta.
- Sistema de vistas genéricas para ciertas tareas comunes.
- Sistema extensible de plantillas con herencia y etiquetas.
- *Dispatcher* de URLs basado en expresiones regulares.
- Sistema *middleware* para el desarrollo de características adicionales como el cacheo, la compresión de salida, la normalización de URLs, la protección CSRF y soporte de sesiones.
- Soporte de internacionalización, incluyendo traducciones.
- Documentación incorporada a través de la aplicación administrativa.

3.4. Elasticsearch

Elasticsearch[10] es una base de datos no relacional con un motor de analítica y análisis distribuido y software libre para todos los tipos de datos, incluidos textuales, numéricos, geoespaciales, estructurados y desestructurados. Está desarrollado en *Apache Lucene* y fue presentado por primera vez en 2010 por Elasticsearch N.V. (ahora conocido como Elastic). Es la base de datos que utiliza la plataforma *Kibotics* para guardar información circunstancial, no estructural como eventos y alarmas.

Es conocido por sus *API REST* simples, naturaleza distribuida, velocidad y escalabilidad. Es el componente principal del *Elastic Stack*, un conjunto de herramientas de software libre para la inserción, el enriquecimiento, el almacenamiento, el análisis y la visualización de datos. Comúnmente referido como *ELK Stack* (por *Elasticsearch*, *Logstash* y *Kibana*), el *Elastic Stack* ahora incluye una gran colección de agentes de envío conocidos como *Beats* para enviar los datos a Elasticsearch.

La velocidad y escalabilidad de Elasticsearch y su capacidad de indexar muchos tipos de contenido significan que puede usarse para una variedad de casos de uso: búsqueda de aplicaciones, búsqueda de sitio web, *logging* y analíticas de log como las utilizadas en *Kibotics* para llevar una monitorización de los usuarios, monitoreo de rendimiento de aplicaciones en navegadores web, análisis y visualización de datos geoespaciales o analíticas de seguridad, entre otros. Entre sus ventajas destacan:

- **Rapidez.** Como Elasticsearch está desarrollado sobre Lucene, es excelente en la búsqueda de texto completo. Elasticsearch también es una plataforma de búsqueda en casi tiempo real, lo que implica que la latencia entre el momento en que se indexa un documento hasta el momento en que se puede buscar en él es muy breve: típicamente, un segundo. Como resultado, Elasticsearch está bien preparado para casos de uso con restricciones de tiempo como analíticas de seguridad y monitoreo de infraestructura.
- **Distribuido.** Los documentos almacenados en Elasticsearch se distribuyen en distintos contenedores conocidos como *shards*, que están duplicados para brindar copias redundantes de los datos en caso de que falle el hardware. La naturaleza distribuida de Elasticsearch le permite escalar horizontalmente a cientos (o incluso miles) de servidores y gestionar *petabytes* de datos.
- **Amplio conjunto de características.** Además de su velocidad, la escalabilidad y la resistencia, Elasticsearch tiene muchas características integradas poderosas que contribuyen a que el almacenamiento y la búsqueda de datos sean incluso más eficientes, como data *rollup* y gestión de ciclo de vida del índice.
- **Simple.** La integración con Beats y Logstash facilita el proceso de datos antes de indexarlos en Elasticsearch. Y Kibana provee visualización en tiempo real de los datos de Elasticsearch así como UI para acceder rápidamente al monitoreo de rendimiento de aplicaciones (APM), los logs y los datos de métricas de infraestructura.

Un índice de Elasticsearch es una colección de documentos relacionados entre sí. Elasticsearch almacena datos como documentos JSON. Cada documento correlaciona un conjunto de claves (nombres de campos o propiedades) con sus valores correspondientes (textos, números, Booleanos, fechas, variedades de valores, geolocalizaciones u otros tipos de datos).

Elasticsearch usa una estructura de datos llamada índice invertido, que está diseñado para permitir búsquedas de texto completo muy rápidas. Un índice invertido hace una lista de cada palabra única que aparece en cualquier documento e identifica todos los documentos en que ocurre cada palabra.

Durante el proceso de indexación, Elasticsearch almacena documentos y construye un índice invertido para poder buscar datos en el documento casi en tiempo real. La indexación comienza con la API de índice, a través de la cual puedes agregar o actualizar un documento JSON en un índice específico.

Logstash es uno de los productos principales del *Elastic Stack*, se usa para agregar y procesar datos y enviarlos a Elasticsearch. *Logstash* es un flujo de procesamiento de datos *open-source* y del lado del servidor que permite introducir datos de múltiples fuentes simultáneamente y enriquecerlos y transformarlos antes de que se indexen en Elasticsearch.

Kibana (Figura 3.1) es una herramienta de visualización y gestión de datos para Elasticsearch que brinda histogramas en tiempo real, gráficos circulares y mapas. *Kibana* también incluye aplicaciones avanzadas, como *Canvas*, que permite a los usuarios crear infografías dinámicas personalizadas con base en sus datos, y *Elastic Maps* para visualizar los datos geoespaciales.

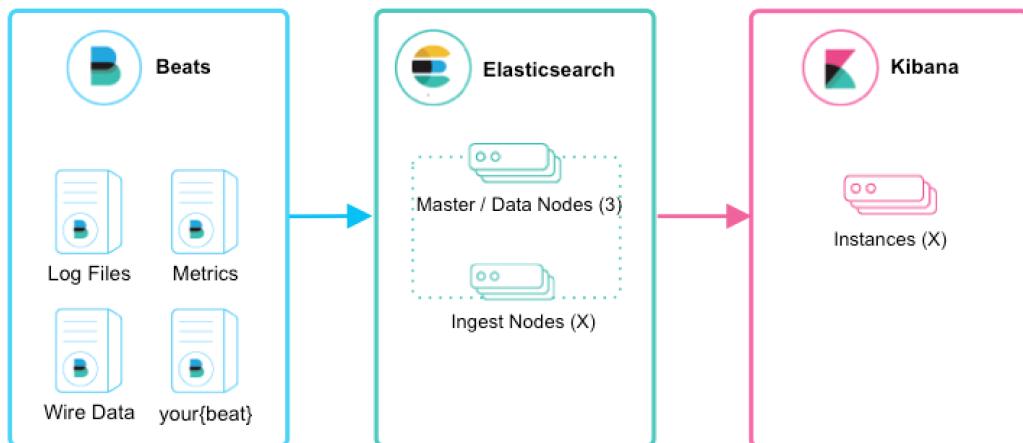


Figura 3.1: Arquitectura de Elasticsearch, Beats y Kibana

3.5. Gestores de paquetes: NPM y Webpack

Un gestor de paquetes es una herramienta software que permite automatizar los procesos de instalación, actualización y eliminación de otro software. Para este proyecto se han utilizado *NPM* (v3.5.2) y *Webpack* (v4.41.2) para la generación de los *bundles* referentes a toda la funcionalidad de *WebSim* dentro de *Kibotics*.

3.5.1. Node Package Manager (NPM)

Node Package Management [11] es un sistema de gestión de dependencias para *Node.js*, que es un entorno de ejecución de *JavaScript*. Permite, configurando un fichero (*package.json*), descargar los paquetes necesarios para el correcto funcionamiento de la aplicación. Para instalar los paquetes declarados en el fichero basta con ejecutar “*npm install*” en el directorio del fichero e, incluso se puede configurar para que se lancen scripts una vez terminada la instalación o el empaquetamiento de los *bundles*. si se combina con *Webpack*.

Webpack [12] es un sistema de *bundling* para empaquetar una aplicación web en producción. Se puede considerar la evolución de *Grunt* [13] y *Gulp*[14] dado que permite automatizar procesos como compilar y transpilar. En este proyecto se ha utilizado para hacer el *bundling* del API de los editores y del simulador de la plataforma *Kibotics*.

3.6. WebRTC

WebRTC⁴ [15] es un proyecto libre de código abierto bajo una licencia BSD que proporciona a los navegadores web y a las aplicaciones móviles comunicación en tiempo real (*Real Time Connection* (RTC)) a través de APIs sencillos⁵. Permite que la comunicación audio, vídeo y datos funcione con conexión *Peer To Peer*, es decir entre máquinas sin necesidad del uso del servidor o de plugins. Esta plataforma está siendo estandarizada por medio de W3C⁶ (World Wide Web Consortium) y del IETF⁷ (Internet Engineering Task Force). En este trabajo lo hemos utilizado para los juegos compartidos síncronos, tanto el chat como la compartición de la ventana del simulador. Por ello, lo describiremos con cierto detalle.

WebRTC tiene soporte en Apple, Google, Microsoft y Mozilla, entre otros. Es mantenido por Google. Su implementación se realiza en *JavaScript*, y sus principales interfaces son:

⁴<https://es.wikipedia.org/wiki/WebRTC>

⁵https://developer.mozilla.org/es/docs/Web/API/WebRTC_API

⁶<https://www.w3.org/>

⁷<https://www.ietf.org/>

- **getUserMedia**⁸. Este método solicita al usuario permisos para usar un dispositivo de entrada de vídeo y/o audio o compartir la cámara y/o micrófono. Si el usuario concede estos permisos, entonces le devolverá un *Promise* que es resuelto por el resultado del objeto *MediaStream*.
- **PeerConnection**⁹. Esta interfaz representa una conexión *WebRTC* entre una computadora local y un par remoto. Provee métodos para: conectar un equipo remoto, mantener y monitorear esa conexión y cerrar la conexión cuando no se necesite.
- **DataChannels**¹⁰. Es un canal de datos en *WebRTC* y te permite enviar texto o datos binarios a través de una conexión activa punto a punto. Esto permite la implementación de chats, por ejemplo.

3.6.1. Protocolos

WebRTC utiliza distintos protocolos para conseguir que el establecimiento de la conexión sea correcto.

- **ICE** (*Interactive Connectivity Establishment*). Es un sistema que permite a tu navegador conectarse con otros pares. Hay multitud de razones por las que esta conexión podría no funcionar. Se necesita superar el cortafuegos que protege la apertura de conexiones, una dirección IP pública que la mayoría de dispositivos no tiene y transmitir los datos a través de un servidor si tu *router* no permite la conexión con otros pares. ICE utiliza servidores STUN y TURN para conseguir esto.
- **STUN** (*Session Traversal Utilities for NAT*). Es un protocolo de descubrimiento de tu IP pública y determina cualquier restricción en tu *router* que restrinja una conexión de pares directa (Figura 3.2).
- **NAT** (*Network Access Translation*). Se utiliza para dar visibilidad a dispositivos que se encuentran en una red privada. Los dispositivos en una red privada tienen sus propias IPs y, desde fuera, comparten la misma IP pública. Para ello el *router* tendrá

⁸<https://developer.mozilla.org/es/docs/Web/API/MediaDevices/getUserMedia>

⁹<https://developer.mozilla.org/es/docs/Web/API/RTCPeerConnection>

¹⁰https://developer.mozilla.org/es/docs/Games/Techniques/WebRTC_data_channels

una IP pública y hará una traducción de su IP pública a la IP privada del dispositivo. De esta manera, cada dispositivo no necesita una IP pública en Internet, pero es posible acceder a él. Algunos enruteadores tienen restricciones sobre qué dispositivos pueden acceder a los que están conectados a él. Es por esto que, en esos casos, es necesario recurrir a TURN.

- **TURN** (*Traversal Using Relays around NAT*). Algunos routers utilizan una restricción llamada “NAT simétrico”. Esto significa que el router sólo acepta conexiones a pares a los que has estado conectado anteriormente. TURN está diseñado para saltar esta restricción creando una conexión a un servidor TURN y transmitiendo la información a través de ese servidor. De esta manera todos los paquetes de la conexión irán destinados a un servidor en lugar de a un dispositivo y será el servidor el encargado de retransmitir los paquetes al otro par. Esto supone una sobrecarga por lo que sólo se utiliza si no hay más opciones (Figura 3.3).

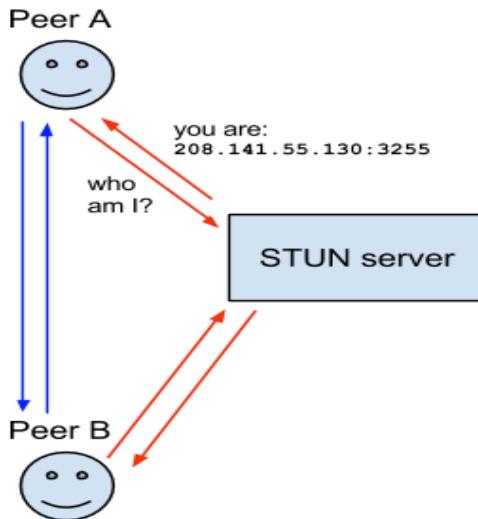


Figura 3.2: Protocolo STUN

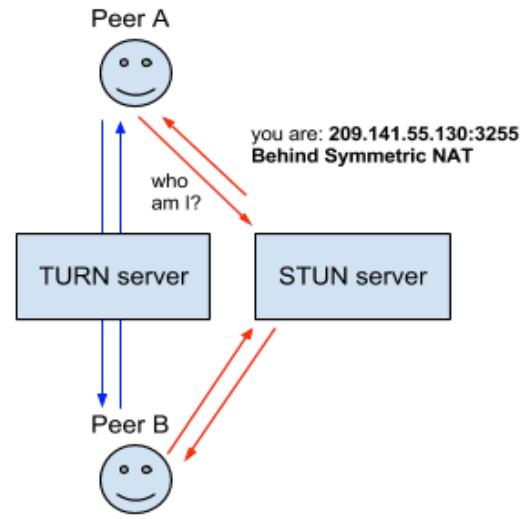


Figura 3.3: Protocolo TURN

3.6.2. Gestión de la conexión

Aunque WebRTC es capaz de crear conexiones entre pares, desafortunadamente, necesita de un servidor auxiliar con el que cambiar la información necesaria para el establecimiento de la conexión. Esto se llama canal o servicio de señalización.

