Ópera aprende a jugar Linea 4 con Aprendizaje por Refuerzo

Juan José García Cárdenas

Universidad de los Andes
Facultad de Ingeniería, Departamento de Ingeniería Eléctrica y Electrónica
Bogotá, Colombia
2019

Ópera aprende a jugar Linea 4 con Aprendizaje por Refuerzo

Juan José García Cárdenas

Universidad de los Andes
Presentado en cumplimiento parcial de los requisitos para el grado de:
Ingeniero electrónico

Asesor:
PhD. Fernando Enrique Lozano Martínez

Coasesores:
Carolina Higuera Arias
Carlos Quintero Peña
Sustentación de proyecto de grado:
Diciembre 17, 2019

Universidad de los Andes
Facultad de Ingeniería, Departamento de Ingeniería Eléctrica y Electrónica
Bogotá, Colombia 2019

Agradeciemientos

En primer lugar, me gustaría agradecer a mis padres María Lilia Cárdenas y José Reinaldo García al igual que a mi madrina Flor María Barón, junto con el resto de mi familia quienes han sido mi eterno soporte a lo largo de toda mi vida. Gracias a ustedes soy la persona que soy el día de hoy.

En segundo lugar, a Fernando Lozano, a Carolina y a Carlos quienes me ha guiado a lo largo de este proyecto.

Por ultimo, pero no menos importante a mi novia Adelaida Zuluaga quien ha llenado mi vida de felicidad a lo largo de esta gran etapa inolvidable y a todos mis amigos de la universidad.

Abstract

Currently, due to the exponential growth of technology and science in terms of robotics and machine learning, we can observe how everyday social robots are increasingly part of our society. That is why the research and investigation along these two topics have become more and more common. However one of the least investigated areas in terms of Machine Learning and robotics is the combination of reinforcement learning with social robotics.
In the current project, is it shown an entretainment application for the social robot Pepper. The main idea of the project is to develop a web application with Q-learning for the classic game Connect 4. The game is displayed in Pepper's tablet and it is used to interact with humans naturally.

Keywords: Q-learning, Reinforcement Learning, Server, Web Application, Game Theory

Tabla de contenido

Lista de figuras

Lista de tablas

1 Introducción

1.1. Motivación

Dentro del área de la robotica, la robotica social ha sido una de las que mayor avances tecnológicos ha tenido actualmente. Con el desarrollo de los robots sociales y comerciales como lo son los creados por compañías como Softbank Robotics o Toyota, se ha logrado que este tipo de tecnología llegue a hoteles, tiendas y diferentes actividades y lugares cotidianas en los países que se encuentran a la vanguardia de este tipo de desarrollos como lo son Japón, gran parte de Asia y algunos países de Europa. Las aplicaciones que se le puede dar a este tipo de robots sociales son incontables y no solamente en el sector domestico u hotelero, sino también en el sector financiero, de entretenimiento o en la industria alimentaria.
Uno de los factores que más ha contribuido a la proliferación de estas tecnologías actualmente son los avances en el área de Machine Learning o aprendizaje de maquinas. A través de todos sus algoritmos y métodos de clasificación y predicción de información han llevado a la robotica a sus extremos, superando sus límites, llevándola a puntos que hace veinte años se creían inalcanzables. Por ejemplo, a través de diferentes métodos de aprendizaje supervisado se puede reconocer diferentes objetos o personas que esta viendo un robot. De esta manera se puede aumentar la naturalidad de la interacción humano-robot. Al mismo tiempo a través de una acción consecutiva se puede hacer que el robot utilizando aprendizaje por refuerzo puede llegar a aprender como usar o transportar objetos de diferente tamaño o forma.
Ahora bien, todos estos avances dentro de la robotica y dentro el área de machine learning, han permitido el desarrollo de muchas aplicaciones en el mundo real. Una de las aplicaciones más conocidas es el área de entretenimiento. Es por esto que el presente proyecto pretende a través de aprendizaje por refuerzo enseñarle a un robot (Pepper) a jugar Linea 4, un juego de mesa clásico y muy familiar. El robot proyectará una interfaz virtual del juego en tablet de manera que el usuario pueda interactuar y divertirse de una manera natural con el robot.

1.2. Objetivos

1.2.1. General

Desarrollar una aplicación que le permita al robot Pepper, con técnicas de aprendizaje por refuerzo, aprender a ganar a cualquier oponente en juegos clásicos de mesa tales como lo son

Linea 4, Ajedrez, Othello, entre otros. De esta manera se pretende desarrollar un aplicativo con este tipo de recursos en el ámbito del entretenimiento.

1.2.2. Específicos

- Desarrollar una aplicación web a través del framework de Python Django, que cuente con una interfaz gráfica a través de la cual el usuario pueda interactuar directamente con el robot. De esta manera se pretende que el usuario este en capacidad de elegir el juego que desea jugar y participar en este.

- Implementar un algoritmo de aprendizaje por refuerzo dentro de la aplicación web de manera que le permita al robot ir mejorando su desempeño a partir del número de veces que este juega con el usuario. De esta manera el robot estará en capacidad de ganarle a cualquier contrincante después de cierto número de iteraciones.

- Realizar una integración de dicha aplicación web con un sistema de respuesta del robot usando los recursos desarrollados por la alianza SinfonIA, a través de ROS. Esto con el objetivo de que Pepper actué como cualquier otro jugador o ser humano que se encuentra jugando estos juegos clásicos.

1.3. Alcance

A lo largo del proyecto se consideraron diferentes alcances y restricciones las cuales definirían hasta que punto se trabajaría a lo largo del proyecto. En la figura se evidencia cuales fueron los puntos a tener en cuenta que restringieron la visión general del proyecto. En primer lugar se evidencia que la aplicación de entretenimiento se limite netamente al juego de linea 4. Esto debido que, a pesar que se tenía planeado el diseño de la interfaz y algoritmos de aprendizaje para más de un juego de mesa, la limitación de tiempo a lo largo del desarrollo del proyecto impidió desarrollar estas funciones para otros juegos de mesa.
Asimismo, para la elaboración de los algoritmos de teoría de juegos, quienes después se convertirán en los oponentes que usará el agente para aprender a jugar el juego en cuestión, solo se consideró los algoritmos más conocidos para juegos adversariales, tales como Minmax, MinMax con Alpha Beta pruning y ExpectMax. Esto debido a que estos algoritmos permiten de una manera sencilla un buen desempeño dentro del juego. Es decir, gracias a estos algoritmos se permite con diferentes niveles de dificultad, tener un oponente fiable que busca maximizar su propia recompensa del juego.
De igual manera, desde un principio se planteó solamente que dentro de los algoritmos por refuerzo implementados, se tiene Q-learning solamente. A pesar de que existen muchos más algoritmos que servirían para construir un agente que aprenda de su experiencia, como

lo es SARSA, Double Q-learning, entre otros, este algoritmo es el más sencillo. De miras al objetivo de este juego, este algoritmo puede servir de la mejor manera bien con una modelación tabular o a través de una red neuronal.
Finalmente, para la interfaz diseñada se propuso realizar una aplicación Web más no una aplicación móvil para ser proyectada en la tablet de Pepper. A pesar de que el sistema operativo con el que cuenta la tablet de Pepper es Android, para disminuir la dificultad del proyecto se propuso una aplicación web con su interfaz gráfica proyectada en el navegador de internet de la misma. Asimismo, esta interfaz web fue desarrollada en el framework de Python Django, teniendo la posibilidad de realizarlo en entornos de otros lenguajes como NodeJS en el caso de Java, entre otros.
A partir de estas restricciones propuesta, al finalizar el proyecto se tiene como entregables finales los siguientes insumos:

- Un aplicativo para usar con el robot Pepper, en donde este sea capaz de jugar al menos un juego de mesa clásico implementado de forma digital a través de una aplicación web desarrollada en el framework de Python Django.
- Varios algoritmos de teoría de juegos integrados al aplicativo del juego en Pepper. Estos algoritmos deben entrenar a los agentes o algoritmos de aprendizaje por refuerzo.
- Varios algoritmo de aprendizaje por refuerzo integrado al aplicativo del juego en Pepper. Este será capaz de hacer que Pepper aprenda a ganarle a cualquier contrincante con el cual Pepper se enfrente.

1.4. Estructura del documento

La estructura con la que contará este documento será, en primer lugar una breve descripción de los objetivos y alcances del proyecto. En estos, se especificará cuales son las limitaciones y restricciones planteadas en este y hasta donde se pretende llegar. Con esto, se pretende especificar los entregables del mismo y que se esta entregando como producto final a la fecha en que se escribe este documento.
Seguido de eso, se presenta una breve explicación de cada uno de los conceptos que son necesarios entender en su totalidad para comprender realmente el desarrollo de este proyecto. En principio se explica el objetivo y razón de ser de los robots sociales, mostrando sus principales aplicaciones y algunos ejemplos de estos. Seguido de esto se explica la visión que se tiene de distintos juegos desde su misma teoría, exponiendo los principales principios para su solución. Por último se presenta una explicación de que es aprendizaje por refuerzo y cuál es la manera apropiada para abordar diferentes problemas desde esta perspectiva.
En el tercer capítulo, se explica toda la metodología implementada a lo largo del proyecto. En este, se muestra y describe el problema que se pretende resolver. Adicionalmente, a través de un diagrama de flujo se explican cual fue la metodología seguida a lo largo del proyecto.

