



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

ARTank: Juego de realidad aumentada combinado con Arduino y MYO, un nuevo paradigma de interacción

Autor

Carlos Manuel Sequí Sánchez(alumno)

Directores

Marcelino Cabrera(tutor)



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, septiembre de 2018



ugr | Universidad
de **Granada**

ARTank: Juego de realidad aumentada combinado con Arduino y MYO, un nuevo paradigma de interacción

Autor

Carlos Manuel Sequí Sánchez

Directores

Marcelino Cabrera

Departamento de
Lenguajes y Sistemas
Informáticos



DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS

ARTank: Juego de realidad aumentada combinado con Arduino y MYO, un nuevo paradigma de interacción

Carlos Manuel Sequí Sánchez(alumno)

Palabras clave: Realidad Aumentada, MYO Armband, Arduino UNO, juego

Resumen

Con el fin de trabajar con temas de mi agrado, crear algo novedoso y aprovechar los conocimientos adquiridos a lo largo del grado, decidí realizar un juego para teléfono móvil mezclando realidad aumentada y el uso de un nuevo paradigma de interacción como es MYO Armband, el cual se encargará de controlar al personaje principal del juego: un tanque diseñado e impreso en 3D por mí mismo que viaja a las órdenes de un arduino UNO.

ARTank: Augmented Reality game combined with Arduino and MYO, a new interaction paradigm

Carlos Manuel Sequí Sánchez(student)

Keywords: Augmented Reality, MYO Armband, Arduino UNO, game

Abstract

In order to work with topics that I like, create something new and take advantage of the knowledge acquired throughout the degree, I decided to make a mobile phone game mixing augmented reality and the use of a new paradigm of interaction such as MYO Armband, which is in charge of controlling the main character of the game: a tank designed and printed in 3D by myself that moves on the orders of an arduino UNO.

Yo, **Carlos Manuel Sequí Sánchez**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**, con DNI 20486926K, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Carlos Manuel Sequí Sánchez

Granada a 6 de septiembre de 2018.

D. **Marcelino Cabrera(tutor1)**, Profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *ARTank: Juego de realidad aumentada combinado con Arduino y MYO, un nuevo paradigma de interacción*, ha sido realizado bajo su supervisión por **Carlos Manuel Sequí Sánchez**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 1 de Septiembre de 2018 .

El director:

Marcelino Cabrera

Agradecimientos

Llegado a este punto, agradezco la paciencia e interés puesto en mi aprendizaje a todos los profesores que han formado parte, en estos cuatro años de carrera, de poner en mis manos la semilla de conocimiento que me servirá para lanzarme al mundo profesional, así como a mi tutor Marcelino Cabrera, quien se ha encargado de ayudarme y supervisar este TFG. Agradezco a toda mi familia y, con mayor énfasis a mis padres y a mi hermano, el interés y el apoyo que me han ofrecido desde el momento en el que entré a la ingeniería, aunque no entiendan del todo las "letras raras" en la pantalla de mi ordenador cuando trabajo, o que no nos enseñen a "hackear" cosas. Por último, agradezco el haber dado con buenos amigos en la misma carrera, con buena compañía todo ha sido más sencillo, ya sabéis.

Índice general

| | |
|---------------------------------------------------------------|-----------|
| 1. Capítulo 1. Introducción | 1 |
| 1.1. Inspiración y Motivación | 1 |
| 1.2. Metodología de trabajo | 2 |
| 1.2.1. SCRUM | 4 |
| 1.2.2. Sprints realizados | 5 |
| 2. Capítulo 2. Estado del arte | 9 |
| 2.1. La realidad aumentada | 9 |
| 2.1.1. Diferencia con la realidad virtual | 9 |
| 2.1.2. Inclusión en la actualidad | 10 |
| 2.2. Interacción hombre-máquina (HCI) | 13 |
| 2.2.1. Historia de la HCI | 13 |
| 2.2.2. Tipos de HCI | 15 |
| 2.3. Impresión 3D | 16 |
| 2.3.1. Aplicaciones en la actualidad | 17 |
| 2.4. Existencia de proyectos similares | 18 |
| 2.5. Crítica al estado del arte y aportes propios | 20 |
| 3. Capítulo 3. Análisis del Hardware | 23 |
| 3.1. Microcontrolador del tanque | 23 |
| 3.1.1. ¿Por qué Arduino? | 24 |
| 3.1.2. Tipos de placas Arduino | 25 |
| 3.2. Módulos complementarios utilizados con Arduino | 27 |
| 3.2.1. Movilidad del tanque. Motores servo | 28 |
| 3.2.2. Sensor de proximidad HC-SR04 | 29 |
| 3.2.3. Módulo WiFi | 31 |
| 3.3. El tanque | 33 |
| 3.3.1. Tipos de plástico. Mi elección | 33 |
| 3.4. MYO Gesture Control Armband | 34 |
| 4. Capítulo 4. Análisis del Software | 37 |
| 4.1. Diseño de modelos 3D. Blender | 37 |
| 4.2. Impresión de piezas 3D | 38 |

| | |
|---------------------------------------------------------------|-----------|
| 4.3. Programación de Arduino | 38 |
| 4.4. Creación del servidor. Netbeans | 39 |
| 4.5. Desarrollo del videojuego. Unity | 40 |
| 5. Capítulo 5. Análisis de requisitos software | 43 |
| 5.1. Descripción del software | 43 |
| 5.2. Requisitos del sistema (juego) | 44 |
| 5.2.1. Requisitos funcionales | 44 |
| 5.2.2. Requisitos no funcionales | 46 |
| 5.2.3. Requisitos de información | 47 |
| 5.3. Requisitos del sistema (Servidor) | 47 |
| 5.3.1. Requisitos funcionales | 47 |
| 5.3.2. Requisitos no funcionales | 47 |
| 5.3.3. Requisitos de información | 47 |
| 5.4. Requisitos del sistema (Arduino) | 48 |
| 5.4.1. Requisitos funcionales | 48 |
| 5.4.2. Requisitos no funcionales | 48 |
| 5.4.3. Requisitos de información | 48 |
| 5.5. Casos de uso | 49 |
| 5.6. Diagramas de clases | 58 |
| 6. Capítulo 6. Implementación | 65 |
| 6.1. Desarrollo del juego | 65 |
| 6.1.1. Escena 1: Menú principal | 65 |
| 6.1.2. Escena 2: El juego | 66 |
| 6.1.3. Escena 3: Game Over | 69 |
| 6.1.4. Escena 4: Finalización exitosa de la partida | 70 |
| 6.2. Implementación del servidor | 70 |
| 6.2.1. TCP/IP | 70 |
| 6.2.2. Servidor | 71 |
| 6.2.3. GestorHebras | 72 |
| 6.2.4. Procesador | 72 |
| 6.3. Implementación del código para Arduino | 73 |
| 6.4. Conexiones hardware Arduino | 74 |
| 6.4.1. Módulo ESP8266 | 75 |
| 6.4.2. Detector de proximidad | 75 |
| 6.4.3. Motores | 76 |
| 6.5. Realización del tanque | 76 |
| 7. Capítulo 7. Conclusiones finales | 79 |
| 7.1. Asignaturas de utilidad | 79 |
| 7.2. Conocimientos aprendidos | 80 |
| 7.3. Valoración final | 81 |

| | |
|----------------------------------------------------------|-----------|
| 8. Capítulo 8. Trabajos futuros | 83 |
| 8.1. Gafas de realidad aumentada | 83 |
| 8.2. Añadir modos de juego | 84 |
| 8.3. Añadir uso de varios armas | 84 |
| 8.4. Animaciones, dinamismo | 85 |
| 8.5. El servidor | 85 |
| 8.6. Niveles de dificultad | 85 |
| 9. Anexo | 91 |
| 9.1. Eje de las ruedas traseras | 92 |
| 9.2. Soporte de los motores servo | 94 |
| 9.3. Caja para guardar la batería portátil | 96 |
| 9.4. Estructura para proteger la placa Arduino | 98 |

Índice de figuras

| | | |
|------|------------------------------------------------------------------------------------------|----|
| 2.1. | Capturas de pantalla de la app Measurekit[9] | 10 |
| 2.2. | Captura de pantalla de un vídeo de YouTube sobre el juego The Machines.[10] | 11 |
| 2.3. | Captura de pantalla de un game play de Pokémon Go[11] . . | 12 |
| 2.4. | Presentación E3 Minecraft + Hololens[12] | 13 |
| 2.5. | Interacción hombre-máquina[13] | 14 |
| 2.6. | Ejemplo de uso de la acetona en modelos 3D[14] | 17 |
| 2.7. | Casa impresa en 3D[15] | 18 |
| 2.8. | Demostración de uso del teclado de Realidad Aumentada jun- to con MYO[16] | 19 |
| 2.9. | Juego Realidad Aumentada + MYO[17] | 19 |
| 3.1. | Placa Arduino [18] | 24 |
| 3.2. | Servomotor [19] | 28 |
| 3.3. | Sensor HC-SR04 [20] | 29 |
| 3.4. | Calculamos el tiempo por centímetro [21] | 30 |
| 3.5. | Calculamos la distancia [21] | 30 |
| 3.6. | Patillas ESP8266 [22] | 31 |
| 3.7. | Foto de mi Prusa i3 Psique Steel | 33 |
| 3.8. | Esquema MYO Armband[23] | 35 |
| 3.9. | Ejemplo de uso de MYO[24] | 36 |
| 4.1. | Interfaz y ejemplo de Blender[25] | 37 |
| 4.2. | Partes básicas del Arduino IDE[26] | 39 |
| 4.3. | Juego Hearthstone, hecho con Unity[27] | 40 |
| 5.1. | Diagrama CU_01: Iniciar juego | 49 |
| 5.2. | Diagrama CU_02: Detección de patrón | 50 |
| 5.3. | Diagrama CU_03: Resolución de colisiones | 51 |
| 5.4. | Diagrama CU_04: Infingir daño a enemigo | 52 |
| 5.5. | Diagrama CU_05: Ofrecer conexión | 53 |
| 5.6. | Diagrama CU_06: Enviar/recibir información | 54 |
| 5.7. | Diagrama CU_07: Realimentación del juego | 55 |
| 5.8. | Diagrama CU_08: Información de proximidad | 56 |

| | | |
|-------|---------------------------------------------------------------------------|----|
| 5.9. | Diagrama CU_09: Movimiento del tanque | 57 |
| 5.10. | Diagrama CU_10: Red WiFi | 58 |
| 5.11. | Diagrama de clases Vuforia | 59 |
| 5.12. | Diagrama de clases escena menú principal del juego | 60 |
| 5.13. | Diagrama de clases escena de juego | 61 |
| 5.14. | Diagrama de clases escena de partida finalizada exitosamente | 62 |
| 5.15. | Diagrama de clases escena de Game Over | 62 |
| 5.16. | Diagrama de clases del servidor | 63 |
| 5.17. | Diagrama de clases del gestor del tanque (programación Arduino) | 64 |
| 6.1. | Three way handshake[28] | 71 |
| 6.2. | Conexión módulo WiFi con Arduino[29] | 75 |
| 6.3. | Conexión detector de proximidad con Arduino[30] | 75 |
| 6.4. | Conexión motores servo con Arduino[31] | 76 |
| 6.5. | Soporte para batería en fase de modelado | 77 |
| 6.6. | Soporte para batería en fase de prueba | 77 |
| 8.1. | Prueba de detección de gestos de las Hololens[32] | 83 |
| 9.1. | Vista general | 92 |
| 9.2. | Vista en planta | 93 |
| 9.3. | Vista frontal | 94 |
| 9.4. | Vista general | 94 |
| 9.5. | Vista en planta | 95 |
| 9.6. | Vista general | 96 |
| 9.7. | Vista lateral | 96 |
| 9.8. | Vista desde abajo | 97 |
| 9.9. | Vista general | 98 |
| 9.10. | Vista en planta | 99 |

Capítulo 1

Introducción

El principal objetivo del proyecto es la creación de un juego para teléfonos móviles con sistema operativo Android, haciendo uso de realidad aumentada para mezclar el entorno físico con objetos y enemigos virtuales a través de la cámara del teléfono móvil, con el fin de ofrecer una experiencia divertida al usuario.

El jugador ha de hacer uso, además, de un sistema de interacción novedoso como es MYO Armband, utilizado en este caso para controlar mediante gestos el tanque, el cual está dirigido por un Arduino UNO que se encarga de comunicarse con el propio brazalete para recibir las órdenes que el usuario manda con el fin de controlar al tanque y con la aplicación de escritorio para enviar información de la distancia de este a un obstáculo mediante un sensor de proximidad.

La misión principal del tanque culmina al llegar a la meta habiendo derrotado con distintos tipos de armas recogidas a los enemigos que por el camino se haya encontrado, evitando aproximarse a los obstáculos que delimitan el camino hacia la meta, ya que estos le harán perder puntos de vida.

He de recalcar que las referencias existentes en la bibliografía de este documento han sido comprobadas una a una hoy día 7 de septiembre de 2018 con el objetivo de cerciorarme de que los enlaces de páginas web a las que hacen referencia son válidos.

1.1. Inspiración y Motivación

El motivo de mayor relevancia que me impulsó a realizar este proyecto fue mi deseo de hacer un **videojuego novedoso**, ya que habiendo cursado asignaturas como Fundamentos de Redes, Computación Ubicua, Programación Gráfica de Videojuegos y Nuevos Paradigmas de Interacción, tenía en mis manos las herramientas básicas para poder desempeñar un proyecto de

este estilo.

Estas asignaturas no solo me proporcionaron las herramientas para embarcarme en esta idea, sino que me hicieron ver, además, que los juegos "futuristas" en los que realidad y ficción se mezclan para interaccionar entre sí mediante **nuevos paradigmas de control** (gestos), están cada vez más próximos y resultan cada vez más sencillos de implementar para desarrolladores principiantes en el ámbito como yo y, por ello, no tuve miedo a la hora de lanzarme al desarrollo del proyecto.

Actualmente existe una cantidad inmensa de juegos para teléfono móvil, la mayor parte de ellos utilizan interacción táctil persona-teléfono y, es aquí donde este juego marca la diferencia con el resto. Haciendo uso de la interacción gestual **introducimos al usuario de teléfono en un mundo distinto** al que está acostumbrado con los juegos convencionales.

Evidentemente existe un problema para el jugador "casual" (aquel que juega de vez en cuando a algún juego), y es que el precio del dispositivo de interacción gestual no es precisamente barato, por lo que seguramente no esté dispuesto a gastar dicha cantidad por jugar a un solo juego, por ello, pienso que va dirigido a aquellos jugadores que desean ir más allá y explorar nuevos horizontes y posibilidades, abriendo paso cada vez más y más al futuro de los videojuegos.

Además de ser un simple juego, **con vistas de futuro** (una vez desarrollado de forma completa) podría estudiarse la forma de **introducirlo en el mundo sanitario**, aprovechando la necesidad de interacción gestual para la rehabilitación de pacientes que hayan podido sufrir un accidente en las extremidades superiores, convirtiéndolo así, en un **serious game**. Lo ya mencionado y el hecho de ser algo novedoso que no he conseguido encontrar desarrollado al completo en ningún lugar de internet, enriquecieron mis ganas de apostar por este proyecto.

1.2. Metodología de trabajo

Tras introducir las funcionalidades básicas del proyecto, me dispongo a describir el método de trabajo utilizado para la realización del mismo. Para la elección de un método de trabajo, antes de iniciar el mismo proyecto, es necesario conocer varios aspectos que influirán directamente sobre el proceso de creación:

- **Tiempo disponible:** tiempo restante hasta la entrega del proyecto.
- **Requisitos funcionales:** objetivos principales a cumplir con el proyecto.
- **Recursos disponibles:** disponibilidad de recursos que te ayuden a

resolver el problema en cuestión (artículos de ayuda, software de desarrollo, documentos de asignaturas cursadas, etc.).

A continuación dispongo los requisitos a cumplir por la metodología escogida antes de comenzar el proyecto:

- Disponemos de **poco tiempo** para la realización del proyecto, por lo que nuestro trabajo se convierte en una **entrega temprana** al "cliente".
- Cabe la existencia de **requisitos cambiantes**, puesto que tanto con las reuniones con el tutor, como por propia decisión, la finalidad del proyecto o ciertos aspectos de este pueden ser modificados para mejorar su funcionalidad.
- Entrega al tutor en cortos periodos de tiempo de software totalmente funcional, convirtiéndose así en un **método iterativo e incremental**, ya que en cada iteración (cada entrega al tutor) se produce un incremento del proyecto a desarrollar, cumpliendo cada vez con más y más requisitos funcionales.
- **Comunicación con el tutor** de forma continua, al menos tras la realización de cada incremento del proyecto con el fin de obtener re-alimentación sobre el trabajo realizado.

Conociendo ya las necesidades a cubrir, sabemos sin lugar a dudas que hemos de hacer uso de una metodología de desarrollo ágil por las características mencionadas. Concretamente me ha interesado decantarme por SCRUM en lugar de otras metodologías como Adaptative Software Development (ASD) o Programación Extrema (XP), ya que es la que más se adapta a los principios de comunicación con el tutor en favor a la realimentación, requisitos cambiantes y desarrollo iterativo e incremental.

Por un lado, **Adaptative Software Development** se basa en la constante adaptación a los constantes cambios y errores que surgen durante el desarrollo. Sus ciclos no se basan en planificación-diseño-construcción de software como en SCRUM. Es un método iterativo en el que, las revisiones tras cada iteración se realizan con el fin de aprender de los errores surgidos para adaptarse a ellos, en SCRUM reiteramos hasta solucionarlo.

Por otro lado tenemos la **Programación Extrema**, metodología en la que se hace énfasis en la adaptación antes que en la previsión. De esta forma se considera normal el cambio de requisitos sobre la marcha, al contrario que con SCRUM, en la que se definen de forma inicial los requisitos del proyecto y, de forma constante, se busca el camino hacia su implementación intentando no cambiarlos para adaptarse a ello, sino cambiarlos para mejorar la

funcionalidad.

1.2.1. SCRUM

Es un modelo de trabajo desarrollado por Ikujiro Nonaka e Hirotaka Takeuchi a principios de los 80 con el fin de analizar la forma de crear nuevos productos por parte de las empresas tecnológicas más grandes [33]. En uno de sus estudios compararon esta novedosa forma de trabajo con el avance en equipo de la melé (scrum en inglés) formada por los jugadores de un equipo de rugby.

Como ya hemos dicho, es una metodología de desarrollo ágil, cuyo transcurso o iteraciones quedan divididas en sprints, al final de cada cual el producto ha de ser un avance de funcionalidades completamente entregable y con la dirección fijada hacia los requisitos propuestos al inicio del proyecto.[38]

Características básicas:

- **Sprints** o incrementos, los cuales se definen por una serie de fases que comentaremos a continuación.
- **Comunicación** entre los equipos de trabajo con el fin de favorecer realimentación y mejorar la idea a desarrollar. En mi caso, las reuniones han sido desarrolladas con mi tutor. En ellas se realizan tanto las planificaciones de sprint (al inicio de cada uno de estos), como las revisiones de sprint (para la comprobación y mejora del trabajo realizado)

Fases de un sprint

Cada sprint a realizar trata de cumplir con una de las funcionalidades del proyecto y, para ello, ha de seguirse un proceso definido que nos ayudará a zanjar dicha funcionalidad de forma robusta. A continuación describimos dichas fases:

- **Planificación:** Reunión con el tutor para la preparación del sprint en la que se define exactamente cual es el propósito a desarrollar durante la ejecución del sprint.
- **Desarrollo:** Ejecución de lo planificado con el tutor. Aquí es donde se desarrolla trabajo con el fin de alcanzar la meta definida durante la planificación. Durante esta fase puede haber más reuniones con el fin de obtener realimentación del trabajo que se está desarrollando. No es posible cambiar los objetivos definidos durante la fase de planificación, por ello esta ha de ser robusta y bien premeditada.

- **Revisión y mejoras:** Una vez terminado el proceso de ejecución, se realiza una última reunión de sprint para revisar el trabajo realizado, de esta forma nos cercioramos de que todo ha salido según lo planeado y, en caso contrario o, en caso de desear cambiar algún objetivo de la planificación, se volverá a ejecutar el sprint desde el inicio (planificación, ejecución y revisión). En caso de que todo salga bien se procederá a la ejecución del siguiente sprint, el cual definirá la realización de una nueva funcionalidad del proyecto habiéndonos asegurado de cumplimentar la funcionalidad anterior de manera robusta.

En **conclusión:** para el proyecto desarrollado, me ha resultado de completa efectividad la elección de la metodología de desarrollo ágil escogida, principalmente por la escasez de tiempo para su desarrollo y por la la estructura de trabajo que presenta, ya que favorece la robustez de la realización de cada una de las funcionalidades que en un principio planteamos mi tutor y yo para el proyecto, ofreciéndome la oportunidad (en tanto en cuanto tiempo disponía) de ampliar funcionalidades una vez cubiertas las básicas.

1.2.2. Sprints realizados

A continuación me dispongo a explicar cada uno de los sprints realizados durante la ejecución del proyecto junto con el tiempo aproximado de realización del mismo:

1. Diseñar y crear el tanque mediante impresión 3D. 2 semanas

Planificación: Primeramente, con el fin de conseguir al personaje principal del juego (el tanque) realizamos las siguientes tareas.

Desarrollo:

- Implementación del modelado 3D de piezas con el software de modelado 3D Blender.
- Elección de material para la impresión 3D (PLA).
- Impresión de las piezas modeladas.
- Ensamblado y mecanizado de las piezas.

2. Controlar el tanque con Arduino UNO. 2 semanas

Planificación: Tras haber construido el tanque, el siguiente paso consiste en la programación de los accesorios y herramientas necesarias para el control del tanque mediante Arduino.

Desarrollo:

- Conexión y programación del movimiento de los motores.
- Conexión y programación del sensor de proximidad.

Una vez programados los accesorios básicos para el control del tanque y, comprobado su funcionamiento, el último paso para la finalización de este sprint consistió en adherir al tanque el microcontrolador Arduino junto con los motores y el sensor de proximidad.

3. Crear el servidor y conectarlo con Arduino. 2 semanas

Planificación: Una vez obtenido un tanque funcional en cuanto a movimiento y detección de obstáculos, el siguiente paso es hacer posible que se mueva sin tener que estar conectado al ordenador.

Desarrollo: Para ello me decidí por crear una comunicación cliente-servidor mediante un módulo WiFi conectado al Arduino del tanque. Estos son los pasos a seguir:

- Conexión y programación del módulo WiFi al Arduino (hace las veces de cliente).
- Creación del servidor en el ordenador con el fin de gestionar la conexión.

Una vez realizado esto, ya era capaz de controlar el tanque a distancia mediante el teclado del ordenador.

4. Crear interacción de MYO con Arduino UNO a través del servidor. 2 semanas

Planificación: Teniendo resuelta la conexión servidor-tanque, hemos de implementar la recepción de datos mediante gestos desde MYO al servidor con bluetooth.

Desarrollo: Dicha tarea consiste básicamente en añadir una librería[37] al servidor ya creado en el sprint anterior. A partir de aquí, solo queda interpretar desde el servidor los datos recibidos de MYO para enviar órdenes al tanque con Arduino, consiguiendo de esta manera mover el tanque mediante gestos con MYO.

5. Conectar Unity con el servidor. 1 semana

Planificación: Una vez resuelta la parte física del proyecto, es necesario crear el juego del que va a ser partícipe el tanque pero, antes que nada, necesitamos comprobar si es posible hacer efectiva la conexión entre el servidor y Unity, el segundo cliente.

Desarrollo: Dentro de esta iteración surgen las siguientes fases:

- Creación del cliente en Unity.
- Comprobación de la posibilidad de intercambio de información entre cliente y servidor.
- Recepción de información de gestos de MYO y de proximidad de Arduino.

