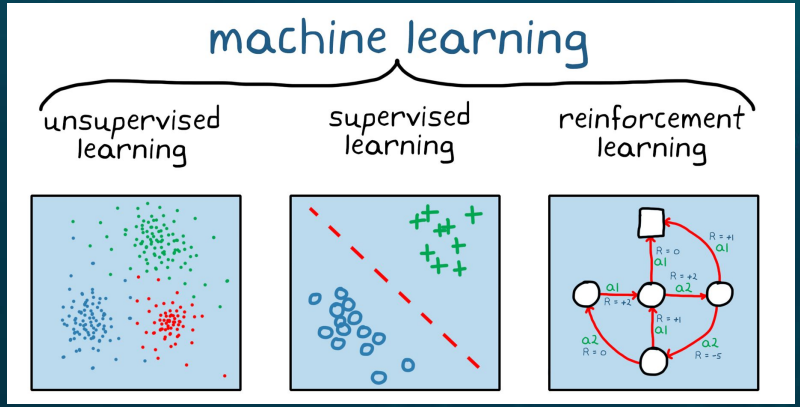
SnakeGame con Reinforcement Learning

Alan Stiven Camacho Restrepo

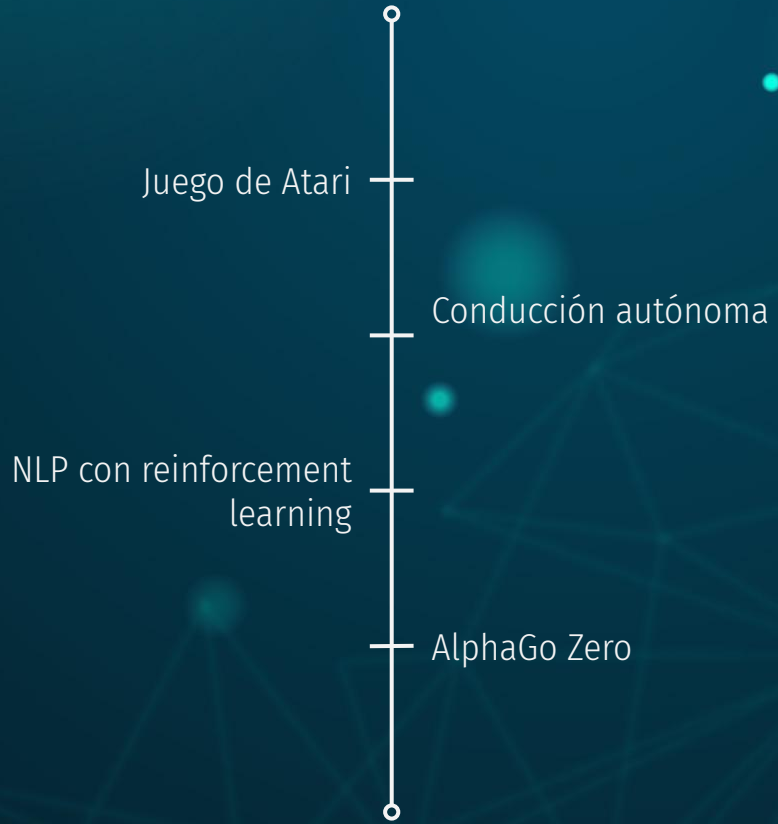
CONTENIDO

¿Qué es el Reinforcement Learning?



- Aprendizaje supervisado y no-supervisado.
- ¿Hay una manera de crear un agente que aprenda a jugar por sí mismo?
- Agentes y recompensas.
- Usos más frecuentes.

