# Reinforcement Learning aplicado en la Solución Automática de un Laberinto

Alejos Arrieta, Fredo Antonino
*20190390D*
*Universidad Nacional de Ingeniería*
Lima, Perú
fredo.alejos.a@uni.pe

Diaz Ramos, Noe Elvis
*20181362A*
*Universidad Nacional de Ingeniería*
Lima, Perú
noe.diaz.r@uni.pe

## I. Enunciado del Caso

**Diseño de un laberinto resuelto automáticamente mediante redes neuronales y programado en Python**

## II. Funcionamiento del Software

Es una implementación de un juego de laberinto en Pygame. Las clases Player y Wall se utilizan para representar al jugador y a las paredes del laberinto, respectivamente. La función arraymax devuelve una lista de índices de elementos máximos en una lista dada. El nivel del laberinto se representa como una lista de cadenas, y se utiliza para crear una lista de paredes y un rectángulo para el final del laberinto.

Después de definir el laberinto, el código define los estados posibles y las acciones que puede tomar el jugador. Los estados se representan como coordenadas (x, y) y las acciones como cadenas "UP", "DOWN", "LEFT", y "RIGHT".

La función de recompensa, reward, toma un estado y una acción como entrada y devuelve una recompensa. Si el estado es el mismo que la posición del final del laberinto, se devuelve una recompensa de 100. De lo contrario, se devuelve una recompensa de 0.

## III. Limitaciones del Software

El laberinto se define como una lista de cadenas, lo que puede ser poco práctico si queremos modificar el laberinto de alguna manera. Puede ser más fácil representar el laberinto como una matriz de celdas o una lista de objetos Wall y End en lugar de una lista de cadenas.

El código no maneja bien la colisión entre el jugador y el final del laberinto. En lugar de devolver la recompensa de 100 cuando el jugador llegue al final, el código debería detener el juego o volver al menú principal o algo similar.

El código no tiene ningún mecanismo para elegir una acción para el jugador. Esto podría ser una limitación importante si quieres implementar algún tipo de inteligencia artificial o aprendizaje automático para que el jugador tome decisiones.

El código no tiene un bucle de juego principal.

## IV. Marco Teórico

### A. Aprendizaje por refuerzo

El aprendizaje por refuerzo es una rama del machine learning en la cual la máquina guía su propio aprendizaje a través de recompensas y castigos. Es decir, consiste en un sistema de instrucción autónomo cuyo camino es indicado según sus aciertos y errores.

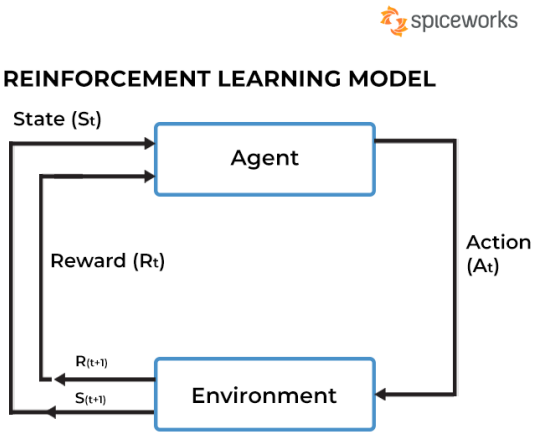


Fig. 1. Modelo básico del aprendizaje por refuerzo. Fuente: Adaptado de [2]

Consta de un aprendizaje empírico, por lo que el agente informático está en constante búsqueda de aquellas decisiones que le premien de algún modo, a la par que evita aquellos caminos que, por experiencia propia, son penalizados.

También, se puede decir que el aprendizaje reforzado es un concepto similar al que utilizan los seres vivos. Esto es, las máquinas aprenden qué decisiones tomar de acuerdo a la situación en la que se encuentren. Además, son capaces de desarrollar estrategias con una visión a largo plazo. [1].

### B. Q-Learning

Se trata de una técnica de aprendizaje por refuerzo. Consiste en que el sistema aprenda una serie de normas que le diga al agente que accion bajo que circunstancias. Entre sus características principales se encuentra en que no requiere un modelo de entorno y puede manejar problemas con transiciones estocásticas y recompensas sin requerir adaptaciones.

*1) Funcionamiento:* El algoritmo del Q-Learning funciona con un conjunto de datos llamada matriz Q, que se actualiza

progresivamente mediante la siguiente función.

$$Q_t^{new} = (1-\alpha)Q_t^{old} + \alpha(r_t + \gamma max_a(Q_{t+1})) \quad (1)$$

Donde, $\alpha$ representa la tasa de aprendizaje, $\gamma$ representa el factor de descuento, el cual es encargado de evaluar las recompensas recibidas anteriormente con un valor mayor que las recibidas posteriormente. Finalmente Q(t) es el valor obtenido de Q según el estado y accion aplicado en el tiempo actual t. Es importante que el significado cualitativo de $max_a(Q_{t+1})$ es un estimado de un valor optimo en el futuro.[3]

## V. Diagrama de Clases

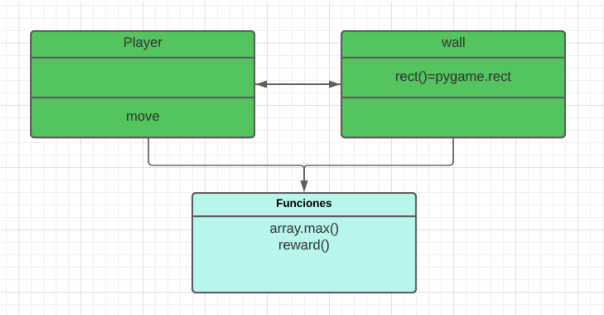


Fig. 2. Diagrama de Clases

## VI. Algoritmo de las Funcionalidades del Software

La clase Player representa al jugador y tiene un método move que se utiliza para mover al jugador en una sola dirección. La clase Wall representa una pared y solo tiene un atributo, rect, que es un objeto pygame.Rect que se utiliza para representar la posición y el tamaño de la pared. La función arraymax toma una lista como entrada y devuelve una lista de índices de elementos máximos en la lista. La lista de cadenas level contiene el diseño del laberinto. La función reward toma un estado y una acción como entrada y devuelve una recompensa. El código inicializa Pygame y crea una ventana para dibujar el laberinto y el jugador. El código crea una lista de paredes y un rectángulo para el final del laberinto a partir de la lista de cadenas level. Los estados y las acciones se definen como variables globales.

## VII. Descripción de las funciones

Dentro de las funciones que se emplearon se encuentran las siguientes:

- **Dentro de el objeto Player** En el objeto player se encuentran las siguientes funciones:
  - **move(self, dx, dy)**: Esta función se encarga de trasladar la ubicacion del jugador, mediante un desplazamiento indicado por dx y dy, esta hace uso de la funcion **move_single_axis(self, dx, dy)**.
  - **move_single_axis(self, dx, dy)**: Esta funcion se encarga de trasladar el jugador en una sola dirección, garantizando que no habra solapamiento entre el jugador y los muros del juego.
- **arraymax(array)** Esta función se encarga de obtener los indices de todos los elementos con el maximo valor dentro del arreglo **array**. Esto se emplea para la elección de dirección del jugador durante el aprendizaje.

## VIII. Descripción de las variables

Para el caso que se pretende resolver (un laberinto). Se identifican las siguientes variables para el proceso de aprendizaje automático.

### A. Variables de entrada

Que tambien pueden ser identificadas como variables de observación. Para este caso las entradas del sistema son las siguientes.

1) **Posición x**
2) **Posición y**

### B. Variables de salida

Estas también pueden ser llamadas variables de acción. Dentro de este trabajo se reconocen 2 variables simples.

1) **Mover a la derecha**
2) **Mover a la izquierda**
3) **Mover hacia arriba**
4) **Mover hacia abajo**

## IX. Capturas del funcionamiento del software

Primero tomaremos una captura del juego (Figura 3).