La información intercambiada es la oferta y la respuesta que contiene el SDP,

mencionado más adelante. El dispositivo A que será el que inicia la conexión, creará una oferta. Enviará esta oferta al dispositivo B usando el canal de señalización. El dispositivo B recibirá la oferta y creará una respuesta. Esta respuesta se enviará al dispositivo A por el canal de señalización.

3.6.2.1. Descriptores de sesión

Los descriptores de sesión (*session descriptions*) son la configuración de un extremo de una conexión WebRTC. La descripción contiene información sobre los datos multimedia que se van a enviar, su formato, el protocolo de transferencia utilizado, la dirección IP y puerto del extremo y demás información necesaria para el establecimiento de la conexión. Esta información se intercambia y se guarda utilizando la *Session Description Protocol (SDP)*¹¹.

Cuando un usuario comienza una conexión WebRTC se crea una descripción especial llamada *Offer*. Esta descripción incluye toda la información necesaria sobre su configuración propuesta para la conexión. El otro usuario responde a esto con la *Answer*, que es una descripción con su propia configuración. En este sentido, ambos dispositivos comparten toda la información necesaria para el intercambio de datos. Este intercambio es manejado utilizando ICE.

Cada usuario, ahora, mantiene dos descriptores, el descriptor local con la información de su configuración, y el descriptor remoto con la información de la configuración el otro usuario.

Este proceso de oferta/respuesta se realiza cuando se establece la conexión, pero también cada vez que es necesario modificar algún aspecto de ésta. Los pasos para este proceso son:

1. El *caller*, o usuario que inicia la conexión, recoge los datos media con *navigator.mediaDevices.getUserMedia()*.
2. El *caller* crea la *RTCPeerConnection* y utiliza la función de *WebRTC RTCPeerConnection.addTrack()* para añadir los datos media recogidos.
3. El *caller* utiliza la función *RTCPeerConnection.createOffer* para generar la oferta.

¹¹<https://tools.ietf.org/html/rfc2327>

4. El *caller* utiliza la función `RTCPeerConnection.setLocalDescription()` para establecer la oferta como descripción local.
5. Después de establecer la descripción local, el *caller* pregunta al pide al servidor STUN que genere los candidatos ICE.
6. El *caller* utiliza el servidor de señalización para transmitir la oferta al receptor de la conexión.
7. El *callee* o receptor de la conexión utiliza la función de `WebRTC RTCPeerConnection.setRemoteDescription()` para establecer la oferta como descripción remota.
8. El *callee* realiza cualquier configuración necesaria como establecer sus datos multimedia para la conexión y añadirlo a la transmisión con la función `RTCPeerConnection.addTracks()`.
9. El *callee* crea una respuesta con la función `RTCPeerConnection.createAnswer`.
10. El *callee* utiliza la función `RTCPeerConnection.setLocalDescription()` con la oferta para establecer su descripción local. En este punto el *callee* ya tiene los dos descriptores de sesión.
11. El *callee* utiliza el servidor de señalización para enviar la respuesta al *caller*.
12. El *caller* recibe la oferta.
13. El *caller* utiliza la función `RTCPeerConnection.setRemoteDescription()` para establecer la respuesta como descriptor remoto. En este punto ambos usuarios tienen las dos descripciones de sesión.

3.6.2.2. Candidatos ICE

Una vez establecidas las configuraciones de sesión en ambos usuarios, hay que intercambiar la información sobre la conexión de red. Esto se conoce como *ICE candidate* y detalla los métodos disponibles para que sea posible la comunicación entre los usuarios (directamente o a través de un servidor TURN). Normalmente, los pares envían su propuesta con los mejores candidatos en primer lugar y en orden descendiente. Los candidatos ideales son *UDP* al ser los más rápidos pero el estándar *ICE* también permite candidatos *TCP*.

Para realizar toda esta configuración, *WebRTC* ofrece el API descrito. Gracias a esto, toda esa configuración es transparente al usuario. Un ejemplo visual de esta conexión puede verse en siguiente figura.

The entire exchange in a complicated diagram

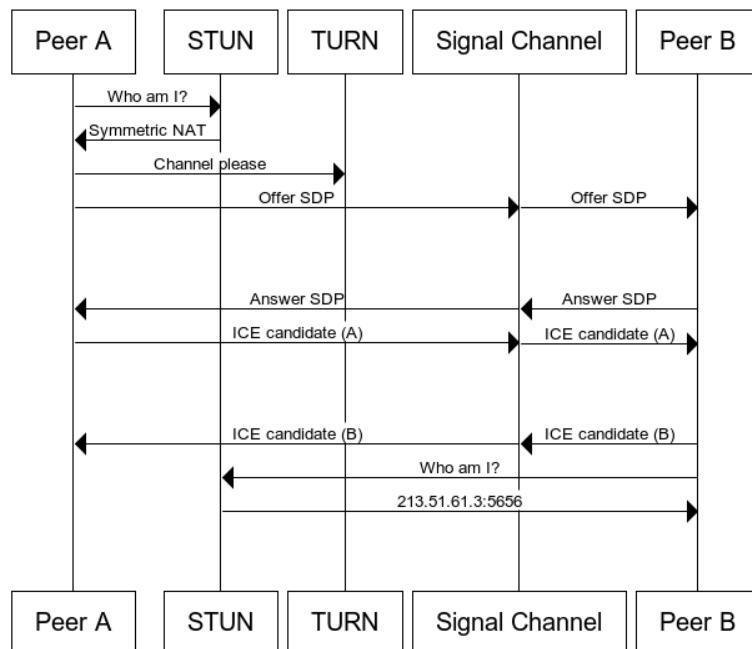


Figura 3.4: Diagrama de establecimiento de sesión en WebRTC con STUN y TURN

3.7. Retransmisión en directo

La retransmisión en directo o *Streaming*, es un término utilizado para definir la visualización de vídeos y audio en tiempo real. Básicamente hay dos tipos de *streaming*:

- **Streaming real.** También conocido como “Live Streaming”. Requiere un servidor especial que difunde la información de audio/vídeo en tiempo real. El reproductor del cliente, lo recibe y visualiza de manera instantánea. Para soportar esta tecnología, se necesitan servidores dedicados potentes con una gran cantidad de recursos y ancho de banda. Este tipo de transmisión utiliza el protocolo UDP¹² para la transmisión

¹²<https://www.ionos.es/digitalguide/servidores/know-how/udp-user-datagram-protocol/>

de datos de vídeo/audio porque soporta la pérdida de paquetes y está orientado a maximizar la velocidad de transmisión.

- **Pseudo-streaming.** También conocido como “Streaming HTTP”, es una alternativa al *Live streaming* cuando no se quieren gastar tantos recursos y dinero en servidores dedicados de difusión. Para conseguir el efecto de difusión en vivo se utilizan *buffers* de datos del vídeo. Para esto, se utiliza el protocolo TCP¹³. Este protocolo no es eficaz en transmisiones en vivo porque tiene controles de pérdidas de paquetes, ya que no soporta estas pérdidas. Este inconveniente se contrarresta al existir el *buffering*, por lo que no existirán parones en la transmisión y la calidad será mejor aunque exista un cierto retardo.

En 2010 se comenzó a desarrollar una técnica de transmisión multimedia en *streaming* llamada DASH (Dynamic Adaptive Streaming over HTTP). Se trata de una técnica de transmisión que detecta el ancho de banda y la CPU para adaptar la transmisión en función de esos parámetros. De esta manera, se consigue maximizar el ancho de banda de la transmisión. Para la transmisión de contenido MPEG (video y audio) sobre página HTTP con HTML-5 se ha desarrollado el estándar MPEG-DASH. Este estándar está orientado a la transmisión de contenido MPEG sobre páginas HTML-5 con conexión HTTP para *streaming*.

Existen plataformas dedicadas a esto. En este trabajo se han escogido dos, *Youtube Live* y *Twitch*. Cada una de ellas ofrece un API distinto para poder realizar los *streamings*.

En este trabajo se han utilizado para la retransmisión en directo y de manera masiva de los torneos automáticos.

3.7.1. API de Youtube Live

La API de *Youtube* de transmisión en vivo te permite crear, actualizar y administrar eventos en *Youtube*. Con esta API puedes programar eventos y asociarlos a transmisiones de vídeo, que representan el contenido de la transmisión. Está formada por dos elementos, la API de datos y la API de identificación de contenido. La API de datos permite a los usuarios administrar sus cuentas, mientras que la API de identificación de contenido

¹³https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisi%C3%B3n

permite interactuar sobre el sistema de administración de derechos. Sin embargo, todos los recursos de los que está compuesta la API de transmisión en vivo se utilizan solo para crear y administrar eventos en vivo.

Para poder utilizar esta API de *Youtube* es necesario acceder a la consola API de *Google* por lo que necesitas una cuenta de *Google* y autorizar tu aplicación en la consola API para que te den un clave de autorización. Con esta clave es posible activar la transmisión en vivo habilitando los campos *YouTube Data API v3* y *YouTube Content ID API*.

3.7.2. API de Twitch

Para acceder a la API de *Twitch* de transmisión en vivo es necesario tener una cuenta activa. Además, ofrece un panel de control para gestionar las conexiones de manera gráfica y extensiones para personalizar y añadir más funcionalidad a la plataforma.

Para poder transmitir, tanto en *Youtube* como en *Twitch*, es necesario utilizar programas de terceros, en este caso se ha utilizado *OBS Studio* [16] para realizar la captura de vídeo.

Gracias a esto, la API de *Twitch*, está orienta a la difusión de contenido por *streaming*. Por ello, ofrecen la posibilidad de difusión de su contenido embebido en un *I-Frame*, que se crea con una sola función. Además, *Twitch* ofrece su API de manera inmediata, sin necesidad alguna de activación ni de llamadas a APIs externas puesto que ofrece un paquete con la propia API (<https://embed.twitch.tv/embed/v1.js>).

Capítulo 4

Juegos compartidos síncronos

En este capítulo abordaremos toda la infraestructura relacionada con los *Juegos compartidos síncronos*. En la primera parte mostraremos las mejoras introducidas, a nivel visual, en la plataforma para, más adelante, centrarnos en el desarrollo de las nuevas plantillas, las mejoras en el servidor, la incorporación de WebRTC en la plataforma y la mejora de los evaluadores.

4.1. Antecedentes

En la plataforma *Kibotics* existe una base previa a estos ejercicios, se llaman “juegos competitivos” o “Juegos compartidos *asíncronos*”. En este tipo de ejercicios el usuario compite siempre contra el mismo usuario, que no se puede elegir, DonPerfecto, cuyo código se carga en un robot específico en el momento de acceder al ejercicio y no es necesaria su participación en vivo en el propio ejercicio. Tampoco se realiza una conexión *WebRTC* dado que sólo interviene un usuario y un solo navegador.

