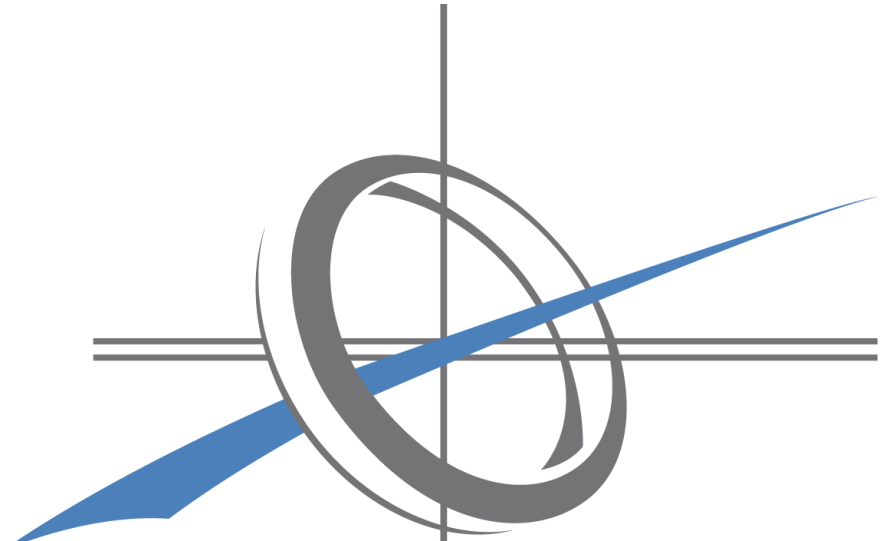


Universidad Politécnica de Madrid

ETSI SISTEMAS INFORMÁTICOS

Proyecto: Five Nights at IA's
Asignatura: Sistemas Inteligentes
Escuela: Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

ÍNDICE

1. INTRODUCCIÓN

Nuestro proyecto consistirá en la creación de dos modelos de inteligencia artificial, uno basado en la detección de imágenes y otro en aprendizaje reforzado, sobre un repositorio de GitHub realizado por el usuario EDUATO que consiste en una recreación del juego Five Nights At Freddy's 2.

El modelo de detección nos servirá para detectar a los enemigos (llamados de aquí en adelante animatrónicos) a partir de un frame del juego.

El modelo de aprendizaje conseguirá en completar una noche en el videojuego apoyándose en el modelo de detección

▪ INICIAR PROYECTO

El videojuego esta creado en **Python 3.10.8,** por lo que será necesario tener instalada esta versión o una superior para un correcto funcionamiento.

Para conseguir iniciar el proyecto, primero se deberán instalar todas las librerías necesarias. Para ello tenemos un archivo **requirements.txt** y para instalar las librerías se deberá situar la terminal de comandos en la carpeta principal del juego e introducir el siguiente comando:

```
pip install -r requirements.txt
```

Una vez descargadas las librerías necesarias se deberá ejecutar el archivo **main.py** situado en la carpeta principal.

▪ CONSIDERACIONES PREVIAS

El videojuego es un survival horror[1], en el que si no conseguimos defendernos de los animatrónicos estos finalizaran la partida realizando un jumpscare[2]

En el **modo IA**, del que hablaremos posteriormente, el modelo de aprendizaje reforzado intentará evitar que nos ataquen con un jumpscare los animatrónicos, pero no es seguro que lo consiga, por lo que deberemos estar atentos para no asustarnos.

En los otros modos depende del usuario que los animatrónicos nos ataquen.

Para evitar asustarnos recomendamos bajar el volumen del videojuego.

2. INTEGRANTES DEL GRUPO

Proyecto	Apellido	Nombre
Five Nights at IA's	VIEDMA GONZÁLEZ	JORGE
	RODRIGUEZ ESCUDERO	LUIS CARLOS
	VALERIANO MARIN	JORGE

[1]Survival horror: subgénero de los videojuegos de terror y de acción y aventura que se centra en el personaje y su travesía por la sobrevivencia, mientras al mismo tiempo se intenta asustar a los jugadores con diferentes elementos del terror

[2]Jumpscare: momento repentino y sorpresivo diseñado para asustar o sobresaltar al jugador.

