Solución de Mario Bros y Air Striker para TEC-Games

J. Alvarado, *estudiante de Ing. en Computación, ITCR,* y B. Badilla, *estudiante de Ing. en Computación, ITCR*

***Resumen*—Anteriormente en la carrera se nos pidió realizar juegos utilizando la programación orientada a objetos. Sin embargo, ahora se nos reta a resolver juegos. Este cambio lo consideramos extremadamente interesante ya que dice mucho del nivel esperado como programadores. Durante este trabajo realizamos dos soluciones para dos juegos diferentes: Mario Bros (World) y Air Striker, utilizando un algoritmo genético y uno probabilístico respectivamente. A lo largo de esta investigación logramos también comprobar que estos algoritmos son más eficientes con respecto al costo, que es la razón principal para el que la empresa pidiera estos algoritmos.**

I. Introducción

En la actualidad los algoritmos genéticos y probabilísticos no son usados tan a menudo debido al tiempo que se requiere para correrlos. Sin embargo, su bajo costo demuestra lo valiosos que son en la industria. En este trabajo vamos a estudiar a fondo ambos algoritmos en los juegos de Mario Bros y Air Striker. Con respecto al juego de Mario Bros nos dimos cuenta que un algoritmo genético normal como los que vimos en clase no iba a fucionar para este juego, ya que son demasiadas variables a tomar en cuenta en el nivel 1. Es debido a esto que tuvimos que implementar una red neuronal llamada NEAT (Neuroevolution of augmenting topologies) la cual, por medio de la librería neat-python, va a construir la red neuronal mientras que nosotros programamos el algoritmo genético visto en clase.

II. Algoritmo Genético

A. Trabajos relacionados

Para este trabajo primero se tuvo que investigar sobre el juego a tratar. En este caso de Mario primero se consultaron diferentes soluciones para poder escoger la más viable con respecto a complejidad y calidad de solución (en tiempo).

Se consultó, primeramente "Teaching IA to Play Super Mario Land" que mientras es un juego diferente proporcionó las funciones y las ideas principales de qué usar para fitness y cómo montar las funciones genéticas.

Otro trabajo que ayudó mucho en la lógica de la programación fue la publicación "Building an AI Model to Play Super Mario" ya que explicó muy bien cómo conectar los algoritmos con el juego como tal.

B. Solución propuesta

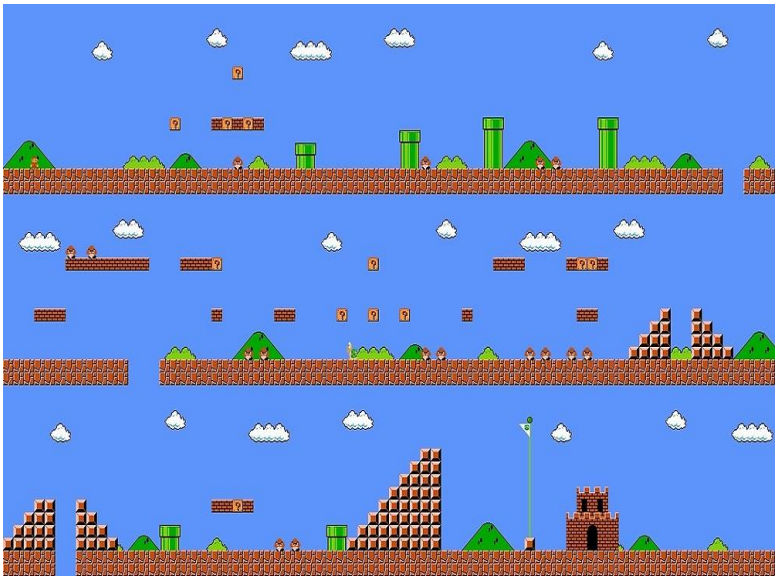


Imagen 1: Se puede observar el nivel 1 de Mario Bros (World). Se contaron alrededor de 50 pasos o comandos que Mario debe ejecutar para ganar el nivel.

Luego de estudiar el mapa del nivel, se llegó a la conclusión que un algoritmo genético normal no sería capaz de eficientemente resolver el juego. Por lo que se decidió utilizar una red neurológica NEAT (Neuroevolution of augmenting topologies). Para crear estas generaciones e ir aplicando el fitness.