Comprobado el funcionamiento de la conexión y la posibilidad de recibir los datos necesarios a través del servidor, solo me faltaba realizar el juego para terminar con el proyecto.

6. Crear del juego. 4 semanas

Planificación: Crear el juego que hará uso de todo lo anteriormente implementado. Aglutinamos en un solo sprint la última fase del proyecto aunque, sin lugar a dudas la más larga, por lo que la dividiré en varios "sub-sprints":

Desarrollo:

- a) Creación del menú principal.
- b) Creación de la pantalla "manual de juego".
- c) Creación del juego:
 - 1) Comprobación del uso de la Realidad Aumentada con la librería Vuforia.
 - 2) Creación de todos los actores del juego con assets de modelos 3D gratuitos de Unity asociándolos a patrones para hacer uso de la Realidad Aumentada.
 - 3) Creación de colisiones entre distintos modelos 3D haciendo uso de Realidad Aumentada.
 - 4) Establecer comportamientos distintos para los siguientes actores:
 - **tanque:** conseguir envolverlo con un bounding box y establecer su barra de vida y energía en el HUD. Disminuir su vida con colisiones en el entorno.
 - **armas:** conseguir que aumente la energía del tanque y desaparezca al colisionar con este.
 - **enemigos:** hacer que resten vida al tanque y crear una barra de vida propia encima del mismo modelo 3D que disminuya cuando reciba disparos del tanque. Hacer que desaparezca y le aumente vida al tanque al morir.
 - **meta:** conseguir la finalización con éxito del juego al colisionar el tanque con ella.
 - 5) Crear HUD (cronómetro de la partida y realimentación de las acciones del tanque).
 - 6) Interpretar las órdenes de MYO para disparar a los enemigos y hacer aparecer en pantalla una realimentación de disparo así como disminuir la energía del tanque.
 - 7) Interpretar la información recibida de Arduino sobre proximidad con obstáculos para hacerla aparecer en la realimentación por pantalla y decrementar la vida del tanque.

- d)* Creación de pantalla Game Over en caso de perder la partida.
- e)* Creación de pantalla resumen en caso de ganar la partida. Se muestra el tiempo empleado para terminarla.

Capítulo 2

Estado del arte

En este capítulo hablaré sobre el estado actual del arte o tecnologías similares al proyecto realizado, así como de cada una de las partes que lo componen.

En primera instancia hablaré sobre la realidad aumentada y su inclusión en los juegos actuales, continuando por la interacción hombre-máquina en relación al dispositivo de interacción gestual que he utilizado, el diseño e impresión 3D en la actualidad y por último, la existencia en la actualidad de un sistema con la mezcla de ambas artes. Además de todo esto, realizaré una pequeña crítica sobre el estado del arte.

2.1. La realidad aumentada

Realidad Aumentada (RA) es el término utilizado para describir la integración de un mundo virtual dentro del mundo real, el físico, a través de un dispositivo tecnológico. Es decir, a través de este dispositivo podremos ver como a la información física existente en el mundo real se le añade información virtual en tiempo real (por ello el término "aumentada", pues se le añade información a lo real, lo ya existente de forma física).

La Realidad Aumentada hace uso de patrones en el entorno físico para añadir, mediante un software creado, la información virtual al dispositivo tecnológico del que se hace uso.

2.1.1. Diferencia con la realidad virtual

La **Realidad Virtual** consiste en aislar al usuario completamente del mundo físico, generando un mundo totalmente virtual, a diferencia de la realidad aumentada, donde el usuario puede disfrutar a través del dispositivo tecnológico de la mezcla de realidad y ficción.

2.1.2. Inclusión en la actualidad

Hoy en día y cada vez más y más, la Realidad Aumentada existe en nuestras vidas cotidianas. Basta con hacer una búsqueda rápida en la App Store de Apple o en la Play Store de Android con la palabra clave "AR" (siglas de Augmented Reality en inglés) para darnos cuenta de que existen infinidad de aplicaciones que hacen uso de este arte. Desde aplicaciones para medir las dimensiones e inclinación de cualquier cosa como MeasureKit, aplicaciones para ver como quedaría nuestra casa con un nuevo mueble en un dormitorio como Ikea Place, hasta aplicaciones como Sky Guide AR que nos guían por el cielo mostrándonos constelaciones, estrellas y planetas incluso no visibles por el ojo humano.

Poco a poco esta tecnología mejora y, con ella, las ideas para hacer uso de la misma. Es, sin duda, una herramienta tanto de utilidad como de ocio, que cada vez irá formando parte más y más de nuestras vidas.

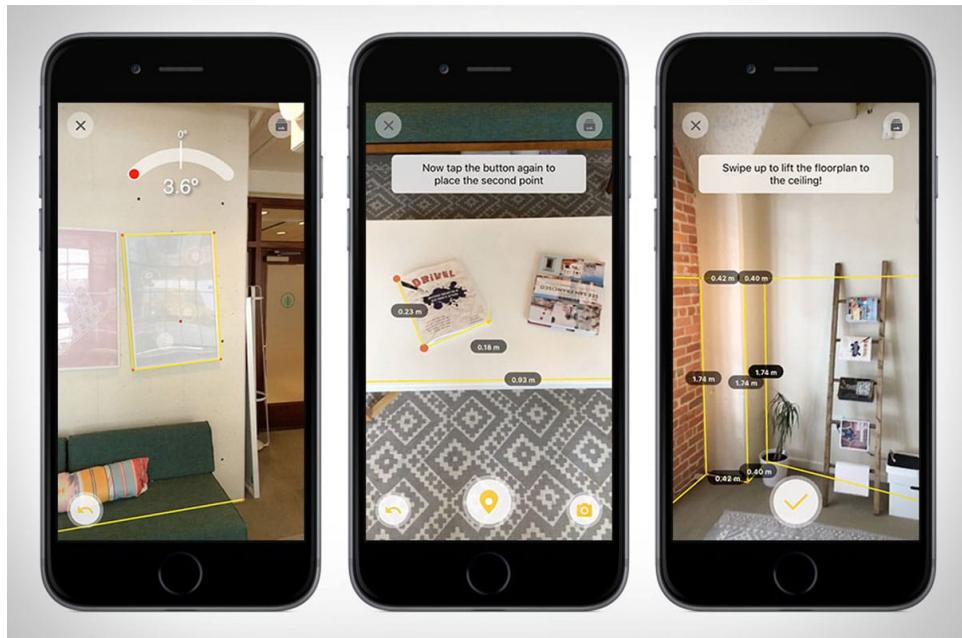


Figura 2.1: Capturas de pantalla de la app Measurekit[9]

RA en los videojuegos

Los videojuegos en formato estándar están bien consolidados en la sociedad, de forma que hoy en día no existe posibilidad de pensar que en un futuro puedan ser completamente sustituidos por la Realidad Aumentada o la Realidad Virtual, siempre existirán colectivos de jugadores que prefieran sentarse en el sofá de su casa con un mando en la mano y una pantalla

enfrente suya. Sin embargo, lo que sí se pretende, es buscar otras formas de entretenimiento en relación a los videojuegos.

Es aquí donde entra en acción la Realidad Aumentada, cooperando con otros modos de juego como la Realidad Virtual, en la inclusión de una nueva experiencia de juego para la inmensa cantidad de usuarios existente.

Se pretende, sobre todo, innovar en la forma en la que el jugador percibe el entorno de juego en la pantalla del dispositivo, provocando una inmersión en la partida y un asombro del jugador superior al conseguido con un videojuego en formato estándar.

Además de estas pretensiones, la Realidad Aumentada tiene la capacidad de hacer que el entorno físico forme parte del juego, no solo utilizándolo como soporte visual para dicho juego. Por ejemplo, este mismo año, para la asignatura Computación UbiCua, he desarrollado junto con un compañero un juego de Realidad Aumentada llamado ScapeRoomAR en el que un jugador, para crear la sala de escape ha de utilizar elementos físicos del entorno donde situar las pistas (modelos 3D) para lograr escapar de la habitación. De esta manera los jugadores que deseen escapar de la habitación deberán analizar todos los elementos de esta con su teléfono móvil para encontrar las pistas que les conduzcan a la salida.

Existen multitud de juegos que hacen uso de la Realidad Aumentada, como por ejemplo "The Machines", utilizado por Apple para hacer una demostración de las posibilidades en los videojuegos de esta técnica. Con dicho juego el usuario podrá establecer sobre una mesa un modelo 3D de la extensión de dicha mesa para desplegar tropas que combatirán a los enemigos, a los cuales el usuario también podrá disparar con distintas armas y bombas con el fin de destruirlos.



Figura 2.2: Captura de pantalla de un vídeo de YouTube sobre el juego The Machines.[10]

Es digno de mencionar en este apartado, un juego como Pokémon Go, el cual además del juego en sí, ofrece la posibilidad de hacer uso de Realidad Aumentada para ver a los Pokémons en el "mundo real". Incluso apunta en las normas del juego que, si al capturar al Pokémon te aproximas demasiado rápido a él, éste huirá, por lo que tendrás que aproximarte de forma cautelosa para poder capturarlo.



Figura 2.3: Captura de pantalla de un game play de Pokémon Go[11]

Por último cabe destacar el uso de un dispositivo como lo son las gafas de Realidad Mixta creadas por Microsoft: Hololens. Son un ordenador autónomo e inalámbrico dotado de sensores, sonido, una pantalla 3D esteroscópica de alta definición que permite al usuario interactuar con ellas mediante interfaces gestuales, de voz y mediante la mirada. En mi opinión, es un dispositivo novedoso que dará de qué hablar en un futuro. Uno de sus usos fue presentado en la gran convención de videojuegos E3 (Electronic Entertainment Expo) del año 2015 donde, con unas Hololens puestas en su cabeza, un usuario jugaba a Minecraft, juego que todos conocemos, cuya empresa fue comprada por Microsoft en el año 2014.



Figura 2.4: Presentación E3 Minecraft + Hololens[12]

2.2. Interacción hombre-máquina (HCI)

Tal como su nombre indica, la interacción hombre-máquina (Human Computer Interaction) hace alusión a la forma en la que una persona se comunica con un computador a través de una interfaz software. Este tipo de interacción existe desde el principio de las máquinas, pues el ser humano siempre ha necesitado comunicarle órdenes con el fin de que realizase alguna acción. El principal objetivo de una interacción hombre máquina, a parte de hacer realidad la comunicación entre ambos, es que esta resulte lo más cómodo posible para el ser humano, incrementando su satisfacción y productividad, y reduciendo al mínimo su frustración al utilizar cualquier tipo de máquina.

2.2.1. Historia de la HCI

Una vez descrita su definición, hacemos un repaso sobre su historia para hacernos una idea de su evolución con el paso de los años.

HCI clásica

No fue hasta los **años 60** cuando comenzó a existir la interacción propiamente dicha con una máquina. Fue más bien en los 70 con la llegada de los monitores, cuando se produjo la aceleración en el intercambio de infor-

mación con el usuario, aunque con una interacción un poco pobre debido al uso de interfaces mal diseñadas y difíciles a la hora de usar y aprender.

En los **años 80** los ordenadores personales (PC) comienzan a ser accesibles a muchos usuarios. Gracias a ello, la interacción se vio obligada a mejorar. El incremento de la eficiencia de las interfaces, vino en parte, de la mano de la existencia de personas con habilidades limitadas. También se comenzó a dar el paso de la UI(terminal negra) a la GUI(Interfaz gráfica), haciendo al usuario entrar en un entorno de interacción con la máquina mucho mas amigable. Además el uso del ratón y los menús facilitaron la realimentación al usuario. Fue también en esta década cuando comenzaron los primeros estudios sobre la HCI.

En los **90** el diseño de interfaces se convierte en una disciplina tratada científicamente, de modo que se produjeron importantes cambios en las interfaces gráficas de usuario (GUI), convirtiéndose así, en un sistema centrado en el usuario que, al fin y al cabo, es quien ha de aprender de manera sencilla a manejar el sistema.

HCI moderna

Actualmente la interacción hombre-máquina es vista como una interacción desarrollada en contextos sociales y de organización en la que diversos dispositivos tienen el fin de satisfacer variadas necesidades de las personas. La parte humana de la interacción se centra en estudiar principalmente su psicología, sus habilidades y sus limitaciones fisiológicas.



Figura 2.5: Interacción hombre-máquina[13]

2.2.2. Tipos de HCI

Una vez expuesta la definición de interacción hombre-máquina y un poco de su historia, me dispongo a describir los **tipos de interacción existentes**:

- **Explicita:** Es un tipo de HCI que pone al usuario en el centro del proceso, quien controla las operaciones del sistema(H2C, Human To Computer, el humano es quien se hace responsable de pensar como funciona el sistema). Al encargarse de ello, es posible que se sienta atosigado debido a tener que controlar varios sistemas. El interfaz ha de tener un buen diseño para que sea efectivo este tipo de interacción, ya que es lo único que ven muchos usuarios, aún así, su uso es complejo.

- **Implícita:** Tipo de HCI en el que las acciones que el usuario realiza no tienen como propósito interaccionar con el sistema de forma explícita, sino llevar a cabo una acción en concreto pero, que el sistema reconoce como una entrada a este para generar dicha acción. En este caso es el sistema el que se hace responsable de las acciones del usuario (C2H) ya que el usuario no ha de preocuparse de la interacción con el sistema, sino que este último es quien, de forma inteligente, capta las intenciones del usuario para producir una respuesta a los estímulos que generados por el humano.

Este tipo de interacción depende mucho del contexto humano, es decir, de cualquier información que caracterice al individuo que está utilizando el sistema. Este contexto humano puede ser complejo de determinar debido principalmente a la existencia de usuarios indecisos, el no determinismo de una persona, y el no determinismo del entorno, por ello el sistema puede requerir de bastante tiempo para construir dicho contexto. En la actualidad hay varios tipos de interfaces implícitas: lenguaje natural, tangibles, gestual, multimodal, etc.

El tipo de interfaz que nos interesa en nuestro proyecto, además del táctil de la pantalla del teléfono móvil donde se ejecutará el juego, es la interfaz gestual implícita, ya que hemos de controlar el tanque mediante gestos de la mano de una forma natural, sin preocuparnos de como funciona nada más que las acciones de girar hacia los lados, avanzar y disparar.

Con respecto a esto, existen múltiples dispositivos de interacción gestual en el mercado, como Kinect, Leap Motion y, el que hemos empleado: MYO Armband.

Myo Gesture Control Armband

En el capítulo de análisis hardware hablaremos con un nivel mayor de profundidad de esta interfaz de interacción gestual. De forma puntual en esta descripción de la HCI, cabe destacar que se trata de un brazalete que, adherido a nuestro antebrazo, es capaz de captar los impulsos eléctricos de los músculos que lo forman, pudiendo distinguir una gran cantidad de gestos de nuestro brazo y, a partir de los cuales he podido lograr el movimiento y el disparo del tanque.

¿Por que no he hecho uso de otros interfaces gestuales de interacción como Leap Motion o Kinect? Básicamente porque MYO Armband me ofrece dos características relevantes para el funcionamiento de mi proyecto/juego:

- **Es transportable:** para jugar al juego es necesario moverse por el espacio siguiendo continuamente la pista del tanque que manejas e investigando donde se encuentran los enemigos y las armas para derrocarlos, cosa que no es posible por ejemplo con Leap Motion, ya que este ha de estar adherido al ordenador y aunque fuese transportable resultaría mucho más incomodo tener que transportarlo de forma activa (no como con MYO, que lo llevas "implantado" en el antebrazo).
- **Es preciso:** me proporciona la suficiente precisión como para detectar los movimientos necesarios para el juego, no necesito más que unos movimientos concretos y sencillos para el desarrollo del movimiento del tanque.

2.3. Impresión 3D

La impresión 3D es una técnica de creación de objetos 3D a partir de la superposición de láminas de un cierto material (generalmente plástico ABS o PLA) gracias a un firmware instalado en un microcontrolador Arduino que es capaz de manejar la temperatura del extrusor de plástico, los motores de extrusión de plástico y de movimiento del extrusor a gusto de los modelos 3D enviados a la lógica de la máquina para reproducirlos materialmente en cierto tiempo. Es decir, con una impresora 3D, podemos obtener una copia real de un modelo 3D realizado por ordenador.

La calidad de impresión depende tanto del material, como de la configuración establecida con el software de impresión utilizado (altura de capa, relleno del objeto, el uso de estructuras para la impresión de voladizos...), así como del acierto a la hora de establecer las temperaturas tanto de la cama donde se imprime el modelo como de fundido del material a utilizar en cuestión para la impresión.

Para ir más allá y conseguir impresiones 3D de alta calidad, existen métodos

como el tratamiento con vapor de acetona, que hacen que las piezas impresas consigan eliminar las rugosidades de la superficie generadas por la propia impresión por capas de la pieza, aportándoles un brillo y una suavidad impecables.



Figura 2.6: Ejemplo de uso de la acetona en modelos 3D[14]

En resumidas cuentas, la impresión 3D es todo un arte complejo de dominar en su completitud que, en la mayoría de los casos (como ha sido el mío), se aprende a base de prueba y error.

2.3.1. Aplicaciones en la actualidad

Existen una inmensidad de ámbitos e ideas a las que aplicar el uso de la impresión 3D y es que, sabiendo utilizar un software de diseño 3D como Blender o Maya (por ejemplo), son pocas las limitaciones de una persona para crear cualquier objeto que pueda imaginar en cuestión de horas.

Según un estudio realizado por la cadena de televisión "La Sexta" [8] el 20 % de los colegios ya incluyen las impresoras 3D dentro de sus actividades, con el fin de enseñar a los más pequeños a manejar esta herramienta que, sin lugar a dudas, en un futuro no muy lejano, se introducirá en las vidas cotidianas de las personas facilitándolas en gran medida.

Como aplicaciones reales ya podemos ver en la actualidad prótesis para discapacitados, como manos, piernas o trozos de huesos, juguetes a gusto de cualquier niño, calzado a medida del diseñador y con todo lujo de personalizaciones, fundas para teléfonos móviles, objetos decorativos, casas impresas en 3D o, algo más peligroso, armas de fuego impresas en 3D.



Figura 2.7: Casa impresa en 3D[15]

Como vemos, un sin fin de aplicaciones son las que guarda la impresión 3D, cada vez más próxima al uso personal y cotidiano por parte de la sociedad.

Haciendo uso de esta herramienta es como decidí crear el tanque usado para el juego, ya que me apasiona este mundo y, recientemente he aprendido a utilizar de forma básica el software de diseño 3D Blender.

2.4. Existencia de proyectos similares

Hoy en día, hasta donde he podido alcanzar en mí búsqueda, existen escasos proyectos en los que se entremezclen el dispositivo de interacción gestual MYO con la Realidad Aumentada, de hecho, solo he encontrado una persona en internet que se haya dedicado a trabajar con ambas artes. Se trata de un investigador de la universidad de Portsmouth (UK) que ha creado una serie de aplicaciones sencillas que combinan los movimientos gesticulares de MYO con la Realidad Aumentada. Uno de sus proyectos es un teclado virtual que nace en pantalla de un patrón físico y por el que puede, con ayuda de MYO deslizarse para escribir lo que deseé.



Figura 2.8: Demostración de uso del teclado de Realidad Aumentada junto con MYO[16]

Además de este, posee bajo su creación varios proyectos del mismo estilo excepto uno, que es el que mayor similitud guarda con mi proyecto de fin de grado: un videojuego que combina Realidad Aumentada con MYO Armband. Se trata de una escena donde se encuentran dos personajes enfrentados en la que, uno de ellos es controlado por el usuario y ha de eliminar a su enemigo mediante gestos con MYO Armband. A continuación una captura de pantalla donde podemos observar el escenario y, en la esquina superior izquierda, el brazo del usuario realizando un gesto (en la imagen no es visible, para ver correctamente los gestos he dejado el enlace en la referencia).



Figura 2.9: Juego Realidad Aumentada + MYO[17]

Tras una intensa búsqueda por la red, repito, es lo único que he logrado encontrar que se asemeja a la idea que he desarrollado con mi proyecto. Lo más cercano a estos dos juegos (el mío y el de la Figura 2.8) son juegos exclusivos de MYO, donde puedes hacer uso del brazalete para controlar el juego en una pantalla (hay varios, de hecho hay un market exclusivo de juegos para MYO) y, juegos exclusivos de realidad aumentada, en los que puedes hacer uso de esta técnica pero sin MYO Armband.

En resumidas cuentas, debido a esta falta de proyectos similares, pienso que he escogido una idea novedosa que, aunque sea de forma minúscula, acerca al jugador un poco más hacia el futuro de los videojuegos al combinar estas dos técnicas novedosas.

2.5. Crítica al estado del arte y aportes propios

La **Realidad Aumentada** ha ido experimentando un crecimiento a pasos agigantados en los últimos años, y es que es una apuesta segura de futuro tanto con fines de entretenimiento como didáctico y de utilidad. Tanto es así que, grandes empresas como Apple con su ARKit (según ellos, la plataforma de AR más grande del mundo), han apostado por su desarrollo en su software más reciente, lo cual me parece una gran idea viendo la cantidad de proyectos y buenas ideas existentes utilizando esta técnica.

En relación a la **interacción hombre-máquina**, cabe destacar que se trata de un arte con más recorrido que, por ejemplo, la Realidad Aumentada. Desde mi punto de vista, no existe una interfaz de usuario óptima para todos los ámbitos, sino que cada cual sirve para cosas distintas. Con esto quiero decir que hay ciertos ámbitos en los que la interacción hombre-máquina ha llegado a su punto álgido(o al menos próximo). Por ejemplo, pienso que la mejor forma de controlar un ordenador es mediante un teclado y un ratón, aunque quizás lo óptimo sería controlarlo con un dispositivo que leyese tus pensamientos y supiese qué deseas en cada momento, pero eso es una utopía de la que aún estamos muy lejos. En otros ámbitos, la sanidad por ejemplo, sería interesante desarrollar de manera más precisa las interfaces gestuales como puede ser MYO Armband, ya que puede ser de utilidad por ejemplo, para la rehabilitación de pacientes con lesiones en las muñecas. Para mi gusto, este tipo de avances tecnológicos reciben menos atención de la necesaria, ya que con un poco más de esfuerzo podrían llegar a sernos de utilidad en un futuro próximo.

Con respecto a la **impresión 3D** solo cabe destacar que es un mundo que, aunque aún le queda mucho por recorrer hasta asentarse en la vida cotidiana de las personas, ya se encuentra realmente avanzado y existe una gran comunidad de "makers" (cultura "do it for yourself") en internet que

respaldan y contribuyen en continuar con el desarrollo de este arte que, sin duda, también jugará un importante papel el día de mañana en nuestras vidas.

Para terminar con la crítica, solo cabe comentar la falta de proyectos existentes en la red cuyo propósito se acerque al desarrollado por mí en esta ocasión. Como ya he dicho, tan solo uno (hasta donde ha podido alcanzar mi búsqueda) se asemeja en cierto modo a mi propuesta, a falta de hacer uso de un personaje real como lo es mi tanque complementado de un Arduino UNO, lo que me lleva a pensar en que podría incluso tratarse de un proyecto pionero en el sentido de haber juntado Myo Armband con Arduino Uno y Realidad Aumentada al mismo tiempo para crear un videojuego para teléfono móvil.

Capítulo 3

Análisis del Hardware

Este capítulo va dedicado al análisis del hardware utilizado para el desarrollo completo del proyecto. Hablaré primeramente del microcontrolador utilizado para el control del tanque, así como para el control de la red WiFi y la captación de información de proximidad a obstáculos, continuaré con la propia creación del tanque y, para terminar, una descripción del dispositivo de interacción gestual utilizado.

3.1. Microcontrolador del tanque

Para conseguir los movimientos y funcionalidades del tanque que comentaré más adelante, he utilizado una sola placa **Arduino**, la cual ha cubierto las necesidades de mi proyecto sin ningún tipo de problema.