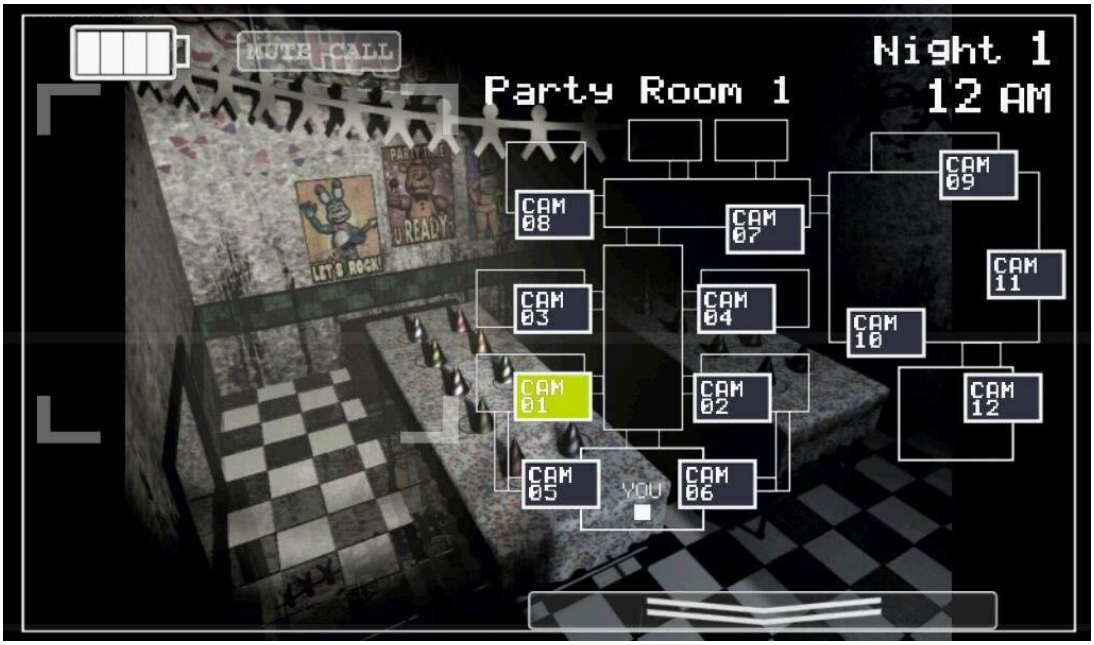
3. INFORMACIÓN SOBRE EL VIDEOJUEGO

Five Nights at Freddy's 2 se trata de un videojuego de supervivencia en el cual debes conseguir pasar 5 noches como vigilante de seguridad en una pizzería. Habrá diferentes animatrónicos situados en varias salas del restaurante y nuestra misión será que durante la noche no consigan atracarnos ni que se nos acabe la batería.

Las noches transcurren desde las 12AM hasta las 6AM (cada noche dura en torno a 7 minutos).

El jugador se encuentra en la oficina, desde la cual podrá realizar las siguientes acciones:

- Mirar al pasillo
- Mirar las ventilaciones que están a la izquierda y derecha del jugador
- Mirar el monitor, donde se podrán ver las diferentes cámaras repartidas por el restaurante
- Ponerse la máscara
- Encender las diferentes luces



Como se puede ver hay un total de doce cámaras, en la parte superior derecha se puede visualizar la hora de la noche "12 AM" la energía actual y además el lugar dónde nos encontramos. Cómo se puede ver estamos en la cámara 1 la cual enfoca a la sala "Party Room 1".

▪ CÁMARAS

Las cámaras enfocaran a las diferentes salas del restaurante, como se puede observar en la siguiente tabla.

En la tabla además supondremos que la cámara 0 es la oficina, lo cual nos será muy útil para posteriormente crear los modelos de inteligencia artificial.

LOCALIZACIÓN	CÁMARA
Oficina	0
Party Room 1	1
Party Room 2	2
Party Room 3	3
Party Room 4	4
Left Air Vent	5
Right Air Vent	6
Main Hall	7
Parts & Service	8
Show Stage	9
Game Area	10
Prize Corner	11
Kid’s Cove	12

▪ ANIMATRÓNICOS

Cada animatrónico tendrá una ruta personalizada, la cual siempre acabará en la oficina.

Puppet

Se encuentra en la cámara 11 (Prize Corner). Para este personaje nos tenemos que encargar de darle cuerda a la caja de música situada en la sala.

En el caso de que se pase el tiempo de la caja de música atacará y terminará la partida, es por ello por lo que cada pocos segundos debemos ver la cámara 11 y dar cuerda a la caja de música

Toy Freddy

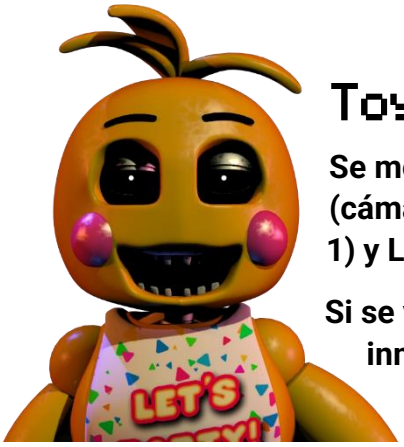
Se encuentra en la cámara 9 (Show Stage) y en la cámara 10 (Game Area).

Si se encuentra en la oficina se debe usar la máscara para que no ataque y la partida termina

Toy Bonnie

Se podrá ver en la cámara 9 (Show Stage),la cámara 3 (Party Room 3) la cámara 4 (Party Room 4), la cámara 2 (Party Room 2) y la cámara 6 (Right Air Vent).

En el caso de ver a Toy Bonnie en la ventilación o la oficina tendremos que ponernos la máscara inmediatamente para que no ataque y se acabe la partida



Toy Chica

Se moverá por la sala Show Stage (cámara 9), Main Hall (cámara 7), Party Room 4 (cámara 4), Party Room 1 (cámara 1) y Left Air Vent (5).

Si se ve en la ventilación o la oficina poner la máscara inmediatamente al igual que con Toy Bonnie.

Mangle

Se podrá ver en la mayoría de las salas del restaurante.

Si se ve en la ventilación derecha o en la oficina, poner la máscara inmediatamente para que no ataque y se termine la partida.

Balloon Boy

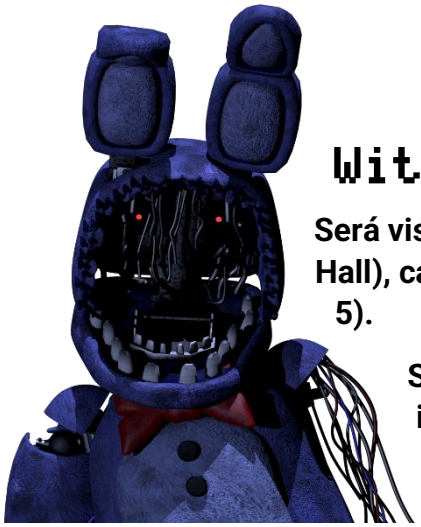
Se moverá por Game Area (cámara 10) y de ahí a Left Air Vent (cámara 5).

Si aparece en la ventilación y no se pone la máscara a tiempo, no nos atacará, pero nos desactivará el uso de luces por un tiempo.

Withered Chica

Se moverá por Parts & Service (cámara 8), Party Room 4 (cámara 4), Party Room 2 (cámara 2) y de ahí a Right Air Vent (cámara 6).

