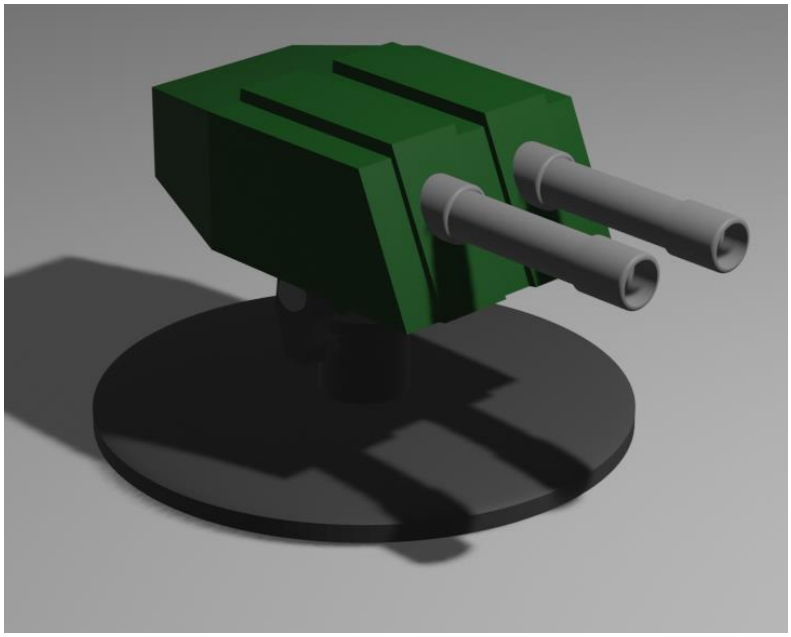


CENTINELA NERF

Equipped robot with a nerf turret, which you can play with, interact and activate guardian mode

PROJECT SPRINT #3.
DATE: 26 May 2020

VÍCTOR AGUILAR MARTÍNEZ (1496221)
DANIEL PARDO NAVARRO (1196895)
MARCO PICAZOS CARRILLO (1493144)
JOSÉ ANTONIO PICAZOS CARRILLO (1495152)

Table of Contents

Final Report
Date: 26 May 2020

CENTINELA NERF

Equipped robot with a nerf turret, which you can play with, interact and activate guardian mode

Project description

Centinela Nerf es un pequeño robot de sobremesa con aspecto de torreta. Está equipado con balas nerf que pueden ser disparadas de forma automática por el propio robot o por el usuario. El robot se sincroniza con una aplicación donde el usuario podrá realizar diversas acciones como activar los motores, dirigir el movimiento de la torreta, disparar y configurar diferentes opciones.

La estructura general del robot se compone dos motores DC que propulsan la bala nerf 3 servomotores. 2 de los servomotores se encargan de las funciones de movimientos del robot en 2 ejes mientras que el tercer servo coloca la bala en los motores para que sea disparada. Todo el control de la parte mecánica y del movimiento se realiza en base a un Arduino.

Por otro lado el robot se equipa con una Raspberry Pi que se encarga de enviar las órdenes de movimiento al Arduino. En este sentido, este dispositivo tiene además una cámara y dispositivos de entrada y salida de sonido. Gracias a ello el robot podrá ver y escuchar al usuario, reconocerlo e interactuar. También puede funcionar de manera autónoma gracias a la función 'Centinela' en la que el robot vigila la zona detectando posibles intrusos dando señales de aviso o incluso disparando.

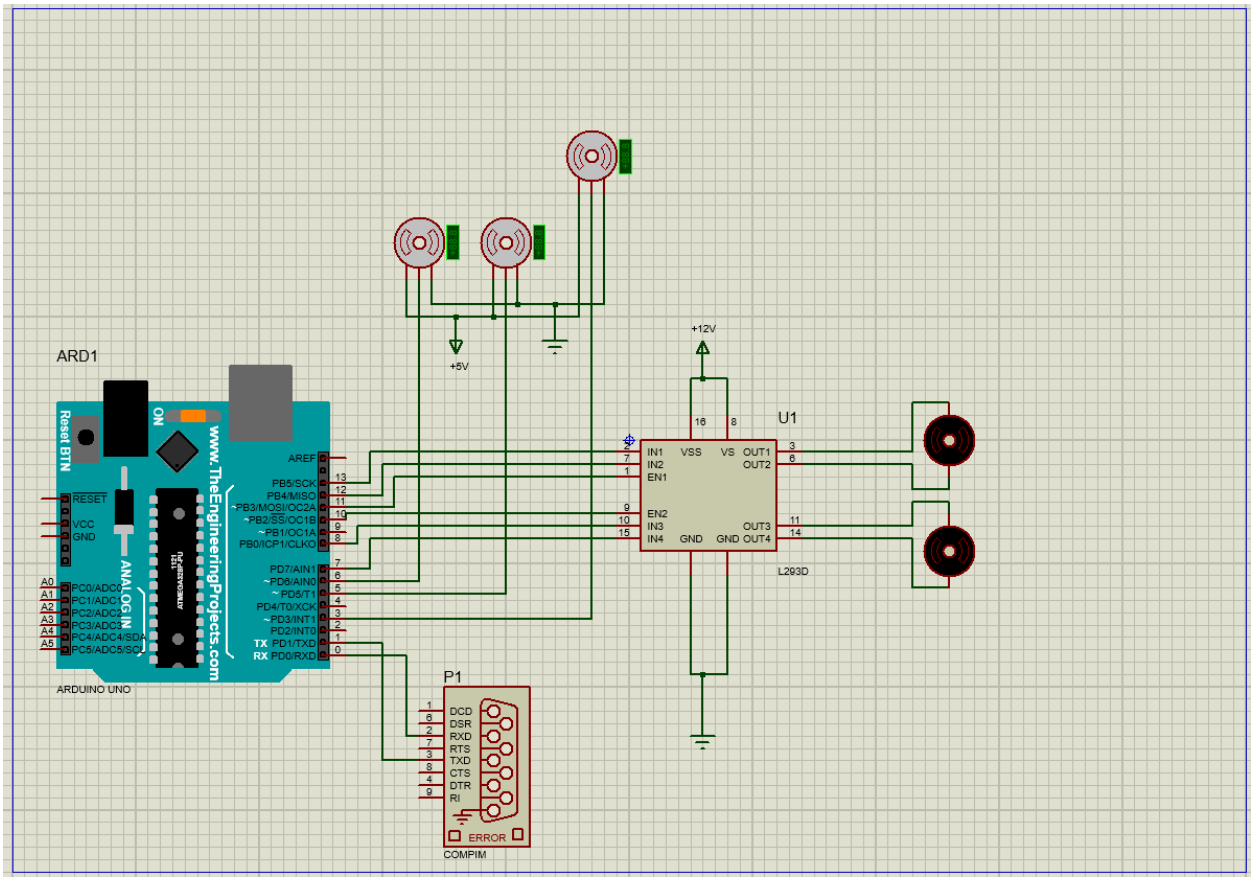
No obstante, dadas las limitaciones provocadas por el COVID-19 el robot no puede ser implementado de manera física, por lo que se desarrolla el proyecto a nivel de software y simulación. En este sentido, para adaptarse a la coyuntura actual se ha modificado parte de la funcionalidad en la simulación para que en vez de disparar balas nerf detecte a las personas si lleve mascarilla o no y en caso de ni llevar les lance una mascarilla.

Electronic components

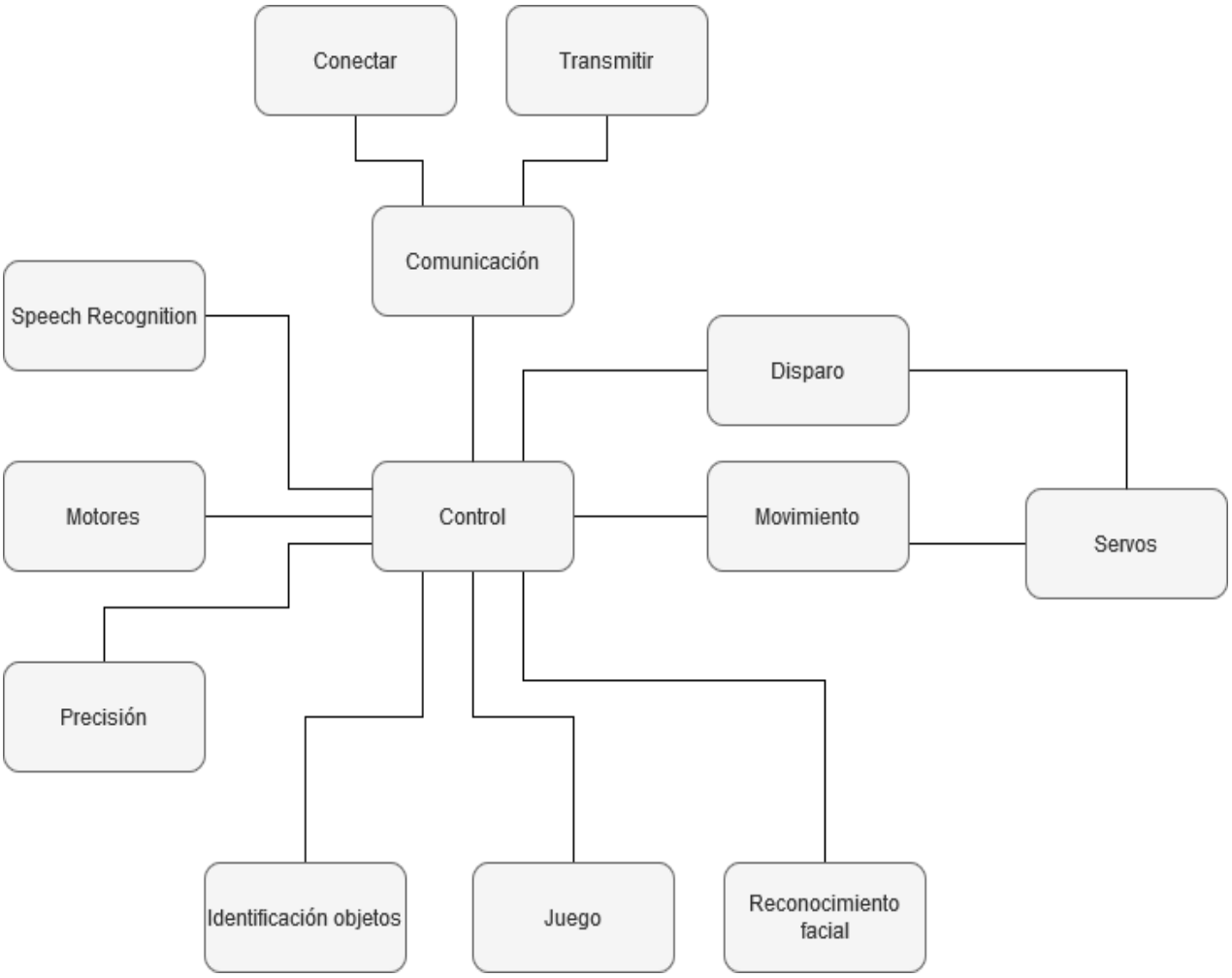
Los components inicialmente previstos son los siguientes:

- *Arduino Nano*
- *Raspberry Pi 3*
- 2x DC motor
- 3x Servo Motor
- Modulo de conexión (Bluetooth)
- We-cam
- Alimentación

Hardware Scheme



Software Architecture



Módulo	Control
Uso	Programa de control que se encarga de comunicar los diferentes módulos y la transmisión de datos
Dependencia	-
Módulo	Motores
Uso	Será el encargado de controlar la velocidad de giro de los motores, encendido y apagado dependiendo de las acciones que reciba del módulo de control.
Dependencia	Control

Módulo	Comunicación
Uso	Módulo general para establecer la comunicación entre el robot y los diferentes dispositivos
Dependencia	Control

Módulo	Conectar
Uso	Se encarga de establecer la conexión entre arduino y los dispositivos como PC o APP
Dependencia	Comunicación

Módulo	Transmitir
Uso	Se encarga de transmitir los datos entre arduino y los dispositivos una vez se ha establecido la comunicación
Dependencia	Comunicación
Módulo	Precisión
Uso	Con este módulo, nos encargaremos de controlar mediante visión por computador si la fiana o las caras del juego están dentro del rango de la torreta, realizando el movimiento del robot en caso de ser necesario
Dependencia	Control

Módulo	Reconocimiento facial
Uso	Mediante visión por computador se encargará de identificar y gestionar los usuarios dueños de la torreta.
Dependencia	Control

Módulo	Identificar Objetos
Uso	A través de visión por computador se encarga de identificar algunos objetos simples. En este caso, debe identificar e interpretar una diana con la que poder jugar
Dependencia	Control

Módulo	Juego
Uso	Este módulo se encarga de establecer los parámetros de un juego de diana y controlar la puntuación. El usuario debe poder jugar manualmente con la torreta sobre una diana y en función de la precisión del disparo sumar una mayor o menor cantidad de puntos.
Dependencia	Control

Módulo	Servos
Uso	Con este módulo gestionaremos los movimientos que realizará la torreta, los movimientos realizados dependen de los parámetros que reciba del módulo de movimiento y disparo. Ya que tenemos dos servos para el movimiento de la torreta y otro para realizar el disparo.
Dependencia	Movimiento, Dispar

Módulo	Movimiento
Uso	A partir de los movimientos del ratón realizados por el usuario genera los movimientos sobre los servos o activa el disparo. Cuando se han interpretado los movimientos a realizar el módulo controla los ejes de los dos servos encargados de mover la base de la torreta.
Dependencia	Control

Módulo	Disparo

Uso	Activa y controla el servo encargado de realizar el disparo con la bala nerf.
Dependencia	Control

Módulo	Speech Recognition
Uso	Mediante este módulo interactuamos con la torreta mediante la voz. Para ello este módulo se encargará de conectar con los servicios de google cloud y interpretar que acciones quiere que realice la torreta, para comunicarlo a control.
Dependencia	Control

Amazing contributions

Jueguete de sobremesa interactivo que ofrece entretenimiento y competencias formativas

- *Interactuación con un juegue con inteligencia artificial*

Una de las posibilidades que tiene este robot es convertirse en una especie de mascota, más allá de ser un simple torreta. Cabe decir que la edad indicada inicial no se puede establecer en este punto del proyecto, pero se plantea en un target group de entre 8 años, ya que las pistolas que existen en el mercado y usan este tipo de balas son a partir de esta edad, y 12 años. En cualquier caso, se trata de un mecanismo user friendly con una aplicación y un sistema muy fácil de aprender. Esto se debe a los módulos de reconocimiento facial y de voz y las posibilidades que la estructura planteada permite. En la Raspberry se pueden crear diferentes funciones mientras que el Arduino se encarga de ejecutarlas. En este aspecto, inicialmente se plantean algunas funcionalidades básicas como reconocer al dueño del juguete para poder interactuar de forma distinta con el resto y pequeños comandos de voz para comunicarse. Sin embargo, las posibilidades de expansión y desarrollo que ofrece la torreta son grandes como por ejemplo funciones de agenda, colocar alarmas a determinados horarios o hacer compañía. Esto puede ser una ayuda tanto a nivel académico y escolar de los niños como facilitar la labor de educación de los padres. Al tratarse de un jueguete interactivo del que pueden hacer incluirle funciones educativos serán más efectivas que los métodos tradicionales.

- *Entretenimiento adquiriendo competencias en tecnológicas*

Por otro lado, el juguete es un robot y además de ofrecer entretenimiento genera una serie de competencias en los niños. El uso de la tecnología está largamente extendido entre los niños, pero sin embargo el mayor porcentaje consiste en videoconsolas, tables o productos similares. Con el robot se pueden aprender otras competencias en el campo tecnológico que actualmente no está siendo tan explotado por los niños. De esta forma, se aproximarían a un robot real y a su funcionamiento, además de ver la interacción entre un aplicación y cómo se replica en un mecanismo físico.

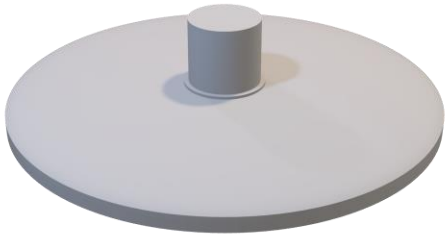
- *Aumenta la creatividad y la imaginación*

Finalmente, cabe recordar que el juegue se mueve sobre dos ejes pero es estático en la base. Esto también implica la necesidad de una interacción del niño con el juguete para moverlo y usarlo en diferentes zonas, lo que fomenta la imaginación y la capacidad creativa sobre su uso. Otro de los aspectos a destacar en este sentido es que se trata de una torreta que dispara pequeños proyectiles de espuma, por lo que un juego clásico asociado es de puntería. El niño tendrá que aprender a posicionar bien los ángulos antes de disparar, lo que le supondrá un cierto esfuerzo intelectual dependiendo de la edad y, además, deberá tener paciencia para mejorar en el manejo a diferencia de otros juguetes donde todo está más automatizado.

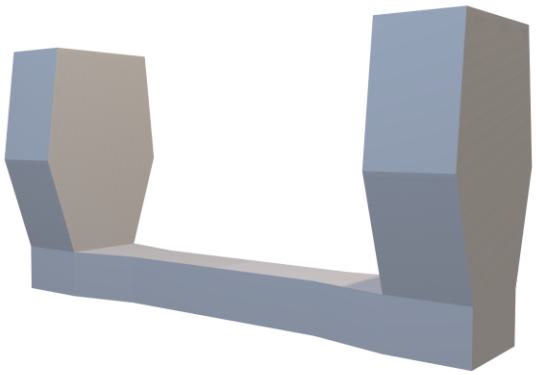
Extra components and 3D pieces

Se han realizado dos modelos de torreta en el proyecto una para ser utilizada en la simulación de Unity y otra para la construcción del objeto físico. Ambos modelos son en 3D y pueden ser imprimidos, sin embargo el modelo usado en Unity es más simple y no se podría implementar en un modelo de robot físico. A continuación se muestra el modelo de Unity:

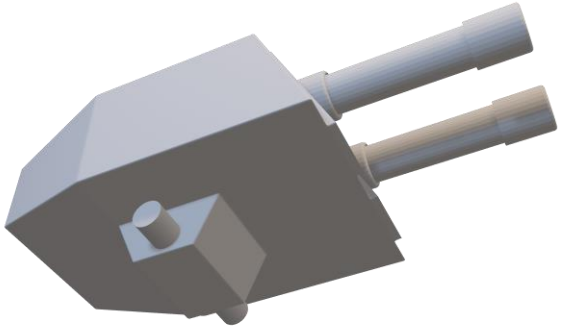
- *PEANA*

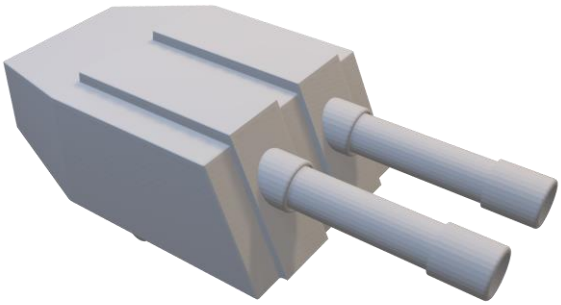


- *SOPORTE*

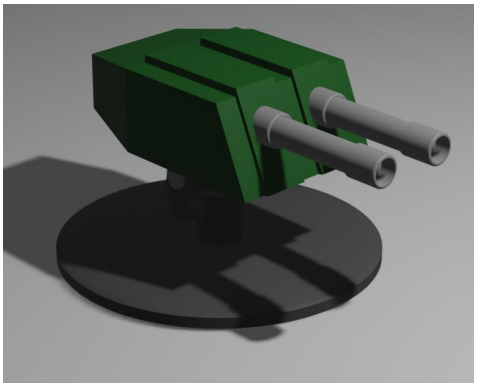


- TOERRETA

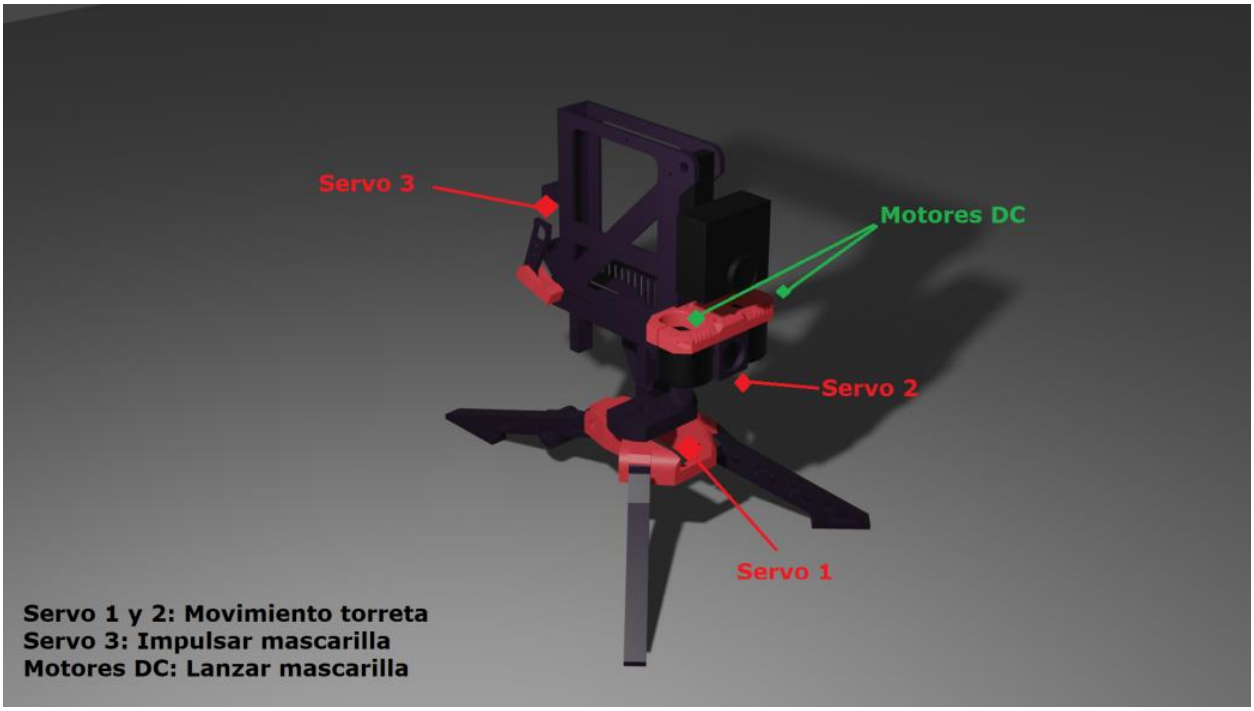




- MONTAJE



Por otro, el modelo 3D de torreta para la construcción física del robot se ha realizado con un total de 13 piezas distintas. Todas las piezas se incluyen en el proyecto y de forma visual el montaje y ensamblaje de las piezas quedaría de la siguiente forma:



Strategy for validation, testing and simulation

Para similar el proyecto se propone dos mecanismos distintos. Por un lado, la parte relacionada con Arduino se realizara en base al programa Proteus, ya que permite realizar el diseño de un esquema y realizar simulaciones directamente con una aplicación. En este sentido, se pretende lanzar la aplicación del programa en Python y realizar una conexión mediante un puerto virtual con el circuito diseñado en Proteus que tiene el diseño y la programación real del Arduino. De esta forma se podrán simular todos los movimientos de los motores, los servos y la correcta comunicación entre la aplicación y el robot. Por lo tanto, con esto se podría simular todo lo relativo a la parte mecánica del robot como si se tratara del objeto físico.

Por otro lado, por lo que respecta a todo la parte de control, cámara, detección de personas o de voz, los módulos se irán testeando e integrando a medida que se programan. Finalmente, se propone simular esta parte programada en un entorno Unity, en este caso primeramente comprobaremos que se ha realizado bien la conexión entre la aplicación de Python (cliente) y Unity (server). Una vez se ha generado correctamente el socket de dicha conexión, lo siguiente que comprobamos será el recibimiento de los mensajes y si su contenido corresponde a una acción válida que debe de simular la torreta como es el caso del movimiento o disparo. Por lo tanto, si los mensajes son correctos el siguiente paso es ver si la torreta realiza el movimiento correcto respecto la accion y informacion recibida en el mensaje. En el caso que realizamos el lanzamiento de la mascarilla tenemos que comprobar que esta impacte con la cara de la persona y active la mascarilla indicando que la ha recibido y no será detectada sin mascarilla de nuevo. En el caso de lanzar a una diana realizaremos el mismo procedimiento que con la persona, pero en este caso una vez impacte la mascarilla debe eliminarse la diana y la mascarilla y volver a generar una diana en el área de aparición.

Para poder generar aleatoriamente las personas y las dianas, dentro de un área hemos tenido que comprobar que se generaban dentro de los rango delimitados marcados y no fuera y también que escoja aleatoriamente las caras de las personas dentro de las posibilidades que tenemos configuradas.

Foreseen risks and contingency plan

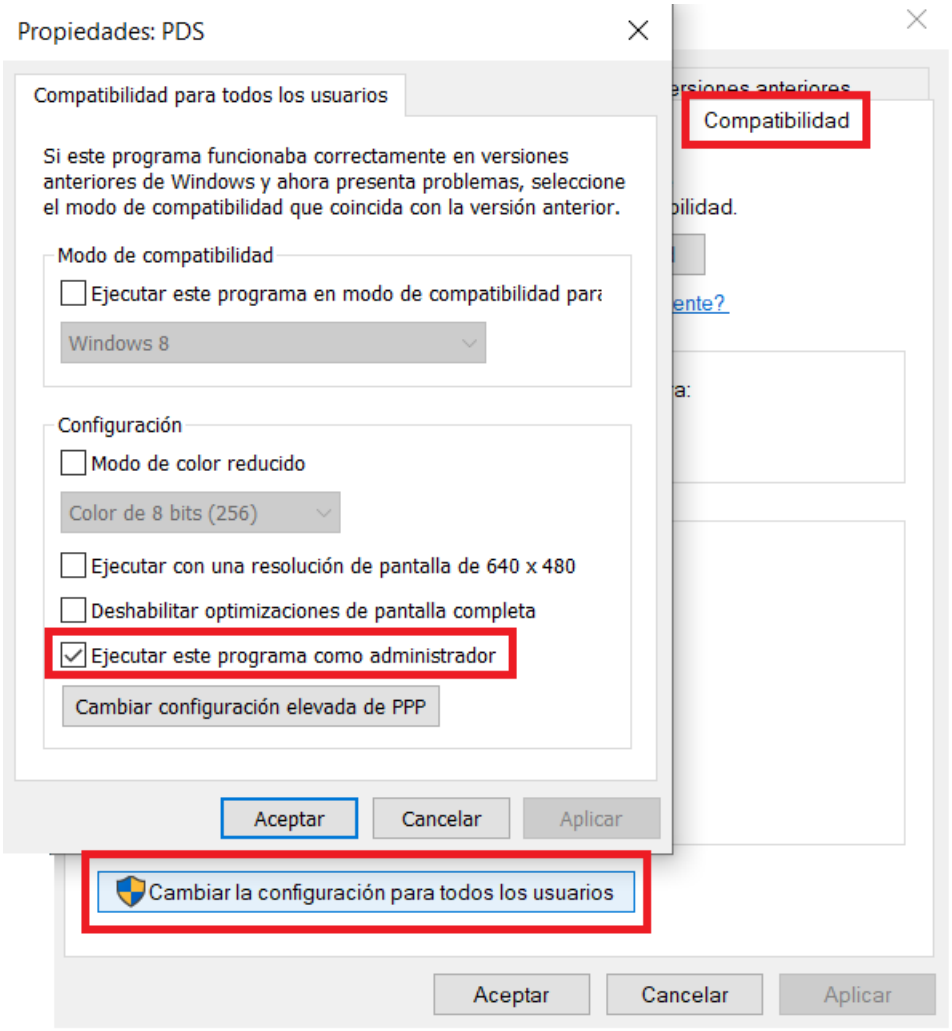
Desaparecen todos los riesgos asociados al Hardware ya que el proyecto se centra en la implementación software.

Risk #	**Description**	**Probability** (High/ Medium /Low)	**Impact** (High/ Medium /Low**)**	**Contingency plan**
1	Fallo en la comunicación	Low	High	Un fallo en la comunicación entre la aplicación y Arduino implica que no se puede realizar nada con el robot. Sin embargo, este problema es muy poco probable y se conocen las formas de establecer comunicación.
2	Fallo reconocimiento facial	Medium	Low	Fallos en los algorismo de reconocimiento facial implicarían una limitación pequeña del robot, ya que simplemente sería incapaz de reconocer si la persona es el dueño o no mientras mantiene el resto de funcionalidades
3	Interacción por voz simple	Low	Medium	Restaría gran parte de capacidad de interacción y formato educativo al juguete, pero es un módulo fácilmente extensible a medida de las necesidades. Por este motivo, se debe programar teniendo en cuenta las posibilidades de expansión.
4	Peligrosidad del jueguete	Low	High	Si el juguete fuera considerado peligroso por las autoridades se puede cambiar el proyectil a más blando posible y bajar la velocidad de los motores para reducir la potencia. El riesgo es muy bajo ya que existen juguetes que disparan proyectiles similares en el mercado y a más potencia calificados a partir de 8 años.