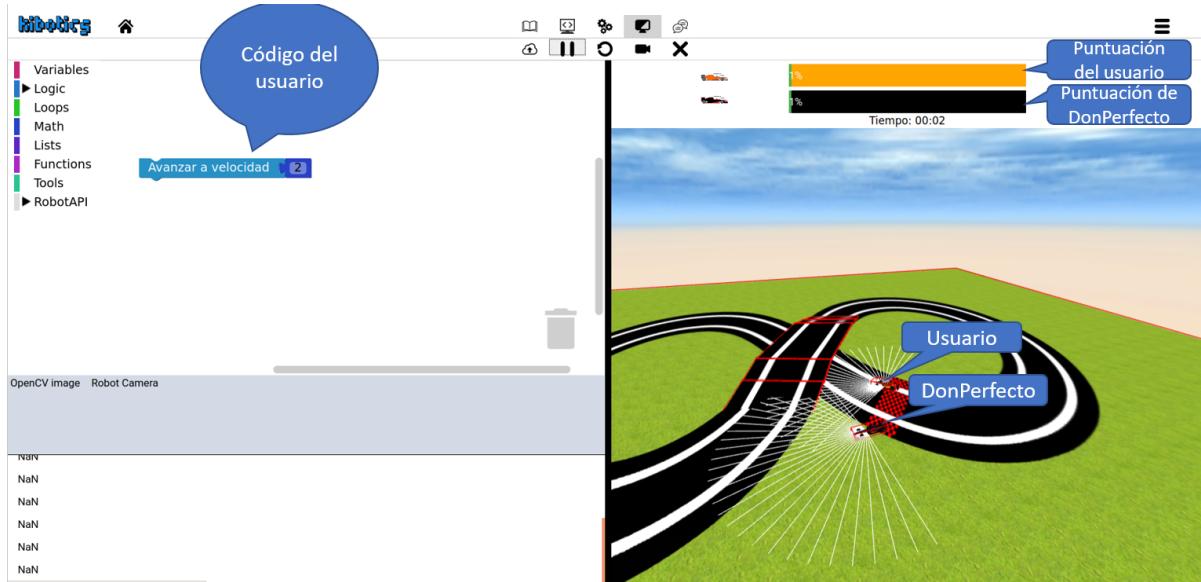


Figura 4.1: Ejecución de un juego competitivo

4.2. Uso de los juegos compartidos síncronos

Aunque estos nuevos ejercicios comparten algunas ventanas como la de bienvenida-teoría, editor y foro, ha sido necesario la creación de una nueva ventana llamada “Competición” (Figura 4.2). En esta ventana aparece multitud de elementos nuevos como el chat, la barra de búsqueda de usuario, los botones de control de ejecución o los controladores de estado de la conexión.

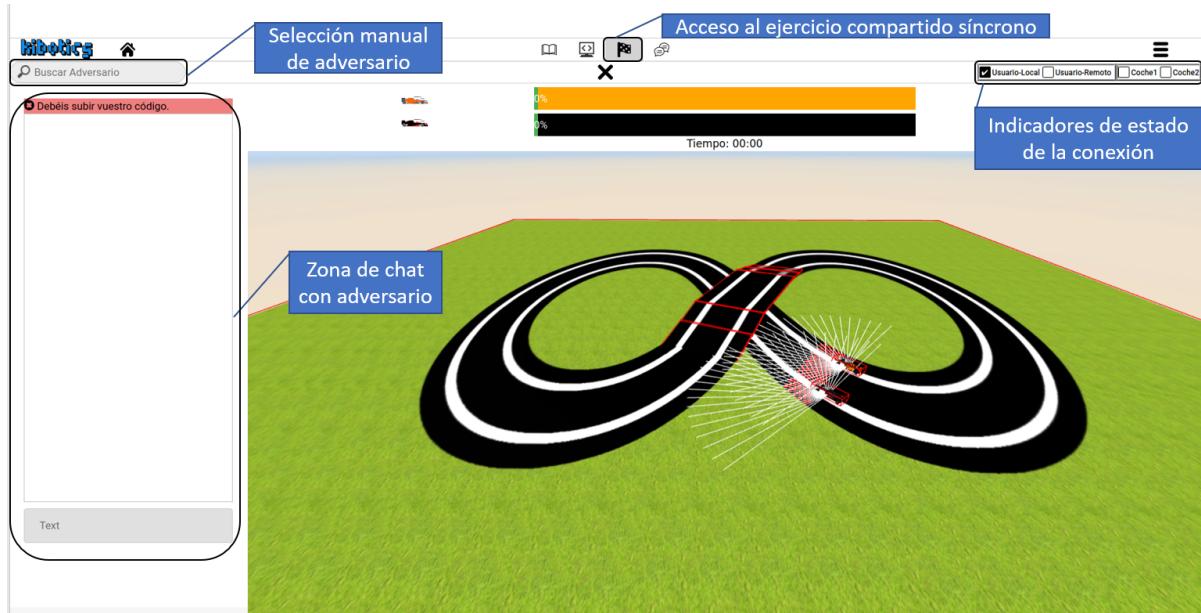


Figura 4.2: Pestaña de competición

Para uno de los participantes, el usuario emisor, con el simulador ejecutando en local. Para el otro participante, el usuario receptor, es muy parecido, pero el vídeo del escenario simulado proviene de una conexión con el emisor.

Los pasos para realizar una conexión en estos ejercicios son muy sencillos:

1. Buscar el nombre de usuario del contrincante.
2. Que el contrincante acepte la petición. En ese momento se iniciará automáticamente la retransmisión en *WebRTC* y el chat.
3. Cargar el código en el coche para los dos usuarios. Cada uno en un coche pero de manera elegible por los usuarios.
4. Comenzar la ejecución.

El resultado de estos pasos es la disputa de un enfrentamiento con otro usuario de manera síncrona, dado que ambos usuarios deben estar presentes (Figura 4.3).

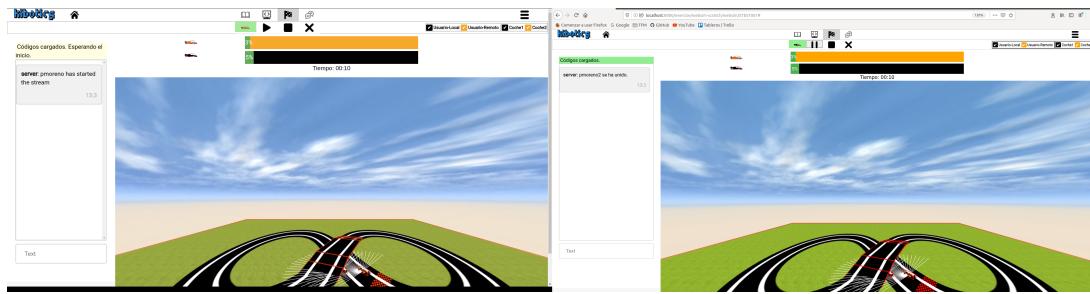


Figura 4.3: Ejemplo de ejecución

Uno será el emisor y otro el receptor. El simulador ejecutará realmente en el ordenador del emisor y se retransmitirá en vivo al receptor.

4.3. Diseño

Para poder realizar este tipo de ejercicios ha sido necesaria la creación de un *software* específico para que funcionen de manera síncrona, partiendo de los antecedentes de juegos compartidos existentes de la plataforma Kibotics.

Aparecen 3 agentes característicos: el servidor y cada uno de los dos navegadores de los usuarios participantes. Ambos navegadores utilizan el servidor web de *Kibotics* como servidor de señalización para conocer su propia dirección y la dirección del contrincante, así como el usuario emisor para recoger los códigos fuente de los ejercicios. Los navegadores se interconectan entre ellos gracias al servidor de señalización y establecen una conexión *WebRTC* para retransmitir el simulador de una navegador al otro (Figura 4.4).

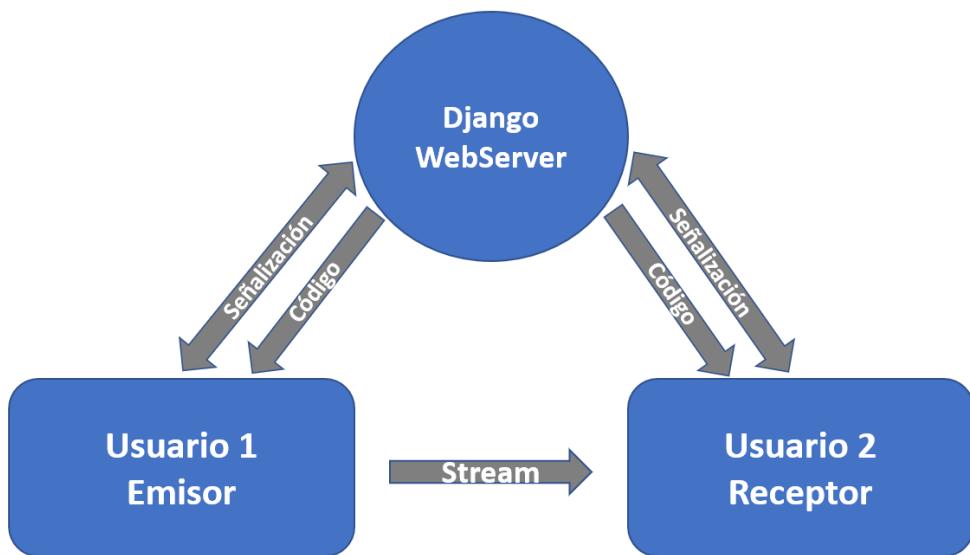


Figura 4.4: Diseño de los juegos compartidos síncronos

El servidor web de *Kibotics* además envía las respectivas páginas web a ambos navegadores, las páginas presentadas en la sección 4.2.

4.4. Elección de contrincante y chat

Para poder enlazar los usuarios mediante *WebRTC* ha sido necesaria la implementación de una barra de búsqueda de usuarios para que el servidor web sea capaz de realizar la conexión por nombres de usuarios en la plataforma. Aprovechando esta conexión, se ha creado un chat para que los usuarios puedan conectarse.

4.4.1. Barra de búsqueda de usuario

En esta barra, el usuario debe introducir el login del contrincante para que el servicio web pueda realizar la búsqueda entre los usuarios conectados actualmente en la plataforma. De esta manera se ha desarrollado un *handler* en *JavaScript* que espera la inserción del botón “Enter” para recoger el valor que tiene la barra de búsqueda y enviarlo, mediante *WebSocket* al servidor.

El servidor recoge ese mensaje entrante por el *WebSocket* y, gracias a un paquete

instalado en *Django* llamado “*Django-online-users*”, se accede a la API que proporciona y podemos saber los usuarios que están en línea y dentro del ejercicio. Gracias a la llamada `OnlineUserActivity.get_user_activities()`, el paquete devuelve un array con los distintos usuarios. De esta manera, sólo es necesario realizar una búsqueda sobre el nombre de usuario contra el que se quiera competir. En el caso en que la búsqueda sea errónea, se enviará un mensaje al usuario informando de la imposibilidad de encontrar al contrincante especificado. Si la búsqueda ha sido satisfactoria, se enviará un mensaje por el *WebSocket* al contrincante con la petición.

El navegador, en el lado del usuario receptor, recibirá esa petición y mostrará un mensaje avisando de este hecho (Figura 4.5).

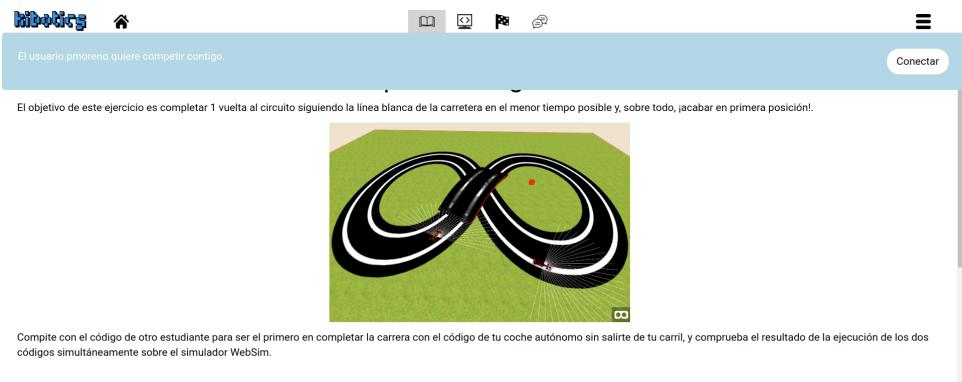


Figura 4.5: Petición de competición al contrincante

Una vez de vuelta aparecerá este mensaje, el contrincante pulsará el botón “Comenzar” y se enviará un mensaje al servidor vía *WebSocket* que lo retransmitirá al usuario para comenzar a realizar la conexión *WebRTC* de manera automática.

4.4.2. Sección de chat

En la venta de competición se puede observar cómo, además del simulador, aparece una nueva sección con un chat. Este chat ha sido implementado con *WebRTC* y *DataChannels* directamente entre los navegadores para disminuir la carga de mensajes al servidor web de la plataforma. De esta manera, los mensajes del chat se envían mediante una conexión *peer to peer*.

Para desarrollar esto se ha creado un *script* en *JavaScript* con toda la configuración

de la conexión WebRTC para transmisión de datos.

En primer lugar se ha utilizado la función `createDataChannel()` para generar la conexión y la función `receiveChannelCallback` para recoger los mensajes, en ambos pares de la conexión. Además, ha sido necesario el desarrollo de una función para recoger el valor de los mensajes tecleados y enviarlos por el *DataChannel*.

4.5. Ejecución compartida y evaluadores

Una vez establecida la conexión *WebRTC*, es necesario realizar la configuración del código en cada uno de los coches así como controlar la ejecución y conocer el estado de la conexión.

4.5.1. Botones de control de la ejecución

Como hemos visto en la figura 4.2, se pueden visualizar algunos botones distintos a los de otros tipos de ejercicios. Esto es así porque la ejecución de éstos depende del sincronismo entre usuarios.

En primer lugar, la barra de control de ejecución sólo mostrará el botón de cerrar la ventana, puesto que es necesario buscar al contrincante. Una vez realizada la conexión, y de manera automática, se muestran los botones de selección de coche para cargar el código, finalización de la competición y el de cerrar la ventana.

