UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

IIMAS SEDE MÉRIDA

SEMESTRE:2022-1

Propuesta de IA para ganar el juego del “Snake”

Proyecto final para el curso de Inteligencia Artificial

por:

Gibrán Zazueta Cruz

02/diciembre/2021

Introducción

En el siguiente trabajo se plantea una solución de Inteligencia artificial para resolver el juego del “Snake”. Este juego apareció en las arcade en la década de los 70’s, desde entonces se han creado muchas versiones distintas del juego. En este trabajo se tiene el siguiente planteamiento:

Una serpiente intenta comerse tantas manzanas como sea posible. Cada vez que lo hace, su cuerpo crece (hacia la cola). La serpiente debe evitar salir del Mundo (NxN celdas) y no debe chocar contra su propio cuerpo.

Objetivo: Comer tantas manzanas como sea posible.

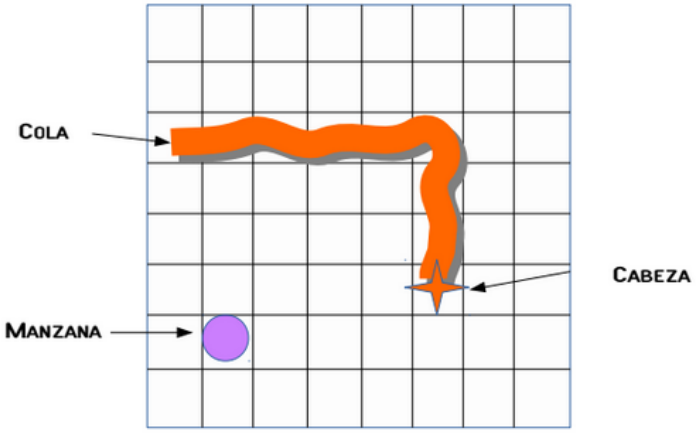

Figura 1. Juego de la serpiente

Al comerse una manzana esta aparece aleatoriamente en otra parte del Mundo.

Para moverse la serpiente puede moverse hacia *Arriba, Abajo, Izquierda, Derecha,* es decir, si intenta rotar hacia ella misma morirá inmediatamente.

Metodología

La resolución propuesta está basado en algoritmos genéticos

Se propone una población de 500 serpientes que será evaluada en el mundo virtual. Para darle a la serpiente la capacidad de decisión se utiliza una red neuronal cuya salida será un vector binario indicando moverse hacia alguna de las 4 direcciones posibles. Como “material genético” se utilizan las matrices de pesos de las serpientes.

A continuación se explica con mayor detalle las características de la solución.

Red neuronal

Capa de entrada

Como entrada a la red neuronal se construye un vector con la información de la dirección de la cola, la dirección de la cabeza y la visión de la serpiente hacia 8 lados, desde la cabeza. Por cada campo de visión la serpiente detecta la distancia hacia un muro, y si hay una manzana o su cuerpo en esa dirección.

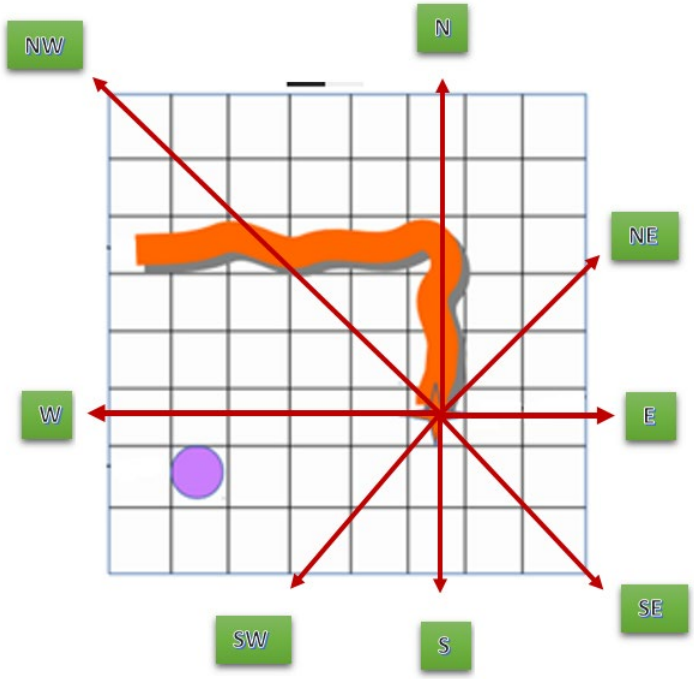

Figura 2. Sistema de visión

En la figura 2 se muestra el funcionamiento del sistema de visión. Para un caso como este se detectaría cuerpo en las direcciones N y NW Y distancia de 5 hacia el muro en la dirección W.