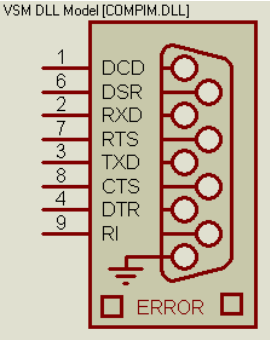
Proteus connection (Arduino simulation)

Para poder simular la comunicación entre una aplicación Python y un Arduino se utiliza el programa Proteus 8.

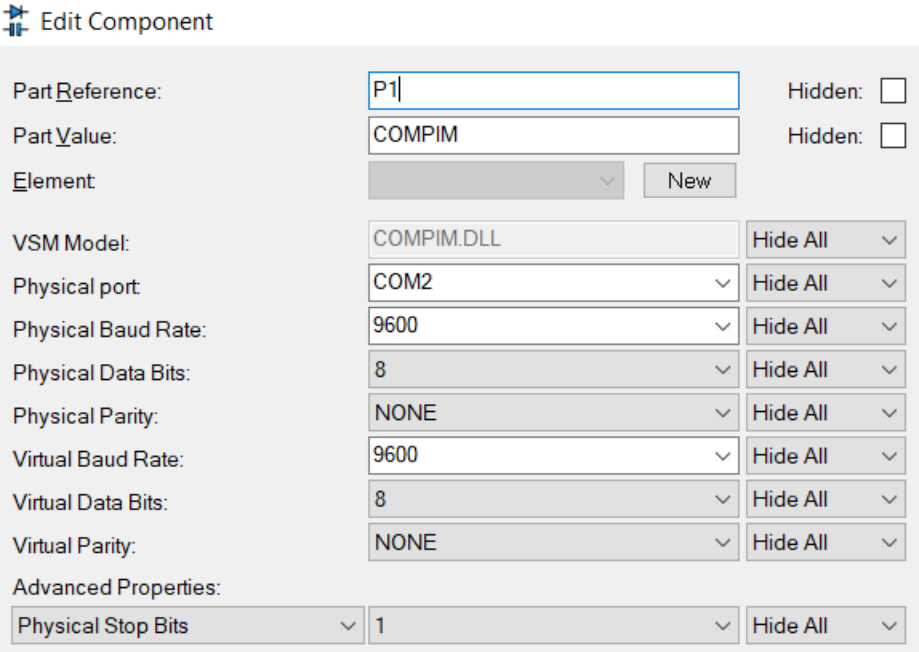
Para que funcione correctamente hay que configurar las opciones dentro de Compatibilidad para que se ejecute como administrador.



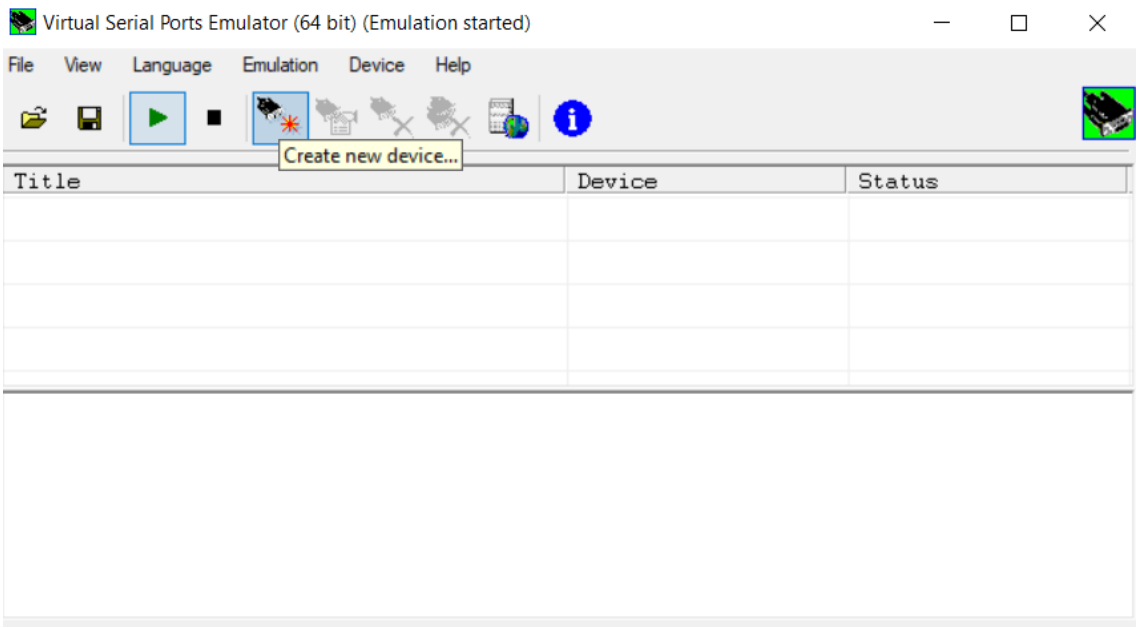
Este programa permite realizar simulaciones con Arduino y establecer comunicaciones entre las aplicaciones. Para ello se utiliza un modelo de puerto físico.



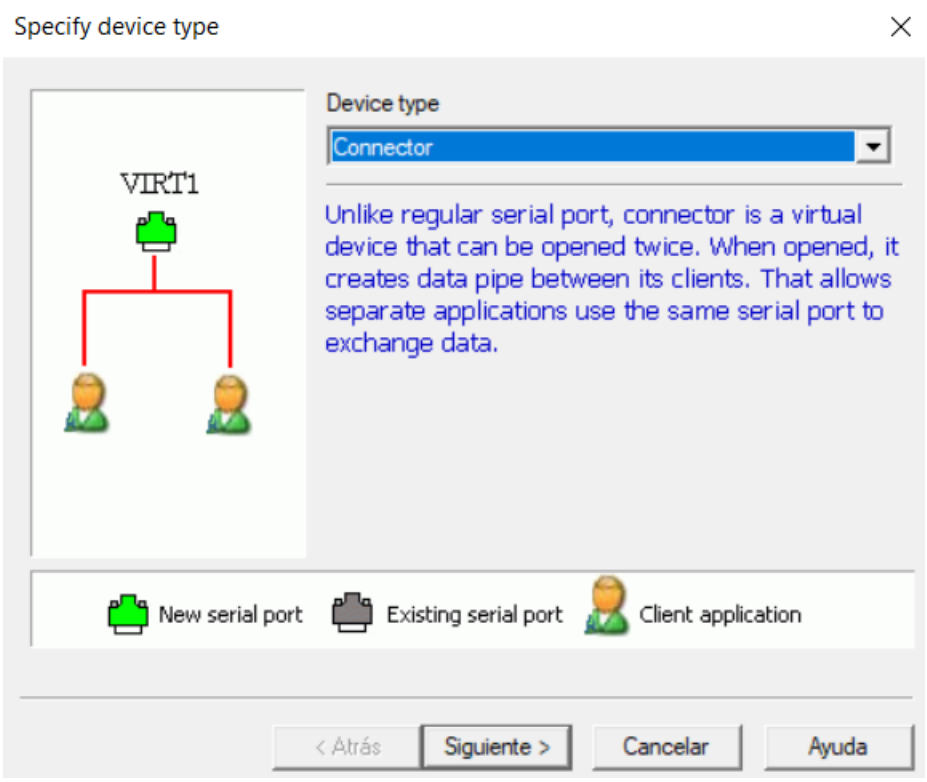
Se configura para que escuche por el puerto COM2 de forma que se pueda establecer una comunicación con la aplicación Python.



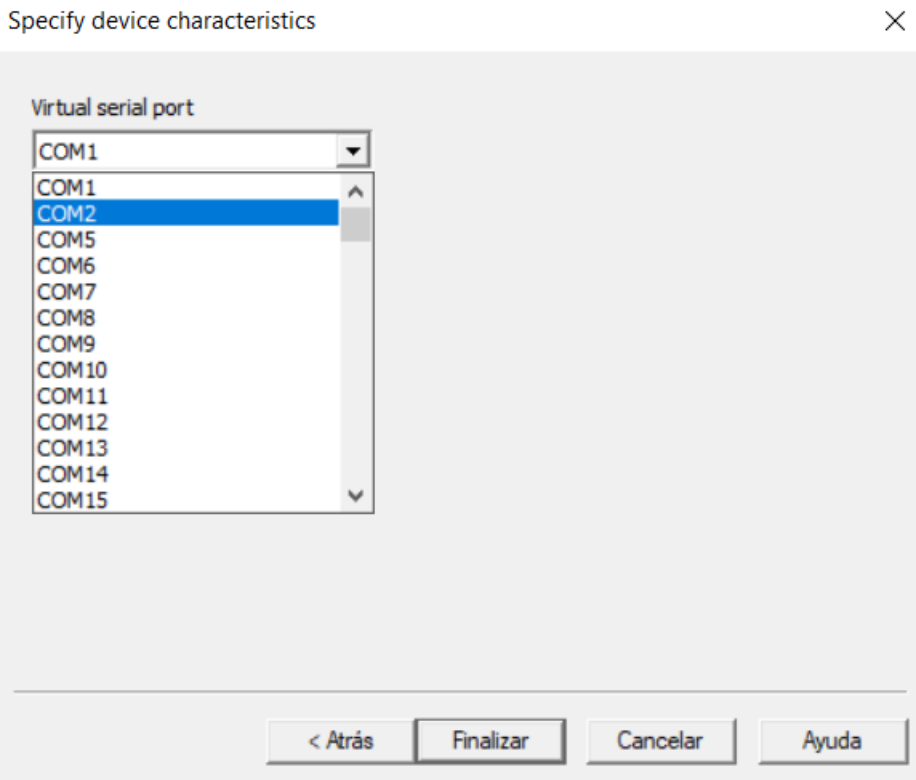
También es necesario descargar librerías de componentes y copiarlas en la carpeta correspondiente. En este caso, se requiere el componente Arduino Uno que no se encuentra en la instalación inicial. Por otro lado, se debe crear un puerto virtual. Para ello se utiliza la aplicación Virtual Serial Port Emulator. Se crea un Puerto virtual nuevo