En primer lugar, se evidencia el proceso de construcción de la interfaz que se proyecta en la tablet de Pepper. Además de esto, se expone cual fue la arquitectura diseñada la cuál permite que cualquier usuario pueda interactuar y jugar contra los algoritmos de teoría de juegos o el agente de Pepper previamente entrenado.
Seguido de esto, se presentan los resultados en el capitulo cuatro. En este se analiza la interfaz realizada más en detalle y todas sus componentes. Además de esto, se muestran las diferentes estadísticas obtenidas por parte de todos los algoritmos, tanto de aprendizaje como de evaluación para teoría de juegos y para aprendizaje por refuerzo. Dentro de estas se encuentra la tasa victorias evaluada para diferentes oponentes de cada uno de los posibles jugadores. Finalmente en el quinto capitulo se encuentran las conclusiones del proyecto. En estas se evaluó el trabajado realizado a lo largo de proyecto y se proponen diferentes alcances y trabajos que se pueden llegar a realizar en un trabajo futuro del mismo.

2 Marco conceptual

A continuación se presentan los principales conceptos que se requerían para entender de la mejor forma el desarrollo del presente proyecto:

2.1. Robots sociales

La robotica social es un área de la robotica dedicada al uso de los robots en un ambiente cotidiano y diario para el beneficio de las personas. Un robot social es aquel que interactúa y se comunica con las personas (de forma sencilla y agradable) siguiendo comportamientos, patrones y normas sociales. Para eso –además de tener apariencia humanoide o semi-humanoide– se necesita que disponga de habilidades que se ubican dentro del dominio de la llamada inteligencia social. La mayoría de los robots actuales funcionan relativamente bien en entornos controlados, como –por ejemplo– los laboratorios, pero surgen problemas cuando operan en entornos más reales, tales como oficinas, casas o fábricas. No obstante, en poco tiempo el robot hogareño se acoplará a la vida del Hombre como alguna vez lo hizo el teléfono, la televisión, el automóvil y la computadora personal. [5]
Entre algunos de los robots que hacen parte de esta línea de robots sociales y que realizan actividades cotidianas se encuentran aquellos construidos por SoftBank Robotics. En principio esta Nao, un pequeño robot creado con propósitos educativos y de investigación que permite cierta interacción limitada con las personas en más de 20 idiomas. Este fue el primer robot creado por esta empresa y cuenta con diferentes sensores, micrófonos y parlantes direccionales para tener una mejor interacción con los humanos. Este robot es usado por compañías como asistente para informar y entretener a todos sus visitantes [6]. Ahora bien, para objeto de este proyecto se usará el robot Pepper, el segundo robot creado por SoftBank Robotics en 2014. Este robot se presenta en la figura **2-1** y sus especificaciones técnicas de manera posterior a esta.
Pepper está diseñado para la interacción con humanos. Cuenta con una serie de sensores y actuadores que le permiten actuar de una manera más natural. Cabe resaltar que toda la información de los sensores y actuadores de Pepper se obtiene con base en la documentación provista por SoftBank. [7]
Adicionalmente, contiene dos tipos de cámara que facilitan el reconocimiento del entorno en el que se encuentra el robot e incluso reconocer objetos y personas que se encuentren cerca al mismo.

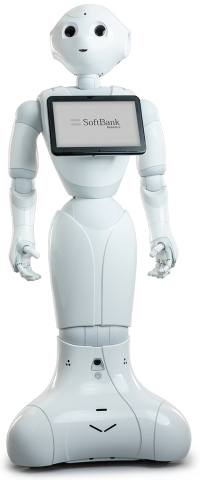

Figura 2-1: Imagen de Pepper

Actuadores

De manera general, Pepper tiene 5 tipos diferentes de motores cuya función corresponde al movimiento de las articulaciones del Robot. Adicionalmente, cuenta con LED's, micrófonos, parlantes y cámaras 2D y de profundidad. Para el enfoque de los proyectos planteados se requiere trabajar más específicamente con los siguientes actuadores:

- **Cámaras 2D:** Están ubicadas en la frente y boca de Pepper. Tienen una resolución de 5MP. Dependiendo de la resolución indicada pueden llegar a una velocidad de captura de hasta 30 fps. Su ángulo de visión es de 55,20°.
- **Cámara de profundidad:** Está ubicada en el ojo derecho de Pepper. Tiene un ángulo de visión de 58°. Puede alcanzar una velocidad de hasta 20 fps.
- **Motores de desplazamiento:** Cuenta con 3 motores DC ubicados en la base dos en el frente y uno atrás.

2.1.1. Tablet

Pepper cuenta con una tablet modelo LG CNS Tablet para la interacción con humanos. Esta cuenta con una memoria RAM de 1GB y una Flash de 4 GB. El objetivo de este elemento de miras a este proyecto de grado es la posibilidad de interactuar con el usuario para que Pepper juegue con el y vaya aprendiendo cómo mejorar su desempeño en el mismo.

Sensores

Pepper cuenta con acelerómetro, giroscopio, láseres, sensores infra-rojos, sonares y encoders de rotación magnética. Adicionalmente, tiene un sensor táctil en la cabeza y una tablet en

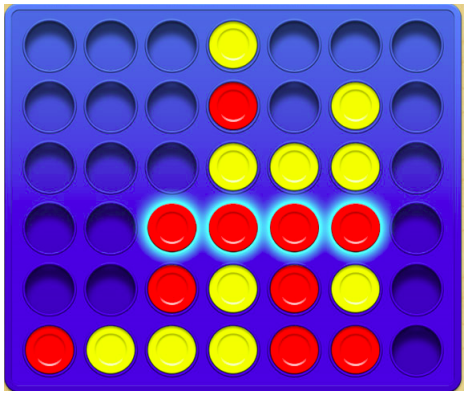

Figura 2-2: Tablero de juego linea 4 [1]

el pecho para facilitar su interacción. Para el enfoque del presente proyecto se trabajará principalmente con los siguientes sensores:

- **Láser:** Cuenta con 3 sensores láser ubicados en la base. Uno de los sensores está orientado hacia el frente del robot. Los dos restantes están orientados hacia los lados. Cada láser tiene una frecuencia de $6.25Hz$ y un ángulo de visión de 60°

- **Sonares:** Cuenta con dos sonares, uno al frente y otro atrás. Su ángulo de visión es de 60° y frecuencia de $42kHz$.

2.2. Juego de Linea 4

Las normas de linea cuatro son relativamente sencillas. Se juega siempre entre 2 jugadores y sobre un tablero de 7x6 casillas. En cada turno cada jugador coloca una ficha de su color en una columna y esta cae hasta la primera casilla disponible. El que consigue ubicar 4 fichas del mismo color seguidas en horizontal, vertical u oblicuo gana. Si nadie lo consigue la partida termina en empate. Un ejemplo del tablero se muestra en la figura **2-2** [1].

2.3. ROS

El sistema operativo del robot (ROS) es un framework flexible para escribir software para robots. Es una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear un comportamiento robusto complejo y robusto en una amplia variedad de plataformas robóticas [8]. A continuación se presentan diferentes conceptos y herramientas con las que cuenta ROS y fueron usadas a lo largo del proyecto:

2.3.1. Nodos:

Un nodo es un proceso que requiere algún tipo de gasto a nivel computacional. Los nodos se combinan unos con otros en un grafo y se comunican unos con otros a través de la transmisión constante de mensajes en tópicos, servicios y/o acciones. Estos nodos están destinados a realizar operaciones muy pequeñas; un sistema de control de robot generalmente comprenderá muchos nodos. Por ejemplo, un nodo controla un telémetro láser, un nodo controla los motores de las ruedas del robot, un nodo realiza la localización, un nodo realiza la planificación de la ruta, un nodo proporciona una vista gráfica del sistema, etc [9].

2.3.2. Tópicos:

Los tópicos se denominan buses sobre los cuales los nodos intercambian mensajes. Los tópicos tienen una semántica anónima de publicación o subscripción que desacopla la producción de información de su consumo. En general, los nodos no son conscientes de con quien se están comunicando. En cambio, los nodos están concentrados en la información que reciben y envían relativos al proceso que están realizando. De manera general, pueden haber muchos subscriptores y publicadores a un mismo tópico [10].

