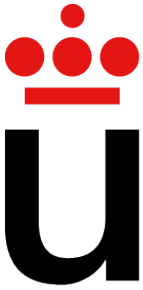

Universidad
Rey Juan Carlos

GRADO EN INGENIERÍA EN TELEMÁTICA

Curso Académico 2021/2022

Trabajo Fin de Grado

GAMIFICACIÓN DE PLATAFORMA UNIBOTICS

Autor : Daniel Hervás Rodao

Tutor : José María Cañas Plaza

Co-Tutor : David Roldán Álvarez

Trabajo Fin de Grado

Gamificación de la Plataforma Unibotics

Autor : Daniel Hervás Rodao

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

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

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 202X

Dedicado a

mi familia / mi abuelo / mi abuela

II

Agradecimientos

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

Resumen

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

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

Lo mejor es escribir el resumen al final.

Summary

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

Índice general

Índice de figuras

Capítulo 1

Introducción

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

En este capítulo introductorio se introducirá el contexto en el que se desarrolla el trabajo, así como los motivos que han motivado a llevarlo a cabo.

El campo de la robótica es muy amplio, en concreto, este TFG se encuentra en el marco de la robótica educativa, destinada a la enzeñanza de la misma.

1.1. Robótica

Los avances en computación de las últimas décadas han sido el impulso que ha permitido la creación de máquinas muy cercanas al ideal de autonomía que se ha perseguido siempre. La robótica está muy relacionada no solo con la rama de la ingeniería, si no, que involucra conocimientos de matemáticas y física para el desarrollos de máquinas autónomas. Uno de los objetivos principales de la robótica es el de facilitar tareas de la vida diaria al ser humano, incluso, en algunas ocasiones, sustituir al ser humano.

La inteligencia artificial, también está muy ligada al campo de la robótica. Los avances en este campo permiten desarrollar sistemas capaces de tener una cierta memoria útil para realizar una serie de funciones.

[1] `https://jderobot.github.io`

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

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

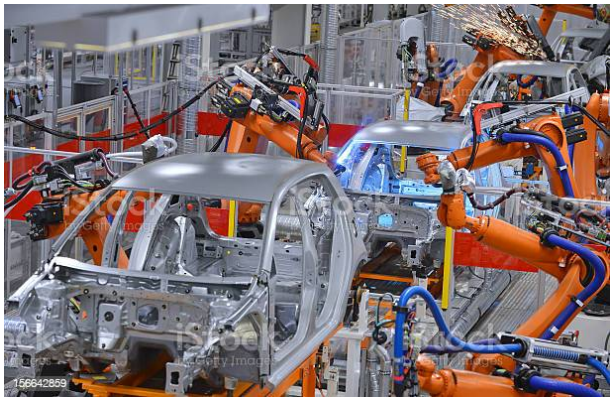

Figura 1.1: Brazo robótico.

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

Figura 1.2: Coche autónomo.

Otro campo de la aplicación de la robótica es la medicina, donde existen robots capaces de filtrar las vibraciones naturales del humano para proporcionarle una gran precisión y seguridad, un claro ejemplo es el robot DaVinci (Figura 1.3). Adicionalmente, hay robots capaces de man-

tener una estabilidad la estabilidad necesaria para caminar sobre dos piernas robóticas, como es el robot ATRIAS (Figura 1.4).

Figura 1.3: Robot DaVinci.

Figura 1.4: Robot ATRIAS.

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

Figura 1.5: Robot militar.

1.2. Componentes robóticas

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

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

lewares robóticos.

Durante el desarrollo de software robótico es preciso realizar una serie de pruebas para comprobar el funcionamiento del código y depurar errores, por lo que se necesitan simuladores que nos proporcionen un entorno cercano a la realidad previa al ensamblado del robot.

1.2.1. Middlewares robóticos

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

El *middleware* robótico más generalizado es ROS[2] (*Robotics Operating System*). *Robotics Operating System* fue desarrollado en 2007 por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte a sus proyectos. A pesar de no ser un sistema operativo, ROS proporciona tales servicios coko la abstracción *hardware*, mecanismos de comunicación entre prosesos, el control de dispositivos de bajo nivel y el mantenimiento de paquetes. *Robotics Operating System* fue desarrollado para sistemas UNIX, aunque en la actualidad está siendo adaptado para su funcionamiento en sistemas operativos como Fedora, Mac OS X, Arch, Gentoo, OpenSUSE, Slackware, Debian o Microsoft Windows.

1.2.2. Simuladores robóticos

El surgimiento de estos *softwares* robóticos está condicionado por la necesidad de realizar pruebas durante el desarrollo del *software* para la detección y depuración de posibles errores antes de llevarlo a un robot real debido al gran coste que suponen.

El simulador más generalizado en la actualidad es *Gazebo*[3]. Su popularidad se debe a su robusto motor de físicas, sus gráficos de alta calidad y su amplio catálogo de robots y escenarios. Es una herramienta de código abierto integrada con ROS, por lo que permite ejecutar *software* robótico en un escenario simulado (Figura 1.6).

[2] `https://www.ros.org/`

[3] `http://gazebosim.org/`

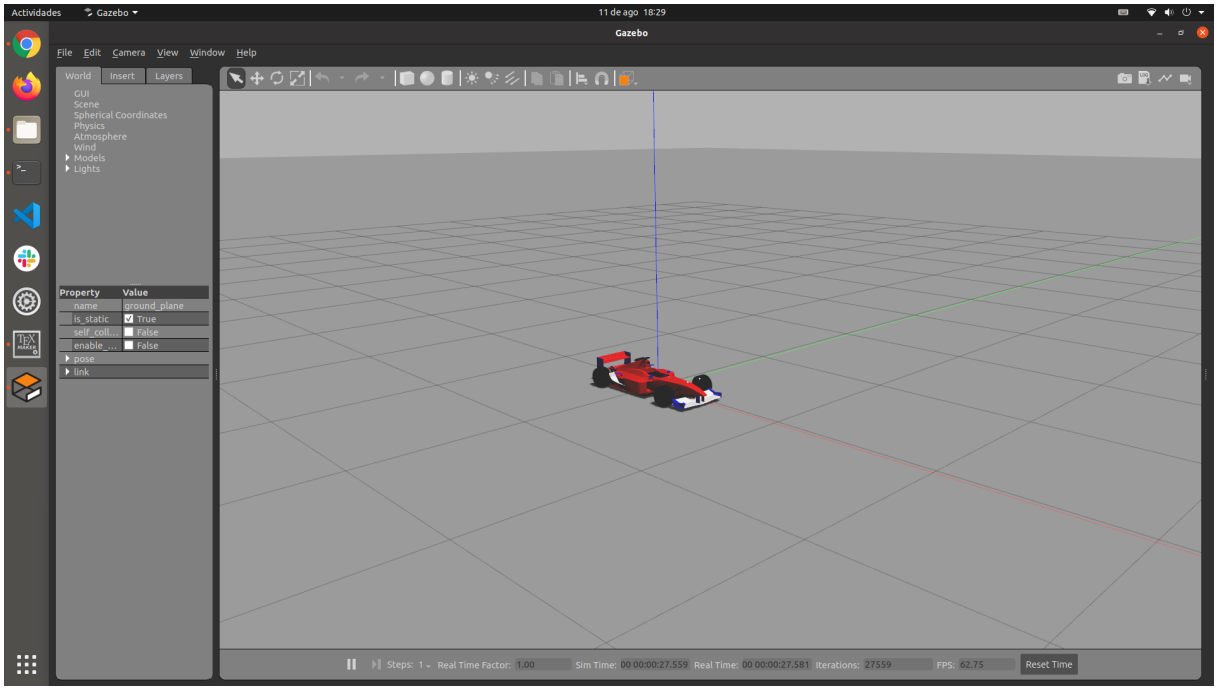

Figura 1.6: Simulador Gazebo.

1.3. Robótica educativa

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

Como se ha comentado, para llevar a cabo este método de educación es necesaria una infraestructura, como puede ser LEGO Mindstroms, un kit de robótica que es capaz de aumentras la capacidad de pensamiento de los alumnos.

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

Capítulo 2

Objetivos

2.1. Objetivo general

2.2. Objetivos específicos

2.3. Planificación temporal

2.4. Control de versiones

Capítulo 3

Herramientas

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

3.1. Lenguaje JavaScript

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

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

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

3.1.1. Caractrerísticas del lenguaje

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

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

3.1.2. Librería jQuery

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

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

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

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

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

3.2. Lenguaje HTML

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

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

Figura 3.1: Documento básico de *HTML*.

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

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

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

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

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

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

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

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

3.3. Hojas de estilo CSS

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

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

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

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

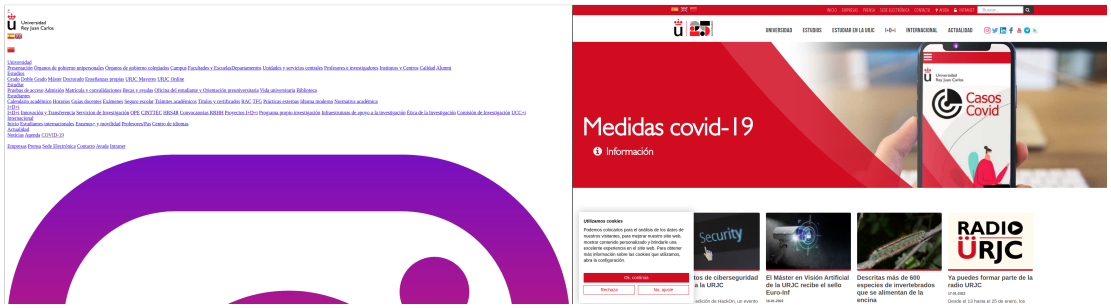

Figura 3.2: Comparativa entre no usar CSS y usarlo.

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

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

3.4. Django para servidores web

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

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

Algunas de las carácterística de Django son:

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

- Aplicaciones incorporables en cualquier página gestionada con Django.

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

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

3.5. WebRTC

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

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

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

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

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

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

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

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

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

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

3.5.1. Protocolos

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

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

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

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

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

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

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

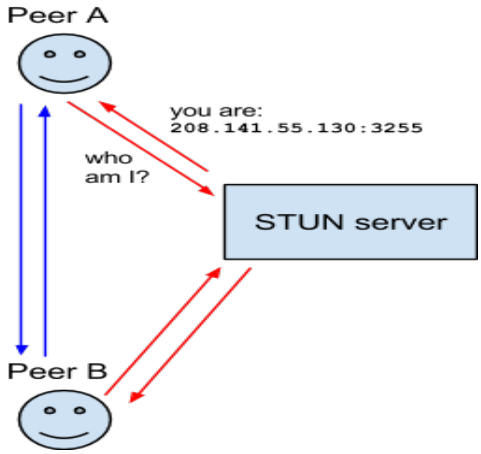

Figura 3.3: Protocolo STUN.

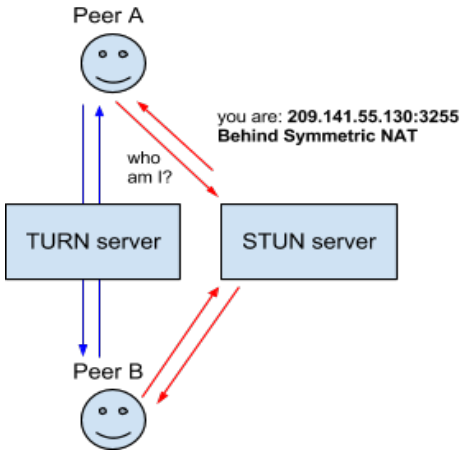

Figura 3.4: Protocolo TURN.

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

3.5.2. Establecimiento de la conexión

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

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

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

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

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

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

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

3.5.3. Candidatos ICE

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

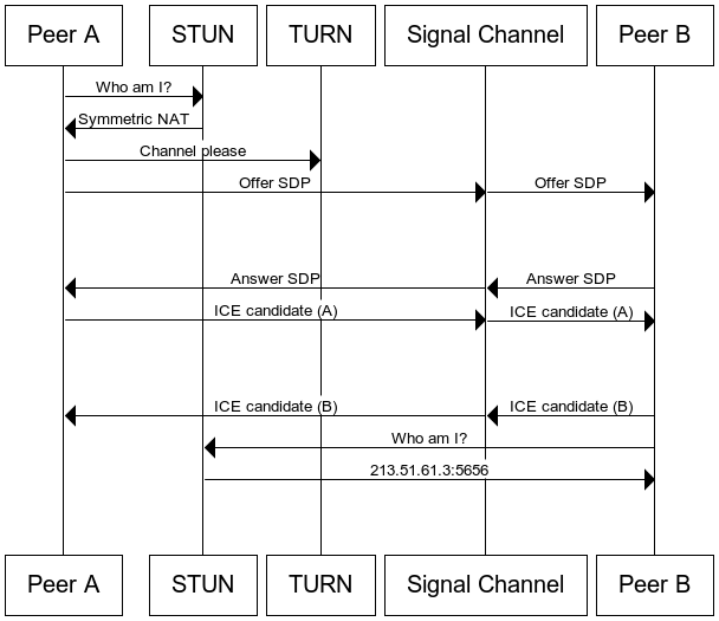

Figura 3.5: Diagrama de intercambio de candidados ICE.

Capítulo 4

Juegos Asíncronos

Aquí viene todo lo que has hecho tú (tecnológicamente). Puedes entrar hasta el detalle. Es la parte más importante de la memoria, porque describe lo que has hecho tú. Eso sí, normalmente aconsejo no poner código, sino diagramas.

4.1. Follow Line Game

Si tu proyecto es un software, siempre es bueno poner la arquitectura (que es cómo se estructura tu programa a "vista de pájaro").

Por ejemplo, puedes verlo en la figura **??**. LaTeX pone las figuras donde mejor cuadran. Y eso quiere decir que quizás no lo haga donde lo hemos puesto... Eso no es malo. A veces queda un poco raro, pero es la filosofía de LaTeX: tú al contenido, que yo me encargo de la maquetación.

Recuerda que toda figura que añadas a tu memoria debe ser explicada. Sí, aunque te parezca evidente lo que se ve en la figura **??**, la figura en sí solamente es un apoyo a tu texto. Así que explica lo que se ve en la figura, haciendo referencia a la misma tal y como ves aquí. Por ejemplo: En la figura **??** se puede ver que la estructura del *parser* básico, que consta de seis componentes diferentes: los datos se obtienen de la red, y según el tipo de dato, se pasará a un *parser* específico y bla, bla, bla...

Si utilizas una base de datos, no te olvides de incluir también un diagrama de entidad-relación.

4.2. Drone Cat Mouse Game

Capítulo 5

Juegos Síncronos

5.1. Synchronous Follow Line Game

Capítulo 6

Experimentos y validación

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

Capítulo 7

Resultados

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

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

Capítulo 8

Conclusiones

8.1. Consecución de objetivos

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

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

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

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

8.2. Aplicación de lo aprendido

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

1. a

2. b

8.3. Lecciones aprendidas

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

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

8.4. Trabajos futuros

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

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

Apéndice A

Manual de usuario

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

Bibliografía

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

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

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

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

[5]

[6]