Si aparece en la oficina poner la máscara inmediatamente.



Withered Bonnie

Será visible en la cámara 8 (Parts & Service), cámara 7 (Main Hall), cámara 1 (Party Room) y de ahí a Left Air Vent (cámara 5).

Si aparece dentro de la oficina poner la máscara inmediatamente para que no ataque.

Withered Freddy

Se moverá por Parts & Service (cámara 8), Main Hall (cámara 7) y Party Room 3 (cámara 3).

Si aparece en la oficina poner la máscara inmediatamente para continuar la partida.

Withered Foxy

Si aparece en el pasillo de la oficina habrá que enfocarle con la linterna hasta que desaparezca para que no nos ataque.

Según la defensa que hay que realizar por cada animatrónico definimos 3 modos de defensa:

	Poner máscara	**Dar cuerda a la caja de música**	**Enfocar pasillo**
Animatrónicos afectados	Toy Freddy Toy Bonnie Toy Chica Balloon Boy Withered Chica Withered Freddy Withered Bonnie	Puppet	Foxy

Saber diferenciar los modos de defensa nos será muy útil a la hora de crear el modelo de aprendizaje reforzado.

Los animatrónicos **withered** podrán aparecer de repente en la oficina y quitarnos el monitor de la pantalla, teniendo que ponernos la máscara en un plazo de pocos segundos para que no nos ataquen.

4. CAMBIOS PREVIOS

Lo realizado en este apartado se podrá observar en los archivos *'files/app_main.py' y 'files/menu/options.py'*

Antes de empezar con la creación de los dos modelos de inteligencia artificial, cambiamos el código fuente para crear un menú de opciones básico, en el cual presionando las teclas '1', '2' o '3' se pueden elegir las opciones:

- ### 1. MODO IA

  En este modo de juego se usará el algoritmo de aprendizaje reforzado para conseguir que se pase una noche del videojuego.

- ### 2.JUEGO GUIADO

  En este modo será el usuario el que interactúe con el videojuego, pero tendrá la ayuda del algoritmo de detección para ver a los animatrónicos.

- ### 3. MODO NORMAL

  Juego con el código fuente original sin ningún cambio ni ayudas de inteligencia artificial.

Además de la creación de un nuevo menú, modificamos el anterior código para conseguir que tras elegir una de las opciones nos lleve directamente a la noche 4.

El cambio entre control por IA, por usuario o usuario por detección, se consigue gracias a las variables en la clase App **'ia_control'** y '**only_detection'.**

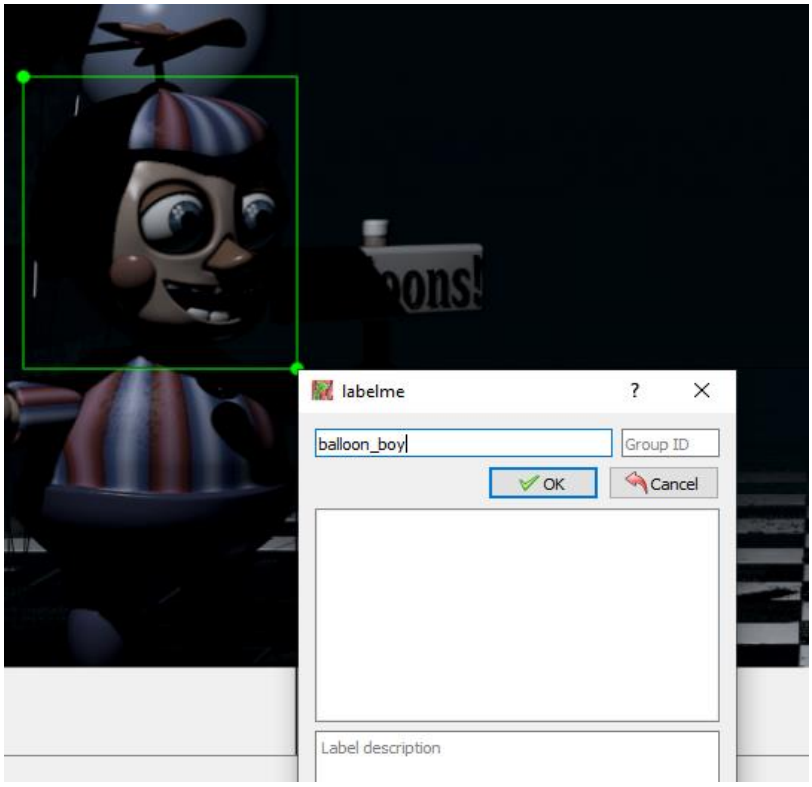
5. MODELO DE DETECCIÓN DE ANIMATRÓNICOS (ENEMIGOS)

• CREACIÓN DEL MODELO

Lo realizado en este subapartado se podrá observar en el path *'files/entrenamiento_modelo_deteccion'* , el archivo colab *crear_modelo_deteccion.ipynb y el modelo YOLOv8 'yolov8m-seg.pt'*

Para realizar este proyecto primero se realizó un modelo para la detección de los animatrónicos.

Para crear el modelo en primer lugar hemos obtenido alrededor de 300 imágenes de los animatrónicos. Por cada una de estas imágenes, hemos usado la librería ***labelme***, la cual permite etiquetar imágenes. Para poder etiquetar a los animatrónicos en las imágenes definimos su posición con un rectángulo y lo etiquetamos con su nombre



Uso de la librería labelme

Por cada imagen etiquetada crea un JSON, el cual hace referencia a la imagen y las coordenadas donde se ubica la etiqueta (el animatrónico).

Cuando ya tenemos todos los JSON de cada imagen, usamos la librería **Labelme2yolo** para crear el DataSet.

Usamos ***Labelme2yolo*** ya que para entrenar nuestro modelo haremos **Transfer Learning** con el algoritmo ***YOLO*** *(You Only Look Once)*.