2.4. Teoría de juegos

La teoría de juegos es un área de la matemática aplicada que utiliza modelos para estudiar interacciones en estructuras formalizadas de incentivos (los llamados «juegos»). La teoría de juegos se ha convertido en una herramienta sumamente importante para la teoría económica y ha contribuido a comprender más adecuadamente la conducta humana frente a la toma de decisiones. Sus investigadores estudian las estrategias óptimas así como el comportamiento previsto y observado de individuos en juegos. Tipos de interacción aparentemente distintos pueden en realidad presentar una estructura de incentivo similar y, por lo tanto, se puede representar mil veces conjuntamente un mismo juego. Dentro de estos juegos se pueden encontrar diferentes tipos de ellos, como lo son juegos de suma cero, adversariales, de información perfecta o imperfecta, entre otros. Asimismo, de acuerdo con el tipo de juego que se tenga y los jugadores que contemplen para el mismo, existen diferentes algoritmos que describen el comportamiento óptimo de cualquier jugador en un juego especifico.
De miras al objetivo del proyecto, los algoritmos que se deben tener en cuenta son aquellos diseñados para juegos adversariales de suma cero y de información perfecta como lo es Linea 4. Es decir, juegos en los cuales se deban enfrentar dos jugadores y la ganancia de uno de los jugadores es la perdida del otro. Además el hecho que sea de información perfecta implica que ambos jugadores conocen las jugadas del otro sin ningún tipo de impedimento. Ahora bien, dentro de esos algoritmos encontramos el algoritmo MinMax, una variación del mismo con Alpha - Beta pruning y ExpectMax. [11]

2.4.1. Algoritmo MinMax

Este algoritmo de decisión busca minimizar recompensa del jugador contrario y maximizar la recompensa del mismo. Este algoritmo es un algoritmo recursivo. Este algoritmo se basa en el hecho de que el jugador principal que esta usando esta estrategia buscará el mejor movimiento para él suponiendo que su adversario buscará el peor movimiento para el jugador principal. Ahora bien, de manera general este algoritmo se puede implementar siguiendo los siguientes pasos [11]:

1. Generación del árbol del juego, donde se generen todos los nodos y sus posibles jugadas hasta cierto punto indicado o hasta que se llegue a un nodo terminal.

2. Para cada nodo terminal o aquel nodo en el cuál se le haya indicado finalizar se calcula su utilidad o recompensa para el jugador en turno.

3. Calcular el valor de los nodos superiores buscando el minimo valor de los inferiores, en caso de ser el turno del adversario, y el máximo valor de los nodos inferiores, en caso de ser el turno del jugador con esta estrategia.

4. Desde el nodo actual, elegir la jugada que tenga máximo valor para el jugador con esta estrategia.

En la figura **2-3** se muestra un pequeño ejemplo de la aplicación de este algoritmo. Como se puede ver, todo el árbol del juego esta generado y para los nodos terminales se cálculo su valor de utilidad. Seguido de esto, haciendo una propagación hacia arriba, se elige la jugada con mayor o menor valor según corresponda el turno de cada jugador. Si es el adversario, se elige la jugada con menor valor de utilidad y si es el jugador con dicha estrategia, se elige la jugada con mayor valor. Finalmente desde el nodo superior o el actual, se elige la jugada con mayor valor siendo esta la representada por línea azul en la rama de la derecha.

2.4.2. Alpha - Beta Pruning

El algoritmo de Alpha-Beta pruning o poda alpha-beta, por su nombre en español, toma el algoritmo de MiniMax previamente explicado y reduce el tamaño del árbol del juego, basados en los parámetros α y β. El problema de la búsqueda Minimax es que el número de estados a explorar es exponencial al número de movimientos. Partiendo de este hecho, la técnica de poda alfa-beta trata de eliminar partes grandes del árbol, aplicándolo a un árbol Minimax estándar, de forma que se devuelva el mismo movimiento que devolvería este, gracias a que la poda de dichas ramas no influye en la decisión final [11]. Este algoritmo se basa en el principio de que a medida que se recorre el árbol del juego mediante MinMax, se van guardando en valores α y β, los valores máximos y mínimos de acuerdo con la jugada que se vaya a realizar y se descartarán ramas cuando se tengan los valores máximos o mínimos posibles.

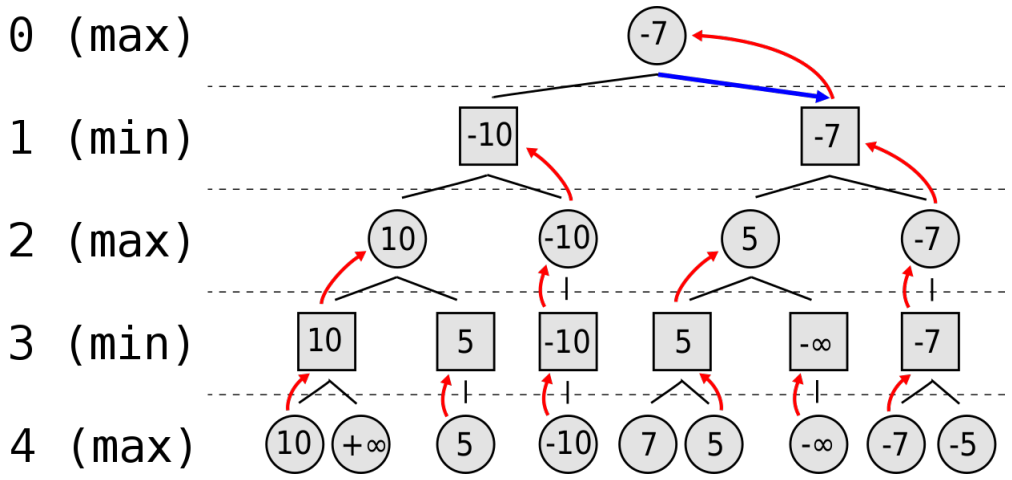

Figura 2-3: Ejemplo de aplicación algoritmo minimax [2]

2.4.3. Algoritmo ExpectMax

El algoritmo de ExpectMax, toma el algoritmo de MinMax y realiza su mismo procedimiento con una diferencia esencial. Este en vez de considerar un adversario óptimo, el cuál buscará la peor jugada para el jugador principal, se considerará un contrincante promedio. Es decir, ahora en vez de minimizar la jugada del contrincante o tomar el valor minimo de recompensa que tiene en esa jugada, se tomará el valor promedio de las ramas subalternas al nodo evaluando. De esta forma, el algoritmo MinMax deja de ser ta exigente con su oponente y es consciente de los posibles errores que pueda llegar a cometer su oponente.

2.5. Aprendizaje por refuerzo

El aprendizaje por refuerzo termina siendo la intersección de muchas áreas entre las cuales se encuentran Machine Learning, Psicología entre otras. Esta se considera como la ciencia de la toma de decisiones. Tal y como sucede para los seres humanos, el aprendizaje por refuerzo es un tipo de aprendizaje de maquinas que se concentra en el aprendizaje a partir de la experiencia. Este no es considerado como aprendizaje supervisado ya que no se cuenta con una serie de datos o de salidas que permitan o indiquen que acción esta bien o mal. En cambio, esta si cuenta con una señal de recompensa que va variando de manera proporcional según sea su estado y acción a realizar. De la misma manera, este no se considera aprendizaje no supervisado puesto que no existe ninguna manera de agrupar la información, estados o acciones de cada agente. El aprendizaje por refuerzo se considera como un nuevo tipo de aprendizaje de maquinas, en el sistema donde actúa es dinámico y la secuencia de tiempo en verdad afecta el proceso. Este tipo de aprendizaje tiene ciertos factores relevantes [12]:

- **Estados:** S describen en que parte del proceso se encuentra el agente. Si se encuentra en un MDP, este describe no solamente el estado actual, sino que también se puede

conocer las acciones que se ha tomado anteriormente y los estados por los cuales a pasado.

- **Acciones:** a, son todas las posibles acciones que el agente puede tomar para pasar de un estado a otro. Estas se toman respecto a la recompensa que tenga cada una de ellas.
- **Recompensa:** r, es la función que define que tan buena o mala es una acción en cierto estado perticular.
- **Función de valor estado:** Determina que tan bueno o malo es el estado S actual:

$$V^{\pi}(S) = E_{\pi}\left\{R_t|S_t = S\right\} = E_{\pi}\left\{\sum_{k=0}^{\text{ínf}} \gamma^k r_{t+k+1}|S_t = S\right\} \tag{2-1}$$

- **Función de valor estado-acción:** Determina que tan bueno o malo es la acción a en un estado S especifico:

$$Q^{\pi}(S) = E_{\pi}\left\{R_t|S_t = S, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\text{ínf}} \gamma^k r_{t+k+1}|S_t = S, a_t = a\right\} \tag{2-2}$$

Ahora bien, el problema que busca solucionar este tipo de aprendizaje se puede definir formalmente como un proceso de decisión de Markov.