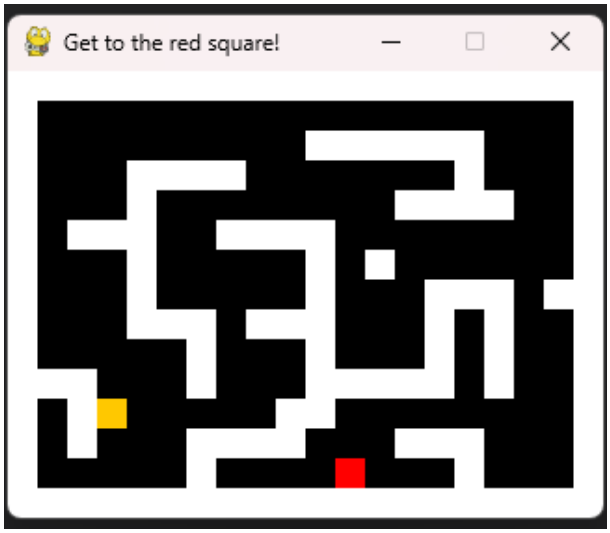


Fig. 3. Muestra del juego en proceso

Segundo mostraremos un link donde podrá visulizar el funcionamiento de programa mediante un video

Ahora los resultados de Q son mostrados en las figuras 4 y 5.

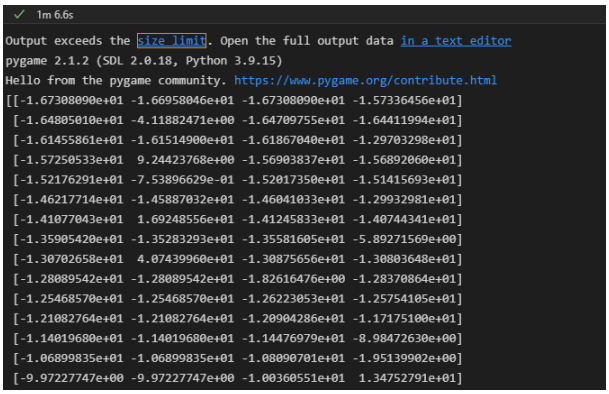


Fig. 4. Primera parte del resultado de Q

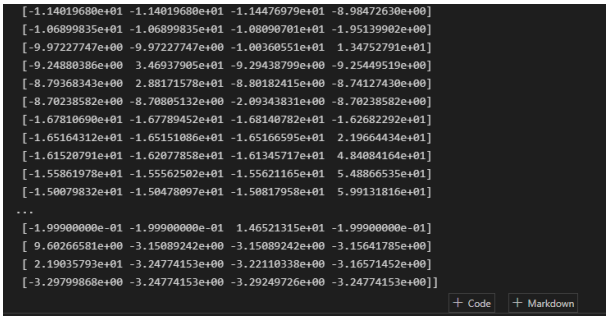


Fig. 5. Segunda parte del resultado de Q

## X. Conclusiones

- Se concluye que el algoritmo de Q-learning funciona satisfactoriamente según los objetivos de este informe, llegando incluso a obtener el camino más corto.
- Se concluye ademas que el sistema podría también resolver laberintos mucho más grandes con el número adecuado de episodios.
- Finalmente, se da como una observación que aunque se trata de una posibilidad extremadamente baja, existe la posibilidad de que tal vez el camino encontrado por el sistema sea uno más largo.

## XI. Código Fuente

El codigo fuente empleado se encuentra en el siguiente link https://drive.google.com/drive/folders/1LFOdUVzh69ejH-xOkXgiasi2vp5oef-O?usp=sharing

## References

[1] J. Torres, *Introducción al aprendizaje por refuerzo profundo*. Independently Published, 2021.
[2] V. Kanade, "What is reinforcement learning? working, algorithms, and uses." [Online]. Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-reinforcement-learning/
[3] J. Beakcheol, K. Myeonghwi, and H. Gaspard, *Q-learning Algorithms: A Comprehensive Classification and Applications*, 2017.