Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

"Diseño e implementación del juego 3 en raya mediante técnicas Reinforcement Learning en el Robot IRB120"

Autor: Sergio Varona Rabasco

Tutor/es: Rafael Barea Navarro

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

"Diseño e implementación del juego 3 en raya mediante técnicas Reinforcement Learning en el Robot IRB120"

Autor: Sergio Varona Rabasco

Tutor: Rafael Barea Navarro

Universidad: Universidad de Alcalá

Tribunal:

Presidente: Francisco Javier Meca Meca

Vocal 1º: Pedro Martín Sánchez

Vocal 2º: Rafael Barea Navarro

Fecha de depósito: septiembre 2021

AGRADECIMIENTOS

Este proyecto da final a una etapa importante de mi vida. Quiero agradecer a las personas que durante esta etapa me han apoyado y dado fuerzas para lograr finalizar mi carrera con éxito.

En primer lugar, a mis compañeros de carrera: César y Koko, por estar siempre a mi lado, Almudena y Laura por ser un ejemplo de trabajo diario y un gran apoyo, Víctor, Pablo y las demás personas que han estado a mi lado a lo largo de esta etapa.

En segundo lugar, a mis amigos de la infancia, que siempre han estado a mi lado, tanto en los buenos y malos momentos, Samuel, Esteban y Óscar.

A todos los profesores que a lo largo de mi etapa de estudiante me han ayudado y servido de inspiración en mis estudios, en especial a Rafael, por apoyarme y ayudarme semanalmente en la realización de este proyecto.

A mi familia, por aguantarme tantos años de carrera y siempre darme ánimos en los momentos más duros de estudio.

Y por último, una mención especial a Cristina, quien ha estado conmigo en todo momento de la etapa final de mis estudios y a pesar de no saber del tema, apoyarme y mostrar interés acerca de mi proyecto.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

GLOSARIO DE ACRÓNIMOS Y ABREVIATURAS

GUI	*Graphical User Interface* (Interfaz Gráfica de Usuario)
GUIDE	*Gui Development Enviromment*
RL	Reinforcement Learning (Aprendizaje por Refuerzo)
IA	Inteligencia Artificial
ABB	*Asea Brown Boveri*
SARSA	*State Action Reward State Action*
Matlab	MATrix LABoratory
TFG	Trabajo Fin de Grado
TFM	Trabajo Fin de Máster
RAE	Real Academia Española
IVA	Impuesto al Valor agregado

RESUMEN

El objetivo de este proyecto es el diseño e implementación del juego 3 en raya en el brazo robótico IRB-120 mediante técnicas de autoaprendizaje, para así lograr jugar a este juego entre el robot y el usuario. Para ello, se realiza el desarrollo del juego mediante Matlab y RobotStudio. Por una parte, la inteligencia artificial y la elección de movimientos serán llevados a cabo mediante Matlab. Por otra parte, el movimiento de las fichas se realiza mediante RobotStudio. Conectado ambas partes, mediante un socket de comunicación, damos por finalizado el proyecto.

Palabras clave: Matlab**,** RobotStudio**,** Tres en raya**,** Q_Learning**,** Sarsa**,** Aprendizaje por refuerzo IRB 120.

ABSTRACT

The objetive of this project is the design and implementation of the tic-tac-toe game in the robotic arm IRB120 trough reinforcement learning techniques, in order to be able to play this game between the robot and the user. For that purpos, the game is developed through Matlab and RobotStudio. On the one hand, the artificial intelligence and the choice of movements will be carried out using Matlab. On the other hand, the movement of the tic-tac-toe chips will be done using RobotStudio. Once the two parts are connected, through a communication socket, the Project was also terminated.

Keywords: Matlab**,** RobotStudio**,** TIC-TAC-TOE**,** Q_Learning**,** Sarsa**,** Reinforcement Learning, IRB 120.

RESUMEN EXTENDIDO

El objetivo de este proyecto es el diseño e implementación del juego 3 en raya en el brazo robótico IRB120, implementando un sistema de inteligencia artificial de aprendizaje por refuerzo, para así lograr jugar a este juego entre el robot y el usuario.

Este proyecto se divide en dos grandes pilares, la parte de programación de movimientos, realizada mediante RobotStudio y la parte de mecánica de juego, realizada mediante Matlab.

El apartado de Matlab se subdivide en diversos puntos. En primer lugar, se ha realizado la inteligencia artificial para la toma de decisión de los movimientos del robot. Esto se ha llevado a cabo mediante técnicas de aprendizaje por refuerzo, imponiendo niveles de juego en base a los entrenamientos realizados por los sistemas diseñados de inteligencia artificial. Estos se diferencian en dos métodos de autoaprendizaje, el método Q_Learning y el método Sarsa, los cuales se pueden elegir para que el robot entrene y adquiera su nivel.

La dificultad depende del número de entrenamientos que se mande realizar a la inteligencia artificial, en nuestro caso el nivel en función de entrenamientos es el siguiente:

- Fácil = 1000 entrenamientos.
- Normal = 10000 entrenamientos.
- Difícil = 100000 entrenamientos.

Después de realizar los entrenamientos se marcará cuantas veces la inteligencia artificial gana, empata y pierde, viendo así una idea aproximada de su nivel.

Además, se ha realizado una interfaz mediante la herramienta de Matlab "guide", que actúa como interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Esta se compone de la configuración inicial, donde se elige el método de entrenamiento, la dificultad y se termina el proceso realizando la conexión. Por otro lado, el tablero de juego donde se ha realizado la elección de movimientos, reinicio del juego y transcurso del juego.

Figura 0.1: Interfaz de la aplicación

En el apartado de Matlab, se han realizado los ficheros y funciones necesarios para completar el transcurso del juego, como actualizar el tablero de la interfaz según los movimientos, la elección de movimientos en cada turno, los movimientos que se envían al brazo robótico, detectar el estado final: ganar, empatar o perder, y la conexión entre Matlab y RobotStudio. Por otra parte, en el apartado de RobotStudio, se ha realizado el diseño de la estación, importando el área de trabajo e incluyendo el brazo robótico y, por último, la herramienta ventosa para poder coger y soltar las piezas de forma óptima.

A continuación, se ha diseñado el tablero y las fichas. Es necesario establecer los diferentes puntos y trayectorias para realizar los movimientos a lo largo del tablero.

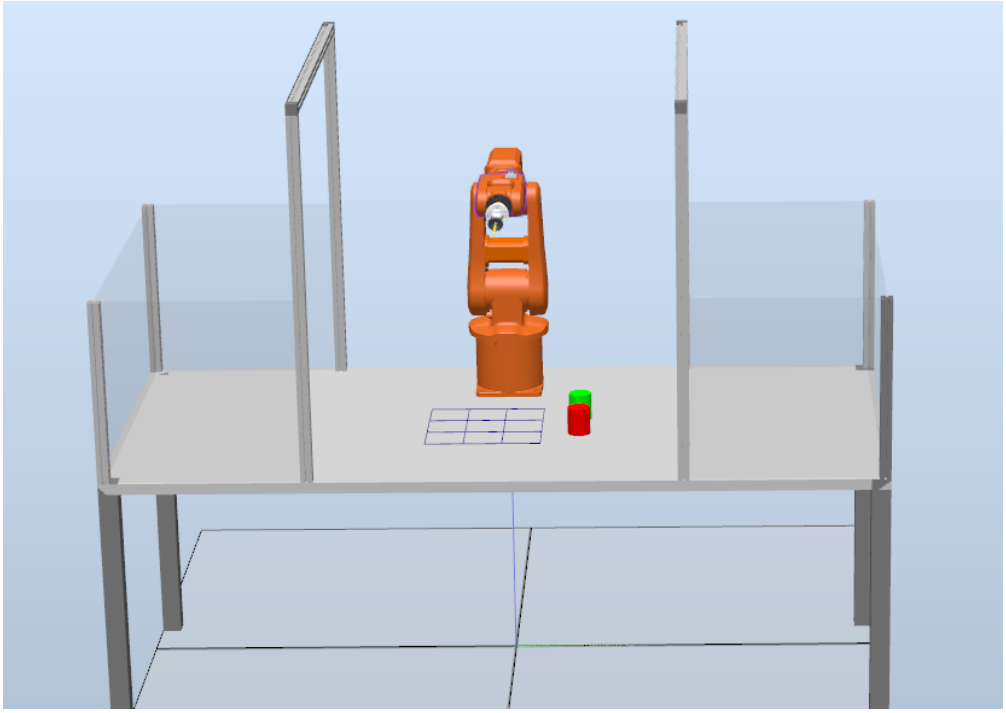

Figura 0.2: Estación inicial RobotStudio

Para la finalización del proyecto, hemos implementado el socket de comunicación realizado por Marek Jerzy Frydrysiak [1] y adaptado por David Paz Hernández [2], que permitirá tanto la salida como la entrada de información por parte del robot para su control.

Con todo lo descrito anteriormente, se da por concluido el proyecto en el que mediante la interfaz de Matlab se han realizado una o varias partidas del juego 3 en raya contra el brazo robótico. Este robot moverá las fichas en función de lo marcado en la interfaz y la respuesta se realiza mediante inteligencia artificial, hasta finalizar la partida con uno de los siguientes estados: victoria, empate o derrota.

CAPÍTULO 1:

INTRODUCCIÓN

1.1 ROBÓTICA

La robótica se define como una ciencia que agrupa varias ramas tecnológicas y disciplinas, con el objetivo de diseñar máquinas robotizadas que sean capaces de realizar tareas automatizadas o simular un determinado comportamiento animal o humano, en función de la capacidad de su software.

Ésta combina varias disciplinas [3], que son:

- Mecánica, conforma las partes físicas del robot.
- Electrónica, diversos circuitos que conforman el robot.
- Informática, nos permite comunicarnos con el robot.
- Inteligencia artificial, que dota al robot con la capacidad de tomar decisiones por sí mismo.
- Ingeniería de control, para realizar cualquier proceso de control de manera óptima.
- Física, que determina las capacidades del robot

Se llama robot a una entidad autómata compuesta por mecánica artificial y un sistema electromecánico, creado mediante la investigación de la ciencia y tecnología.

El campo de la robótica industrial se define como el diseño, estudio y uso de robot para un desempeño en procesos industriales. Por tanto, un robot industrial es aquel que es diseñado y programado para ser utilizado en el sector industrial con el objetivo de automatizar un proceso. La mayoría de los robots industriales son utilizados en cadenas de montajes, realizando trabajos que requieren de una gran precisión y en un tiempo muy reducido, lo que es una gran ventaja si lo comparamos con un operario que debido al peso que ha de manejar, la precisión y el tiempo, restaría una gran rentabilidad al proceso [4].

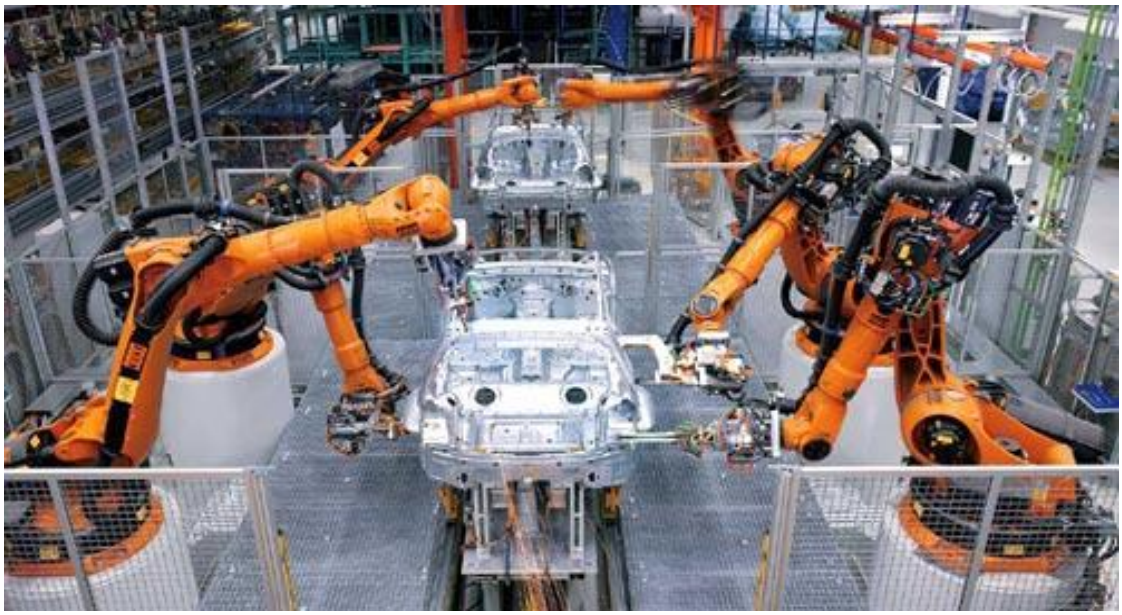

Figura 1.1: Cadena de montaje

El tipo de robot que por su estructura es más usual en la industria es el llamado brazo robótico. Se define como un tipo de brazo mecánico, normalmente programable, con funciones parecidas a las de un brazo humano. Este puede ser la suma total del mecanismo o puede ser parte de un robot más complejo. Requieren de articulaciones para poder realizar movimientos rotacionales, traslacionales o desplazamientos lineales.

Figura 1.2: Ejemplo de brazo robótico

Un robot está formado por los siguientes elementos:

- Dispositivos de entrada/salida, que permiten la comunicación, como un monitor o una caja de comandos.
- Dispositivos especiales, como los ejes que facilitan el movimiento transversal del manipulador y las estaciones de ensamblaje, que su función es sujetar las distintas piezas de trabajo.
- Manipulador, como componente principal. Formado por una serie de elementos estructurales o eslabones unidos mediante articulaciones, para lograr un movimiento relativo entre dos eslabones consecutivos.
- Controlador, es el que regula los movimientos del manipulador, las acciones, los cálculos y procesado de información. Este almacena programas y envía y recibe señales a otras máquinas o herramientas.

Figura 1.3: Manipulador y controlador

1.2 TRES EN RAYA

Se datan juegos de mesa desde la época Neolítica, los cuales evolucionaron en la Edad de Bronce. La historia del tres en raya data de hace casi mil años, situada en la antigua Persia, desde donde mercaderes italianos lo exportaron a Europa y lo extendieron. Se convirtió en un juego muy popular en las clases bajas durante la edad Media.

El juego tres en raya se basa en conseguir tener 3 figuras iguales en raya, ya sea horizontal, diagonal o verticalmente. Para ello, consta de un tablero 3x3, en el que por turnos cada jugador va poniendo una figura en una casilla vacía, hasta que uno logra la victoria o se acaban los huecos, lo que da lugar a empate. El juego perfecto termina siempre en empate sin importar la colocación de ficha del primer jugador. En cuanto a las figuras, son una X para un jugador y el otro utilizará un O. Es un juego simple, pero con estrategia, lo que lo hace ideal como herramienta pedagógica para transmitir los conceptos de teoría de juegos. Se puede jugar con diversos materiales, tales como fichas y tablero o papel y boli.

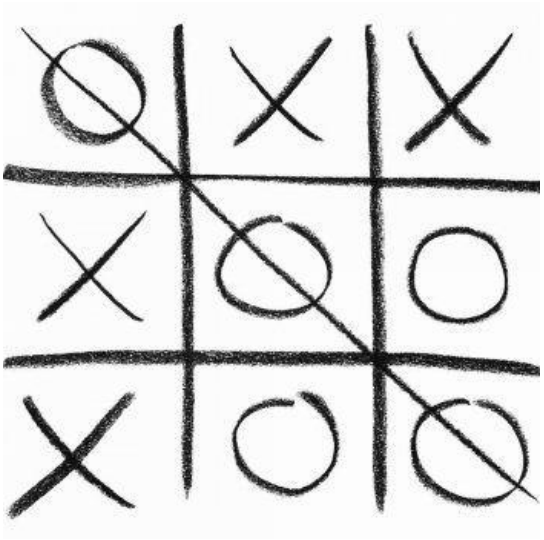

Figura 1.4: Ejemplo partida 3 en raya

1.3 INTELIGENCIA ARTIFICIAL

Según la Real Academia Española [5], se define el concepto de inteligencia como:

- Capacidad de entender o comprender.
- Capacidad de resolver problemas.
- Conocimiento, comprensión, acto de entender.
- Sentido en que se puede tomar una proposición, un dicho o una expresión.
- Habilidad, destreza y experiencia.

En términos generales, se define la inteligencia como la capacidad de percibir o inferir información y retenerla como conocimiento para aplicarlo a comportamientos adaptativos dentro de un contexto o entorno.

Si asociamos el concepto de inteligencia al mundo de la informática, expresada por maquinas, sus procesadores y softwares, llegamos al concepto de inteligencia artificial (IA).

La RAE describe el concepto de inteligencia artificial como [5]:
disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico.
Normalmente el término IA, se asocia a cuando una maquina imita las funciones cognitivas que los humanos asocian con otras mentes humanas, como tener la capacidad de percibir, razonar, aprender o resolver problemas.

Se distinguen cuatro tipos de inteligencia artificial o sistemas basados en IA:

1 Reactiva**:** Capacidad limitada, obtienen una respuesta a diferentes tipos de estímulos, pero su funcionalidad no está basada en la memoria. En conclusión, no pueden aprender, ya que no hacen uso de experiencias pasadas. Un ejemplo es la máquina de IBM llamada Deep Blue, que en 1997 logró derrotar al campeón del mundo de ajedrez, Garry Kasparov.

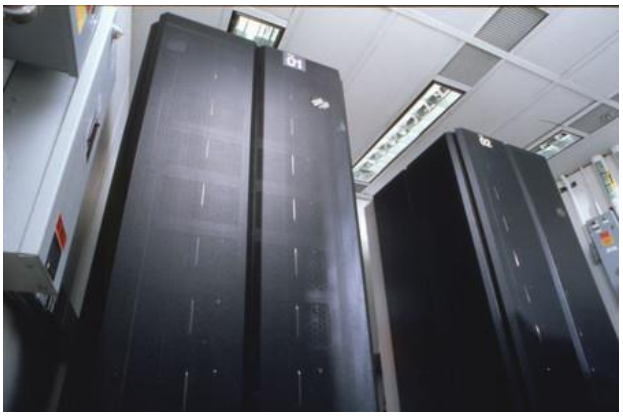

Figura 1.5: Ordenador Deep Blue

2 Memoria limitada: tiene la capacidad de reaccionar a estímulos y puede aprender de información previa para la toma de decisiones. Este método se basa en usar una gran cantidad de datos para entrenar, teniendo así referencias a las que acudir a la hora de resolver un problema o tarea. Muchas de las aplicaciones que conocemos y utilizamos a

diario utilizan este tipo de IA. Unos de los ejemplos más comunes son los asistentes virtuales (Siri o Alexa).

Figura 1.6: Asistente virtual Alexa

3 Teoría de la mente: está aún en proceso, cuando este tipo de IA se lleve a cabo, las máquinas podrían llegar a entender a los actores con los que interactúen identificando sus necesidades, creencias, procesos mentales o emociones.

4 Autoconciencia: constituye la fase final de los tipos de IA, se daría cuando un sistema o máquina logre ser consciente de su propia existencia. Este nivel es la meta que se intenta alcanzar en esta área, aunque esto significa muchos años más de esfuerzo y trabajo.