2.5.1. Proceso de decisión de Markov

Es un proceso que cuenta con los elementos $< S, A, P, R, \gamma >$, donde P es la matriz de transición de estado y γ es el factor de descuento, el cuál establece que tan importante serán las acciones futuras o siguientes. Lo más importante de este proceso es que cumple con la propiedad de Markov, es decir que a través del s_t se puede conocer la historia y los demás estados visitados. Además de esto, la política de comportamiento π que sigue cualquier agente dentro de un problema de decisión de MDP. Esta esta definida por [12]:

$$\pi(a|S) = P[A_t = a|S_t = s] \tag{2-3}$$

2.5.2. Temporal Difference Learning

Inicialmente este tipo de aprendizaje fue planteado como una combinación entre programación dinámica y los métodos de Monte Carlo [12]. Consiste en la implementación de una

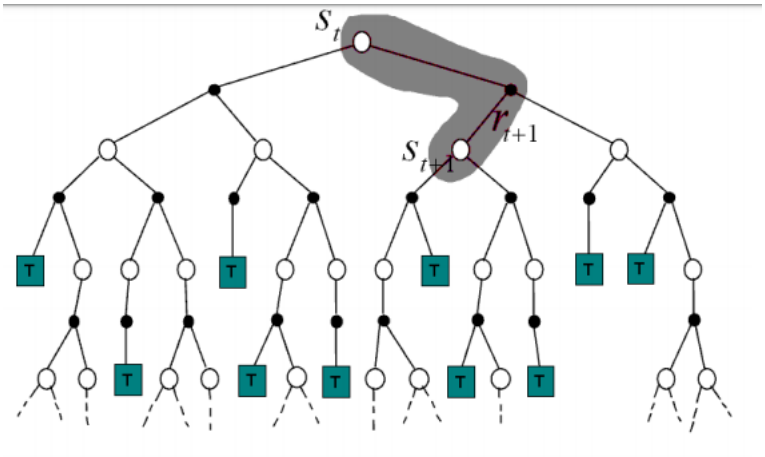

Figura 2-4: Ubicación de los estados necesarios en TD [3]

función de valor que se actualiza constantemente, sin necesidad de esperar por algún estado final, y que no necesita pasar por todos los estados y acciones existentes. Es por esto que no es necesario un modelo del ambiente ya que la función de valor se actualizará solo con los estados visitados [3]. La gran mayoría de métodos de aprendizaje por refuerzo tienen su origen en este tipo de aprendizaje. La función de valor se actualiza según:

$$V(S_t) \leftarrow V(S_t) + \alpha[r_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2-4}$$

En la figura **2-4**, tomada del trabajo de grado de Edwin Torres García [3], se evidencia claramente como la actualización de un estado depende del valor estimado del siguiente estado.

2.5.3. Q-learning

Este algoritmo extiende las ideas de TD aplicadas a la función del par estado-acción $Q(S, a)$. Sin embargo, este no se basa en una política de comportamiento óptima sino a la función de valor tomando la acción que mayor valor de $Q(S, a)$ tiene. Con esto se puede determinar la política de comportamiento óptima:

$$\pi^*(s) = argmax_{a \in A} Q(S, a) \tag{2-5}$$

Asimismo, esta función de valor para el par estado-acción se actualiza de la siguiente manera:

$$Q(S_t, a) \leftarrow Q(S_t, a) + \alpha[r_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, a)] \tag{2-6}$$

3 Propuesta metodológica

3.1. Planteamiento del problema

El problema a solucionar para este proyecto es enseñarle al robot Pepper a jugar Linea 4 en una aplicación digital a través de aprendizaje por refuerzo. Este problema tiene como base tres componentes: en primer lugar se tiene que diseñar el entorno del juego en el cuál tanto el robot como el usuario van a jugar y a través del cuál van a interactuar, el agente de aprendizaje por refuerzo que este en capacidad de jugar contra algún oponente y uno o varios modelos desde una perspectiva diferente de teoría de juegos que sirvan como punto de comparación y como oponentes de entrenamiento para el agente de RL.
En primer lugar es necesario construir y diseñar el entorno de juego en el cuál interactuarán tanto el robot Pepper como el usuario. Para esto, como bien ya se aclaró en el alcance del proyecto solamente se consideró el diseño de una aplicación Web para que sea proyectada en la pantalla táctil de Pepper.
Adicionalmente para la parte del agente de aprendizaje por refuerzo se tiene que, el juego de linea 4 se puede describir como un proceso de decisión de Markov, en el sentido que cuenta con las siguientes características:

- **Estados:** Los estados considerados son todas las configuraciones del tablero en el cuál debe jugar el agente de aprendizaje por refuerzo. Estos siempre tendrán la misma dimensión del tablero completo del juego, es decir una matriz 6x7.

- **Acciones:** Por parte de las acciones disponibles para este MPD, se tendrá que siempre que el juego lo permita, se podrá poner una ficha en cualquiera de las 7 columnas que compone el tablero de juego. Dada la naturaleza el juego, en donde los jugadores solamente pueden elegir en que columna van a jugar, más no la fila y columna exacta, se pueden definir el espacio de acciones como cualquiera de estas siete columnas.

- **Recompensa:** Como señal de recompensa se puede diseñar dos diferentes. Una de ellas es la que se llama recompensa dispersa, en donde solamente al finalizar el juego se le da una recompensa al agente. De esta manera para cada caso correspondiente, la recompensa se describe de la siguiente manera:
  - Si el agente gana: $R = 1$
  - Si el agente pierde: $R = -1$

- Si el agente empata el juego: $R = 0{,}5$

Una segunda señal de recompensa puede ser una que no recompense al final sino a lo largo del juego, después de cada una de las acciones, verificando que tan buena es la jugada que realizó. Ahora bien, esta verificación la realiza respecto a cierto número de jugadas en el futuro. Es decir, dependiendo según sea el caso, el valor de la recompensa para una acción en especifico variará si para la siguiente o siguientes jugadas esta me lleva a apilar fichas de la manera deseada. Por ejemplo, si tengo la posibilidad de formar 3 fichas de manera horizontal gracias a la acción que acabo de tomar esta acción tendrá una recompensa mayor que el resto. En la figura **3-1** se evidencia de una manera más clara este tipo de recompensa por jugada. Allí se muestra el puntaje obtenido por cada uno de los casos que existen fichas consecutivas según la dirección que exista para formar linea 4.

- **Tasa de descuento:** Como tasa de descuento del juego se tomará inicialmente 1, dado que en principio no se tiene conocimiento previo de la potencial y futura influencia de las futuras jugadas sobre la actual.

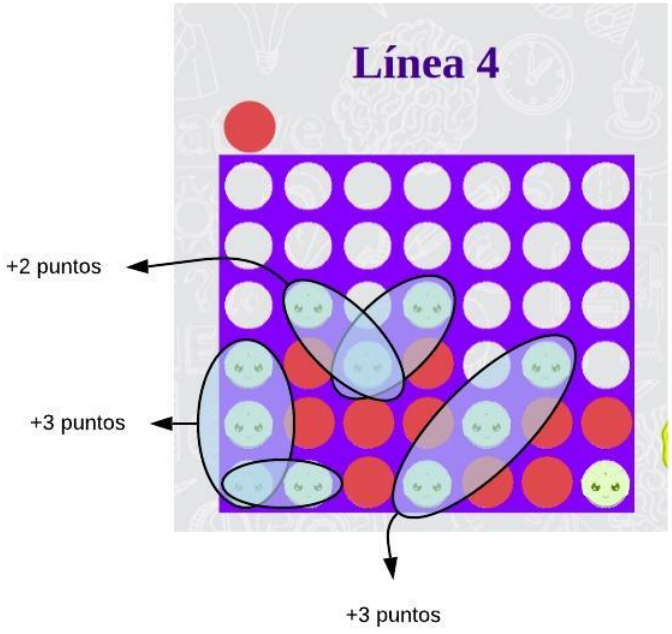

Figura 3-1: Segunda opción de recompensa establecida

Finalmente como parte del problema se desea obtener uno o varios modelos programados a través de teoría de juegos que sirvan como referencia para el agente de aprendizaje por refuerzo. Es decir, estos serán capaces de tener un desempeño igual o mejor que el de aprendizaje por refuerzo y el objetivo de los mismos es llegar a realizar una comparación de desempeño.