C. Metodología de la investigación

A continuación, se va a explicar nuestra solución. Implementamos el algoritmo genético NEAT, basando el fitness en la meta, la cual sería la posición 3000. Una vez empieza a jugar un sujeto, al morir su posición se convierte en el fitness. Una vez se corren los 150 sujetos se escogen los mejores 147 y se reproducen entre sí, esta función lo que hace es tomar aleatoriamente la mitad del gen y mezclarlo con el otro padre. Estas generaciones van creciendo por lo que implementamos un proceso de mutación de ramificación progresivo, que se extiende a medida que crece el programa.

D. Diagrama de flujo

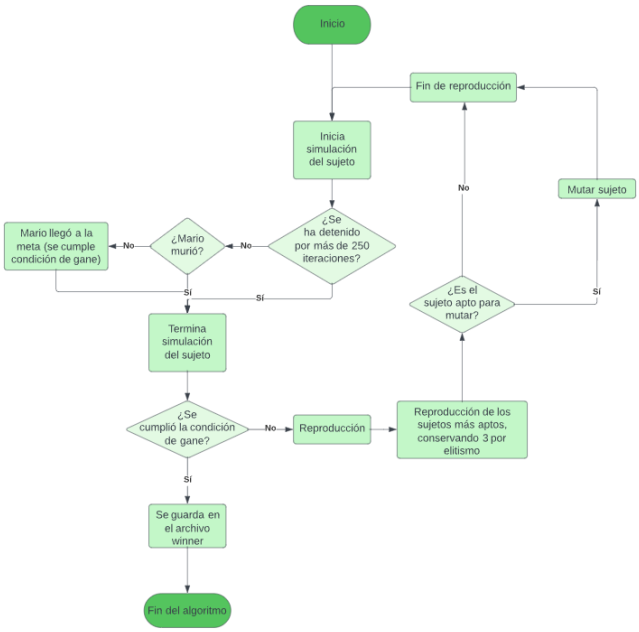


Imagen 2. En el siguiente diagrama de flujo se explica la lógica y funcionamiento del algoritmo genético.

E. Estrategia de ramificación

Para el algoritmo genético cruzamos 148 individuos a partir de los padres más aptos (aquellos que tienen el fitness más alto). El fitness se calcula de acuerdo a la posición en x de Mario. De esta manera el número más alto es quien llegue más lejos.

También aplicamos el elitismo por medio de 3 individuos para conservar sus genes en la siguiente generación. Así es como se crea un árbol familiar progresivo.

F. Tipos de cruces genéticos realizados

Se utilizó un "Single Point Crossover" que básicamente toma la mitad del gen al azar para juntarlo con el otro padre y así producir a su decendencia.

G. Tipos de mutación aplicada

En este caso la mutación trabaja con un valor aleatorio. Como entrada tiene el cromosoma anteriormente discutido, al cual se le suma este valor aleatorio elevado a la potencia de la mutación que está en nuestro archivo de configuración.

H. Maximización de recursos

Para lograr una mayor maximización de recursos se creó una función que bajara la calidad de imagen del juego. Esto convirtiéndola en una imagen blanco y negro y unidimensional. Esto para que sea más fácil de leer para la computadora.

Cabe destacar el booleano implementado para eliminar la interfaz gráfica. Esto provoca que el programa corra las generaciones más rápido.

1.Medición Empírica

Operaciones	*Tamaños del arreglo*					
	10	50	100	200	500	1000
Asignaciones	1143038	1143038	1149058	26167510	26167510	82969463
Comparaciones	379665	379665	381575	580185	8580958	27078262
Cantidad de líneas ejecutadas	2725354	2725364	2746418	5822603	13426204	118704994
Tiempo de ejecución	43.456	43.704	46.236	69.300	1078.101	3301.711
Cantidad de líneas del código	50					

En esta imagen se puede observar los resultados de la medición empírica.

2. Factor de Crecimiento

Talla	*Factor talla*	*Factor Asig*	*Factor Comp*	*Factor Cantidad de líneas ejecutadas*	*Factor Tiempo de ejecución*
De 10 a 50	5	1143038/1143038=1	379665/379665=1	2725364/2725364=1	43.704/43.704=1
De 50 a 100	2	1149058/1143038=1.005	381575/379665=1.005	2746418/2725364=1.008	46.236/43.704=1
De 100 a 200	2	26167510/1149058=1.521	580185/381575=1.521	5822603/2746418=2.12	69.300/46.236=1.499
De 500 a 1000	2	82969463/26167510=3.171	27078262/858958=3.156	118704994/13426204=8.841	3301.711/1078.101= 3.063
De 50 a 500	10	26167510/1143038=22.893	8580958/379665=22.601	13426204/2725364=4.926	1078.101/43.704= 24.668
De 100 a 1000	10	82969463/1149058=72.207	27078262/381575=70.964	118704994/2746418=43.222	3301.711/46.236= 71.410

Clasificación del comportamiento de las **asignaciones**	Cuadrática $O(n^2)$
Clasificación del comportamiento de las **comparaciones**	Cuadrática $O(n^2)$

Clasificación según su entrada de los datos use la notación O Grande según corresponda	
Entrada de los datos	**Aleatorios**
Clasificación	Cuadrática $O(n^2)$

En estas imágenes se pueden observar los resultados del factor de crecimiento.

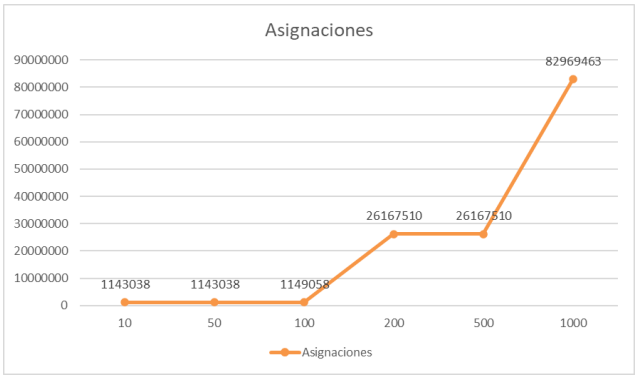
3.Medición Analítica

Código fuente Solo se analiza el código del método **Genético**.	Medición de líneas ejecutadas en el peor de los casos (línea por línea)

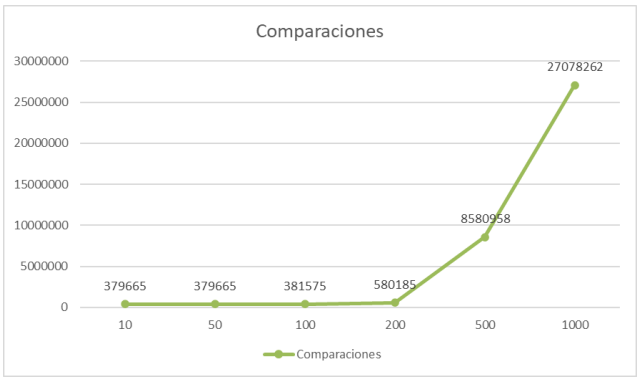
def downscale_image(ob, size_x, size_y):
"""Ajustes a la pantalla"""
# Cambiar el tamano de la imageen 	
ob = cv2.resize(ob, (size_x, size_y))
# Cambiar la imagen a blanco y negro. 	
ob = cv2.cvtColor(ob, cv2.COLOR_BGR2GRAY)
# Convertir la imagen en una matriz de dos dimensiones. 	
ob = np.reshape(ob, (size_x, size_y))
# Convertir el la matriz en un arreglo unidimensional. 	
imgarray = np.ndarray.flatten(ob)
return imgarray
	
	+1
	
	+1
	
	+1
	
	+1

def eval_genomes(genomes, config):
for genome_id, genome in genomes:
# Obtener la primera observacion del nivel. 	
ob = env.reset()
 	
# Obtener el tamano de la imagen y los colores. 	
size_x, size_y, img_color= env.observation_space.shape
 	
# Definir el tamano de la imagen reducida. (1/8 de la original) 	
size_x = int(size_x/8)
size_y = int(size_y/8)
 	
# Crear la red neuronal recurrente. 	
net = neat.nn.recurrent.RecurrentNetwork.create(genome, config)
 	
# Variable para guardar el fitness maximo actual. 	
current_max_fitness = 0
# Variable para guardar el fitness de cada red neuronal. 	
fitness_current = 0
# Variable para reconocer si la red neuronal se ha quedado estancada. 	
stag_counter = 0
	
	N^2+1
	n+1
	
	+1
	
	+1
	+1
	
	n+1
	
	+1
	
	+1
	
	+1

4. Medición Gráfica



Los resultados de las asignaciones muestran un aumento drástico entre los 100 y 200.



En esta imagen de las comparaciones se puede observar un aumento más fluido, que se muestra más cuadrático.

III. Algoritmo Probabilístico

A. *Trabajos relacionados*

Actualmente los algoritmos probabilistas o probabilísticos tienen múltiples usos en la industria de la computación. Uno de ellos es el del aprendizaje automático.

Es un hecho que la programación probabilística se puede utilizar para resolver muchos problemas de Machine Learning, Algunos ejemplos para los cuáles se ha aplicado programación probabilística en el mundo real son, el matchmaking de juadores en videojuegos, clasificado de correos electrónicos. Todo esto mediante el mecanismo de analizar suposiciones en un programa probabilístico y utilizar sus resultados en el problema presentado.

Entre la clasificación de algoritmos probabilistas encontramos el tipo Monte Carlo, que es el más utilizado, que, aunque no siempre se tenga la misma solución o la solución más óptima a un problema, se tiene la garantía de que siempre encontrará una solución.

B. *Solución propuesta*

La solución presentada se basa en un algoritmo de tipo Monte Carlo, muy popular y utilizado entre los algoritmos probabilistas.

Su funcionamiento se basa en crear posibles modelos de soluciones para un problema, que además se podrán recalcular para mejorar la solución encontrada. Para el problema de la resolución del juego Air Striker se utilizó esta estrategia.

Su funcionamiento se basa en generar movimientos aleatorios para la nave, ya sea para izquierda, derecha, centro y sus disparos. Dependiendo del éxito que haya mostrado un movimiento, este será agregado o no como parte de la solución, y cada vez que se presente un problema, se recalculan los últimos 20 movimientos, para corregir la trayectoria de la nave.

Como una mejora, la probabilidad del disparo de la nave se bajó a un 30%, para que, de esta forma, la nave no se quedara sin balas y la ejecución fuera un poco más eficiente.

C. *Metodología de la investigación*

Entre los conceptos necesarios para la realización del algoritmo se encuentran, el funcionamiento del mecanismo Monte Carlo, y el funcionamiento de Gym Retro. En cuanto a esto último, es completamente necesario para la ejecución del juego, y su utilización de parámetros. Lo más necesario es saber que “info[“gameover”]” corresponde a un número, con un significado respecto al estado actual del juego. Se analizaron el 9, para saber cuando el juego está activo, y el 2 para saber cuándo la nave explotó. También es importante saber el significado de cada parámetro de action, en la siguiente imagen se podrá observar.

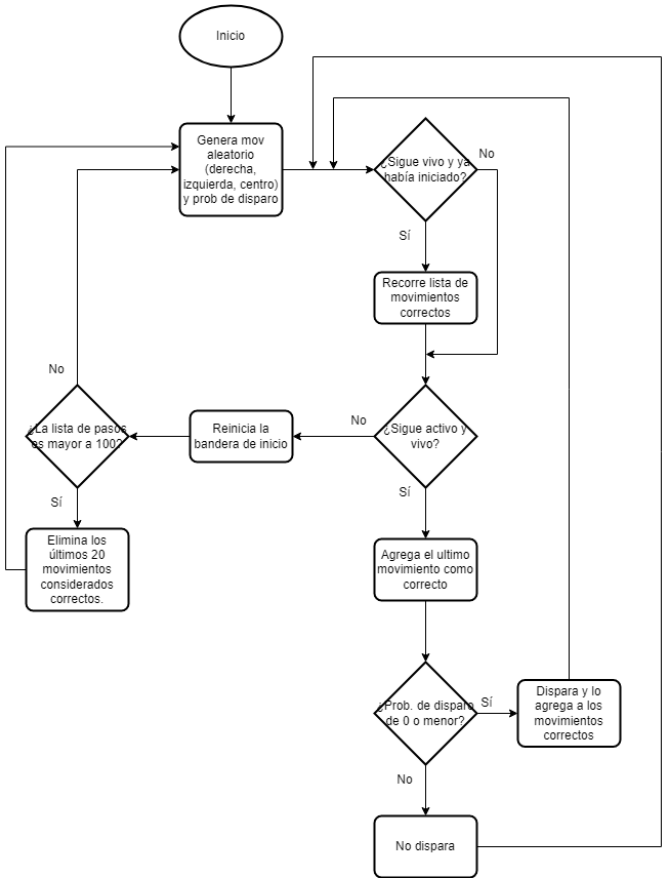
```
action = [randint(0, 1), 0, 0, 0, 0, 0, randint(0, 1), randint(0, 1), 0, 1, 0, 0]
#             0=B (disparo)
#             1
#             2
#             3
#             4=arriba
#             5=abajo
#             6=izquierda
#             7=derecha
#             8=A
#             9
#             10
#             11
```

Imagen 2. Action y sus parámetros.

Otro factor de gran importancia es “env”, que hace referencia al ambiente de juego, y tiene sus propias funciones, como reset, render, step (uno de los más importantes, pues se refiere al movimiento a hacer por parte de la nave), y close, entre otros.

Todo esto fue utilizado durante la realización del algoritmo.

D. Diagrama de flujo



1.Medición Empírica

Operaciones	*Cantidad de iteraciones*					
	10	50	100	200	500	1000
Asignaciones	43	190	400	802	3751	11727
Comparaciones	64	313	633	1267	4999	14353
Cantidad de líneas ejecutadas	132	617	1267	2537	13684	44932
Tiempo de ejecución (seg)	0.649	1.483	2.586	4.686	39.682	147.322
Cantidad de líneas del código	30					

2. Factor de Crecimiento

Talla		*Factor talla*	*Factor Asig*	*Factor Comp*	*Factor Cantidad de líneas ejecutadas*	*Factor Tiempo de ejecución*
De 10 a 50	Datos aleatorios	5	190/43=4,418604651	313/64=4,890625	617/132=4,67424242	1,483/0,649=2,28505393
De 50 a 100	Datos aleatorios	2	400/190=2,105263158	633/313=2,022364217	1267/617=2,0534846	2,586/1,483=1,74376264
De 100 a 200	Datos aleatorios	2	802/400=2,005	1267/633=2,001579779	2537/1267=2,0023678	4,686/2,586=1,81206497
De 500 a 1000	Datos aleatorios	2	11727/3751=3,126366302	14353/4999=2,871174235	13684/2537=3,28354282	147,322/39,682=3,71256489
De 50 a 500	Datos aleatorios	10	3751/190=19,74210526	4999/313=15,97124601	13684/617=22,178282	39,682/1,483=26,7579231
De 100 a 1000	Datos aleatorios	10	11727/43=272,7209302	14353/64=224,265625	44932/132=340,393939	147,322/0,649=226,998459

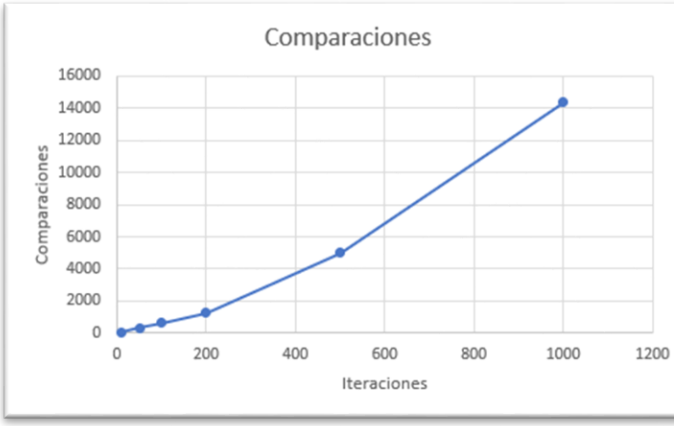
Clasificación del comportamiento de las **asignaciones**	Cuadrático
Clasificación del comportamiento de las **comparaciones**	Cuadrático

Clasificación según su entrada de los datos use la notación O Grande según corresponda	
Entrada de los datos	**Aleatorios**
Clasificación	Cuadrático

3.Medición Analítica

Código fuente Solo se analiza el código del método **Probabilístico**.	Medición de líneas ejecutadas en el peor de los casos (línea por línea)
while nivel1:	n+1
env.render()	n+1
probDisparo = random()	n+1
action = [0, 0, 0, 0, 0, 0, randint(0,1), randint(0,1), 0, 1, 0, 0]	n+1
ob, rew, done, info = env.step(action)	n+1
dormir()	n+1+1
if inicio and info['gameover'] ==9:	n+1
inicio = False	n+1
for i in range(len(movCorrectos)):	n*n
env.render()	n(n+1)
ob, rew, done, info = env.step(movCorrectos[i])	n(n+1)
dormir()	n(n+1+1)
if info['gameover'] ==2:	n(n+1)
break	n(n+1)
if info['gameover'] ==9:	n+1
movCorrectos.append(action)	n+1
if info['gameover'] ==2:	n+1
inicio = True	n+1
obs = env.reset()	n+1
if (len(movCorrectos)>100):	n+1
for e in range(20):	n*n
movCorrectos.pop(len(movCorrectos)-e-1)	n(n+1)
if probDisparo <= 0.30:	n+2
action = [randint(0, 1), 0, 0, 0, 0, 0, randint(0, 1), randint(0, 1), 0, 1, 0, 0]	n+1
ob, rew, done, info = env.step(action)	n+1
dormir()	n+1+1
if info['gameover'] == 9:	n+1
movCorrectos.append(action)	n+1
if done:	n+1
obs = env.reset()	n+1
Total (la suma de todos los pasos)	8n^2+29n+25
Clasificación en notación O Grande	Cuadrático

