UNIVERSIDAD PRIVADA BOLIVIANA

FACULTAD DE INGENIERÍA Y ARQUITECTURA

CARRERA DE SISTEMAS COMPUTACIONALES

APLICACIÓN DE Q-LEARNING EN LUDO

Estudiante: Carlos Andrés Coronado Arrazola

Materia: Inteligencia Artificial

Docente: Lesly Zerna Orellana

Cochabamba, mayo 2021

1. Justificación

Se escogió este tema debido a la fascinación del concepto de que una máquina va aprendiendo a partir de la experiencia. Los demás conceptos de Machine Learning se ven limitados por la data disponible que tienen para ser entrenados, si el mismo concepto se aplicará a un agente inteligente basado en supervised learning cuya tarea sea la de ganar un juego, su inteligencia y capacidad se vería limitada a la de las personas de las que se obtuvo la data sobre la cual este fue entrenado. El aprendizaje reforzado no solo permite al agente no depender de data extraída, sino que también le permite superarse a sí mismo.

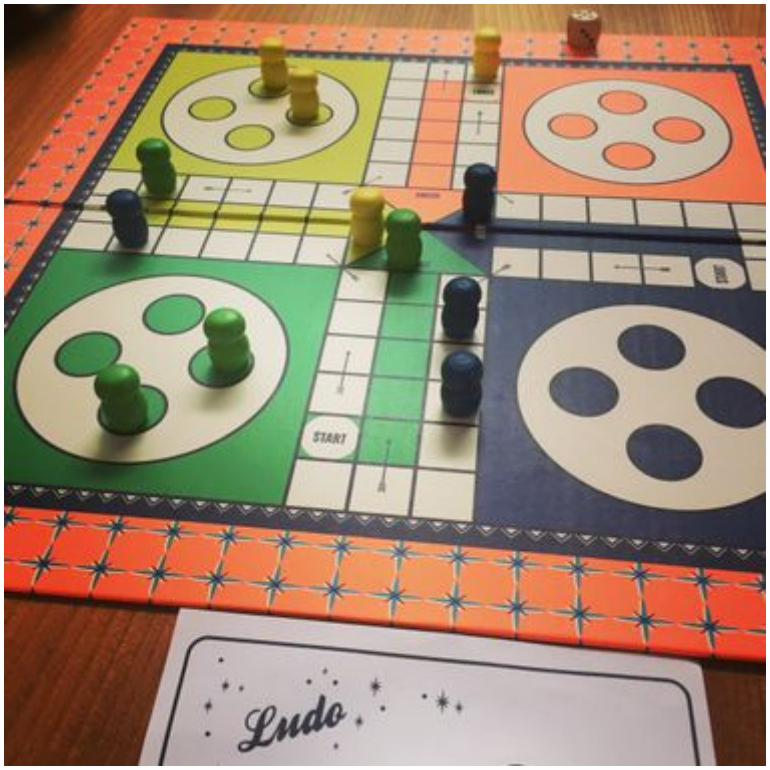
2. Detalle de Desarrollo

a. Descripción del Proyecto

El objetivo del proyecto es crear un agente inteligente capaz de aprender a partir de la experiencia obtenida mediante las partidas jugadas, ya sea contra sí mismo o contra personas. El programa se considerará óptimo si el agente consigue ganar al menos dos tercios de los juegos contra personas.

i. Descripción del juego Ludo

El **objetivo del juego** es llevar todos tus peones a la casa designada que se encuentra en el centro del tablero



Las reglas más importantes son:

- Todos los peones empiezan en la base.
- Para sacar un peón se necesita obtener un 6 con el dado.

- Cuando se obtiene un 6, el jugador puede sacar un peón o mover otro, si es que ya tiene uno en el tablero.
- El jugador que saque 6 tiene otro turno para lanzar el dado.
- Solo se puede mover uno de los peones que se tiene en el tablero a la vez.
- Para capturar los peones de los oponentes se debe de caer en la misma casilla en la que se encuentran estos.
- Si dos o más de los peones de un jugador se aglomeran en una casilla se forma un bloqueo.

Las cosas necesarias para jugar son : 1 dado de 6 caras, 1 tablero, 16 peones (4 de cada tipo)

ii. Reglas omitidas para mayor simplicidad en el desarrollo

Si algunas de las reglas son implementadas agregarían demasiada complejidad innecesaria al desarrollo del programa. Las reglas omitidas son las siguientes:

- Si dos o más de los peones de un jugador se aglomeran en una casilla se forma un bloqueo.
    - Justificación.- Se omitió esta regla debido a que un constante control sobre cuales casillas están bloqueadas y cuáles no requeriría de implementar una lógica que resultaría muy pesada para el programa. Si bien hay una entidad que realiza un control constante sobre el tablero y donde se encuentran sus piezas, únicamente maneja un tipo de dato y todo el programa está implementado alrededor de esta entidad. Se puede implementar otra estructura de datos donde almacenar estas casillas bloqueadas, pero también se tienen que guardar cuantos peones forman el bloqueo, cuando se mueve uno el programa tendría que verificar si este peón estaba formando un bloqueo o no.
    - Con esto se determina que si un Jugador puede avanzar a una casilla pero ya se encuentra otro peón del mismo jugador en esa localidad, el movimiento es inválido.
- Si el jugador saca 6 tiene otro turno para lanzar el dado.
    - Justificación.- En este caso esta regla no implementa una lógica muy compleja, pero el agente inteligente que jugará la partida está planteado con una percepción parcial de su ambiente, por lo que no puede ver una jugada más allá, en otras palabras no es capaz de realizar jugadas compuestas, esto se explicará con mayor detalle más adelante. Debido a esto, se determinó que no tendría demasiada relevancia implementar esta regla o no.

b. Tecnologías utilizadas

i. Algoritmo Q-Learning

Reinforcement Learning se basa en dar recompensas y penalidades al agente inteligente en función de lo que este decida hacer. El algoritmo Q-Learning, el cual será usado, se basa en el uso de una tabla ***s* => *a*** (estado => acción), en esta tabla es donde se guardará el valor adjudicado a una acción dado cierto estado y, cada que se tenga un estado y se requiera una acción se recurrirá a la tabla para obtener la acción con valor más alto.

Los valores de la Q-Table serán determinados por la ecuación:

$$Q(s,a) \leftarrow Q(s,a) \ + \ \alpha[r \ + \ \gamma \ * \ max_{a'} Q(s',a') \ - \ Q(s,a)]$$

Para determinar cuando el agente explorará y cuando explotará el conocimiento obtenido se usará **epsilon greedy.**

ii. Epsilon greedy

Es una estrategia de aprendizaje diseñada para resolver el problema de exploración y explotación de los conocimientos obtenidos por parte del agente. Primeramente se determina una variable llamada ***exploration rate*** con un valor inicial de 1, esta representa la probabilidad de que el agente inteligente explore el ambiente en vez de explotar los conocimientos obtenidos. Con un valor de 1 es 100% seguro de que el agente empezará explorando el ambiente. Con cada episodio se irá disminuyendo el valor de ***exploration rate*** haciendo cada vez más probable que el agente explote su conocimiento.

La idea para explotar esta técnica es usar a ***exploration rate*** como un threshold, generando un número ***x*** random entre 0 y 1 y utilizando el siguiente criterio:

si x > exploration rate:
    Escoger una acción a realizar mediante exploración
sino
    Escoger una acción a realizar mediante explotación

iii. Pickle

Pickle es un módulo que implementa protocolos binarios para serializar y deserializar un objeto en python. Se debe de poder guardar los estados de las Q-Tables generadas por el programa, esto para no tener que entrenar al programa de 0 cada que se quiera jugar contra el. Las razones por las que se escogió Pickle por sobre módulos como Json o Pandas con el .csv, son los siguientes:

- Pickle puede serializar prácticamente cualquier objeto de python. El formato usado por pickle es específico para python, esto libra de restricciones impuestos por estándares como los impuestos en el módulo Json y cómo en este caso se espera que solo el programa interactúa con los archivos, no es necesario que otros programas implementados en otros lenguajes deban de poder abrirlo.
- Serialización simple, ideal si se busca un rendimiento ideal en tiempo de ejecución.
- La lectura o escritura de un archivo .pickle o pkl es más rápido que la de un archivo .csv, aspecto importante debido a que la Q-Table para 4 peones en el tablero es considerablemente grande y el programa debe de cargar los estados guardados y guardarlos igualmente.

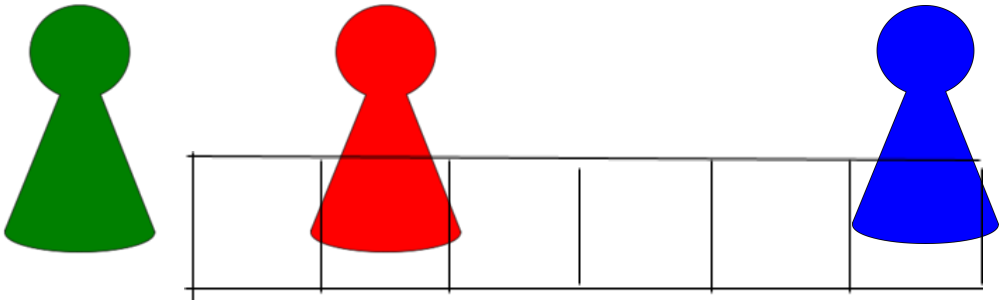
c. Agente Inteligente: tratamiento de datos

El agente inteligente podrá escoger una acción **a** del conjunto de acciones **A** dado un cierto estado **s** dentro del conjunto de estados **S**. La consecuencia de realizar esta acción desencadenará un nuevo estado **s’ ϵ S** y se obtendrá un valor **r** por esta consecuencia.

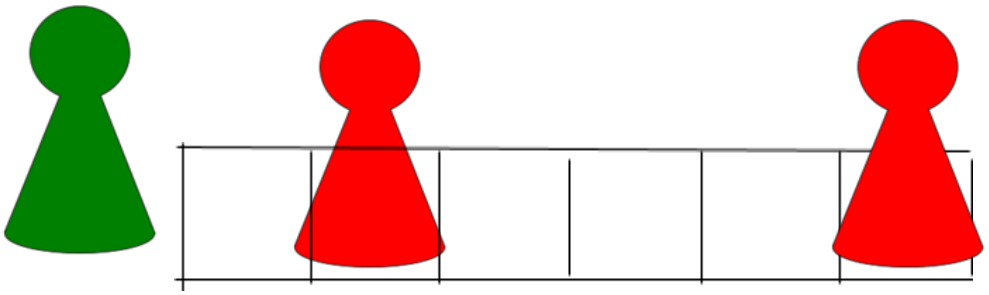
Debido a la naturaleza del juego, el conjunto de acciones **A** que pueda realizar el agente variará de acuerdo al número que salga en el dado y de acuerdo a la cantidad de peones que se tenga en el tablero.

Características del agente

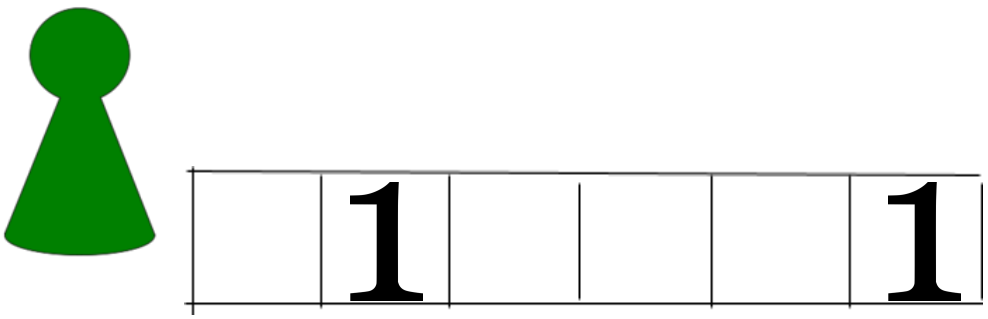
- El agente no tendrá un entorno completamente observable, sino más bien uno parcial. El tablero de Ludo consta de 52 casillas en la que todos los jugadores pueden estar y 5 que están destinadas a un color en específico. Con 52 casillas y 16 peones, la cantidad de estados se dispararía, es por esto que se decidió que el agente únicamente tenga constancia de las 6 casillas de en frente de cada peón que se encuentre en el tablero, puesto a que el número máximo de casillas que el peón puede recorrer cada que lanza el dado es de 6.



  El agente no tiene porqué diferenciar entre los demás jugadores, considerará a todos los demás como oponentes o enemigos. Esto simplifica aún más la percepción del agente sobre su entorno, pues únicamente verá casillas vacías u ocupadas.



  Esto se lo puede ver de manera binaria, o está un enemigo en la casilla o no, esto reduce la cantidad de estados a solo $2^6 = 64$ si se tiene a un solo peón en el tablero.



Si los cuatro peones se encuentran en el tablero la cantidad de estados sería de $64^4 = 16777216$

Los estados en binario se representarán en forma numérica dentro del Q-Table

La posible desventaja es que el agente no aprenderá a realizar jugadas compuestas de varios movimientos, pues solo podrá ver un movimiento delante de él. Otra desventaja es que el agente no tendrá constancia del entorno detrás, por lo que tomará sus decisiones sin tomar en cuenta si hay oponentes a sus espaldas, cosa que puede ser un aspecto relevante al momento de decidir si mover un peón o no.

- Las acciones que el jugador podrá realizar serán:
    - Mover a los peones que se encuentren en el tablero y que cumplan las reglas establecidas para el juego
    - Poner a un peón en el tablero si es que el número que se sacó con el dado es 6

  Se tendrá una Q-Table para cada cantidad disponible de peones a mover, exceptuando la ocasión en la que haya solo un peón disponible.

  Se plantea las Q-Tables de la siguiente manera:
    - Para x > 1 peones disponibles

State	Move Pawn 1	Move Pawn 2	….	Move Pawn x	Put Pawn on Table
(0, 0, 0, 0)	q1_1*	q1_2*	….	q1_4*	q1_5*
…..					
(64, 64, 64, 64)	….	….	….	….	….

Así, si es que el dado devuelve un valor a menor a 6 se omite la última columna en el análisis.

d. Límites y alcances

i. Límites del agente

El agente no aprenderá a realizar jugadas compuestas de varios movimientos, pues éste sólo podrá ver un movimiento delante de él a la vez. Además, el agente no tendrá constancia del entorno detrás, por lo que tomará sus decisiones sin tomar en cuenta si hay oponentes a sus espaldas, cosa que puede ser un aspecto relevante al momento de decidir si mover un peón.

ii. Límites del programa

El programa no cuenta con un apartado gráfico, solo se puede jugar contra el agente por consola, cosa que no lo hace tan “user friendly”. También las Q-Tables no se encuentran

optimizadas, por lo que ocupan un espacio considerable, y al no tener un apartado gráfico no puede desplegar de manera gráfica cuál es la posible decisión que el agente pueda tomar, esto para comprender mejor las decisiones que este tome.

iii. Alcances del agente

Con el suficiente entrenamiento el agente puede jugar en un nivel aceptable, esto pues el no conocer el entorno detrás de él lo limita a no conocer el riesgo de mover una pieza u otra, pero para solo conocer su entorno de adelante, pues es capaz de tomar buenas decisiones con solo esa información.

iv. Alcances del programa

Se puede entrenar al agente desde 0 y desde un estado que se haya guardado, esto es posible debido a que el programa guarda los estados de las Q-Tables y los puede volver a cargar, esto es útil para que el agente no se tenga que entrenar de 0 cada vez

e. Entrenamiento del modelo

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  until s is terminal

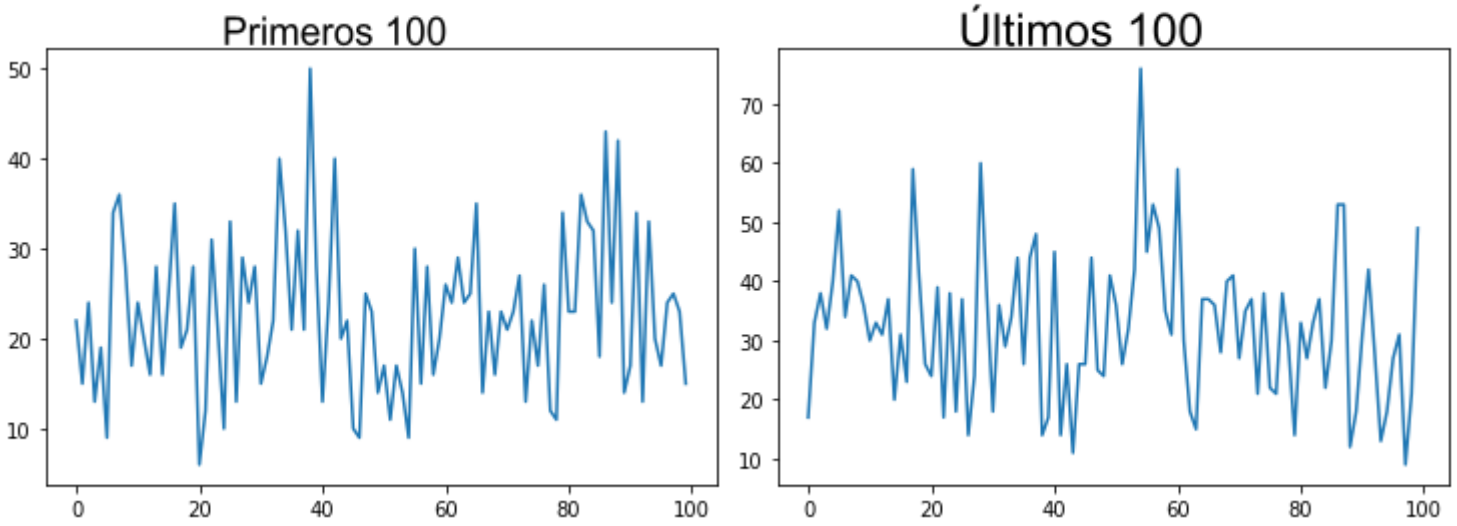
Se utilizó como base el algoritmo de arriba, con la diferencia de que cada step dentro del episodio es una ronda de turnos en el juego. Si se hace que el agente juegue contra sí mismo un número ***e*** de episodios, con ***s*** steps y ***4*** jugadores se obtiene un total de $e * s * 4$ como máximo número de iteraciones, en otras palabras, la capacidad de que el agente pueda jugar contra sí mismo multiplica por 4 su ritmo de aprendizaje.

No se manejo penalties debido a que el agente no sabría el porqué de un penalty debido a que no es capaz de ver el riesgo de mover una pieza. En cuanto a las recompensas, esta se las otorga cuando el agente pone un peón en el tablero, cuando come a un peón de otro jugador y cuando lleva a un peón hasta su casa designada.