1.4 ESTRUCTURA GENERAL DEL SISTEMA

Este proyecto combina la programación de inteligencia artificial por autoaprendizaje y el uso del brazo robótico IRB120 mediante los softwares de Matlab y RobotStudio.
Matlab se divide en tres partes:

1. Diseño e implementación de la IA mediante técnicas de aprendizaje por refuerzo.
2. Diseño de la interfaz de juego.
3. Lógica de juego.

Mediante el método de autoaprendizaje la máquina realizará la toma de decisión en el juego 3 en raya, basándose en el número de entrenamientos realizados anteriormente, lo que determinará su nivel como jugador, el cual se divide en tres niveles. El nivel más bajo estaría diseñado para jugar contra niños, ya que tiene una dificultad muy baja y si se sabe jugar al juego siempre se logrará fácilmente la victoria, el nivel medio está diseñado para gente principiante en el juego, ya que es un nivel más complejo, pero si se tiene experiencia no supone un reto ganar a la IA. Por último, el nivel avanzado, está diseñado para que sea casi imposible ganar a la IA, por tanto, por muy buen jugador que se sea la partida acabará en empate o derrota para el usuario.

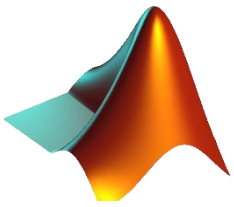

Figura 1.7: Icono Matlab

En el software de ABB se implementa una estación de trabajo, que contiene, el brazo robótico IRB-120, la herramienta ventosa, el tablero de juego y las fichas de colores rojo (usuario) y verde (robot).

Figura 1.8: Icono RobotStudio

A RobotStudio le llegará la información de cada movimiento para mover las fichas en el tablero, mientras que en Matlab tenemos la estructura completa del juego con un interfaz sobre la que se realizarán los movimientos. Por tanto, existirá una comunicación entre ambos softwares, diseñada por un socket de comunicación.

1.5 DIAGRAMA DEL PROYECTO

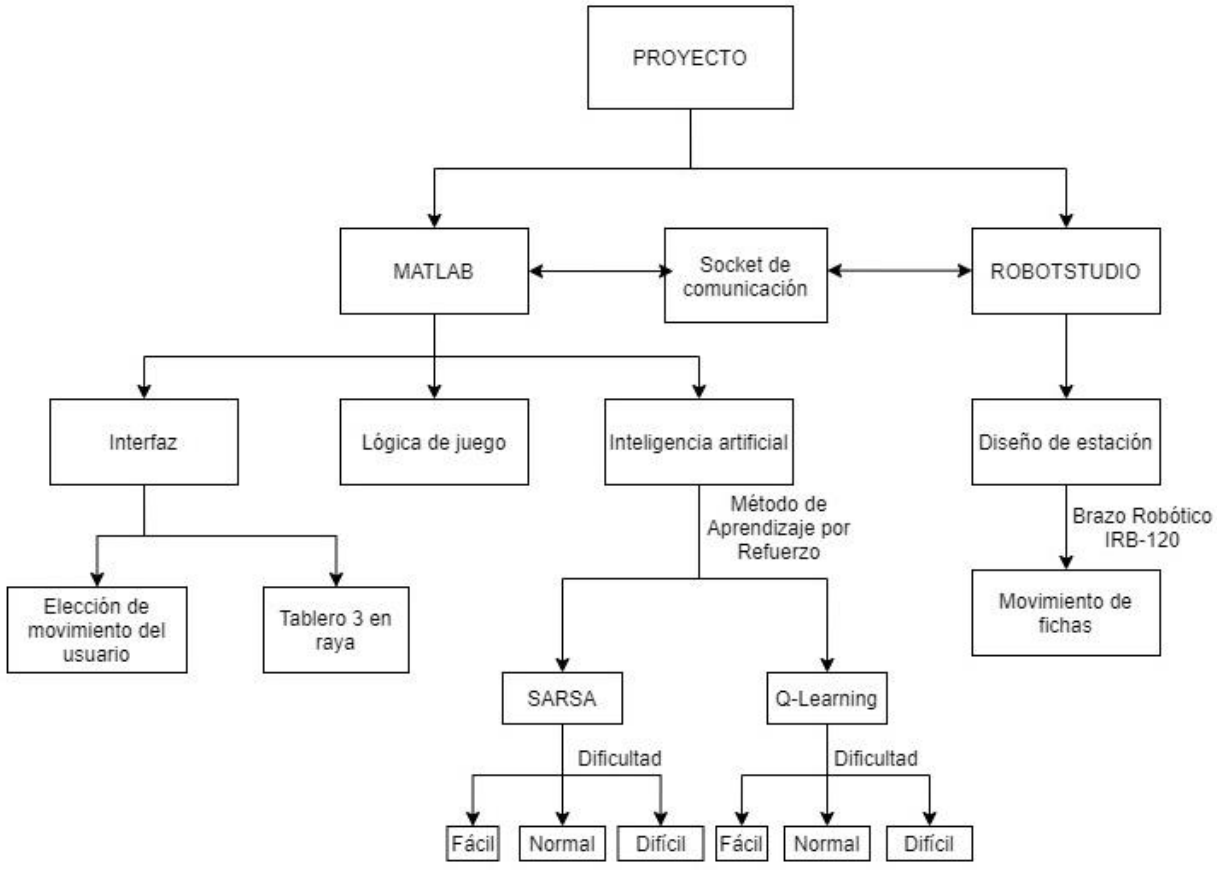

Figura 1.9: Diagrama del proyecto [6]

1.6 CONOCIMIENTOS PREVIOS

Los conocimientos básicos necesarios para la realización del proyecto son los siguientes:

- Conocimiento sobre sistemas robotizados, protocolos de comunicación y control robótico.

- Para realizar cualquier proyecto con casi cualquier herramienta robótica son necesarios conocimientos de programación. En este proyecto en concreto se utilizan los lenguajes: RAPID de ABB y Matlab.

- Conocimientos básicos sobre diseño 3D para el diseño de la estación para el apartado simulado.

- Conocimientos sobre Inteligencia artificial, más en concreto sobre el funcionamiento del "Reinforcement Learning" a nivel medio o avanzado, para poder implementar una IA basada en los métodos "Q_Learning" y "SARSA".

CAPÍTULO 2:

INTELIGENCIA ARTIFICIAL:

APRENDIZAJE POR REFUERZO

2.1 INTRODUCCIÓN

La mayoría de las herramientas necesarias para la realización de la inteligencia artificial basada en autoaprendizaje en este proyecto, vienen incluidas en la "Toolbox" de "Reinforcement Learning" de Matlab [7].

Esta "Toolbox" proporciona los algoritmos, aplicaciones y funciones necesarios para realizar la inteligencia artificial.

El aprendizaje automático, también conocido como "Machine Learning" en inglés, es una rama de la inteligencia artificial que proporciona la habilidad de aprender a las máquinas, a partir del análisis de datos, identificar patrones y tomar decisiones con mínima intervención humana.

Este método se puede dividir en tres categorías, que son el "Unsupervised Learning", el "Supervised Learning", "Deep Learning" y el "Reinforcement Learning". En este último se centra el proyecto.

El aprendizaje por refuerzo, llamado "Reinforcement Learning" en inglés, es un software que toma acciones y realiza observaciones dentro de un entorno. Se basa en un sistema de recompensas, ya que su objetivo es aprender a actuar de una manera que logre maximizar el número de recompensas esperadas. En conclusión, el agente actúa en el entorno y aprende por el método de prueba y error, maximizando sus recompensas. Por tanto, este método usado a largo plazo obtiene una gran capacidad de mejora, debido al método descrito de prueba y error [8].

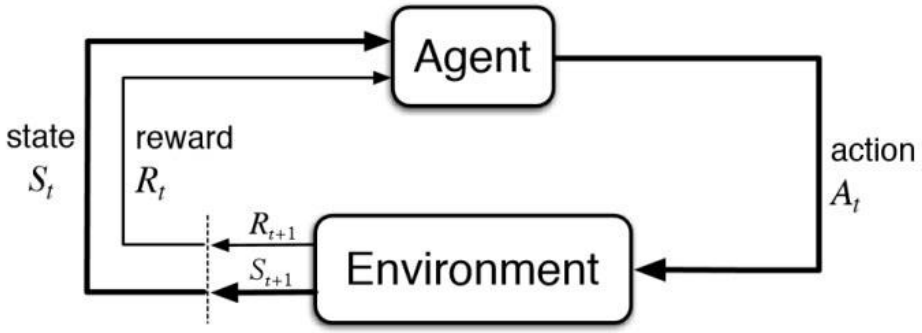

Figura 2.1: Esquema básico aprendizaje por refuerzo

2.2 ELEMENTOS DEL APRENDIZAJE POR REFUERZO

Se distinguen cuatro elementos principales en un sistema de aprendizaje por refuerzo: la recompensa, la función de valor, la política y un modelo para el entorno.

2.2.1 Recompensa

La recompensa define el objetivo en un problema de aprendizaje por refuerzo. En cada paso de la ejecución del programa, se provee al agente de una recompensa en forma numérica. El objetivo del agente es lograr maximizar la recompensa total. La recompensa tiene la capacidad de determinar qué resultados o estados son negativos, neutros o beneficiosos para el agente. Por ejemplo, en el juego tres en raya, si se decide poner la última ficha y con ella se gana la partida, se obtiene una recompensa beneficiosa, si después de realizar un movimiento se pierde la partida esta recompensa seria negativa, y por último, si no influye directamente en el resultado, la recompensa será neutra. Las recompensas han de ser cuantificadas en base a una función que dependa del estado del entorno y las acciones seleccionadas.

2.2.2 Función de valor

Esta función se define como la recompensa máxima que el agente será capaz de acumular desde el estado en el que se encuentra. Se diferencia con la recompensa, en que la recompensa es inmediata, mientras que la función de valor depende del tiempo, de visitar ciertos estados cuando se han visitado otros debido a recompensas disponibles de estados. En otras palabras, un estado otorgará una recompensa inmediata de pequeño valor, no obstante, es un punto de partida para llegar a otros estados que otorgarán una recompensa mayor en un futuro, o al contrario.

2.2.3 Política

Esta determina el comportamiento del agente en un estado concreto, determina la acción óptima para cada estado. En términos de psicología se asemejaría a las asociaciones estímulo respuesta. Como conclusión, la política del aprendizaje por refuerzo puede ser una función que nos da la acción a realizar o una tabla con el par estado-mejor acción.

2.2.4 Modelo

Es un componente opcional en el aprendizaje por refuerzo, con el cual, se puede estudiar a qué estados se llegan cuando se eligen determinadas acciones, al estar situados en estados previos. Los métodos libres de modelos son los más comunes, son explícitamente agentes de ensayo-error.

2.3 TIPOS DE APRENDIZAJE POR REFUERZO

Podemos diferenciar el aprendizaje por refuerzo activo y pasivo. Éstos están libres de modelo [9].

2.3.1 Aprendizaje por Refuerzo Pasivo

En este caso, el agente aprende de lo observado, no tiene ninguna decisión sobre la política, ya que ésta debe ser fijada con anterioridad y no varía a lo largo del entrenamiento. El agente aprende de los valores de estado mediante la evaluación directa y la diferencia temporal. La primera consiste en realizar experimentos y calcular el valor medio de la recompensa obtenida en cada estado. Por otro lado, el método de diferencia temporal se basa en una regla de actualización "bootstrapping", que consiste en estimar los valores de las recompensas inmediatas y futuras de forma similar a la programación dinámica.

2.3.2 Aprendizaje por Refuerzo Activo

El aprendizaje por refuerzo activo no contempla una política fija, por el contrario, se actualizan progresivamente y buscan converger a una política óptima. Podemos diferenciar dos algoritmos que derivan del método de diferencia temporal: el algoritmo "SARSA" y el algoritmo "Q-Learning".

2.3.2.1 Algoritmo Q_Learning

Este método es un ejemplo de una técnica de aprendizaje por refuerzo sin modelo llamada método de diferencia temporal. Su nombre deriva del uso de la diferencia entre los valores de Q en dos pasos de tiempo sucesivos para actualizar el valor de Q.

La regla de actuación de los valores es la siguiente:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)) \qquad (1)$$

Es un algoritmo "*off-policy*", debido a que actualiza sus valores Q tomando el mejor valor de Q del par estado-próximo o par estado-acción "*greedy*" ($máx_a Q(S_{t+1},a)$), lo que se traduce en que estima la recompensa descontada total de los pares estado-acción de forma similar a una política "*greedy*", contrariamente al "SARSA".

Explicando el algoritmo, la función de valor de acción aprendida, Q (S_t, A_t) se aproxima directamente a la función de valor de acción óptima, Q (S_{t+1}, a), independientemente de la política de comportamiento que se esté siguiendo, siempre que todos los pares de acciones de estado sean visitados y actualizados durante el aprendizaje.

Una vez que el agente actúa en el estado S_t, recibe una recompensa inmediata R_t y pasa al siguiente estado S_t. Una vez en el estado S_{t+1}, el agente conoce todas las acciones disponibles en los valores S_{t+1} y Q para todos los pares de acciones del estado en S_{t+1}, utilizando la tabla Q, sin realizar ninguna acción. Ahora, la acción utiliza una de las acciones en s '(según la política de destino) para calcular el valor Q de destino para el par de acciones de la estación (S_t, A_t). Se usa

la diferencia del valor Q objetivo y el valor Q actual para el par de acciones estatales (S_t, A_t) para actualizar el valor Q para ese par. La siguiente acción para actualizar el valor de Q se elige utilizando la política de destino (siempre una política codiciosa para el aprendizaje de Q) en lugar de la política de comportamiento.

Como la política de destino (siempre es la política codiciosa de Q-Learning) para actualizar el valor de Q es diferente de la política de comportamiento, Q-Learning es un algoritmo de aprendizaje fuera de política.

La tasa de aprendizaje alfa (0 menos que alfa menos que 1) determina la velocidad a la que cambia el valor de Q. Si alfa está cerca de cero, el aprendizaje es lento, mientras que si alfa está cerca de 1, el aprendizaje es rápido.

2.3.2.2 Algoritmo SARSA

El algoritmo SARSA deriva del método basado en diferencia temporal, el cual es un método "*on-policy*", por tanto, tiene una política inicial y actúa hasta el final de cada episodio.

La regla de actuación de los valores es la siguiente:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)) \quad (2)$$

Esta actualización se realiza en cada transición de un estado S_t a un estado S_{t+1}. Si S_t es un estado final, sería $Q(S_{t+1}, A_{t+1})=0$. Este método comienza con una política de función Q generada aleatoriamente. Posteriormente se ejecuta un episodio y en cada paso se actualizan los valores. Cuando termina el episodio la política se actualiza, de modo que se eligen las acciones con mayor valor en cada estado para que formen parte de la política.

Este algoritmo se describe como un algoritmo de Q_Learning modificado donde la política de destino es la misma que la política de comportamiento. Los dos pares de estado-acción consecutivos y la recompensa inmediata recibida por el agente durante la transición del primer estado al siguiente determinan el valor Q actualizado, por lo que este método se llama SARSA: Acción Estatal Recompensa Acción Estatal ("*State Action Reward State Action*"). Como la política de objetivos es la misma que la política de comportamiento, SARSA es un algoritmo de aprendizaje de políticas [10].

2.4 CODIGO DESARROLLADO EN MATLAB

Este es el código realizado sobre la inteligencia artificial, y se basa en los principios teóricos explicados anteriormente, donde por el método de aprendizaje por refuerzo, otorgando recompensas en los numerosos entrenamientos se crea una IA con un nivel de dificultad determinado, contra la que nos enfrentaremos posteriormente.

El código utilizado, viene explicado ya en los ficheros de Matlab con la funcionalidad de cada función descrita en el mismo código. Aquí, se describe de forma breve y precisa, cada función necesaria para la implementación de los algoritmos Q_Learning y SARSA:

- **boltzmannGreedyAction***:* Utiliza la distribución de Boltzmann para encontrar la probabilidad de cada acción posible. Describe la distribución de probabilidad para las posibles acciones y el índice de acción para esta distribución. Esta función depende de Q, *actionMatrix* y temperatura.

- **randomAgentMove (possibleActions)**: Esta función realiza el movimiento del agente aleatorio, en ella declaramos "*stateIndex*", "*chooseAction* y *posibleActions".* En primer lugar, se define "*stateIndex",* como el siguiente índice de estado después de que el agente aleatorio hace un movimiento. A continuación, la función de "*chooseAction" es* la elección de una acción a realizar aleatoriamente. Por otro lado, se define "*posibleActions",* como el índice de estado posible del siguiente estado debido a las acciones emprendidas en el estado actual.

- **epsilonGreedyAction**: Utiliza la "*epsilon greedy policy"* para elegir la acción para el estado dado, esto se realiza para garantizar una explotación y exploración suficientes. Esta función depende de Q, "actionMatrix" y épsilon.

- **findActionsforStates (tttTable)**: Esta función como su nombre indica busca acciones para los estados de la "matriz Table". Es una matriz del orden 3^9 x 9 ya que para cada estado hay como máximo 9 acciones posibles, es decir, uno para cada estado vacío (dependiendo de su la siguiente acción para ese estado es X o O) con un máximo de 9 estados vacíos. Para cada "*stateIndex"* en "*actionMatrix"* se almacenan la fila de posibles "*actionIndex"*. Si "*actionIndex"* = 0 significa que ese estado es inalcanzable.

- **findRewardForAgentAction** y **findRewardForUserAction**: Como sus nombres indican, "*findRewardForAgentAction"* busca las recompensas por la acción del agente y "*findRewardForUserAction"* busca las recompensas por la acción del usuario.
  Se determina si "*StateIndex"* es un estado terminal o no (variable "isTerminal"). Si es un estado terminal se definen las recompensas:
  - Si gana, recompensa=+2
  - Si pierde, recompensa=-2
  - Si empata, recompensa=-1
  - Si seguimos en juego, recompensa = 0

  Estas funciones dependen de las variables "*whoWon"*, "*reward"* y "*isTerminal"* (explicada anteriormente).
  Se define la variable "*whoWon",* la cual nos da la información de quién gana la partida, dependiendo de su valor. Si gana Robot (*whoWon*= 0), usuario (*whoWon*= 1), empate (*whoWon*= 2) o el juego sique en proceso (*whoWon*= 3).
  La variable "*reward"* (recompensa), depende de la variable anteriormente descrita "*whoWon", y* toma valores de la siguiente forma:

- *whoWon*= 0, entonces *reward* = +2.
- *whoWon*= 1, entonces *reward* = -2.
- *whoWon*= 2, entonces *reward* = -1.
- *whoWon*= 3, entonces *reward* = 0.

- **stateIndexForTable (tttTable)**: Esta función encuentra el estado de la tabla de 3 en raya dada.

- **tableForStateIndex:** Encuentra una tabla de tic tac toe para un estado en particular.

- **whoseChance:** Esta función otorga valores a *"randAgentChance",* dependiendo del agente, de la siguiente manera:
  - randAgentChance == 1, para el *random agent.*
  - randAgentChance == 0, para el *learning agent.*
  - randAgentChance == -1, si el estado no es accesible.

Ahora, se definen las funciones que realizarán el entrenamiento dependiendo del método de aprendizaje por refuerzo escogido en la interfaz. La diferencia entre ambos métodos viene dada, ya que, en el método Q_Learning, actualiza la estimación a partir de la estimación máxima de posibles próximas acciones, independientemente de la acción que se haya realizado. Mientras que en el método Sarsa, la política de destino es la misma que la de comportamiento, y se basa en las acciones pasadas. Ambos comparten la misma estructura, la diferencia a la hora de su programación es su algoritmo, ya que su ecuación de estados es diferente, como hemos explicado ya, pero a excepción de este cambio su código será el mismo.