El algoritmo ***YOLO*** es un sistema de detección de objetos en tiempo real, el cual hace uso de una única red neuronal convolucional para detectar objetos en imágenes.

Este algoritmo funciona muy bien para detectar múltiples objetos, cómo personas, vehículos, alimentos, animales… pero en nuestro caso no nos serviría ya que queremos detectar los animatrónicos del videojuego.

Para solucionar este problema haremos uso de la técnica **Transfer Learning** (aprendizaje transferido),que consiste en usar un modelo ya entrenado para desarrollar rápidamente modelos eficaces y resolver problemas complejos.

Labelme2yolo se encarga de recoger todos los JSON anteriormente creados y crear un dataset el cual puede ser usado para hacer transfer learning. El dataset se encuentra en la carpeta "*DATASET_V2*" y dentro de ella podemos encontrar las carpetas "*train*" , "*val*" y el archivo "*dataset.yaml*".

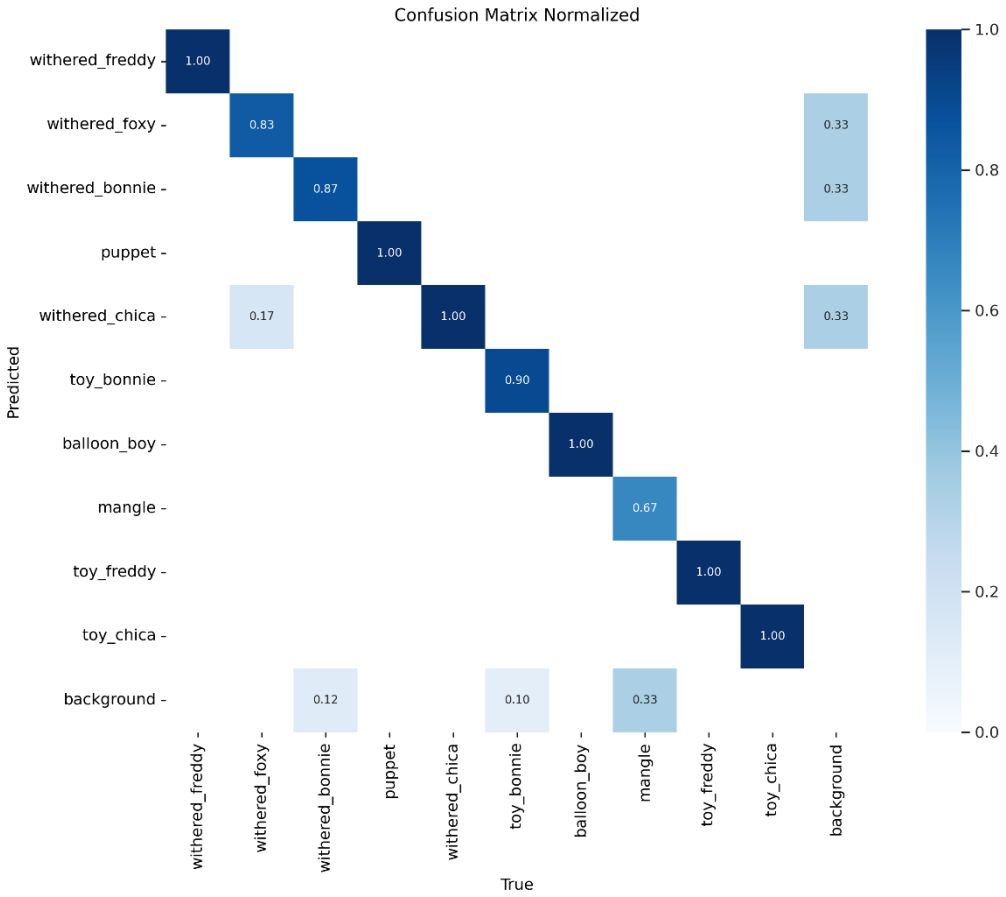
En cada subcarpeta se encuentras otras dos carpetas: "images", donde se guardan las imágenes, y "labels" donde se guardan las coordenadas de los animatrónicos de esas imágenes.

La subcarpeta *"train"* se usará para entrenamiento y "*val"* para test.

Como el entrenamiento del modelo es un proceso que requiere mucha GPU usaremos Google Colab para que se encargue de hacerlo en el script "***crear_modelo_deteccion.ipynb****"*.

En este archivo usaremos la librería ***ultralytics*** con la cual podremos crear el modelo usando la técnica transfer learning con YOLO (habiendo descargado previamente el modelo YOLOv8 **'yolov8m-seg.pt'**) y usando el dataset creado.

Para reflejar el rendimiento del modelo podemos fijarnos en la matriz de confusión resultante:



Vemos que el modelo funciona bastante bien, teniendo los peores resultados con el animatrónico de mangle, que un 33% de las veces confundió el background con mangle.

• USO DEL MODELO

Lo realizado en este subapartado se podrá observar en el archivo *'files/modoIA/detection.py' y 'files/modoIA/anim_paths.py'*

Para usar el modelo de detección, primero nos enfocamos en que el juego no tenga problemas de rendimiento, creando un hilo que sale del proceso principal, que se encargará exclusivamente de realizar las predicciones de detección.

Este hilo lo llamamos **detection_thread**, y se encargará de tomar un array del frame actual del juego (definiendo que sea BGR y de tipo uint8) y realizar una predicción cada 0.1 segundos usando el modelo previamente creado. Para evitar falsos positivos impondremos que debe tener como mínimo un 0.67 de confianza la predicción para que sea usada

```python
def __init__(self,Lock,env_var,surface,anim_path):

    self.detection_thread = Thread(target=self.run_detection, daemon=True)
    self.stop_detection = Event()
    self.start_detection_thread()

def detection(self):
        with self.lock:
            frame_array = pygame.surfarray.array3d(self.game_surface)
            frame_array =frame_array.swapaxes(0,1)
            frame_array = cv2.cvtColor(frame_array, cv2.COLOR_RGB2BGR)
            frame_array = frame_array.astype(np.uint8)
            self.results = self.model.predict(frame_array, conf=0.67)
            self.check_anim()

def run_detection(self):
    while not self.stop_detection.is_set():
        self.detection()
        time.sleep(0.05)



def start_detection_thread(self):
    self.detection_thread.start()

def stop_detection_thread(self):
    self.stop_detection.set()
    self.detection_thread.join()
```

Hilo de detección

Por cada detección usaremos una función llamada “**check_anim()**”. Esta función se encargará de recoger el resultado de la predicción del modelo y tendrá dos funcionalidades:

- Usará la función “**check_location()**” de la clase **“anim_path”**, la cual usará como argumentos el animatrónico detectado y el número de cámara actual para saber si la predicción hecha es un falso positivo. Esto se logrará debido a que, como hemos visto en el punto 3, cada animatrónico seguirá un camino personalizado, por lo que, si el número de cámara actual no pertenece al camino del animatrónico detectado, no se usará esa predicción.

```
class AnimPaths:
    def __init__(self):
        self.anim_location={
         "toy_freddy":[9,10,0],
         "toy_chica":[9,7,4,1,5,0],
         "toy_bonnie":[9,3,4,2,6,0],
         "withered_bonnie":[8,7,1,5,0],
         "withered_chica":[8,4,2,6,0],
         "withered_foxy":[8,0],
         "withered_freddy":[8,7,3,0],
         "balloon_boy":[10,5,0],
         "mangle":[12,11,6,0],
         "puppet":[11,0],
        }
    def check_location(self, anim,num_camera):
        if num_camera in self.anim_location[anim]:
            return True
        else: return False
```

Función check_location

- Una vez evitado el falso positivo, se pasarán los animatrónicos a la función **“fill_dictionary”** la cual introducirá los animatrónicos detectados en un diccionario llamado “**anim_dict**”, del cual hablaremos posteriormente, de forma que la clave del diccionario sea el número de cámara actual y el valor sea un array con los animatrónicos que se han detectado en esa cámara. Si el animatrónico detectado ya estaba en una cámara, se quitará de ese array y pasará al array del número de cámara actual.

```
def fill_dictionary(self,current_anims):
        if not any(self.env_var.anim_dict.values()):
                # Si no hay animatrónicos registrados en ninguna cámara, registrar los actuales en la
cámara actual
                self.env_var.anim_dict[self.env_var.num_camera] = set(current_anims)
        else:
                # Si hay animatrónicos registrados, compara con los actuales y actualiza el anim_dict
                for anim_set in self.env_var.anim_dict.values():
                    common_anims = anim_set.intersection(current_anims)
                    if common_anims:
                        # Elimina los aniamtrónicos de su ubicación actual en el diccionario
                        anim_set.difference_update(common_anims)
                        # Agrega las aniamtrónicos al anim_dict[num_camera]
                        self.env_var.anim_dict[self.env_var.num_camera].update(common_anims)
                    else:
                        self.env_var.anim_dict[self.env_var.num_camera].update(current_anims)
```

Función fill_dictionary

```
def check_anim(self):
            current_anims = []
            if self.results is not None:
                for r in self.results:
                    for i in range(len(r.boxes)):
                        class_id = int(r.boxes.cls[i].numpy())
                        class_name = self.model.names[class_id]
                        if self.anim_path.check_location(class_name,self.env_var.num_camera):
                            current_anims.append(class_name)
            self.fill_dictionary(current_anims)
```

Función check_anim

6. MODELO DE APRENDIZAJE REFORZADO

• VARIABLES DE ENTORNO

Lo realizado en este subapartado se podrá observar en el archivo *'files/modoIA/env_variables.py'*

Las variables de entorno desempeñan un papel fundamental para el control del juego a partir de la IA.

Las hemos organizado todas en una clase ya que esto facilita su gestión y contribuye con el tema de la coherencia y eficiencia del código.

Nos hemos encargado de modificar el código fuente del juego, para que cuando se elija el modo IA en vez de controlarse manualmente (por la persona que esté jugando), la IA consiga controlar el juego por las variables de entorno que tenemos creadas.

Debido a que hay acciones que para que se realicen satisfactoriamente no sirve con que se realice la acción, sino que debe mantenerse en el tiempo, (como la acción de darle cuerda a la caja de música) creamos también unos temporizadores:

- **'action_timers'**: Diccionario que almacena el tiempo transcurrido desde la última acción en diversas situaciones. Estas situaciones como hemos mencionado a lo largo del proyecto son distintas, ya sean observar el monitor, defenderse de un enemigo…

- **'time_action_foxy'**: Variable que acumula el tiempo que llevamos defendiéndonos del animatrónico Foxy.

- **'time_action_puppet'**:Variable que acumula el tiempo que llevamos defendiéndonos del animatrónico Puppet.

- **'time_defense'**: Variable que acumula el tiempo que llevamos con la mácara.

Tendremos la variable '**game_over'** que nos servirá para representar el final del juego

Además de las variables que actúan como temporizadores y la variable game_over , también hay otras variables permitirán a la IA manejar el juego:

- '**turn_to_left'**: Booleano para controlar el movimiento a la izquierda en la oficina.

- '**turn_to_right'**: Booleano que actúa igual que "turn_to_left" pero en dirección contraria. (A la derecha).

- '**hallway':** Booleano que al estar en True emitirá luz en el pasillo de la oficina. Nos será muy útil para defendernos de Foxy.

- '**right_vent'**: Booleano que representa si la luz en la ventilación de la derecha está encendida o apagada.