- *Botones de selección de coche.* Estos dos botones permiten al usuario elegir en qué coche estará cargado su código.
- *Botón de finalización de la competición.* Este botón permite a los usuarios finalizar la conexión *WebRTC* y volver al estado original para poder seleccionar otro usuario sin necesidad de salir del ejercicio.
- *Botón de cierre de ventana.* Este botón permite al usuario volver a la página principal del ejercicio sin terminar la conexión.

Una vez cargados el código de los usuarios en cada coche, se mostrará el botón de inicio para cada usuario. Como un usuario es el receptor de transmisión, ha sido necesario

configurar su botón de inicio para que le envíe una señal al usuario que está transmitiendo y se inicie la simulación. De esta manera, ambos usuarios pueden controlar la ejecución de manera individual (Figura 4.6).



Figura 4.6: Botones de control de ejecución tras cargar los códigos

4.5.2. Indicadores del estado de la conexión

Para que los usuarios conozcan el estado de la conexión en todo momento, se han desarrollado una serie de indicadores:

- **Conexión del usuario 1.** Indicador que muestra si el primer usuario está conectado. Aparecerá marcado por defecto ya que se trata del usuario local.
- **Conexión del usuario 2.** Indicador que muestra el estado de la conexión con el usuario remoto. Si la conexión todavía no ha sido realizada, estará desmarcado.
- **Código en el coche 1.** Indicador que muestra si algún código ha sido cargado en el primer coche.
- **Código en el coche 2.** Indicador que muestra si algún código ha sido cargado en el segundo coche.

A continuación, se muestran algunos ejemplos de estos indicadores:



Figura 4.7: Indicadores con ambos usuarios conectados

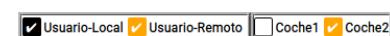


Figura 4.8: Indicadores con un coche cargado

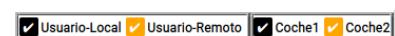


Figura 4.9: Indicadores con ambos coches cargados

Toda esta configuración de indicadores se realiza mediante mensajes vía *WebSocket* entre los usuarios y el servidor de señalización.

4.5.3. WebRTC

Una vez elegido contrincante y recibida la petición, ha sido necesario el desarrollo de todo los pasos descritos en la sección 3.6.2.1 para que la conexión se establezca de manera automática.

Para realizar la conexión, cada cliente necesita un código distinto dependiendo de si es el *caller* (usuario emisor) o el *callee* (usuario receptor). Todo este código se ha recogido en un fichero llamado “webrtc.js”.

En primer lugar, se ha creado una función llamada `startStreaming()` en la que el *caller* realiza los pasos 1-6 de la conexión WebRTC.

Una vez enviada la oferta al servidor de señalización mediante *WebSocket*, reenvía la oferta al *callee*. El *callee* recibe esta oferta y, mediante la función `startRemoteStreaming()`, realiza los pasos 7-11 y envía la respuesta al servidor de señalización mediante *Websocket* una vez más.

El servidor de señalización envía la respuesta al *caller* que, gracias a la función `setAnswer()`, acepta la respuesta. En este punto ambos usuarios tienen establecidas las configuraciones locales y remotas de la conexión y comienzan a enviar los candidatos ICE.

Para aceptar los candidatos ICE y escoger el que mejor se ajuste a ambos, los usuarios deben enviar sus candidatos al servidor de señalización (función `onIceCandidate()`, que el otro usuario los acepte y le reenvíe la confirmación (función `onAddIceCandidate()`).

Gracias a esto, el *callee* no necesita tener en simulador activado, sólo un elemento HTML que soporte la recepción *WebRTC*, un elemento “video”.

Para soportar este establecimiento de comunicación, se ha tenido que desarrollado el servidor de señalización en el servidor *Django*. Para ello, *Django* ofrece la posibilidad de crear este servidor con *Django-channels*.

Para configurar los *Django-channels*, se ha creado un fichero llamado “*consumer.py*” en el que se recogen todos los mensajes del *WebSocket*, se analizan y se responde gracias a los “*Django-groups*” o grupos de *Django* que están enlazados en el mismo *WebSocket*. Como la plataforma *Kibotics* cuenta con la versión v1.5 de *Django-channels*, ha sido necesaria la implementación de distintas variables de datos en las que se recogen los usuarios y sus canales para poder realizar reenvíos selectivos, algo que en esta versión no está soportado.

Además, se ha tenido que dotar de toda la funcionalidad necesaria para gestionar cada uno de los distintos mensajes, ya sean de configuración de *WebRTC*, de búsqueda de usuarios o de petición de códigos para que el servidor tenga la funcionalidad necesaria para responderlos.

4.5.4. Carga flexible de código

Para facilitar a los usuarios la certeza de qué coche es el suyo, se ha desarrollado un módulo *software* que permite una carga flexible del código del usuario en cualquiera de los coches. Esto significa que cada usuario puede cargar su código en el coche que prefiera.

Para ello, ha sido necesario una implementación nueva en el API que ofrece el simulador *WebSim* para recoger el código asignado a cada coche para cada usuario. Se ha realizado una actualización del código que tiene el simulador antes de iniciar la ejecución de la simulación. Anteriormente, el simulador recogía el código guardado por el usuario y el código descargado del usuario “DonPerfecto” al iniciar el simulador. Con esta mejora, los códigos cargados en los coches serán los últimos actualizados por ambos usuarios.

Para que el receptor pueda enviar su código fuente al usuario anfitrión de la simulación, basta con que modifique su código y lo guarde. Una vez que escoja el coche, se envía un mensaje al servidor mediante *WebSocket* y el servidor realiza una petición de código fuente a *GitHub* con el nombre del usuario requerido. Una vez obtenido este código, lo envía al usuario anfitrión que guarda el código en el coche correspondiente.

Una vez que uno de los dos usuarios ha escogido un coche, éste se muestra en verde y desaparece para el otro usuario, de esta manera se evitan conflictos en cuanto al guardado múltiple en el mismo robot (Figuras 4.10 y 4.11).



Figura 4.10: Botones de carga iniciales sin tener códigos cargados



Figura 4.11: Botón de carga tras cargar un código

Además, se ha configurado la opción de poder retirar el código guardado en el coche para que el usuario pueda realizar modificaciones y guardar el nuevo código actualizado.

Para poder realizar esta característica de manera sincronizada ha tenido que desarro-

llarse, tanto en el lado del cliente como en el lado del servidor, todo el procesamiento de mensajes y desarrollo de tareas.

4.5.5. Ejercicios

Para poder desarrollar los *juegos compartidos síncronos* ha sido necesario la generación de 4 ejercicios nuevos a partir de dos ya existentes. Aprovechando la creación de los ejercicios competitivos donde los alumnos competían contra un usuario preestablecido, se han generado dos ejercicios para *Scratch* y otros dos para *Python* (ejercicio atravesabosque y sigue-línea) (Figuras 4.12, 4.13, 4.14 y 4.15). Para diferenciarlos, ha sido necesario la creación de un campo nuevo que especifica si se trata de un ejercicio competitivo o de un juego compartido. En el lado del servidor ha sido necesario el procesado de este nuevo cambio para renderizarlo con las plantillas.

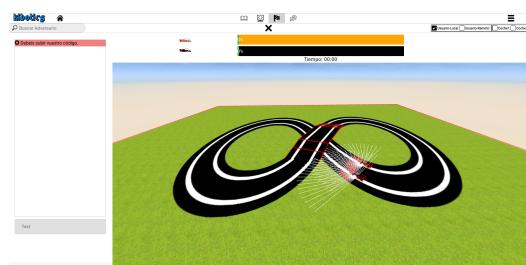


Figura 4.12: Juego compartido Sigue-líneas en Scratch

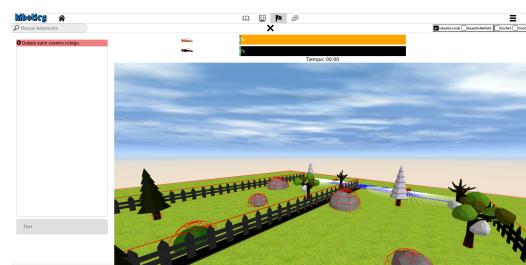


Figura 4.13: Juego compartido Atravesabosque en Scratch

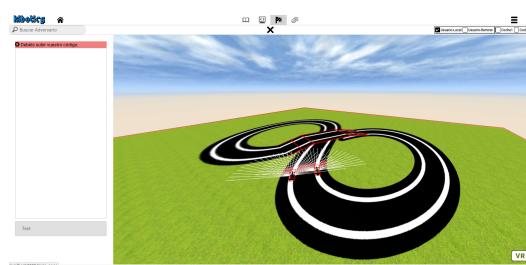


Figura 4.14: Juego compartido Sigue-líneas en Python

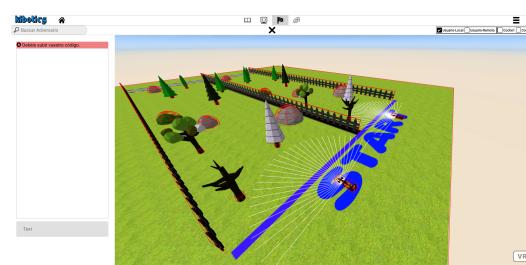


Figura 4.15: Juegos compartido Atravesabosque en Python

4.5.6. Evaluadores automáticos

Por último, aunque la conexión *WebRTC* ha solucionado el problema de la transmisión del simulador, ha sido necesario el desarrollo de las instrucciones capaces de sincronizar los evaluadores. Para ello, se han aprovechado los *DataChannels* de *WebRTC* creados para los mensajes del chat y se ha desarrollado un código capaz de recoger el estado de cada evaluador en el simulador de la máquina del emisor y enviarlo al otro usuario, al receptor. En el otro cliente ha sido necesario desarrollar la recepción y procesado de estos mensajes para realizar la sincronización (Figura 4.16).

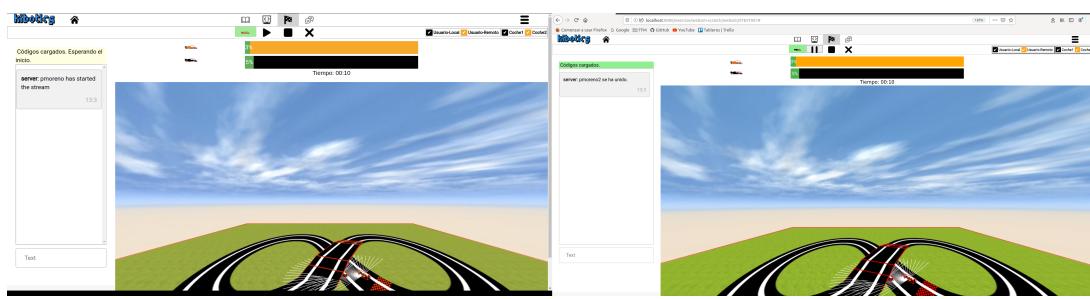


Figura 4.16: Ejemplo de sincronismo entre evaluadores

Para una mejor visualización del funcionamiento de estos juegos, hay dos vídeos, disponibles en Youtube, uno en *Scratch*¹ y otro en *Python*².

¹<https://youtu.be/Wnhkljs-HT8>

²<https://youtu.be/gZGvd1gcHPE>

Capítulo 5

Torneos

Una vez completados los *juegos compartidos síncronos*, se ha dado el paso final en cuanto a la extensión de la gamificación dentro de la plataforma *Kibotics*. En esta línea se han desarrollado dos tipos de torneos automáticos en *Kibotics*, el primero basado en ranking por puntos y el segundo en enfrentamientos directos. Para poder desarrollar todo esto, se han creado una plantilla HTML para cada torneo, así como el código necesario en los ejercicios y evaluadores para recoger las puntuaciones y una nueva figura llamada maestro de ceremonias para el desarrollo de los torneos.

5.1. Antecedentes

Los ejercicios sobre los que se han basado los torneos provienen de dos fuentes distintas, en función del torneo. La primera fuente son los ejercicios de “Retos Individuales”, en concreto el “Sigue-líneas” en *Python* y en *Scratch*. Estos retos están formados por un solo robot, con su correspondiente evaluador, y han dado pie a los torneos de ranking. La segunda fuente son los “juegos compartidos síncronos” que ya cuentan con dos robots y sus correspondientes evaluadores. Este tipo de juegos han dado pie a los torneos de enfrentamientos.

La principal diferencia con los retos y los juegos, fuera de los torneos, es que, desde dentro de los torneos los evaluadores se encargan de enviar la información necesaria al servidor web para guardarla en una base de datos y, después, mostrarse en su correspondiente página del torneo de manera automática. Otra diferencia es que los

usuarios de la plataforma lanzan los retos y los juegos fuera de los torneos y tienen un papel activo. En contraste, para los torneos se ha creado la figura del maestro de ceremonias que se encarga de realizar las ejecuciones. Los evaluadores detectan esta figura y, una vez terminada la ejecución, realizan el envío de datos.