Estos ficheros toman el nombre de "Q_Sarsa.m" y "Q_Learning.m". Su función es generar una matriz, llamada Q. Se determina su tamaño en función del numero de estados por casilla posibles. El número total de posibles estados en de 3^9, ya que hay 3 estados posibles por cada una de las 9 casillas: vacío, marcado con X y marcado con O. Por tanto, la matriz Q tendrá un tamaño de 19683x9. Esta matriz ira tomando valores en base a las recompensas obtenidas al realizar su entrenamiento.

El código realizado para ambos algoritmos de IA se explica a continuación:

En primer lugar, definimos las variables que utilizaremos:

- N = Será el número de episodios (partidas) que se realizarán en el entrenamiento, viene determinada por la dificultad elegida en la interfaz de Matlab. De esta forma:
  - Fácil = 1000 entrenamientos.
  - Normal = 10000 entrenamientos.
  - Difícil = 100000 entrenamientos.

  "tttTable" = Es una tabla de 9 posiciones que corresponde con las posiciones del tablero 3 en raya. En cada posición dependiendo del estado puede estar marcada con un 0, 1 o 2. Correspondiendo 0 al estado vacío, 1 a tener la ficha X y 2 a tener la ficha O. La ficha X corresponde al robot y la ficha O al jugador.

  Un ejemplo sería: "tttTable" = [1,2,0,1,0,1,0,2,0], esto correspondería con:

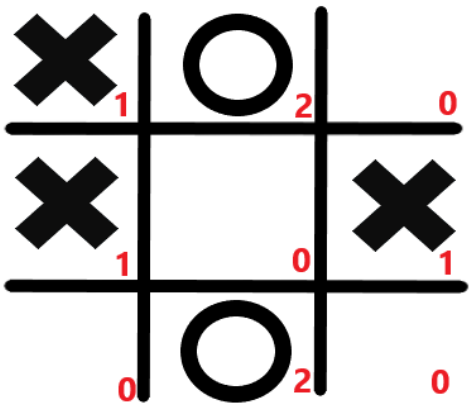

Figura 2.2: Ejemplo estados Tabla

- Y= "discounting factor" (factor de descuento), es episódico y está destinado a terminar en un estado de ganar / perder / empatar.
- "temperatureInitial"
- "temperatureFinal"
- Matriz Q
- "robotWins" = episodios ganados por el robot.
- "userWins" = episodios ganados por el usuario.
- "draw" = empates.

Tras definir las variables y saber el número de episodios para el entrenamiento, se realiza el bucle de episodios, llamado "*Episode Loop*". Este al llegar al estado terminal de cada partida añade un número al contador de resultados ("*robotWins*", "*userWins*" o "*draw*"), dependiendo del estado final de la partida. Esto es así hasta terminar el número de episodios.

Durante la ejecución se irá mostrando en pantalla al finalizar cada partida: la diferencia de Q y el recuento de episodios. La diferencia de Q muestra la recompensa obtenida en ese entrenamiento y el recuento de episodios, informa del número de entrenamiento que se ha realizado.

Finalmente, se muestran en la pantalla de comandos los resultados del autoaprendizaje dando los valores numéricos de cada una de estas variables, que se han ido incrementando con cada partida simulada: *"robotWins"*, "*userWins*" y "*draw*". También se nos generará nuestra matriz Q.

Veamos un ejemplo con 10 episodios:

```
>> Q = Q_Learning;
Diferencia de Q: 1
Recuento de episodios: 1

Diferencia de Q: -5.000000e-01
Recuento de episodios: 2

Diferencia de Q: 1
Recuento de episodios: 3

Diferencia de Q: -1
Recuento de episodios: 4

Diferencia de Q: -5.000000e-01
Recuento de episodios: 5

Diferencia de Q: -5.000000e-01
Recuento de episodios: 6

Diferencia de Q: 1
Recuento de episodios: 7

Diferencia de Q: 1
Recuento de episodios: 8

Diferencia de Q: -1
Recuento de episodios: 9

Diferencia de Q: 7.333333e-01
Recuento de episodios: 10

Robot Gana: 5
Usuario Gana: 2
Empate: 3
```

Figura 2.3: Ejemplo simulacion Q_Learning

Una vez se ha generado la matriz Q, se debe utilizar a la hora de jugar una partida, para ello, se ha implementado un fichero llamado "TurnoIA.m". Este fichero se encarga mediante la matriz Q, en la elección del movimiento del robot. De manera que en función del movimiento del usuario busca en la matriz Q el estado mas codicioso, por tanto, el estado que mas opciones tiene para ganar. Cuantos más entrenamientos haya realizado la IA mejor será la opción elegida en base a la matriz Q, ya que la matriz habrá almacenado más datos y su elección será óptima.

2.5 SIMULACIÓN DE LA IA EN MATLAB

Veamos un ejemplo de la ejecución de cada uno de estos algoritmos. Se utilizan las 3 dificultades, cuanto mayor sea el número de episodios mayor será el porcentaje de victorias obtenido por la IA al ir aprendiendo partida a partida.

Primero, se realiza con el algoritmo "Q_Learning**",** el comando para ejecutar este algoritmo:

Q = Q_Learning;.

1. Dificultad Fácil -> N= 1000

```
Robot Gana: 354
Usuario Gana: 523
Empate: 123
```

Figura 2.4: Q_Learning fácil

Porcentaje de victoria IA = 35.4%

2. Dificultad Media -> N= 10000

```
Robot Gana: 5977
Usuario Gana: 2855
Empate: 1168
```

Figura 2.5: Q_Learning normal

Porcentaje de victoria IA = 59.77%

3. Dificultad Difícil -> N= 100000

```
Robot Gana: 81396
Usuario Gana: 9892
Empate: 8712
```

Figura 2.6: Q_Learning difícil

Porcentaje de victoria IA = 81,396%

Ahora se realiza el mismo procedimiento con el algoritmo **SARSA**; el comando para ejecutar este algoritmo es: *Q = Q_Sarsa;*.

1. Dificultad Fácil -> N= 1000

```
Gana Robot: 328
Gana Usuario: 558
Empate: 114
```

Figura 2.7: SARSA fácil

Porcentaje de victoria IA = 32.8%

2. Dificultad Media -> N= 10000

```
Gana Robot: 5098
Gana Usuario: 3823
Empate: 1079
```

Figura 2.8: SARSA normal

Porcentaje de victoria IA = 50.98%

3. Dificultad Difícil -> N= 100000

```
Gana Robot: 62493
Gana Usuario: 26572
Empate: 10935
```

Figura 2.9: SARSA difícil

Porcentaje de victoria IA = 62.493%

Como conclusión, tras realizar numerosas simulaciones con los distintos niveles de dificultad, al aumentar el número de episodios aumenta la dificultad, ya que, el porcentaje de victoria de la IA aumenta considerablemente. También, se observa que por el método Q_Learning se suele obtener una mayor dificultad de la IA (mayor porcentaje de victoria) en comparación con el método SARSA, realizando el mismo número de episodios en ambas. Aunque esto puede variar ya que depende de algunos factores de aleatoriedad, y no es una diferencia constante.

2.6 GENERAR MATRICES Q

La Inteligencia artificial se debería realizar antes de ejecutar el proyecto, para así, no esperar a que se realicen los entrenamientos, que pueden tardar varios minutos en el caso de tener una dificultad avanzada.

Se ha decidido realizar todas las matrices Q que se pudiesen necesitar a la hora de ejecutar el programa y guardarlas, para así solo tener que cargar sus valores al iniciar el programa. A continuación, se detalla este proceso.

En primer lugar, se deben pensar cuantas matrices diferentes son necesarias. En este caso, al tener tres dificultades y dos modos de entrenamiento, se necesitan seis matrices Q. Para ello, se crea el script "GenerarQ.m".

El funcionamiento de este código es sencillo, su estructura se basa en, definir primero el número de entrenamientos con la variable "N", posteriormente ejecutar la función "Q_Learning" y guardar su valor en la variable "Ql (nivel de dificultad)" del mismo modo a continuación con el método SARSA. Como se observa en la Figura 2.10.

```
%Facil
N = 1000;
Ql_Facil = Q_Learning;
Qs_Facil = Q_Sarsa;

%Normal
N = 10000;
Ql_Normal = Q_Learning;
Qs_Normal = Q_Sarsa;

%Dificil
N = 100000;
Ql_Dificil = Q_Learning;
Qs_Dificil = Q_Sarsa;
```

Figura 2.10: Código generar Qs

Tras generar estas matrices, se deben guardar para posteriormente solo tener que cargarlas al ejecutar el programa. Para ello se utilizan las siguientes sentencias:

1) Guardar las matrices generadas:
   >> save Q_matrices.mat Ql_Facil Ql_Normal Ql_Dificil Qs_Facil Qs_Normal Qs_Dificil
   De esta forma guardamos en el archivo "Q_matrices.mat" las 6 matrices generadas.

2) Cargar las matrices guardadas:
   >> load('Q_matrices.mat')

   Así, se carga el contenido del fichero "Q_matrices.mat", el cual contiene las 6 matrices generadas anteriormente; esta instrucción se ejecutará al iniciar el programa.

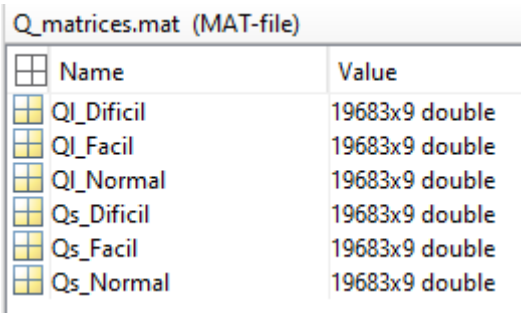

Figura 2.11: Matrices Q guardadas

Con esto, se puede concluir la parte de inteligencia artificial, donde se ha explicado la toma de decisiones del robot en base al aprendizaje por refuerzo y cómo generar las distintas matrices Q necesarias para cada dificultad de juego y modo de entrenamiento.

CAPÍTULO 3:

LÓGICA DE JUEGO

3.1 INTRODUCCIÓN

En este capítulo se explica la lógica de juego realizada en Matlab para el juego 3 en raya. Como ya se ha explicado anteriormente, si ambos jugadores poseen un alto nivel de juego, la partida acabará siempre en empate. Por ello, en este proyecto se ha decidido incluir varias dificultades; por tanto, si estamos en una dificultad baja o media es muy probable que, si juega una persona con habilidad gane la partida, pero si se juega en un nivel avanzado, será imposible ganar la partida, pero si será fácil lograr el empate si juega un usuario con un alto nivel la partida. Por tanto, se pueden ver todos los resultados posibles en función de la dificultad elegida y el nivel del usuario. Esto da como máximo 9 turnos para terminar un juego, ya que en 9 turnos se terminarían las casillas para colocar ficha.

La ficha del usuario será la de color rojo y la del robot la de color verde.

A continuación, se describen los pasos realizados y la estructura del programa en Matlab.

3.2 MODO DE JUEGO

El modo de juego realizado para este proyecto es el del usuario contra el robot. El usuario es controlado por la persona y el robot que corresponde con el brazo robótico IRB120, que tomara sus decisiones en función de la IA. Siempre comenzará el usuario a colocar ficha, debido a como se ha realizado la respuesta de la inteligencia artificial.

3.3 MECÁNICA DEL JUEGO

El método utilizado para fijar la posición de las fichas es mediante la interfaz de Matlab creada con la herramienta GUI, de la que se trata más adelante. A la parte lógica del programa llegará una tabla de 9 posiciones, que corresponde a las 9 casillas del tablero del juego. A partir de esta tabla, el programa determinará si el juego ha finalizado, continúa o quién ha ganado.

Esta tabla funciona mediante 3 estados, dependiendo del número que haya en cada posición determinará qué hay en una casilla. Los tres estados diferentes y cómo se han descrito en este programa se presentan a continuación:

Estado ficha usuario -> 2

Estado ficha robot -> 1

Estado casilla vacía -> 0

Por tanto, cada vez que el usuario coloque una ficha, el número de la posición de la tabla correspondiente a esa casilla cambiará de 0 a 2, o si es el robot el que coloca su ficha cambiaría de 0 a 1.

Poniendo un ejemplo, inicialmente la variable "Table" se declara de la siguiente forma:

Table: [0 0 0 0 0 0 0 0 0]

Comienza jugando el usuario, este decide poner su ficha en la primera casilla, por tanto, la tabla cambia de la siguiente forma:

Table: [**2** 0 0 0 0 0 0 0 0]

Ahora le toca responder al robot, el cual responde en la casilla central del tablero, lo que correspondería a la posición central de la tabla, posición 5:

Table: [2 0 0 0 **1** 0 0 0 0]

Ahora el jugador decide posicionar en la segunda casilla, colocando así el estado 2 en la segunda posición de la tabla:

Table: [2 **2** 0 0 1 0 0 0 0]

La IA responde posicionándose en la última casilla del tablero, lo que corresponde a la última posición de la tabla:

Table: [2 2 0 0 1 0 0 0 **1**]

Por último, para ganar la partida el usuario decide colocar su ficha en la tercera casilla:

Table: [2 2 **2** 0 1 0 0 0 1]

Con lo que gana la partida, entonces aparece el mensaje de que el usuario ganó la partida y ésta termina.

Por cada cambio en la variable "Table", se enviará la información al programa realizado en RobotStudio para el movimiento de cada ficha y se actualizará el tablero realizado en la interfaz gráfica, lo que se explicará posteriormente.

3.4 INTRODUCCIÓN DE MOVIMIENTOS

La elección de movimientos se realiza mediante la interfaz gráfica que se ha diseñado y que se explica más adelante. Habrá nueve botones, los cuales corresponden a cada casilla del tablero. De forma sencilla e intuitiva el usuario deberá pulsar el botón correspondiente a la casilla en la que quiere colocar su ficha.

El problema viene a la hora de pulsar una casilla donde ya hay una ficha colocada; la solución que hemos realizado para este problema es desactivar cada casilla en la que haya una ficha posteriormente de la colocación de ésta, por tanto, cuando el usuario o robot coloquen ficha en una casilla, el botón correspondiente a esta casilla desaparecerá como se puede observar en la figura 3.4.

Figura 3.1: Introducción movimientos GUI

3.5 MOVIMIENTO DE FICHAS EN INTERFAZ GRÁFICA Y ROBOTSTUDIO

En este apartado se explica cómo le llega la información de cada movimiento al realizarse, ya sea del usuario o de la IA a la interfaz creada en Matlab y al software donde el brazo robótico moverá las fichas.

En primer lugar, se definen tres nuevas variables que actuarán como variables auxiliares para captar en una tabla solo el último movimiento realizado, estas son:

Tr = [0 0 0 0 0 0 0 0 0]; -> Posición aislada del movimiento del robot.

Tp = [0 0 0 0 0 0 0 0 0]; -> Posición aislada del movimiento del usuario.

T3 = [0 0 0 0 0 0 0 0 0]; -> Variable para cálculo**.**

Estas variables, junto a la variable "*Table"* son utilizadas en la función "TurnoIA". Su código es el siguiente en cada turno:

```
Tp=Table-T3; %Guardar en Tp solo el movimiento actual
FilaColumna;
```

//Elección del movimiento de la IA

```
Tr=Table-(Tp+T3); %Guardar en Tr solo el movimiento actual
T3=Table;
FilaColumna;
```

Se explica su utilidad con un ejemplo, y al final se explica la función de *FilaColumna*.

- Turno 1:

Usuario coloca ficha en la posición 2 -> Table: [0 **2** 0 0 0 0 0 0 0]

Tp =Table-T3 = [0 2 0 0 0 0 0 0 0] - [0 0 0 0 0 0 0 0 0] = [0 **2** 0 0 0 0 0 0 0].

Está aislado el movimiento del usuario en Tp

Se pasa a "FilaColumna*"*.

IA responde colocando ficha en casilla 5 -> Table: [0 **2** 0 0 **1** 0 0 0 0].

Tr=Table-(Tp+T3) = [0 2 0 0 1 0 0 0 0] – ([0 2 0 0 0 0 0 0 0] + [0 0 0 0 0 0 0 0 0]) =

= [0 0 0 0 **1** 0 0 0 0]

T3=Table = [0 **2** 0 0 **1** 0 0 0 0].

Está aislado el movimiento del robot en Tr

Se pasa a "FilaColumna*"*.

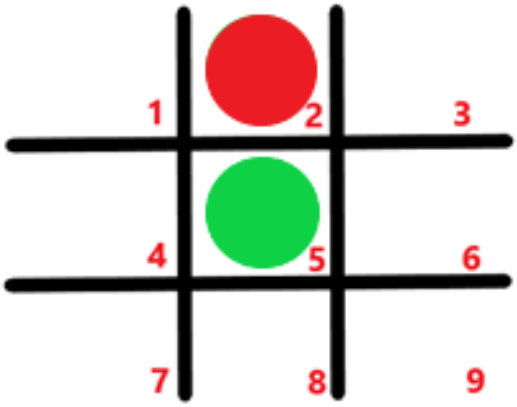

Figura 3.2: Ejemplo partida Turno1

- Turno 2:

Usuario coloca ficha en la posición 7 -> Table: [0 2 0 0 1 0 **2** 0 0]

Tp =Table-T3 = [0 2 0 0 1 0 2 0 0] - [0 2 0 0 1 0 0 0 0] = [0 0 0 0 0 0 **2** 0 0].

Está aislado el movimiento del usuario en Tp

Se pasa a "FilaColumna".

IA responde colocando ficha en casilla 6 -> Table: [0 2 0 0 1 **1** 2 0 0].

Tr=Table-(Tp+T3) = [0 2 0 0 1 1 2 0 0] – ([0 0 0 0 0 0 2 0 0]+ [0 2 0 0 1 0 0 0 0]) =

= [0 0 0 0 0 **1** 0 0 0].

T3=Table = [0 2 0 0 1 1 2 0 0].

Está aislado el movimiento del robot en Tr

Se pasa a "FilaColumna".

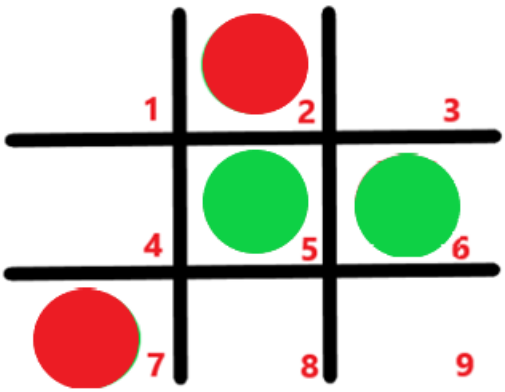

Figura 3.3: Ejemplo partida Turno2

Este proceso se repetiría hasta finalizar una partida. Ahora, se explica el script que es llamado después de cada movimiento, "FilaColumna". En este fichero se inician las variables fila y columna con valor 0. Posteriormente se separan con la función "if" los turnos del robot y del usuario, de tal manera que, si se estaba en el turno del jugador al entrar a este código, se entrará en "if jug==2", por el contrario, se entrará en "if jug==1". En cada una de las anteriores funciones hay nueve funciones "if", que determinan una posición de la tabla. Si se estaba en el turno del robot, corresponderá a la tabla Tr, por el contrario, estando en el turno de la persona corresponderá a la tabla Tp. Dentro de estos se da un valor a las variables fila y columna, dependiendo de la posición de la ficha, por ejemplo, si estaba en la posición central del tablero, el valor de fila será 2 y el de columna también será 2. Este es un ejemplo de una función "if" realizada:

```
    if Tr(2)==1 //Posición 2 de la tabla Tr igual a 1
fila=1;
columna=2;
    end
```

Después de definir el valor de estas variables y salir de las funciones "if", se realiza la llamada a los ficheros:

```
ActualizarGUI;
MovimientoRobotStudio;
```

Estos se basan en el valor de las variables jug, fila y columna. Mediante la variable jug se sabe el turno del jugador que mueve ficha y mediante las variables fila y columna, donde se debe mover la ficha.