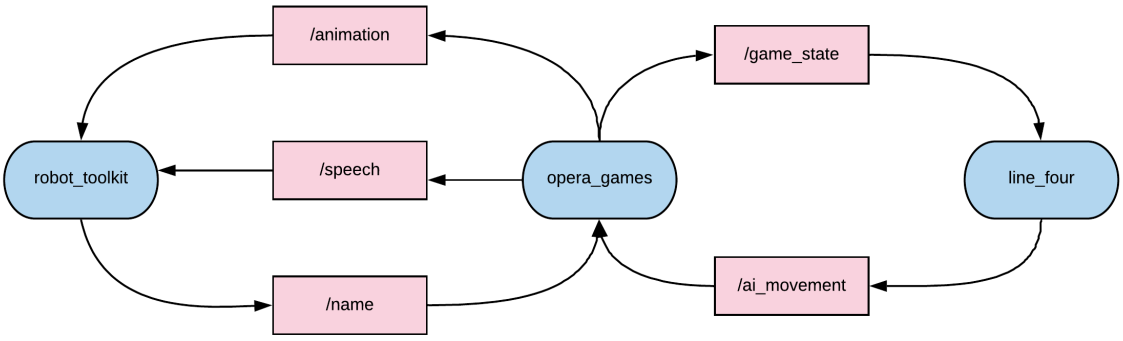

Figura 3-2: Arquitectura principal del problema planteado

De igual forma, estos contrincantes clásicos servirán como oponentes de entrenamiento del agente de aprendizaje por refuerzo. Es decir, la idea se resumirá en que esto agentes competirán contra el agente de aprendizaje por refuerzo y de esta forma lo entrenarán.

3.2. Propuesta metodológica

Dentro de esta sección se especificará la metodología implementada a lo largo del proyecto de miras a la solución del problemas previamente planteado. En la figura **3-2** se establece la arquitectura principal del problema. En este se especifica cada uno de los módulos desarrollados a lo largo del proyecto y como estos funcionan de manera conjunta con el robot. Vale la pena aclarar que de manera adicional allí esta incluido el diseño y la implementación de los algoritmos o agentes de teoría de juegos.

- **Opera Games:** Ahora bien, en primer lugar el nodo principal que conforma el sistema es el nodo de la interfaz. Este nodo o módulo llamada así por la posibilidad de realizar allí más de un juego, es conformado por un servidor Web que alberga toda la interfaz gráfica del juego y la conecta con el sistema operativo ROS. A través de este sistema operativo es por donde se comunicará con los diferentes servicios con los que cuenta el robot. Estos servicios del robot se explicarán en más detalle más adelante. Este servidor web fue hecho a través del framework de python Django-Channels. Este framework permite una comunicación entre las páginas HTML y el servidor sujeto a una arquitectura hecha en python a través de lo que se conoce como Web Sockets. Esta estructura se resume en la figura **3-3**. Allí se evidencia que en el Frontend de la aplicación se tiene una interfaz o página web como cualquier otra, en donde a través de la cuál el usuario interactúa con el Backend de la misma. Para el presente proyecto dicho Backend, construido en el módulo de *Consumers*, corresponde a la comunicación entre el servidor, el robot y el algoritmo de aprendizaje por refuerzo que este jugando.

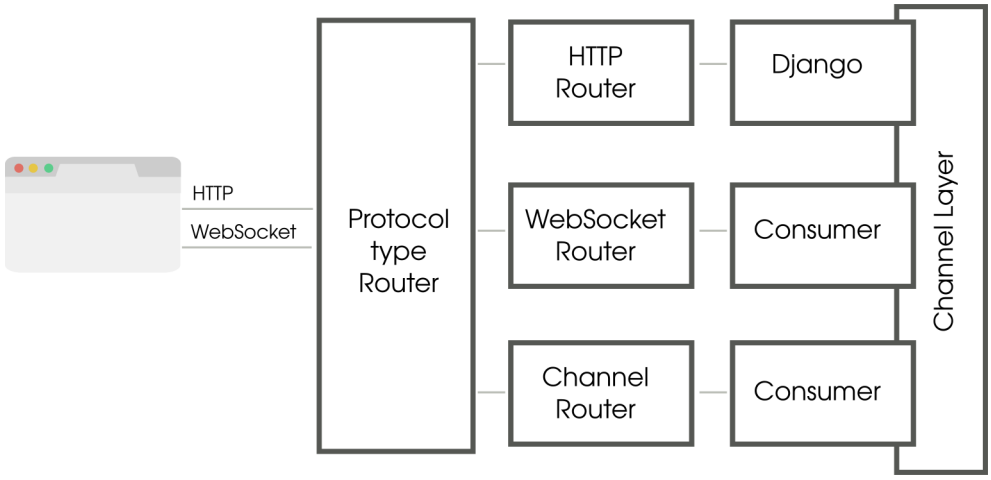

Figura 3-3: Arquitectura de un servidor Django-Channels [4]

Esta comunicación con estos otros dos nodos, se realiza a través de tópicos con diferentes estructuras en sus mensajes. El tópico llamado *game_state* que le envía el servidor al nodo de *line_four*, es una representación del estado del tablero del juego. Es por esto, que este es una matriz 6x7 con valores de 0, 1 o 2; donde 0 son las casillas donde no hay ficha o movida aún, 1 es donde el jugador usuario tiene fichas y 2 donde el agente de aprendizaje por refuerzo tiene fichas. Asimismo, el tópico al cuál esta suscrito *ai_movement* es la respuesta de dicho algoritmo de aprendizaje por refuerzo al estado del juego actual. Este valor es uno entre 0 y 6, lo cuál representa la columna en la cuál se querrá poner la ficha.

Ahora bien, por el lado de la comunicación del robot se tienen otros tópicos diferentes. En primer lugar se tiene el tópico de *speech* en el cuál el servidor le pública una cadena de caracteres al robot para que este lo reproduzca en su parlante. Esta cadena de caracteres representan las expresiones más comunes que diría el robot para ciertos casos del juego. Es decir, según sea la situación el robot reaccionará contra su oponente de forma verbal. Asimismo el tópico de *animations* hace que con este mismo objetivo el robot haga ciertas animaciones únicas según sea el caso.

- **Robot Toolkit:** Este nodo es el único que corre internamente en el robot. Este es una capa de aplicaciones realizada por la alianza SinfonIA durante el presente año. Su función principal es comunicar toda la información de bajo nivel, como lo es la imagen de las cámaras, información de sensores, micrófono y altavoces a una capa superior. De esta forma, con solo uno publicar en un tópico de *speech* ya es suficiente para que el robot hable lo que se le pida y en el idioma de su preferencia. Asimismo, este facilidad de las cosas sucede con otros sensores o actuadores. Este es el caso de las cámaras a

través de las cuales con la solicitud de un servicio se puede configurar la resolución, brillo, contraste y demás variables a considerar al momento de importar la imagen de la cámara del robot [13]. Esta capa de bajo nivel fue realizada con el objetivo de facilitar el desarrollo de aplicaciones en inteligencia artificial, de forma que fuera más sencillo y simple la obtención de datos y manejar el robot desde otra capa de jerarquía superior.

- **Line Four:** Este nodo comprende el agente de aprendizaje por refuerzo que es entrenado para que juegue contra el usuario a través del robot. Para la creación de este agente se utilizó el algoritmo de Q-learning previamente explicado. Este algoritmo fue implementado de dos maneras diferentes. En primer lugar se implementó teniendo en cuenta una representación tabular del juego en la cual es considerada la manera más apropiada de hacerlo ya que cada estado se representa de manera discreta. En segundo lugar se implementó a través de una aproximación del espacio de estados del juego con una red neuronal.

  Para el primer caso, debido a la extensión del espacio de estados, la tabla con la representación de todos los estados y sus acciones se fue construyendo a medida que este se iba entrenando. El pseudo algoritmo implementado fue el mostrado en el algoritmo número 1. Allí se evidencia el proceso que se realiza para una iteración o un juego completo entre el agente y el usuario o su oponente. Este algoritmo se realiza de manera iterativa durante el número de juegos por los cuales se quiere entrenar el agente. Al finalizar se almacenan en un diccionario los valores Q de los estado y acciones visitados en cada juego.

  Ahora bien, como segunda alternativa se manejo la aproximación del estado del juego a través de una red neuronal. La única diferencia respecto al algoritmo presentado anteriormente es que al momento de actualizar los valores Q, estos no se guardan en una tabla estática sino que se van guardando en un archivo de memoria que contiene la información y el resultado de 300 jugadas anteriores. Una vez pasan estos 300 jugadas posibles, se reestrena la red neuronal teniendo en cuenta su estructura y sus parámetros, mostrados en la tabla **3-1**. Esta red neuronal tiene como entrada una matriz 7x7, en donde la primera fila de dicha matriz corresponde a la acción en cada una de las acciones tomadas y la submatrix 6x7 representa el estado del juego para dicha jugada. Para cada memoria posible se entrenó a lo largo de 5 épocas. La estructura de la red fue tomada de un articulo de Towards Science, en donde realizan el mismo procedimiento con el juego de Linea 4 [14]. La diferencia entre la metodología implementada en dicho articulo y en el presente proyecto es que ellos entrenan su agente jugando contra si mismo y contra un agente aleatorio, en cambio en este proyecto el agente de aprendizaje por refuerzo es entrenado jugando contra los agentes de teoría de juegos.