- '**left_vent'**: Booleano que representa lo mismo que right_vent pero orientado a la izquierda.

- '**open_monitor'**: Booleano el cual si está a True se abrirá el monitor y si está en False se verá la oficina.

- '**put_mask'**: Booleano que representa el uso de la máscara por parte de la IA. El uso de la máscara nos servirá para defendernos de diferentes animatrónicos como hemos visto anteriormente.

- '**num_camera'**: Integer que representa el número de cámara actual (si es 0 estaremos en la oficina).

- '**flashlight'**: Booleano para encender la luz en las cámaras.

- '**music_box':** Booleano que al estar en true y posicionarnos en la cámara 11, dará cuerda a la caja de música. Debemos tener cuidado con Puppet ya que podría terminar el juego en el caso de que se pase de tiempo la caja de música.

Además de las variables anteriormente mencionadas, tendremos el diccionario mencionado en el punto 4:

- '**anim_dict'**: este diccionario asigna a cada cámara del juego un conjunto de animatrónicos detectados por el modelo de detección.

Finalmente contaremos con una variable llamada **'log'** la cual es una cadena de texto que nos será de utilidad para mostrar por pantalla que acción se está reproduciendo.

En cuanto a las funcionas de la clase, nos sirven para resetear todas las variables, lo cual nos será útil para utilizar a la hora de crear el modelo de aprendizaje reforzado:

- **'reset()'**: Resetea algunas variables a su valor original.
- **'reset_action_duration()'**: Reinicia los temporizadores.

```
class EnvironmentVariables:
    def __init__(self,position):

        #Tiempos
        self.action_timers = {"observe_office": None, "observe_monitor": None, "defense_normal": None,
"defense_foxy": None,
                              "defensa_balloon_boy": None, "defensa_puppet": None
                              }
        self.time_action_foxy=0
        self.time_action_puppet=0
        self.time_defense=0
        self.position=position

        #VARIABLES DE ENTORNO

        #en pasillo
        self.turn_to_left = False
        self.turn_to_right = False
        self.hallway = False
        self.right_vent = False
        self.left_vent = False
        self.open_monitor = False
        self.put_mask = False


        #en el monitor
        self.num_camera = 9
        self.flashlight=False
        self.music_box = True

        #Dict de los animatronicos
        self.anim_map = {i: set() for i in range(0, 13)}

        #LOG
        self.log=""
        #Game_over
        self.game_over=False
    def reset(self):
        self.turn_to_left = False
        self.turn_to_right = False
        self.hallway = False
        self.right_vent = False
        self.left_vent = False
        self.open_monitor = False
        self.put_mask = False
        self.jumpscare=False
        self.anim_map = {i: set() for i in range(0, 13)}
        self.num_camera = 9
        self.flashlight=False
        self.music_box = True
        self.game_over=False
        self.reset_action_duration()
  def reset_action_duration(self):
        self.action_timers = {"observe_office": None, "observe_monitor": None, "defense_normal": None,
"defense_foxy": None,
                              "defensa_balloon_boy": None, "defensa_puppet": None, }
        self.time_action_foxy=0
        self.time_action_puppet=0
        self.time_defense=0
```

Clase EnvironmentVariables

• ACCIONES

Lo realizado en este subapartado se podrá observar en el archivo *'files/modoIA/action_manager.py'*

Una vez definidas todas las variables de entorno, crearemos acciones que la IA va a realizar para que no tenga que modificar directamente las variables de entorno. Esto nos ayudará a tener un código más claro y eficiente.

Podemos diferenciar tres tipos de acciones, las de observación ,las de defensa y las de iluminación.

En las de observación se encuentran las acciones:

- **turn_left, turn_right y change_camera**

En las de defensa están las acciones:

- **defense_puppet, defense_foxy y defense_normal**

Y en cuanto la iluminación se controlará por las acciones:

- **turn_on_light** y **turn_off_light**

Todas estas acciones tienen como cometido realizar cambios en varias variables de entorno a la vez para mantener un correcto funcionamiento del juego.

Además, controlarán los temporizadores, haciendo que los temporizadores dedicados a la defensa sean acumulativos, mientras que los dedicados a la observación vuelvan a 0 cada vez que se cambia entre oficina y monitor

```python
def change_camera(self):
    with self.lock:
        self.env_action.log="CAMBIAR CAMARA"
        self.start_action_timer("observe_monitor")
        self.pause_timers('observe_monitor')
        if(self.env_action.action_timers['observe_office'] is not None):
            self.env_action.action_timers['observe_office'] = None

        self.env_action.put_mask = False
        self.env_action.open_monitor = True
        probabilidad = random.randint(1, 100)

        # Dar más prioridad a las cámaras 5 y 6 (cámaras de ventilación)
        if probabilidad <= 30:
            self.env_action.num_camera = random.choice([5, 6])
        else:
            self.env_action.num_camera = random.randint(1, 12)
        self.turn_on_light()
```

Acción dedicada a cambiar la cámara

• PROBLEMA DE CONCURRENCIA

Como las detecciones se hacen en un hilo y el proceso principal cambia el número de cámara gracias a las acciones creadas, nos encontramos con una condición de carrera.

La condición de carrera se debe a que, si justo después de que se haga la predicción con el modelo de detección, se ejecuta una de las acciones anteriormente mencionadas, cambiarían el número de cámara, por lo que tendríamos los animatrónicos que se acaban de detectar y la variable del número de cámara no sería igual al número de cámara en el que se han detectado a los animatrónicos.

Esto provocaría que si se ejecuta la función "**check_anim()"** (anteriormente mencionada), introduciría los animatrónicos detectados en un número de cámara erróneo.

Para solucionarlo, crearemos un Lock que usarán tanto la función de detección como las acciones

• ENTORNO DE APRENDIZAJE REFORZADO

Lo realizado en este subapartado se podrá observar en el archivo *'files/modoIA/env_RL.py'*

El entorno para crear el aprendizaje Reforzado está hecho con la librería **GYM**.