El fichero "ActualizarGUI" actualizará el tablero que se ha diseñado en la interfaz gráfica de Matlab, marcando la casilla donde se ha colocado la ficha. Sin embargo, el fichero "MovimientoRobotStudio" manda las instrucciones al software de ABB para realizar el movimiento mediante el brazo robótico y coloca las fichas en el tablero.

3.6 FIN DEL JUEGO

El fin del juego llega cuando se logra por parte del usuario o del robot tener tres fichas en raya, horizontal, vertical o diagonalmente, o cuando se completan todas las casillas y se llega al empate.

El programa determina el final del juego cuando se llega a uno de los tres estados terminales, por tanto, cuando detecta que se encuentra en el estado terminal de la partida, entra en una función "Switch Case", donde dependiendo del valor de la variable "*whoWonIfTerminal",* se llega a uno de los tres estados terminales. Estos son los valores que puede tomar:

- "whoWonIfTerminal = 0" -> Victoria del robot.
- "whoWonIfTerminal = 1" -> Victoria del usuario.
- "whoWonIfTerminal = 2" -> Empate.

Dentro de cada caso, hay un mensaje que da el resultado por la ventana de comandos de Matlab, gracias a la función "*fprintf"* y, al mismo tiempo, saltará una ventana emergente con el resultado sobre la interfaz de programa, gracias a la función "*h = warndlg()".*

Una vez llegados a este punto final de la partida, hay dos opciones. En primer lugar, si no se quiere seguir jugando, se cierra el programa. En segundo lugar, si se quiere jugar un nuevo juego, se pulsa el botón "Jugar de Nuevo" y automáticamente el robot recogerá las fichas y se reiniciará el tablero en la interfaz. Ahora, se podrá cambiar la configuración inicial de nuevo, pero esta vez al presionar el botón "Conectar", solo se aplicarán los cambios de configuración, ya que no es necesario volver a realizar la conexión.

3.7 DIAGRAMA DE UNA PARTIDA

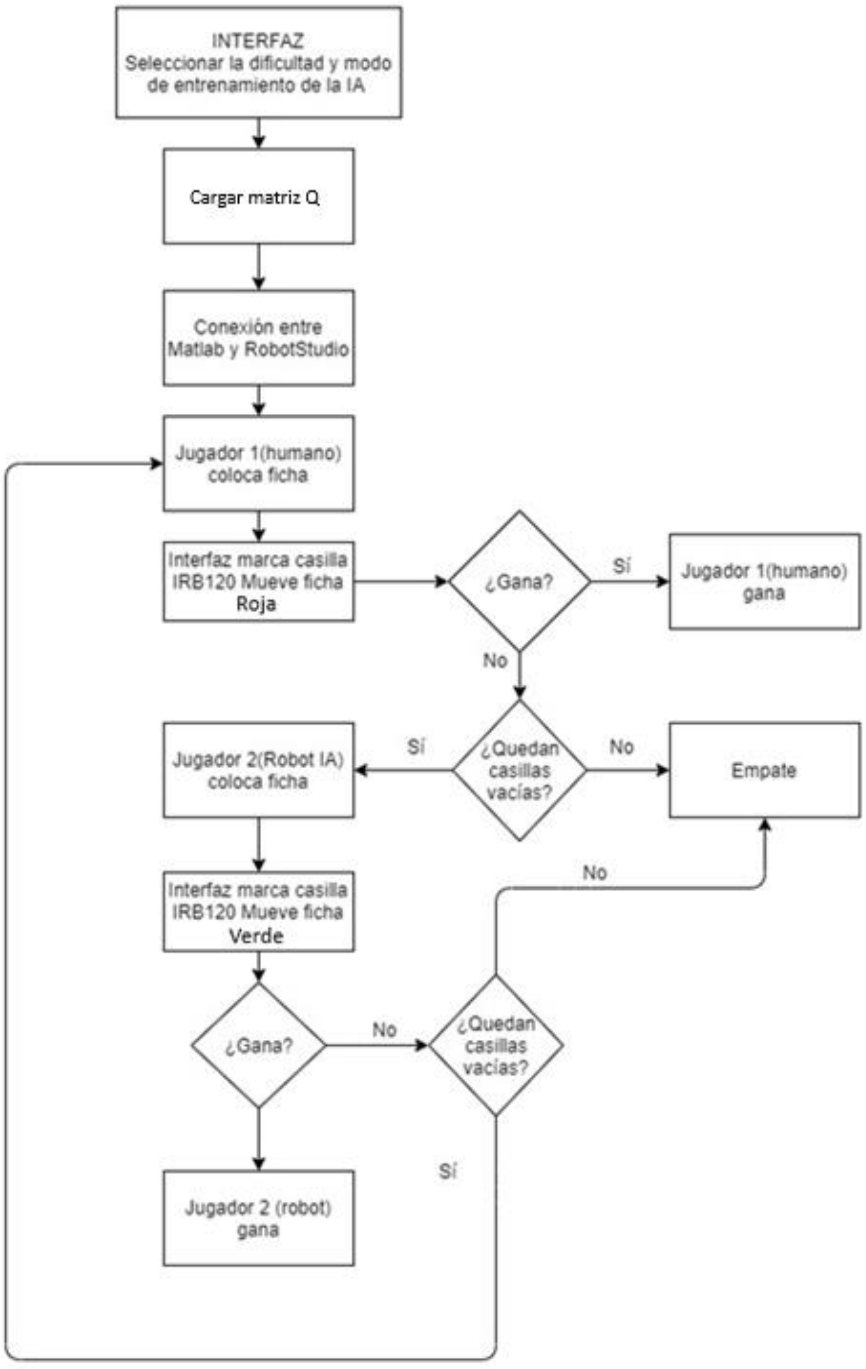

Figura 3.4: Diagrama de una partida [4]

CAPÍTULO 4:

ROBOSTUDIO

4.1 INTRODUCCIÓN

En este capítulo se describe el proceso realizado en RobotStudio para este proyecto. Este apartado se basa en el TFG realizado por David Paz Hernández, llamado "Implementación del juego 3 en raya mediante el Robot IRB120." [2], tanto la creación de la estación, movimientos y trayectorias como la comunicación del socket con Matlab.

RobotStudio es un software de simulación y programación offline de la empresa ABB. La programación offline es la mejor manera de maximizar el retorno de la inversión para los sistemas de robot. Este software ofrece una réplica digital completa (Digital Twin) de activos o sistemas físicos para que se pueda ver lo que sucede en su línea de producción de forma remota, lo que proporciona ventajas como arranque rápido, transiciones fluidas e incremento de productividad [11].

En este software de ABB, se puede realizar un programa independientemente del robot real, una vez finalizado se puede subir al robot real para su ejecución.

En cuanto a la simulación, este programa se basa en "ABB VirtualController", que permite realizar las simulaciones de manera muy realista, utilizando programas de robot reales y archivos de configuración idénticos a los utilizados en el taller.

Este software se basa en la programación en lenguaje RAPID, un lenguaje de programación textual de alto nivel, desarrollado también por la empresa ABB.

Un programa RAPID, es una secuencia de instrucciones que controlan el robot y en general consta de las siguientes partes:

1. Rutina principal -> Main, donde se inicia la ejecución.
2. Conjunto de subrutinas, divide el programa en partes, a fin de obtener un programa modular.
3. Datos del programa que, definen posiciones, sistemas de coordenadas, valores numéricos…

4.2 SOCKET DE COMUNICACIÓN

Esta parte de la aplicación se basa en el proyecto realizado por Marek Jerzy Frydrysiak en su Trabajo de Fin de Grado, que podemos consultar en la bibliografía [1].

Se desarrolla un socket de comunicación de cliente programado en lenguaje C, y el servidor (programado en RAPID) que se ejecuta en el controlador del brazo robot. Para este proyecto se utiliza el mismo socket, pero es fundamental comprender la programación del servidor para poder utilizarlo correctamente. Se centra en los puntos principales del socket, para profundizar más en el tema consultar el TFG del autor.

Las funciones utilizadas para la creación del socket se pueden ver en el código de "TCPipConnectionHandler". Las funciones se encargan de establecer una conexión, de la escucha de datos y de su envío. Estos son los componentes en RAPID:

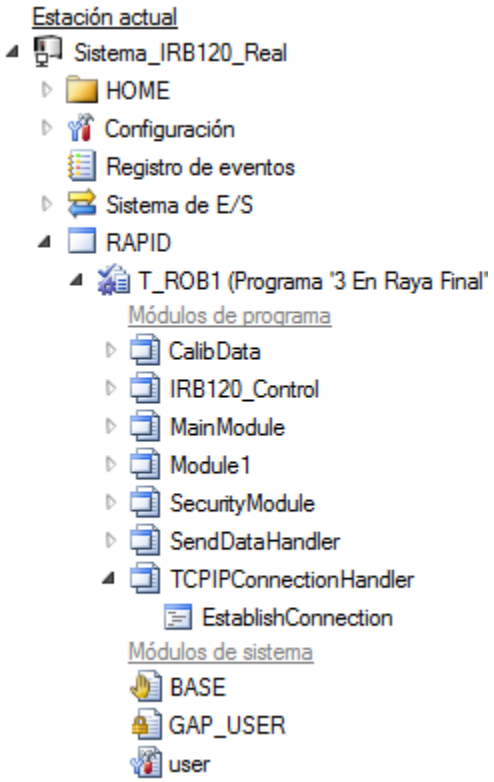

Figura 4.1: Componentes RAPID

Los módulos del programa RAPID son:

- **CalibData**: Este módulo nos permite realizar la elección de la herramienta y modificar la precisión orientación y velocidad necesaria.

- **IRB120_Control:** Lo consideramos como el núcleo principal del programa, ya que, en él se describen los procedimientos a seguir por el robot ante los datos recibidos exteriormente. También se encarga de solucionar o advertir sobre los posibles errores y se encarga del control de datos externos.

- **MainModule**: Este bloque se encarga de los diferentes procedimientos que permiten la conexión y comunicación con el software externo, procesa los datos y peticiones.

- **Module1:** En este módulo se describen las diferentes trayectorias y funciones que realizará el brazo robótico durante el transcurso del programa, definiéndose también todos los puntos usados en estas trayectorias.

- **SecurityModule**: Ofrece seguridad al usuario durante la ejecución del programa. Se puede comprobar las limitaciones de los movimientos del robot. También nos permite realizar cambios en los límites mediante aplicación del cliente.

- **SendDataHandler**: Su función es responder a un estado actual de la ejecución del programa. Contiene el procedimiento "SendDataProcedure", que se encarga de enviar los datos a una aplicación del cliente. El programa servidor informa al usuario de que se

ha establecido conexión, que se han superado las limitaciones de espacio de trabajo o de que se ha producido un error.

- **TCPIPConnection**: Este módulo se encarga de establecer la conexión. Contiene el "EstablishConnection", encargado de establecer la comunicación TCP/IP.

El conjunto de estos módulos del programa conforma el programa RAPID, que realizará la salida y entrada de información por parte del robot. Estos módulos están divididos dependiendo de su función y todo son imprescindibles para trabajar con seguridad con el robot.

4.3 DISEÑO DE LA ESTACIÓN

En estos apartados se describen los elementos necesarios en el entorno de RobotStudio para la realización del programa. Se simulan el área de trabajo del laboratorio de la universidad donde se encuentra el brazo robótico IRB120.

En primer lugar, importamos la mesa sobre la que se encuentra el robot, en cuanto a su colocación se coloca en el centro del espacio. Como se puede ver en la Figura 4.2, esta tiene situado su centro en el centro del plano (x,y) que corresponde al suelo y sube la altura que tiene la mesa.

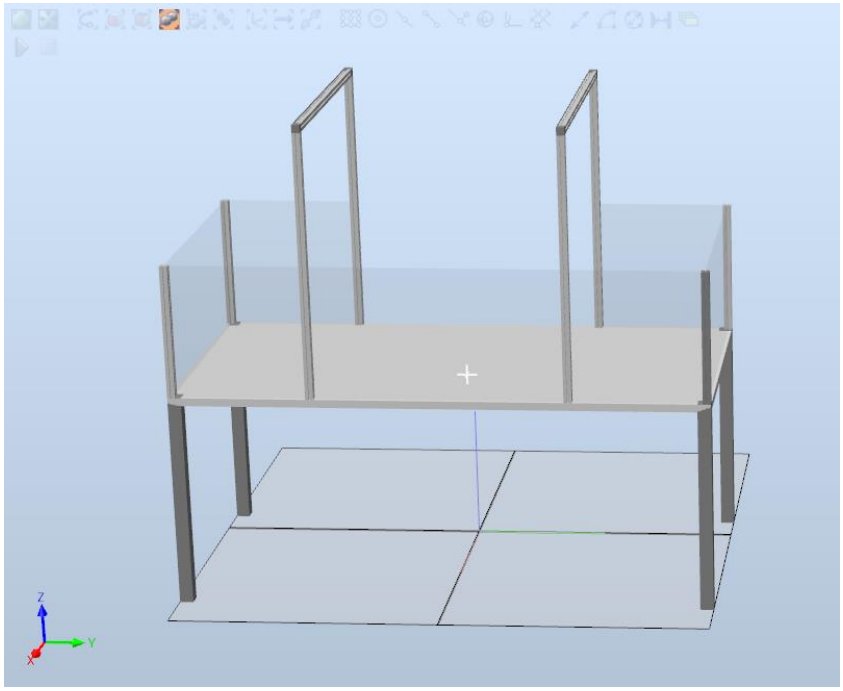

Figura 4.2: Mesa de trabajo

A continuación, se importa la pieza más importante de esta parte del proyecto, el brazo robótico IRB120, una copia del robot obtenida en el laboratorio.

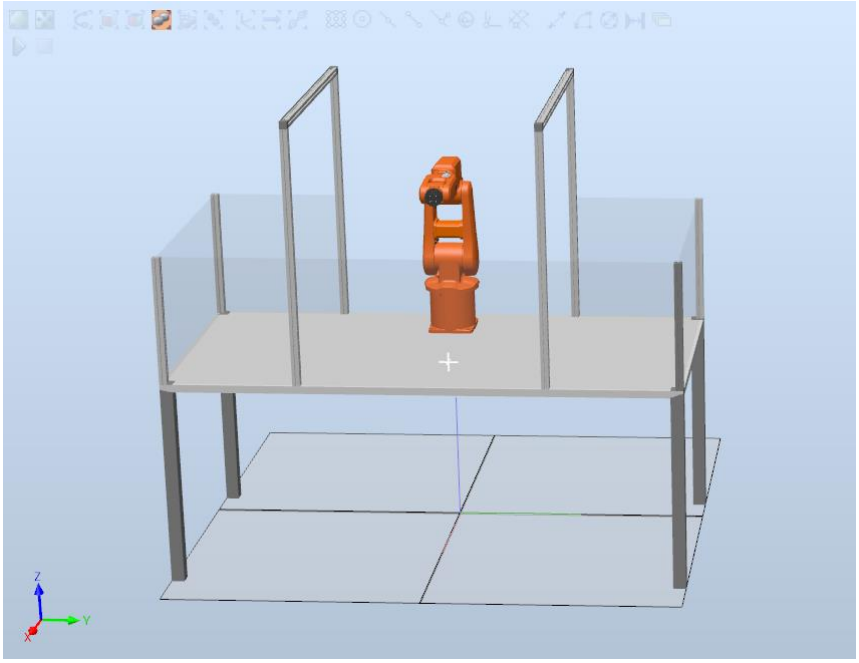

Figura 4.3: Mesa de trabajo con brazo robótico

Tras la colocación del brazo robótico se debe pensar en que herramienta es la adecuada para mover las fichas por el tablero. En este caso se decide utilizar la herramienta ventosa, ya que, otra opción sería la herramienta *gripper*, pero para la función que se necesita es más cómodo trabajar con una ventosa. Se importa la herramienta de ventosa, que corresponde al modelo exacto que hay en el laboratorio.

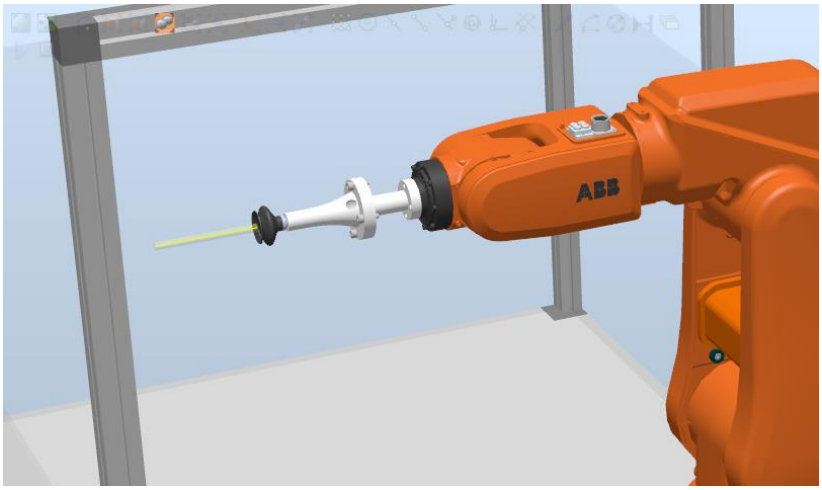

Figura 4.4: Herramienta ventosa acoplada

Una vez se tiene la estación y herramientas listas, debemos añadir lo necesario para la realización del juego 3 en raya. Para ello, necesitamos un tablero 3x3 y las fichas para cada jugador. Debemos tener en cuenta que la casilla del tablero debe ser lo suficientemente grande como para que la ficha entre a la perfección. Hemos decidido utilizar las fichas y tablero ya diseñados hace tiempo para una aplicación similar que estaban en el laboratorio.

El tablero se compone de casillas con dimensión 10x10 cm, por tanto, el tablero en su totalidad tiene unas dimensiones de 30x30 cm. Para situarlo, el centro del robot coincidirá con la línea que divide la segunda y tercera columna y se encontrará a 25 cm en el eje x de distancia del robot, como se puede ver en Figura 4.5.

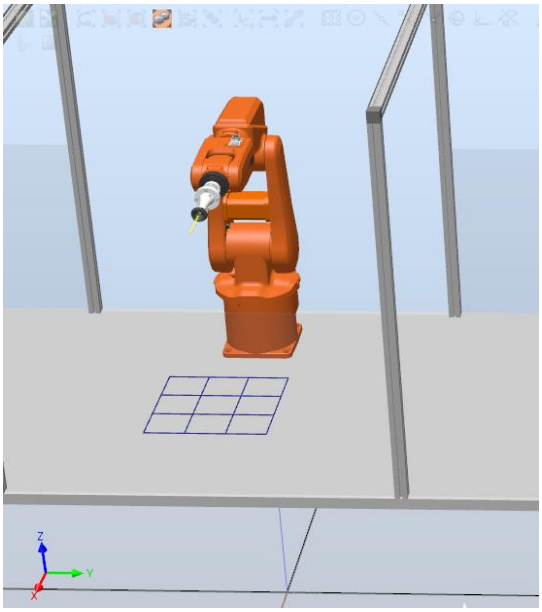

Figura 4.5: Tablero 3 en raya en estación de trabajo

Ahora se pasa al diseño las fichas, las cuales serán diseñadas como cilindros a los que se le dan las dimensiones de las fichas que hay en el laboratorio. Para la ubicación de cada montón de fichas se toma como referencia la ubicación del robot. El centro de las fichas rojas se encuentra en la posición (425,200) mm. El centro de las fichas verdes se encuentra en la posición (300,200) mm, como se puede ver en la Figura 4.6.