¿Por que no he utilizado otro tipo de dispositivos tales como Raspberry Pi o BeagleBone? Muy sencillo: estos tipos de dispositivos gozan de mayores capacidades de procesamiento de datos, sí, pero no es algo que mi sistema busque. Se tratan de pequeños ordenadores capaces de ejecutar sistemas operativos, lo cual difiere mucho en requerimientos hardware de las necesidades de mi proyecto, las cuales son totalmente asequibles para las capacidades de Arduino. A continuación explicaré de forma detallada el por qué de mi elección en cuanto al hardware controlador del tanque.



Figura 3.1: Placa Arduino [18]

3.1.1. ¿Por qué Arduino?

Las principales causas de haberme decantado por el microcontrolador programable Arduino son las siguientes:

- **Cumple con lo necesario:** a la hora de pensar en qué hacer como proyecto de fin de grado y, una vez desarrollada la idea en mi cabeza, me pregunté qué tipo de sensores y herramientas me serían necesarias para llevar a cabo dicho proyecto. Arduino me ofrecía todo lo necesario para ello: sensor de proximidad, motores y adaptador wifi podrían ser acoplados sin problemas al microcontrolador para llevar a cabo las funcionalidades necesarias. Además, en caso de disponer de tiempo y de que lo creyese conveniente, me permitiría añadir aún más y más dispositivos electrónicos como podrían ser un mando a distancia, una pantalla LCD, LED's de colores, etc. con ayuda de las llamadas "shields", que no son más que placas de circuitos acoplables a una placa Arduino con el fin de incrementar sus capacidades o funcionalidades (su hardware).
- **Fácilmente programable:** simplemente es necesario tener soltura en C como lenguaje de programación. Con los sensores necesarios, código poco extenso y un poco de imaginación, la capacidad multidisciplinar de este microcontrolador es abrumadora.
- **Open hardware:** lo que permite abaratizar costes a la hora de comprar sensores y herramientas, ya que disponemos del lujo de poder hacernos

con complementos e incluso placas de otros distribuidores. Además nos permite la comercialización de nuestros propios diseños hardware incluyendo dicha placa en ellos.

- **Software libre:** existe una cantidad inmensa de proyectos Arduino realizados por la Comunidad en internet por lo que, seguramente casi cualquier idea que se le ocurra a un desarrollador (o algo similar) ya esté implementada en alguna parte de "la red", lo cual resulta de gran ayuda a la hora de llevar a cabo cualquier proyecto que se tenga en mente.
- **Gran soporte de la Comunidad:** en internet no solo existen proyectos iguales o similares a lo deseado, sino que además cualquier problema que pueda surgir queda respaldado por una amplia Comunidad de desarrolladores de todo el mundo. Ya sea preguntando personalmente o bien consultando preguntas ya existentes, resulta trivial solucionar problemas relacionados con Arduino en foros (principalmente) dedicados a ello.

3.1.2. Tipos de placas Arduino

Existen en el mercado una gran cantidad de tipos de placas de circuitos integrados Arduino, cada una con unas especificaciones concretas en cuanto a memoria, microcontrolador, voltaje de funcionamiento, etc., lo que permite a un desarrollador escoger a su gusto las características hardware que le permitirán implementar un proyecto en concreto.

A continuación voy a explicar las nociones básicas para saber escoger una u otra placa Arduino a la hora de realizar un proyecto. Comienzo explicando los tipos de memorias que utilizan los microcontroladores que incorporan las placas Arduino y, seguidamente, procederé a explicar los distintos tipos de microcontroladores para, más tarde, poder diferenciar entre unos y otros y así decantarse de manera correcta por un tipo de placa Arduino.

Tipos de memorias:

- **SRAM** (Static Random Access Memory): sirve para el almacenamiento de datos y variables de forma volátil, es decir, durante la ejecución de un programa. Se utiliza como un banco de registros, cuando no tiene alimentación eléctrica deja de almacenar los datos del programa(memoria volátil). Es de capacidad limitada, por lo que es necesario hacer un uso conciencioso de ella para no agotarla durante la ejecución y evitar resultados inesperados.
- **EEPROM** (Electrically Erasable Programmable Read-Only Memory): es un tipo de memoria no volátil (no necesita corriente eléctrica para

almacenar información en ella) la cual puede ser programada y borrada eléctricamente. Usualmente se utiliza para almacenar las constantes de un programa.

- **Flash:** es la memoria del programa, donde se almacena este ya compilado junto con el bootloader (el nos permite programar Arduino a través del puerto serie. Este se ejecuta siempre en el arranque de la placa para comprobar si se está intentado programar sobre ella o no). No es posible modificar los datos almacenados en esta memoria, para ello, han de ser copiados a la memoria SRAM.

Como he comentado inicialmente, a continuación describiré los tipos de **microcontroladores existentes en placas Arduino**:

- **ATmega328[4]:** Microcontrolador con arquitectura de 8 bits de la familia AVR de Atmel(microcontroladores RISC, es decir, con un conjunto de instrucciones reducidas y de tamaño fijo). Consta de una memoria **flash de 32KB**, una **SRAM de 2KB** y una **EEPROM de 1KB** además de una velocidad máxima de cómputo de **20 MIPS** (millones de instrucciones por segundo).
- **ATmega32u4[5]:** comparte exactamente las mismas características que la anterior exceptuando la **SRAM** que en este caso es de **2.5KB** y además es un poco más lenta, trabajando a una velocidad de 16 MIPS.
- **ATmega2560[3]:** precisa de un incremento sustancial en memoria con **256KB de memoria flash, 8KB de SRAM y 4KB de EEPROM**. Mantiene las 16 MIPS como velocidad máxima del microprocesador descrito anteriormente.
- **ATmega168[2]:** conserva la velocidad de procesamiento de estos dos últimos microprocesadores descritos (**16 MIPS**), dispone de **512B de EEPROM, 1KB de SRAM y 16KB de memoria flash**.
- **AT91SAM3X8E[1]:** es el más potente de los microprocesadores, pues consta de una memoria **flash de 512KB, SRAM de hasta 100KB** y una velocidad máxima de 84MHz (siendo 16 el estándar para este tipo de procesadores).

Finalmente, conocidas las nociones necesarias, dispongo las **principales placas Arduino[36]**:

- **Arduino Uno:** es la versión más estándar y a la vez la más conocida de este tipo de placas. Dispone de un procesador Atmega328.

- **Arduino Mega:** es la **versión más potente** de Arduino con microcontrolador **de 8 bits** ya que consta del procesador Atmega2560, además de ser la versión que más pines entrada/salida tiene. Ideal para proyectos de complejidad alta.
- **Arduino Due:** es la **versión con mayor capacidad de procesamiento**, ya que está basado en un microprocesador de **32 bit** en lugar de 8 y posee el microporcesador AT91SAM3X8E, el más potente de los descritos anteriormente.
- **Arduino Yún:** posee el microcontrolador ATmega32u4 y, además, incorpora un microprocesador MIPS con un Sistema Operativo Linux embebido, lo que facilita, a diferencia de las demás placas, la comunicación con una distribución Linux, aparte de proporcionar características como WiFi, Ethernet y lector de tarjetas Micro-SD. Ideal para su uso en el ámbito del Internet Of Things
- **Arduino Nano:** dispone de un microprocesador ATmega328, al igual que la versión estándar de Arduino. La principal diferencia es su pequeño tamaño.
- **LilyPad Arduino:** precisa de un microprocesador ATmega328 y, su uso principal son los wearables y e-textiles, es decir, para ser cosidos e instalados sobre prendas habituales.

Una vez expuestos los principales tipos de placas Arduino existentes procedo a defender mi elección. Para el desarrollo de este proyecto decidí escoger la primera de las placas explicadas, Arduino Uno, ya que, habiendo estudiado las características de esta, se adaptaba perfectamente a los requerimientos principales de memoria y capacidad computacional. Además, anteriormente ya la había utilizado para algún pequeño proyecto, por lo que, poseyendo un ejemplar y conociendo de forma básica su funcionamiento, no había necesidad alguna de hacer uso de otro tipo de placa. Para su alimentación energética he hecho uso de una batería externa de 10000mAh, más que suficiente para el uso de Arduino UNO durante un buen rato, ya que apenas consume.

3.2. Módulos complementarios utilizados con Arduino

Como es de esperar, no basta solo con poseer una placa Arduino para poder hacer uso de las funciones requeridas en este proyecto. Además de ello hemos de hacer uso de una serie de accesorios acoplables a nuestra placa que a continuación describiré y que permitirán al tanque implementado realizar

movimientos(motores), medir la proximidad a un obstáculo cercano (sensor de proximidad) y enviar y recibir información a través de una conexión con el servidor creado (módulo WiFi).

3.2.1. Movilidad del tanque. Motores servo

Para poder realizar giros hacia un lado y otro, así como para poder avanzar, el tanque ha de estar equipado de al menos dos motores que se encarguen de ello. Para cumplir con dicha funcionalidad he hecho uso de dos motores servo, los cuales son idóneos para robótica, ya que permiten realizar giros precisos (un determinado número de vueltas) con un mayor par de torsión (torque, fuerza de giro) que un motor DC normal y corriente. Estos últimos, los motores DC (corriente continua), no pueden ser utilizados ya que son motores incapaces de realizar una cantidad determinada de vueltas, simplemente giran y giran mientras exista corriente eléctrica en su interior. Un servomotor no consta simplemente del motor, sino que precisa además, de un sistema mecánico que le proporciona esa fuerza de torsión al motor en sí, por lo que un servomotor queda compuesto de elementos eléctricos (un motor DC normal y corriente en el interior), mecánicos (la estructura externa a la que se acopla el eje del motor DC formada por engranajes) y electrónicos (controladores de la lógica del motor).

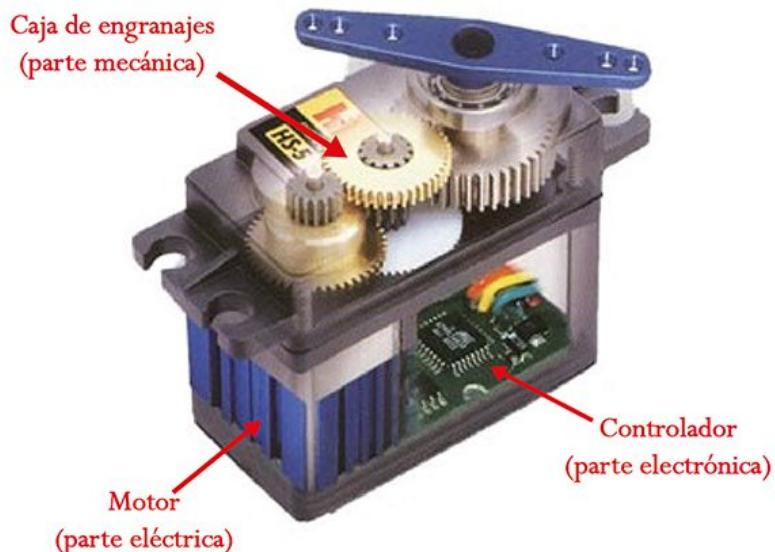


Figura 3.2: Servomotor [19]

En definitiva, un servomotor no es un simple motor, sino que es un sistema compuesto por un motor DC, un sistema electrónico para controlarlo y una estructura que le proporciona propiedades de control y precisión superiores a las de un motor normal y que, me resultan completamente útiles para la implementación del movimiento del tanque desarrollado.

Los motores servos que adquirí tenían una restricción en el giro que solo les permitía rotar 180 grados, por lo que tuve que desmontarlos, cortar la pieza que delimitaba el movimiento y sustituir el potenciómetro existente (el cual se encarga del control del giro) en el interior del mecanismo del motor por dos resistencias de 2K Ohms y una tolerancia del 1%. Gracias a ello los motores servo pasaron a poder girar 360 grados hacia ambos lados.

3.2.2. Sensor de proximidad HC-SR04

[7] Con el fin de proporcionar al tanque unos "ojos" con los que saber si está próximo a un obstáculo y además, cómo de próximo se encuentra a este, he hecho uso del sensor de proximidad HC-SR04.

Se basa en un sensor de ultra sonidos que emite ondas a una frecuencia no audible por el ser humano (40KHz en este caso) que nos permite, a partir de una serie de cálculos con la emisión y recepción de dichas ondas, conocer distancias de los objetos interpuestos en un ángulo de 15 grados a una distancia de entre 2 y 400 centímetros.



Figura 3.3: Sensor HC-SR04 [20]

Composición

Consta de cuatro pines o patillas:

- **Alimentación(Vcc)**: encargada de recibir los 5 voltios de corriente continua.
- **Disparador(Trig)**: cuyo trabajo es enviar ondas sónicas.
- **Receptor(Echo)**: que se encarga de recibir esas mismas ondas que envía el disparador.
- **Tierra(Gnd)**: en caso de seguridad.

Funcionamiento

El funcionamiento del sensor consiste en medir el tiempo en el que una onda sale desde el disparador, rebota en un obstáculo dentro del rango [2,400]cm y regresa al receptor.

Conociendo la velocidad del sonido (343 m/s) con unas condiciones de temperatura y presión estándares, podemos deducir de la siguiente forma el tiempo que tarda el sonido en recorrer una cierta distancia (en este caso un centímetro):

$$343 \frac{m}{s} \cdot 100 \frac{cm}{m} \cdot \frac{1}{1000000} \frac{s}{\mu s} = \frac{1}{29.2} \frac{cm}{\mu s}$$

Figura 3.4: Calculamos el tiempo por centímetro [21]

Ahora sabemos que el sonido tarda 29.2 microsegundos en recorrer un centímetro por lo que, con una simple regla de tres, sabiendo el tiempo que tarda en llegar una onda desde que salió desde el disparador, podemos calcular la distancia recorrida por la onda, teniendo así la distancia desde el tanque hasta el obstáculo más próximo. La fórmula es la siguiente:

$$Distancia(cm) = \frac{Tiempo(\mu s)}{29.2 \cdot 2}$$

Figura 3.5: Calculamos la distancia [21]

Dividimos entre 2 el tiempo empleado porque **solo** nos interesa el tiempo que ha tardao la onda en ir (o para volver), no nos interesa el empleado para ir **y** volver. De esta forma ya sabemos con certeza la distancia en centímetros del tanque a su primer obstáculo.

Desventajas

Este tipo de sensores disponen de una baja precisión, la cual puede verse gravemente afectada por la orientación de los obstáculos a medir, ya que pueden provocar reflejos en la onda que producirán malas mediciones. Así mismo son poco adecuados para entornos exteriores y con un gran número de objetos. Para futuras versiones este es un aspecto a mejorar, sería necesario encontrar un sensor de mejor calidad aunque resulte más costoso económicamente.

3.2.3. Módulo WiFi

[6]Para poder enviar y recibir información del servidor principal, he dotado de conexión WiFi a la placa Arduino que incorpora el tanque con el módulo ESP8266. Este módulo se trata de un chip integrado que dispone de un microcontrolador con conexión WiFi y es compatible con el protocolo TCP/IP (del cual hablaremos más tarde). Sirve para dar acceso a una red a cualquier microcontrolador, en este caso nuestra placa Arduino.

Con este módulo conseguimos crear una red WiFi a la que nuestro servidor y nuestro teléfono móvil han de conectarse para poderse intercambiar información entre ellos mediante sockets TCP. Es una pieza fundamental del proyecto ya que, sin el uso de esta red no sería posible la comunicación entre los distintos dispositivos y no hubiera podido llevarlo a cabo.

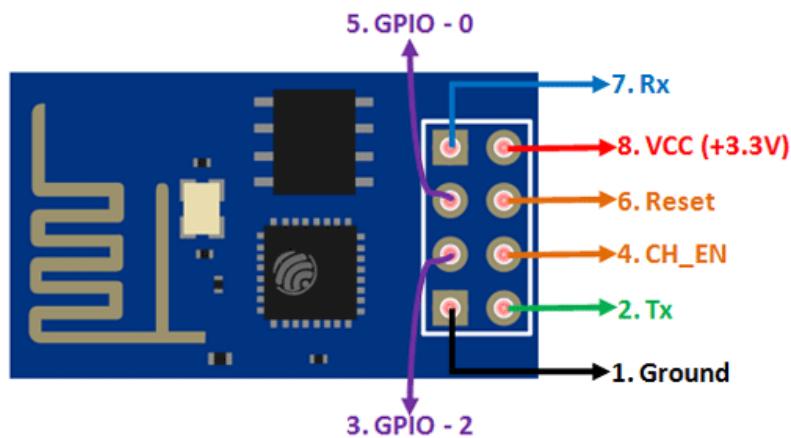


Figura 3.6: Patillas ESP8266 [22]

Características

[35] Integra un procesador Tensilica L106 RISC de 32 bit que consigue un consumo extremadamente bajo con una velocidad máxima de reloj de 160MHz.

Con el fin de ahorrar consumo, dispone de tres modos de operación:

- **Active mode:** rendimiento máximo.
- **Sleep mode:** el único elemento activo es el reloj, con el fin de mantener la sincronización. Permanece en la espera de estímulos que le hagan despertar. Mantiene en la memoria los datos de conexión para que no sea necesario volver a establecer la conexión. Consumo entre 0.6mA y 1mA.
- **Deep sleep:** el reloj permanece activo pero inoperativo. Los datos no almacenados se pierden.

Además, el módulo ESP8266 es capaz de soportar altas temperaturas, por lo que tiene, según Espressif (el fabricante) una alta duración, pudiendo funcionar de manera consistente en entornos industriales. Dispone de una memoria flas de 512KB y puede ser utilizado como estación, como punto de acceso o como ambos combinados.

Conectores

Dispone de 8 conectores tal como podemos ver en la Figura 3.6:

- **TX:** Transmisor asíncrono universal(UAT)[34]. Se hace uso de el para enviar caracteres.
- **CH_PD:** Chip power down/Chip enable.
- **RST:** Reset.
- **VCC:** Alimentación de 3.3V.
- **GND:** Tierra.
- **GPIO2:** Entrada/salida de propósito general 2.
- **GPIO0:** Entrada/salida de propósito general 0.
- **Rx:** Receptor asíncrono universal(UAR). Se hace uso de el para recibir caracteres.

Los GPIO pueden ser configurados para ser entrada o salida y estos pueden ser activados o desactivados.

3.3. El tanque

Para la construcción del personaje principal del juego, el tanque, he hecho uso de mi impresora 3D Prusa i3 Psique Steel, de forma que el tanque entero es una combinación de piezas de plástico (PLA), diseñadas también por mí mismo (en gran parte) con el software de modelado 3D Blender. El tanque en sí consta de tornillos y tuercas para acoplar las distintas piezas. Tiene una estructura para el microcontrolador Arduino, otra para la batería, soporte para el eje de las ruedas traseras, una rueda de gusano a cada lado del tanque y soportes para los motores.



Figura 3.7: Foto de mi Prusa i3 Psique Steel

3.3.1. Tipos de plástico. Mi elección

Los tipos de plástico para la impresión 3D son los siguientes:

- **ABS**(acrilonitrilo butadieno estireno): derivado del petróleo, con una

resistencia y flexibilidad elevadas, así como su resistencia a altas temperaturas (temperatura de impresión a unos 230 grados centígrados). Emite gases, requiere de una cama caliente para poder imprimirse con él y resulta complicado a la hora de imprimir, si no se sabe controlar pueden surgir varios problemas (resquebrajamientos a mitad de impresión, levantamiento de la cama en medio de la impresión, etc.).

- **PLA**(ácido poliláctico): es un termoplástico biodegradable fácilmente en agua y óxido de carbono, hecho de recursos renovables como la caña de azúcar o el almidón de maíz. No emite gases nocivos, no es necesario usar una cama caliente para su impresión y es mucho más sencillo de controlar a la hora de imprimir, por lo que los problemas se reducen de gran manera con respecto a la impresión con ABS. A cambio, es menos resistente, soporta temperaturas inferiores a las del ABS (temperatura de impresión sobre 180 grados centígrados) y el mecanizado, pintado y pegado de piezas tras haberlas impreso, es más complicado.

Para este proyecto, como ya he dicho, he utilizado para la realización de las piezas el tipo de plástico PLA, debido a su facilidad de impresión, los pocos requisitos mecánicos (soporte de tracción, torsión o compresión de las piezas) del tanque y, sobre todo, porque es un tipo de plástico biodegradable que además, no produce gases nocivos al ser impreso.

3.4. MYO Gesture Control Armband

Por último hablaremos del aparato desarrollado por Thalmic Labs que hace posible el control a distancia mediante gestos del tanque. MYO, como ya hemos comentado en alguna ocasión, es un brazalete con un sistema de captación de impulsos eléctricos de nuestros músculos el cual le permite distinguir una gran cantidad de los gestos que puede realizar el brazo de un ser humano, ya sea movimientos de muñeca, dedos, brazo o antebrazo.

¿Por qué no hacer uso de una muñequera en lugar de un brazalete? Sencillamente porque el antebrazo logra captar una cantidad mayor de gestos que la muñeca, ofreciendo un abanico mayor de posibilidades.

Componentes del brazalete:

- **Cadena de eslabones:** sensores eléctricos (EMG) de grado médico de acero inoxidable de alta sensibilidad en cada uno de los ocho eslabones, lo cual le otorga su gran precisión.
- **Bluetooth 4.0:** hace uso de este tipo de conectividad para la conexión con los dispositivos que pretende controlar.

- **LEDs:** indicadores de batería, su estado de conexión y ciertos estímulos.
- **Procesador ARM Cortex M4**
- **Batería de litio:** que proporciona alrededor de un día de autonomía.
- **Puerto Micro USB:** tanto para cargar el dispositivo como para sincronizarlo con el receptor adherido al ordenador en caso de querer hacer uso de este.
- **Unidad de medición inercial (IMU)** que contiene los siguientes elementos:
 - **Giroscopio** de tres ejes.
 - **Magnetómetro** de tres ejes.
 - **Acelerómetro** de tres ejes.
- **Feedback háptico:** mediante vibraciones de corta, media y larga intensidad.

Para beneficiarse de la alta precisión de MYO es aconsejable (sino necesario) crear un perfil personal de gestos. Consiste en una guía rápida en la que se pide al usuario realizar una serie de gestos básicos que servirán para calibrar los movimientos de cada individuo.

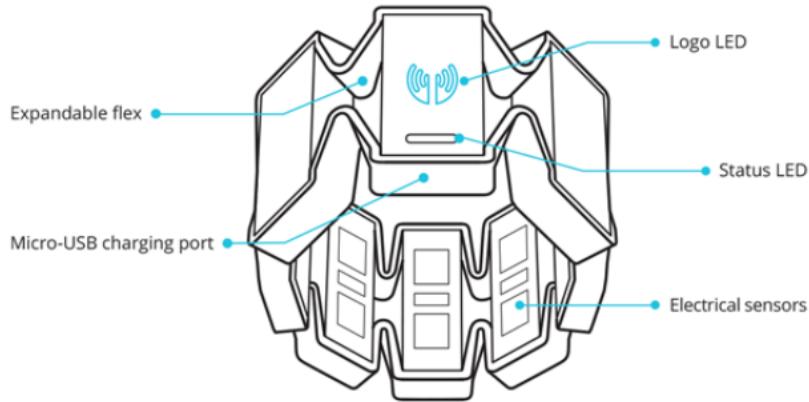


Figura 3.8: Esquema MYO Armband[23]

Una de las aplicaciones más relevantes para las que puede ser desarrollado este dispositivo es para la **accesibilidad** a funciones o acciones de diversos ámbitos de **personas con movilidad reducida**. Con un sistema como MYO Armband, una persona discapacitada puede controlar ciertos

mecanismos de su hogar como un sistema de riego, la televisión, las persianas o cualquier aparato eléctrico de su casa con tan solo registrar ciertos movimientos programados con dichos fines.