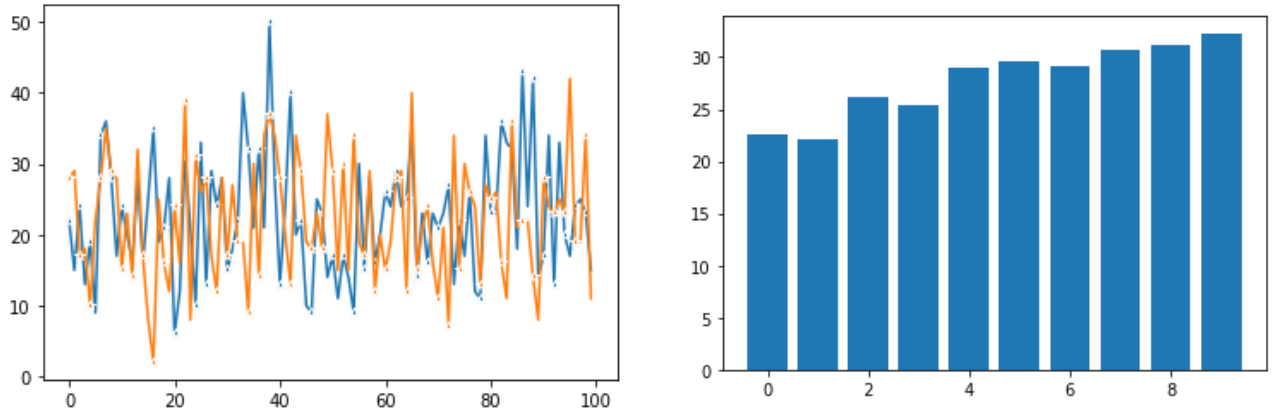
f. Evaluación del modelo

Para esta muestra se entrenó al modelo con 1000 episodios con 100 steps cada uno, dando como máximo número de iteraciones = 400000. De los resultados se obtuvo los siguientes gráficos:

Ya entrenando al algoritmo con otros 500 episodios se obtuvo los siguientes gráficos:



Comparando los gráficos se puede apreciar picos más altos en los rewards de los últimos 100 episodios. Esta diferencia se muestra más distinguible en el siguiente gráfico:



En el gráfico de arriba a la izquierda se puede apreciar una considerable mejora, siendo representado con el color naranja las últimas 100 iteraciones y con color azul las primeras 100. Lo que cabe resaltar de este gráfico es que en los primeros 100 episodios el agente inteligente tiene una media de rewards de 22.66, en cambio en los últimos 100 episodios muestra una mejora con una media de 32.3 prueba de que existe una leve mejora, esto se ve mejor apreciado en el gráfico de arriba a la derecha. Si bien para 400000 iteraciones esto puede parecer poco margen de mejora, se debe tomar en cuenta la abrumadora cantidad de estados con la que se trabajó, por lo que para que el agente alcance valores más altos debe pasar por otra abrumadora cantidad de entrenamiento.

g. Recomendaciones y conclusiones

Si se va a realizar un proyecto haciendo uso de Q-Learning, se recomienda ver el problema de la manera más simple posible, para así contar con la menor cantidad posible de estados. La desventaja de este método de aprendizaje reforzado es que la eficiencia sobre el aprendizaje del agente inteligente depende de las dimensiones de la Q-Table que se use y de los valores otorgados a los rewards o penalties, pues a mayor cantidad de estados, más entrenamiento tendrá que tener el agente para poder obtener experiencia sobre todos ellos.

En conclusión, el aprendizaje reforzado es un campo con un rango de aplicaciones enorme, de las cuales unas pueden ser más relevantes que otras. En este caso, el proyecto tenía un propósito de aprendizaje, para indagar sobre el algoritmo de Q-Learning, pero si se agrega un apartado gráfico que muestre el funcionamiento del agente inteligente de forma gráfica, si se realizan ciertas optimizaciones en el uso de Q-Tables y además se agregan ciertas consideraciones como que el agente tenga constancia del entorno detrás de él, se puede obtener un proyecto que tenga más relevancia, pues puede servir como ejemplo didáctico para quienes quieran indagar y comprender mejor este tema.

h. Link del Repositorio

https://github.com/carloscoronad0/Smart_Ludo