Además, para facilitar a los usuarios la visualización de los torneos en internet, se ha desarrollado un script de difusión masiva en *Twitch* y *Youtube Live* para que, desde la página de torneos, los usuarios tengan una ventana de visualización del torneo.

5.2. Diseño

El diseño realizado (Figura 5.1) para ambos torneos es muy similar, consta de las mismas partes y se realiza en manera muy parecida. Tiene 3 partes principales muy diferenciadas:

- **Página del ejercicio.** Entorno de ejecución de la simulación. Incluye el evaluador, que es el encargado de la recogida y transmisión de los resultados de la ejecución al servidor. También se tiene en cuenta la parte del servidor que se encarga de procesar la petición y hace la llamada a la base de datos de *Elasticsearch*.
- **Base de Datos.** Mediante la petición de *Elasticsearch* el servidor guarda en la base de datos toda la información correspondiente a la ejecución del ejercicio.
- **Página web de torneo.** En el servidor se ha creado una página web que renderiza los datos obtenidos hasta e momento relativo al torneo(puntuación, ganadores de los enfrentamientos, etc) y muestra los resultados parciales del torneo.

Además de estos puntos, ambos tipos de torneo cuentan con dos elementos adicionales que le aportan características distintivas la desarrollo. Para automatizar los torneos se han generado unos ficheros JSON (uno para cada tipo de torneo) en el que se instancian las ejecuciones. De esta manera, desde el ejercicio, se cargan los datos y se ejecutan los códigos.

También incluye la retransmisión del ejercicio en curso actualmente, de modo que todos los aficionados pueden seguir en vivo, continuamente, el desarrollo del torneo desde

su comienzo hasta el final, con un ganador. Gracias a una ventana en la página de los torneos, se pueden visualizar las ejecuciones de manera masiva.

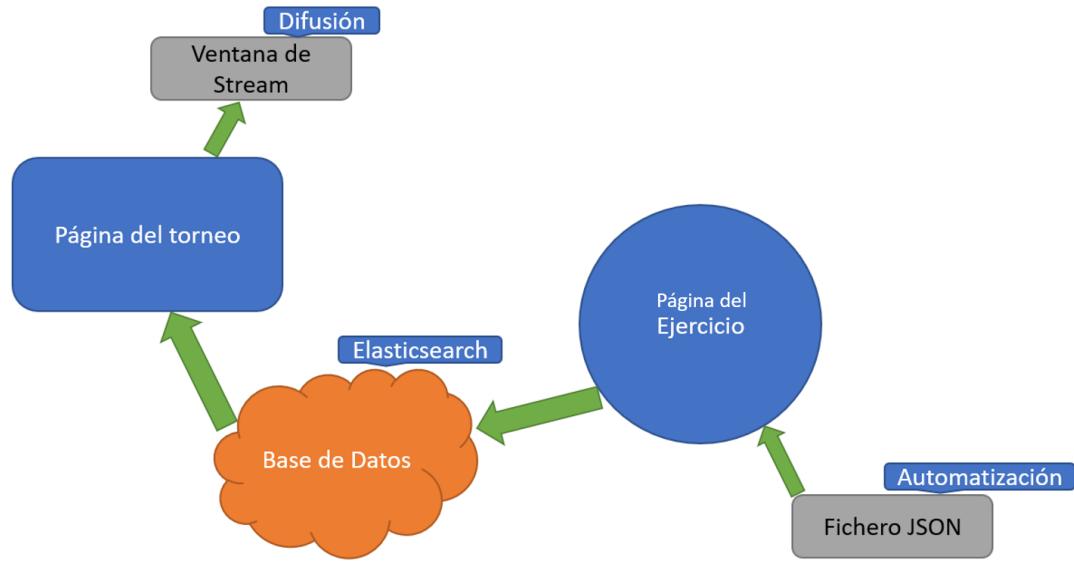


Figura 5.1: Diseño de la infraestructura de los torneos

5.2.1. Torneos de ranking

Este tipo de torneos se caracteriza por soportar un solo robot en el escenario simulado y un solo evaluador. De esta manera, el torneo está compuesto por ejecuciones en el mismo escenario del código de cada usuario participante. Estos torneos se parecen a las cronometradas individuales del ciclismo en la vida real. En la página web del torneo se mostrará un listado con los usuarios y su menor tiempo en una ejecución. Las peticiones de guardado en la base de datos tienen los siguientes campos:

- competition_id: *string* con el id del torneo.
- username: *string* con el nombre del usuario.
- time: *date* con el tiempo de la ejecución.
- exercise: *string* con el nombre del ejercicio.
- date: *date* con la fecha de la ejecución.

5.2.2. Torneos de enfrentamiento

Este tipo de torneos se caracteriza por soportar dos robots y dos evaluadores automáticos (uno para cada robot). De esta manera, el torneo está compuesto por una ejecución por enfrentamiento y en la que habrá un ganador y un perdedor. El ganador será el que complete una vuelta a circuito más rápido que su contrincante. En la página web del torneo se mostrará un listado con cada enfrentamiento, el resultado, el tiempo y las distintas rondas.

Las peticiones de guardado en la base de datos tendrán los siguientes campos:

- competition_id: *string* con el id del torneo.
- username1: *string* con el nombre del usuario 1.
- username2: *string* con el nombre del usuario 2.
- user_winner: *string* con el nombre del usuario ganador.
- time: *date* con el tiempo de la ejecución.
- exercise: *string* con el nombre del ejercicio.
- date: *date* con la fecha de la ejecución.

En este tipo de torneos es necesaria una lectura completa de los enfrentamientos y almacenarla en una variable DOM de *JavaScript*. Esto es así porque debe ser modificada tras cada enfrentamiento con el nuevo ganador para ir completando las rondas (Figura 5.2). Dado que no se puede saber de antemano, en el momento de arrancar la competición porque depende de los ganadores en cada ronda.

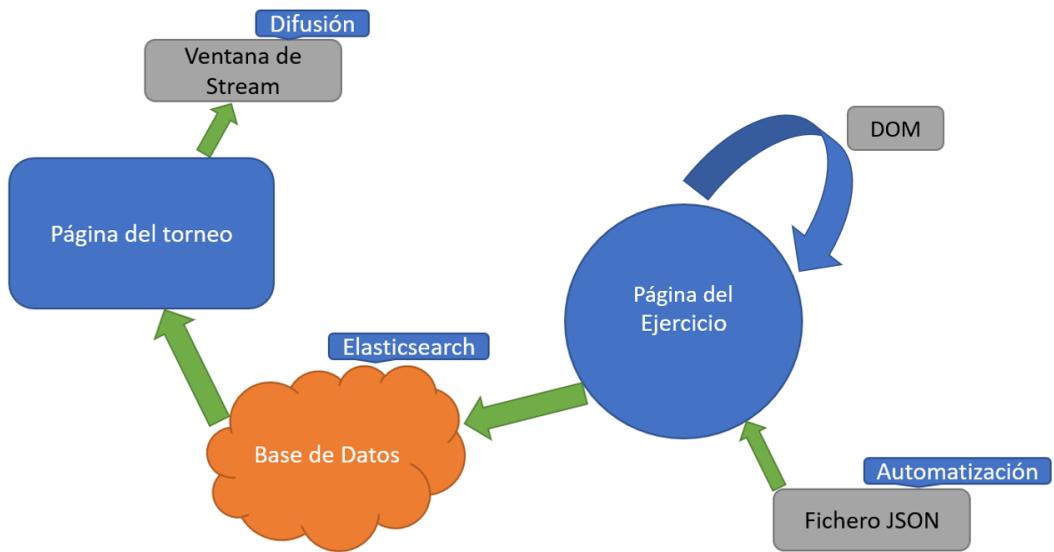


Figura 5.2: Diseño de la infraestructura de los torneos de enfrentamiento

5.3. Torneos de ranking

Este tipo de torneos se caracterizan por incluir un robot y un evaluador. Para la clasificación en el torneo se tiene en cuenta la puntuación que consiga el usuario en cada ejecución.

5.3.1. Plantilla de la página web del torneo

Estos torneos se basan en rondas de tiempo en las que el usuario con el robot más rápido será el que gane. Para ello se han recogido los datos que envían los evaluadores y se renderizan en esta plantilla mostrando las clasificaciones en orden ascendente. En caso de que dos usuarios tengan el mismo tiempo, estará en primer lugar el usuario que antes desarrolló su código.

En cuanto a la plantilla del torneo, se trata de una plantilla de *Django*, creada con HTML que incorpora los datos recibidos desde el servidor web al renderizar la página y cuenta con un “div” específico para incorporar la retransmisión en vivo del torneo (Figura 5.3).

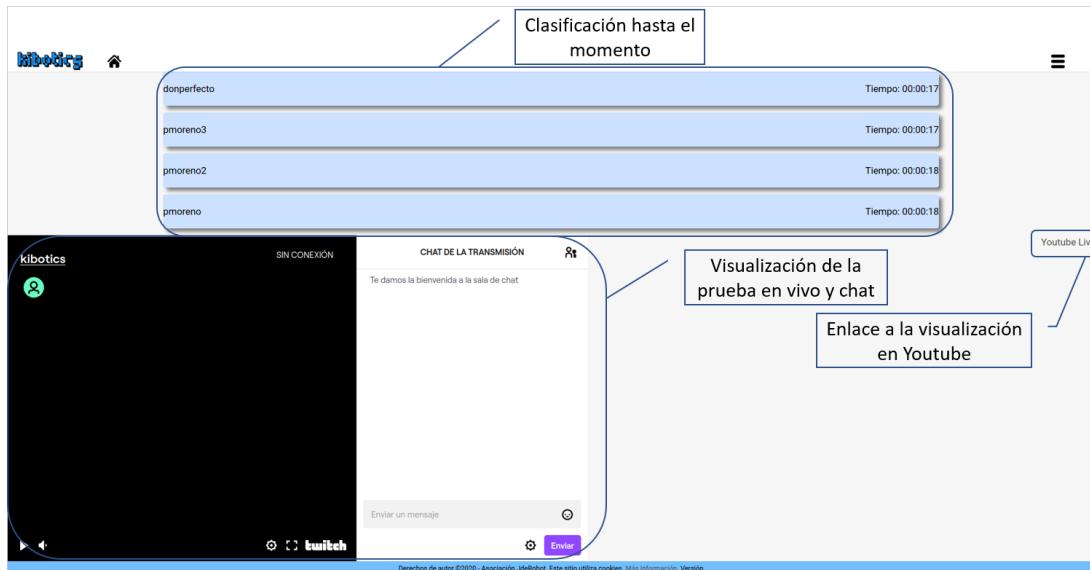


Figura 5.3: Plantilla de torneos de ranking

Gracias a esta página web, es posible la visualización del torneo para todo el mundo. De esta manera, aunque haya usuarios que no estén participando, pueden acceder a esta página y visualizar el torneo en directo gracias a la ventana de *Twitch* y pueden interactuar con otros usuarios con el chat del directo. Con esto se fomenta la interacción social de la plataforma.

5.3.2. Infraestructura del servidor

La infraestructura necesaria para estos torneos requiere: que los evaluadores envíen los resultados de cada participante al servidor web, para que se grabe en una base de datos, y que la página web lea continuamente esa base de datos y muestre sus contenidos, es decir, los resultados hasta el momento, en un página web.

Para permitir el acceso a la base de datos, ha sido necesario la implementación de un modelo de datos en *Elasticsearch* para los torneos de ranking.

En primer lugar, el servidor realiza una petición de una *query* en *Elasticsearch* con la función

```
Search(index="kibotics_ranking_log").query({"match": {"competition_id": "1"}}).filter()
```

Esta función devuelve un tipo de datos de *Elasticsearch* que se recorre con la función

`scan()`. Mientras recorremos la *query* de datos, recogemos el tiempo y el usuario.

El principal problema reside en que *Elasticsearch* no admite guardar tiempos en su base de datos, solo admite fechas. Es por eso que el servidor tiene que realizar distintas modificaciones en la fecha para obtener el tiempo.

Una vez obtenidos todos los tiempos, guardados en un diccionario con clave el usuario y valor su tiempo más bajo, se realiza una ordenación del diccionario en función de los tiempos, se convierte en una tupla y se envía a la plantilla HTML renderizado.

5.3.3. Modelo de datos en Elasticsearch para los torneos

Para poder guardar los datos de los resultados de los enfrentamientos, se ha desarrollado un modelo específico de datos en *Elasticsearch*. Para el desarrollo de este modelo, *Django* da soporte a las bases de datos de *Elasticsearch* en el fichero *documents.py*, en el que se desarrollan los modelos como clases y los tipos de datos como atributos. Para el modelo de datos de estos torneos, el código es el siguiente:

```
class CompetitionRanking(Document):
    competition_id = Text()
    username = Text()
    time = Date()
    exercise = Text()
    date = Date()

class Index:
    name = 'kibotics_ranking_log'
    settings = {
        'number_of_shards': 1,
        'number_of_replicas': 0
    }
```

El almacenamiento de datos se lleva a cabo por el servidor, sin embargo, son los evaluadores los encargados de elaborar estos datos y enviarlos para que el servidor los almacene. Esto es así porque *Elasticsearch* sólo opera en el lado del servidor, lógicamente, no en el lado del cliente web.