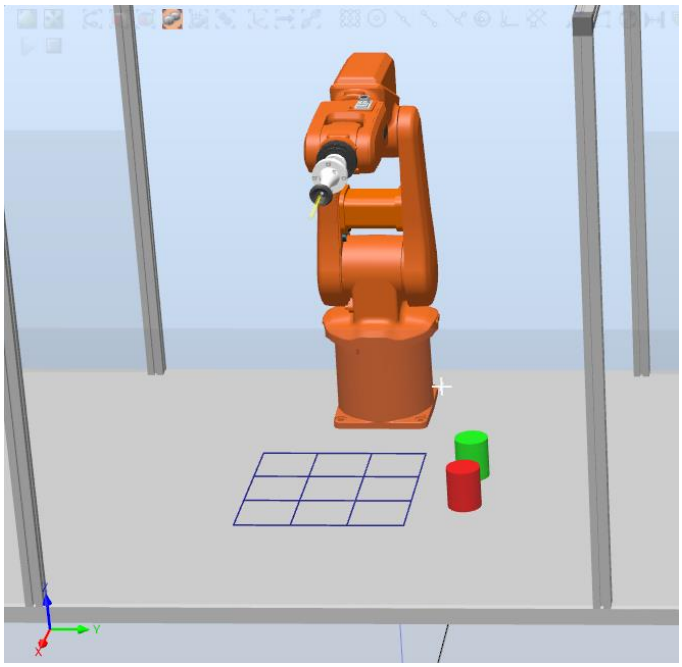

Figura 4.6: Colocación fichas área de trabajo

Se han diseñado 5 fichas rojas y 5 verdes.

Se concluye la creación de la estación en RobotStudio. Para esta aplicación solo se necesitan las herramientas descritas. Ahora se definen los puntos y trayectorias para la realización de movimientos y colocación de fichas en el tablero.

4.4 PUNTOS Y TRAYECTORIAS

Una vez acabado el entorno de trabajo, es necesario definir las posiciones de los objetos empleados. Se establecen estas posiciones para referirnos a ellas desde el código en RAPID y poder definir los distintos puntos y trayectorias necesarias. Los puntos esenciales por definir serán: las 9 posiciones del tablero, las 9 posiciones de las diferentes fichas en sus montones correspondientes, y varias posiciones auxiliares, claves para a la hora de definir las distintas trayectorias.

Como posiciones intermedias se realizan, una centralizada en el tablero, pero a 20 cm de él, una en posición de reposo y otra encima de los montones para acceder a las fichas.

En la Figura 4.7 se observan todas las posiciones descritas para la realización de este proyecto.

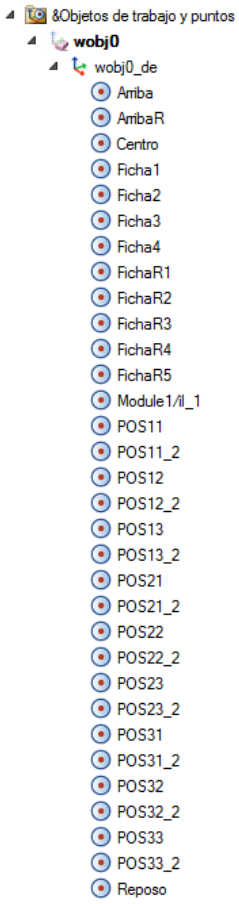

Figura 4.7: Posiciones RobotStudio

Se observa que para cada uno de los 9 puntos se ha descrito otra posición en el mismo lugar, pero elevada para poder acceder a ellos con mayor comodidad. Como se ve el programa cuenta con 32 posiciones en total.

En cuanto a las trayectorias, cada una está formada por el movimiento entre diferentes puntos, de los antes descritos. La lista de trayectorias descritas para este proyecto se muestra en la Figura 4.8.

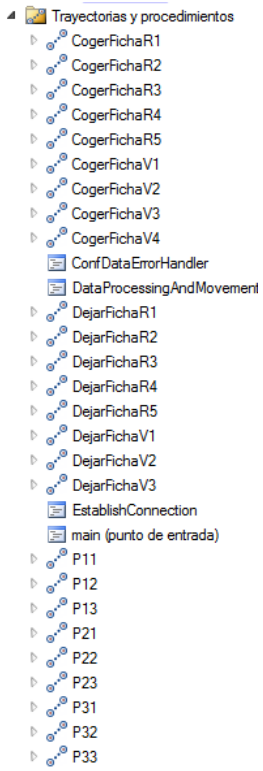

Figura 4.8: Lista de trayectorias

Esto es un ejemplo de la trayectoria definida para ver los pasos seguidos a la hora de realizarla. En este caso, la trayectoria P33.

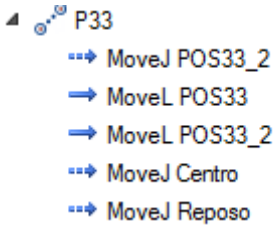

Figura 4.9: Trayectoria P33

En la Figura 4.9 se puede ver el ejemplo de esta trayectoria, la cual se ejecutará tras coger una ficha, por lo que el brazo robótico estará situado justo encima del montón de fichas. Posteriormente, realiza un *MoveJ* (movimiento rápido) a la posición que se encuentra encima de la casilla destino, en este caso la última casilla del tablero, la casilla 9 (o casilla 3,3), luego bajará y volverá a subir, irá al centro del tablero, y de ahí a la posición de reposo como viene descrito. Se observa este proceso en líneas discontinuas en la Figura 4.10.

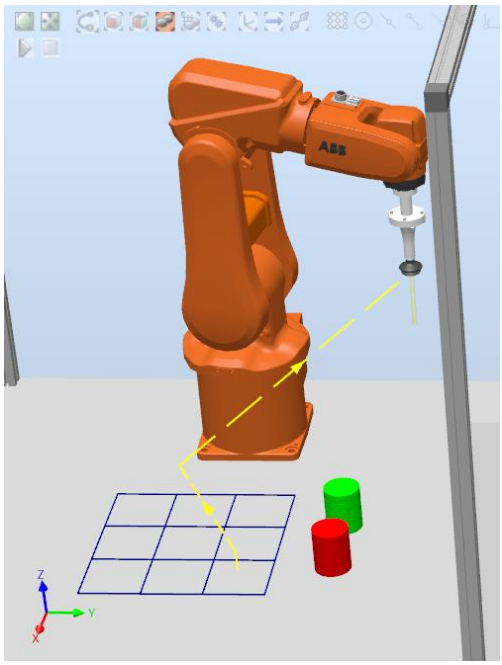

Figura 4.10: Trayectoria P33 Robot

Se han descrito un total de 36 trayectorias, las cuales se pueden ver en la Figura 4.11 sobre la estación.

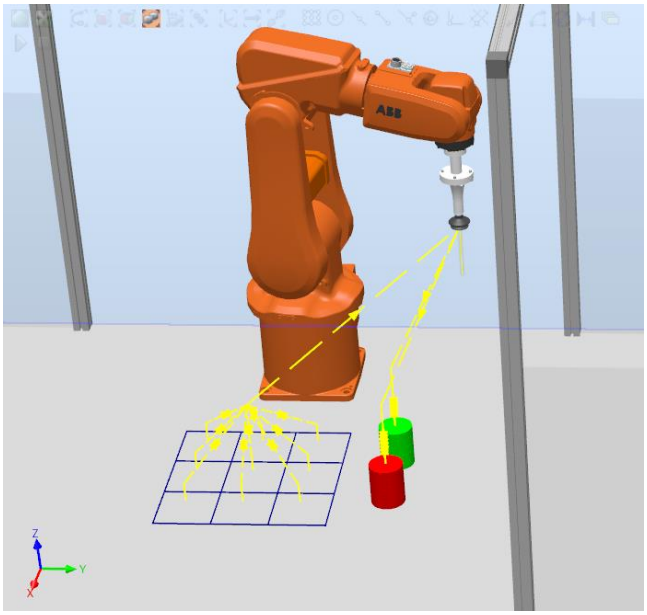

Figura 4.11: Trayectorias estación

A continuación, se describe la parte de programación realizada en RAPID, en la cual se trabaja con estos puntos y trayectorias.

4.5 PROGRAMACIÓN EN RAPID

Después de lo anterior, se definen las posiciones exactas de cada punto de forma rápida, ya que la estructura ya viene dada con anterioridad. En la Figura 4.12 se observan todos los puntos descritos para este proyecto, junto a sus características, con su posición exacta marcada.

```
MODULE Module1

    CONST robtarget POS11:=[[250,-150,15],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS11_2:=[[250,-150,50],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS12:=[[250,-50,15],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS12_2:=[[250,-50,50],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS13:=[[250,50,15],[0,0,1,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS13_2:=[[250,50,50],[0,0,1,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS21:=[[350,-150,15],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS21_2:=[[350,-150,50],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS22:=[[350,-50,15],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS22_2:=[[350,-50,50],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS23:=[[350,50,15],[0,0,1,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS23_2:=[[350,50,50],[0,0,1,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS31:=[[450,-150,15],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS31_2:=[[450,-150,50],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS32:=[[450,-50,15],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS32_2:=[[450,-50,50],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS33:=[[450,50,15],[0,0,1,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget POS33_2:=[[450,50,50],[0,0,1,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];



    CONST robtarget Centro:=[[350,-50,150],[0,0,1,0],[-1,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget Reposo:=[[50,300,350],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget Ficha0:=[[300,200,81],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget Ficha1:=[[300,200,66],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget Ficha2:=[[300,200,51],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget Ficha3:=[[300,200,36],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget Ficha4:=[[300,200,21],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget ArribaR:=[[425,200,200],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget FichaR1:=[[425,200,81],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget FichaR2:=[[425,200,66],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget FichaR3:=[[425,200,51],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget FichaR4:=[[425,200,36],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget FichaR5:=[[425,200,21],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    CONST robtarget Arriba:=[[300,200,200],[0,0,1,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
```

Figura 4.12: Puntos en RAPID

Una vez definidos los puntos, se pasa a las trayectorias, que también han sido diseñadas de forma manual, se modifican y así se dejan de la forma que se quiera. Se modifica la velocidad de movimiento del robot en ellas, para no tener problemas a la hora de llevarlo a la realidad, con un valor tope de v200. A continuación, se fija la precisión con la que se pasa por los puntos, en este caso se marca la precisión "fine".

Por último, para el tipo de movimiento, se usarán los comandos "MoveL" y "MoveJ". El comando "MoveL", realizará el movimiento de un punto a otro, haciendo la punta de la herramienta una trayectoria recta, moviendo solo lo necesario, los eslabones del robot. Sin embargo, el comando "MoveJ" llevará la herramienta de un punto a otro, moviéndose de la manera más sencilla para el brazo robótico. Se utilizará el primero para los movimientos en vertical y el segundo comando descrito para el resto de los movimientos.

La estructura que se usa para definir una trayectoria mediante RAPID es la siguiente:

MoveL/J Punto, velocidad, precisión, herramienta;

Esto se repetirá varias veces en función de por cuantos puntos pase esa determinada trayectoria. Se ve como ejemplo la trayectoria P33, la cual lleva la ficha a la última casilla del tablero, mediante la herramienta ventosa. También se utilizan las sentencias como V0 o V1 que activan o desactivan la ventosa.

```
138     PROC P33()
139         MoveJ Centro,v200,fine,tool_Ventosa_SMC_ZPT32BN_B01\WObj:=wobj0;
140         MoveJ POS33_2,v200,fine,tool_Ventosa_SMC_ZPT32BN_B01\WObj:=wobj0;
141         MoveL POS33,v200,fine,tool_Ventosa_SMC_ZPT32BN_B01\WObj:=wobj0;
142         V0;
143         MoveL POS33_2,v200,fine,tool_Ventosa_SMC_ZPT32BN_B01\WObj:=wobj0;
144         MoveJ Centro,v200,fine,tool_Ventosa_SMC_ZPT32BN_B01\WObj:=wobj0;
145         MoveJ Reposo,v200,fine,tool_Ventosa_SMC_ZPT32BN_B01\WObj:=wobj0;
146     ENDPROC
```

Figura 4.13: Trayectoria P33 RAPID

En varias trayectorias es necesario activar o desactivar la ventosa, para así soltar o coger la ficha, y para ello se utilizan las siguientes sentencias de código:

```
42      PROC V1()
43              WaitTime 1;
44              SetDO DO10_9,1; !ACTIVAR SEÑAL VENTOSA REAL
45              SetDO CogerCilindro, 1;
46              WaitTime 1;
47              SetDO CogerCilindro, 0;
48      ENDPROC
49
50      PROC V0()
51              WaitTime 1;
52              SetDO DO10_9,0; !DESACTIVAR SEÑAL VENTOSA REAL
53              SetDO SoltarCilindro, 1;
54              WaitTime 1;
55              SetDO SoltarCilindro, 0;
56      ENDPROC
```

Figura 4.14: Activar/Desactivar ventosa

Con esto se concluye la parte de programación en RAPID del proyecto, donde se han definido todos los movimientos para que el robot logre simular el juego 3 en raya a la perfección.

4.6 COMUNICACIÓN CON EL ROBOT

Este apartado se centra en la parte de RAPID en el módulo "IRB120_Control", en el cual se encuentran los casos que realiza el robot al recibir los datos de Matlab.

En primer lugar, se definen las variables necesarias durante la comunicación con Matlab, posteriormente se comprueba el estado de la conexión del robot, el dato recibido y se realizan diferentes acciones en función del valor de ese dato recibido.

Se describen ahora los casos que puede realizar el robot al recibir los datos:

1. Movimiento de los ejes.
2. Rotación de los ejes.
3. Posición de los ejes.
4. Movimiento a objetos, procedimientos de seguridad y aplicaciones.
5. Control de velocidad
6. Activación de la ventosa, activa o desactiva la succión de la ventosa.
7. Movimiento a las diferentes posiciones del tablero para dejar las fichas.
8. Movimiento para recoger las fichas de los montones donde se encuentran inicialmente.
9. Movimiento a las diferentes posiciones del tablero para recoger las fichas.
10. Movimiento para depositar las fichas de los montones donde se encontraban inicialmente.

La comunicación se produce de la siguiente manera: Matlab envía el número del caso a ejecutar, y dentro de cada caso el número de trayectoria a realizar. Por ejemplo, Matlab envía los números 8 y 3 y en RAPID se interpreta que dentro del caso 8 es el caso 3, es decir, coger la ficha del montón y sería coger la ficha verde 3.

CAPÍTULO 5:

INTERFAZ GRÁFICA MATLAB

5.1 INTRODUCCIÓN

La interfaz gráfica de usuario, llamada "GUI", del inglés "*Graphical User Interface"*, es un software informático que actúa como interfaz de usuario. La principal función de la GUI consiste en proporcionar un entorno visual sencillo, que permite la comunicación con un determinado software. Estas permiten un control sencillo de las aplicaciones software, mediante el ratón, lo cual elimina la necesidad de aprender un lenguaje y escribir comandos, cuando buscamos ejecutar una aplicación [12].

La GUI incluye controles tales como: botones, controles deslizantes, menús o barras de herramientas.

Antes de que se desarrollaran las GUI, solo las personas con conocimientos de informática podían manejar un computador. En cambio, con las GUI bajó la complejidad de uso de los computadores, con comandos por acciones predeterminadas simbolizadas por elementos visuales muy sencillos de comprender.

La interfaz es un elemento muy importante para los programas, aplicaciones o sistemas operativos. Se estima que un gran número de usuarios abandonan un sitio web debido a la interfaz gráfica, ya que, no está optimizada a sus necesidades y expectativas.

Se puede identificar una buena interfaz gráfica por los siguientes factores:

- Es sencilla de comprender y de usar.
- Es fácil entender y recordar su funcionamiento.
- La información está adecuadamente ordenada mediante iconos, menús…
- Operaciones intuitivas, reversibles y rápidas.
- Los elementos principales son identificables.

La herramienta "*guide" (Gui Development Enviromment)* de Matlab permite diseñar y definir un conjunto de elementos que, de forma fácil e intuitiva, son manejados por el usuario. Esta herramienta permite el desarrollo de interfaces gráficas. Las GUI pueden ser modificadas mediante "*guide"* y programando en el código correspondiente a la interfaz creada en Matlab.

5.2 DISEÑO DE LA INTERFAZ

Antes de crear la interfaz, el programa se manejaba desde la lista de comandos de Matlab, de forma que, el programa pedía una casilla con valor numérico (del 1 al 9), correspondiente con las 9 casillas del tablero y el usuario debía escribir un número en el que no hubiese ya una ficha. Cada turno se mostraba por la ventana de comandos el valor de la variable "*Table"*, viendo así que estado había en cada casilla.

Para mejorar la aplicación se propuso realizar una interfaz gráfica, que contuviese un tablero donde seleccionar el movimiento de fichas del usuario y ver a tiempo real en un tablero cómo va la partida. Esto le aporta comodidad al usuario del programa, ya que, también se decidió incluir la elección del método de entrenamiento y la dificultad del juego que, de no tener interfaz, se deberían ejecutar llamando a sus funciones correspondientes en un orden determinado al igual que a la hora de realizar la conexión con el robot.

En la siguiente figura se puede ver la interfaz de la herramienta "*guide",* para el diseño de nuestra interfaz, en donde, tenemos a la izquierda los iconos de creación de botones, texto, bloques…

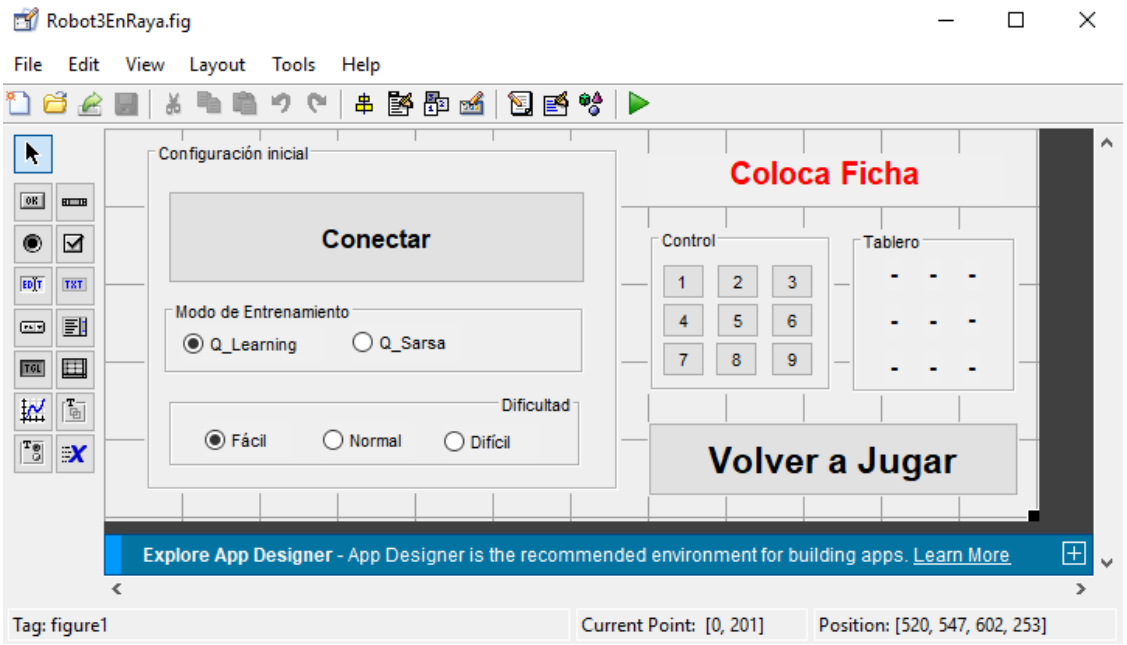

Figura 5.1: Interfaz herramienta guide

Mediante esta herramienta gráfica hemos diseñado la colocación de cada elemento para nuestra interfaz y hemos dado nombre y función a cada uno de ellos.