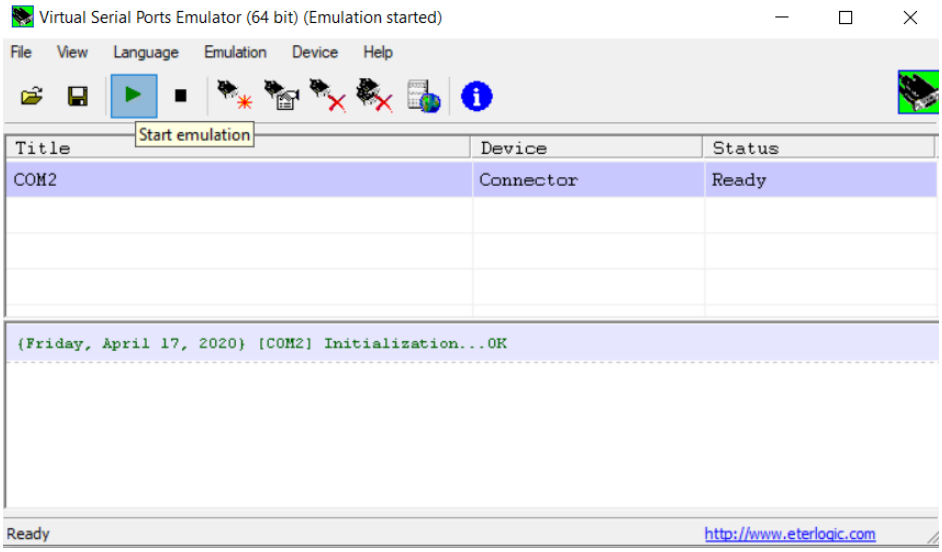
Se tipo conector.



Y en este caso seleccionamos el puerto COM2 que se ha configurado anteriormente para la comunicación.

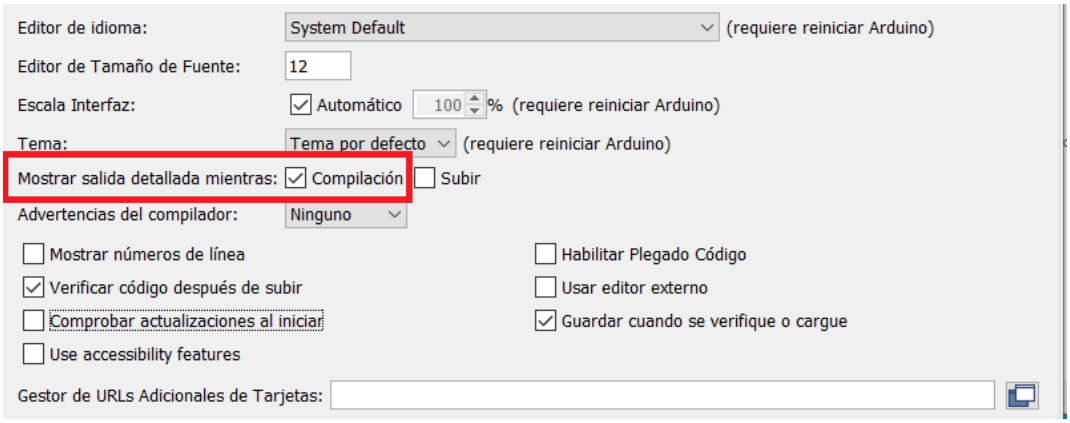


Inicializamos la emulación y ya se podría realizar la comunicación a través de este puerto entre una aplicación en Python y el simulador de Arduino.

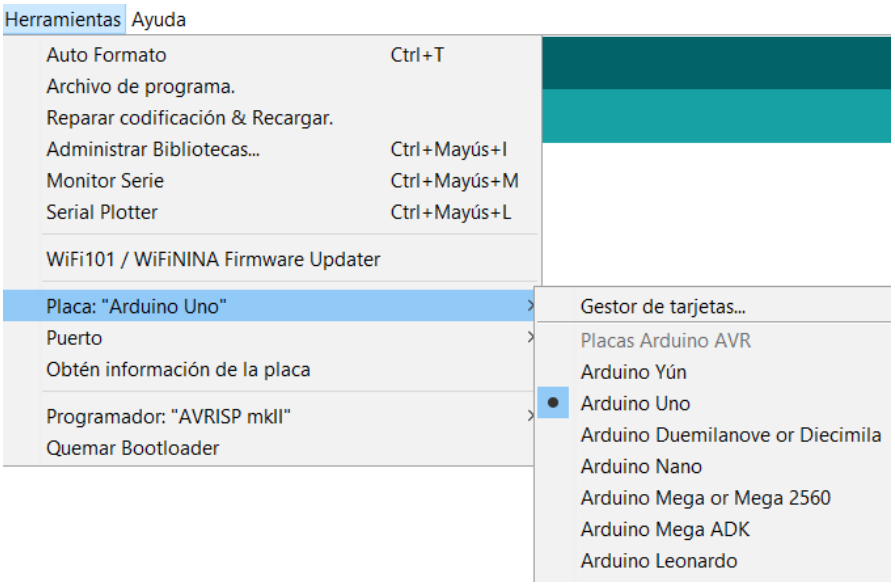


Para realizar simulaciones en Proteus se requieren los archivos hexadecimales (.hex) de la compilación. Por ello, el código Arduino se realiza en el programa Arduino IDE que permite conseguir estos archivos.

En Archivo →Preferencias se debe marcar la opción de mostrar la salida de Compilación



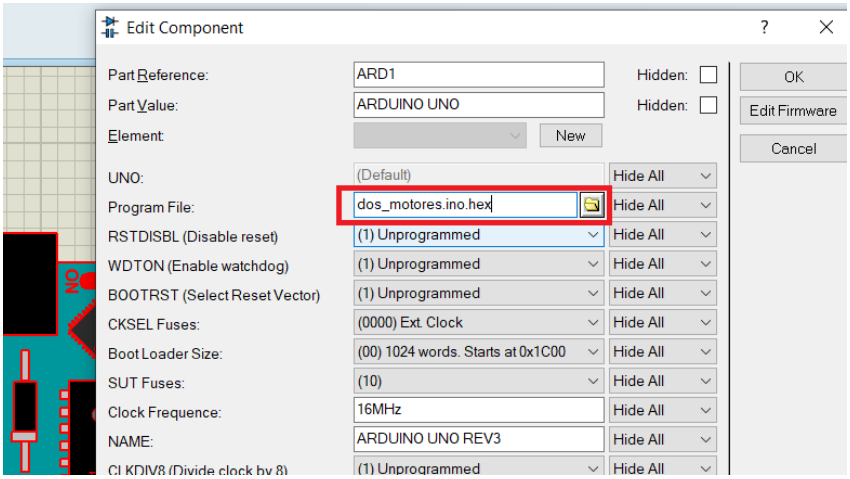
También hay que seleccionar el tipo Arduino que se está utilizando en la simulación para la compilación del código



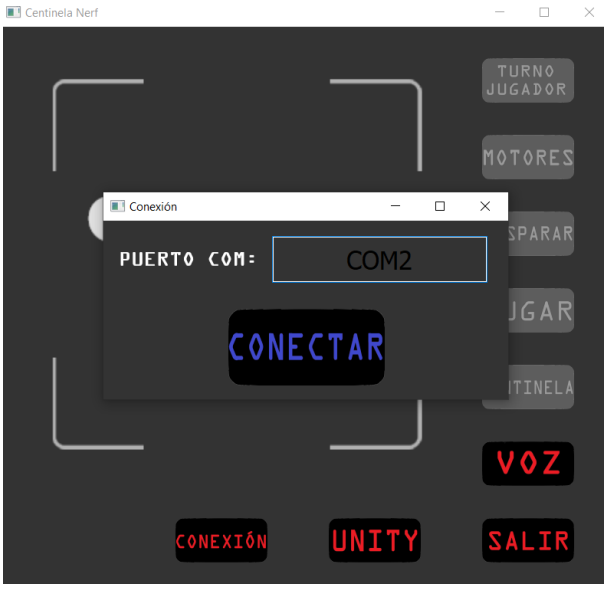
Una vez se compila el programa, se debe buscar el archivo .hex generado que se guarda en una ruta de archivos temporales. Al final de la compilación se puede encontrar la ruta donde se encuentra el archivo de forma que lo podemos guardar.

```
\AppData\\Local\\Temp\\arduino_build_340360/dos_motores.ino.
\\Local\\Temp\\arduino_build_340360/dos_motores.ino.hex"
```

Una vez tenemos el archivo de Arduino hexadecimal lo cargamos en el componente del simulador Proteus para que pueda ejecutarse.



Con esto se puede establecer una comunicación Serial entre Python y el simulador a través del puerto COM2 que se ha definido.



Code and algorithms

Python

En general se han realizado comentarios sobre las líneas del código para facilitar la comprensión de su funcionamiento. A continuación, se expondrá la funcionalidad básica que se desarrolla en cada archivo que integra el proyecto.