Cuando una ejecución de un ejercicio dentro de un torneo termina, el evaluador recoge la información necesaria como el tiempo, los nombres de usuarios o la fecha, y envía los

datos mediante el *WebSocket* al servidor. Éste procesa los datos provenientes del mensaje y guarda el resultado en el modelo de datos creado. A continuación, se muestra un ejemplo para torneos de ranking:

```
CompetitionRanking(  
    competition_id=msg["comp_id"],  
    username=userWinner,  
    time=timeWinner,  
    exercise=exercise,  
    date=msg["date"]  
).save()
```

5.3.4. Maestro de ceremonias

La creación de rol o usuario especial ha sido necesaria para la realización de los torneos. De esta manera ha sido sencillo separar la ejecución de un ejercicio de un usuario normal y corriente, que ejecuta su propio código fuente, de la ejecución dentro del torneo. Así, para iniciar un torneo sólo es necesario que el usuario se apunte y el maestro de ceremonias ejecutará su código en el ejercicio del torneo que corresponda.

Ha sido necesario modificar las plantillas de los ejercicios, tanto en *Scratch* como en *Python*. En ellas, se ha limitado la cantidad de pestañas a la principal y a la de simulación (Figura 5.4). Cuando el maestro de ceremonias ejecuta los ejercicios no hay pestaña de edición ni pestaña dual.

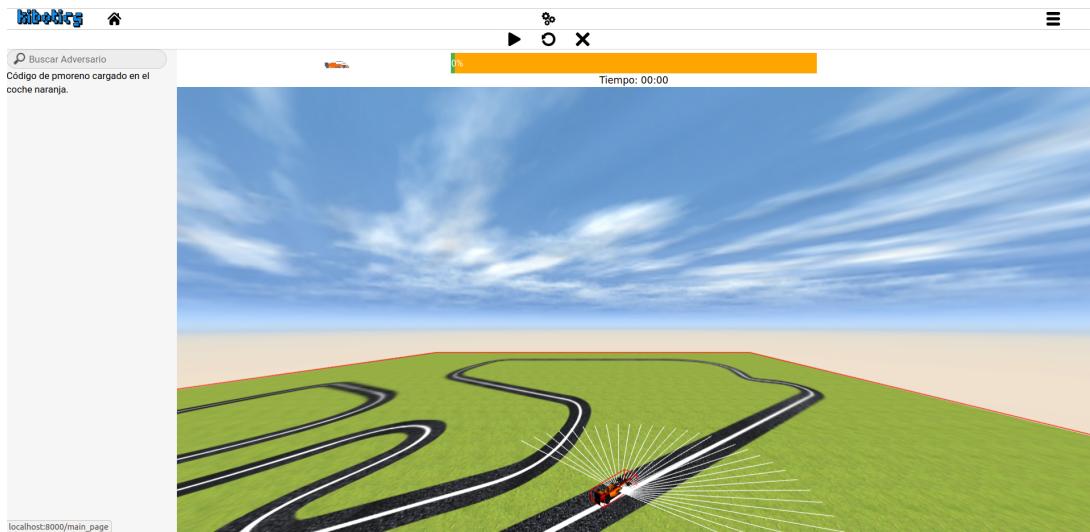


Figura 5.4: Ventana de ejecución con el maestro de ceremonia

Como puede verse en la figura 5.4, aparece un elemento de búsqueda por si el maestro de ceremonias tiene que buscar a algún usuario. Se ha desarrollado toda la infraestructura para que se envíe el nombre del usuario al servidor mediante *WebSocket* y el servidor responda con el código fuente del usuario elegido. Una vez cargado ese código fuente, aparecerá un mensaje y se podrá comenzar la ejecución.

Una vez terminada la ejecución, se ha implementado una nueva característica en el botón de reinicio por la que se libera el código fuente de los usuarios anteriores y se preparan los coches para recibir el código de los nuevos usuarios.

5.3.5. Automatización

Para facilitar la ejecución de los torneos, se ha desarrollado una automatización de todo su proceso. El torneo de ranking típicamente consta de un ejercicio que se ejecuta con el código fuente de un conjunto de usuarios, los participantes en la competición, que se van ejecutando en secuencia la misma prueba. Esta automatización se ha llevado a cabo en dos partes: automatización de los evaluadores y automatización de las ejecuciones.

Primero, en los evaluadores se ha desarrollado un *API* que permite hacer uso de ellos como una biblioteca *JavaScript* desde un entorno distinto a la ejecución de un ejercicio competitivo asíncrono. Esto se ha hecho para poder utilizar estas funciones desde el manejador de botones desarrollado en *Kibotics*.

Este manejador de botones es un código *JavaScript* donde se recoge toda la funcionalidad de los botones de control del simulador. De esta manera, al incorporarse el *API* de los evaluadores es posible comenzar, parar o reiniciar el evaluador de manera síncrona a la ejecución del simulador.

Gracias a esto, se ha desarrollado una función de comenzar, otra de parar y otra de reiniciar el evaluador.

Además, cuando un coche completa la vuelta, el evaluador es capaz de reconocerlo y enviar el usuario y su tiempo de manera automática hacia el servidor con el *WebSocket* y el servidor guarda estos datos en el modelo de torneo creado que corresponda.

Segundo, para que el maestro de ceremonias no tenga que estar buscando manualmente al siguiente usuario en cada nueva ejecución, se ha desarrollado un conjunto de ficheros JSON específicos para cada torneo en el que se detalla la secuencia de enfrentamientos o ejecuciones para que el maestro de ceremonias sólo deba comenzar y reiniciar la ejecución, el resto es completamente automático.

Además, y en línea con la automatización de los torneos, se ha programado un reinicio específico para la figura del maestro de ceremonias en el que se cargará, de manera automática, el código del siguiente participante al reiniciar la simulación y el evaluador.

En el caso de los torneos de ranking, el fichero JSON se compone de una lista de diccionarios con el nombre del usuario. (Figura 5.5). Esta ejecución está formada por los ficheros JSON donde se explicita la secuencia de enfrentamientos del torneo, los analizadores de esos ficheros y la infraestructura que permite lanzar automáticamente la ejecución de un código fuente y su evaluación.

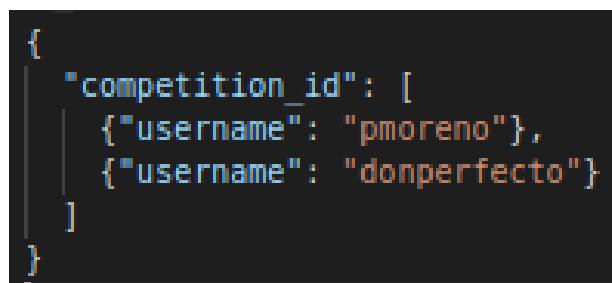


Figura 5.5: Ejemplo de fichero JSON para torneos de ranking

En este caso, sólo es necesario ir leyendo del fichero el siguiente usuario. Para ello se

ha desarrollado una función específica que realiza esta tarea:

```
var i = 0;
function getUserCode() {
    $.getJSON("/static/competitions_json/ranking.json", function(json) {
        if (json.competition_id[i]) {
            username1 = json.competition_id[i].username;

            sendUser(username1);
            i++;
        } else {
            sendUser(undefined);
        }
    });
}
```

Para tener una mejor idea del funcionamiento de este tipo de torneos, se adjunta un vídeo con una ejecución del torneo <https://youtu.be/1283-paXscg>.

5.4. Torneos de enfrentamientos

En este tipo de torneos se incluyen dos robots y un evaluador que determina el ganador del enfrentamiento y se puede apoyar en otros dos individuales. Se tiene en cuenta el usuario ganador del enfrentamiento aunque también se recoge el tiempo empleado.

En estos torneos se realizan enfrentamientos 1 contra 1 y el ganador se clasifica para la siguiente ronda. El participante que pierda será eliminado del torneo. De esta manera, el número de participantes se reduce a la mitad en cada ronda de manera progresiva (Figura 5.6).



Figura 5.6: Ejemplo torneo de enfrentamiento

5.4.1. Plantilla de la página web del torneo

Estos torneos se basan en enfrentamientos directos en los que el alumno que primero complete la vuelta será el que se clasifique a la siguiente hasta llegar a la final. Para lograr esto, son los evaluadores individuales los que se encargan de recoger el tiempo y el usuario y guardarlo en un modelo de *Elasticsearch*. Una vez se acceda a la página del torneo, el servidor renderizará los datos y se mostrarán en la plantilla HTML. En este caso también se ha añadido una ventana y un enlace para poder seguir el torneo en directo (Figura 5.7).



Figura 5.7: Plantilla de torneos de enfrentamiento

En la figura 5.7 se puede ver, en la parte superior izquierda, el resultado del enfrentamiento con dos rondas (semifinal y final). En color verde aparece el usuario vencedor del enfrentamiento junto con el tiempo empleado. Debajo, está el directo embebido de *Twitch* junto con el chat de directo.

5.4.2. Infraestructura del servidor

En los torneos de enfrentamientos, el renderizado de la página del torneo es algo más sencillo que en los torneo de ranking debido a que en los torneos de ranking no existen múltiples tiempos para el mismo usuario. De esta manera no es necesario estar haciendo comprobaciones de si el usuario existe y si tiene un tiempo mayor o menor.

En primer lugar, se realiza una petición a la base de datos de *Elasticsearch* con la función

```
Search(index="kibotics_classification\log").query({ "match": { "competition_id": "1" }}).  
filter()
```

que nos devuelve la *query* con todas las entradas almacenadas. Tras esto, se utiliza la función *scan()* para recorrer la *query* y se almacenan en un diccionario los enfrentamientos.

La clave del diccionario será el usuario que ha ganado y el valor está compuesto por una lista con los dos usuarios y el tiempo del ganador. Una vez obtenidos los datos de todos los usuarios, se renderiza y se envía a la plantilla HTML.

5.4.3. Modelo de Elasticsearch

En primer lugar ha sido necesaria la creación de dos modelos de datos en *Elasticsearch* para almacenar los distintos tipos de torneos. Para ello, *Django* ofrece un fichero en el que declarar estos tipos de datos llamado “*documents.py*”. Mediante la declaración de una clase con los distintos atributos y métodos, se crea el modelo en *Elasticsearch*:

```
class CompetitionClassification(Document):  
    competition_id = Text()  
    username1 = Text()  
    username2 = Text()  
    user_winner = Text()  
    time = Date()
```

```
exercise = Text()
date = Date()

class Index:
    # Name of the Elasticsearch index
    name = 'kibotics_classification_log'
    # See Elasticsearch Indices API reference for available settings
    settings = {
        'number_of_shards': 1,
        'number_of_replicas': 0
    }
```

El almacenamiento de datos se lleva a cabo por el servidor, sin embargo, son los evaluadores los encargados de recoger estos datos y enviarlos para que el servidor los almacene. Esto es así porque *Elasticsearch* sólo opera en el lado del servidor, lógicamente, no en el lado del cliente.

Cuando una ejecución de un enfrentamiento dentro de un torneo termina, los evaluadores recogen la información necesaria como el tiempo, los nombres de usuarios que se han enfrentado o el usuario vencedor, y envían los datos mediante el *WebSocket* al servidor. Éste procesa los datos provenientes del mensaje y guarda el resultado en el modelo de datos creado. A continuación, se muestra un ejemplo para torneos de enfrentamientos:

```
CompetitionClassification(
    competition_id=msg["comp_id"],
    username1=msg["user1"],
    username2=msg["user2"],
    user_winner=msg["winner"],
    time=timeWinner,
    exercise=msg["exercise"],
    date=msg["date"]
).save()
```

5.4.4. Maestro de ceremonias

Similar al maestro de ceremonias empleado para los torneos de ranking, este maestro de ceremonias es el encargado de ejecutar las distintas simulaciones que componen el

torneo. La peculiaridad aquí es que es necesario la búsqueda o lectura del código fuente de dos usuarios para cada ejecución (Figura 5.8).