El vector se forma desde el norte hacia la izquierda, para el ejemplo los 3 sistemas de detección quedarían de la siguiente manera:

$$dist\_muro = [5, 5, 5, 2, 2, 2, 2, 2]$$

$$existe\_cuerpo = [1, 1, 0, 0, 0, 0, 0, 0\ ]$$

$$existe\_manzana = [0, 0, 0, 0, 0, 0, 0, 0\ ]$$

La idea de utilizar 8 direcciones es encontrar un equilibrio entre darle una buena capacidad a la serpiente de explorar el mundo contra no crear un vector de entradas de gran tamaño que aumente el tiempo de ejecución.

Como cada dirección da información de muro, manzana y cuerpo, con el sistema propuesto se suman un total suman 24 entradas a la red

Lo más adecuado es que la serpiente sea capaz de observar todo el mundo (que es el caso de un humano cuando juega el juego), sin embargo esto resultaría en un algoritmo muy tardado para escenarios de tamaño considerable.

En cuanto la dirección de la cabeza y de la cola, estas están codificadas en one-hot de la siguiente manera:

‘0001’→ movimiento hacia East

‘0010’→ movimiento hacia West

‘0100’→ movimiento hacia South

‘1000’→ movimiento hacia North

La idea de usar la cabeza y la cola es que, cuando el juego se pone complicado, normalmente es elemental observar estas dos direcciones para saber hacia dónde conviene moverse.

Considerando estas 8 entradas en total el vector de entrada a la red tiene una longitud de 32.

Capa oculta

La capa oculta consta de 2 capas de 20 y 12 neuronas, respectivamente. Estas son activadas mediante la función ReLu

Capa de salida

Finalmente la capa de salida entrega un vector de longitud 4. Este representa una codificación one-hot igual a la ya explicada.

Para la capa de salida se utilizó una función de activación sigmoidal

En la figura 3 se puede observa la red neuronal completa.

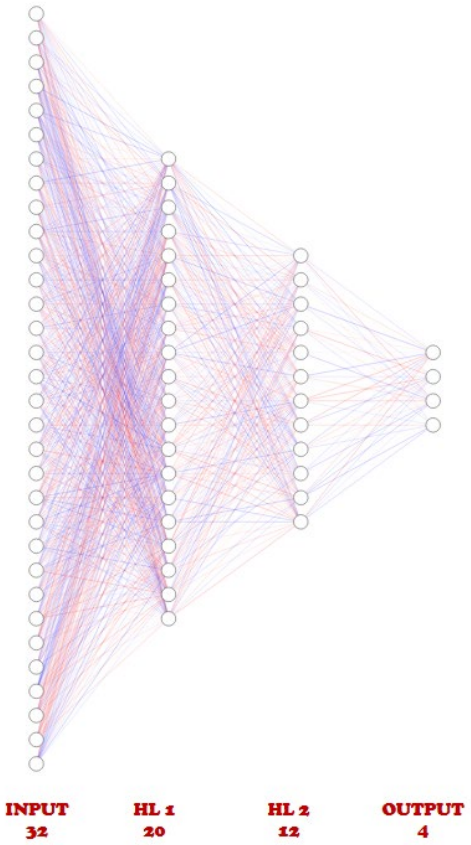

Figura3. Red neuronal propuesta

Algoritmo Genético

Para el algoritmo genético se propone una población de 500 serpientes. Los pesos de la red se generarán para cada una de manera aleatoria con un rango de 1 a -1.

En la evaluación del modelo se considera, cuantas manzanas comió la serpiente y cuantos pasos dio antes de morir. Esto es, como es obvio, para premiar a las serpientes que comen más manzanas, pero también a las que lo hacen más rápido.

$$Fitness = manzanas - steps/50$$

Para la selección se utilizó una selección tipo ruleta donde las serpientes con mayor puntuación de fitness tienen más probabilidad de ser elegidas.