Algorithm 1 Algoritmo de Q-learning para agente de RL de forma Tabular

Input: *target*: α (Tasa de aprendizaje), γ (Tasa de descuento)

Output: Q_{Values}: vector con los valores Q actualizados para los estados y acciones visitados, *Rewards* recompensa acumulada a lo largo de cada juego

$rewards = 0$
while GameOver == False **do**
    if Opponent Action Arrived == True **then**
        Opponent Action Arrived = False
        QValues, currentState = Update GameField
        action = choose_action(currentState)
        Opponent Action Arrived = False
        Reward = action_Reward(GameField, action)
        $rewards$ += Reward
        QValues(currentState) = QValues(currentState) + α*(Reward + $\gamma * max$(QValues(nextState)) − $QValues(currentState)$)
        Send game field to Opponent
Begin new game
return *rewards*

Capa (Tipo)	**Dimensión de salida**	**Número de parametros**
Convolucional 2D	(7,7,32)	832
LeakyReLU	(7,7,32)	0
Convolucional 2D	(7,7,32)	16416
LeakyReLU	(7,7,32)	0
Convolucional 2D	(7,7,32)	16416
LeakyReLU	(7,7,32)	0
Convolucional 2D	(7,7,32)	16416
LeakyReLU	(7,7,32)	0
Flatten	(1568)	0
Densa	49	76881
LeakyReLU	(49)	0
Densa	49	350
LeakyReLU	(49)	0
Densa	49	56
LeakyReLU	(49)	0
Densa	49	8
Número total de párametros: 176,623		

Tabla 3-1: Estructura de la red neuronal usada como representación del juego de Linea 4

Finalmente, es preciso aclarar la metodología realizada a lo largo del diseño y realización de los agentes de teoría de juegos. En primer lugar se implementó la solución de este juego desde la perspectiva del algoritmo de MinMax. El Pseudo código de dicho algoritmo se presenta en al algoritmo número 2. Allí se precisa como se calcula de manera iterativa la mejor acción para el agente según el nivel de profundidad que se especifique. Este nivel de profundidad equivale al número de jugadas que se verá a futuro y el número de jugadas que se calcularan a futuro el valor o estado del juego. El valor del juego calculado al finalizar la profundidad es básicamente el número de fichas consecutivas en cualquiera de las direcciones que potencialmente le podría dar la victoria al agente de aprendizaje por refuerzo. Este algoritmo no se especifica dentro del pseudocódigo mostrado a continuación debido a su extensión y las variables a considerar.

Algorithm 2 Algoritmo de MinMax para el juego de Linea 4

Input: *target*:Depth o nivel de profundidad
Output: *Action*: La acción óptima que el agente debe tomar

```
1: if Player == 0 then
2:     Mejor Valor = -100
3:     for Move in Available Moves do
4:         PlaceColumn(Board,Move)
5:         MoveValue = repeat MiniMax(Board, depth-1, Change Player)
6:         Mejor Valor = max(Mejor Valor, MoveValue)
7:     return Mejor Valor
8: if Player == 1 then
9:     Mejor Valor = 100
10:    for Move in Available Moves do
11:        PlaceColumn(Board,Move)
12:        MoveValue = repeat MiniMax(Board, depth-1, Change Player)
13:        Mejor Valor = min(Mejor Valor, MoveValue)
14:    return Mejor Valor
15: if Depth == 0 or Game is Over then
16:    if Board.Winner is 0 then
17:        return 10000
18:    if Board.Winner is 1 then
19:        return -10000
       Score = Score(board)
```

4 Resultados

En este capitulo se presentan los resultados finales de cada uno de los procesos explicados en la metodología. Por motivos de organización, estos se han decidido dividir en tres partes principales. Al igual que el problema planteado en principio, los resultados estarán dados por la interfaz del juego de linea 4 realizada, los resultados de los adversarios de teoría de juegos y finalmente los resultado en entrenamiento y validación para los agentes con aprendizaje por refuerzo.

4.1. Interfaz de Linea 4 realizada

En este módulo se presenta el resultado de la interfaz diseñada para el juego de linea 4 y en general para todo el proyecto. En la figura **4-1** se presenta la interfaz principal de la aplicación Web. Esta interfaz al ser proyectada en la tablet de Pepper, es la primera que sale y con la primera que interactuará con el usuario. Inicialmente solamente esta la posibilidad de escoger el juego de linea 4, sin embargo, la idea es que como trabajo futuro se desarrollen las interfaces de otros juegos como lo son Othello, ajedrez entre otros. Si esto fuera de esta forma, en esta interfaz principal sería donde el usuario podría elegir el juego de su preferencia. Adicionalmente, dado que por cada juego puede existir diferentes niveles de dificultad, al seleccionar el usuario el juego de su preferencia se desplegará un menú con todas las posibilidades de juego. En la figura **4-2**, se evidencian todas las opciones posibles para que el usuario juegue según su propio criterio. En esta se evidencia como opciones, diferentes niveles de dificultad fácil, medio o difícil y una opción de enseñarle a Ópera a jugar. Las opciones de los distintos niveles de dificultad son jugados contra los adversarios de teoría de juegos. Esto debido a que ellos no requieren ningún tipo de evolución o enseñanza por parte nadie. Por el lado de la opción de enseñarle al robot a jugar, se tiene la posibilidad de que el usuario le enseñe a jugar al agente de aprendizaje por refuerzo. El modelo va guardando todos los estados y acciones realizadas a lo largo de 10 juegos. Una vez sucede esto, el robot le comunica al usuario que esta aprendiendo lo que le acaba de enseñar, dando como resultado el re-entrenamiento de la red neuronal con la información adquirida.
Ahora bien, la interfaz final del juego de linea se presenta en la figura **4-3**. El usuario siempre jugará de primeras para darle mayor oportunidad de ganar sobre el agente de aprendizaje por refuerzo. La ficha del usuario será representada por el color amarillo y la del robot será representada por la cara o símbolo de Pepper en color rojo. A lo largo del juego el robot interactúa de la manera más natural posible con el usuario, reaccionando a cada jugada que

Figura 4-1: Interfaz principal de la aplicación Web

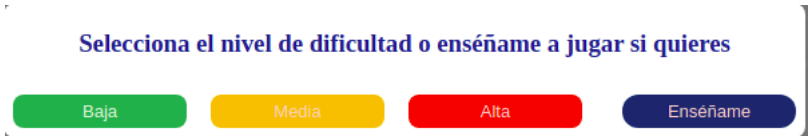

Figura 4-2: Selección de la dificultad para el juego de Linea 4

vayan realizando cada uno. Al finalizar, el robot le muestra al usuario el resultado y reacciona con este, en cualquier caso, bien sea una victoria, perdida o empate. De igual manera, la interfaz cuenta con un botón de salida por si el usuario no quiere jugar más y un par de botones en donde se muestran estadísticas y los puntajes más altos en el presente juego. Al igual que con la interfaz principal, la idea sería que estas estadísticas y puntajes record, varíen respecto a cada uno de los juegos, mostrando no solamente el desempeño que tienen los usuarios que deciden jugar contra Opera sino también el desempeño de Opera misma.

4.2. Adversarios de teoría de Juegos para Linea 4

Como marco de referencia se entrenaron diferentes agentes de teoría de juegos. Como se mencionó en el capitulo anterior se programaron en total tres agentes para el caso de Linea 4: minimax, minimax con alpha-beta pruning y expectmax. Todos estos algoritmos dependían de la profundidad o dificultad en la cuál se seleccionará. Es decir, el hecho de ver dos jugadas adelante (dificultad fácil), hacía que cualquiera de los tres algoritmos no fuera un oponente promedio. En sentido contrario, si cualquiera de estos algoritmo veía más de 6 u 8 jugadas

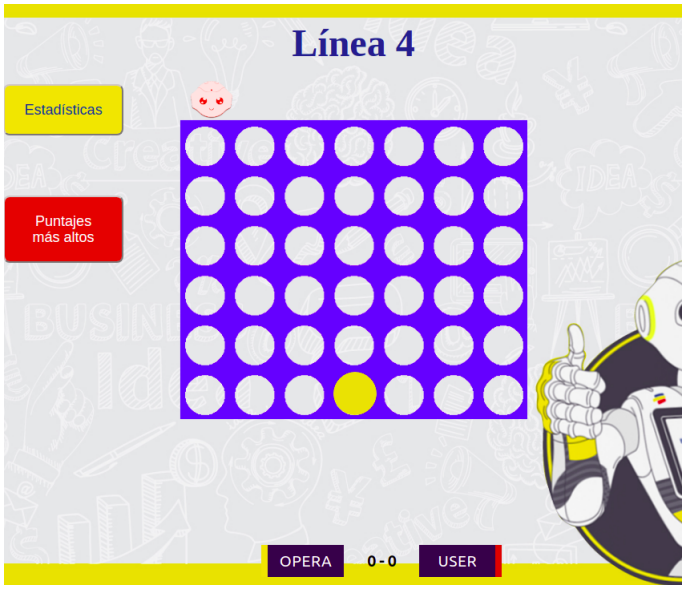

Figura 4-3: Interfaz del juego linea 4

adelante (dificultad alta), era imposible obtener una victoria sobre el mismo. Como ya se dijo anteriormente estos algoritmos son los que componen los diferentes niveles de dificultad de la aplicación y estos son los que juegan contra el usuario que lo selecciona.