5.3 FUNCIONABILIDAD DE LA INTERFAZ

Dividiremos la interfaz en dos bloques, un primer bloque en el que definiremos los parámetros iniciales del programa y un segundo bloque en el que el usuario tomara su decisión de movimiento y podremos ver el transcurso de la partida, al mismo tiempo que en el brazo robótico.

Figura 5.2: Interfaz Matlab

Antes de comenzar el juego se debe elegir la dificultad y el modo de entrenamiento. Por ello, en la interfaz se ha habilitado una configuración inicial que contiene lo descrito, la cual se lanza cuando se conecta al robot. Primero se elige el método de entrenamiento por refuerzo, ya sea Q_Learning o SARSA. Al mismo tiempo se selecciona la dificultad a la que se quiere enfrentar el usuario, teniendo tres niveles de dificultad disponibles.

Una vez seleccionadas las características deseadas de dificultad y modo de entrenamiento, hay que presionar el botón de conectar. Al presionar conectar, se selecciona la matriz Q correspondiente a la dificultad y modo de entrenamientos seleccionados y se realiza la transmisión de datos entre ambos softwares (solo la primera vez que se pulsa). Una vez se presiona el botón conectar, ya no será posible volver a cambiar la configuración inicial, hasta finalizar la partida ya que, desaparecerá de la interfaz, como se observa en la Figura 5.3.

Figura 5.3: Interfaz conectada

Es muy importante haber activado el modo de simulación en RobotStudio antes de realizar la conexión entre ambos softwares, ya que, si no está en la posición inicial de la simulación no se realizará la conexión con RobotStudio.

A la hora de elegir la dificultad, se debe saber que, a mayor dificultad mayor número de entrenamientos. Y si se realizase el entrenamiento en la ejecución del programa sería un problema debido al tiempo de espera.

Para realizar el entrenamiento se debería esperar el tiempo que el programa tarda en ejecutar el entrenamiento. Se ha realizado una estimación de tiempos con cada dificultad:

- Fácil: menos de 1 s
- Normal: alrededor de 7 s
- Difícil: aproximadamente 54s

Como se observa, hay una gran diferencia de tiempos entre difícil y las distintas dificultades. Esto es debido a la diferencia de entrenamientos entre las distintas dificultades: la dificultad fácil es de 1000 entrenamientos, la normal es de 10000 y la difícil de 100000.

Para solventar este problema se implementaron las matrices Q de los seis casos posibles en el programa, de esta forma, solo se deben cargar los datos de las matrices y en función de la dificultad y modo de entrenamiento usar una de estas matrices. Así no será necesario la ejecución del entrenamiento durante la ejecución del programa.

En cualquier momento de la partida (mientras el robot no esté en movimiento), se podrá pulsar el botón "Volver a Jugar", el cual ordenará al robot a recoger las fichas del tablero y reiniciará el tablero de la interfaz. Además, permitirá cambiar la configuración inicial, para jugar así de nuevo al juego.

5.4 CÓDIGO GUI EN MATLAB

Al crear la interfaz desde la herramienta "GUIDE", se genera el código correspondiente a esta interfaz, en la cual damos función a cada elemento creado gráficamente.

En este caso el script generado toma el mismo nombre que la interfaz, llamado "Robot3EnRaya.m"

En primer lugar, se genera una función llamada "Robot3EnRaya_OpeningFcn" que arrancará junto a la interfaz, donde se han declarado todas las variables globales necesarias para este proyecto y se cargan las matrices Q, para posteriormente seleccionar la que se quiere usar. Todo ello se puede ver en la figura 5.4.

```
global fila columna jug TurnosVerde TurnosRojo robot Q Table Tr Tp T3
global Ql_Facil Qs_Facil Ql_Normal Qs_Normal Qs_Dificil Ql_Dificil c
fila=0;
columna=0;
TurnosVerde=0;
TurnosRojo=14;
c=0;%Variable para conectar solo la primera vez que pulsamos el botón conectar

Table = [0 0 0 0 0 0 0 0 0];  %Table indica las posiciones del juego
Tr = [0 0 0 0 0 0 0 0 0]; %Posición aislada del movimiento del robot
Tp = [0 0 0 0 0 0 0 0 0]; %Posición aislada del mov de la persona
T3 = [0 0 0 0 0 0 0 0 0]; %Variable para cálculo

load('Q_matrices.mat') %Cargar las 6 matrices Q anteriormente guardadas
```

Figura 5.4: Función Robot3EnRaya_OpeningFcn

A continuación, se ven nueve funciones, cada una corresponde a cada botón del tablero que se ve en la interfaz. Estas nueve funciones siguen exactamente la misma estructura:

```
% --- Al pulsar casilla 1
function B11_Callback(hObject, eventdata, handles)
global fila columna jug Q Table Tr Tp T3
jug=2;
EnableControl=0;
Control;
fila=1;
columna=1;
Table(1) = 2;
ActualizarGUI;
TurnoIA
ActualizarGUI;
EnableControl=1;
Control
```

Figura 5.5: función B11_Callback

Primero, se declaran una serie de variables necesarias. A continuación, se da un valor 2 a la variable jug, ya que siempre que se presione un botón será el turno del usuario y no del robot. Posteriormente, se da un valor a las variables fila, columna y "Table" (i). Después, para actualizar el tablero en la interfaz, se nombra al fichero como "ActualizarGUI". Luego, se ejecutará el turno del robot y se volverá a actualizar el tablero para marcar la casilla seleccionada por el robot.

Los ficheros que se utilizan son:

1) Control

La función del script "control" es mantener desactivados o activados los botones de las casillas y Volver a Jugar, ya que mientras se estén ejecutando las ordenes de mover fichas en el robot el usuario deberá esperar y no poder mover ficha o reiniciar el juego hasta su turno. Esta página de código viene controlada por la variable "*EnableControl*", la cual puede tomar valores de 0 a 2. Su función es la siguiente:

- Sí "*EnableControl*" es igual a 0, se desactivan los botones de las nueve casillas del tablero y el botón Volver a jugar.
- Sí "*EnableControl*" es igual a 1, se activan los botones de las nueve casillas del tablero y el botón Volver a jugar.
- Sí "*EnableControl*" es igual a 2, se desactivan los botones de las nueve casillas del tablero y se activa el botón Volver a jugar.

Para la activación o desactivación de los botones se utiliza la función "set", que tiene la siguiente estructura: set (H, Name, Value). Un ejemplo de uso para deshabilitar el botón correspondiente a la primera casilla sería:

```
set(handles.B11,'Enable','off');
```

Como se ve en primer lugar, se selecciona el botón, luego la acción de habilitar (Enable) y finalmente el estado apagar (off).

2) TurnoIA

La función de "TurnoIA", como su nombre indica es realizar el turno de la inteligencia artificial, se ocupará de decidir en base a la matriz Q seleccionada cual será el posicionamiento de la ficha del robot y de darle la instrucción para mover la ficha. También determinará en caso de final de la partida, cuál ha sido el resultado de ésta.

3) ActualizarGUI

Se ocupa de marcar las casillas en la interfaz, marcando con una O roja si es del jugador y verde si es del robot.

Figura 5.6: Tablero GUI

Para ello se utiliza la función "set" de nuevo, con las siguientes sentencias:

```
set(handles.T11,'String','O');  //Utilizar forma O

set(handles.T11,'ForegroundColor','green'); //Poner color verde

set(handles.T11,'ForegroundColor','red');  //Poner color rojo
```

Ahora se explica la función del botón "conectar". Esta función recoge el modo de entramiento y la dificultad que se ha marcado en la interfaz, para así, seleccionar su matriz Q correspondiente y cargarla para el turno del robot. Se han creado dos variables para recoger estos datos. La variable D recoge la dificultad seleccionada y la variable M que recoge el modo seleccionado. De la siguiente forma:

```
%Obtenemos los datos sobre la dificultad
    Dificultad = get(handles.Dificultad,'SelectedObject');
    switch get(Dificultad,'Tag')
       case 'Facil',   D=1;
       case 'Normal', D=2;
       case 'Dificil',D=3;
    end

%Obtenemos los datos sobre el modo entrenamiento
    modo=get(handles.modo,'SelectedObject');
    switch get(modo,'Tag')
       case 'Q_Learning',   M=1;
       case 'Q_Sarsa',   M=2;
    end
```

Figura 5.7: Código Variables D y M

Para seleccionar la Q en base a estas variables se han realizado varias funciones "if", como se puede ver en la Figura 5.8.

```
%Damos valor a Q en función del modo y la dificultad
    if (D==1) && (M==1)
        Q = Q1_Facil;
    end
    if (D==2) && (M==1)
        Q = Q1_Normal;
    end
    if (D==3) && (M==1)
        Q = Q1_Dificil;
    end
    if (D==1) && (M==2)
        Q = Qs_Facil;
    end
    if (D==2) && (M==2)
        Q = Qs_Normal;
    end
    if (D==3) && (M==2)
        Q = Qs_Dificil;
    end
```

Figura 5.8: Código para la elección de Q

Una vez está todo listo solo falta realizar la conexión con el robot, para ello se llama al fichero conectar y éste nos conectará con la simulación en RobotStudio.

```
  if (c == 0) %Para que solo conecte la primera vez que le damos
Conectar
robot.TCPDireccion(10);
Esperar(4);
c = 1;
  end
```

Figura 5.9: Código para conectar

Como se observa en la Figura 5.10, solo se conecta la primera vez que se llama a esta función, para ello, se ha creado la variable auxiliar "c". Si se volviera a pulsar este botón sería solo para cambiar la matriz Q y no sería necesario repetir la conexión.

```
% --- Al pulsar VolverJugar.
function VolverJugar_Callback(hObject, eventdata, handles)
global TurnosVerde TurnosRojo fila columna Table Tr Tp T3 robot
EnableControl=0;
Control
RecogerRobot;
BorrarTablero;
          %Reseteo las variables

          Table = [0 0 0 0 0 0 0 0 0];  %Table indica las posiciones del juego
          Tr = [0 0 0 0 0 0 0 0 0]; %Posición aislada del movimiento del robot
          Tp = [0 0 0 0 0 0 0 0 0]; %Posición aislada del mov de la persona
          T3 = [0 0 0 0 0 0 0 0 0]; %Variable para cálculo
          fila=0;
          columna=0;
          TurnosRojo=14;
          TurnosVerde=0;
set(handles.conectar,'Enable','on');
set(handles.modo,'Visible','on');
set(handles.Dificultad,'Visible','on');
```

Figura 5.10: función VolverJugar_Callback

En último lugar, está la función correspondiente al botón "Volver a Jugar". Este pulsador, se podrá pulsar en cualquier momento de la partida mientras sea el turno del usuario. Primero, el robot recogerá las fichas y borrará el tablero debido a las funciones "RecogerRobot" y "BorrarTablero". Para reiniciar el juego será necesario reiniciar las variables globales descritas al inicio del fichero. Por ello, se reiniciarán en esta función, como se puede ver en la Figura 5.10que describe esta función. Se activará de nuevo la configuración inicial habilitando los botones correspondientes.

5.5 SOCKET DE COMUNICACIÓN

Para este proyecto se ha utilizado la adaptación al socket de comunicación diseñado por Marek Jerzy Frydrysiak, en su proyecto "Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station" [1]. Este socket fue modificado para poder realizar la comunicación con Matlab, en el proyecto "Desarrollo de una interfaz para el control del robot IRB120 desde Matlab", desarrollado por Azahara Gutiérrez Corbacho [13]. Este socket de comunicación fue implementado en el proyecto de David Paz Hernández, "Implementación del juego 3 en raya mediante el Robot IRB120" [2], el cual lo usa para una aplicación similar a la de este proyecto. En este caso, se usa el socket directamente de este último proyecto. Por tanto, si se desea profundizar más en este apartado, se recomienda consultar los proyectos citados anteriormente.

Por tanto, solo se explica la parte dedicada a Matlab, que permite realizar la conexión con el robot, ya que, la parte de RobotStudio ya fue explicada en su capítulo correspondiente.

Los datos que recibirá el robot serán enviados mediante una clase creada en Matlab.

La clase creada en Matlab corresponde con el fichero llamado "irb120.m", el cual se divide en varias partes como se observa en la Figura 5.11.

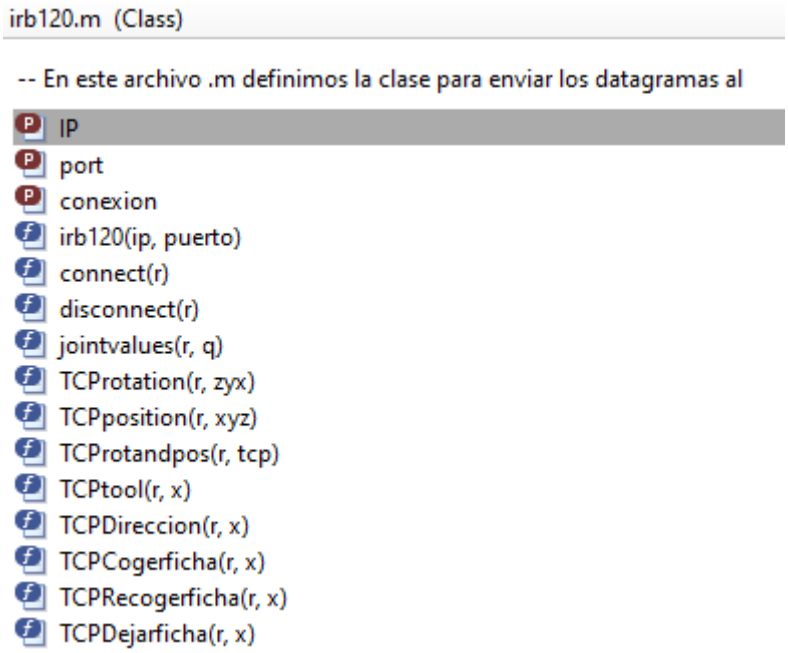

Figura 5.11: Class irb120.m

Los apartados IP, port y conexión, proporcionan a la clase la dirección y puerto de comunicación. Conexión nos permite mantener la comunicación con el robot.

De estos métodos solo se utilizan: "connect" para establecer la conexión; "TCPDireccion", que desplaza el robot a una de las 9 posiciones del tablero, y requiere por entrada un número entre el 1 y el 9; "TCPCogerficha" que coge la ficha del montón correspondiente, verde o rojo, y de la posición correspondiente dentro del montón; "TCPRecogerficha", el cual fue creado para

recoger las fichas del tablero, lleva las fichas del tablero al montón correspondiente y; "TCPDejarficha", al igual que la anterior, deja las fichas en el montón correspondiente.

La estructura de los métodos es la siguiente: en primer lugar, se envía al robot el dato de entrada que reciben, ya sea variables articulares, modificadores de herramientas, IP o acción a realizar. Para ello se crea una variable que incluye el método de la clase RAPID que realizará la acción y dentro del método el sub-apartado que la ejecutará. Este dato es enviado al robot, que lo procesará y realizará las acciones pertinentes.

Como se ha mencionado antes, la conexión se produce al ejecutar el fichero "Conectar.m"; éste se encarga de crear la variable robot, la cual recoge la IP con la que se quiere conectar, en este caso la de simulación de RobotStudio únicamente. Finalmente, se ejecuta la sentencia "robot.connect", la cual realiza la conexión a la IP seleccionada.

CAPÍTULO 6:

CONCLUSIONES Y TRABAJOS FUTUROS

6.1 CONCLUSIONES

Para este proyecto se han utilizado varias herramientas, entre ellas, los sistemas de inteligencia artificial por refuerzo, el socket de comunicación, la lógica de juego, el diseño y programación del robot.

Para la realización de este proyecto se han tratado diversos temas relacionados con la ingeniería, lo que hace de él, un proyecto muy completo.

Exponiendo los puntos más importantes que se han realizado para este proyecto, cabe destacar:

- Diseño mediante RobotStudio de una estación virtual para trabajar en modo simulación.
- Programación mediante RAPID de la estación y definir las trayectorias, funciones, posiciones y lógica.
- Implementación del Socket de comunicación, para poder comunicarse con Matlab en RobotStudio.
- Programación de la lógica de juego.
- Realización de un programa estructurado, con varias funciones y ficheros definidos. Se buscó realizar el programa lo más intuitivo posible, dividiendo el código en partes concretas, facilitando la detección de errores y comprensión del programa en su conjunto.
- El apartado de Inteligencia Artificial en Matlab está dividido en cada función necesaria que posteriormente se implementó en el fichero del tipo de entrenamiento (Q_Learning y Q_Sarsa), para su mayor organización y mejor comprensión.
- Diseño de la interfaz gráfica para realizar un programa fácil de utilizar para el usuario.
- Implementación de protocolos de comunicación para realizar la comunicación con el robot.

A continuación, se comentan los problemas y las conclusiones que han surgido durante la realización del proyecto.

- En cuanto a los métodos de entrenamiento, se observó que para lograr un nivel avanzado en la IA se debían realizar un gran número de entrenamientos. Esto suponía una larga espera para la ejecución del programa. En un principio, se realizó la ejecución de los dos modos de entrenamientos, con las diversas dificultades al iniciar el programa. Esto se realizó para que el usuario pudiese ver el resultado de los entrenamientos (victorias, derrotas y empates) en función del número de dichos entrenamientos. Al caer en la cuenta de que era una espera demasiado larga, se buscó una solución para, en vez de generar las matrices Q que se generan al realizar los entrenamientos, solo tener que cargarlas al iniciar el programa. Para ello, se creó un fichero que generase las matrices necesarias, y se guardaron para así, solo tener que cargarlas a la hora de iniciar el programa y así solventar el problema de la espera.

- En un principio se controlaba el juego desde la ventana de comandos de Matlab, marcando numéricamente la casilla donde se quería colocar la ficha y viéndose cada movimiento en la variable "Table" que representa el tablero. Esta forma de jugar al juego no era nada intuitiva y dependía de que el usuario supiese controlar la ventana de comandos de Matlab, por tanto, se diseñó una interfaz gráfica con la herramienta "guide" de Matlab para así lograr un manejo fácil para el usuario.

- A la hora de crear la interfaz, surgieron algunos problemas al definir las funciones tipo "handle", ya que si se ejecutaba una función de este tipo que no fuese llamada desde el fichero principal de la interfaz, nos daba un error, que no se lograba solventar. Por tanto, se cambió la estructura del programa y se adecuó a esta restricción a la hora de utilizar este tipo de funciones.