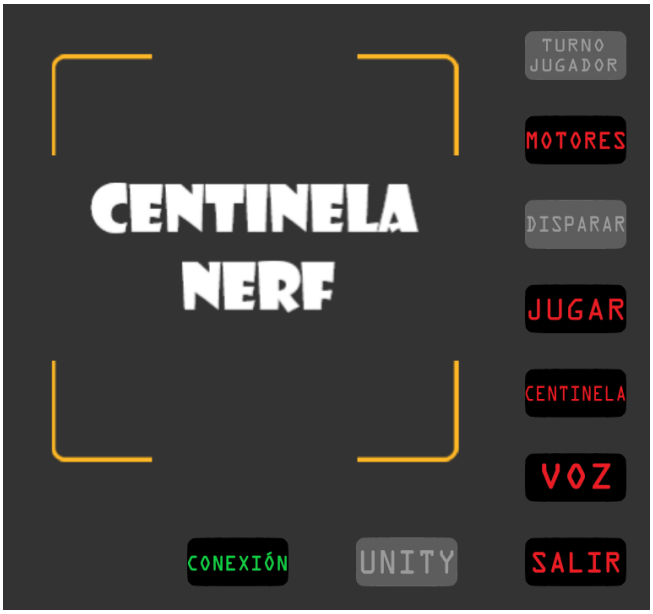
- Control.py
  Hemos creado un pequeño programa con interfaz que nos permite controlar tanto la comunicación como transmisión de mensajes entre los diferentes módulos y una interacción del usuario con el robot. La interfaz se muestra de la siguiente forma:

  Inicialmente las opciones que se pueden hacer es la conexión mediante click de los botones o comando de voz activando la opción de 'Voz' o salir de la aplicación. La opción 'Unity' establece una conexión del software con unity para realizar la simulación del robot y usarlo desde allí. La opción de conexión es la que establece una conexión mediante el puerto COM2 con el arduino. En este caso, se realiza una simulación virtualizando un puerto y conetandolo a Proteus para comprobar el correcto funcionamiento de los componentes físicos.

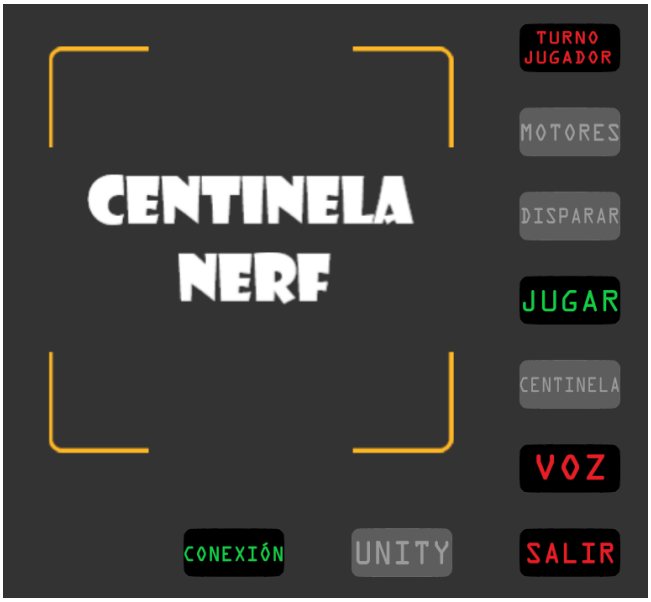
  El control del movimiento del robot se realiza pasando el ratón por encima del pad central de la aplicación. El movimiento del ratón por esta zona se leerá para

sincronizarse con el movimiento del robot. Haciendo un clic derecho estando en el pad de control de robot se realiza un disparo en el caso de que los motores se encuentren encendidos.

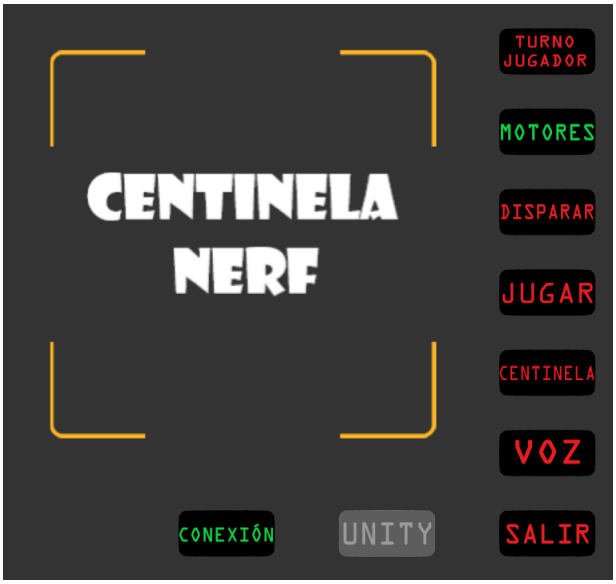
Una vez se ha realizado la conexión con el dispositivo se lanza la captura de pantalla de la cámara, por lo que se implementa un bucle infinito hasta que el usuario decide cerrar la aplicación, que se muestra de la siguiente forma:



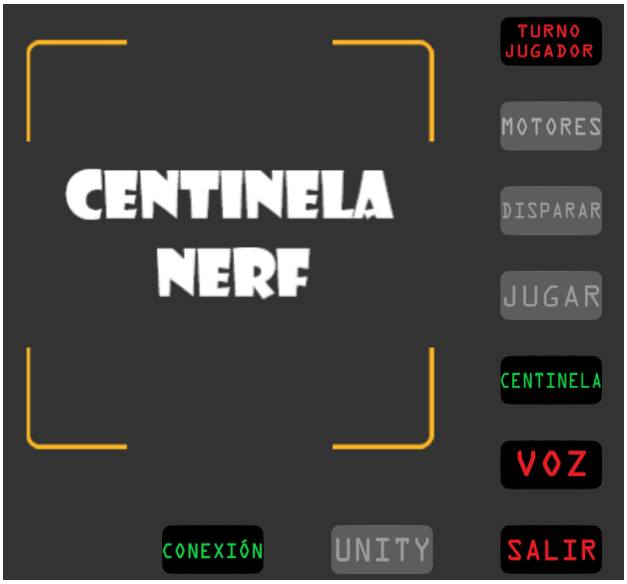
El botón de ‘Turno Jugador’ solo se activa cuando se usa el modo ‘Jugar’ e indica si esta disparando el robot, en color rojo, o si es el turno del jugador, en color verde. El modo ‘Jugar’ enciendo los motores del robot en caso de que estén apagados y busca una diana a la que apuntar para disparar de forma automática. Una vez hecho el disparo, será el turno del jugador para disparar. El resto de opciones se desactivan mientras esté este modo activo.



La opción ‘Motores’ enciende los motores del robot y habilitad la posibilidad de realizar un dispara. Si los motores no están encendidos no se puede disparar.



El modo ‘Centinela’ permite que el robot funcione de forma autónoma, por lo que se deja de tener control a su funcionalidad mientras este modo esté activo. El robot realiza un barrida dentro de su espacio de movimiento en busca de personas, por lo que se implementa un detector facial. Cuando encuentra una persona, detecta si lleva mascarilla o no para lanzarle una mascarilla. Además, cuando ha detectado a una persona envía diferentes mensajes en función de si se lleva mascarilla o no.



El modulo de ‘Voz’ permite activar todas las opciones del robot mediante comandos y se recibe una confirmación de error en caso de no poder realizarse la acción solicitada.

Desde el modulo de control se crean todas las variables necesarias para controlar la aplicación y se activan y desactivan las opciones mediante funciones para cada uno de los botones. También este modulo es el encargado de establecer y mantener la conexión con el dispositivo y enviar los mensajes de comunicación, aunque esta funcionalidad de desarrolla en módulos separados.

- Movimiento y disparo

  El controlador se encarga de conseguir las coordenadas X Y del usuario al mover el ratón sobre el pad de la aplicación. Una vez detectadas estas coordenadas sobre la pantalla se convierten al rango de envió de datos del mensaje (0-253) para que el Arduino las sepa interpretar y convertir en ángulos de los servos. El disparo se implementa con un click encima del pad. Para que el disparo se efectue los motores del robot deben estar encendidos. Mediante el

programa Proteus, podemos simular la parte del arduino en la cual tenemos conectados todos los servos y motores DC que forman nuestra torreta. Según las órdenes que recibe por parte del controlador moveremos los servos y también podemos activar los motores DC para realizar la acción de disparar.

- Motores

Los motores se encienden y apagan mediante el botón de la aplicación. La velocidad a la que giran los motores puede variarse para cada uno de ellos de forma individual en el esquema implementado, sin embargo no pude ser controlado por el usuario. Para encender o apagar los motores hay que hacer uso de un booleano e indicar la velocidad en el mensaje que se envía al Arduino. El Arduino leerá el mensaje recibido y encenderá o apagará los motores con la velocidad señalada.
El control de velocidad se establece a través del Pin Enable conectados a salidas analógicas. De esta forma, la velocidad del motor se puede variar en el rango 0 a 255, que es la velocidad máxima. Esta velocidad se controla desde la aplicación Python, aunque no es accesible para el usuario. Cuando el usuario cierra la aplicación con el botón Exit los motores se apagan de forma automática.

