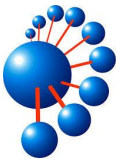


Proyecto Semestral: Flappy Bird

Inteligencia Artificial (2023-1)

Integrantes: Dazhi Enrique Feng Zong
Pablo Ignacio Zapata Schifferli
Profesor: Julio Godoy
Ayudante: Felipe Cerda
Fecha: 16 de julio, 2023

Introducción

En este proyecto, se resuelve el problema de desarrollar una IA para el popular juego Flappy Bird utilizando técnicas de aprendizaje por refuerzo, en las que se incluyen: Q-Learning, SARSA, red neuronal y NEAT.

El juego Flappy Bird, lanzado en 2013, se ha convertido en un desafío popular debido a su simplicidad y adictividad. El objetivo del jugador es guiar a un pájaro a través de una serie de tuberías sin chocar con ellas. La dificultad radica en el control preciso del vuelo del pájaro, ya que solo puede volar hacia arriba o caer por gravedad.

Las propiedades del ambiente del juego son: parcialmente observable (se puede ver solo 2 o 3 tuberías máximo), estocástico (las alturas de las tuberías son generadas de manera aleatoria), secuencial (es un juego que depende de la acción anterior), estático (para la IA el ambiente no cambia cuando está “pensando”), discreto (número limitado de acciones y tuberías observables) y un solo agente (el pájaro).

En este informe, se exploran tres enfoques principales para desarrollar una IA de Flappy Bird: Q-learning/SARSA, red neuronal con algoritmo genético y NEAT (NeuroEvolution of Augmenting Topologies).

Q-learning y SARSA son algoritmos de aprendizaje por refuerzo que permiten al agente de IA aprender a través de la interacción con el entorno. Estos algoritmos buscan maximizar una función de recompensa a largo plazo, lo que implica tomar decisiones óptimas en cada paso del juego. Por otro lado, las redes neuronales permiten mejorar las habilidades del agente mediante la evolución y selección de modelos neurales. Los algoritmos genéticos se basan en principios biológicos de selección natural y reproducción, permitiendo la creación de modelos neuronales más eficientes y adaptados a la tarea específica del juego. Además, se incorpora NEAT, que combina redes neuronales con algoritmos genéticos para evolucionar topologías de redes neuronales. NEAT permite la creación y modificación automática de la estructura de la red neuronal, lo que facilita la adaptación y mejora del rendimiento de la IA.

Revisión bibliográfica

Se revisaron algunos repositorios principalmente para obtener los parámetros de algunas constantes (como el factor de aprendizaje, factor de descuento, epsilon en caso de Q-Learning o las configuraciones en NEAT), más que ver cómo desarrollar la IA en sí.

- https://arxiv.org/pdf/2003.09579.pdf
- https://github.com/anthonyli358/FlapPyBird-Reinforcement-Learning
- https://github.com/techwithtim/NEAT-Flappy-Bird

Métodos propuestos para resolver el problema

- **Q-learning/SARSA:** Se guarda el estado actual del juego, para decidir si saltar o no saltar como próxima acción.
- **Red neuronal con algoritmo genético:** Entrenar una red neuronal usando un algoritmo genético.
- **NEAT (NeuroEvolution of Augmenting Topologies):** Usar NEAT para comparación de métodos.

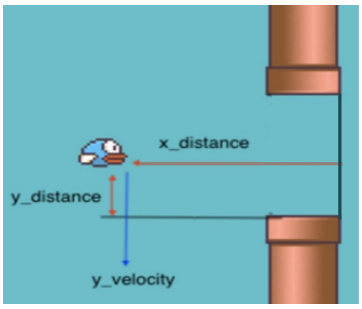
Implementación

En azul lo que ya estaba hecho.

Q-Learning/SARSA

Función para obtener el estado. El estado se compone de:

- ydiff: altura del agente con respecto al tubo inferior más cercano.
- xdiff: distancia horizontal del agente a la tubería de la derecha.
- vel: velocidad del agente, cayendo o acaba de saltar.
- ydiff1: altura del agente con respecto al segundo tubo inferior.