APLICACIONES DEL RL



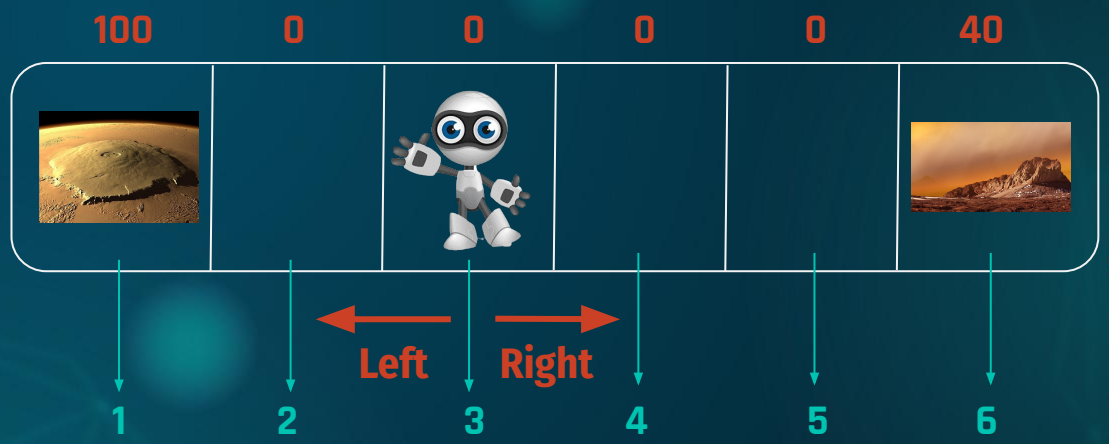
02 PLANTEAMIENTO DEL PROBLEMA

OBJETIVO GENERAL

- Construir un algoritmo de deep learning que permita que el juego de la culebra se juegue por sí misma.

OBJETIVO ESPECIFICOS

- Entorno del juego.
- Agente.
- Recompensas.
- Visualización del juego.



- Estado.
- Acción.
- Recompensa.
- Nuevo estado.

`(s,a,R(s),s')`

Return: R1 + γ R2 + γ^2 R3 + ...

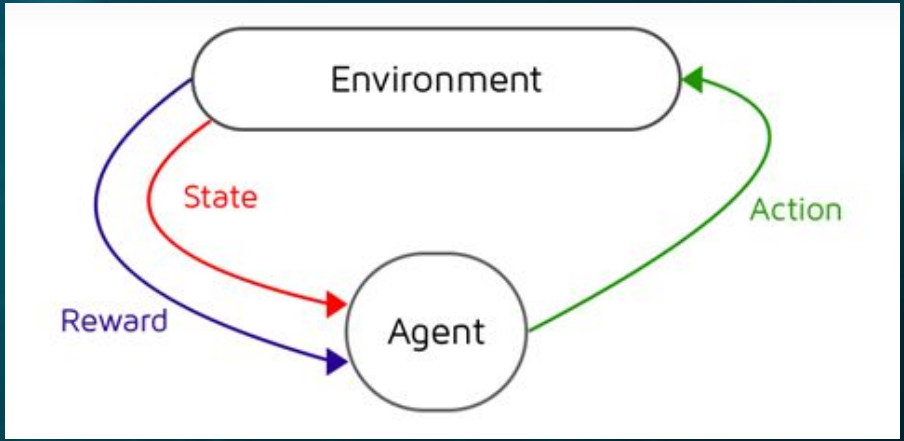
γ → Discount factor.
(Entre 0 y 1)

- Maximizar la función Return permite llegar a la recompensa más alta.

Policy function:

$$\pi(s) : S -> A$$

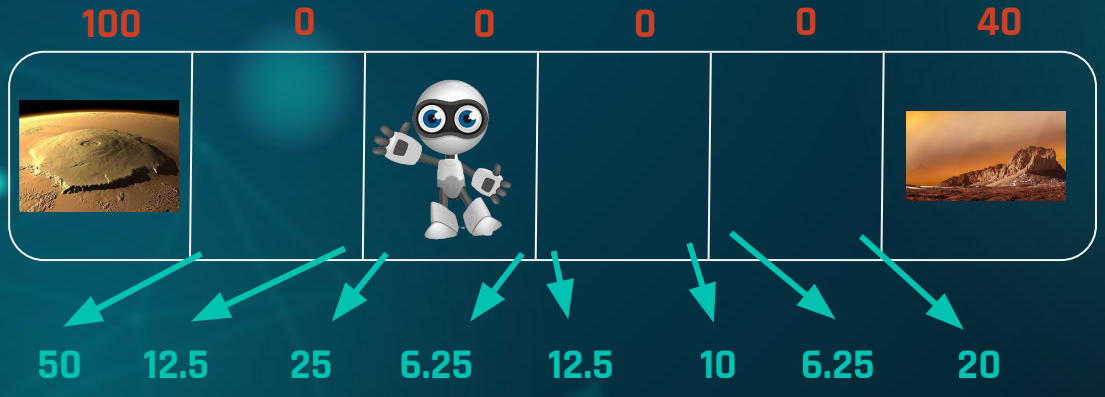
- Hallar la función que permita tomar una acción ***a*** en un estado ***s*** para que la función return se maximize.



Q-FUNCTION

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

El mejor Return



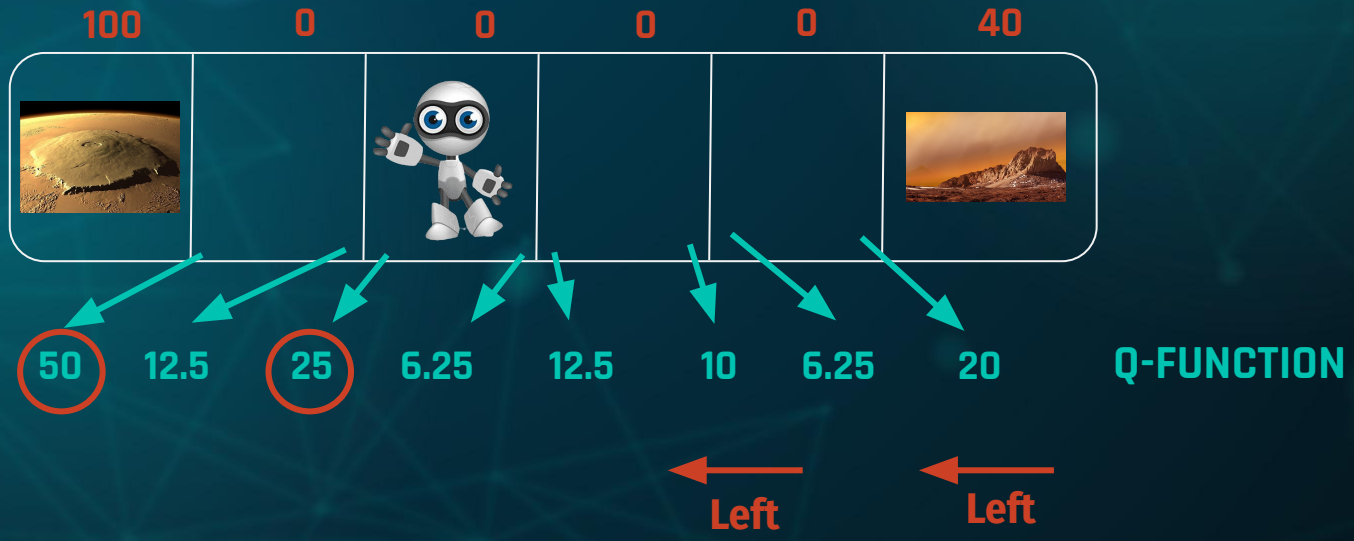
$\gamma = 0.5$

La acción más óptima en el estado s es la dada por el valor máximo de Q.

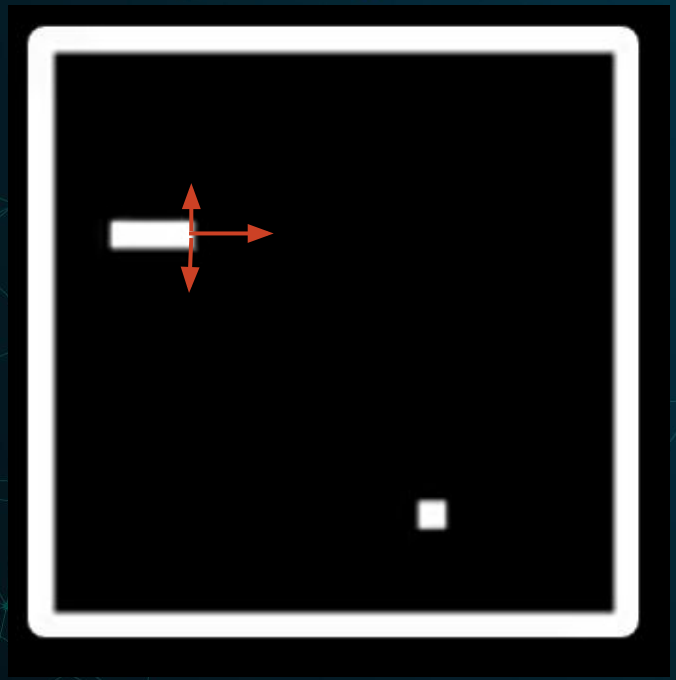
Policy function — **Q-function**

LA META PRINCIPAL DE LA FUNCIÓN POLICY

Hallar una función que permita saber qué acción **a** se debe tomar en el estado **s** de tal forma que el agente tome el camino más óptimo a su destino. (Proceso iterativo)



03 METODOLOGÍA

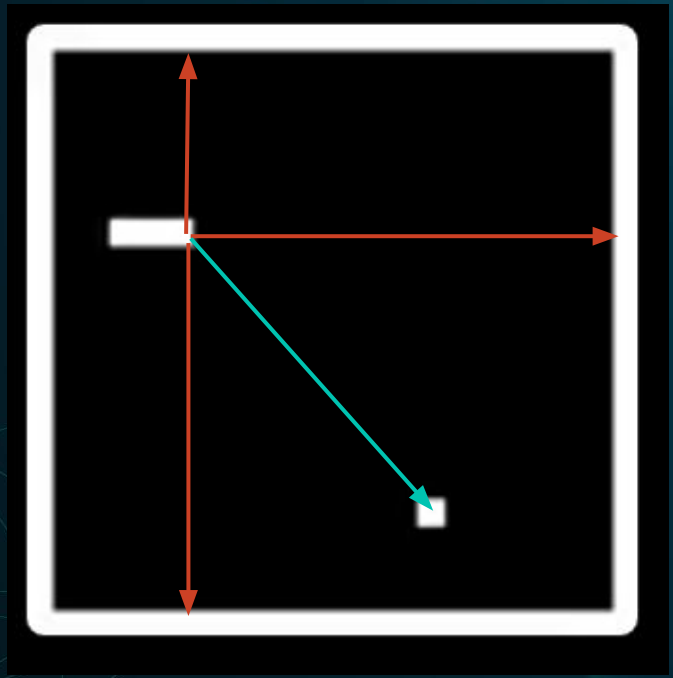


Entorno de la culebra.

Acciones:
Seguir derecho = 0
Derecha = 1
Izquierda = 2

Recompensas:
Comer alimento = + 10
Chocar = -10
De lo contrario = 0

ESTADO DE LA CULEBRA



Peligro seguir derecho	=	0
Peligro derecha		0
Peligro izquierda		0
Dirección izquierda		0
Dirección derecha		1
Dirección arriba		0
Dirección abajo		0
Comida izquierda		0
Comida derecha		1
Comida arriba		0
Comida abajo		1

Tamaño del estado = `(11,)`

ESTRUCTURA DE LAS REDES NEURONALES

Capa de entrada
Estados de la culebra
`(11,)`

Capa de salida
Valores de la función Q para ese estado de entrada

Capas ocultas
Ajustar los pesos ω

IIMPLEMENTACIÓN

Uso de **dos** redes neuronales con la misma estructura.

$$\underbrace{R + \gamma \max_{a'} \hat{Q}(s', a'; w^-) - Q(s, a; w)}_{\text{Error}}$$

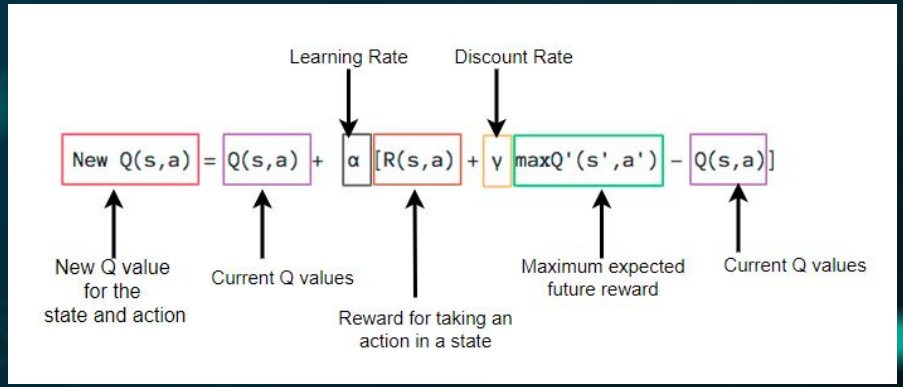
$$\underbrace{R + \gamma \max_{a'} \hat{Q}(s', a'; w^-)}_{\text{y target}}$$

RED NEURONAL 1
Con pesos ω'

RED NEURONAL 2
Con pesos ω

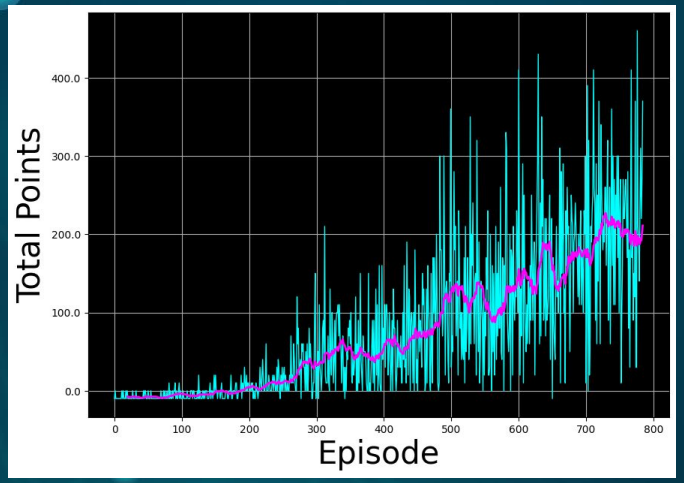
- Se inicializan los pesos aleatoriamente.
- Los pesos ω de la RN2 cambian constantemente aplicando gradiente descendiente.
- Cada C iteraciones:
  - Generar los *y targets* con la RN1.
  - Actualizar ω' con ω. (Soft update)

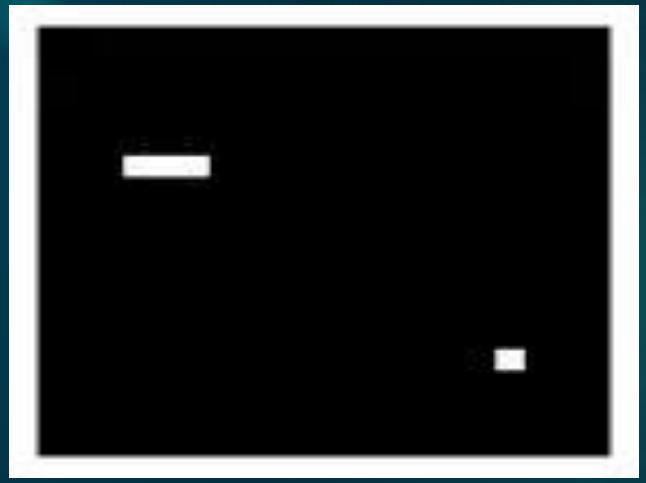
$$\omega' = \tau\,\omega + (1-\tau)\,\omega'$$



ALGORITMO

Initialize memory buffer D with capacity N
Initialize Q-Network with random weights w
Initialize target $\hat{Q}$-Network with weights $w^{-} = w$
for episode $i = 1$ ***to*** M **do**
    Receive initial observation state S_1
    for $t = 1$ ***to*** T **do**
        Observe state S_t and choose action A_t using an ϵ-greedy policy
        Take action A_t in the environment, receive reward R_t and next state S_{t+1}
        Store experience tuple (S_t, A_t, R_t, S_{t+1}) in memory buffer D
        Every C steps perform a learning update:
        Sample random mini-batch of experience tuples (S_j, A_j, R_j, S_{j+1}) from D
        Set $y_j = R_j$ if episode terminates at step $j + 1$, otherwise set $y_i = R_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a')$
        Perform a gradient descent step on $(y_j - Q(s_j, a_j; w))^2$ with respect to the Q-Network weights w
        Update the weights of the $\hat{Q}$-Network using a soft update
    end
end



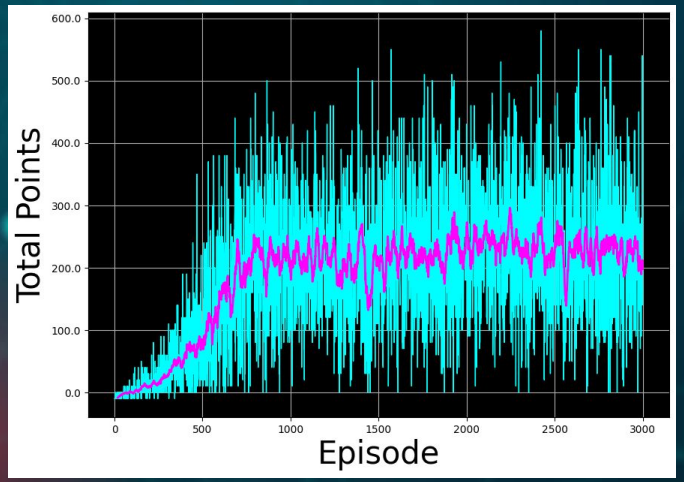


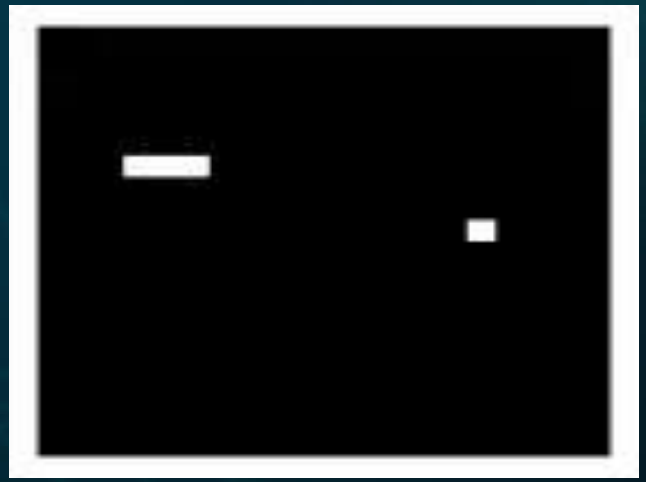
α = 0.001

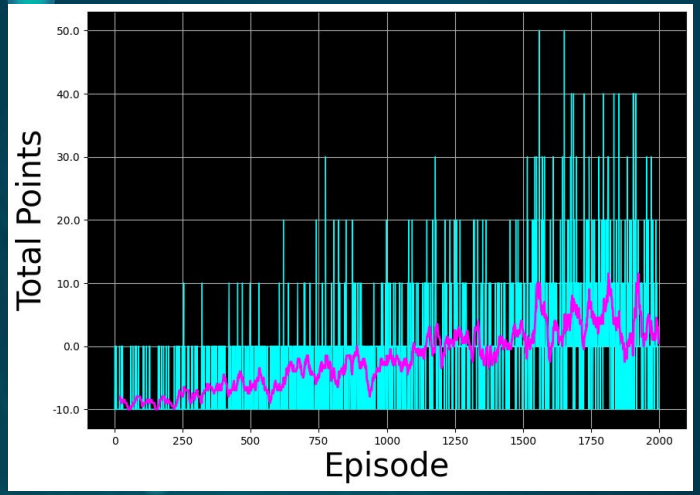
τ = 0.001

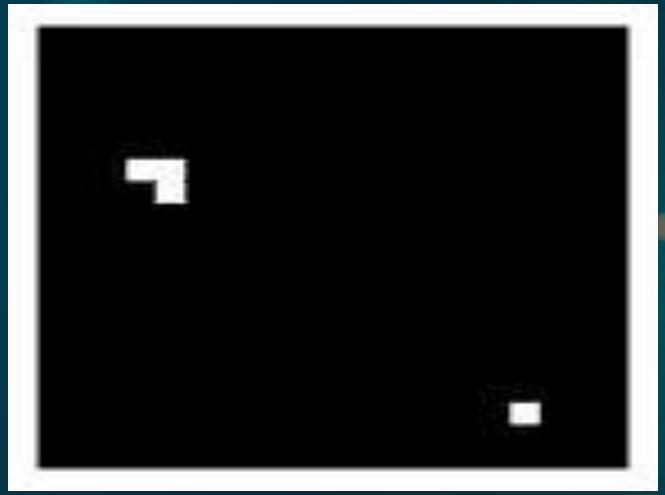
γ = 0.995

2 Capas ocultas con 64 neuronas.







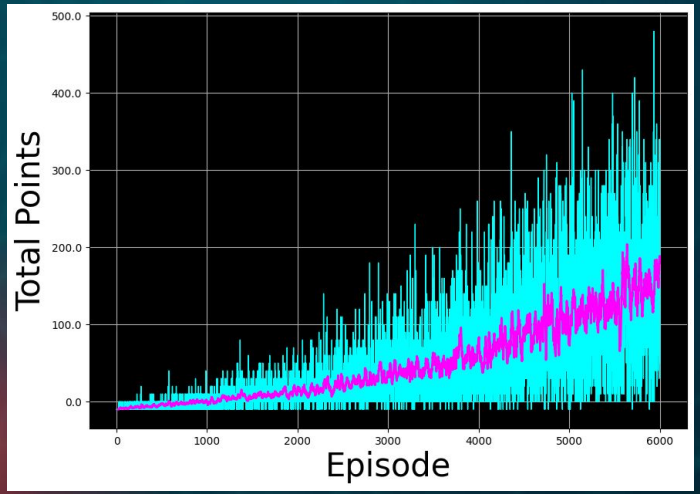


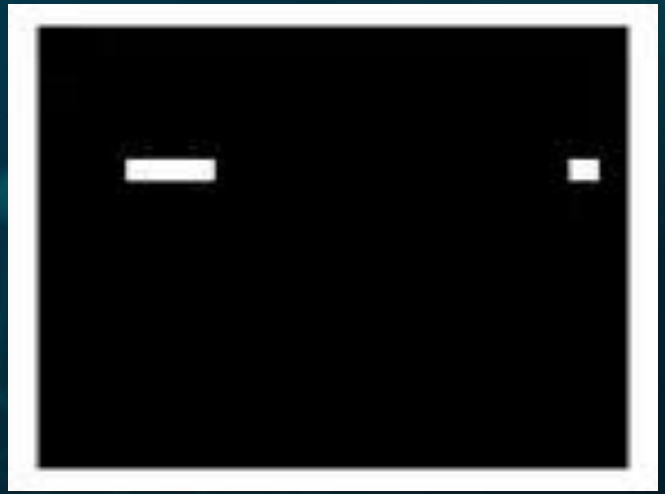
α = 0.0001

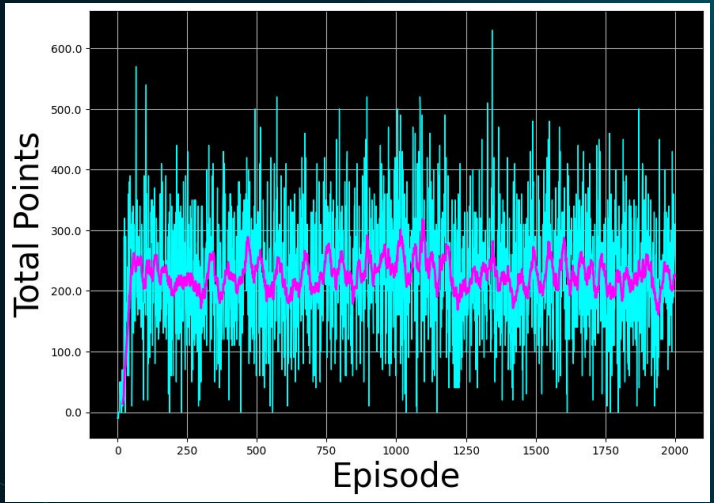
τ = 0.1

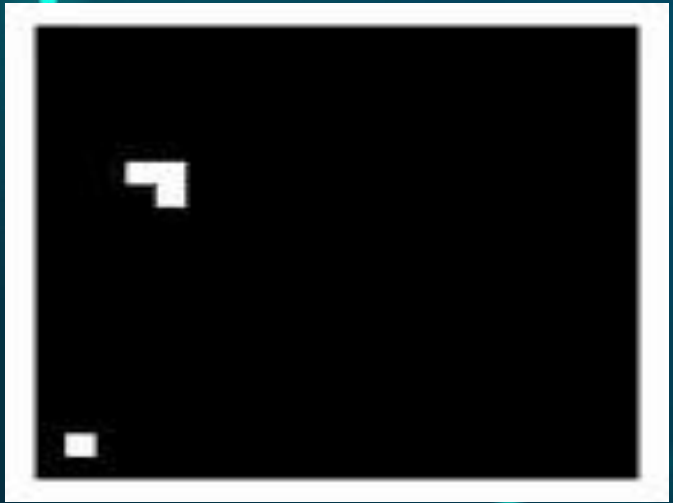
γ = 0.8

1 Capa oculta con 256 neuronas.







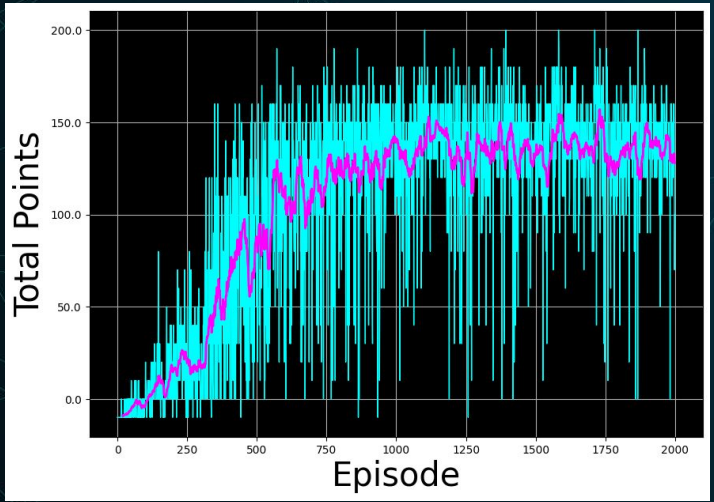


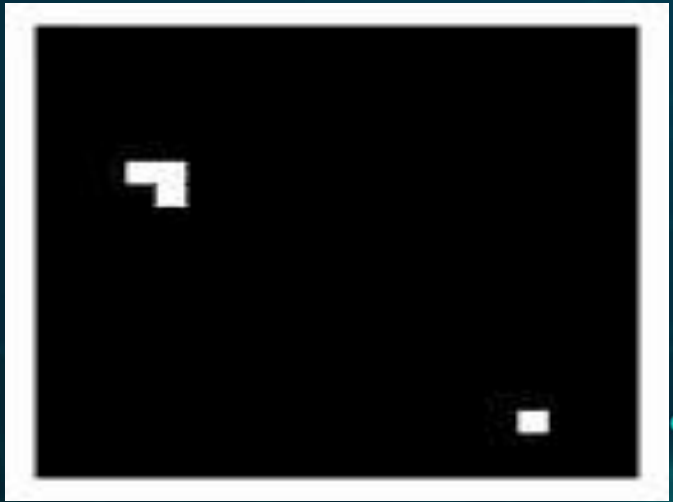
α = 0.001

τ = 0.01

γ = 0.9

2 Capas ocultas con 64 neuronas.





CONCLUSIONES Y PERSPECTIVAS

01	02	03	04
Se construyó una red neuronal que permitió que la culebra se jugara por sí sola.	El entorno, las recompensas, los estados y las acciones de la culebra fueron óptimos para jugar parte del juego.	Se pudo crear un entorno gráfico para la visualización del juego entrenado.	Con las redes entrenadas no se pudo completar el juego entero de la culebra.

- El reinforcement learning es un algoritmo que permite que un agente aprenda, pero es complejo saber las condiciones iniciales que debe tener el algoritmo para que haya muy buenos resultados.
- Se propone mejorar los parámetros con gráficas dependientes de las condiciones, y con las estructuras de las redes.
- Se propone usar algoritmos como los genéticos.

REFERENCIAS

- B Ravi Kiran , Ibrahim Sobh, *Deep Reinforcement Learning for Autonomous Driving: A Survey.*
- *https://neptune.ai/blog/reinforcement-learning-applications*
- *https://github.com/AleksaC/gym-snake*
- Curso de Reinforcement learning de Coursera:

  https://www.coursera.org/learn/unsupervised-learning-recommenders-reinforcement-learning?specialization=machine-learning-introduction

THANKS!