En términos de desempeño, los tres algoritmos pueden llegar a tener el mismo desempeño y este es proporcional al tiempo de procesamiento según su nivel de dificultad. En las gráficas **4-4** y **4-5** se presentan una comparación entre cada uno de los algoritmos y su tiempo de procesamiento. En estas dos gráficas se puede ver que el algoritmo que más tiempo de procesamiento toma es el MiniMax debido a la exploración completa que realiza de todo el árbol del juego. Es evidente que el tiempo de procesamiento de cada uno de los algoritmos tiene un comportamiento directamente exponencial respecto al nivel de dificultad que se selecciona. Adicionalmente, a pesar de que no era uno de los propósitos del proyecto, se realizó la comparación entre la implementación de estos algoritmos en diferentes lenguajes de programación (Python y C++), obteniendo como resultado que la implementación en C++ es mucho más eficiente y rápida que la de Python. Al comparar los tiempos de procesamiento de las dos gráficas en cuestión, el de Python supera los 500 segundos para un nivel de dificultad de 8. En cambio, para el caso de C++ se ve que este no supera los 2 segundos para el mismo nivel de dificultad.

Finalmente en la gráfica **4-6** se muestra la comparación de tiempos de procesamiento para un juego normal. En el eje x se muestra el número de la jugada y en el y el tiempo de procesamiento que le toma a cada uno de los lenguajes realizarlo. En esta gráfica nuevamente se vuelve a comprobar la diferencia en términos de eficiencia computacional que tiene C++ sobre Python.

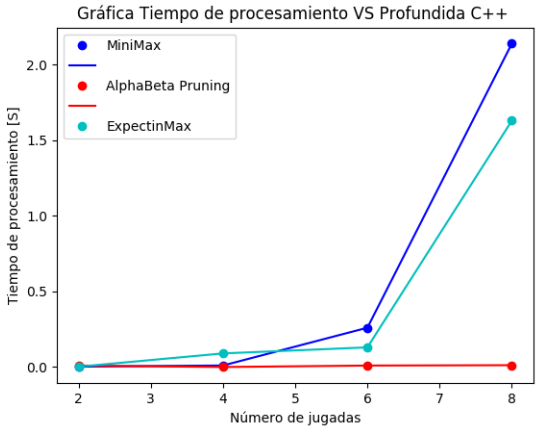

Figura 4-4: Comparación de tiempo de procesamiento entre algoritmos para C++

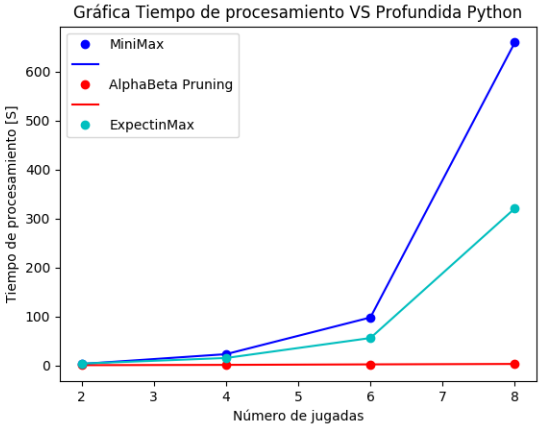

Figura 4-5: Comparación de tiempo de procesamiento entre algoritmos para Python

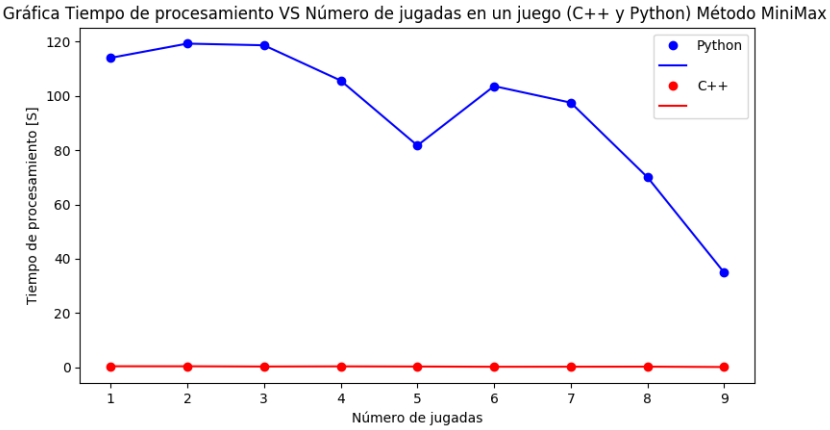

Figura 4-6: Comparación de tiempo de procesamiento entre algoritmos en C++ y Python

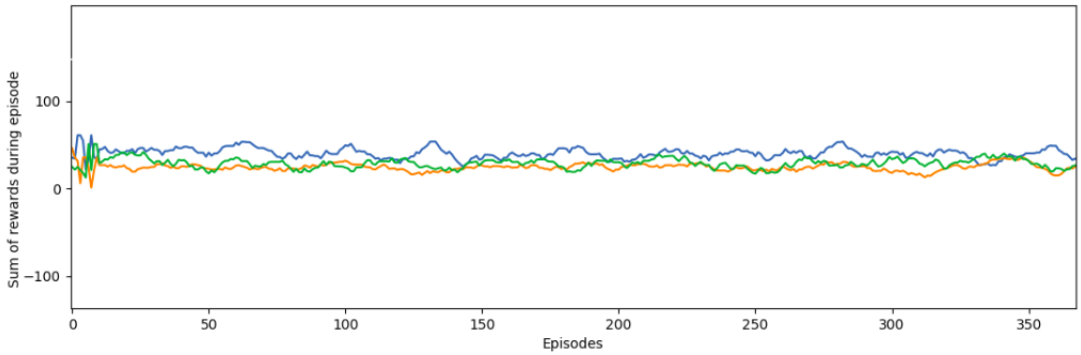

Figura 4-7: Recompensa acumulada para Q-learning tabular a lo largo de los episodios

4.3. Q-learning para Linea 4

Por último se presentan los resultados de la implementación de Q-Learning para la construcción de un agente que aprenda a través de aprendizaje por refuerzo a jugar Linea 4. En primer lugar, se presentan los resultados obtenidos con una representación tabular del juego y seguido de esto, los resultados obtenidos con una red neuronal como función de aproximación del juego.

4.3.1. Q-learning Tabular para Linea 4

En principio se consideró la representación discreta y tabular del juego de Linea 4, debido a que al poder llegar a ser representado como una matriz de una dimensión definida, la mejor manera de representarlo sería esta. Se implementó su recompensa de manera dispersa, como ya explicó en el capitulo de metodología, y durante más de 500 episodios se decidió entrenar jugando con una agente de teoría de juegos. En la figura **4-7** se puede ver el resultado de la recompensa acumulada que tuvo el agente de aprendizaje por refuerzo, siendo entrenado por un agente MiniMax de diferentes dificultades. Normalmente esta recompensa acumulada debería ir creciendo hasta converger a un punto donde el desempeño del agente es óptimo. Ahora bien, como se puede ver en la gráfica en cuestión, para ninguno de los niveles de dificultad ocurre este comportamiento asintótico hacia un valor de recompensa. Esto se puede explicar a partir de los resultados obtenidos en la gráfica **4-8**, en donde se muestra un total de 25000 estado visitados a lo largo de 25000 iteraciones de juegos contra el mismo oponente (Minimax). En esta gráfica se puede ver que solamente uno de los 25000 estados vistos, es visitado todas las veces. Para los demás, solamente alrededor de 10 estados son visitados más de 10000 veces y la mayoría más de 22000 estados solo son visitados una vez en todas las iteraciones realizadas. Es por esto que, debido a la longitud del espacio de estados que tiene este juego no es viable entrenar un agente a través de aprendizaje por refuerzo a través de un algoritmo de Q-learning tabular que sepa jugar linea 4.