Se divide por 10 ydiff, xdiff y ydiff1 para no tener tantos valores (y estados). Si la segunda tubería está muy lejos del pájaro, se tomará ydiff1 como 0, reduciendo aún más la cantidad de estados.

```
function getState(y, pipe, vel)
      # y es altura del pájaro, pipe es arreglo con las pipes y vel
        es la velocidad
      xdiff = pipe[0]["x"]
      ydiff = pipe[0]["y"] - y
      if xdiff cerca del pipe[1] then
            ydiff1 = pipe[1]["y"] - y
      else
            ydiff1 = 0
      end if
      ydiff = round(ydiff // 10)
      xdiff = round(xdiff // 10)
      ydiff1 = round(y1 // 10)
      return (ydiff, xdiff, vel, ydiff1)
end function
```

Función egreedy modificado que recibe el estado y devuelve la acción a tomar. Si para un estado, solo tiene un valor Q (sólo hay registrado 1 acción) y es negativo se toma la acción contraria a la única registrada (para explorar).

```
function egreedy(state)
      if random() < e then
            A = elección aleatoria entre [True, False]
      else
            if sólo hay 1 valor Q (negativo) para ese estado then
                  A = tomar la otra acción no registrada
            else
                  A = tomar la acción con Q mayor
            end if
      end if
      return A
end function
```

Funciones con las fórmulas de SARSA y Q-Learning.

```
Q = diccionario para almacenar Q values
a = 0.8  # tasa de aprendizaje
g = 1 # factor de descuento
e = 0.1 # epsilon
function sarsa(state, prevAction, nextAction, R, newState)
      currentQ = Obtener valor Q con state y prevAction
      nextQ = Obtener valor Q con newState y nextAction
      newQ = currentQ + a * (R + g * nextQ - currentQ)
      Q[(state, prevAction)] = newQ
end function

function qLearning(state, prevAction, R, newState)
     currentQ = Obtener valor Q con state y prevAction
     maxQ = Obtener el valor Q máximo de newState
     newQ = currentQ + a * (R + g * maxQ - currentQ)
     Q[(state, prevAction)] = newQ
end function
```

Función que contiene el juego principal. Como en los 2 algoritmos hay partes que son iguales, se decidió dejar ambos algoritmos en una función, pero agregando ifs para las partes que son exclusivas para cada algoritmo.

Los pasos del algoritmo son los siguientes:

1. Inicialización de parámetros.
    - Para SARSA también se escoge la acción con egreedy.
2. Luego se ejecuta el bucle while infinito del juego.
    - Para Q-Learning, inmediatamente después de iniciar el bucle while se escoge la acción.
3. Se ejecuta la acción escogida.
4. Se obtiene el estado y las recompensas. +0.5 por estar a la altura del gap de las tuberías -0.1 sino.
5. Si el pájaro choca, -1000 de recompensa. Se guardan 2 Q Values, uno con el estado anterior y otro con el ante anterior. Luego se reinicia el juego.
6. Si pasa por una tubería, logra un punto y recompensa de +5.
7. Se guarda el Q value con las variables necesarias.
    - Para SARSA, se escoge la próxima acción antes de guardar el Q value.
8. Se actualizan las variables, luego de que se haya almacenado el Q value.

Además, para la elección de acciones, sólo se usará egreedy cuando el pájaro está cerca de las tuberías. Sino, se escoge de manera que el pájaro esté centrado (no caiga o suba demasiado). Estos estados en el que el pájaro está lejos de las tuberías y la acción no es escogida por egreedy, no se guardan en la tabla de Q values.

```
function mainGame()
      saltar = False # acción a tomar
      r = 0 # reward
      state = getState(playery, lowerPipes, playerVelY)
      # Inicializar SARSA
      if SARSA then
            prevAction = egreedy(state)
      end if
      while true do
            if QLEARNING then
                  # elegir acción en q-learning
                  if pájaro lejos o sobre tubería then
                        prevAction = acción para que no caiga al
            suelo
                                          o suba demasiado
                  else
                        prevAction = egreedy(state)
                  end if
            end if
            # acá se hace el salto con la acción escogida
            saltar = prevAction
            newState = getState(playery, lowerPipes, playerVelY)
```

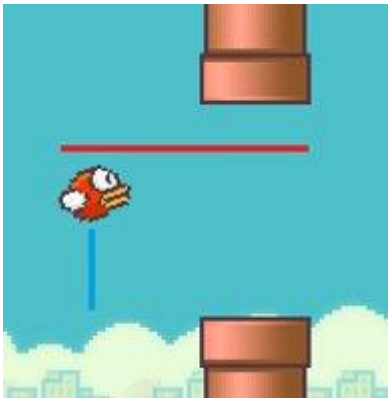
```
            # si el pájaro está en una altura cercana al gap
            if altura del pájaro casi igual a altura del gap then
                  r += 0.5
            else r -= 0.1
            end if
            # el juego verifica si el pájaro choca
            if pajaro crash then
                  r = -1000
                  # se guardan con 2 estados anteriores
                  if SARSA then
                        sarsa(state, prevAction, nextAction, r,
                        newState)
                        sarsa(prevState, prevprevact, prevAction, r,
                         state)
                  end if
                  if QLEARNING then
                        qLearning(state, prevAction, r, newState)
                        qLearning(prevState, prevprevact, r, state)
                  end if
                  mainGame() # reiniciar juego
            endif
            if pájaro pasa por tubería then r = 5 end if
            if SARSA then
                  # elegir próxima acción para SARSA
                  if pájaro lejos o sobre tubería then
                        nextAction = acción para que no caiga al
            suelo
                                          o suba demasiado
                  else
                        nextAction = egreedy(state)
                        # si el pájaro está cerca de la tubería, el
                        estado es válido y se almacena
                        sarsa(state, prevAction, nextAction, r,
                        newState)
                  end if
            end if
            # si el pájaro está cerca de la tubería, el estado es
            válido y se almacena
            if QLEARNING y cerca de las tuberías then
                  qLearning(state, prevAction, r, newState)
            end if
            prevState= state
            prevprevact = prevAction
            state = newState
            if SARSA then prevAction = nextAction end if
      end while
end function
```

Red neuronal

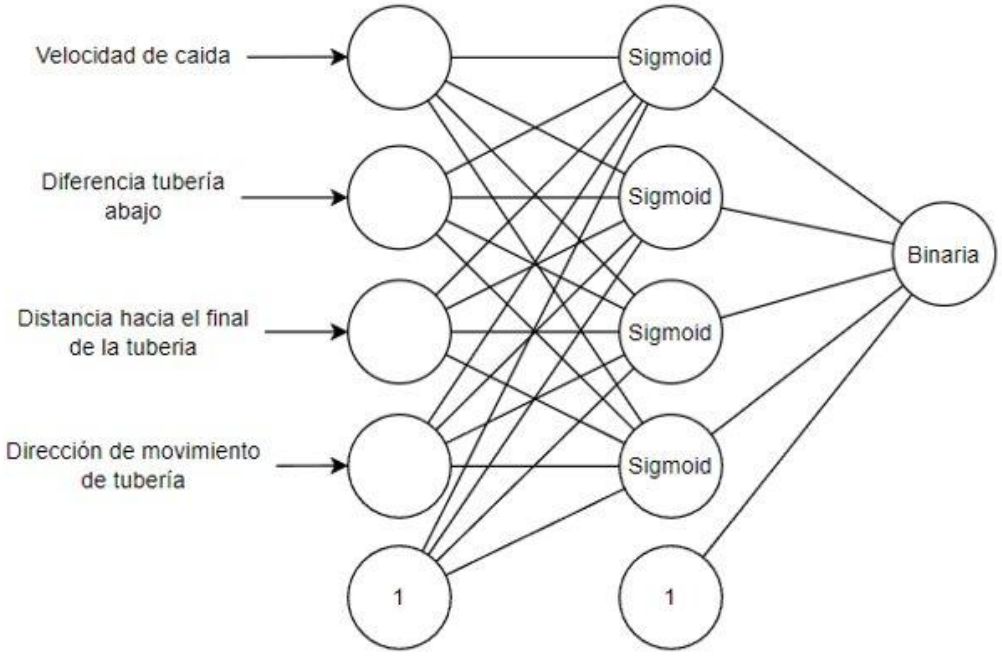
La red neuronal es almacenada en arreglos de numpy. El input está normalizado y el cálculo del valor de cada nodo es mediante producto punto, de numpy también, entre la capa anterior y los pesos asociados a esta, para luego pasar el valor a la función respectiva. La función binaria de la capa de salida entrega un 1 si el valor es mayor a 1, 0 en caso contrario.

Las coordenadas del pájaro y tuberías están en sus esquinas superiores izquierdas. El eje y aumenta hacia abajo:

- Velocidad de caída: Del pájaro
- Diferencia tubería abajo: tuberiaAbajo.y - (pajaro.y + pajaro.alto)
- Distancia hacia el final de la tubería: (tuberiaAbajo.x + tuberia.ancho) - pajaro.x
- Dirección de movimiento de la tubería: -1 si se mueve hacia arriba, 1 hacia abajo.
    - Este nodo está desactivado en el Flappy bird normal.



En rojo, distancia hacia el final de la tubería
En azul, diferencia tubería abajo.



Algoritmo genético

Se crea la población, 1000 en este caso, con pesos de manera aleatoria entre -1 y 1 y fitness 0.

While(true):

1. Se ejecuta el Flappy Bird para calcular el puntaje de cada individuo y su distancia de choque hacia la apertura de la tubería, si choca.
2. Se actualiza el fitness de cada individuo de la siguiente manera:
    - fitness = (fitness + puntaje)/2
3. Se ordenan a los individuos basados en su fitness, individuos con igual fitness son ordenados de acuerdo a su distancia de choque, menor distancia es mejor.
4. Se reemplaza al 10% peor de la población:
    - Padre 1: Aleatorio de los 10 mejores.
    - Padre 2: Aleatorio del 10% mejor, distinto del padre 1.
    - Cada peso del hijo se reemplaza por el peso de alguno de sus padres. Su fitness también es elegido de esta manera.
5. Cada individuo, menos el top 10, puede mutar con probabilidad 0.2:
    - Cada peso del individuo tiene un 0.2 de probabilidad de ser sumado con un valor aleatorio entre [-0.02, 0.02]

NEAT

Se usó la biblioteca neat-python. A diferencia del algoritmo genético, el fitness de cada genoma se actualiza de la siguiente manera:

- fitness = puntaje

La red usada es Feedforward, con funciones de activación tanh. Debido a la gran cantidad de opciones de configuración, es muy probable que estas no sean óptimas.

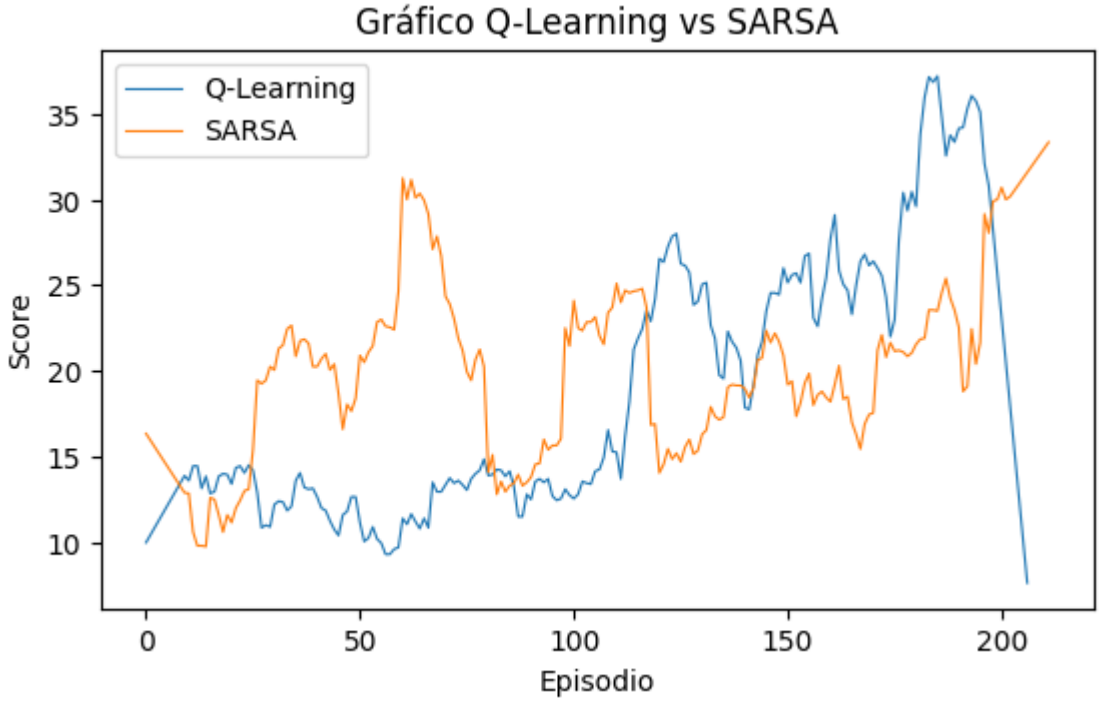
La mayoría de las configuraciones de NEAT (*neat_config.ini*) se obtuvieron de otro código. Aunque se realizaron modificaciones y pruebas en algunas de ellas, una gran parte se mantiene sin cambios.

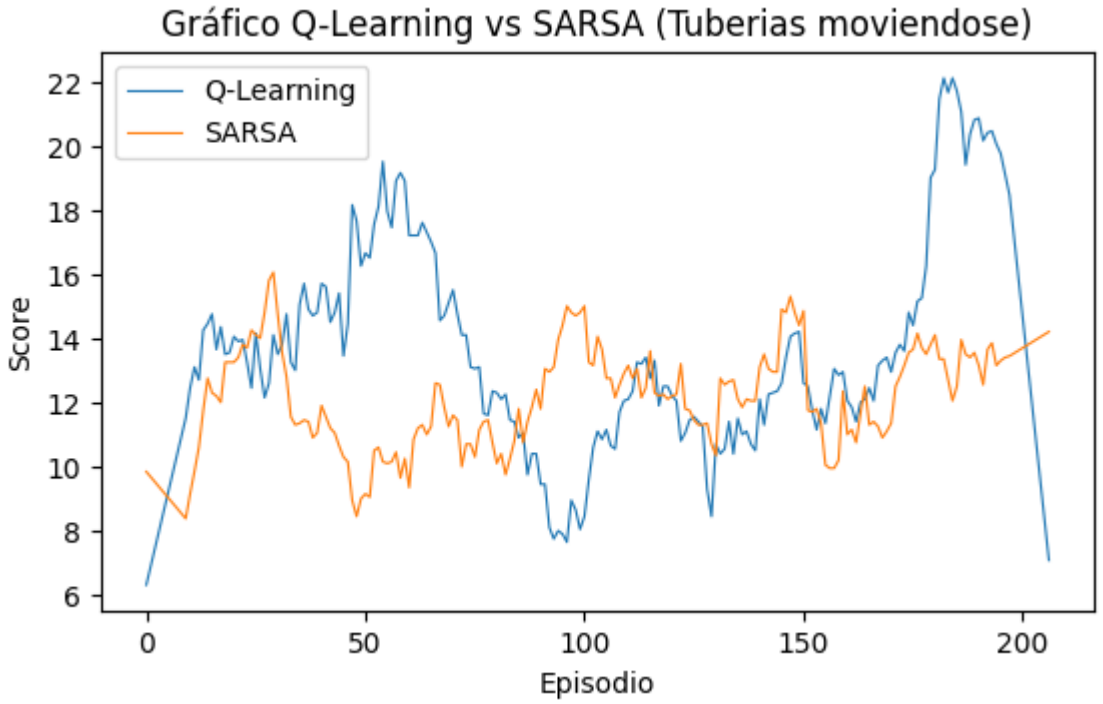
Experimentación

Debido al caso fatal que el agente no puede superar de manera consistente (a veces sí y otras veces no), el puntaje depende en gran medida de si se presenta este caso. Por lo tanto, no siempre se observa un aumento progresivo de los puntajes, ya que dependen de la aparición aleatoria de este caso. Esto genera un patrón en el que los puntajes pueden aumentar durante ciertos períodos de tiempo, lo que refleja el aprendizaje y la mejora del agente, pero luego disminuyen debido a la presencia del caso fatal, el cual es la causa de la naturaleza ascendente y descendente de las curvas en el gráfico. El caso fatal se explicará más adelante en las Dificultades.

Q-Learning/SARSA

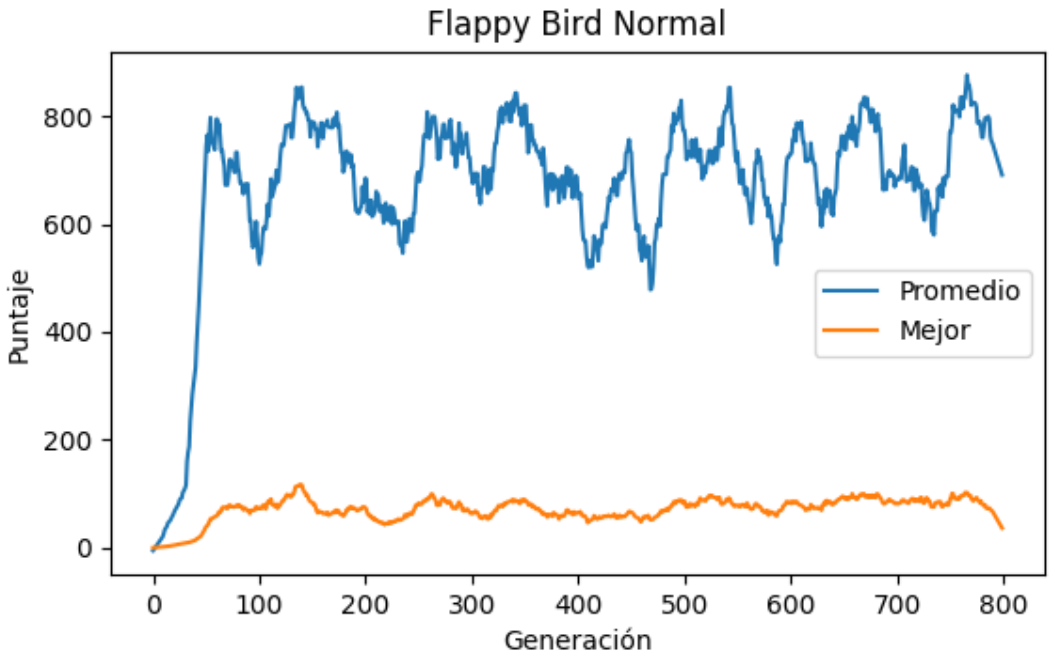
Ignorando las caídas, se puede apreciar que a mayor episodio, mayor es el puntaje máximo, es decir, el agente está aprendiendo y se puede ver más claramente en la curva del algoritmo Q-Learning. Pero, también se puede ver la presencia del caso fatal, en el cual, después de alcanzar un puntaje alto, el agente vuelve a obtener un puntaje bajo.

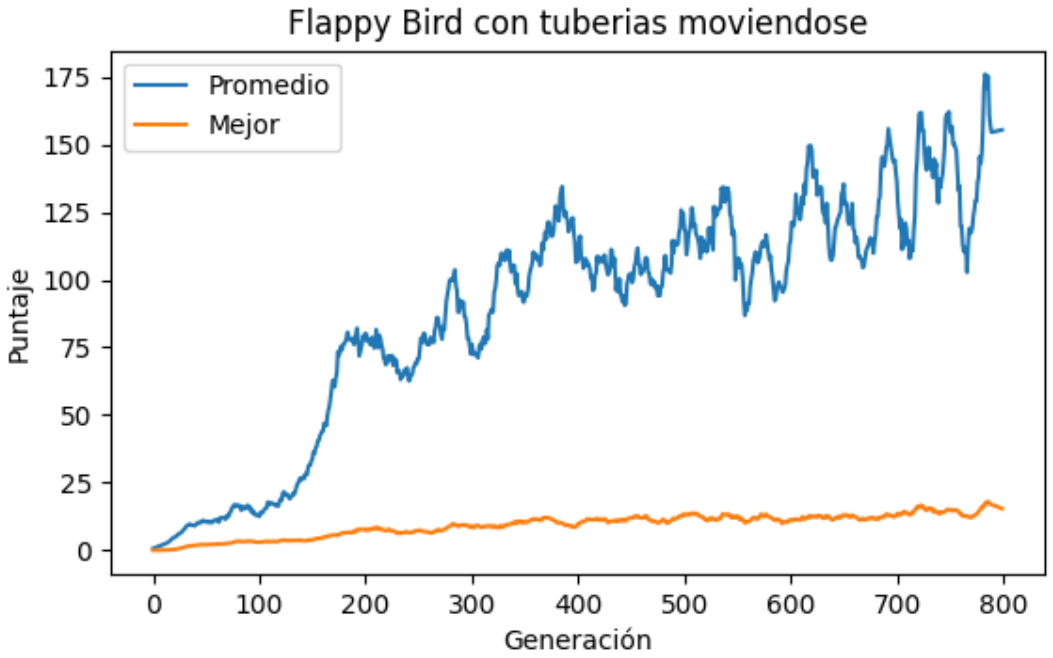




Red neuronal + Algoritmo genético

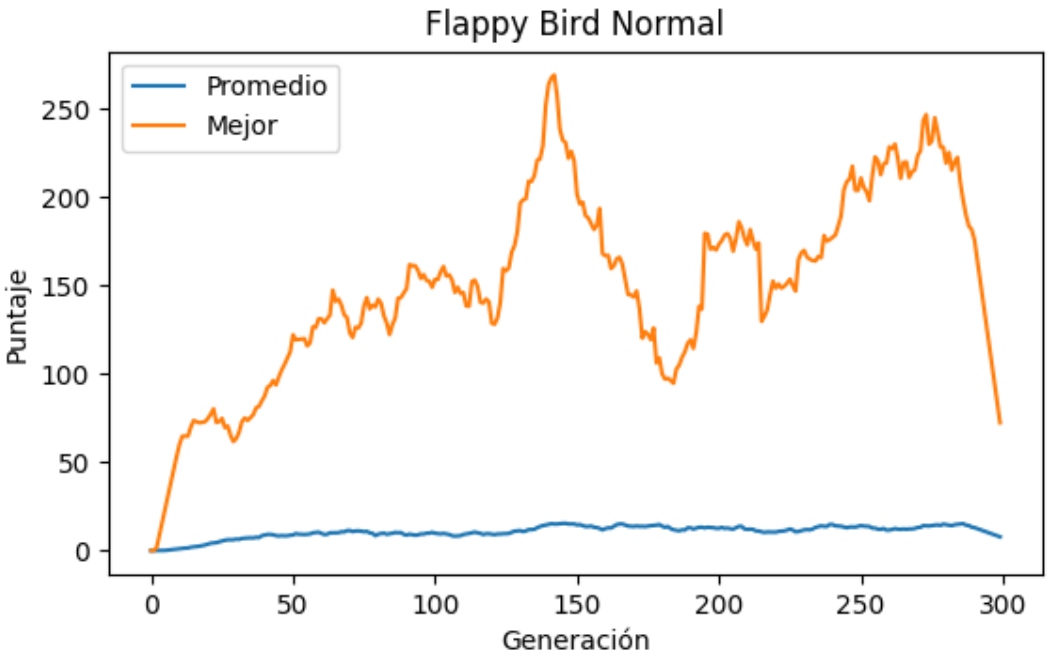
Hay un límite de 1000 tuberías por temas de tiempo. Ambos casos tienen un problema, si un individuo logra un puntaje alto en una generación, no hay garantías que en la siguiente logre otro puntaje alto. En relación a las tuberías moviéndose, si bien hay un peor rendimiento que el Flappy Bird normal, por ser un ambiente con mayor dificultad, se logra apreciar un constante incremento, con suficientes generaciones tal vez pueda alcanzar al rendimiento del Flappy Bird normal.

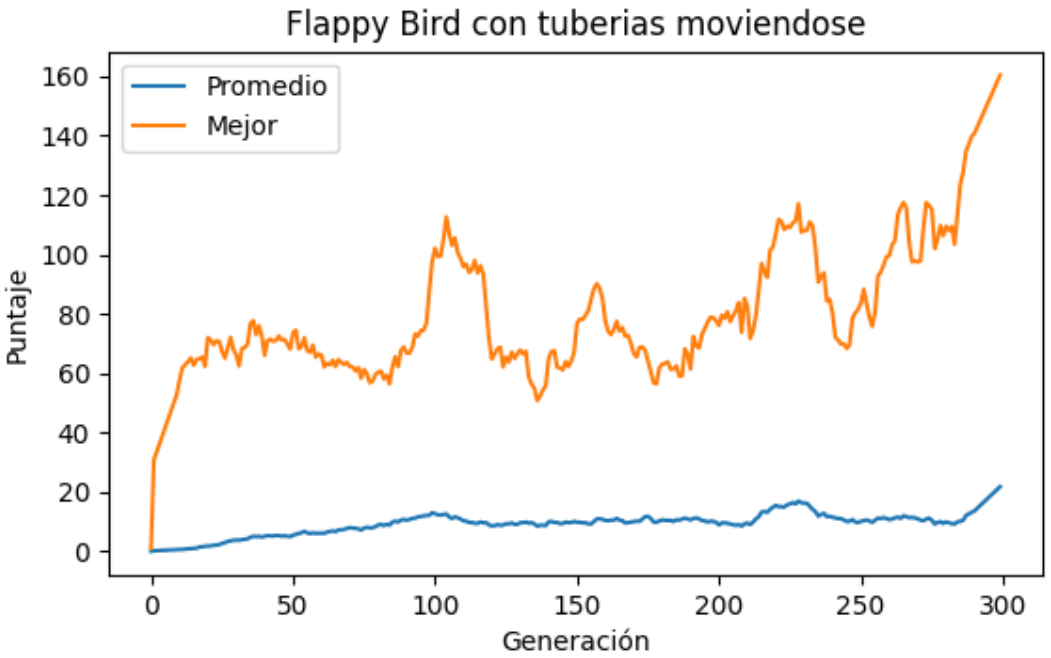




NEAT

Debido a la gran cantidad de opciones de configuración de NEAT, es muy probable que estas no sean óptimas para el problema, logrando un peor rendimiento en relación al algoritmo genético. Aun así, NEAT logra una ventaja en las primeras generaciones, pudiendo alcanzar puntajes de 50 en las primeras 5 generaciones donde el algoritmo genético está con puntajes de 1 o 2. Esta ventaja se acentúa aún más en el escenario con tuberías moviéndose, donde inclusive el rendimiento general es mejor al del algoritmo genético.





Dificultades

- Para Q-Learning/SARSA: Hay demasiados posibles estados: 512*288*20*512*2 (posibles valores para ydiff, xdiff, vel, ydiff1 y action). El agente se demoraría demasiado en aprender, además de que muchos de estos estados no son relevantes para el juego.
- **Solución:** Para solucionar este problema, es necesario reducir la cantidad de estados considerados. Logramos esto mediante diferentes estrategias: primero, dividimos los valores de ydiff, xdiff e ydiff1 por 10, ya que no importa si el agente se encuentra en el píxel 31 o 32, sino que basta con saber que está en el píxel 30. Además, eliminamos los estados "ruido", es decir, aquellos en los que el agente se encuentra muy lejos de las tuberías. Solo tomamos en cuenta los estados cercanos a las tuberías, ya que son los relevantes para el juego. Asimismo, para simplificar el proceso, no aplicamos Q-Learning o SARSA a los estados lejanos de las tuberías, y limitamos los movimientos del agente. Si el agente se encuentra demasiado por encima de una tubería, la próxima acción que se elige es saltar sin excepción.
- El tiempo de juego es significante, por ejemplo, llegar a la tubería 1000 toma alrededor de 6 minutos. Probar el algoritmo genético en este estado toma bastante tiempo, es por esto que se tuvo que sacar el limitador de FPS del juego para que corriera lo más rápido posible.
- Caso fatal: Situación poco común donde la tubería actual es considerablemente más alta que la siguiente tubería y la posición y aceleración del pájaro juegan en contra resultando en un choque inminente. En este escenario, la IA tiende a saltar antes de ingresar a la tubería para evitar chocar. Sin embargo, al salir de la tubería actual, se ve obligada a saltar nuevamente para evitar chocar. Este segundo salto provoca que la IA choque con la siguiente tubería, que se encuentra a una altura considerablemente más baja. El problema es que la IA no puede reducir su velocidad antes de caer y pasar por debajo de la tubería inferior, lo que resulta en la colisión con la tubería superior debido a la incapacidad de desacelerar. Para solucionarlo, la IA debe aprender a saltar en el momento preciso, de manera que pueda descender de manera adecuada y pasar por debajo de la tubería inferior sin colisionar con la tubería superior. Sin embargo, este logro no siempre es consistente, ya que a veces la IA logra realizar el salto en el momento adecuado y la mayoría de las veces no lo consigue.
    - Nos gustaría pensar que es evitable, sin embargo al buscar otras implementaciones, estas tenían mayor aceleración de caída y/o distancias casi el doble de grande entre tuberías en relación a nuestro Flappy Bird.

Conclusiones

Se logró que la IA aprendiera a jugar flappy bird con todos los métodos propuestos, Q-Learning, SARSA, una red neuronal entrenada con un algoritmo genético y NEAT. Sin embargo, el rendimiento de cada uno es muy diferente.

En el caso de Q-Learning y SARSA, la cantidad de estados posibles era importante. Con el objetivo de acelerar el proceso de aprendizaje, se realizó una reducción significativa en la cantidad de estados posibles. Sin embargo, esta simplificación conlleva el riesgo de perder precisión en la representación del estado, lo que resultó en que estos métodos obtuvieron el peor rendimiento. Además estos métodos junto con la reducción de estados deja al agente muy susceptible al caso fatal.

En relación a la red neuronal entrenada con un algoritmo genético, este método obtuvo el mejor puntaje en el Flappy Bird normal y estuvo a la par con NEAT en el Flappy Bird normal con tuberías moviéndose. Sin embargo, este método obtiene malos resultados en las primeras generaciones y como depende del azar, ejecutar el algoritmo genético múltiples veces obtendrá múltiples resultados en relación a la convergencia.

En NEAT, si bien la configuración de este es muy probablemente subóptima, se obtuvieron resultados decentes en Flappy Bird normal y, como se mencionó anteriormente, resultados a la par con el algoritmo genético en Flappy Bird con tuberías moviéndose. Cabe destacar que NEAT aprende muy rápido en las primeras generaciones a diferencia de los otros métodos, sin embargo, este aprendizaje se limita en generaciones avanzadas.

Finalmente, ningún método pudo evitar por completo el caso fatal. Tanto en el algoritmo genético como NEAT se probó pasando como input los datos de la siguiente tubería para que la IA aprendiera a posicionarse mejor, sin embargo, no hubo diferencias visibles en el aprendizaje.