4. Medición Gráfica



El gráfico de ambos elementos, tanto comparaciones como asignaciones, coincide con la conclusión a la que se llegó en la medición empírica. Efectivamente, se observa una gran tendencia a un gráfico correspondiente a una función cuadrática, que, de hecho, también fue la conclusión a la que se llegó en la medición analítica.

Después de ver todos los resultados en los diferentes análisis, se puede concluir que el algoritmo probabilista utilizado tiene una clasificación de cuadrático.

IV. Evaluación

La solución genética de Mario Bros, luego de correrlo varias veces se llegó a la conclusión que es un algoritmo cuadrático, tomando en cuenta también los resultados de la medición analítica. Con respecto a su rentabilidad, es bastante eficiente en lo bajo que es el costo sin embargo el tiempo que tarda en correr va más allá de las expectativas. Sin embargo, si logró resolver el juego.

En cuanto a la solución probabilística para el Air Striker, se corrió el algoritmo múltiples veces para observar su comportamiento, una de ellas logrando completar hasta el nivel 3. Se clasificó al mismo como un algoritmo cuadrático.

Su medición pudo ser un poco más complicada, ay que no se puede medir de la misma forma que un algoritmo de ordenamiento. Debido a esto, se decidió utilizar la cantidad de iteraciones o veces que se ejecuta el algoritmo para su respectiva medición.

V. Conclusiones y trabajos futuros

El algoritmo genético es bastante tedioso a la hora de ver resultados debido a que los genes que se toman son aleatorios por lo que el producto de dos padres no siempre va a ser un Mario que logre llegar aún más lejos.

Sería interesante intentar el algoritmo en otros niveles para comprobar su eficiencia. Y cambiar la función de crossover para lograr una mutación más efectiva.

En cuanto al algoritmo probabilista, se concluye que al haber una restricción debido a que los movimientos generados inicialmente son aleatorios, se hace un poco lento al inicio encontrar una ruta correcta.

Una mejora importante a futuro sería optimizar la generación de movimientos aleatorios después de un error, para lograr una corrección en la solución de forma más eficiente. También sería muy interesante y útil dejar al algoritmo corriendo por más tiempo, completando más niveles, y seguir capturando los datos de medición, para tener un análisis más profundo de su comportamiento.

VI. Referencias

- Aprendizaje automático mediante programación probabilística. (s/f). Microsoft.com. Recuperado el 27 de noviembre de 2022, de https://learn.microsoft.com/es-es/archive/msdn-magazine/2019/january/net-machine-learning-through-probabilistic-programming

- Caparrini, F. S., & Windmill Web Work. (2021, noviembre 4). Algoritmo de Monte Carlo aplicado a Búsquedas en Espacios de Estados. Cs.Us.Es. http://www.cs.us.es/~fsancho/?e=189

- EALDE. (2020, agosto 26). En qué consiste el método de simulación de Monte Carlo. EALDE Business School. https://www.ealde.es/metodo-simulacion-monte-carlo/

- Getting Started — Gym Retro documentation. (s/f). Readthedocs.Io. Recuperado el 27 de noviembre de 2022, de https://retro.readthedocs.io/en/latest/getting_started.html

- Gym Retro Documentation OpenAI. (2020). Readthedocs.org. https://readthedocs.org/projects/retro/downloads/pdf/latest/

- O. Santiago, “Teaching AI to play super mario land-genetic algorithm,” *Medium*, 13-Jul-2021. [Online]. Available: https://towardsdatascience.com/teaching-ai-to-play-super-mario-land-genetic-algorithm-dde42c814e16. [Accessed: 28-Nov-2022].

- T. kubheka, “Build an AI model to play super mario,” *Medium*, 12-Jan-2022. [Online]. Available: https://python.plainenglish.io/build-an-ai-model-to-play-super-mario-7607b1ec1e17. [Accessed: 28-Nov-2022].