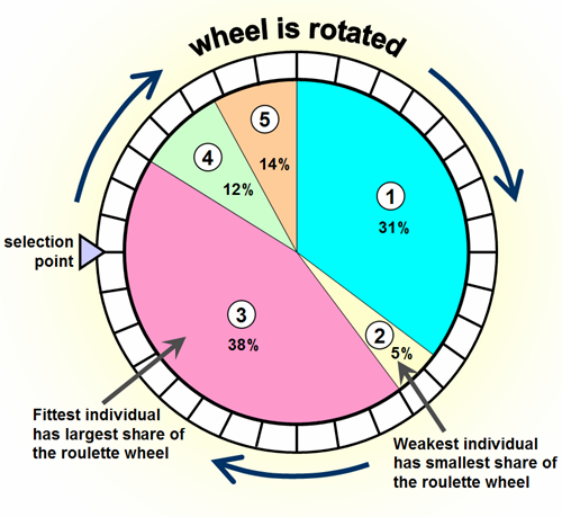

Figura 4. Imagen demostrativa método ruleta

Después de la selección se realizó una cruza en 1 punto. El punto de cruza se eligió al azar al igual que los dos padres de los dos hijos.

Finalmente se utilizó una mutación donde a cada peso de la matriz se le dio una probabilidad del 7% de mutar. Como los pesos van en valores muy pequeños la cantidad de mutación estuvo entre -0.2 y 0.2.

Resultados

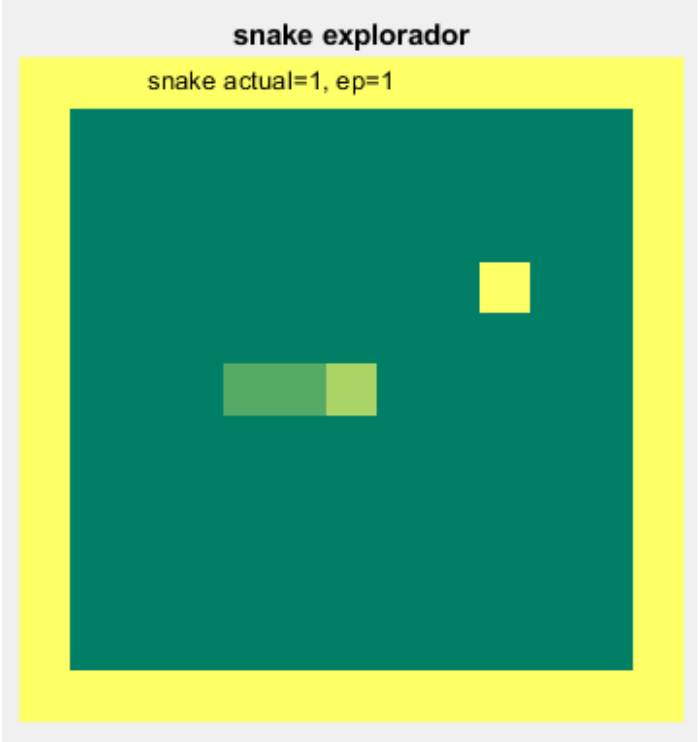

Figura 5. Interfaz del juego

Algunos comportamientos interesantes encontrados fueron:

- En las primeras generaciones la serpiente se quedaba quieta por lo que se consideró que si esta no realizaba ningún movimiento moría
- Se encontraron problemas con serpientes que se resistían a morir por lo que se limitó la cantidad de pasos, sin conseguir manzana, a 100. Después de estos la serpiente moría
- Como la serpiente puede morir al dar vuelta hacia la dirección de su cuerpo, una de las primeras cosas que aprendía era a evitar hacer esto.

Después de 700 épocas se tuvo un resultado de score máximo mostrado en la figura 6

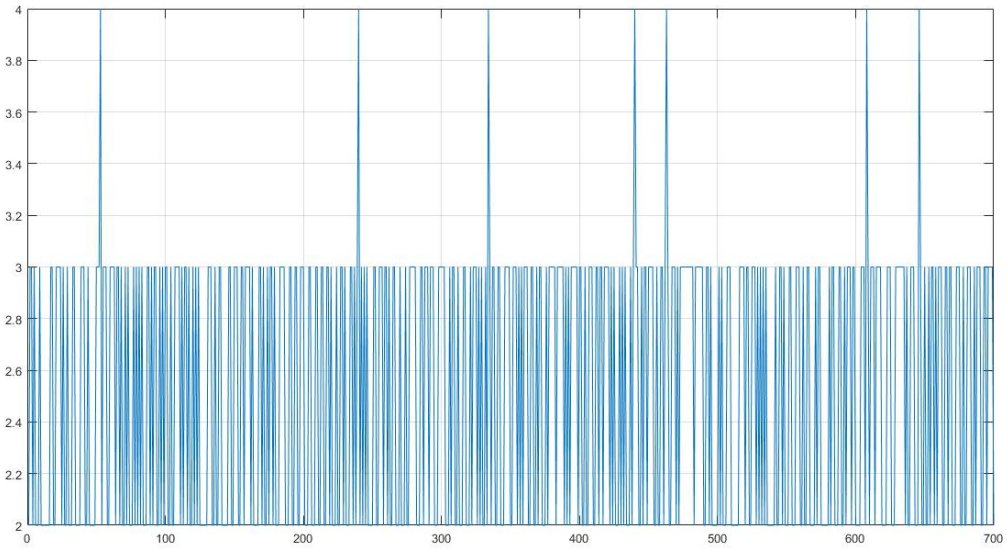

Figura 6. Resultado 700 generaciones

Donde el score máximo alcanzado es 4

Después Se tomó la matriz resultante y se volvió a entrenar por 700 generaciones. En esta ocasión se forzó al algoritmo a incluir el 10% de los mejores resultados en la

siguiente generación, sin embargo, esto no generó mejora.

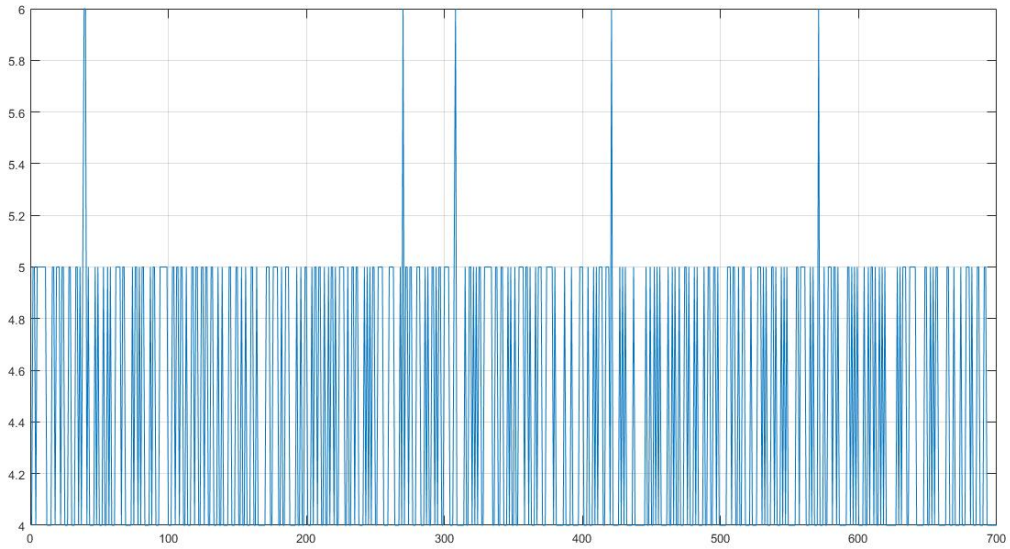

Figura 7. Resultado 2da vuelta

Conclusiones

Se considera que la falta de buenos resultados del algoritmo se puede deber a:

- Una mala elección de la función fitness
- Una mala elección en las funciones de activación de la red
- Falta de entrenamiento
- Errores en el código

Por otro lado, se logró implementar un buen reconocimiento del mundo que podría ser aprovechado para refinar esta técnica o buscar otro tipo de soluciones como aprendizaje reforzado.