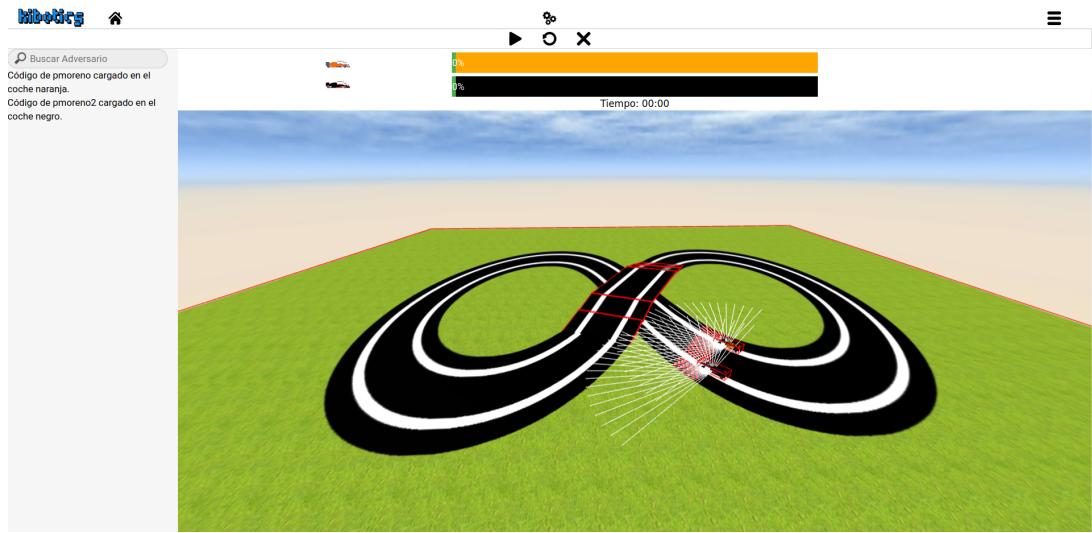


Figura 5.8: Ventana de ejecución con el maestro de ceremonia para enfrentamientos

En estos torneos también se cuenta con la característica del reinicio y lectura de la siguiente pareja de contrincantes.

5.4.5. Automatización

De manera análoga a los torneos de ranking, la funcionalidad de automatización incluida es similar. Sin embargo, para poder realizar la automatización de estos torneos, ha sido necesario desarrollar código extra para introducir los enfrentamientos de las sucesivas rondas.

En los torneos de ranking las ejecuciones necesarias se conocen de antemano, pues son una por cada usuario participante. Por el contrario, en los torneos de enfrentamientos no, sólo se saben los enfrentamientos de la primera ronda, pero los de las siguientes dependen de los vencedores de la ronda previa y, por lo tanto, se desconocen a priori. Se van generando dinámicamente sobre la marcha.

En el caso de los evaluadores, la distinción radica en el modelo desarrollado de la base de datos comentado en la sección 5.4.3.

El código necesario en los evaluadores comprueba si el último enfrentamiento de

la última ronda está vacío o ya hay un usuario en él. En el primer caso creará un enfrentamiento nuevo y en el segundo caso llenará el usuario que falta:

```
if (json.competition_id.round[json.competition_id.round.length-1]
    [json.competition_id.round[json.competition_id.round.length-1].length-1].username2 ==
        undefined) {
    json.competition_id.round[json.competition_id.round.length-1]
    [json.competition_id.round[json.competition_id.round.length-1].length-1].username2 =
        winner_user;
} else {
    json.competition_id.round[json.competition_id.round.length-1].
    push({"username1": winner_user});
}
```

5.4.5.1. Automatización

En el caso de los torneos de enfrentamientos, el fichero JSON se compone de un diccionario de rondas y, para cada ronda una lista de diccionarios con los distintos enfrentamientos (Figura 5.9).



```
{
  "competition_id": {
    "round": [
      [
        [
          {
            "username1": "pmoreno",
            "username2": "pmoreno2",
            "username1": "donperfecto",
            "username2": "pmoreno3"
          }
        ]
      ]
    ]
  }
}
```

Figura 5.9: Ejemplo de fichero JSON para torneos de enfrentamiento

En estos torneos es necesaria la modificación del objeto *JavaScript* con los enfrentamientos para añadir a los usuarios vencedores y poder seguir haciendo las siguientes rondas. Para ello se ha leído el fichero y se ha cargado dentro de una variable global en *JavaScript*. De esta manera, cuando el evaluador establece el usuario vencedor, realiza

CAPÍTULO 5. TORNEOS

una modificación sobre esta variable para ir añadiendo los distintos usuarios:

```
var json = (function() {
    var json = null;
    $.ajax({
        'async': false,
        'global': false,
        'url': "/static/competitions_json/classification.json",
        'dataType': "json",
        'success': function(data) {
            json = data;
        }
    });
    return json;
})();

var round = 0;
var match = 0;
function getUsersCode() {

    if (json.competition_id.round[round][match]) {
        username1 = json.competition_id.round[round][match].username1;
        username2 = json.competition_id.round[round][match].username2;

        console.log(username1, username2);
        sendUser(username1);
        match++;
    } else {
        round++;
        match = 0;
        if (json.competition_id.round[round][match]) {
            username1 = json.competition_id.round[round][match].username1;
            username2 = json.competition_id.round[round][match].username2;

            console.log(username1, username2);
            sendUser(username1);
            match++;
        } else {
            sendUser(undefined);
        }
    }
}
```

}

Con la automatización de los evaluadores y de la ejecución de enfrentamientos se ha automatizado casi todo el desarrollo de los torneos a excepción del comienzo y el reinicio por si algún código es erróneo y no es capaz de terminar una vuelta. En cuanto al reinicio y comienzo de una ejecución, cuando un usuario es capaz de completar la ejecución, se ha implementado la carga y ejecución de manera automática, pero se ha añadido la opción de que el maestro de ceremonias pueda parar e iniciar las ejecuciones manualmente por si los códigos no están bien desarrollados y son incapaces de terminar la ejecución.

Para tener una mejor idea del funcionamiento de estos torneos, se ha creado un vídeo demostrativo de una ejecución de estos torneos <https://youtu.be/qAZsUvwiUwg>.

5.5. Retransmisión en vivo

Por último, y fomentando el aspecto social en línea con lo desarrollado en este trabajo, se ha implementado una ventana de visualización en *Twitch* y un enlace a la página de directos de *Kibotics* en *Youtube Live* para que los usuarios puedan seguir el desarrollo del torneo desde la propia página web de torneo dentro de la plataforma *Kibotics*.

5.5.1. Retransmisión en Twitch

Ésta ha sido la retransmisión final elegida en este trabajo por ofrecer el API más sencillo en cuanto a la incorporación de su *streaming*. Para poder realizar esto, ha sido necesario registrarse en la plataforma *Twitch* y acceder a la clave privada de retransmisión. Además, es necesaria la descarga de un software de terceros dedicado a la retransmisión de contenido en directo llamado *OBS Studio* y su correspondiente configuración.

El API que ofrece esta plataforma para embeber vídeos es bastante sencillo, basta con crear un “div” HTML con un nombre específico, importar su biblioteca en *JavaScript* y realizar una llamada a este API con distintos parámetros, como usuario del *streaming* o dimensiones del vídeo. De esta manera es posible embeber el directo y el propio chat que puede ser utilizado incluso desde la página web del torneo (Figura 5.10).

```
<!-- Add a placeholder for the Twitch embed -->
<div id="twitch-embed"></div>
```

```
<!-- Load the Twitch embed script -->
<script src="https://embed.twitch.tv/embed/v1.js"></script>

<!-- Create a Twitch.Embed object that will render within
the "twitch-embed" root element. -->
<script type="text/javascript">
new Twitch.Embed("twitch-embed", {
    width: 854,
    height: 480,
    channel: "kibotics",
    // only needed if your site is also embedded on
    // embed.example.com and othersite.example.com
    parent: ["embed.example.com", "othersite.example.com"]
});
</script>
```

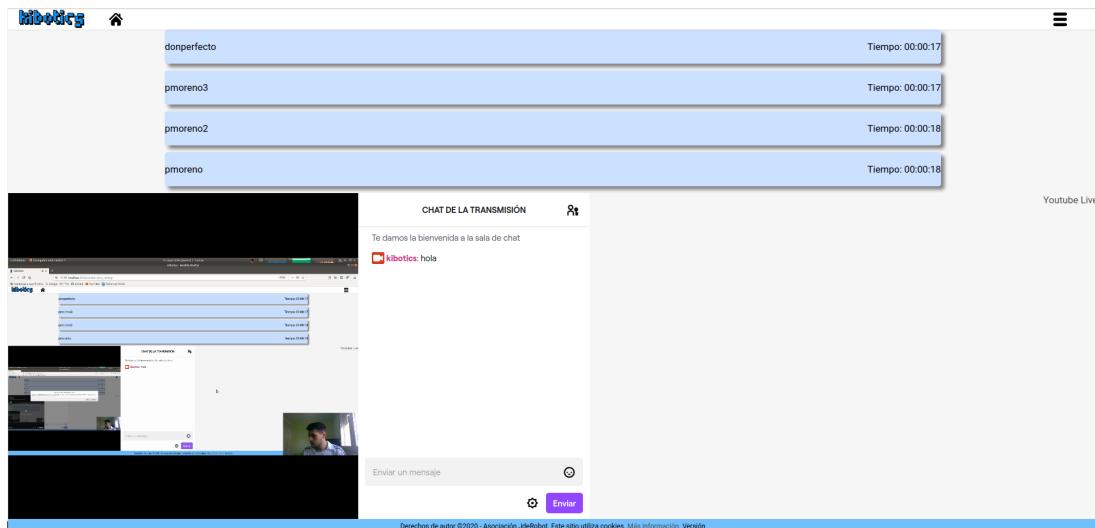


Figura 5.10: Página de torneos con Twitch activo

5.5.2. Retransmisión en Youtube Live

Esta plataforma ofrece la misma funcionalidad que *Twitch* pero con el añadido que, desde su *API*, es posible comenzar la creación de contenido en directo. Aunque es una *API* bastante potente, en este trabajo se ha elegido *Twitch* por su auge en cuanto a contenido infantil orientado a videojuegos. También, la política de derechos de autor de *Youtube* es

mucho más estricta y no ha permitido embeber su vídeo en una página web de terceros lo que provocaba un corte inmediato en la retransmisión. Además, *Youtube* cuenta con un problema añadido, trata cada retransmisión del mismo usuario de manera independiente lo que provoca que el enlace en cada retransmisión sea distinto.

El último problema se ha solucionado accediendo a la clave privada del usuario que realiza la retransmisión dado que es el propio usuario *Kibotics*, sin embargo, la única vía posible para poder visualizar los directos en *Youtube*, es la implantación de un enlace a la propia página de *Youtube* en el HTML de los torneos.

Otro inconveniente planteado es la imposibilidad de realizar un directo en ambas plataformas al mismo tiempo, por lo que finalmente se ha seleccionado *Twitch* como plataforma principal. Aun así, se han realizado todos los pasos para retransmitir en *Youtube Live*:

1. Crear una cuenta de *Google*.
2. Registrar la aplicación en *Google* para que pueda enviar peticiones al API.
3. Activar los campos *Youtube Data API v3* y *Youtube Content ID API*.
4. Pedir la activación de *streamings* en *Youtube*.

Los tres primeros pasos se han dado para permitir un desarrollo futuro de contenido automático utilizando la API de *Youtube*, aunque, como se ha descrito anteriormente, hay bastantes inconvenientes que solucionar primero. Para realizar una retransmisión, *Youtube* no necesita de *software* de terceros como *Twitch* con *OBS Studio* aunque es altamente recomendable para retransmitir elementos distintos de la cámara (Figura 5.11).

CAPÍTULO 5. TORNEOS

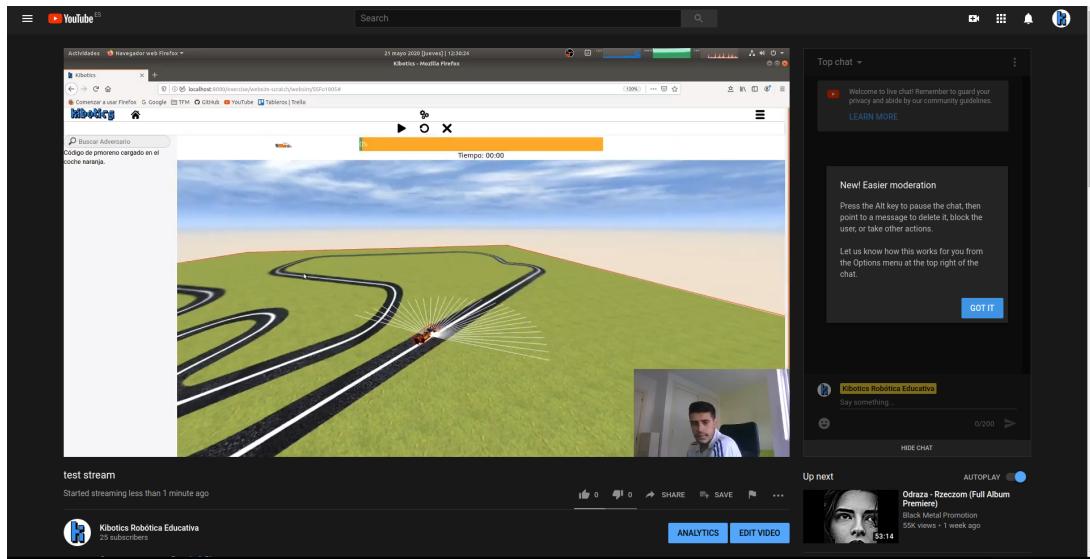


Figura 5.11: Página Youtube en directo

Para que se pueda apreciar mejor la retransmisión de los torneos, se adjuntan dos videos, el primero con la retransmisión desde la página web del torneo en la plataforma *Kibotics* y el segundo desde la plataforma *Twitch* para que se pueda apreciar cómo es el mismo contenido: <https://youtu.be/gWqsnwsCxVk> y <https://youtu.be/o2wgbRnS1Ak>.