La librería Gym es una herramienta desarrollada por OpenAI para crear y trabajar con entornos de aprendizaje reforzado (RL, por sus siglas en inglés). El propósito principal de Gym es proporcionar un conjunto estandarizado de entornos.

Este entorno se encarga de definir un espacio de acciones ya nombradas y luego define un espacio de observaciones, las cuales serán las variables de entorno junto a booleanos que faciliten la observación al modelo. En el entorno tendremos tres funciones importantes:

- **Step():** Función que recibiendo como parámetro una acción, devuelve, entre otros datos, la recompensa obtenida y un booleano llamado **done** que representa el fin del juego.

  A la hora de hacer las recompensas se diferencia entre estar defendiendo u observando, siempre teniendo en cuenta el temporizador de la caja de música de Puppet.

  Si en el diccionario anim_dict encontramos que en la oficina o las ventilaciones (claves 0,5 y 6 del anim_dict) se encuentra un animatrónico, daremos una recompensa positiva si la IA ejecuta la acción para defenderse de este. En cambio, si no hay peligros, daremos a la IA recompensa positiva si ejecuta las acciones de observación.

  Si el temporizador de la observación de la oficina lleva más de los segundos permitidos, daremos recompensa negativa para conseguir que cambie a la observación del monitor y viceversa.

  En este caso nos serán útiles los temporizadores anteriormente mencionados, ya que a la hora de defendernos no servirá simplemente con ejecutar una acción, sino que esta deberá permanecer durante unos segundos.

- **_get_Observation()**: nos devolverá el estado del espacio de observaciones

- **Reset()**: reiniciará las variables de entorno cada vez que un episodio termine usando las funciones de reset de la clase **EnvironmentVariables**

• ALGORITMO APRENDIZAJE REFORZADO (Q-LEARNING)

Lo realizado en este subapartado se podrá observar en el archivo *'files/modoIA/modo_ia.py'*

Una vez definido el entorno a usar, creamos la función **train_model,** que se basa en el algoritmo Q-Learning.

Q-learning es un algoritmo de aprendizaje por refuerzo basado en la idea de aprendizaje mediante ensayo y error. Su principal objetivo es descubrir la estrategia óptima que guía las acciones del agente para maximizar el valor esperado de las recompensas futuras.

Estos valores se almacenan en una tabla conocida como Q-table, un mapa que asocia cada estado con todas las acciones posibles y sus respectivos valores de utilidad, es decir, la ganancia esperada por el agente cuando realiza una determinada acción en un estado específico.

El proceso de aprendizaje implica exploración y explotación, donde la inteligencia artificial va tomando decisiones aleatorias o basadas en el aprendizaje obtenido previamente.

La ecuación que actualiza los Q-Values es la siguiente:

$$\mathbf{new\_value} = (\mathbf{1} - \mathbf{self.alpha}) * \mathbf{old\_value} + \mathbf{self.alpha} * (\mathbf{reward} + \mathbf{self.gamma} * \mathbf{next\_max})$$

la cual se aplica en cada iteración, de esta manera lo que conseguimos es ajustar los valores de la tabla Q para así maximizar las recompensas futuras

La función **train_model** se basará en un bucle en el que definiendo el número de noches (episodios) a realizar, ejecutará otro bucle del cual no saldrá hasta que la variable **done** (definida en el entorno) no sea verdadera. Una vez terminado todos los episodios guardamos el modelo gracias a la librería **joblib**.

Como el espacio de observaciones es muy grande, para que la tabla q-table identifique cada observación, se realizará una conversión a un hash único por cada observación realizada.

También crearemos un archivo CSV para visualizar mejor las acciones y que recompensas obtienen según la observación realizada.

Una vez creado el modelo, la IA ya estará entrenada y podremos ejecutarla con la función **run_model**, en el que daremos 5 intentos a la IA para que consiga pasarse el juego, si no lo consigue volverá al menú principal.

Durante la ejecución del modelo recomendamos ver la terminal para ver los mensajes en los que se detallan como se dan la recompensas.

8. DIBUJO EN PANTALLA

Lo realizado en este subapartado se podrá observar en el archivo *'files/modoIA/draw_ia.py'*

Para visualizar la detección de animatrónicos realizada por el modelo de detección y la acción que se ejecute a partir del modelo de aprendizaje reforzado crearemos dos funciones:

- **draw_rects**: a partir de los resultados de la predicción dibujará un recuadro en pantalla sobre el animatrónico detectado junto a su nombre y la confianza de esta detección.

```
def draw_rects(self,ia):
        if  ia.detection.results is not None:
            for r in  ia.detection.results:
                for i in range(len(r.boxes)):
                    class_id = int(r.boxes.cls[i].numpy())
                    confidence = r.boxes.conf[i].numpy()
                    class_name = ia.detection.model.names[class_id]

                    # Obtener las coordenadas ajustadas a las dimensiones actuales
                    x_min, y_min, x_max, y_max = map(int, r.boxes.xyxy[i, :4].numpy())
                    pygame.draw.rect(self.game_surface, (255, 0, 0), (x_min, y_min, x_max - x_min, y_max
- y_min), 5)

                    font = pygame.font.Font("five-nights-at-freddys.ttf", 36)
                    label_text = font.render(f"{class_name} {confidence:.2f}", True, (255, 255, 255))
                    self.game_surface.blit(label_text, (x_min, y_max + 5))
ain_text_x, main_text_y))
        self.game_surface.blit(rect, ((self.game_surface.get_width() - rect.get_width()) // 2,
self.game_surface.get_height() - 80))
```