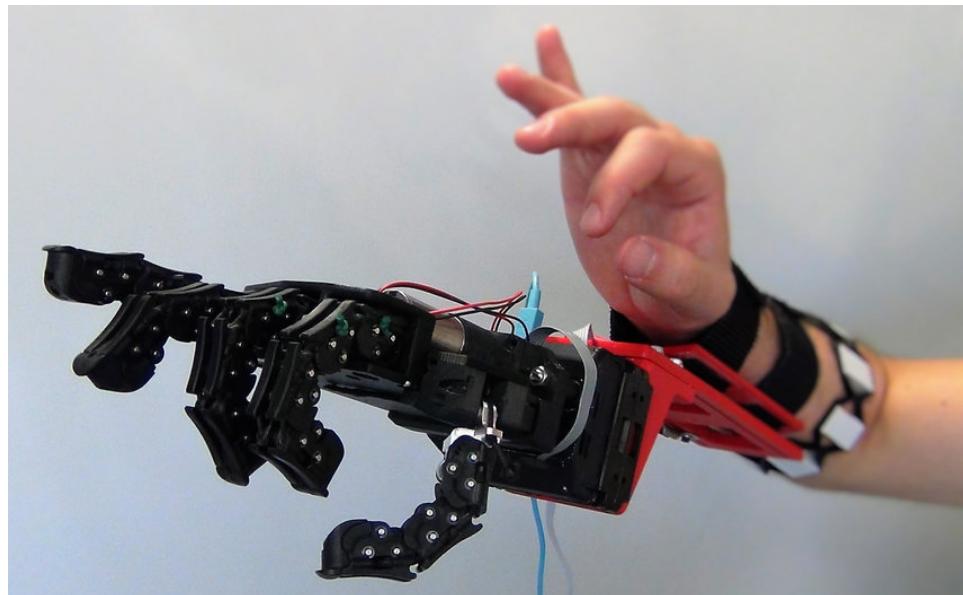


Figura 3.9: Ejemplo de uso de MYO[24]

Cabe destacar que MYO Armband es compatible con dispositivos Windows, Mac OS, iOS y Android, lo que hace que todo tipo de usuarios pueda tener acceso a su uso y al desarrollo de aplicaciones para todo tipo de terminales.

Actualmente existen una gran cantidad de aplicaciones y juegos con las que utilizar MYO. Aunque es un tanto incómodo para el uso cotidiano, he podido comprobar personalmente que también es posible controlar ciertas funciones de un ordenador con sistema operativo Windows con unos cuantos gestos preestablecidos en el software que instala el dispositivo para poder funcionar.

Como punto en contra del dispositivo, cabe destacar que en ocasiones la toma de interpretaciones gestuales no es correcta o no es detectada, posiblemente debido a una mala calibración antes de ser utilizado, por error del propio dispositivo (cosa que pienso que es poco probable, aunque desconozco) o simplemente por hacer un mal uso de este.

Capítulo 4

Análisis del Software

Tras analizar al completo el hardware utilizado para la creación del proyecto, pasamos a continuación a describir el software que me ha ayudado con el diseño de modelos 3D, con la impresión de piezas para el tanque, la programación del microcontrolador Arduino y sus módulos, con la creación del servidor y, por último, con el desarrollo del juego.

4.1. Diseño de modelos 3D. Blender

Blender es un programa software libre creado para tareas como modelado, animación e iluminación (por ejemplo). Además de esto, Blender posee un motor de juegos interno, por lo que se pueden desarrollar juegos con él.

Es criticado por poseer una interfaz poco intuitiva pero, con el uso, el usuario llega a darse cuenta de que resulta cómoda debido a su versatilidad, la configuración personalizada de la que ha sido dotado.



Figura 4.1: Interfaz y ejemplo de Blender[25]

Posee una alta variedad de primitivas geométricas, cinemática inversa, edición de audio y vídeo, características interactivas para videojuegos, como lo son las colisiones y las recreaciones dinámicas. En conclusión, se trata de una herramienta completa para tareas como renderizado o creación de gráficos tridimensionales aparte de las ya mencionadas.

Como experiencia con Blender a la hora de diseñar las piezas que componen el tanque del videojuego, he de decir que al principio, al no conocer el funcionamiento del programa, la interfaz resulta abrumadora, pues hay botones por todas partes. Aun así, poco a poco fui aprendiendo a manejarla con la herramienta para ser capaz de desarrollar el trabajo que he realizado y, he de decir que a día de hoy (que no soy, para nada, un usuario avanzado en el uso de Blender) me siento cómodo con esta herramienta.

4.2. Impresión de piezas 3D

Para esta tarea he utilizado el software de impresión avanzada Simplify 3D, el cual permite controlar cada uno de los aspectos a la hora de imprimir los diseños 3D. Desde temperatura de cada uno de los componentes de la máquina, pasando por altura de capa de la impresión, cambios de temperatura y velocidad durante la impresión, hasta cambio en los movimientos de la impresora a la hora de imprimir en caso de que la forma automatizada en la que decide el propio software realizar la pieza no sea del agrado del usuario. Como digo, es un software completo que permite adaptarse a las necesidades y gustos del usuario que lo maneja.

4.3. Programación de Arduino

Arduino cuenta con su propio software de desarrollo. Se trata de un IDE(Entorno de Desarrollo Integrado) escrito en el lenguaje de programación Java y bajo la licencia GNU, que sirve básicamente para pasar código (escrito en lenguaje C) a la placa Arduino.

El funcionamiento básico del programa es muy sencillo, se basa en el uso de dos funciones: la función de inicialización del programa (`setup()`) y, a continuación, un bucle infinito (`loop()`) donde se desempeñará la función deseada.

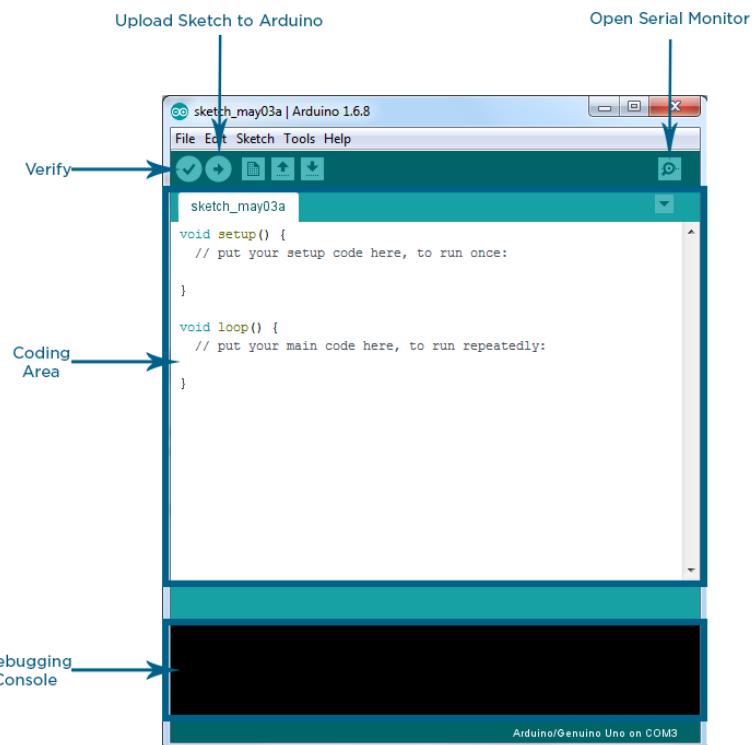


Figura 4.2: Partes básicas del Arduino IDE[26]

Una vez programado lo deseado, para convertir el código ejecutable en un archivo de texto en codificación hexadecimal que se cargará en el firmware de la placa, Arduino IDE utiliza el programa "avrdude".

4.4. Creación del servidor. Netbeans

Desde aquí controlamos el flujo de datos entre Arduino, MYO y Unity, por lo que resulta una parte imprescindible del proyecto. Para la implementación del servidor he hecho uso del IDE Netbeans de Apache Software Foundation, principalmente por tres motivos:

1. Me ofrece todas las funcionalidades necesarias.
2. Me siento cómodo con él, además de que ya sabía utilizarlo por haberlo usado en asignaturas como Programación y Diseño Orientado a Objetos.
3. Es un producto libre y gratuito que goza de plenitud de funcionalidades sin restricciones de uso.

4.5. Desarrollo del videojuego. Unity

Por último, hablaré del motor de videojuegos que me ha servido de utilidad para realizar el proyecto: Unity. Consiste en una plataforma de desarrollo de videojuegos multiplataforma creada por Unity Technologies. Es el motor de videojuegos, junto con Unreal Engine, más utilizado hoy en día por desarrolladores de todas partes del mundo debido principalmente a su potencial. Es una herramienta que, de forma gratuita, permite a cualquier desarrollador acceder a la gran mayoría de funcionalidades de la aplicación y, muy importante, permite descargar assets gratuitos del Asset Store incorporado en la herramienta.

Unity presenta las siguientes **funcionalidades**:

- Trabajo tanto en 2D como en 3D.
- Inteligencia artificial.
- Scripting
- Puede usarse junto con Blender, 3ds Max, Adobe Photoshop y una gran cantidad de programás más que le proporcionan un aumento de calidad a la hora del desarrollo.
- Creación de juegos multijugador en red.
- Tratamiento de gráficos y físicas.
- Tratamiento de audio.
- Creación de animaciones.
- Soporte para Realidad Virtual y Realidad Aumentada.
- Acceso a repositorios Open-siurce



Figura 4.3: Juego Hearthstone, hecho con Unity[27]

¿Por qué he escogido Unity? Son varios los motivos:

- Presenta un manual muy bien detallado y una comunidad de desarrolladores inmensa a la que poder recurrir en caso de no conocer alguna funcionalidad o para resolver algún problema.
- El lenguaje que utiliza para el scripting (C#) es muy similar a C++, por lo que aprenderlo bien no me iba a resultar costoso.
- Consta de soporte para Realidad Aumentada y funciones de red, ambas cosas necesarias para el desarrollo del proyecto.

Por contra, durante la realización del proyecto he vivido la mala experiencia del uso de colisiones con modelos de Realidad Aumentada, ya que éstas dependen en alto grado de la calidad de la cámara y luminosidad del entorno para captar de forma correcta la interacción entre ambos objetos virtuales, por lo que en ocasiones el fallo es inevitable.

Capítulo 5

Análisis de requisitos software

A continuación procedemos con el **análisis de requisitos** software que no es más que el estudio de las necesidades de los usuarios con el fin de llegar a una especificación de requisitos que ha de cumplir el sistema para su correcto funcionamiento y para la satisfacción del usuario que lo utiliza. Entra dentro de este estudio el refinamiento de dichos requisitos.

Para la completar la finalidad de este capítulo comenzaré describiendo el problema a solventar, seguido de los requisitos (funcionales, no funcionales y de información) que ha de cumplir y terminando con los casos de uso (y sus actores) del software en objeto de estudio.

5.1. Descripción del software

El sistema que creado consiste en un videojuego cuyo objetivo básico es dirigir un tanque físico mediante ciertos gestos para llegar a una meta en el menor tiempo posible derrocando enemigos virtuales situados en el recorrido con armas virtuales que pueden recogerse por el mismo.

El usuario ha de poder controlar el tanque sin problemas con los 3 gestos establecidos para evitar obstáculos, así como recoger armas, matar enemigos y alcanzar la meta. Además, evidentemente, de ser capaz de seleccionar las distintas opciones del menú antes de comenzar el juego.

El sistema será capaz de mostrar una realimentación visual al usuario de:

- **Tiempo** total empleado.
- **Distancia** a un obstáculo físico.

- **Disparo** con el arma del tanque con cada gesto pertinente para ello.
- **Acciones** realizadas por el tanque como recoger un arma, tener un enemigo al alcance y eliminarlo.
- **Vida y energía** restante del tanque.
- **Vida** restante del enemigo.

Ahora que ya conocemos el funcionamiento del juego y las posibilidades del usuario con él, procedemos a formalizar todos los requerimientos de las tres partes que componen el software (el juego, el servidor y Arduino) separados en tres tipos: funcionales, no funcionales y de información.

5.2. Requisitos del sistema (juego)

5.2.1. Requisitos funcionales

Estas son las funcionalidades principales que el software ha de ofrecer al usuario de la aplicación, toda acción posible realizable por la persona que está jugando.

- **RF-1. Control del menú de inicio de juego.** El usuario podrá elegir la opción que desee en el menú de juego.
 - **RF-1.1. Jugar al juego.** El usuario tendrá la posibilidad de comenzar la partida tras leer las reglas del juego.
 - **RF-1.2. Opciones.** El usuario podrá modificar la configuración del juego a su gusto.
 - **RF-1.3. Salir del juego.** El usuario podrá salir del juego si lo desea.
- **RF-2. Jugar al juego.** El usuario tendrá la posibilidad de realizar todas las acciones necesarias para el desarrollo de una partida.
 - **RF-2.1. Conexión al servidor.** El juego se conectará de forma automática al servidor creado para el proyecto con el fin de recibir información para el correcto funcionamiento del juego.
 - **RF-2.2. Detección de patrones.** El sistema ha de ser capaz de detectar todos los patrones existentes en el juego.
 - **RF-2.2.1 Detección del tanque.** El jugador será capaz de, aproximándose al tanque, ver en pantalla su burbuja envolvente.

- **RF-2.2.2. Detección de armas.** El jugador será capaz de, aproximándose a un arma, verla en pantalla.
- **RF-2.2.3. Detección de enemigos.** El jugador será capaz de, aproximándose a un enemigo, verlo en pantalla.
- **RF-2.2.4. Detección de la meta.** El jugador será capaz de, aproximándose a la meta, detectarla para finalizar la partida.
- **RF-2.3. Resolución de colisiones.** El sistema ha de saber como actuar con cada una de las colisiones posibles durante la partida.
 - **RF-2.3.1. Colisión tanque-arma.** El software ha de realizar de forma automática la recolección de un arma por parte del tanque al colisionar. El arma desaparece.
 - **RF-2.3.2 Colisión tanque-enemigo.** El software ha de comenzar de forma inmediata el ataque del enemigo al tanque y la realimentación de la vida del enemigo al usuario cuando el tanque colisione con el enemigo.
 - **RF-2.3.3 Colisión tanque-meta.** El software ha de gestionar automáticamente la finalización de la partida al colisionar el tanque con la meta.
- **RF-2.4. Infingir daño a enemigo.** El jugador será capaz de restarle puntos de vida al enemigo una vez esté en contacto con él, tenga un arma en la mano(energía suficiente) y realice el gesto pertinente.
- **RF-2.5. Realimentación al usuario.** El sistema debe informar en todo momento de los sucesos que ocurren durante la partida de forma clara, con colores que no se acoplen con el fondo de la pantalla.
 - **RF-2.5.1. Realimentación HUD del tanque.** Cuando el usuario detecte el tanque con el dispositivo móvil, ha de aparecer en pantalla su barra de energía (que incrementará al recoger armas y decrementará al atacar a un enemigo) y de vida. Esta última decrementará de forma automática cuando se aproxime a un obstáculo físico y cuando le ataque un enemigo; incrementará cuando derrote al enemigo.
 - **RF-2.5.2. Realimentación de estado y acciones del tanque.** En todo momento se debe informar si el tanque se encuentra próximo a un obstáculo, si ha recogido un arma, si ha detectado un enemigo, si ha eliminado a un enemigo o si tiene suficiente energía como para infingir daño al enemigo.
 - **RF-2.5.3. Realimentación de disparo.** Cada vez que el jugador realice el gesto oportuno para disparar a un enemigo,

el sistema ha de hacer aparecer en pantalla la realimentación propia de disparo.

- **RF-2.5.4. Tiempo transcurrido.** El jugador ha de saber el tiempo que lleva transcurrido desde que comenzó a jugar.

- **RF-3. Control del menú de final de partida.**

- **RF-3.1 Volver a jugar.** El jugador puede volver a iniciar la misma partida si lo desea.
- **RF-3.2 Salir del juego.** El jugador puede salir del juego si lo desea.
- **RF-3.3 Gestión de finalización de partida por fracaso.** Cuando el enemigo derrote al tanque, la partida ha de terminar de forma automática, dando lugar a la pantalla "Game Over".
- **RF-3.4 Gestión de finalización de partida exitosa.** Cuando el tanque llega a la meta tras derrotar a todos los enemigos el sistema ha de hacer aparecer la pantalla de fin de partida con el tiempo total utilizado.

5.2.2. Requisitos no funcionales

Tipo de requisitos referido a aquellos que presentan funcionalidades del software no realizables de forma activa por el usuario, sino funcionalidades que el propio software ha de cumplir, así como restricciones a la hora de su uso.

- **RNF-1. Plataforma de uso.** El software deberá controlar que se pueda usar solo en dispositivos Android.
- **RNF-2. Conexión al servidor.** El juego mantendrá la conexión de forma estable durante toda la partida. Además, la conexión ha de ser rápida, eficiente y segura, manteniendo los principios básicos de seguridad de la información que son: Confidencialidad, Integridad, Disponibilidad de un sistema.
- **RNF-3. Realimentación al usuario legible.** La realimentación del sistema al usuario ha de ser legible de forma clara, con colores que no se acoplen con el fondo de la pantalla, por ejemplo.
- **RNF-4. Correcto funcionamiento.** El juego ha de funcionar de manera fluida sin cortes mientras se juega ni errores de ejecución.

5.2.3. Requisitos de información

Estos requisitos son los encargados de indicar la información útil que el software ha de almacenar.

- **RI-1 Tiempo de juego.** Con el fin de indicar al jugador el tiempo que ha empleado para pasarse la partida.
- **RI-2 Vida del tanque.** Para que el jugador conozca la vida que le queda.
- **RI-3 Energía del tanque.** Para que el jugador conozca la energía disponible que le queda para atacar a un enemigo.
- **RI-4 Vida del enemigo.** Para que el jugador conozca durante la batalla la vida que le queda a su enemigo.

5.3. Requisitos del sistema (Servidor)

5.3.1. Requisitos funcionales

- **RF-1. Ofrecer conexión.** Como servidor que es, ha de encargarse de hacer las veces de nexo entre el jugador, el juego y el tanque.
- **RF-2. Envíos de información necesaria.** Ha de ofrecer la posibilidad al jugador de enviar órdenes al tanque, a éste de enviar órdenes al juego.

5.3.2. Requisitos no funcionales

- **RNF-1. Correcto funcionamiento.** El sistema se encarga de gestionar los envíos de sus clientes (jugador, tanque y juego) de forma interna para que el sistema funcione de manera correcta sin ningún tipo de problemas.
- **RNF-2. Funcionamiento de la conexión.** La conexión ha de ser rápida, eficiente y segura, manteniendo los principios básicos de seguridad de la información que son: Confidencialidad, Integridad, Disponibilidad de un sistema.

5.3.3. Requisitos de información

- **RI-1. Almacenamiento de información recibida.** Toda la información recibida por parte de sus clientes ha de ser almacenada durante un corto plazo de tiempo para su posterior tratamiento.

5.4. Requisitos del sistema (Arduino)

5.4.1. Requisitos funcionales

- **RF-1. Ofrecer información de proximidad.** Ha de ser capaz de calcular y enviar información de proximidad al servidor.
- **RF-2. Movimiento de tanque.** Ha de ser capaz de gestionar las órdenes recibidas por el servidor para mover el tanque en función de estas.
- **RF-3. Ofrecer red de conexión.** Mediante el módulo WiFi instalado en Arduino, se ofrecerá una red para crear la conexión del sistema.

5.4.2. Requisitos no funcionales

- **RNF-1. Red de conexión.** La red de conexión ofrecida para el funcionamiento del sistema ha de ser completamente estable, sin fallo alguno.
- **RNF-2. Correcto funcionamiento** El sistema ha de funcionar de manera correcta, rápida y eficiente, por lo que ha de calcularse sin error la proximidad a obstáculos e interpretar de forma correcta las órdenes del servidor.

5.4.3. Requisitos de información

- **RI-1. Información recibida.** Se requiere el almacenamiento de la información recibida por parte del servidor durante un corto periodo de tiempo para su posterior tratamiento.

5.5. Casos de uso

Cada caso de uso representa la descripción de cada una de las acciones que pueden realizarse en el software. Las acciones son realizadas por un **actor** que interactúa con el software. Los casos de uso describen la secuencia de interacción entre el actor y el sistema para dar lugar a una de las funcionalidades de este último, así como sus estados de error, propósito y resumen. Los actores son elementos externos al software que interactúan con él para pedirle algo, y digo "elementos" porque un actor puede ser tanto una persona como un sistema externo al software. A continuación dispongo los casos de uso del proyecto:

| Caso de Uso | Iniciar juego desde el menú | CU_01 |
|---------------|--------------------------------|------------------------------|
| Actores | Jugador(principal) | |
| Tipo | Primario, extendido y real | |
| Referencias | RF-1 | |
| Precondición | Haber iniciado la aplicación. | |
| Postcondición | Juego ha comenzado la partida. | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha 28/08/2018 Versión 1.0 |

| |
|----------------------------------------------------------------------|
| Propósito |
| Dar lugar al comienzo de la partida con las reglas del juego claras. |

| |
|--------------------------------------------------------------------|
| Resumen |
| El jugador comienza el juego conociendo el funcionamiento de éste. |

| Curso Normal | |
|---------------------------------------------------------|-----------------------------------------------|
| Actor | Sistema |
| 1 El jugador pulsa el botón de PLAY | 2 Hace aparecer las instrucciones del juego |
| 3 Pulsa el botón LET'S GO tras leer las instrucciones | 4 Inicia el curso normal del juego en sí |

| |
|------------------------------------------------|
| Cursos Alternos |
| 4a No se produce la conexión con el servidor |
| 1 El juego no se ejecuta correctamente |

Figura 5.1: Diagrama CU_01: Iniciar juego

| Caso de Uso | Detección de patrón | | | CU_02 |
|---------------|------------------------------------------------------------------------------------------------|-------|------------|-------------|
| Actores | Jugador(principal) | | | |
| Tipo | Primario, extendido y real | | | |
| Referencias | RF-2.2 | | | |
| Precondición | El jugador ha de apuntar con la cámara del dispositivo a un patrón reconocible por el sistema. | | | |
| Postcondición | Aparecerá en pantalla un modelo 3D situado encima del patrón. | | | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha | 28/08/2018 | Versión 1.0 |

| Propósito |
|------------------------------------------------------------|
| Ver en la pantalla del dispositivo un modelo 3D del juego. |

| Resumen |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| El jugador, al apuntar con la cámara a un patrón, aparecerá en pantalla un enemigo, la burbuja englobante del tanque, un arma o la gema que indica la meta final. |

| Curso Normal | |
|-------------------------------------|-------------------------------------------------------------------------------|
| Actor | Sistema |
| 1 Apunta con la cámara a un patrón. | 2 Hace mostrar en pantalla el modelo y/o información asociada a dicho patrón. |

| Cursos Alternos | |
|-----------------|-------------------------------------------------------------|
| 1a | Apunta a un patrón no conocido por el sistema. |
| 1 | El sistema no hace nada. |
| 2a | Poca luminosidad en el ambiente o baja calidad de la cámara |
| 1 | El sistema no hace nada. |