Capítulo 6

Conclusiones

Una vez documentado el software diseñado y desarrollado a lo largo de este TFM y la funcionalidad que ofrece, se dedica este capítulo a la comprobación de los objetivos alcanzados, así como a la explicación de los conocimientos adquiridos y una breve exposición de posibles mejoras en la plataforma y de las líneas de actuación futuras.

6.1. Conclusiones

El objetivo principal de la refactorización de la plataforma y la incorporación de los *Juegos-Compartidos* y los *Torneos* ha sido alcanzado con éxito. Este objetivo se ha subdividido en dos objetivos secundarios.

El primer subobjetivo ha sido el diseño y el desarrollo de los *Juegos-Compartidos síncronos* y ha sido alcanzado satisfactoriamente. Gracias a la tecnología *WebRTC* ha sido posible desarrollar cuatro nuevas prácticas para la plataforma en las que los usuarios pueden competir uno contra uno simultáneamente. Para lograr este objetivo ha sido necesario una implementación nueva de las plantillas, la creación del chat, la redirección de la ventana de *A-Frame* y el soporte para el mismo en el lado del servidor con *Django-Channels*.

El segundo subobjetivo ha sido la creación de los *Torneos*. Este objetivo también ha sido alcanzado y para su consecución ha sido necesaria la tecnología *Elasticsearch* en el servidor para grabar la puntuación de los ejercicios en torneos de ranking o el ganador en los torneos de enfrentamientos, y para leerlos automáticamente desde una página web

asociada a cada torneo. Ambos se han integrado en la plataforma *Kibotics*. Y, además, la generación de una nueva figura llamada maestro de ceremonias. Éste ejecuta los códigos de los participantes de manera automática con un fichero *JSON*. Además se han realizado los pasos necesarios para incluir la retransmisión en directo de *Twitch* en un *IFrame* empotrado en la página de los torneos para que se puedan ver en tiempo real así como un enlace a *Youtube-Live*.

Para la consecución de todos estos objetivos, y de manera personal, se ha alcanzado el objetivo de una mejora de conocimiento en un campo tan heterogéneo como el desarrollo web. Gracias a ello, se han adquirido conocimientos en campos tan diversos como *JavaScript*, *HTML*, *CSS*, *JQuery*, *WebRTC*, *Django*, *Django-Channels*, *Twitch* y *Youtube API*, *Elasticsearch*, entre otros.

6.2. Trabajos futuros

Las contribuciones desde este TFM han ampliado la funcionalidad disponible en la plataforma Kibotics. Aun así hay muchas mejoras abordables para seguir mejorando la plataforma. Algunas de estas mejoras, relacionadas con este trabajo son:

- **Prácticas multirobot:** La inclusión de prácticas con más de dos robots es un punto muy interesante para poder hacer torneos masivos. Por ejemplo juegos compartidos síncronos como un partido de fútbol de 2 contra 2, al estilo de la *RoboCupJunior*.
- **Desarrollo de un vídeo-chat:** De esta manera los contrincantes pueden hablar entre ellos en lugar de utilizar el chat de texto únicamente. *WebRTC* ofrece funcionalidad para ello, aunque habrá que ver si el mayor consumo de ancho de banda debido al vídeo permite una ejecución fluida del juego compartido.
- **Nuevos torneos:** En los que los participantes jueguen directamente y no mediante el maestro de ceremonias. Por ejemplo, fomentando que grupos de amigos puedan crear sus propios torneos.
- **Nuevos Juegos-Compartidos síncronos:** La introducción de nuevos juegos, proporcionaría una mayor diversidad en el elenco de prácticas existentes en *Kibotics*.

- **Retransmisión automática en Yotube Live.** Uno de los aspectos más importantes en línea con este trabajo, es la creación de una retransmisión automática en *Yotube Live* utilizando el API que ofrece. Por ejemplo, incluyendo nuevas pruebas parecidas existentes en campeonatos de robótica educativa como *First Lego League*, *Vex* o *RoboCup Junior*.

Apéndice A

Refactorización en Kibotics

A la hora de incorporar los *Juegos-Compartidos síncronos* y los torneos automáticos y siguiendo la línea de una mejora en la experiencia social de los usuarios, se han realizado distintas mejoras en la plataforma *Kibotics*.

En primer lugar, se ha desarrollado un página personal para que el usuario pueda visualizar sus datos personales como usuario, nombre correo o fecha de finalización de suscripción a la plataforma. También cuenta con una foto del usuario (Figura A.1).

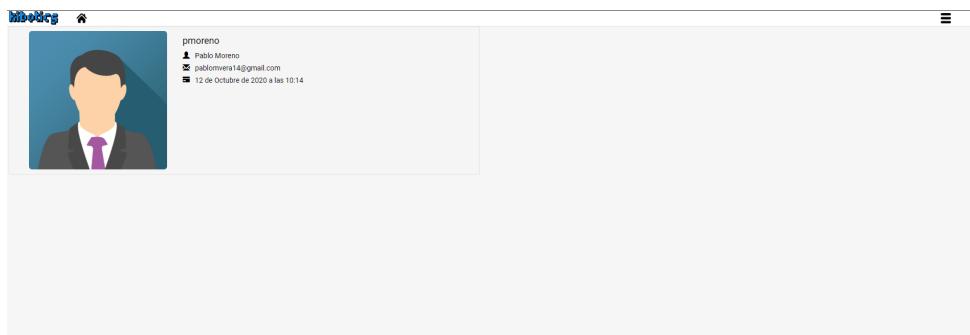


Figura A.1: Página personal de usuario

Otro cambio importante ha sido la incorporación de una nueva pestaña en cada ejercicio donde está empotrado el foro para que el usuario tenga la posibilidad de realizar preguntas de manera directa. Con esta mejora, la experiencia del usuario se ve incrementada al tener soporte dentro de los propios ejercicios (Figura A.2). Para ello, se ha desarrollado un “div” nuevo en la plantilla HTML y se ha utilizado un *I-Frame* para

CAPÍTULO A. ANEXO I

embeber el foro de *Kibotics*. Al tratarse de un foro en *Discourse*¹, es la manera inmediata y soportada por su API.

Para ello, ha sido necesario un estudio del campo *X-FRAME OPTION* de las cabeceras HTTP que se envían en las peticiones/respuestas de *Discourse*. Este campo está, por defecto, establecido a “SAMEORIGIN” para denegar las peticiones de incorporar páginas web dentro de otras para evitar la suplantación de identidad en páginas de terceros (*clickjacking*).

Se ha configurado *Discourse* para añadir la dirección de *Kibotics* a la excepción de esta cabecera para que permita su visualización. De esta manera se ha seguido manteniendo la seguridad frente a *clickjacking*² y poder mostrar el foro en *Kibotics* sin ningún problema.

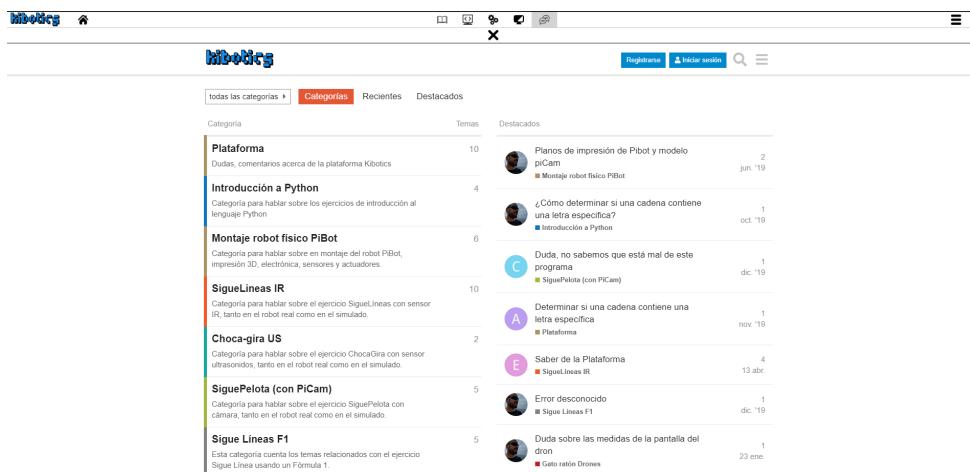


Figura A.2: Foro de Kibotics empotrado en el ejercicio

Otra mejora en el camino hacia los *Juegos-Compartidos síncronos* ha sido la refactorización de la visualización de las pestañas dentro de cada ejercicio en la plataforma. Para ello se han desarrollado distintos flujos de código en función de la pestaña que se desea visualizar junto con los botones propios de cada una de ellas.

Esto ha sido necesario para reajustar las ventanas de simulación y editor a cada modo o para establecer los botones necesarios y adaptarlo a los distintos ejercicios existentes en la plataforma (Figuras A.3 y A.4).

¹<https://www.discourse.org/>

²<https://es.wikipedia.org/wiki/Clickjacking>

CAPÍTULO A. ANEXO I

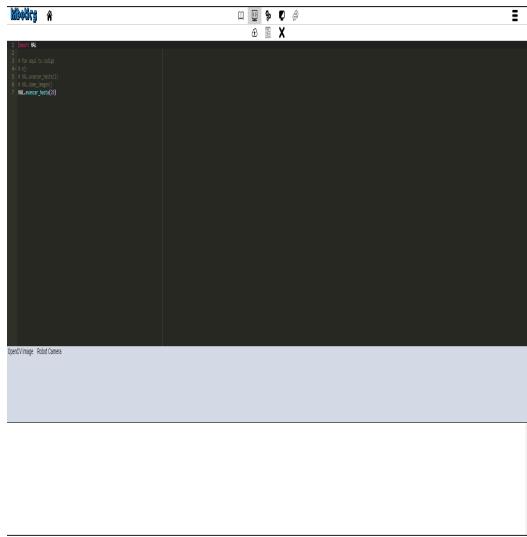


Figura A.3: Ventana de editor de código



Figura A.4: Ventana de simulación

Para todo lo relacionado con la visualización de las distintas ventanas de cada ejercicio se ha generado un fichero llamado “*display_windows.js*” que recoge toda esta funcionalidad para que sea transparente a la plantilla HTML. Así, en la plantilla sólo se contemplan las distintas ventanas y botones de los que se compone el ejercicio. Para ello, en la plantilla se han generado dos barras de botones distintas que están centradas en las barras de ejercicio y de ventana que contienen las ventanas y los botones de cada ventana respectivamente. Estás barras están formadas por una lista de elementos en HTML orientados horizontalmente en lugar de verticalmente. Todas estas preferencias visuales se han conseguido utilizando CSS y las clases predeterminadas que ofrece *Bootstrap*³.

³<https://getbootstrap.com/docs/4.4/getting-started/introduction/>

Bibliografía

- [1] Tecnología, programación y robótica en eso. <https://n9.cl/7lqc>.
- [2] Programación y robótica en ep. <http://www.telemadrid.es/noticias/madrid/madrilenos-impartiran-Programacion-Robotica-Primaria-0-2158584125--20190914103816.html>.
- [3] Scratch. <https://scratch.mit.edu/>.
- [4] Lego. <https://www.lego.com/es-es/categories/coding-for-kids>.
- [5] Kodu. <https://www.kodugamelab.com/>.
- [6] Snap. <https://snap.berkeley.edu/>.
- [7] Reinhard Budde Thorsten Leimbach Beate Jost, Markus Ketterl. Ieee international symposium on multimedia 'graphical programming environments for educational robots: Open roberta - yet another one?', 2014.
- [8] Django. documentación oficial de django. <https://www.djangoproject.com/>.
- [9] Django. documentación de wikipedia de django. [https://es.wikipedia.org/wiki/Django_\(framework\)](https://es.wikipedia.org/wiki/Django_(framework)).
- [10] Elasticsearch. documentación oficial de elasticsearch. <https://www.elastic.co/es/>.
- [11] Isaac z. schlueter. documentación oficial de npm. <https://www.npmjs.com/>.
- [12] Webpack. documentación oficial de webpack. <https://webpack.js.org/>.
- [13] Grunt. documentación oficial de grunt. <https://gruntjs.com/>.
- [14] Gulp. documentación oficial de gulp. <https://gulpjs.com/>.
- [15] Webrtc. documentación oficial de webrtc. <https://webrtc.org/>.
- [16] Obs. documentación oficial de obs. <https://obsproject.com/es>.

BIBLIOGRAFÍA

- [17] Álvaro paniagua tena. websim, simulador de robots con tecnologías web vr. (2018).
https://github.com/RoboticsLabURJC/2018-tfg-alvaro_paniagua.
- [18] Rubén Álvarez marín. mejoras en entorno de robótica educativa para niños. (2019).
https://github.com/RoboticsLabURJC/2018-tfg-alvaro_paniagua.