- Comunicación Serial Python-Arduino
  Se crea una función para establecer una comunicación por Arduino.
  - Conectar.py: Mediante un puerto virtual, este módulo crea una conexión la simulación del arduino. En primer lugar comprobamos si podemos establecer la conexión, en el caso de ser positivo creamos la conexión sino enviamos un mensaje de que no se ha podido conectar con el arduino. También encontramos la función de desconectar para cerrar la conexión la establecida.
  - Transmitir.py: Este módulo nos permite enviar y recibir mensajes con el simulador de arduino (Proteus). Para enviar mensajes se sigue la siguiente estructura de datos a nivel de byte:

    [byte inicio, eje x, eje y, motor on/off, velocidad, disparo, byte final]
    - byte inicio = 255
    - eje x, arriba y abajo = entre 0 y 253
    - eje y, movimiento lateral = entre 0 y 253
    - motor on/ff = 0 apagar - 1 encender
    - velocidad = entre 0 y 253
    - disparo = 0 no disparar - 1 disparar
    - byte final = 254

En caso de ser necesario se ampliaría esta comunicación añadiendo más elementos al mensaje pero conservando el mismo mecanismo y estructura.

- IdentificarDiana.py
  En este módulo gracias se hace uso de la librería OpenCV para detectar una diana. El método empleado para realizar la detección se basa en el detector de contornos de Canny. Para ello, el frame original de la imagen se pasa escala de gris

Y s aplica el detector de contornos de Canny con un elemento estructurante de 5x5 en forma de cruz. De esta forma, se obtienen los contornos de la imagen

Finalmente se realiza una dilatacion para expandir los contornos.

Una vez la imagen está tratada se puede aplicar un detector de círculos sobre la imagen para identificar la dina. En este sentido, se desarrollan dos algoritmos distintos, uno que detecta los contornos para determinar los círculos y otro basado en la transformada de Hoguh para detectar círculos. Con esto, se puede determinar la posición del círculo y su centro para poder pasar la información al robot y que pueda apuntar a la diana. A modo visual se pone una capa de los datos que se detectan para poder comprobar la funcionalidad del algoritmo.

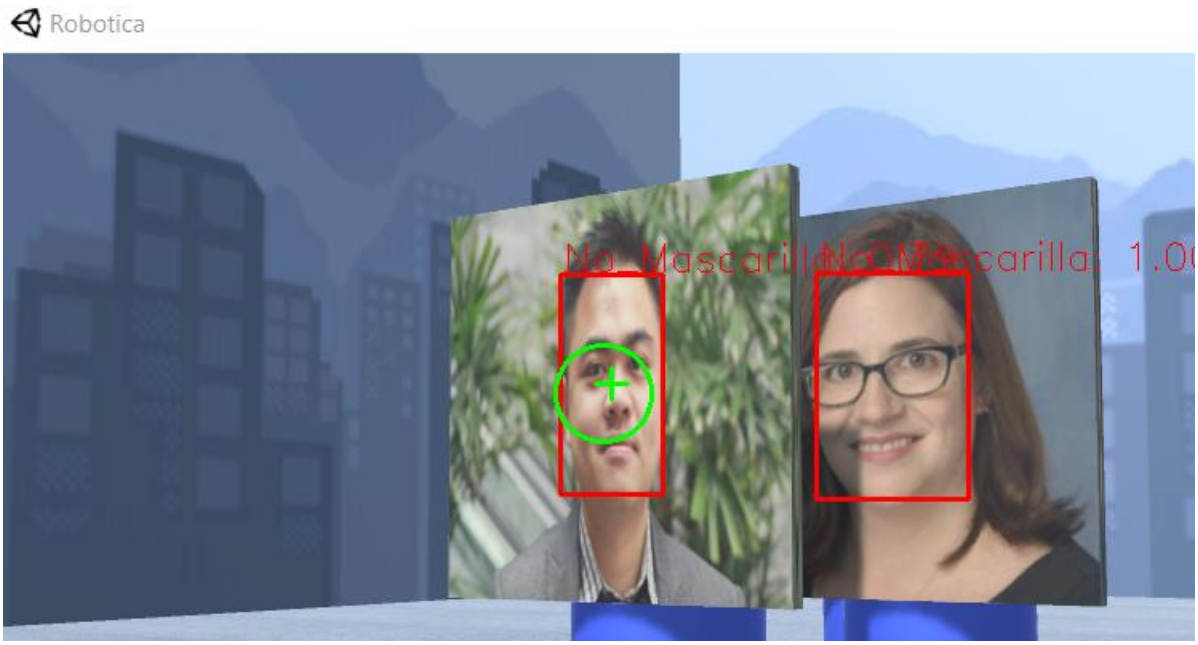
- Rec_facial_mascarilla: Para poder realizar la detección de mascarilla se debe realizar primero un reconocimiento facial para detectar las caras de una imagen y después determinar si la persona lleva mascarilla para poder actuar de forma apropiada. En este caso se utiliza un modelo de red neuronal preentrenado en pytorch que permite el reconocimiento de caras y de mascarilla con gran precisión hasta más de 4 metros de distancia de la cámara. A partir de la información que se obtiene de aplicar la red al frame, se calculan diferentes datos como la distancia que hay del centro de la cara detectada al centro de la imagen, que es donde apunta el robot. Toda está información se procesa y se devuelve al módulo 'Centinela' que en base a si se ha detectado o no una cara, si lleva mascarilla y la posición a la que se encuentra emitirá mensajes sonoros y realizará movimientos necesarios llegando a lanzar una mascarilla. A modo

visual se pone una capa de los datos que se detectan para poder comprobar la funcionalidad del algoritmo.

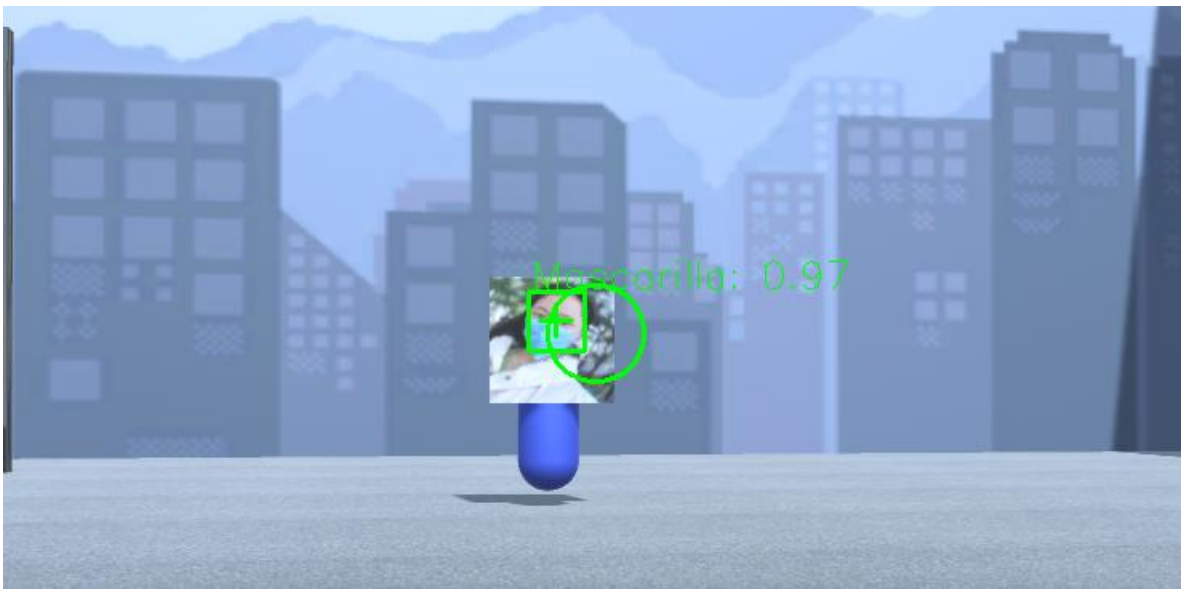
- Speech:
  El módulo de reconocimiento de voz o speech está formado por dos funciones base hablar y responder y la llamada al reconomiento de voz de Google cloud. Esto permitirá una interacción mediante la voz del usuario con la torreta y de la torreta con el usuario. El módulo principal que se encarga de la tarea es speech_recognition.py.

  - Reconocimiento_voz → se hace uso de Google.cloud.speech para realizar el reconocimiento de voz de forma rápida y precisa. Cuando se activa la opción de voz se llama a este modulo que realiza una conexión al servidor cloud para procesar el mensaje. Una vez se obtiene el mensaje se responde a la función speech_recognition.py con el código del mensaje detectado para que procese la petición del usuario.

  - Hablar: esta función utilizaremos las librerías playsound para ejecutar los archivos de audio , gTTs una librería de google que nos permitirá convertir el texto en audio y por último os para eliminar el archivo creado una vez ejecutado. Recibirá por parámetro el string con el texto a convertir. Este texto será convertido mediante gTTs en audio, el cual guardaremos para ser reproducido posteriormente. Una vez reproducido será eliminado para no tener residuos de audios pasados.
  - Responder: este última función se encarga de procesar el código recibido de la detección de Google cloud para responder de forma adecuada al usuario y realizar las acciones que se han pedido.
- Centinela.py
  Esta función se encargar de establecer el modo Centinela del robot con la clase modoVigilar. Si está activada, el usuario pierde el control de movimiento y disparo del robot que funciona de manera autónoma. El robot usa la cámara para detectar la presencia de personas y reconocer si es el dueño o un desconocido. Si detecta una cara en su campo de visión, el robot se moverá hasta centrar a esa persona en la imagen. Por lo tanto, perseguirá los movimientos de las personas hasta que queden centradas. A continuación, se muestra la detección de dos caras sin mascarilla en color rojo. El circulo central señala cuando el objetivo esta centrado. En caso de estar centrado aparecerá en verde y si no se tiene ningún objetivo centrado en rojo. Cuando el objetivo está centrado y no lleva mascarilla el robot lanza una mascarilla.



Si se detecta una cara con mascarilla el recuadro sobre la cara es de color verde. Además, los mensajes que el robot dice son diferentes en ambos casos.



En el caso de que no se detecte ninguna persona en la imagen, el robot realizara movimientos aleatorios de vigilancia hasta detectar una nueva cara. También se controlará el algoritmo del modo de juego, aunque la llamada a las funciones que

realizan la detección es distinta, el movimiento del robot para fijar el objetivo es el mismo en los dos casos.