Función draw_rects

- **write_log**: dibujará un recuadro blanco en el inferior de la pantalla y hará uso de la cadena de texto **log** definida en las variables de entorno para mostrar en pantalla la acción realizada.

```
    def write_log(self,rect,log):
        font_large = pygame.font.Font("five-nights-at-freddys.ttf", 36)
        main_text_surface = font_large.render(log, True, (0, 0, 0))
        main_text_x = (rect.get_width() - main_text_surface.get_width()) // 2
        main_text_y = (rect.get_height() - main_text_surface.get_height()) // 2
        rect.fill((255, 255, 255, 220))
        rect.blit(main_text_surface, (main_text_x, main_text_y))
        self.game_surface.blit(rect, ((self.game_surface.get_width() - rect.get_width()) // 2,
self.game_surface.get_height() - 80))
```

Función write_log

9. CONCLUSIÓN

La creación de un modelo de inteligencia artificial que consiga pasarse una noche en el videojuego *FIVE NIGHTS AT FREDDY'S 2* nos ha supuesto un reto en el que a medida que avanzábamos hemos ido aprendiendo conceptos como **transfer learning**, el modelo de detección **YOLO** o el algoritmo **Q-Learning**.

En cuanto al modelo de detección, conseguimos que no perjudicara severamente el rendimiento del videojuego, creando un hilo exclusivo para la predicción del modelo.

Conforme íbamos avanzando aprendíamos más sobre la ejecución del videojuego, lo cual nos permitió perfeccionar las acciones realizadas para la defensa y observación, disminuyéndolas solo a 6 acciones y crear un buen sistema de recompensas en el que tener en cuenta las necesidades de defensa de cada animatrónico y la observación de todo el restaurante.

También tuvimos que lidiar con el problema de concurrencia, solucionándolo con un Lock para evitar un asincronismo en la ejecución del modelo de aprendizaje reforzado junto al modelo de detección.

En conclusión, la combinación de los modelos de detección y de aprendizaje reforzado ha sido un reto en el que gracias al perfeccionamiento de ambos modelos consideramos que hemos conseguido realizar la finalidad del proyecto con éxito.

• MEJORAS Y AMPLIACIONES

En cuanto a las mejoras y ampliaciones de los modelos de inteligencia artificial creados podemos destacar:

- El dibujo del rectángulo de detección de animatrónicos no se realiza a tiempo real, sino que se hace de modo secuencial en la función **train_model** o **run_model** si estas en el **modo IA**, o en la función **update** de la clase **game_controller** del juego si estas en el modo **Juego guiado**.

  Esto hace que en vez de salir el recuadro justo cuando aparece un animatrónico y luego desaparezca cuando ya no está, tardará unos segundos en aparecer y en irse, lo que provoca que a pesar de que si se rellena bien el diccionario de animatrónicos **anim_dict**, en pantalla de forma visual puede aparecernos unos segundos más tarde este recuadro.

  Para conseguir solucionarlo se deberá ampliar el código creando otro hilo que trabaje a tiempo real dibujando los rectángulos que se encuentren en la variable **results** (variable del resultado de la predicción del modelo).

- En cuanto al modelo de aprendizaje reforzado, no tenemos en cuenta que sea la IA la que controle el encendido y apagado de luces, ya que esto complicaría demasiado el código. En cambio, el encendido y apagado de luces lo controlamos dentro de las 6 acciones, haciendo que por ejemplo si se ejecuta la función **defense_foxy** se enciendan las luces para iluminar el pasillo. Es una solución que funciona, pero no hacemos que sea el modelo el que controle que la energía de la batería no se acabe.

  Se podría solucionar añadiendo las acciones **turn_on_light** y **turn_off_light** al espacio de acciones del entorno de aprendizaje reforzado, sin embargo, se complicaría demasiado el código de obtención de recompensas y seguramente habría que cambiar el modelo para que sea en vez de por aprendizaje reforzado por aprendizaje profundo.

- Actualmente para saber si un animatrónico ya no está en la oficina o las ventilaciones lo hacemos a partir de la ejecución de acciones durante ciertos segundos. Es decir que, si por ejemplo aparece Foxy en el pasillo, para saber que esa amenaza ha pasado suponemos que tras 6 segundos ejecutando la acción de defensa de Foxy ya no está en la sala y lo eliminamos del diccionario de animatrónicos **‘anim_dict’**.
  Para mejorarlo deberíamos hacerlo a partir del modelo de detección, en el que, si estando en cualquier sala no detecta ningún animatrónico, lo elimine del

diccionario. Sin embargo, esto tiene un inconveniente, y es que algunas cámaras rotan y en la oficina también debemos girar la cámara para identificar diferentes animatrónicos. Por lo que, si la cámara esta girada y el animatrónico no entra en el espacio de visión, el modelo de detección no nos devolvería ningún animatrónico detectado a pesar de que si está en la sala.
Esto lo podríamos solucionar haciendo que el diccionario no solo diferencie por el número de cámaras, sino que también lo haga teniendo en cuenta si la cámara está girada usando las variables del código fuente del videojuego. Por lo que por ejemplo en vez de tener en el diccionario:

0: {'toy_freddy','mangle'}

Se cambiaría por:

0_left: {}

0_center: {'withered_foxy'}

0_right: {'mangle'}

10. BIBLIOGRAFÍA

- *Repositorio de la recreación del videojuego realizada en Python*: *https://github.com/EDUATO/fnaf-in-pygame/tree/master*
- *Documentación sobre el videojuego*: *https://freddy-fazbears-pizza.fandom.com/es/wiki/Five_Nights_at_Freddy%27s_2*
- *Creación del dataset para el modelo de detección:* *https://opensistemas.com/paso-a-paso-entrenamiento-de-la-red-neuronal/*
- *Predicción con la librería ultralytics:* *https://docs.ultralytics.com/es/modes/predict/*
- *Entorno de aprendizaje reforzado y algoritmo q-learning:* *https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/*