Figura 5.2: Diagrama CU_02: Detección de patrón

| Caso de Uso | Resolución de colisiones | CU_03 |
|---------------|----------------------------------------------------------------------|------------------------------|
| Actores | Jugador(principal) | |
| Tipo | Primario, extendido y real | |
| Referencias | RF-2.3 | |
| Precondición | Existen en pantalla dos o más modelos 3D reconocidos por el sistema. | |
| Postcondición | Se produce la acción pertinente a la colisión. | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha 28/08/2018 Versión 1.0 |

| Propósito |
|----------------------------------------------------------------------------|
| Generar un comportamiento en el juego en función de la colisión producida. |

| Resumen |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| El jugador hace colisionar el tanque con un arma, un enemigo o la meta, lo que producirá un aumento de energía, pérdida de vida por parte del tanque y realimentación de la vida del enemigo o el fin del juego (respectivamente). |

| Curso Normal | |
|---------------------------------------|--------------------------------------------------------------------------------------|
| Actor | Sistema |
| 1a Produce la colisión tanque-arma | 2a Aumenta la vida del tanque |
| 1b Produce la colisión tanque-enemigo | 2b Comienza a disminuir la vida del tanque y aparece la vida del enemigo en pantalla |
| 1c Produce la colisión tanque-meta | 2c Finaliza el juego |

| Cursos Alternos | |
|-----------------|----------------------------------------------------------------------------------------|
| 1a | No se detecta bien la colisión |
| 1 | El sistema no hace nada |
| 1b.a | El sistema deja de detectar la colisión con el enemigo |
| 1 | El tanque deja de perder vida y desaparece la barra de vida del enemigo de la pantalla |
| 1c.a | El jugador no ha eliminado a todos los enemigos de la partida |
| 1 | El sistema no hace nada. |

Figura 5.3: Diagrama CU_03: Resolución de colisiones

| | | |
|---------------|--------------------------------------------|------------------------------|
| Caso de Uso | Inflingir daño a enemigo | CU_04 |
| Actores | Jugador(principal) | |
| Tipo | Primario, extendido y real | |
| Referencias | RF-2.4 | |
| Precondición | El tanque está en contacto con el enemigo. | |
| Postcondición | La vida del enemigo disminuye. | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha 28/08/2018 Versión 1.0 |

| |
|-----------------------------------------------------|
| Propósito |
| Disminuir la vida del enemigo por parte del tanque. |

| |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Resumen |
| El jugador hace el gesto de atacar y el tanque infinge cierta cantidad de daño al enemigo. Se visualiza mediante realimentación de la barra de vida del enemigo. |

| Curso Normal | |
|------------------------|--------------------------------------------------------------------------------|
| Actor | Sistema |
| 1 Hace gesto de ataque | 2 Disminuye la vida del enemigo |
| | 3 Disminuye la energía del tanque |
| | 4 Informa al jugador de que está atacando mediante realimentación por pantalla |

| | |
|-----------------|-------------------------------------------------------------------|
| Cursos Alternos | |
| 1a | No hace bien el gesto o no lo reconoce MYO |
| 1 | El sistema no hace nada |
| 1b | No existe conexión entre el juego y el servidor |
| 1 | El sistema no hace nada |
| 2a | La vida del enemigo es inferior a 0 |
| 1 | Desaparece el enemigo de la pantalla y aumenta la vida del tanque |
| 2b | El tanque no tiene energía |
| 1 | El sistema avisa al jugador y no hace nada |

Figura 5.4: Diagrama CU_04: Inflingir daño a enemigo

| | | |
|---------------|----------------------------------------------------------------------|------------------------------------|
| Caso de Uso | Ofrecer conexión | CU_05 |
| Actores | Juego(principal), Arduino(principal) | |
| Tipo | Primario, extendido y real | |
| Referencias | RF-1 (Servidor) | |
| Precondición | Ha de existir una red WiFi donde ofrecer la conexión | |
| Postcondición | Los actores son capaces de enviar y recibir información del servidor | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha 28/08/2018 Versión 1.0 |

| |
|------------------------------------------------|
| Propósito |
| Ofrecer un canal de comunicación a los actores |

| |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Resumen |
| El programa crea el servidor para comunicarse con los actores. Crea un sockets para la comunicación con los actores, con sus respectivos canales de entrada de información y salida de la misma. |

| Curso Normal | |
|--------------------------------|---------------------------------------------------------------------------|
| Actor | Sistema |
| | 1 Crea servidor con un puerto y una IP |
| | 2 Permanece a la escucha de peticiones de conexión |
| 3 Realiza petición de conexión | 4 Crea los flujos de entrada y salida de datos del socket correspondiente |
| | 5 Lanza una hebra para la gestión de dicha conexión |

| | |
|-----------------|-----------------------------------------------------------|
| Cursos Alternos | |
| 4a | Interferencias en la red |
| 1 | No se detecta la petición de conexión por parte del actor |
| 4b | Se produce una desconexión por parte del actor |
| 1 | Tratamiento de errores |
| 5a | Se produce una desconexión por parte del actor |
| 1 | Tratamiento de errores |

Figura 5.5: Diagrama CU_05: Ofrecer conexión

| | | |
|---------------|--------------------------------------------------------------------------------------------------------|------------------------------------|
| Caso de Uso | Enviar/recibir información necesaria | CU_06 |
| Actores | Juego(principal), Arduino(principal), MYO Armband(principal) | |
| Tipo | Primario, extendido y real | |
| Referencias | RF-2 (servidor) | |
| Precondición | Existe conexión entre ambos host y hay información disponible para enviar por parte de alguno de ellos | |
| Postcondición | El host receptor ha recibido su paquete de información | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha 28/08/2018 Versión 1.0 |

Propósito

La información enviada por uno de los host ha de ser recibida de manera correcta por el otro host de la conexión

Resumen

El servidor envía/recibe información o una orden a/de un host cliente sin ningún tipo de problemas.

Curso Normal

| | Actor | Sistema |
|----|----------------------------------------------|---------------------------------------------------|
| 1a | Espera la recepción de información/órdenes | 2a Envía información o una orden. |
| 3a | La recibe sin modificaciones en el contenido | |
| | | 1b Espera la recepción de información o una orden |
| 2b | Envía información o una orden. | 3b La recibe sin modificaciones en el contenido |

Cursos Alternos

| | |
|------|-------------------------------------|
| 2a.a | No existe conexión por algún motivo |
| | Tratamiento de errores |
| 1b.a | No existe conexión por algún motivo |
| | Tratamiento de errores |

Figura 5.6: Diagrama CU_06: Enviar/recibir información

| | | | | | | |
|----------------------|------------------------------------------------------------|--------------|------------|----------------|--|--|
| Caso de Uso | Realimentación de juego | | | CU_07 | | |
| Actores | Jugador(principal) | | | | | |
| Tipo | Primario, extendido y real | | | | | |
| Referencias | RF-2.5 | | | | | |
| Precondición | Ha ocurrido algún suceso por parte del tanque o el jugador | | | | | |
| Postcondición | Se muestra la realimentación en pantalla | | | | | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha | 28/08/2018 | Versión | | |
| | | | | 1.0 | | |

Propósito

Informar al usuario del estado del tanque en cada momento

Resumen

Si el usuario realiza un ataque ha de ser realimentado de que se está cumpliendo. Si el tanque se aproxima a un obstáculo demasiado, recoge un arma o entra en contacto con el enemigo el jugador ha de ser informado.

Curso Normal

| Actor | | Sistema | |
|--------------|--------------------------------|----------------|----------------------------------|
| 1a | Tanque se aproxima a obstáculo | 2a | Informa de la proximidad |
| 1b | Tanque colisiona con elemento | 2b | Informa de colisión con elemento |
| 1c | Tanque ataca sin energía | 2c | Informa de la falta de energía |
| 1d | Tanque escapa de enemigo | 2d | Informa de haber escapado |
| 1e | Jugador ataca | 2e | Informa del ataque realizado |

Cursos Alternos

| | |
|------|---------------------------------------------|
| 2b.a | No se detecta la colisión de forma correcta |
| 1 | El sistema no hace nada |
| 2e.a | No se detecta el gesto de ataque correcto |
| 1 | No se hace nada |

Figura 5.7: Diagrama CU_07: Realimentación del juego

| | | |
|---------------|---------------------------------------------------------------------------|------------------------------------|
| Caso de Uso | Ofrecer información sobre proximidad | CU_08 |
| Actores | Jugador(principal) | |
| Tipo | Primario, extendido y real | |
| Referencias | RF-1 (Arduino) | |
| Precondición | El sistema de reconocimiento de proximidad funciona correctamente | |
| Postcondición | El jugador dispone en pantalla de la proximidad del tanque a un obstáculo | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha 28/08/2018 Versión 1.0 |

| |
|------------------------------------------------------|
| Propósito |
| Calcular y enviar información de proximidad al juego |

| |
|----------------------------------------------------------------------------------------------------------------------|
| Resumen |
| Arduino calcula la proximidad del tanque a un obstáculo de forma reiterada y la envía al juego a través del servidor |

| Curso Normal | | | |
|--------------|----------------------------------------------------|---------|----------------------------------------------------|
| | Actor | Sistema | |
| | | 1 | Calcula proximidad del tanque a un obstáculo |
| | | 2 | Envía la distancia al juego a través del servidor. |
| 3 | El juego recibe la distancia a través del servidor | | |
| 4 | La muestra en el HUD | | |

| | |
|-----------------|--------------------------------------------------------------|
| Cursos Alternos | |
| 2a | No existe conexión efectiva en la comunicación con el juego. |
| 1 | Tratamiento de errores en la comunicación |
| | |

Figura 5.8: Diagrama CU_08: Información de proximidad

| | | |
|---------------|---------------------------------------------------|------------------------------|
| Caso de Uso | Movimiento tanque | CU_09 |
| Actores | Jugador(principal) | |
| Tipo | Primario, extendido y real | |
| Referencias | RF-2 (Arduino) | |
| Precondición | Existe conexión efectiva entre jugador y tanque | |
| Postcondición | El tanque se mueve hacia donde indique el jugador | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha 28/08/2018 Versión 1.0 |

| |
|--------------------------------------------------|
| Propósito |
| Mover el tanque mediante las ordenes del jugador |

| |
|----------------------------------------------------------------------|
| Resumen |
| El jugador indica una orden de movimiento del tanque y este se mueve |

| Curso Normal | | |
|--------------|------------------|-------------------------------------------------------------------------|
| | Actor | Sistema |
| 1 | Realiza un gesto | 2 Interpreta el gesto |
| | | 3 Ordena a los motores moverse hacia donde indique dicha interpretación |

| Cursos Alternos | |
|-----------------|-----------------------------------------------------------------------|
| 2a | No se recibe el gesto o no se recibe el correcto debido a la conexión |
| 1 | El sistema no hace nada |
| | |

Figura 5.9: Diagrama CU_09: Movimiento del tanque

| | | |
|----------------------|---------------------------------------------------------------------|------------------------------------|
| Caso de Uso | Red WiFi | CU_10 |
| Actores | Juego(principal), Servidor(principal) | |
| Tipo | Primario, extendido y real | |
| Referencias | RF-3(Arduino) | |
| Precondición | Funciona de forma correcta el módulo WiFi | |
| Postcondición | Se crea una red para la conexión de todos los sistemas del proyecto | |
| Autor | Carlos Manuel Sequí Sánchez | Fecha 28/08/2018 Versión 1.0 |

Propósito

Crear red donde realizar una conexión para gestionar el contacto entre los sistemas del proyecto

Resumen

El módulo realiza la inicialización pertinente y crea la red.

Curso Normal

| | Actor | Sistema |
|---|---------------------|----------------------------------|
| | | 1 Inicialización del módulo WiFi |
| 2 | Servidor se conecta | |
| 3 | Juego se conecta | |

Cursos Alternos

2a No funciona de manera correcta el módulo WiFi

1 Vuelve al paso 1

3a No funciona de manera correcta el módulo WiFi

1 Vuelve al paso 1

Figura 5.10: Diagrama CU_10: Red WiFi

5.6. Diagramas de clases

Por último, como parte del análisis del software llevado a cabo, procedo a mostrar los diagramas de clases pertenecientes al código desarrollado, los cuales nos mostrarán las interacciones entre las distintas clases del proyecto, así como sus atributos y métodos.

Primeramente podemos ver el diagrama de clases de Vuforia, la librería de Realidad Aumentada utilizada para el juego.

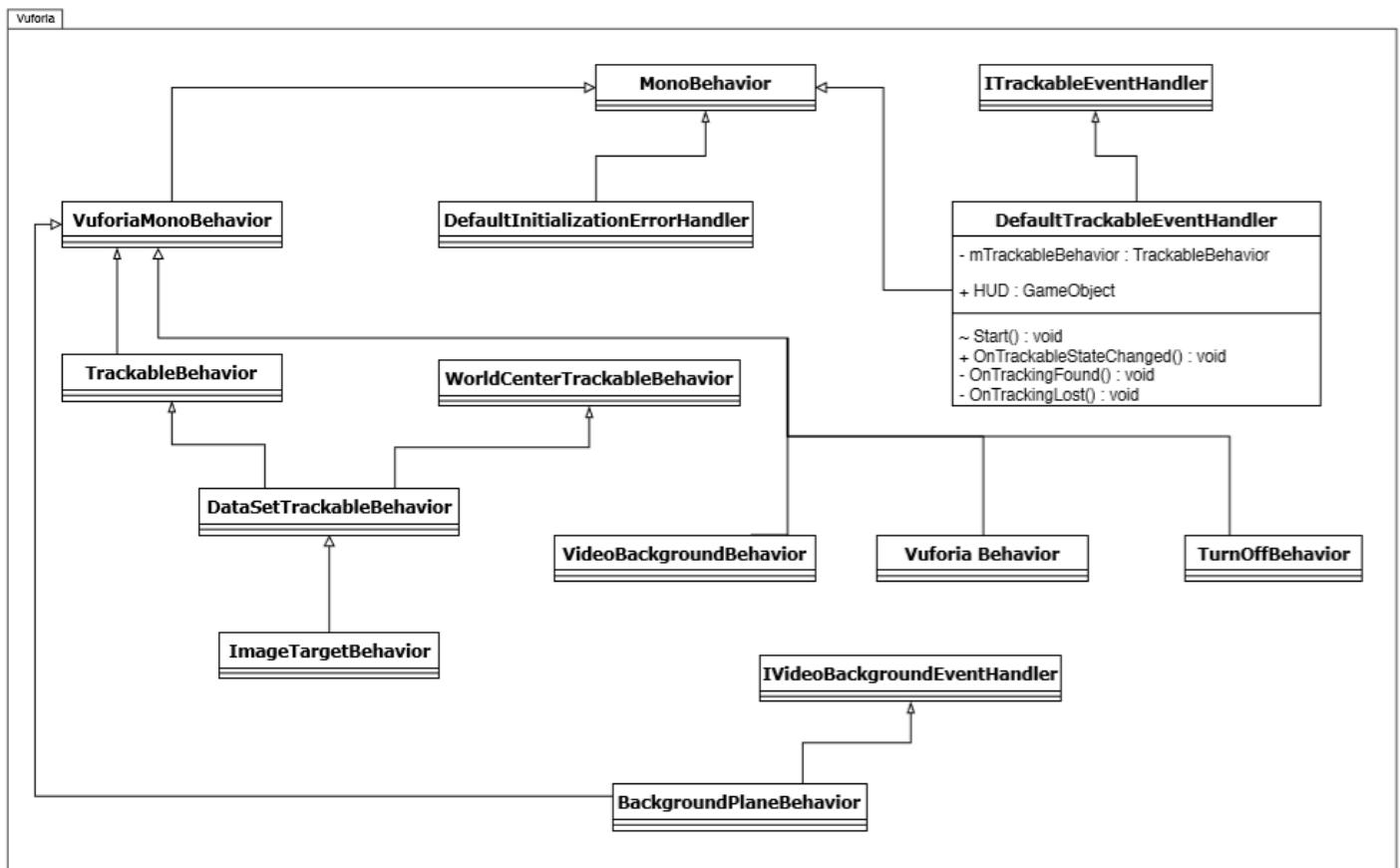


Figura 5.11: Diagrama de clases Vuforia

En este primer diagrama podemos observar la herencia llevada a cabo por Vuforia. Quedan indicados únicamente los atributos y métodos de la única clase que he tenido que modificar para mi software: `DefaultTrackableEventHandler`.

A continuación procedo a mostrar los diagramas de clases de la aplicación dividida por escenas de Unity. Esta es la forma de trabajar de este IDE de desarrollo de videojuegos, las escenas, en cada una hay un conjunto de objetos y funciones asociadas a esos objetos distintas al resto de escenas. Las clases que no interactúan de forma aparente en el diagrama con el resto de clases es porque en Unity esas clases son utilizadas internamente por objetos para llevar a cabo su funcionalidad.

1. Menú principal

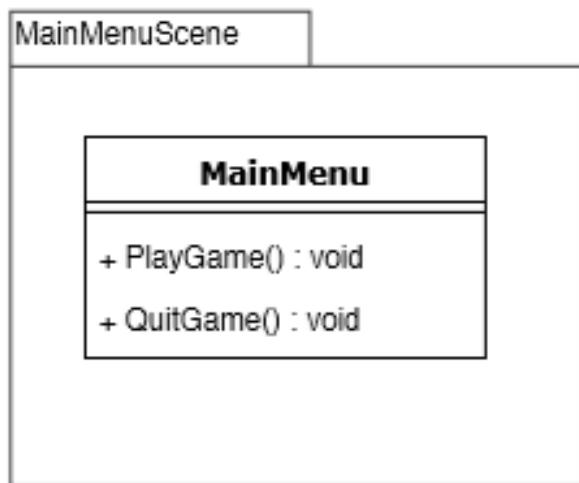


Figura 5.12: Diagrama de clases escena menú principal del juego

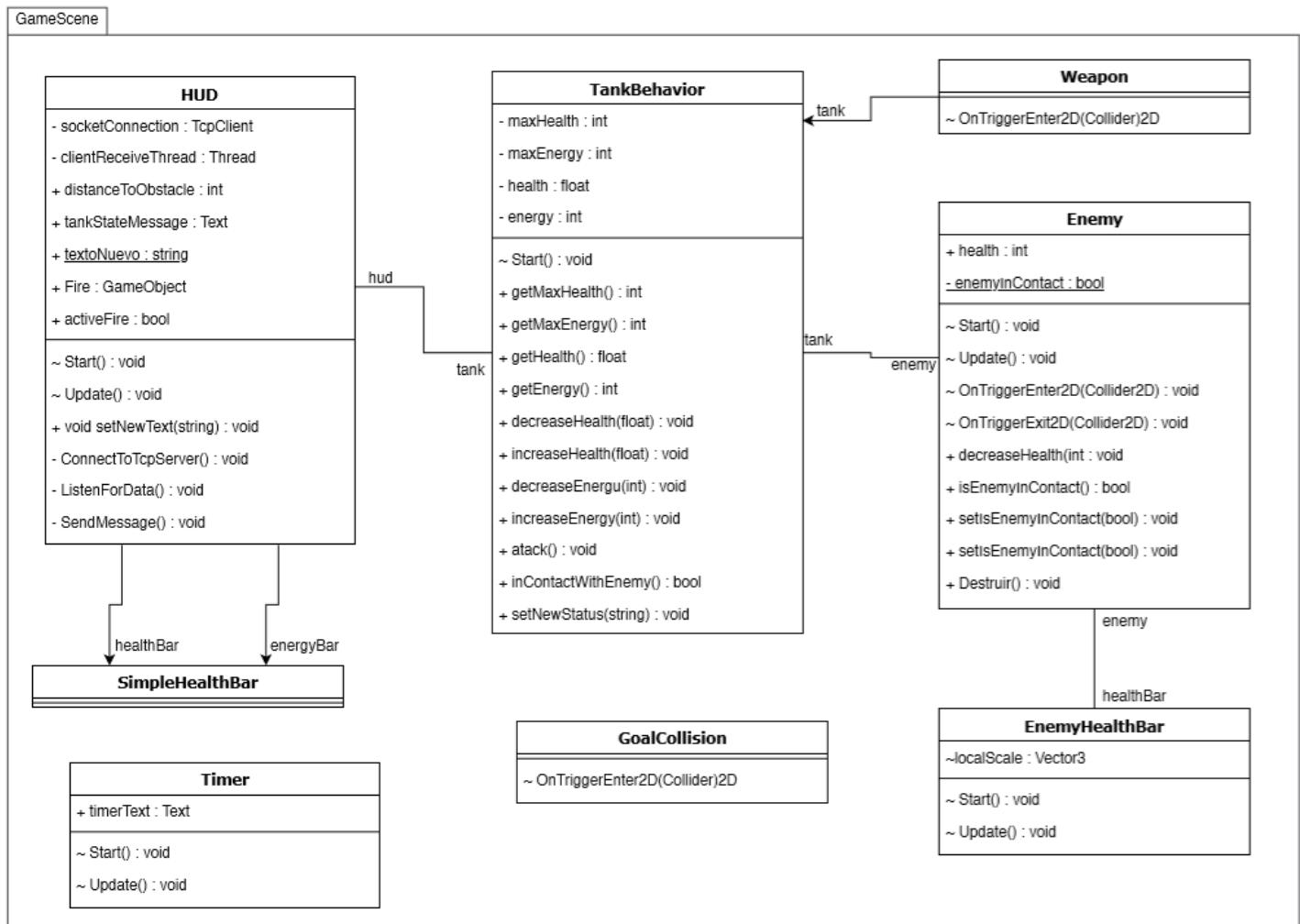


Figura 5.13: Diagrama de clases escena de juego

3. **Meta alcanzada.** Cuando se alcanza la meta habiendo eliminado a los enemigos se gestiona una pantalla de finalización de partida que muestra el tiempo empleado para terminarla.

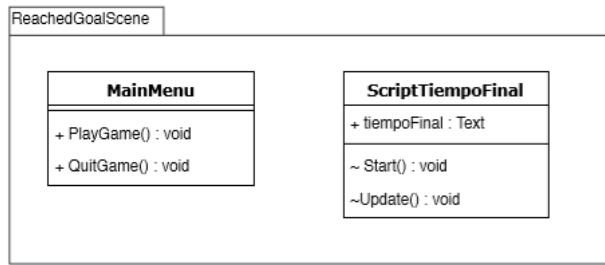


Figura 5.14: Diagrama de clases escena de partida finalizada exitosamente

4. **Game Over**. Ocurre al ser eliminado por un enemigo.