- Finalmente, hay una carpeta 'Old_code' con los algoritmos de reconocimiento facial inicialmente planteados antes de la reconversión del proyecto a lanzar mascarillas. Se ha usado uno de los algoritmos más usados para reconocimiento facial, el algoritmo Local Binary Patterns Histograms (LBPH) que está integrado en la librería openCV de python. Està dividido en dos archivos, el primero serà el que hace el entrenamiento de la red y el segundo que será el que reconozca a tiempo real si es el dueño. Además, se añade a este modulo el cálculo de la distancia a la que se encuentra la cara detectada respecto del centro. Con esta distancia se podrá mover los servos del robot para que persiga la cara y la centre.
    - Entrenamiento: Para hacer el entrenamiento necesitamos crear el dataset. Este será construido a partir de imágenes etiquetada del dueño y imágenes de personas sacadas de internet para distinguir al dueño. Esta parte genera un archivo donde se almacenarán los resultados del entrenamiento.
    - Reconocimiento: En esta parte se coge cada frame de la webcam, se recorta la parte de la cara y se pasa esta parte de la imagen donde sale la cara a la función que predice previamente entrenada y si coincide con la etiqueta del dueño no se disparará.

Unity

Para realizar la simulación física del robot, hemos utilizado Unity para crear diferentes entornos en los que simular el funcionamiento. La arquitectura interna del simulador se divide en los módulos de conexión que nos permitirá conectar la aplicación con el simulador, movimiento de la torreta que se encargará de aplicar el movimiento recibido a la torreta simulada, detección colisión para gestionar cuando una mascarilla es entregada o no a una persona y por último los dos módulos de reaparición objetos y movimiento de personas que nos permiten crear diferentes entornos para la simulación

Conexión - Sockets.cs
Nuestra aplicación base está creada en python y necesitábamos una conexión simultánea entre la aplicación y la simulación para que los movimientos fueran fluidos. Es por esto que la conexión ha sido creada mediante sockets.

El simulador se encarga de levantar un Servidor y recibir las peticiones del cliente mediante el puerto 13000. Una vez la aplicación envía la petición de conexión al simulador, este la acepta el nuevo cliente y crea un thread para recibir las peticiones de este. En este punto el servidor recibirá los mensajes y estos serán tratados, cada mensaje estará compuesto por la letra m para movimiento y d para disparo seguido de los parámetros necesarios para hacer la acción. En el caso de querer enviar más de una acción por mensaje estos deberán ser separados por “&”.

Por cada mensaje de cada transmisión separamos la acción de sus parámetros mediante un “@”. Una vez cada mensaje de la transmisión ha sido tratado añadimos a una cola cada acción con sus parámetros correspondientes para su posterior ejecución por el módulo de movimiento torreta. De esta manera evitaremos que si llegan varias peticiones en tiempos muy cortos el sistema se colapse y pueda gestionar todas esas acciones en el orden que han sido recibidas.

Ejemplos de mensajes :

- Movimiento "M@100,50"&"
- Disparo "D@1&"

Movimiento torreta - ControlTorreta.cs

Este módulo es esencial para el correcto funcionamiento de la simulación, es el encargado de gestionar los inputs de movimiento que recibimos de la aplicación y aplicarlos sobre la torreta. Para ello este script está relacionado con la torreta y cuando el cliente (aplicación Python) envía el mensaje con el movimiento o lanzar mascarilla el módulo lo traduce y aplica los parámetros correspondientes sobre la torreta. Hay que recordar que unity no realiza ningún movimiento por su cuenta, los movimientos de apuntar, seguimiento y disparo son todos enviados por la aplicación cliente y la simulación se encarga de aplicarla.

Para simular la acción de lanzar la mascarilla en un entorno real hemos añadido una fuerza a la mascarilla en el ángulo en el que se encuentre mirando la torreta simulando la fuerza que aplican los dos motores DC. Las mascarillas que son lanzadas por la torreta tienen una masa que se ve afectada por la gravedad, de esta manera conseguimos darle un toque más real a la simulación y podremos observar como su la trayectoria del lanzamiento realiza un movimiento de parábola.

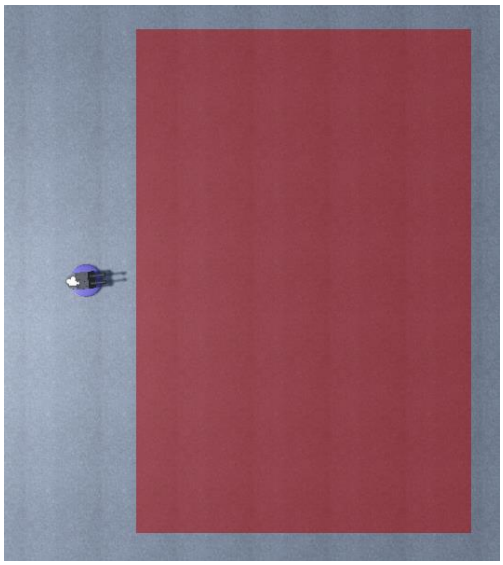
Detección colisión - Mascarilla

Para detectar que la mascarilla colisiona con la cara de la persona, hemos creado este script que nos permite controlar si el objeto con el que ha colisionado es una cara en el caso de que sea afirmativo el módulo se encarga de activar la mascarilla en la persona para que sea visible. De esta manera el robot podrá controlar si el objetivo al que le ha lanzado la mascarilla lo ha recibido porque detectará que ahora lleva la mascarilla puesta y habrá reducido la propagación o contagio del covid-19.

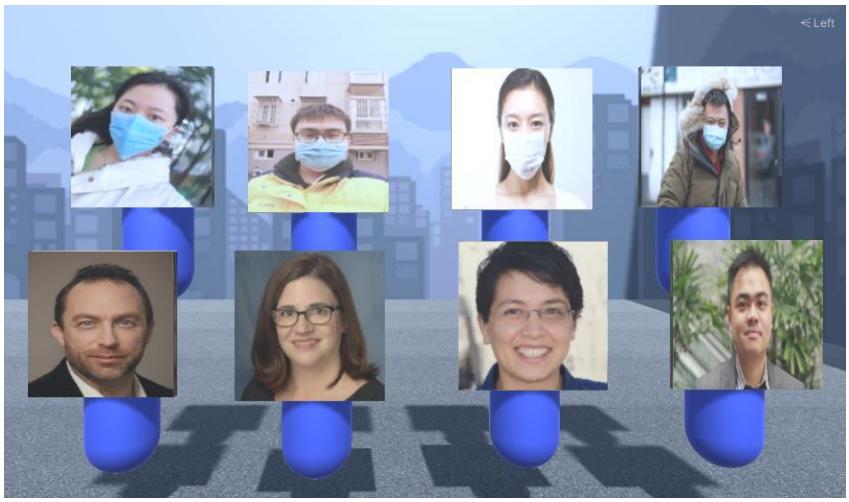
Reaparición Objetos - SpawnObjetos.cs

Nuestro robot cuenta con dos contextos diferenciados, el modo centinela en el que se encarga de buscar si alguna persona no lleva mascarilla para lanzarle una y el modo juego en el que el robot lanza las mascarillas contra dianas.

Con estos dos contextos necesitamos poder cambiar el entorno de alrededor de la torreta según el modo en el que esté la torreta. Es por esto que hemos creado un script que nos permite hacer aparecer a las personas o la diana de forma aleatoria en un área delimitada manualmente para que todos los objetivos estén al alcance. En la siguiente imagen podemos observar en color rojo la zona delimitada para el respawn de objetos.



En el modo centinela, generamos un grupo de X personas distribuidas por el espacio, donde cada persona tendrá una cara elegida al azar entre un grupo de 8 imágenes donde 4 son con mascarilla y las restantes sin, para poder tener personas que lleven y otras que no lleven mascarilla y tener un entorno más real.



El modo jugar, se encarga de crear una única diana en un lugar al azar de todo el área de respawn. En este modo tendremos en cuenta que una vez impacte una mascarilla contra la diana esta desaparecerá y aparecerá una nueva en otro lugar al azar del terreno marcado para el respawn.

- Movimiento Personas - PersonaPatrulla.cs

Hemos querido simular el movimiento de las personas caminando con el objetivo de que la simulación sea más realista y poder mostrar el seguimiento del robot cuando una persona no tiene mascarilla, para ello las personas que aparecen en el área de respawn inician una rutina desplazamiento a una velocidad constante hasta una pared y después reanuda su movimiento en dirección contrario hasta la siguiente pared. Este movimiento lo realizan de manera constante y se realiza en bucle para que siempre estén en movimiento y parecer una simulación más real.

- Interfaz simulación

La interfaz de simulación es una parte muy importante de la simulación ya que nos permite comprobar en todo momento el estado en el que se encuentra la torreta y si los movimiento que realizan son los correctos y nos sirve como input para la aplicación capturando lo que estaría observando la cámara de la torreta en estado real.

Durante la simulación en Unity la interfaz que podremos observar será la siguiente: en la parte superior izquierda podemos ver lo que está viendo la torreta gracias a su cámara situada encima. Esta cámara será la que capturamos con nuestra aplicación en Python y nos permitirá

realizar todo el proceso de detección de las caras de las personas que llevan mascarilla o no y de las dianas si estamos en modo jugar.

En la parte inferior derecha podremos ver la torreta desde una vista posterior para poder ver como realiza los movimientos y el estado en que se encuentra.

References

This project has been inspired by the following Internet projects:

https://www.littlefrenchkev.com/bluetooth-nerf-turret

Others projects:

https://www.electronicshub.org/robotics-projects-ideas/

https://www.youtube.com/watch?v=Lp8eSCuQqb0