- Una vez terminada la interfaz, nos topamos con que era necesario desactivar de alguna forma los botones durante ciertas partes de la partida, para no dar órdenes mientras se ejecutan los movimientos del robot. Para solucionar este problema se creó un fichero que mediante una variable desactivaba o activaba ciertos botones de la interfaz gráfica. También se implementó una barra de tiempo para marcar las esperas y así, ver los tiempos de carga del robot.

- En cuanto al socket de comunicación, solo se implementó en Matlab y RobotStudio, basándonse en el socket aplicado en el proyecto "Implementación del juego 3 en raya mediante el Robot IRB120", de David Paz Hernandez. Por tanto, no hubo ningún problema a la hora de realizar las conexiones entre Softwares.

6.2 TRABAJOS FUTUROS

Posibles mejoras o trabajos futuros sobre nuestro proyecto podrían ser:

En primer lugar, como continuación de este proyecto, se podría realizar la implementación del programa en el robot real y no solo en el simulado. Por otra parte, en cuanto a la inteligencia artificial, se podrían implementar diversos modos para su comparación como el "Deep Learning". Además, se podrían poner en funcionamiento diversos modos de juego como, por ejemplo, un juego con dos jugadores, o dar la opción al robot a ser el primero en realizar su movimiento. También existe la posibilidad de mejorar el socket de comunicación, para conocer con mayor exactitud la posición del robot. Otra opción es implementar un sistema de visión artificial para poder jugar físicamente contra el robot, de manera que, detecte las fichas movidas por el usuario y la posición del tablero.

En cuanto a las técnicas desarrolladas sobre la inteligencia artificial, se podrían implementar para diversas aplicaciones como juegos más complejos, por ejemplo, ajedrez o cuatro en raya.

CAPÍTULO 7:

MANUAL DE USUARIO

Y EJEMPLO DE FUNCIONAMIENTO

En este capítulo, se explica al usuario cómo manejar la aplicación y cómo jugar con las distintas configuraciones diseñadas. Para ello, se realiza un ejemplo del funcionamiento del proyecto, explicado paso a paso. Detallado con capturas de cada paso realizado en dicho proyecto.

7.1 MANUAL DE USUARIO

Desde Matlab se abre el archivo "Instrucciones.md", en el cual, se explican las instrucciones de usuario del programa.

Las instrucciones de la aplicación son las siguientes:

A. Ejecución del programa:

1) RobotStudio

1.1) Abrir el archivo ROBOT_SIMULADO

1.2) Activar simulación

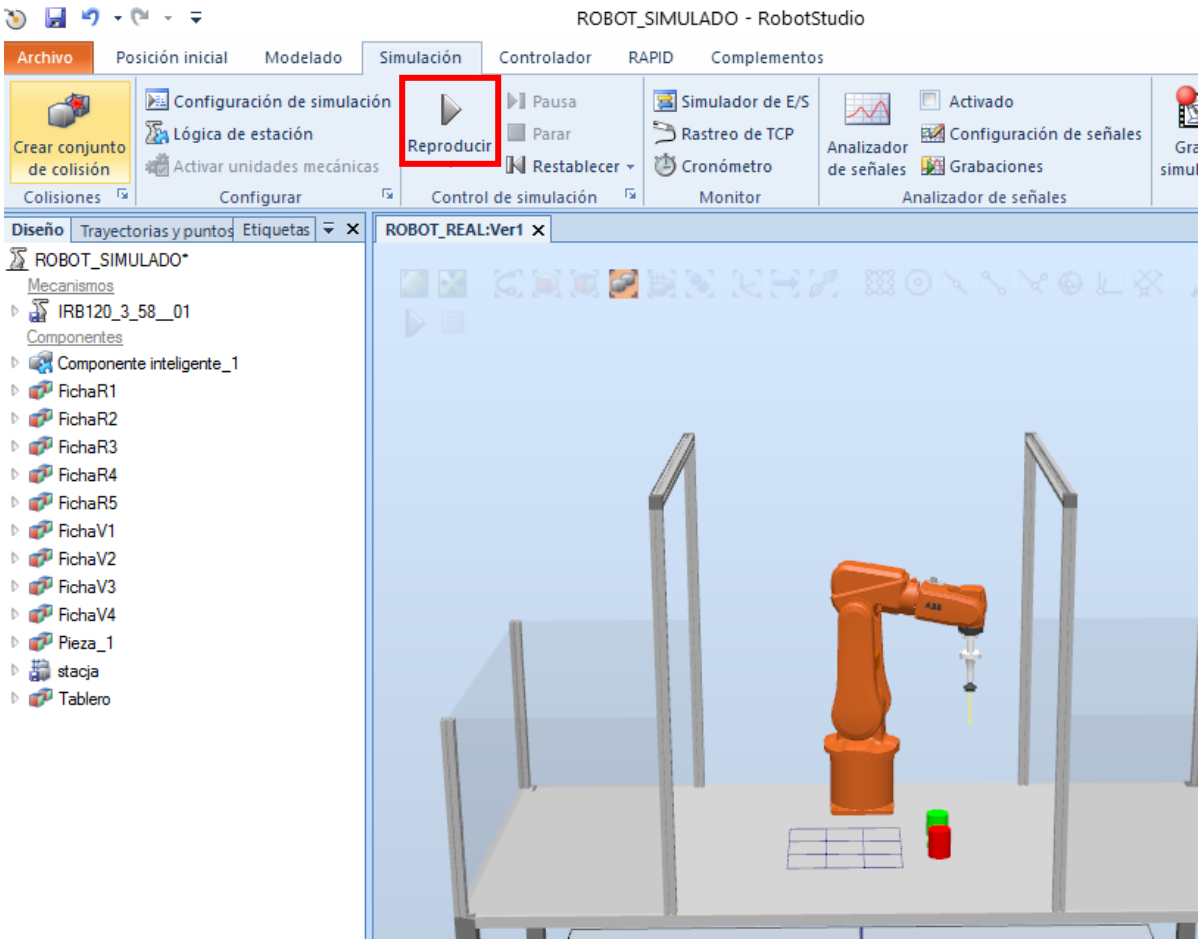

Figura 7.1: Activar Simulación

2) Matlab

2.1) Ejecutar el archivo Robot3EnRaya.m

2.2) Ir a la ventana emergente de interfaz gráfica

Figura 7.2: Inicio aplicación Matlab

B. Comenzar una partida (Interfaz gráfica)

1) Selecionar la configuracion inicial

1.1) Modo de juego --> Q_Learning o Q_Sarsa

1.2) Dificultad --> Facil, Medio o Difícil

3) Pulsar conectar

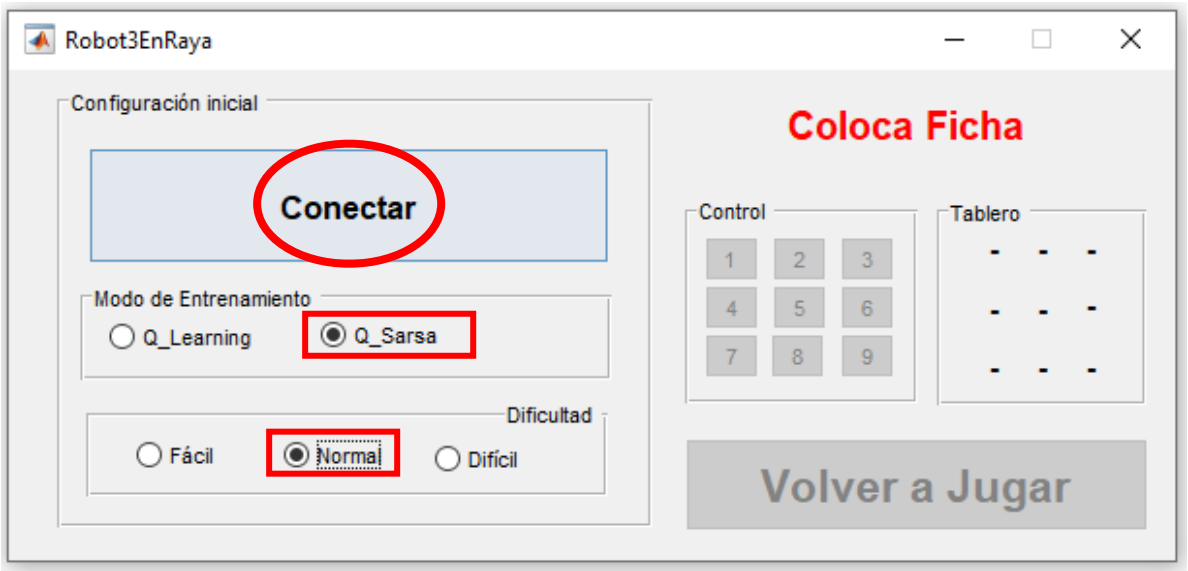

Figura 7.3: Comenzar una partida

C. Transcurso partida

1) Comienza el usuario, colocar ficha con los botones del 1 al 9 del tablero.

Figura 7.4: Colocar fichas GUI

3) Repetir hasta finalizar partida.

D. Fin de partida o Reinicio (Interfaz gráfica)

1) Pulsamos jugar de nuevo

2) Repetimos el punto B.

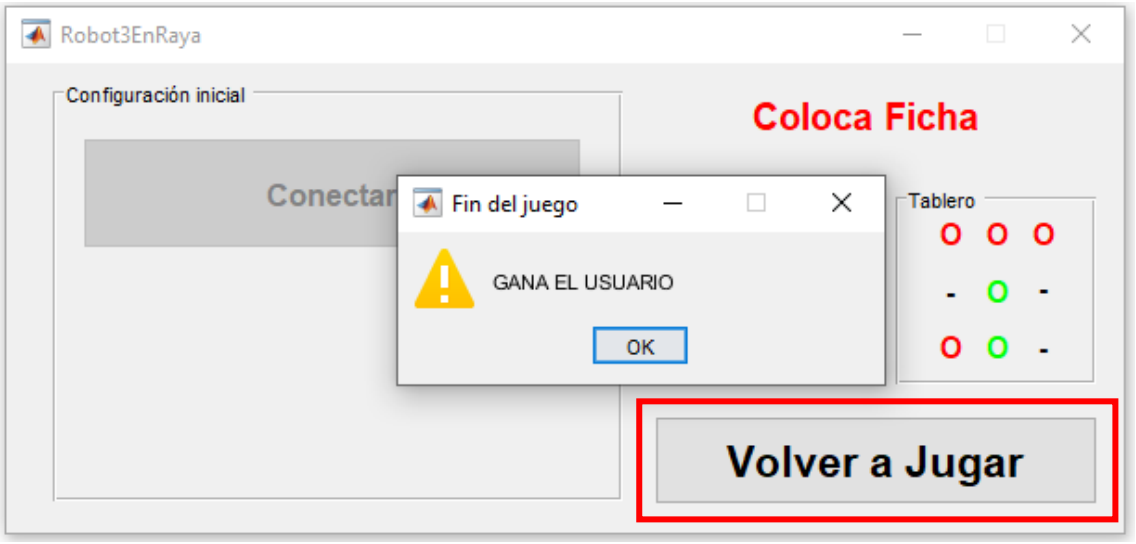

Figura 7.5: Fin del juego GUI

E. Cerrar aplicación

1) RobotStudio

1.1) Parar y Restablecer la simulación

1.2) Cerrar el programa

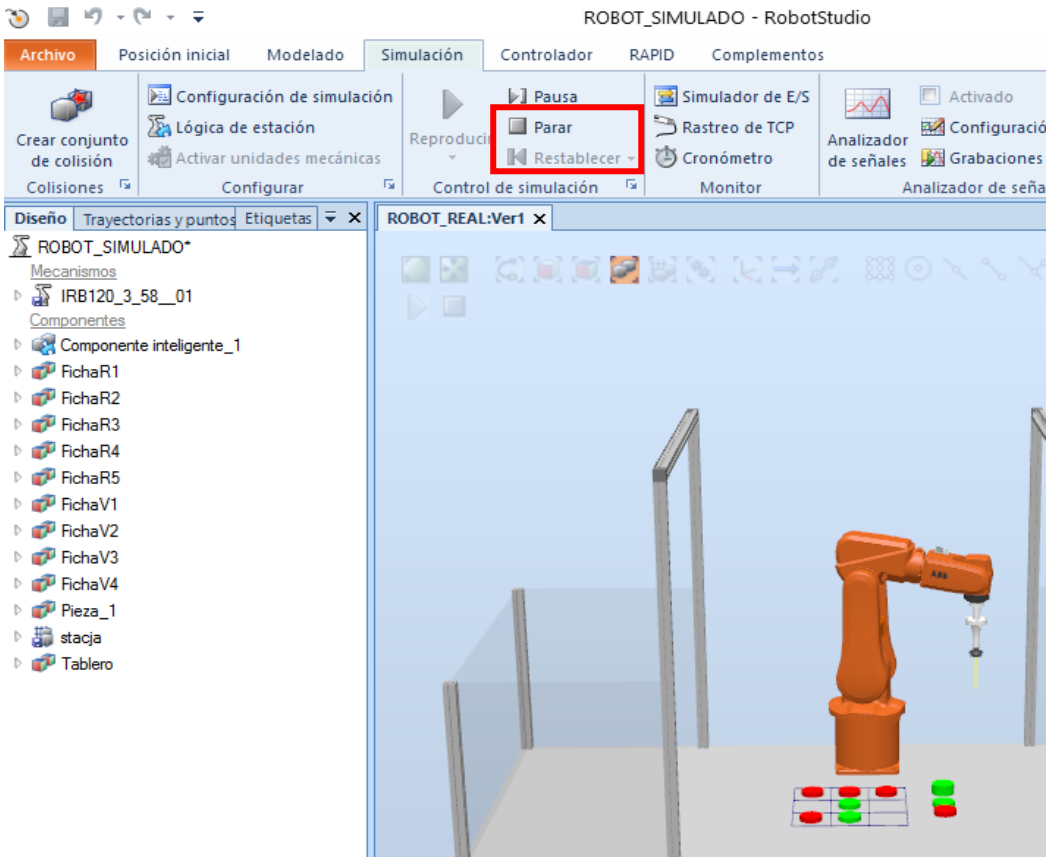

Figura 7.6: Terminar simulación

2) Matlab

2.1) Cerrar la ventana de interfaz gráfica

2.2) Cerrar el programa

7.2 EJEMPLO DE FUNCIONAMIENTO

Siguiendo con el manual de instrucciones, lo primero que se debe hacer es abrir el programa en RobotStudio y activar la simulación. Cuando se activa la simulación, el robot queda a la espera de órdenes que recibirá de la aplicación de Matlab. El robot permanecerá inmóvil hasta realizar la conexión.

Se inicia la aplicación desde Matlab, ejecutando el fichero "Robot3EnRaya.m". Como se puede ver en la Figura 7.7, aparecerá una ventana emergente con la interfaz gráfica.

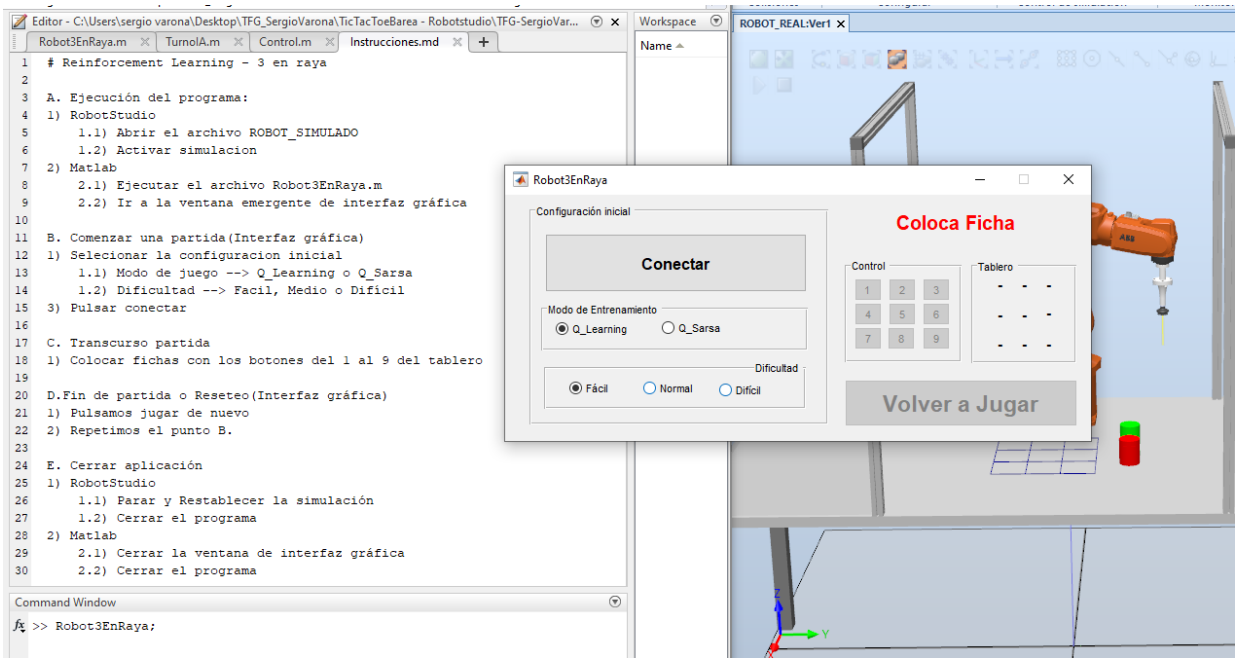

Figura 7.7: Inicio aplicación

Ahora se debe seleccionar la configuración inicial que se va a usar en nuestra partida, en nuestro ejemplo seleccionaremos "Q_Learning" y "Normal". Posteriormente se presiona el botón conectar y comenzará la transmisión de datos.

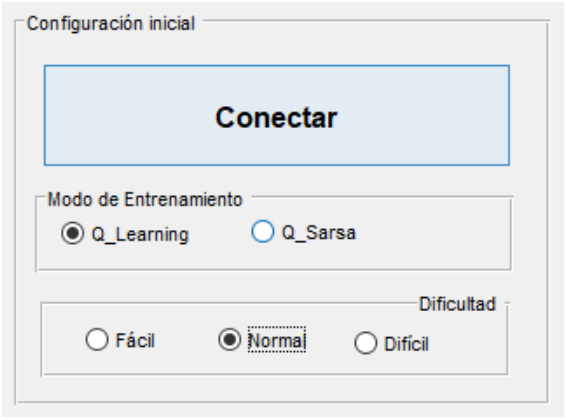

Figura 7.8: Configuración inicial ejemplo

Al realizarse la conexión el robot se mueve a la posición marcada de reposo, donde todas sus articulaciones están a 0º.