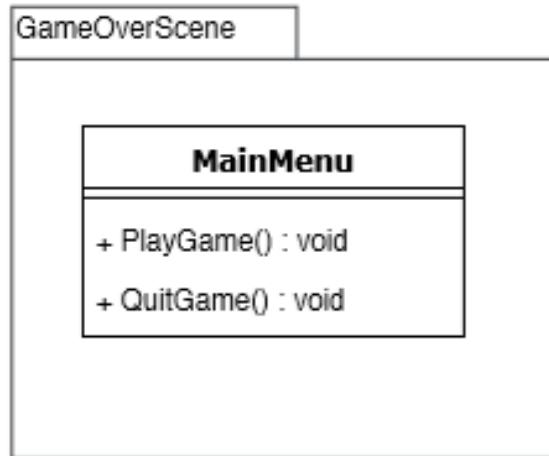


Figura 5.15: Diagrama de clases escena de Game Over

Una vez mostrados los diagramas de clases del juego en sí, mostramos a continuación el del servidor y el de la programación de la placa Arduino:

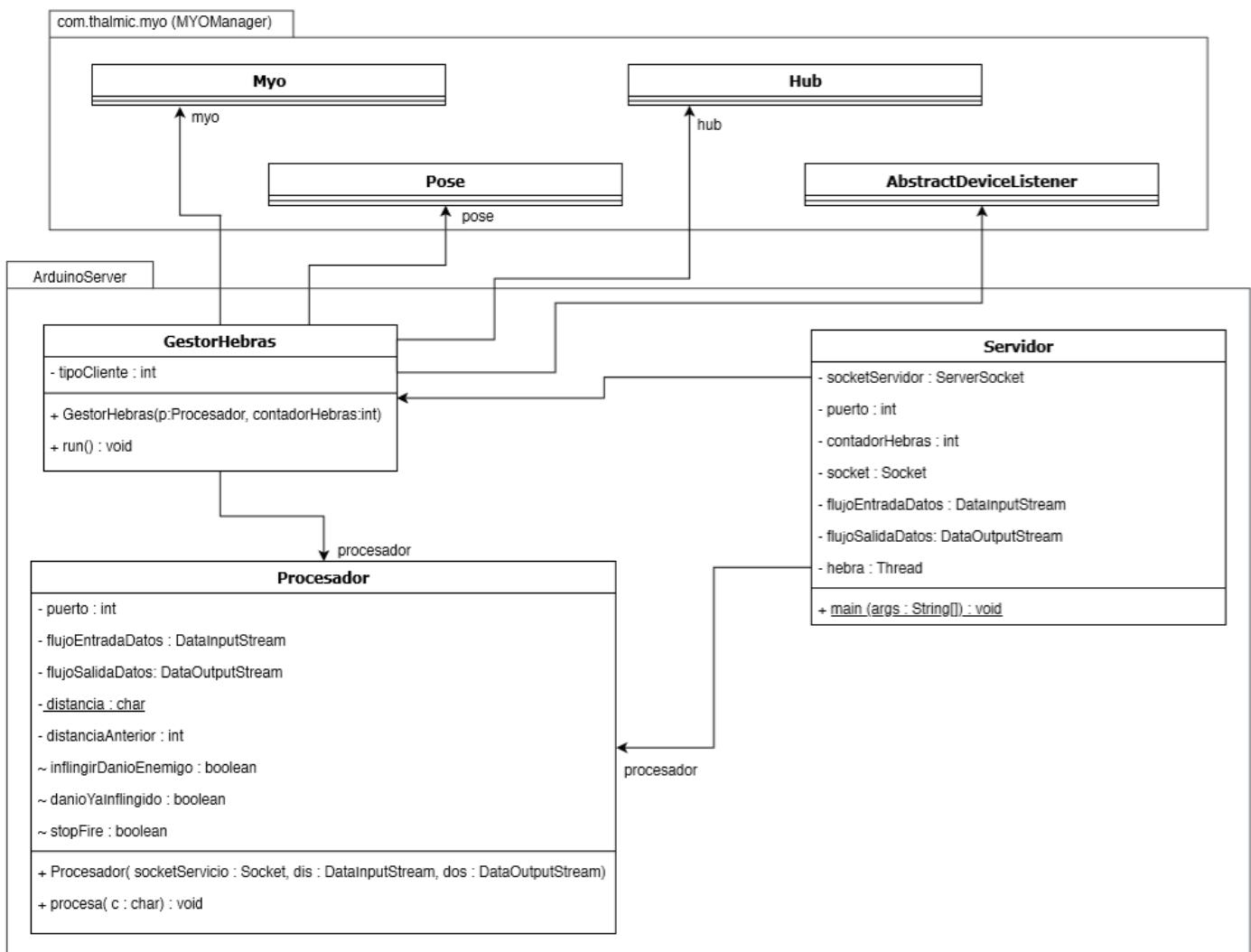


Figura 5.16: Diagrama de clases del servidor

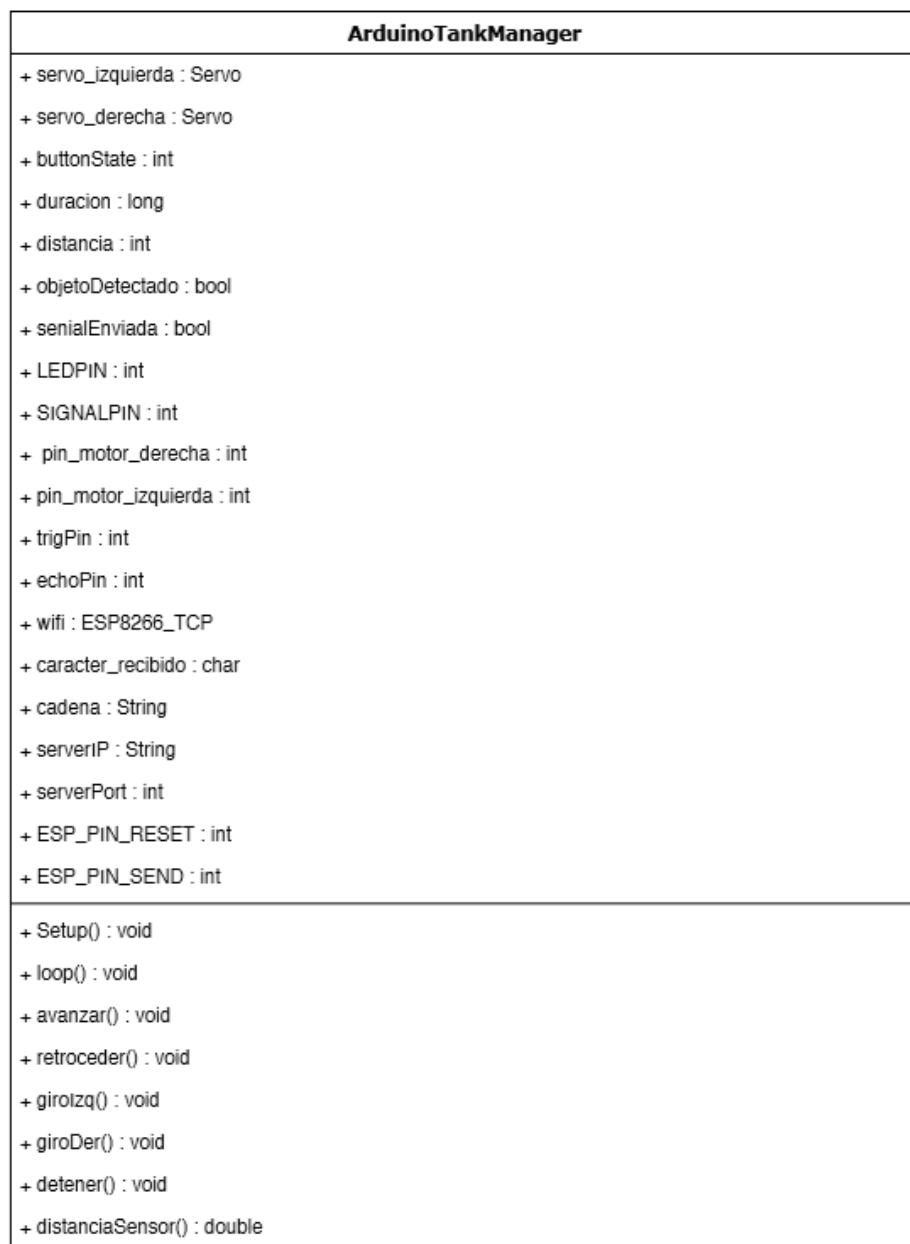


Figura 5.17: Diagrama de clases del gestor del tanque (programación Arduino)

Capítulo 6

Implementación

El objetivo de este capítulo es describir las clases y métodos más relevantes del proyecto a un nivel inferior de lo hecho hasta el momento. Comenzaré con la descripción del desarrollo del propio juego en sí, continuando con la implementación del servidor y finalizando con la realización del código que ejecuta la placa Arduino.

6.1. Desarrollo del juego

El juego, como ya he comentado en varias ocasiones, ha sido implementado al completo en Unity3D, el cual, como también he mencionado en otras tantas ocasiones, trabaja con escenas compuestas de objetos cuyo comportamiento puede ser implementado tanto en el editor del IDE, como con scripts escritos en el lenguaje de programación C#. Es por ello que voy a dividir esta sección en tantas como escenas disponga mi aplicación. Además, seguiré el orden en el que han sido desarrolladas tal como indico en los sprints realizados de la sección 1.2.2 de esta documentación.

6.1.1. Escena 1: Menú principal

En esta escena, como vemos en la siguiente figura, se muestran tres botones con los que interaccionar:

- **Play:** nos muestra la siguiente pantalla, que es una guía rápida de cómo jugar al juego. A continuación comenzará el juego.
- **Options:** para cambiar configuración del juego en un futuro, tal como sonido del juego o dificultad, por ejemplo.
- **Quit:** nos permite salir de la aplicación.

Para la implementación de esta escena simplemente he creado un canvas (lienzo) sobre el que añadir gameObjects (objetos) como lo son la imagen de fondo y los botones para que puedan ser visibles en la aplicación, es decir, sin canvas no aparecerían en pantalla.

El diseño de los botones ha sido creado totalmente en el editor de Unity, es decir, no he necesitado hacer uso de scripting para ello, simplemente he agregado propiedades de color y realimentación visual al usuario desde el propio IDE. Sin embargo, para dotarles de funcionalidad, he creado y añadido a cada botón un script que contiene implementada la siguiente clase:

- **MainMenu**: la cual consta de dos métodos simples:
 - **PlayGame()**: que nos permitirá pasar a la siguiente escena del juego.
 - **QuitGame()**: que nos ofrece la posibilidad de salir de la aplicación.

6.1.2. Escena 2: El juego

En esta escena se encuentra el grueso del desarrollo software del juego. Sus principales componentes son:

- **DefaultTrackableEventHandler**: encargado de la detección de patrones para hacer efectiva la Realidad Aumentada.
- **HUD** (Head-Up Display): que nos muestra información en pantalla sobre la vida y energía del tanque, realimentación sobre su estado actual (proximidad a obstáculos, detección de enemigos y armas, etc.), realimentación de disparo y el tiempo transcurrido desde el comienzo de la partida. Además se encarga de la recepción de información por parte del servidor mediante una conexión TCP.
- **TankBehavior**: que, como su nombre indica, se encarga de dotar de comportamiento al tanque.
- **Weapon**: encargada de asignar comportamiento a las armas.
- **Enemy**: para dotar de comportamiento al enemigo.
- **GoalCollision**: para la finalización exitosa del juego.

Una vez descritas las principales clases que componen la presente escena, procedo a profundizar en cada una de ellas.

DefaultTrackableEventHandler

Como ya he comentado, es una clase perteneciente a la librería de Vuforia para la implementación de Realidad Aumentada.

En ella, además, he implementado la funcionalidad de hacer aparecer el HUD en cuanto se detecte al tanque con el dispositivo móvil, es decir, se encarga de activar y desactivar tanto el HUD como la bola que envuelve al tanque cuando entra y sale del campo de visión de la cámara del teléfono móvil.

Métodos relevantes:

- **OnTrackingFound()**: gestiona la **aparición** de un modelo 3D sobre el patrón detectado por la cámara.
- **OnTrackingLost()**: gestiona la **desaparición** de un modelo 3D cuando la cámara deja de detectar dicho patrón.

HUD

Se trata de la clase principal del juego, ya que como he descrito anteriormente, se encarga de la comunicación con el servidor mediante conexión TCP, sin lo que no sería posible recibir la información necesaria para el desarrollo del juego, dando lugar tanto a la falta de realimentación al usuario de los sucesos que ocurren durante la partida, como al impedimento de restarle vida al tanque debido a la proximidad a un obstáculo o restarle vida a un enemigo al recibir la orden de disparo mediante el gesto correspondiente.

Métodos relevantes:

- **ConnectToTcpServer**: realiza la petición de conexión del cliente (el juego) al servidor. Este método es lanzado nada más comenzar la ejecución del juego.
- **ListenForData**: se encarga de conectarse al socket creado por el servidor mediante su dirección ip y el puerto correspondientes para abrir la vía de comunicación entre ambos. Además, como su nombre indica, permanece a la escucha del canal de comunicación en espera de información enviada por el servidor. Cualquier tipo de información que reciba por parte de este, es gestionada para realizar las funciones correspondientes a las órdenes recibidas (ataque del tanque a un enemigo, informar al usuario de la proximidad del tanque a un obstáculo e informar al usuario de que está atacando mediante realimentación visual).

TankBehavior

Esta clase nos permite modificar y acceder a la vida y energía del tanque en caso de que recoja un arma o esté siendo atacado por un enemigo por ejemplo, así como atacar al enemigo y conocer su estado con respecto a estos y las armas (si ha entrado en contacto con alguno de ellos).

Métodos relevantes:

- **attack()**: infinge daño al enemigo y hace disminuir su propia energía.
- **setNewStatus(string status)**: comunica al HUD su estado actual (arma recogida, enemigo detectado...) para la realimentación al usuario.

Weapon

La funcionalidad de esta clase es muy sencilla: al entrar en contacto con el tanque (OnTriggerEnter2D), ejecuta las sentencias necesarias para proporcionarle energía y a su vez desaparecer de la pantalla del dispositivo móvil, haciendo que no sea posible su reaparición al ser detectado el patrón sobre el que reposa dicho modelo 3D. Además notifica al tanque el suceso de haber colisionado ambos modelos para que lo haga saber al usuario mediante el HUD.

Este script se asocia en el editor de Unity a cada una de las armas existentes como modelos 3D que tienen asociado un patrón de Realidad Aumentada.

Enemy

Para controlar el comportamiento del enemigo disponemos de esta clase. Gestiona los eventos de entrar en contacto y salir del contacto con el tanque, así como el de recibir daño por parte de este y atacarle. Tiene asociado otro script (otra clase: EnemyHealthBar) que se encarga de la gestión visual de la salud del enemigo para que el jugador pueda ver su vida restante.

Métodos relevantes

- **Update()**: en caso de que estén en contacto enemigo y tanque, el enemigo infinge daño al tanque de forma continuada hasta perder el contacto o terminar con su vida, en cuyo caso gestiona la finalización de la escena actual para dar paso a la escena GameOver.
- **OnTriggerEnter2D()**: se asigna de forma automática como enemigo del tanque (para que este pueda hacer uso de sus métodos), le notifica

de la colisión con él y activa la barra de vida del enemigo mediante la clase mencionada en la descripción de esta clase.

- **OnTriggerExit2D()**: notifica al tanque el haber escapado del enemigo, hace desaparecer la barra de vida de la vista del jugador y, mediante el mismo booleano que permite al método Update() inflingir daño al tanque, se hace que el enemigo ya no tenga al alcance al tanque para seguir atacándole.
- **decreaseHealth**: permite al tanque atacar al enemigo disminuyendo su vida.
- **increaseTankHealthOnDeath**: al morir el enemigo el tanque gana una dosis de vida extra.
- **Destruir()**: para hacer desaparecer de la pantalla el gameObject correspondiente al enemigo.

Este script se asocia en el editor de Unity a cada uno de los enemigos existentes como modelos 3D que tienen asociado un patrón de Realidad Aumentada.

Clase EnemyHealthBar

controla el gameObject correspondiente a la barra visual de vida del enemigo y, cuando su salud es inferior que cero, hace desaparecer al enemigo de la pantalla e incrementa la vida del tanque mediante los métodos descritos.

GoalCollision

En cuanto el tanque recoge la gema, que es el objetivo de la partida, esta clase se encarga simplemente de gestionar la finalización de la escena actual para dar paso a la escena de finalización exitosa de partida.

6.1.3. Escena 3: Game Over

Como ya he comentado, cuando el enemigo termina con la vida del tanque, automáticamente se procede a realizar el cambio de la escena 2 a la escena de Game Over. Esta escena simplemente tiene asociado el mismo script que el menú principal, solo que esta vez, dándole al jugador la única posibilidad de salir del juego.

6.1.4. Escena 4: Finalización exitosa de la partida

Cuando el tanque recoge la gema objetivo de la partida, entonces se pasa de la escena de juego a esta, la cual muestra en pantalla el tiempo empleado desde el comienzo de la partida hasta la recogida de la gema (utilizando la clase `scriptTiempoFinal`). Para ello hace uso de una clase estática que permite compartir valores de variables entre escenas distintas(`scriptDatosFinalJuego`), y a la que podemos acceder desde cualquier clase del juego. Sin esta clase no sería posible mostrar el tiempo de juego empleado.

6.2. Implementación del servidor

A continuación procedo a describir el desarrollo y funcionamiento del código que hace las veces de servidor en el proyecto.

Como ya comenté en su momento, esta parte del proyecto ha sido desarrollada en el IDE Netbeans mediante el lenguaje de programación Java.

Aquí, además del código para la gestión de las órdenes de MYO, creación del servidor y envío y recepción de información de clientes, es donde se incorpora el código que hace posible la detección de los gestos de MYO, el cual he tomado de una librería existente entre los repositorios públicos de la web GitHub [37].

Antes de nada quiero explicar en qué consiste una conexión mediante el protocolo TCP/IP.

6.2.1. TCP/IP

El Transmission Control Protocol o Internet Protocol es, como su nombre indica, un **protocolo de la capa de transporte** en el modelo OSI(Open System Interconnection) que, en una conexión en la que existe un flujo de datos, nos garantiza que estos serán entregados de forma segura al destino correspondiente sin errores y en el orden en el que hayan sido transmitidos. Además permite distinguir la aplicación a la que enviar los datos dentro de una máquina gracias al concepto de "puerto".

Posee la característica de ofrecer transmisión "full-duplex", por lo que la comunicación puede ser en ambos sentidos y de forma paralela. Además tiene control de flujo y errores, de conexión y de congestión.

Es un protocolo orientado a conexión, lo que nos indica que cliente y servidor han de solicitar y aceptar la conexión antes de comenzar con la transmisión de datos entre ellos mediante el denominado "three way handshake", explicado a continuación:

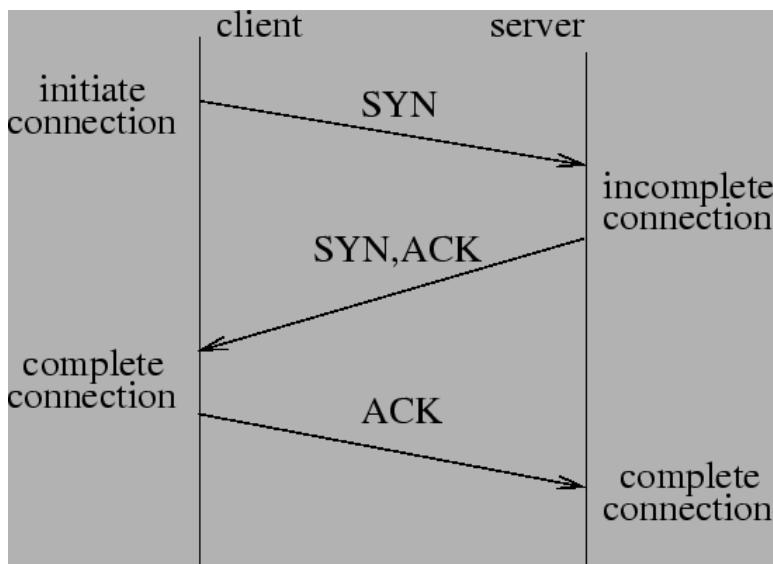


Figura 6.1: Three way handshake[28]

- Solicitud de conexión:** El host cliente envía un bit SYN al servidor para solicitar la conexión con un número de secuencia determinado.
- Aceptación de conexión y solicitud en sentido inverso:** el servidor envía una confirmación de recepción ACK y un segmento TCP de vuelta al host cliente en el que el número de secuencia recibido lo incrementa en una unidad (hace acuse). Además, al ser full-duplex, el servidor también realiza una petición de conexión con un bit SYN junto a otro número de secuencia.
- Conexión establecida:** el host cliente (el inicial) hace acuse del número de secuencia recibido por el servidor y notifica a este con otro bit ACK del establecimiento de la conexión.

Una vez entendido el funcionamiento del protocolo TCP procedo a explicar las tres principales clases del código desarrollado: Servidor, GestorHebras y Procesador.

6.2.2. Servidor

En esta primera clase se implementa el código para la creación del servidor en sí. La ejecución normal del servidor creará primeramente una hebra paralela para la conexión con el cliente 1 (tanque) y otra hebra en segundo lugar para la conexión con el cliente 2 (el juego en Unity). A continuación describo los pasos a seguir para la creación de ambas hebras.

1. Primeramente se crea un ServerSocket que nos servirá para, mediante el método accept(), esperar la petición de conexión de un cliente.
2. Una vez aceptada una petición de conexión del cliente se crearán los flujos de entrada y salida de datos del socket para el envío y recepción de información con ese cliente.
3. Creamos un procesador de información (de la clase Procesador) con dichos flujos de entrada/salida de datos.
4. Creamos la hebra para enviarla a la clase GestorHebras utilizando el procesador de información recientemente creado y, que esta se encargue del funcionamiento de cada una de las distintas hebras.

6.2.3. GestorHebras

Aquí es donde, dependiendo de la hebra que se ejecute, se ejecutarán unas funcionalidades u otras:

- **Hebra 1: conexión con MYO y Arduino.** Se encargará de crear un listener activo para recibir los distintos gestos del dispositivo MYO y llamar al procesador de órdenes (de la clase Procesador) para ejecutar la orden correspondiente. De este modo, aquí es donde se tienen en cuenta los gestos que provocan las acciones como el avance del tanque, giro a ambos lados y disparo (también se controla la realimentación de disparo en el juego desde aquí). Además, esta hebra es la encargada también de, en caso de que el flujo de entrada contenga información, ordenar al servidor que realice la lectura de la información de distancia que el tanque envía al servidor.
- **Hebra 2: conexión con el juego.** Esta hebra será la encargada tan sólo de comunicar al procesador que realice las acciones pertinentes al envío de información al juego.

6.2.4. Procesador

Esta es la clase que entra en contacto directo con los clientes, tanto para recibir datos de proximidad del tanque como para enviar información al juego. Dependiendo del parámetro que cada una de las hebras haya utilizado para llamar al método principal **procesa(char parametro)** de la clase Procesador, se realizará una u otra acción:

- Envío de datos al juego:

- Señal de disparo y aparición en pantalla de la realimentación de disparo.
 - Señal de desaparición de la realimentación de disparo en pantalla.
 - Información sobre la proximidad del tanque a un obstáculo.
- Lectura de la información que recibe Arduino sobre la distancia del tanque a un obstáculo próximo.
 - Envío de señal de movimiento al tanque (giro o avance).

6.3. Implementación del código para Arduino

Por último me dispongo a explicar el desarrollo del código que ejecuta la placa Arduino desarrollado en el IDE propio para el control del tanque. Este código consta de dos funciones principales (`Setup()` y `loop()`) y varias auxiliares definidas por mí para el cálculo de distancia con el sensor de proximidad y para el control de motores a la hora de mover el tanque en función de las órdenes que se reciban desde el host servidor.

La función `Setup()` sirve para la inicialización de variables y la función `loop()` se ejecuta en forma de bucle infinito(), de modo que cada uno de los apartados que explico a continuación son ejecutados en bucle justo en el orden en el que los voy a describir.

Conexión al servidor

Primeramente, antes de cualquier cosa, la placa Arduino se conecta al servidor TCP especificando la dirección ip y el puerto y, haciendo uso de los comandos de control del módulo wifi ESP8266, siendo comprobado en cada iteración del método `loop()` que la conexión sigue siendo efectiva (en cuyo caso simplemente no hará nada, dando paso a las siguientes acciones).

Cálculo de la distancia

Una vez comprobado que la conexión está garantizada, la primera que se hace es llamar a la función `distanciaSensor()` para, haciendo uso del sensor de proximidad, calcular la distancia al obstáculo más próximo al tanque. La función mencionada se encarga de activar y desactivar el trigger para lanzar la onda que recorrerá el espacio y será recogida por el pin echo para hacer los cálculos correspondientes con el fin de obtener la distancia tal como expliqué en el análisis del hardware.

Tratamiento de la distancia

El sensor de proximidad trabaja enviando señales de distancia de forma periódica cada pocas milésimas de segundo. En mi juego he decidido limitar a que simplemente me avise el sensor de distancia una sola vez cuando el tanque se encuentre a menos de diez centímetros de un obstáculo. Para ello, mediante booleanos y condicionales, si la distancia es menor que diez centímetros y aún no se ha enviado la señal al servidor, entonces la enviará por el socket creado; en caso contrario, si la distancia es mayor o igual que 10 o ya se ha enviado previamente una señal, simplemente se despreciará dicho dato.

Movimiento del tanque

Una vez enviada (o no) la distancia actual del tanque a un obstáculo, solo queda comprobar si el socket que comparte con el servidor contiene información para ser leída, es decir, falta comprobar si el servidor ha enviado una orden de movimiento al tanque e interpretar dicha orden. Para ello, como ya he dicho, mediante el comando correspondiente comprobamos si en el puerto serial definido para la recepción de datos existe información. Dicha información, tal y como he programado el sistema, siempre será un carácter de los cuatro posibles que he decidido para los cuatro movimientos básicos del tanque: girar izquierda, girar derecha, avanzar y retroceder. Una vez recibido el carácter del servidor, solo he de hacer 4 condicionales para decidir qué función llamar para el movimiento del tanque. Cada una de las funciones de movimiento creadas, simplemente contienen un par de líneas para controlar el giro de cada motor del tanque.

6.4. Conexiones hardware Arduino

Esta sección va dedicada a mostrar las conexiones entre los accesorios o módulos utilizados con Arduino y la placa en sí. Mostraré las conexiones de forma independiente aunque, en la práctica, todas se conectan al mismo tiempo a la misma placa.

6.4.1. Módulo ESP8266

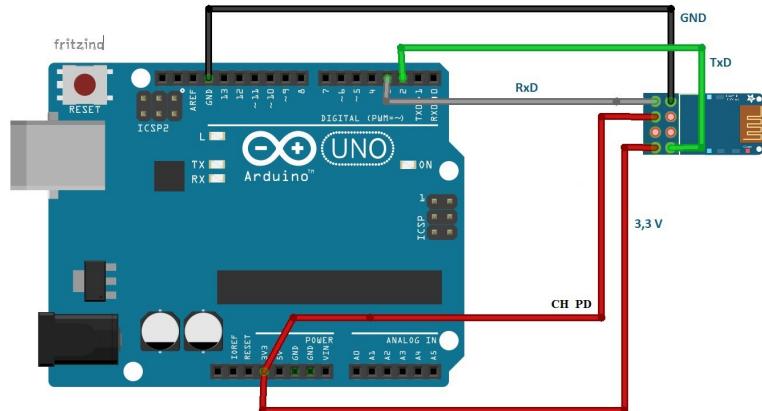


Figura 6.2: Conexión módulo WiFi con Arduino[29]

6.4.2. Detector de proximidad

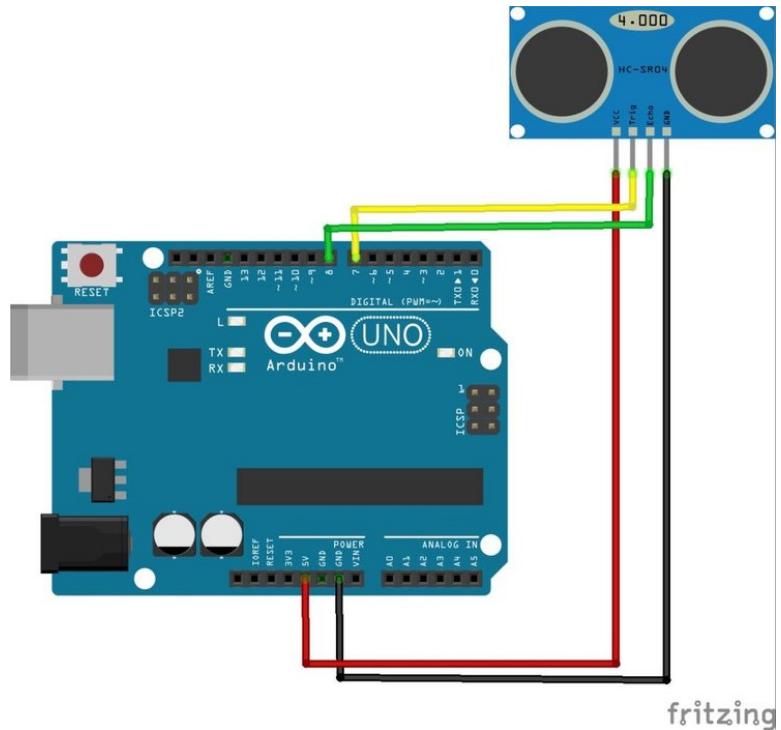


Figura 6.3: Conexión detector de proximidad con Arduino[30]

6.4.3. Motores

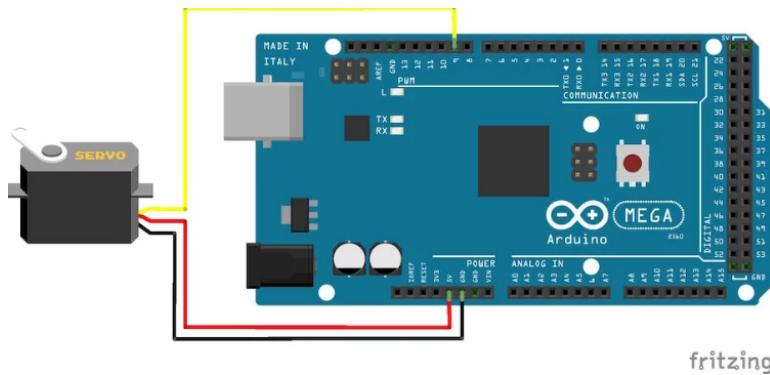


Figura 6.4: Conexión motores servo con Arduino[31]

6.5. Realización del tanque

El proceso de realización del tanque consta de tres fases: diseño, impresión y prueba de piezas.

Diseño

Realizado, como ya comenté en su momento, en el software de modelado 3D Blender, el proceso de modelado ha sido llevado a cabo mediante técnicas de extrusión, división, adición y eliminación de caras y aristas, operaciones booleanas entre figuras geométricas y reducción de vértices y aristas para dotar de simplicidad a los propios modelos. Todos han sido estudiados de forma que se priorice la eficiencia en cuanto a gasto de luz en el tiempo de impresión y de filamento. A continuación mostraré un ejemplo de modelo 3D durante el proceso de modelado.

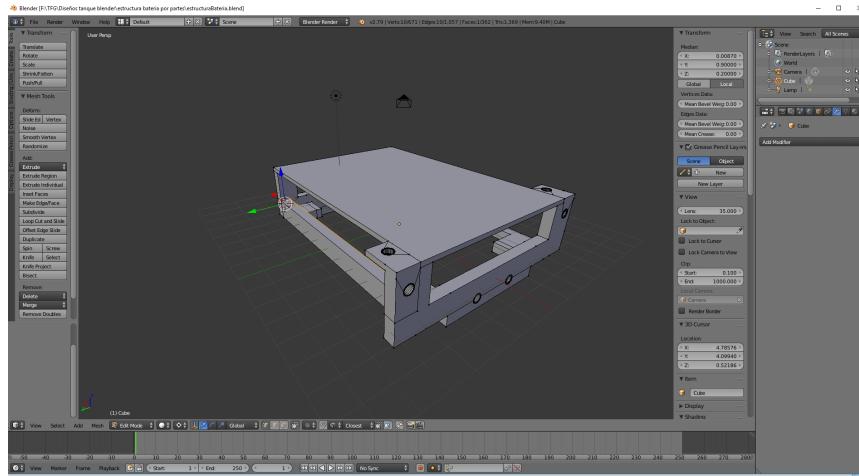


Figura 6.5: Soporte para batería en fase de modelado

Impresión y prueba

Como ya he expliqué en el capítulo 3 dedicado al análisis del hardware, he utilizado mi impresora 3D Prusa i3 Psique Steel para, mediante el uso de filamento PLA, imprimir los modelos diseñados en Blender 3D que constituirán el tanque en sí. Sin más que añadir a lo ya explicado en el capítulo 3, muestro una fotografía en la que podemos ver como realizo una prueba de medidas y soporte de peso de la estructura para la batería que he realizado.

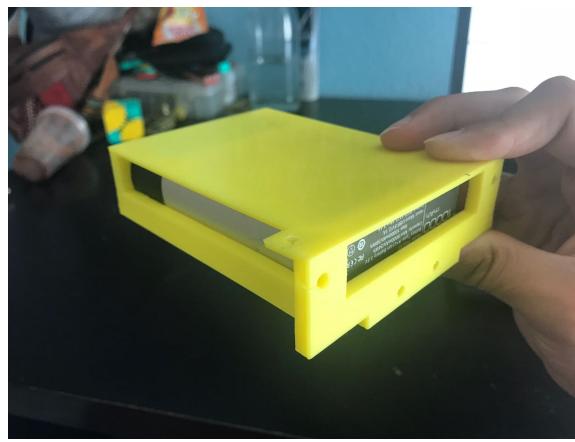


Figura 6.6: Soporte para batería en fase de prueba

El resto de piezas realizadas por mí mismo se encuentran en el anexo del proyecto y todas y cada una de ellas han seguido el mismo proceso de creación.

Capítulo 7

Conclusiones finales

Finalmente, una vez explicado el desarrollo completo del proyecto, es el momento de mirar hacia atrás para cerciorarme de las asignaturas del grado cursadas sin las que no hubiera sido capaz de desenvolverme de manera tan efectiva para la realización del proyecto, así como darme cuenta de los conocimientos que he adquirido durante esta etapa de mi carrera y, por último, realizar una valoración de todo ello.

7.1. Asignaturas de utilidad

Haciendo un repaso por los campos de conocimiento abarcados en todo el proyecto, me doy cuenta de que han sido muchas las asignaturas que me han ayudado a desarrollar cierto criterio para manejar la situación y la toma de decisiones de una manera lógica. A continuación menciono las que pienso que han influido en mayor medida:

- **Asignaturas de programación pura:** tales como **Fundamentos de Programación**, **Metodología de la Programación** o **Metaheurísticas** por ejemplo, ya que en mi opinión, sirven de base para comenzar a desarrollar el pensamiento de un programador novel como somos los estudiantes del grado, sin importar el lenguaje utilizado en cada una de las asignaturas. Además, teniendo en cuenta esto último, Unity hace uso de C#, el cual es muy similar a C++, el lenguaje que más he utilizado en la carrera, así como Arduino hace uso de C y, Netbeans hace uso de Java, lenguaje también muy usado en el grado.
- **Nuevos Paradigmas de Interacción:** asignatura que me motivó a hacer uso del dispositivo de interacción gestual MYO Armband, habiéndolo probado por primera vez en algunas de sus clases.

- **Sistemas Concurrentes y Distribuidos/Sistemas Operativos:** es aquí donde aprendí el significado y uso de la palabra "hebra", concepto del cual he hecho uso a la hora de crear el servidor para que pudiese gestionar peticiones de varios clientes.
- **Programación y Diseño Orientado a Objetos/Fundamentos de Ingeniería del Software:** PDOO fue la primera asignatura con la que creé un juego, el Napakalaki. Ésta me hizo ver por primera vez la envergadura de un proyecto real y me enseñó las bases de la gestión de clases y métodos para, más tarde, en FIS, aprender los conceptos más básicos de la ingeniería del software con UML, ayudándonos a ser capaces de realizar un análisis del software efectivo.
- **Fundamentos de Redes:** ha sido la asignatura que me ha enseñado los conocimientos necesarios para la creación de la arquitectura cliente-servidor mediante sockets TCP.
- **Computación Ubicua e Inteligencia Ambiental:** me ha aportado muchos conocimientos para este proyecto. Fue la asignatura con la que comencé a hacer uso de Unity y el lenguaje de programación C# y además, me introduce con ella en el mundo de la Realidad Aumentada con un juego que hacía uso de la misma.

Además de las asignaturas mencionadas he necesitado hacer uso de mis capacidades autodidactas para aprender a hacer uso tanto del software de modelado 3D Blender como de la misma placa Arduino junto con todos sus sensores y módulos, pues es cierto que no he aprendido nada relacionado con estos dos campos de estudio durante el desarrollo del grado.

7.2. Conocimientos aprendidos

Antes de comenzar a desarrollar cualquiera de las partes que componen el proyecto pensé que, en general, me iba a resultar bastante menos complicado resolver el problema que me planteé en un principio. Con el paso del tiempo han surgido muchas complicaciones relacionadas con los cuatro bloques que componen el sistema: MYO Armband, Arduino, servidor y el juego en sí. Gracias a la resolución de los problemas que han ido surgiendo a lo largo de todo el proyecto, he logrado consolidar en mi cabeza conocimientos posiblemente útiles para mi futuro.

Como he comentado en el apartado anterior, antes de comenzar ya tenía en mis manos algo de experiencia con el uso de **Unity** y la **Realidad Aumentada**, sin embargo, en esa pequeña experiencia vivida en la asignatura CUIA no me había enfrentado solo a un proyecto tan complicado, ni había

hecho uso exactamente de las mismas herramientas. He aprendido por ejemplo, a tratar las colisiones entre modelos 3D, así como un nuevo uso de la Realidad Aumentada y, he ganado experiencia en otros aspectos del IDE Unity y su scripting.

En relación con MYO Armband, he aprendido a manejar (de forma esqueta, hay que decirlo) su comportamiento para hacer uso de sus gestos mediante la librería que mencioné en su momento, lo cual me abre las puertas a futuros proyectos que puedan ser planteados en mi pensamiento.

Con respecto al servidor he fortalecido mis conocimientos acerca de sockets TCP y el funcionamiento de las hebras en Java, cosa que no había utilizado nunca en dicho lenguaje.

Finalmente, he aprendido a utilizar de manera efectiva el funcionamiento de ciertos módulos de Arduino como el ESP8266, encargado de la conexión WiFi y el traspaso de información, así como el tratamiento de las distancias con el sensor de proximidad.

7.3. Valoración final

Mi conclusión final sobre el sistema desarrollado se basa en una mezcla de **esfuerzo**, por el hecho de haber tenido que hacer uso de muchas horas de trabajo delante de mi ordenador y conocimientos adquiridos tanto en el grado como por cuenta propia, y **orgullo**, por haber sido capaz de desarrollar un sistema tan complejo como nunca antes había hecho, viendo el potencial de las herramientas que el grado ha aportado a mi conocimiento.

Como dije anteriormente, pensé que me sería menos costosa la realización del proyecto en general, pero aun así, estoy contento por haber escogido un Trabajo de Fin de Grado como el que he elegido, ya que ha sido de mi total interés y agrado, por ello no me arrepiento de las horas empleadas para su finalización, ya que además siento que he enriquecido mis conocimientos en un área que es de mi agrado, lo cual me abre puertas para nuevos proyectos en un futuro.

En forma de valoración propia aunque, como ya he dicho, estoy orgulloso del trabajo realizado y los conocimientos adquiridos, me gustaría que el sistema respondiese de mejor forma en cuanto al uso, no por motivos de implementación, sino por motivos que quedan fuera de mi alcance con el hardware del que dispongo, me explico: en ocasiones MYO Armband no detecta de manera efectiva los gestos, posiblemente debido a una calibración del sistema poco efectiva, así como el funcionamiento del juego en sí, ya que las colisiones entre modelos de realidad aumentada utilizadas en Unity dependen en gran medida de la calidad de la cámara del dispositivo o de si se capta en unas buenas condiciones de luminosidad y tamaño de los patrones.

Capítulo 8

Trabajos futuros

Por último me queda por exponer las mejoras que en un futuro, al continuar con el desarrollo del proyecto, me gustaría realizar.

8.1. Gafas de realidad aumentada

Primero de todo me gustaría hacer que el jugador pudiese hacer uso del juego sin tener que llevar el teléfono en la mano para crear una inmersión superior en el juego en sí. Dicha hazaña sería posible con unas gafas de realidad aumentada como las Hololens de Microsoft o las Google Glass. Pienso que no debería ser un trabajo excesivamente costoso, pues simplemente sería hacer uso de la librería que utiliza Unity para el uso de aplicaciones en dispositivos de Realidad Virtual (que en este caso sirve para el mismo objetivo).



Figura 8.1: Prueba de detección de gestos de las Hololens[32]

8.2. Añadir modos de juego

Actualmente el único modo de juego es el contrarreloj, en el que has de eliminar a los enemigos en el menor tiempo posible para marcar un nuevo récord. En un futuro me gustaría meter los siguientes modos de juego:

- **Competición cooperativa:** con dos tanques, dos personas pueden jugar al mismo tiempo. Deberían situarse cada uno en un extremo de la zona de juego con enemigos y armas repartidos por todo el campo. El objetivo consistiría en eliminar a los enemigos que encuentres conforme avances y además, buscar a tu enemigo principal (el otro jugador) para eliminarlo, ya que contendrá la gema que, una vez lo hayas eliminado, deberás llevar al "Rey de las Gemas", un personaje ficticio creado que te recompensará cuando le entregues dicha gema, lo que te hará ganar la partida.
- **El escondite inglés:** con el tanque situado en el suelo y rodeado de unos 6 patrones de enemigos (situados en círculo alrededor del tanque), cada cierto tiempo un solo enemigo de los 6 elegido al azar comenzará a moverse mientras los demás permanecen quietos. En dicha cantidad de tiempo, el jugador deberá tratar de hacer rotar al tanque hasta orientarlo hacia el enemigo que está en movimiento para disparar hasta eliminarlo. Si el jugador no llega a tiempo para eliminarlo perderá puntos de vida.
- **Tank ScapeRoom:** en el modo de juego contrarreloj que he desarrollado, el jugador saber perfectamente donde se encuentra la pista y qué camino seguir para llegar hasta ella. Con este modo de juego, haciendo uso del User Defined Target (patrón de Realidad Aumentada elegido por el usuario), uno de los dos jugadores, sin que el otro se dé cuenta, ha de escoger varios objetos de la sala de juego donde situar pistas que su contrincante deberá recoger para resolver un acertijo creado por el mismo enemigo del jugador, lo que le dará la victoria. De esta forma el jugador habrá de recorrer varios caminos hasta encontrar todas las pistas que su contrincante ha dejado esparcidas por el campo, recogiendo armas y eliminando a enemigos tal como ya está implementado. Para añadirle competitividad, evidentemente, habrá que hacerlo en la menor cantidad de tiempo posible para ganar al contrincante.

8.3. Añadir uso de varios armas

Mi idea es crear un HUD de armas donde poder escoger una u otra en función del daño que realicen al enemigo. De esta forma ante un enemigo

que sea más fuerte que otro podré elegir el arma conveniente. El sistema de obtención de energía no será como elde ahora, aquí cada arma tendrá su nivel de energía, al contrario que la implementación actual en el que la energía es del tanque y aumenta conforme va recogiendo armas.

8.4. Animaciones, dinamismo

Actualmente la realimentación de disparo, por ejemplo, consiste en una imagen que sale en pantalla cuando el jugador aprieta el puño con el MYO. El objetivo en este aspecto es hacer que el propio tanque dispare un rayo láser, una flecha o una bala en función del tipo de arma que maneje en el momento de disparar. De la misma forma me gustaría hacer que los enemigos estuviesen animados en ese aspecto, así como las armas a la hora de ser recogidas. También me gustaría introducir efectos especiales que favorezcan la inmersión y sorpresa del jugador durante la experiencia.

8.5. El servidor

Como proyecto de futuro del prototipo de juego que he creado, el servidor central además de interconectar a todos los sistemas, se encargará de almacenar información de los jugadores así como ofrecer la conexión de varios al mismo tiempo para realizar partidas multijugador que, con motivo de la falta de recursos (tanque y dispositivos MYO) no ha podido realizarse en esta ocasión. El principal uso es almacenar un registro de jugadores con los tiempos mínimos realizados en la contrarreloj, de esta forma la competitividad aumenta al conocer la tabla de tiempos mínimos, pues un jugador siempre buscará encontrarse en el número uno. Solo sería necesario una base de datos y, un método de controlar los tiempos para partidas idénticas, es decir, no es lo mismo terminar una partida en 2 minutos en la cual hay 3 enemigos, que una partida en la que existen 10 enemigos.

Además este servidor almacenará información de tiempos del jugador en concreto, es decir, cada jugador podrá ver una tabla con sus propios tiempos en todas las partidas desarrolladas.

8.6. Niveles de dificultad

El modo historia del juego se basará en niveles de dificultad. Se pondrán retos desde los más simples a los más complicados conforme se avanza en la historia. Cada vez habrá un número superior de enemigos y

se desbloquearán armas (proporcionando nuevos patrones que imprimir en papel para poder usar) al terminar cada nivel.

Bibliografía

- [1] At91sam3x8e. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf.
- [2] Atmega168. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2545-8-bit- AVR-Microcontroller-ATmega48-88-168_Datasheet.pdf.
- [3] Atmega2560. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit- AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf.
- [4] Atmega328. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit- AVR-Microcontroller-ATmega328-328P_Datasheet.pdf.
- [5] Atmega32u4. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit- AVR-ATmega16U4-32U4_Datasheet.pdf.
- [6] Documentación esp8266. https://www.espressif.com/sites/default/files/0a-esp8266ex_datasheet_en_1.pdf.
- [7] Documentación sensor hc-sr04. <https://www.mouser.com/ds/2/813/HCSR04-1022824.pdf>.
- [8] Estudio de la sexta tv noticias sobre el uso de las impresoras 3d en las escuelas. <https://www.lasexta.com/noticias/ciencia-tecnologia/las-impresoras-3d-llegan-a-la-escuela-el-20-de-colegios-las-incluye-en-sus-actividades-2017030158b704d40cf2894da459ab53.html>.
- [9] Figura 2.1: Measurekit. <https://www.newsledge.com/tapmeasure-ios-11-app/>.
- [10] Figura 2.2: The machines. <https://www.youtube.com/watch?v=QYJo3YUdgCA>.

- [11] Figura 2.3: Pokemon go. <https://support.pokemongo.nianticlabs.com/hc/es/articles/115015868188-Captura-de-Pok%C3%A9mon-en-modo-RA-solo-iOS->.
- [12] Figura 2.4: Minecraft and hololens. <https://www.xataka.com/videojuegos/e3-2015-minecraft-tiene-mas-sentido-para-microsoft-con-hololens>.
- [13] Figura 2.5: Interacción hombre máquina. <https://inproduction.net/news/the-exciting-history-of-cad-drawings/>.
- [14] Figura 2.6: Ejemplo de uso de la acetona en modelos 3d. <https://www.hwlible.com/mejora-la-calidad-tus-impresiones-3d-este-sencillo-truco/>.
- [15] Figura 2.7: Casa impresa en 3d. <https://clipset.20minutos.es/las-casas-impresas-en-3d-ya-son-una-realidad-y-son-muy-baratas/>.
- [16] Figura 2.8: demostración ar + myo. https://www.youtube.com/watch?v=mUSn_soFeiw&index=2&list=PLTrr4Q-tY8MAgagnoN4NMJqAiPKwiV82w.
- [17] Figura 2.9: Juego ar + myo. <https://www.youtube.com/watch?v=gmRP0lITY-w&index=6&list=PLTrr4Q-tY8MAgagnoN4NMJqAiPKwiV82w>.
- [18] Figura 3.1: Placa arduino. <https://www.reichelt.com/de/en/arduino-uno-rev-3-atmega328-usb-arduino-uno-p119045.html>.
- [19] Figura 3.2: Servomotor. <http://panamahitek.com/que-es-y-como-funciona-un-servomotor/>.
- [20] Figura 3.3: Sensor hc-sr04. <https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html>.
- [21] Figura 3.3 y 3.4: cálculo de distancia. <https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/>.
- [22] Figura 3.6: conexiones esp8266. <https://components101.com/wireless/esp8266-pinout-configuration-features-datasheet>.
- [23] Figura 3.7: esquema myo. <https://www.robotshop.com/en/myo-gesture-control-armband-black.html>.
- [24] Figura 3.8: aplicacion myo. https://www.researchgate.net/figure/Pisa-IIT-SoftHand-2-controlled-though-MYO-armband_fig1_323735959.

- [25] Figura 4.1: ejemplo blender. https://docs.blender.org/manual/es/dev/getting_started/about/introduction.html.
- [26] Figura 4.2: Ide arduino. <https://core-electronics.com.au/tutorials/arduino-ide-tutorial.html>.
- [27] Figura 4.3: Hearthstone. https://en.wikipedia.org/wiki/Gameplay_of_Hearthstone#/media/File:Hearthstone_screenshot.png.
- [28] Figura 6.1: Tcp. <http://www.keyboardbanger.com/tcp-three-way-handshake/>.
- [29] Figura 6.2: Conexión esp8266. <https://aprendiendoarduino.wordpress.com/2016/12/21/arduino-conectado-a-internet/>.
- [30] Figura 6.3: Conexión hc-sr04. <https://www.instructables.com/id/Very-Simple-HC-SR04-Connection-to-Arduino-Example/>.
- [31] Figura 6.4: Conexión motores servo. <https://www.allaboutcircuits.com/projects/servo-motor-control-with-an-arduino/>.
- [32] Figura 8.1: Prueba de detección de gestos de las hololens. <https://www.3djuegos.com/noticias-ver/159984/hololens-al-descubierto-microsoft-explica-las-funciones-del/>.
- [33] Historia scrum. <https://www.scruminc.com/takeuchi-and-nonaka-roots-of-scrum/>.
- [34] Información sobre pines tx y rx de esp8266. <http://cursos.mcielectronics.cl/señales-tx-y-rx-1/>.
- [35] Pagina oficial de espressif, esp8266. <https://www.espressif.com/en/products/hardware/esp8266ex/overview>.
- [36] Placas arduino. <https://www.arduino.cc/en/Main/Products>.
- [37] repositorio librería myo. <https://github.com/NicholasAStuart/myo-java>.
- [38] Scrum. <https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf#zoom=100>.

Capítulo 9

Anexo

El presente anexo lo realizo con el fin de prestar constancia de los modelos realizados por mí mismo para el tanque utilizado en el juego. Cada pieza mostrará un plano general así como distintas perspectivas con las medidas utilizadas.

9.1. Eje de las ruedas traseras

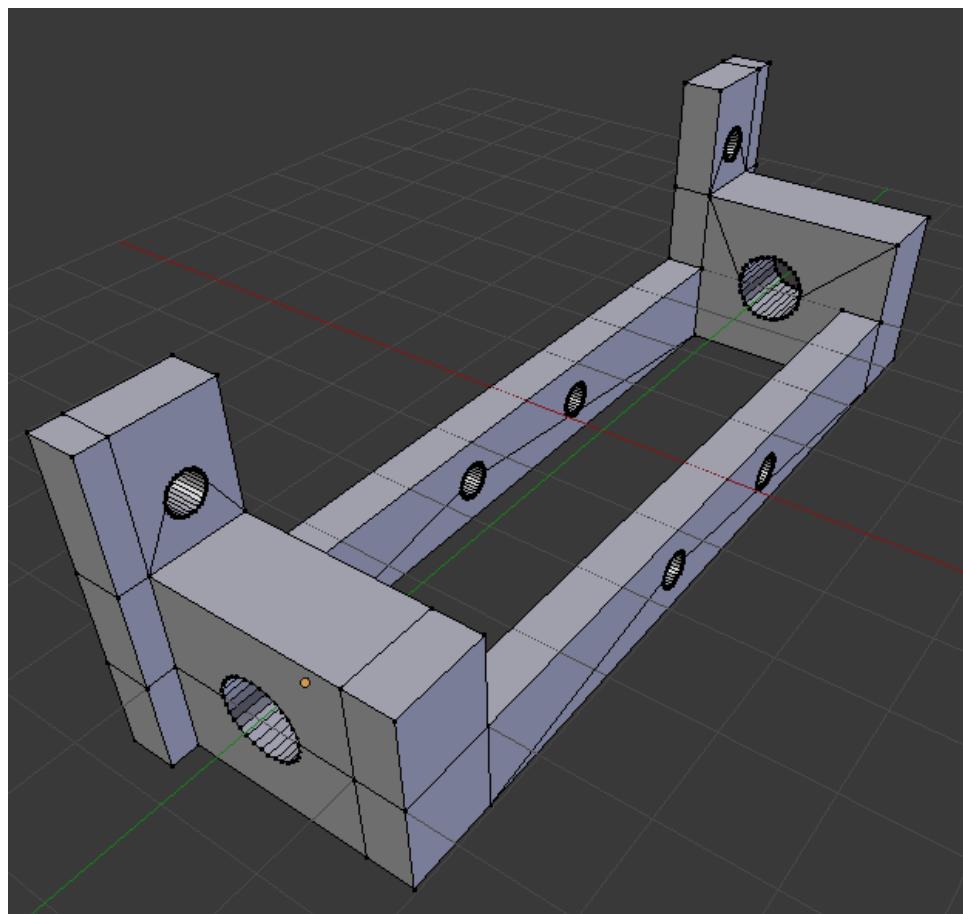


Figura 9.1: Vista general

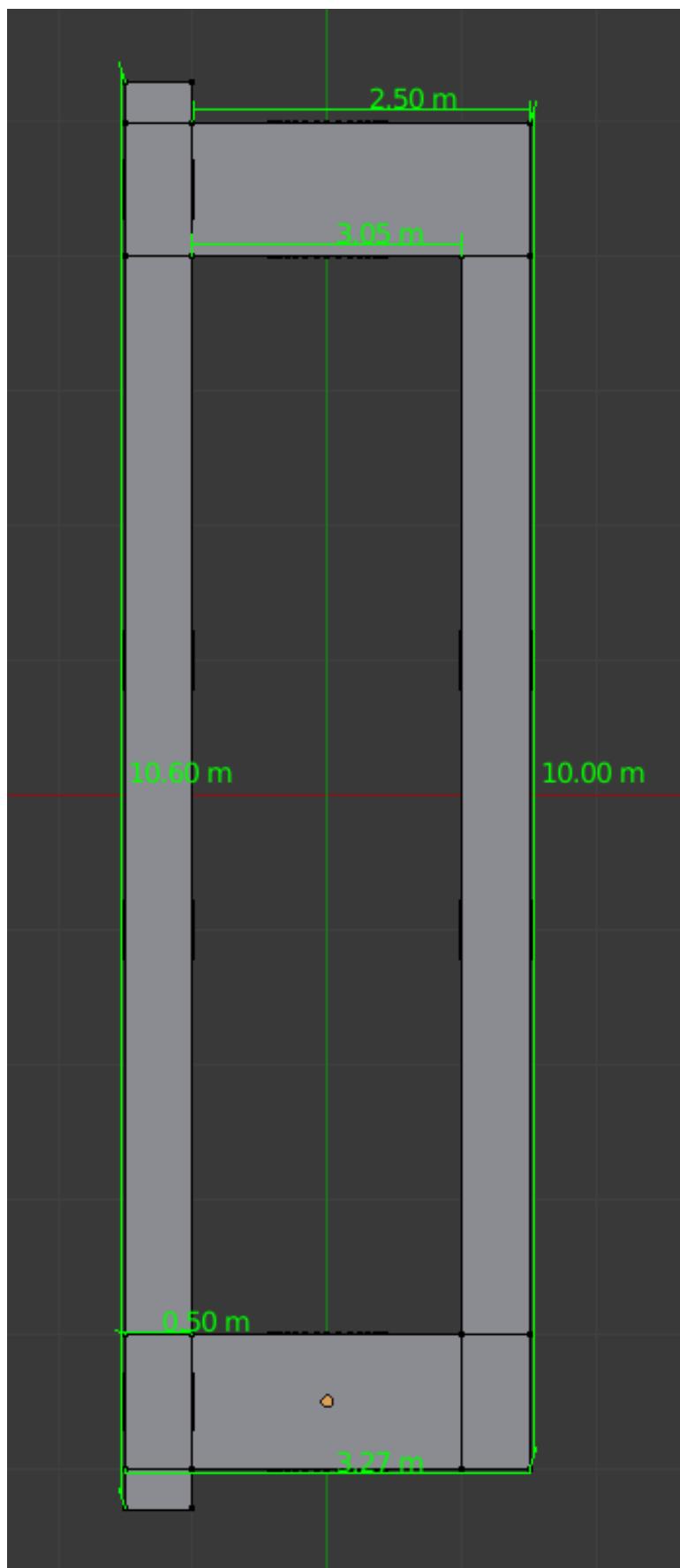


Figura 9.2: Vista en planta

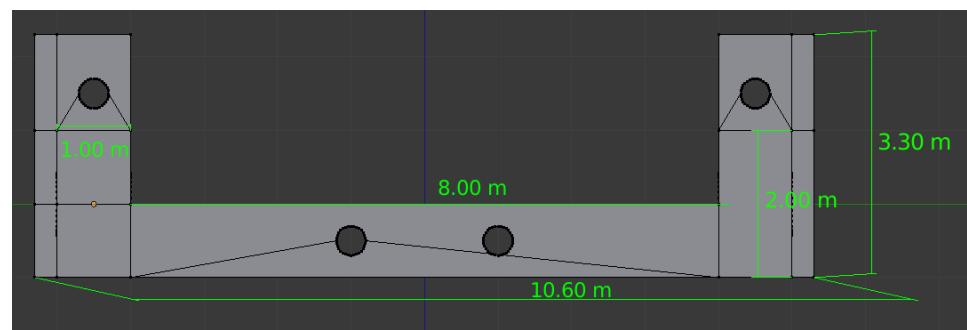


Figura 9.3: Vista frontal

9.2. Soporte de los motores servo

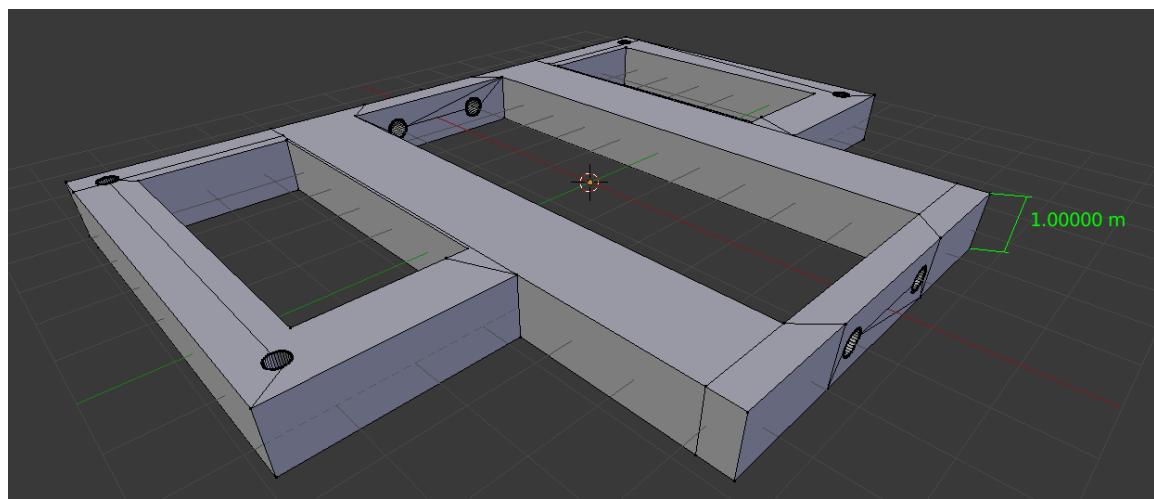


Figura 9.4: Vista general

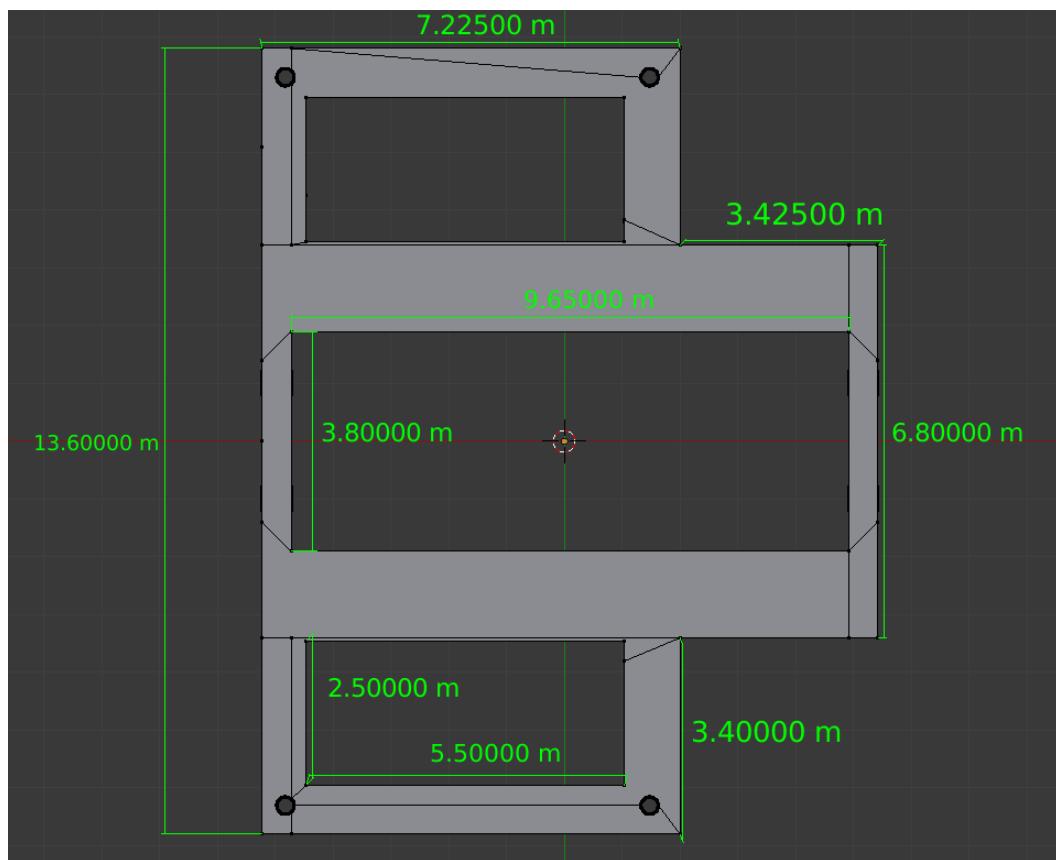


Figura 9.5: Vista en planta

9.3. Caja para guardar la batería portátil

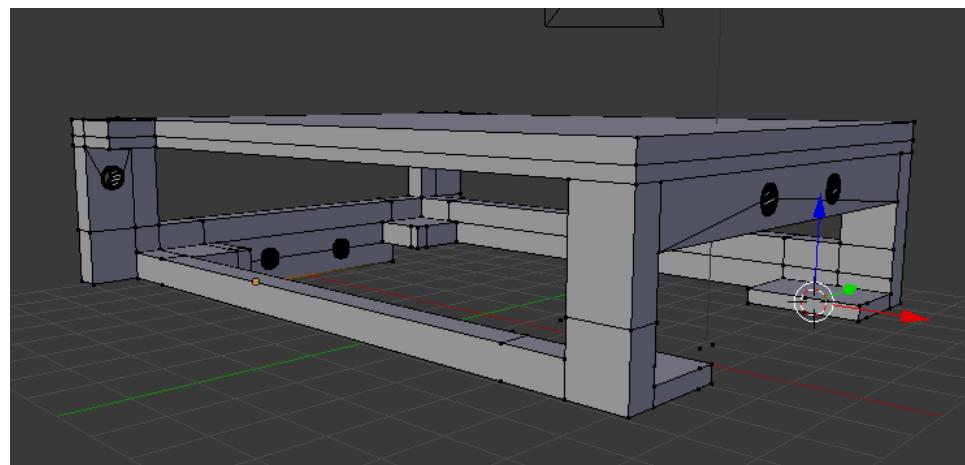


Figura 9.6: Vista general

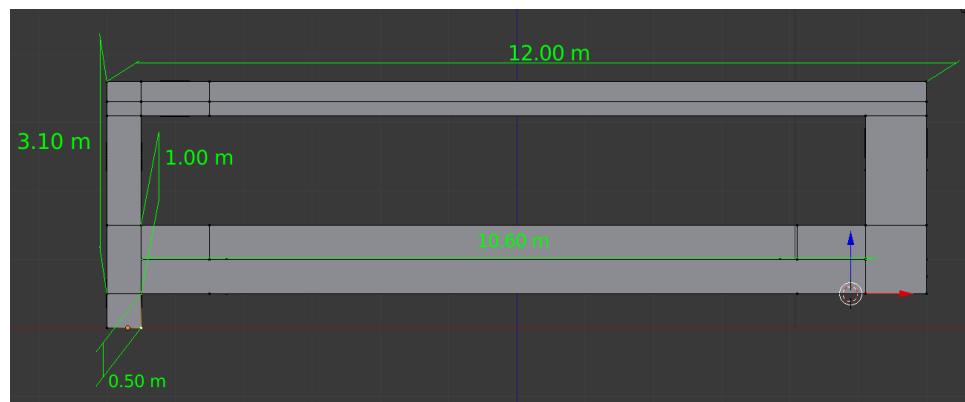


Figura 9.7: Vista lateral

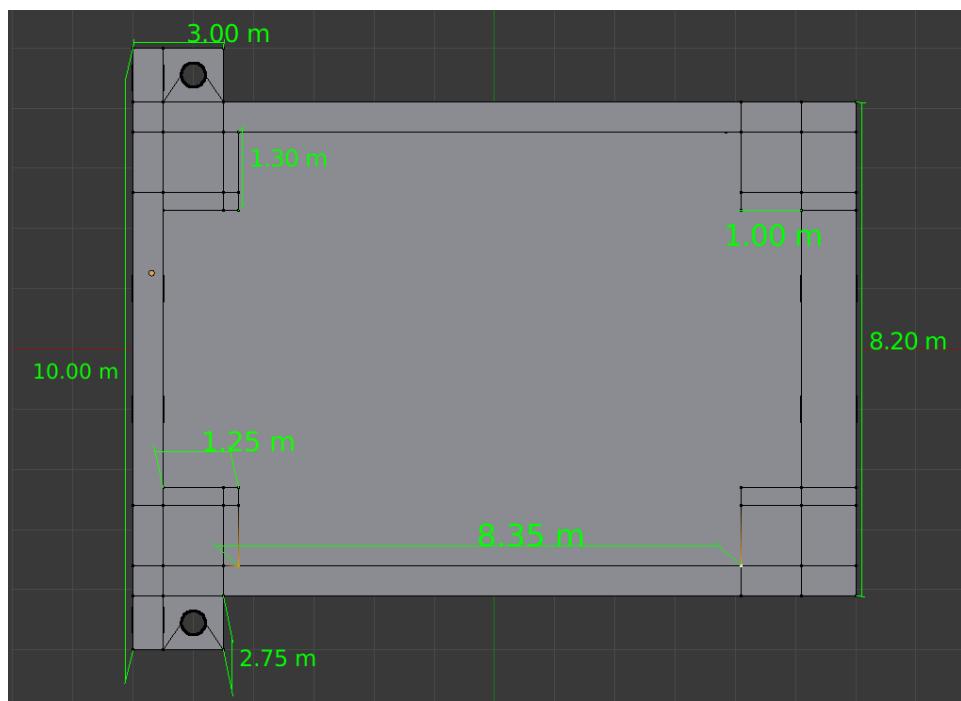


Figura 9.8: Vista desde abajo

9.4. Estructura para proteger la placa Arduino

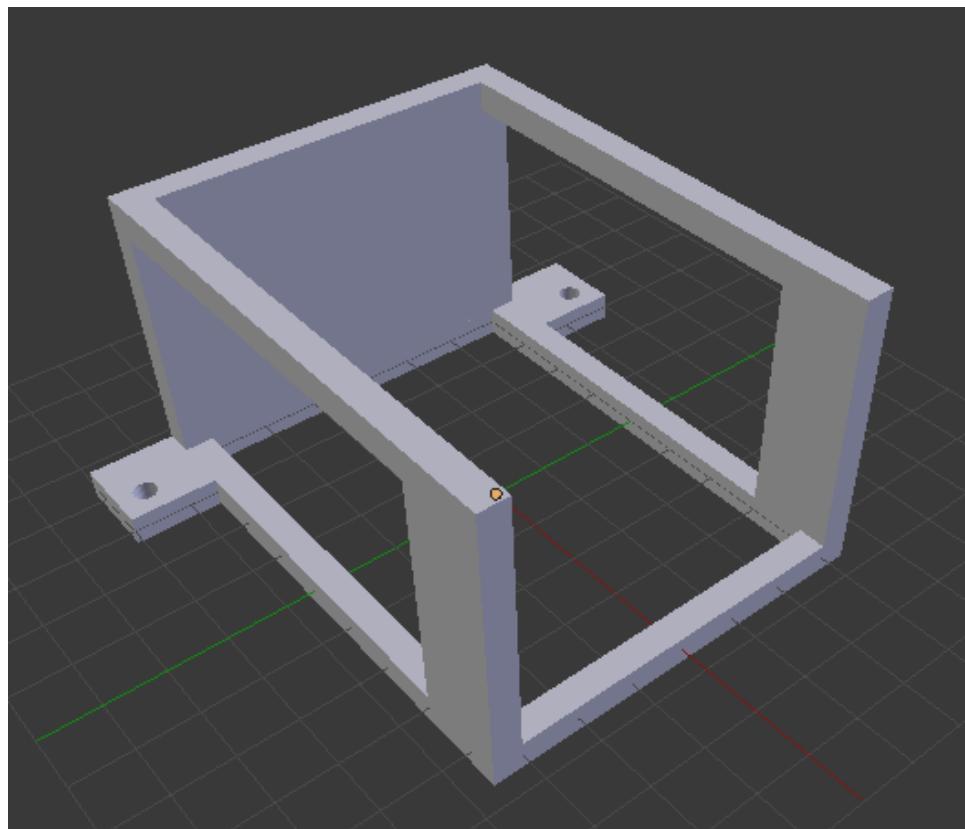


Figura 9.9: Vista general

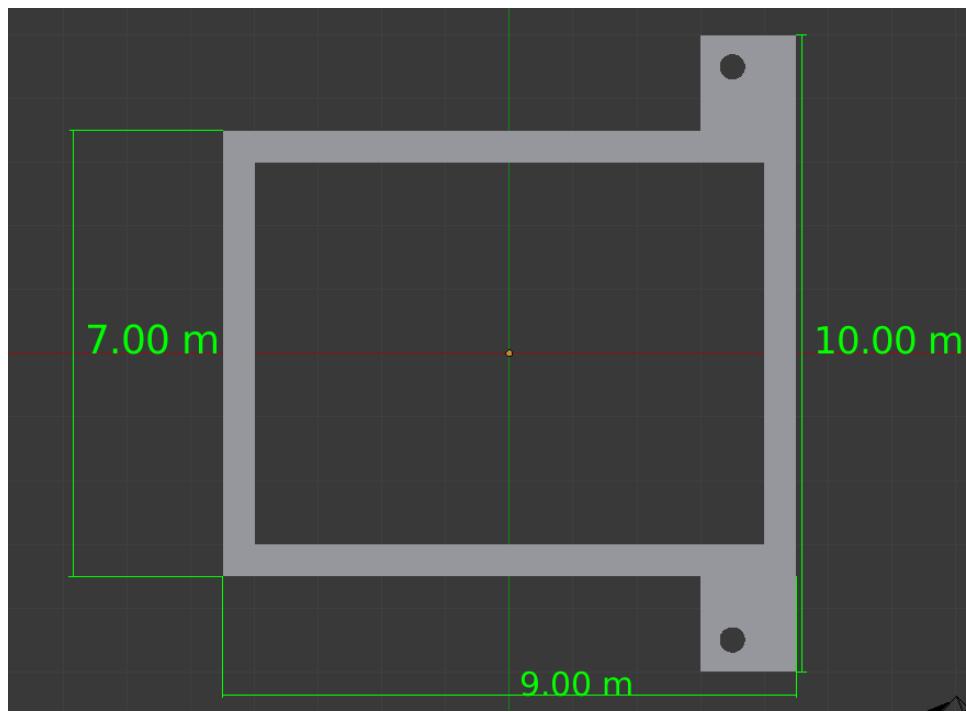


Figura 9.10: Vista en planta