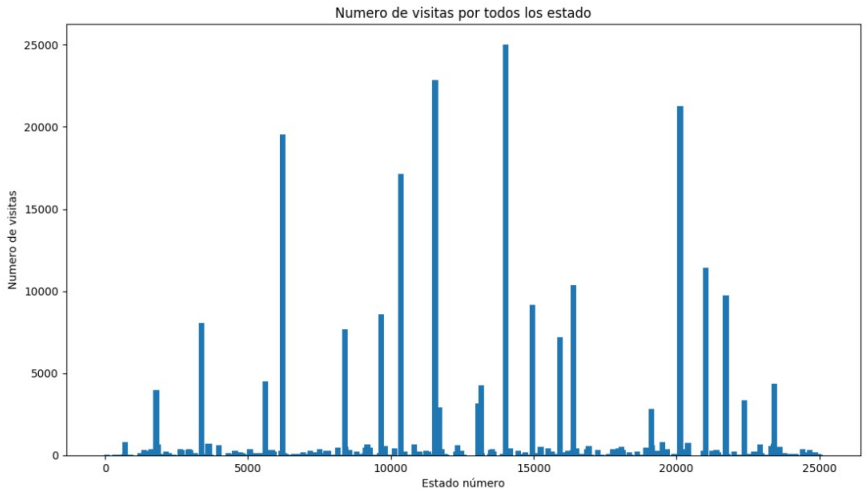

Figura 4-8: Estados posibles del juego vs Número de veces visitadas

4.3.2. Q-learning Red Neuronal para Linea 4

Por último se decidió implementar a través de la representación aproximada de una red neuronal el estado del juego de Linea 4. De esta manera no se tendría problemas de memoria o tiempo de entrenamiento ya que sea cuál sea el estado la red se encargaría de aproximar al estado más próximo cuando llegará a un estado desconocido. Dentro de los agentes entrenados caben resaltar que se entrenaron contra los siguientes agentes:

- Random
- Minimax con dificultad 2
- Minimax con dificultad 4
- ExpectMax con dificultad 2
- ExpectMax con dificultad 4
- Random - Expectimax con dificultad 2 - Minimax con dificultad 4
- Random - Expectimax con dificultad 2 - Expectimax con dificultad 4

Cada uno de estos agentes fue entrenado por cerca de 30000 iteraciones y validado con 1000 juegos contra otro tipo de oponentes. La mayoría de estos en el estado de validación no tuvo el desempeño esperado. Al validar sus resultados se evidenció que solo le gana una vez a su mismo agente contra el cuál estaba entrenando. En la tabla **4-1** se puede evidenciar estos resultados.

Entrenado por	**Adversario**	**Juegos ganados por RL de 1000**	**Juegos ganados por adversario de 1000**
Random	Random	449	542
Random	ExpectMax Depth 2	1	999
MiniMax Depth 2	Random	479	511
MiniMax Depth 2	ExpectMax Depth 2	1	999
ExpectMax Depth 2	Random	468	523
ExpectMax Depth 2	ExpectMax Depth 2	2	998
Random ExpectMax Depth 2 MiniMax Depth 2	Random	483	517
Random ExpectMax Depth 2 MiniMax Depth 2	ExpectMax Depth 2	26	974
Random ExpectMax Depth 2 ExpectMax Depth 4	Random	683	317
Random ExpectMax Depth 2 ExpectMax Depth 4	ExpectMax Depth 2	416	584

Tabla 4-1: Juegos ganados por el agente de RL entrenado por diferentes agentes de teoría de juegos vs Juegos ganados por diferentes agentes de teoría de juegos y aleatorio

En esta tabla se muestra que al entrenar con agentes de teoría de juegos y validar después con agentes aleatorios, existe más del 40 % de probabilidad de ganar. Sin embargo, al validar con el propio agente con el cuál fue entrenado, se evidencia una falencia en términos de entrenamiento o ya dentro de la estructura de la red seleccionada. Esto debido a que existe menos del 1 % de probabilidad de victoria para el agente de Q-learning entrenado. Estos casos anteriores se repite para los algoritmos de MiniMax y ExpectMax, sin importar la profundidad seleccionada. Es importante recalcar el hecho de que entre más complejo sea el oponente, se requerirá un mayor número de iteraciones a lo largo del aprendizaje. Esto debido a que al aumentar la dificultad implica que durante el entrenamiento le va a costar mucho ganar y teniendo una recompensa dispersa como la que se está manejando, esto requerirá un número alto de iteraciones.
Por otro lado, cuando se decide combinar diferentes oponentes a lo largo del entrenamiento, los resultados mejoran sustancialmente. A pesar de que en el caso que se combinan los tres casos u oponentes usados (Random, ExpectMax y MiniMax) el porcentaje de victoria contra el agente de teoría de juegos sigue siendo menos del 10 %, es evidente que aumenta respecto a los casos no combinados. Ahora bien, cuando se combinan tres agentes con un nivel de dificultad menor (Random, ExpectMax dificultad 2 y ExpectMax dificultad 4) el resultado ya llega a ser mayor al 40 % para el caso contra teoría de juegos.
Esto demuestra que para mejorar su desempeño hizo falta un mayor número de iteraciones y mayor tiempo de simulación para encontrar un agente que juegue de una manera óptima. Al mismo tiempo este último resultado demuestra la necesidad de incluir estudios de Transfer Learning en este tipo de proyectos debido a que es necesario estudiar más en detalle cuál es la combinación de agentes perfecta contra los cuales el agente de aprendizaje por refuerzo se debe entrenar.

5 Conclusiones

En primer lugar, es relevante mencionar la integración de un sistema de robotica social como lo es el robot Pepper con diferentes módulos que conforman este proyecto. En principio, el desarrollo y establecimiento de una servidor web que sirva como interfaz gráfica para la interacción humano robot no solamente se limita para una aplicación de entretenimiento como lo fue el caso de este proyecto. Es más las aplicaciones que se le puede dar a este tipo de desarrollos e integraciones con el robot son demasiadas, dentro de las cuales se puede mencionar la guía dentro de un establecimiento público o privado, una alternativa para la interacción oral con el propio robot, entre otros.

Por otro lado, se logró solucionar a través de diferentes algoritmos de teoría de juegos el juego en cuestión de Linea 4. Entre ellos se tienen MiniMax, ExpectMax y MinMax con Alpha-Beta pruning. Entre ellos se comparó el desempeño en términos computacionales y de consumo operacional. Además de su propio desempeño en implementados en diferentes lenguajes de programación (C++ y Python). Estos algoritmos sirven como marco de referencia para los agentes que fueron entrenados a través de aprendizaje por refuerzo. Además se debe resaltar el hecho de que estos algoritmos son quienes entrenan a dichos agentes de RL. Esto con el objetivo de entrenar un agente que tuviera un propósito y fuera capaz de superar el oponente promedio.

Ahora bien, por parte de dicho agente entrenado a partir de aprendizaje por refuerzo fue implementado el algoritmo de Q-learning. Este algoritmo se puede implementar de forma tabular o a través de una función de aproximación del juego como lo es una red neuronal. Por el lado tabular, se encontró que debido a la longitud del espacio de estados con el que cuenta el juego de linea 4, no es viable o eficiente en términos computacionales entrenar un agente a través de esta manera. Visitante más de 22000 estados solamente una vez a lo largo de 25000 iteraciones, se puede concluir que este tipo de implementación tabular solamente se puede realizar para problemas descritos como problemas de decisión de Markov pequeños o con un espacio de estados más recudido.

Al usar una red neuronal como función de aproximación del juego se obtuvo que esta debe ser entrenada de manera paulatina, enseñándole poco a poco diferentes jugadas buenas y malas que le permitan aprender algo previo a jugar contra un agente con un objetivo especifico como lo son los de teoría de juegos. Además de esto, se ve la necesidad de realizar un estudio alrededor de Transfer Learning en donde se especifique con mayor detalle que tanto puede llegar aprender un agente de RL contra otro cualquiera, y que tanto de dicho aprendizaje utiliza contra otro agente diferente.

Finalmente, para futuros proyectos relacionados con este tema, sería importante tener en cuenta el tiempo de procesamiento o entrenamiento de los agentes en aprendizaje por refuerzo para ser lo más eficientes posibles al momento de entrenarlos. Además se debería evaluar los diferentes parámetros con los que cuenta la función de aprendizaje Q, como lo son el factor de descuento y la tasa de aprendizaje para cada caso, encontrando los valores que optimicen el comportamiento a lo largo del aprendizaje del agente.

Bibliografía

[1] C. Arena, "Juego de linea 4,"

[2] N. Nogueria, "Minimax algorithm,"

[3] E. T. Garcia, *Diseño y construcción de un programa de computador capaz de jugar Othello basada en Aprendizaje por refuerzo.* 2005.

[4] A. Ravindran, "Understanding django channels,"

[5] S. Moriello, "Robots sociales, la nueva generación," 2008.

[6] S. G. Corp, "Nao,"

[7] S. G. Corp, "Pepper robot: Technical specifications,"

[8] ROS, "About ros,"

[9] ROS, "About ros nodes,"

[10] ROS, "Abour ros topics,"

[11] E. S. Michael Maschler and S. Zamir, *Game Theory.* 2013.

[12] R. S. Sutton and A. Barto, *Reinforcement learning: an introduction.* 1992.

[13] A. SinfonIA, "Robot toolkit,"

[14] G. Wisney, "Deep reinforcement learning and monte carlo tree search with connect 4,"