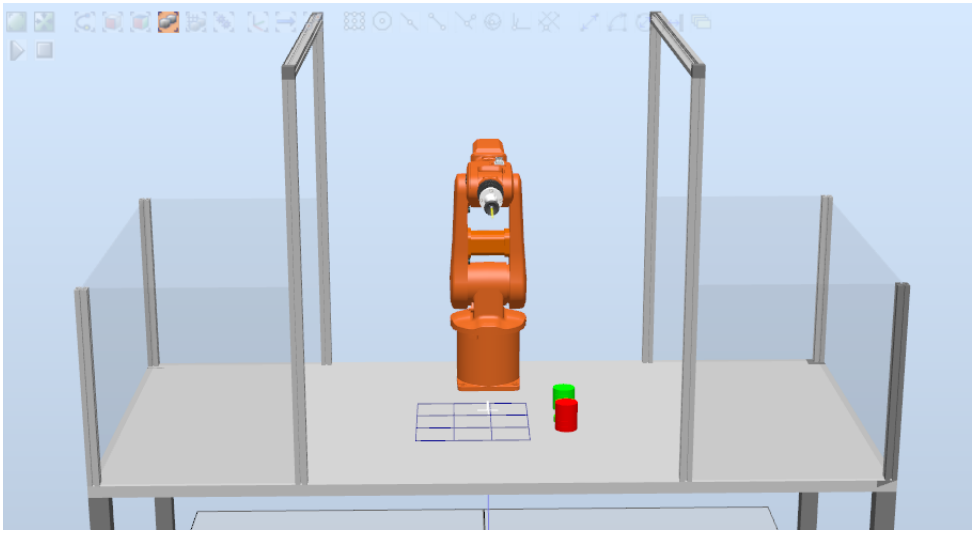

Figura 7.9: Posición de reposo

De la posición de reposo, pasa a la posición para recoger las fichas de los montones, una posición que se repetirá siempre que el robot este esperando a que el usuario realice su turno, donde se tiene visibilidad completa del tablero.

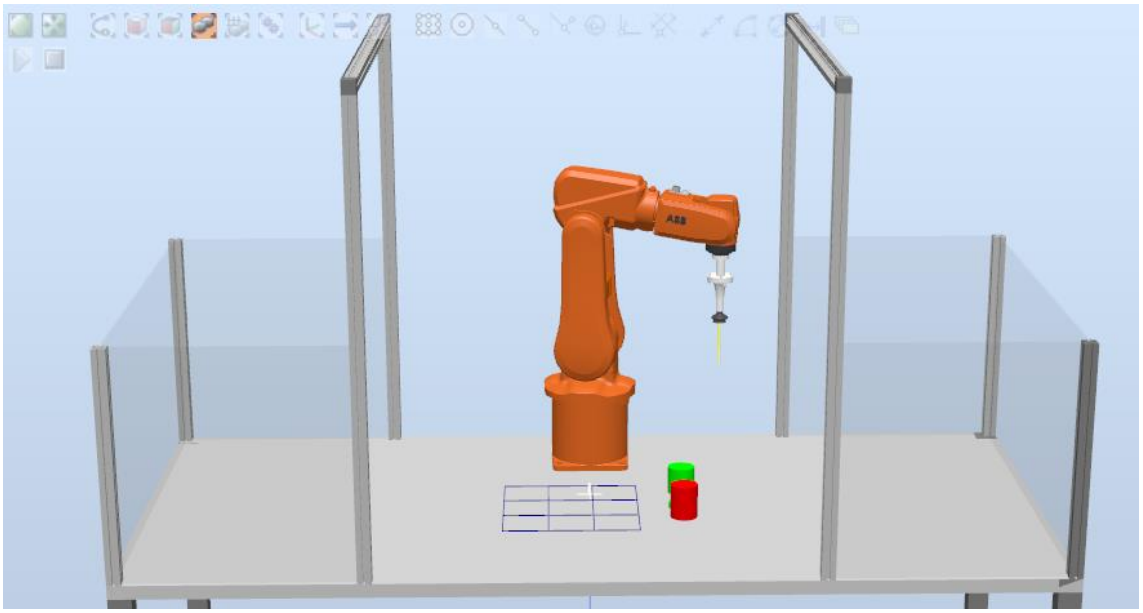

Figura 7.10: posición recoger fichas

Una vez completada la conexión, en la interfaz ya solo se podrá interactuar con el tablero o con el botón volver a jugar si se desea cambiar la configuración inicial del juego o reiniciar la partida.

Figura 7.11: Conexión realizada GUI

Ahora se pasa a seleccionar nuestro primer movimiento; en este ejemplo, se selecciona la casilla 1, pulsando el botón correspondiente. Se debe tener en cuenta que, durante el turno del robot, estarán deshabilitados los controles para evitar fallos, hasta que el robot termine su turno. También aparecerá una ventana emergente con una barra de carga para marca el tiempo de espera.

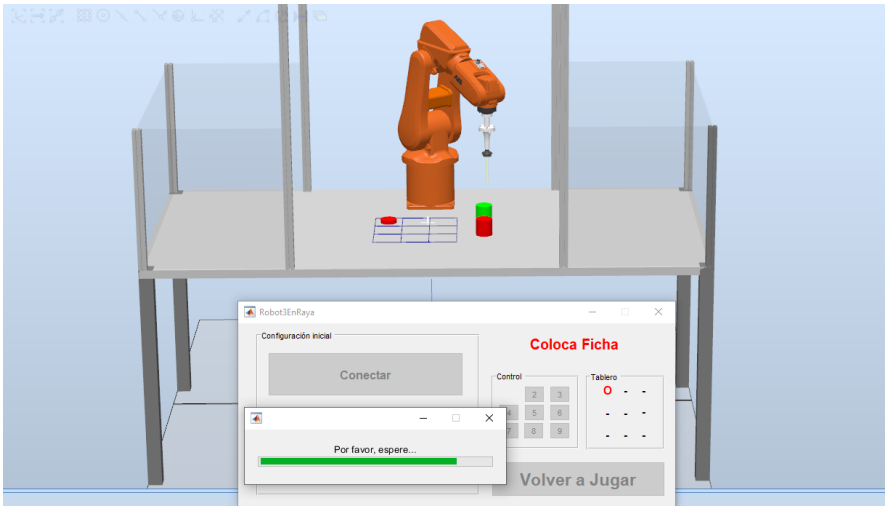

Figura 7.12: Robot en ejecución

Como se ve en la Figura 7.12, en el apartado tablero de la GUI, se van actualizando las casillas marcadas al igual que está en el tablero del robot. También se observa cómo van desapareciendo los pulsadores donde ya hay una ficha colocada para no poder repetir movimiento ni colocar en una casilla ocupada.

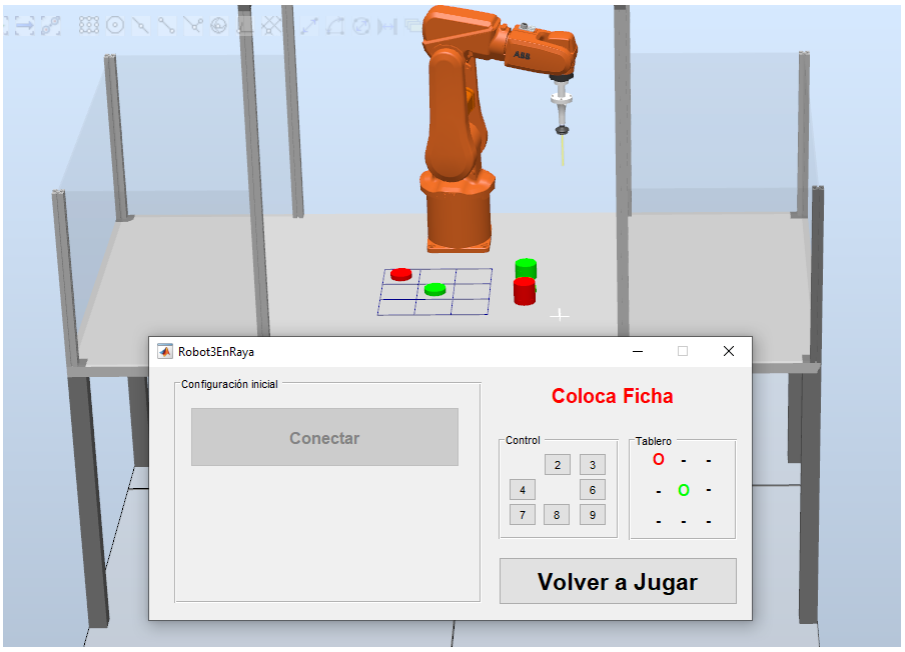

Figura 7.13: Primer turno ejemplo

Después, se continuará la partida hasta llegar a un estado final, en nuestro caso, la partida terminó con victoria para el usuario. Al realizar el robot el movimiento de la ficha que termina el juego nos saltará el mensaje con el resultado de la partida, como se observa en la Figura 7.14.

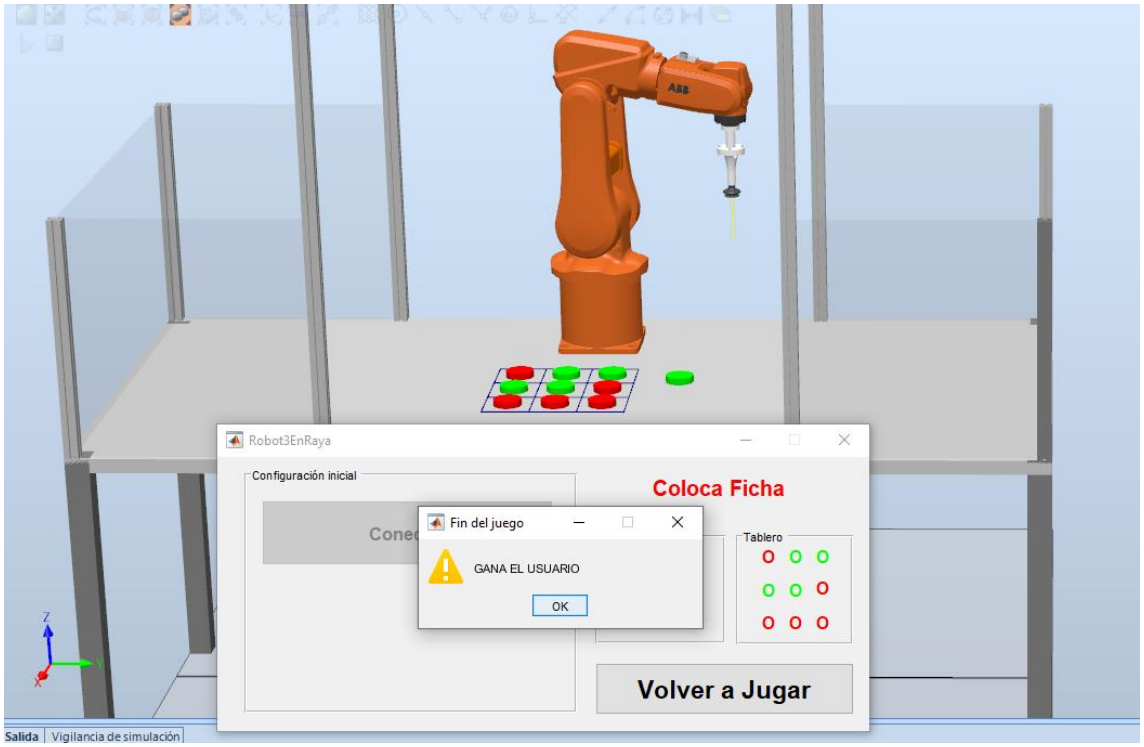

Figura 7.14: Final del juego ejemplo

Durante el transcurso del juego, en la ventana de comandos de Matlab se han ido marcando los movimientos realizados, por tanto, si se quiere revisar la partida se puede ver el movimiento realizado por el usuario y el robot en cada turno, siento el usuario el número 2 y el robot el 1. El tablero corresponde con las 9 posiciones de la tabla.

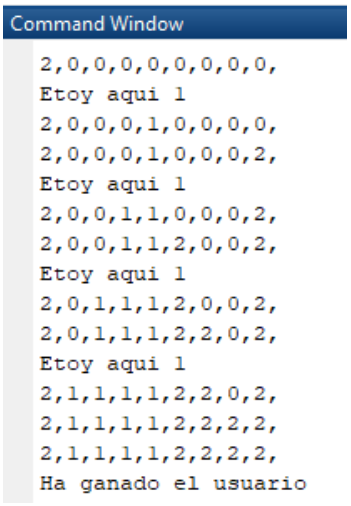

Figura 7.15: Ventana comando

Finalmente, si se desea realizar un juego nuevo, se marca Volver a Jugar. En primer lugar, el robot recogerá las fichas del tablero, colocándolas en su posición inicial y reiniciará el tablero de la interfaz, haciendo visible de nuevo los nueve botones del apartado control. Por último, se solicitará al usuario cambiar la configuración inicial, por si se quiere cambiar el método de entrenamiento o la dificultad del juego. Si no es el caso, se pulsa conectar sin cambiar ningún parámetro; en caso contrario, se cambian los parámetros y se pulsa conectar, esta vez ya no es necesario que se realice la conexión entre softwares, por tanto, solo cambiará la configuración inicial, en el caso de haberlo seleccionado.

CAPÍTULO 8:

PRESUPUESTO

8.1 COSTES MATERIALES

En este apartado se describe el coste de los materiales utilizados para la realización del proyecto. En el tema software, se debe obtener las licencias correspondientes a los programas utilizados, en este caso el precio marcado de cada uno de ellos es el de su licencia de al menos 3 meses.

Objeto		Cantidad	Precio unidad €	Precio total €
Hardware	Robot ABB-IRB120	1	11000€	11000€
	Ordenador	1	700€	700€
Software	Windows 10	1	100€	100€
	RobotStudio	1	273,93€	273,93€
	Matlab R2019a	1	120€	120€
	Microsoft Office 2020	1	149€	149€
Total				12.342,93€

8.2 COSTES TOTALES

Para hallar los costes totales, se suman los costes materiales calculados anteriormente y los costes profesionales. Se tiene en cuenta el IVA, a la hora de sacar el coste total del proyecto.

Para realizar los costes profesionales calculamos el tiempo invertido y el precio por hora para un ingeniero. Esta información se ha obtenido del BOE actualizado el día 15 de febrero de 2021.

Trabajador	Horas trabajadas	Salario por Horas	Coste total
Ingeniero	540h	11.68€	6.307,20€
Total			6.307,20€

A continuación, se calculan los costes totales del proyecto:

Objeto	Coste
Costes materiales	12.342,93€
Costes profesionales	6.307,20€
Subtotal	18.650,13€
IVA (21%)	3.916,53€
TOTAL	22.566,66€

BIBLIOGRAFIA

[1] M. Jerzy Frydrysiak, "Socket based communication in Robotstudio for controlling ABB-IRB120 robot. Desing and development of a palletizing station".

[2] D. Paz Hernández, "Implementación del juego 3 en raya mediante el Robot IRB120", 2016.

[3] Disciplinas de la robótica, consultado en: https://sites.google.com/site/robotsovedientes/home/disciplinas-1.

[4] "Robot industriales. Qué es un robot industrial, tipos y ejemplos de robots", en Revista de Robots, 29 Julio 2021.

[5] Real Academia Española, consultado en: https://dle.rae.es/inteligencia.

[6] Diagramas, consultado en: https://www.diagrams.net/.

[7] "Página de la aplicación MathWorks", consultado en: https://es.mathworks.com/.

[8] "Conceptos de inteligencia artificial: qué es el aprendizaje por refuerzo", consultado en: https://www.xataka.com/inteligencia-artificial/conceptos-inteligencia-artificial-que-aprendizaje-refuerzo.

[9] P. San José Barrios. "Comparación de técnicas de aprendizaje por refuerzo jugando a un videojuego de tenis", 2019.

[10] "La diferencia entre Q_Learning y el algoritmo Sarsa", en Programador clic, 2021.

[11] "Página del Manual de RobotStudio", consultado en: https://new.abb.com/products/robotics/es/robotstudio.

[12] "Creación de apps con interfaces gráficas de usuario en MATLAB", consultado en: https://es.mathworks.com/discovery/matlab-gui.html

[13] A. Gutiérrez Corbacho, "Desarrollo de una interfaz para el control del robot IRB120 desde Matlab", 2014.

Apéndice A:

PLIEGO DE CONDICIONES

En este apartado se describen las herramientas tanto físicas como informáticas que han sido necesarias para la realización del proyecto.

- **Software**

  - Windows 10
  - ABB RobotStudio
  - Matlab R2019a
  - Microsoft office 2021
  - Diagrams

- **Hardware**

  Ordenador sobremesa

  - Procesador AMD Ryzen 5
  - Tarjeta RAM 16 GB
  - Tarjeta gráfica Asus GeForce GTX 1650

  Brazo robótico ABB-IRB 120

  - Altura de 58 cm
  - Peso de 25 kg
  - Soporta cargas de hasta 3 kg
  - Controlador IRC5 Compact

Apéndice B:

CÓDIGO IMPLEMENTADO

En este punto se realiza una breve descripción de cada fichero implementado en Matlab, para así entender el código realizado en este proyecto. En el fichero Instrucciones.md, se enumeran los pasos para la ejecución de este programa.

B.1 INTELIGENCIA ARTIFICIAL

- **boltzmannGreedyAction.m***:* Utiliza la distribución de Boltzmann para encontrar la probabilidad de cada acción posible. Describe la distribución de probabilidad para las posibles acciones y el índice de acción para esta distribución.
- **randomAgentMove.m**: Esta función realiza el movimiento del agente aleatorio.
- **epsilonGreedyAction.m**: Utiliza la *epsilon greedy policy* para elegir la acción para el estado dado, esto se realiza para garantizar una explotación y exploración suficientes.
- **findActionsforStates.m:** Busca acciones para los estados de la matriz Table. Es una matriz del orden 3^9 x 9 ya que para cada estado hay como máximo 9 acciones posibles.
- **FindRewardForAgentAction.m**: Busca las recompensas por la acción del agente
- **findRewardForUserAction.m**: Busca las recompensas por la acción del usuario.
- **stateIndexForTable.m**: Esta función encuentra el estado de la tabla de 3 en raya dada.
- **tableForStateIndex.m:** Encuentra una tabla de 3 en raya para un estado en particular.
- **whoseChance.m:** Esta función otorga valores a *randAgentChance,* dependiendo del agente.
- **Q_Sarsa.m:** Genera la matriz Q mediante el método de entrenamiento por refuerzo Sarsa.
- **Q_Learning.m:** Genera la matriz Q mediante el método de entrenamiento por refuerzo Q_Learning.
- **GenerarQ.m:** Genera 6 matrices Q, para cada nivel de dificultad y para cada método de ntrenamiento.
- **Q_matrices.mat:** Guarda las matrices Q generadas en el fichero GenerarQ.m.

- **TurnoIA.m:** Realiza el turno del robot y da el resultado de la partida.

B.2 INTERFAZ GRÁFICA

- **Robot3EnRaya.m:** Código de la interfaz gráfica, con la función de cada elemento.
- **Robot3EnRaya.fig:** Interfaz gráfica creada.
- **Esperar.m:** Crea una interfaz gráfica con barras de carga, para cuando hay una pausa.
- **Control.m:** Permite activar y desactivar los botones de las nueve casillas del tablero y el boton Volver a Jugar.
- **BorrarTablero.m:** Reinicia el tablero de la interfaz borrando las fichas.
- **ActualizarGUI.m:** Actualiza el tablero diseñado en la interfaz cada vez que se coloca una ficha.

B.3 COMUNICACIÓN CON ROBOT

- **RecogerRobot.m:** Manda al brazo robótico que recoja las fichas del tablero en RobotStudio.
- **MovimientoRobotStudio.m:** Manda al brazo robótico mover las fichas a cada posición del tablero en RobotStudio.
- **Irb120.m:** En este archivo definimos la clase para enviar los datagramas al robot desde la misma.
- **FilaColumna.m:** Manda la instrucción de cuál ha sido el movimiento realizado y si lo realizo el robot o el usuario, dando valor a las variables fila y columna.
- **Conectar.m:** Realiza la conexión con el software del brazo robótico.

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR