Romina: Una Inteligencia Artificial para Tetris

Edward Fernández 10-11121
Carlos Ferreira 11-10323
Stefani Castellanos 11-11394

RESUMEN

Dentro del campo de Inteligencia Artificial los juegos han sido ampliamente estudiados, por esta razón se decidió utilizar técnicas de Aprendizaje de Máquinas, específicamente algoritmos genéticos para crear a Romina: una Inteligencia Artificial capaz de jugar Tetris utilizando cuatro estrategias para determinar la mejor jugada posible en un instante del juego. Romina fue entrenada durante cinco días discontinuos y logró obtener una mejora en su evaluación de aproximadamente el 80%.

---

1. INTRODUCCIÓN

Los videojuegos son aplicaciones interactivas orientadas al entretenimiento que, a través de ciertos mandos o controles, permiten simular experiencias en la pantalla de un televisor, una computadora u otro dispositivo electrónico. Generalmente, los juegos de video hacen uso de otras maneras, aparte de la imagen, de proveer la interactividad e información al jugador. [1]

Típicamente, los videojuegos permiten recrear entornos y situaciones virtuales en los que el jugador puede controlar uno o varios personajes, para conseguir uno o varios objetivos según una serie de reglas determinadas. Uno de los juegos más reconocido, y al que ha sido tema de estudio, es el Tetris. De acuerdo a lo expuesto en [8], este ha sido históricamente uno de los videojuegos más versionados y es, junto a las Torres de Hanoi, el predilecto por los programadores noveles de juego.

El Tetris es un videojuego de lógica en donde varios tetrominos, figuras geométricas compuestas por cuatro bloques cuadrados unidos de forma ortogonal, se van generando de forma aleatoria y van cayendo en un tablero. El jugador no puede impedir esta caída, pero puede decidir la rotación de cada una de las piezas, la cual puede ser 0°, 90°, 180°, 270°, y en qué lugar debe caer. Cuando una línea horizontal se llega a completar, esta desaparece y todas las piezas que están por encima descienden una posición, permitiendo liberar espacio del tablero de juego, permitiendo generar nuevas piezas. El juego culmina cuando las piezas se llegan hasta el tope del tablero, interfiriendo la creación de más piezas.

Para hacer los juegos más atractivos, los desarrolladores de videojuegos utilizan diferentes tipos de técnicas para producir la ilusión de inteligencia en el comportamiento de los personajes no jugadores (PNJ), también conocido como Inteligencia Artificial (IA). Es un agente electrónico que puede "pensar", evaluar y actuar en ciertos principios de la optimización para cumplir con una meta o propósito, por lo tanto, en este trabajo se desea construir una IA que pueda jugar Tetris de la mejor manera posible utilizando unas estrategias (heurísticas) y algoritmo genético.

Se decidió utilizar un algoritmo genético debido a que el componente aleatorio en la generación de piezas impide la creación de un algoritmo de búsqueda eficiente para encontrar la solución óptima, además, no existe tal óptimo, puesto que la puntuación del Tetris siempre puede mejorar.

2. MARCO TEÓRICO

2.1. Algoritmo genético

Son métodos adaptativos que pueden ser usados para identificar la mejor hipótesis, es decir, aquella que optimiza una métrica determinada, dentro de un espacio de posibles candidatos [6]. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acuerdo con los principios de la selección natural. La evolución de dichas soluciones hacia valores aceptables del problema depende en buena medida de una adecuada codificación de las mismas y la métrica elegida.

Los algoritmos genéticos funcionan entre el conjunto de soluciones de un problema llamado fenotipo, y el conjunto de individuos de una población natural constituida por una colección de hipótesis [3]. La información de cada solución es codificada en una cadena, generalmente binaria, llamada cromosoma. Los símbolos que forman la cadena son llamados los genes. Estos cromosomas evolucionan a través de iteraciones, llamadas generaciones. En cada generación, los cromosomas son evaluados usando alguna medida de aptitud (*fitness)*. Las siguientes generaciones (nuevos cromosomas), son generadas aplicando los operadores genéticos repetidamente, siendo estos los operadores de selección, cruzamiento, mutación y reemplazo. [3]

2.2. Pygame

Es un conjunto de módulos del lenguaje Python que está diseñado para facilitar la escritura y desarrollo de software multimedia, como videojuegos en dos dimensiones, de una manera sencilla. Está orientado al manejo de sprites y requiere la librería multimedia SDL.[5]

2.3. Trabajos relacionados

El proyecto está basado en las ideas introducidas por Yiyuan Lee [2] en las que explica cómo obtener el "bot" (casi) perfecto para jugar Tetris. De dicho trabajo, se tomaron las estrategias de juego y los elementos fundamentales del algoritmo genético.

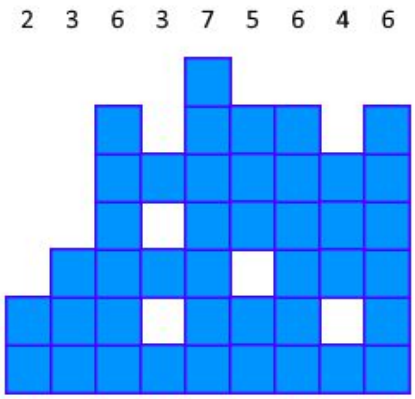
Debido a que no era el enfoque de la investigación realizar el juego, se utilizó un Tetris desarrollado en Python [8] puesto que, al estar utilizando el mismo lenguaje para el algoritmo genético, era sencillo de incluir.

3. HEURÍSTICAS

La utilidad de cada movimiento es calculada evaluando la rejilla (tablero) en la que se producirá el movimiento según cuatro heurísticas, basadas en las reglas del Tetris: altura total o altura agregada, líneas completadas, agujeros e irregularidades. Cada una de estas, la IA intentará minimizar o maximizar.

3.1. Altura agregada

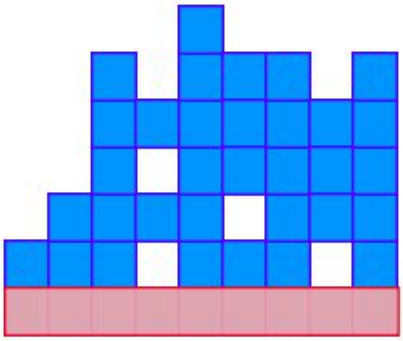
Esta heurística indica que tan "alto" está el tablero del juego. Para computar la altura agregada se deben sumar las alturas de cada columna en el juego actual, el cual corresponde a la distancia que existe entre el cuadro más alto de cada columna al cuadro más profundo de la misma. Se desea minimizar este valor, debido a que se pueden colocar más piezas antes de llegar al tope del tablero.



Altura agregada = 42

3.2. Líneas completadas

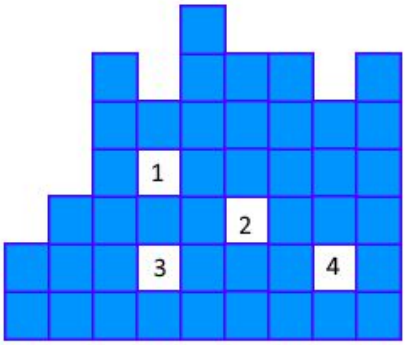
Esta heurística indica el número de líneas completadas existentes en el tablero de juego actual. En este caso, se desea maximizar este valor debido a que este es proporcional a la puntuación del jugador.



Líneas completadas = 1

3.3. Cantidad de Agujeros

Un agujero está definido como un espacio en el tablero en el cual no existe una pieza de Tetris, pero que tiene al menos otra por encima de él. Para poder eliminarlo del tablero se deben completar todas las líneas que estén por encima de él, por lo que se desea minimizar esta cantidad ya que complica la resolución del juego.

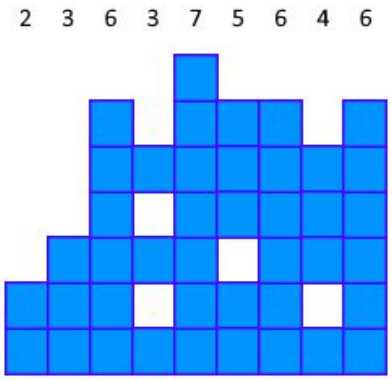


Número de Agujeros = 4

3.4. Irregularidades

Indica que la diferencia de alturas que existe entre una columna y la siguiente debe ser la mínima posible. Esto se debe a que al tener, en un instante del juego, columnas con irregularidades, una

columna sea mucho más alta que otra, hace más dificultoso colocar una nueva pieza y por lo tanto completar una línea nueva. Se desea minimizar este valor, debido a que se desea asegurar que el tope del tablero sea lo más monótona posible.



Irregularidades = 18

$$Irregularidades = \sum_{i=1}^{n-1} |número\ de\ bloques\ columna_i - número\ de\ bloques\ columna_{i+1}|$$

4. IMPLEMENTACIÓN

4.1 El algoritmo Genético

El algoritmo genético fue implementado utilizando la librería *pyevolve* [4] de *Python,* debido a que provee funciones que facilitan el desarrollo del algoritmo. También permite adaptar los métodos para el problema que se desea desarrollar y proveer facilidades para la creación de las estadísticas.

Debido a que el algoritmo requería varios días de entrenamiento se realizó un módulo de "guardado y recuperación" que provee la facilidad de, cada 5 generaciones, respaldar los individuos de esa población e indicar cual es el que posee mejor fitness. Esto permite detener las evolución en un momento dado y poder continuarla luego, sin perder el avance obtenido hasta el momento.

Para el correcto funcionamiento de los algoritmos genéticos es necesario establecer al menos ocho criterios, por lo que a continuación se explican qué decisiones fueron tomadas para la escogencia de los mismos:

1. **Cromosoma:** cada individuo está compuesto por un arreglo de enteros de cardinalidad cuatro (4), cuyos valores se encuentran dentro del rango -100 y 100. Cada una de ellas representa la importancia (peso) de la heurística particular. La primera posición corresponde a la "altura agregada", la segunda a las "líneas completadas", la tercera a la cantidad de "agujeros" y la última a la cantidad de "irregularidades".

   En la primera iteración se inicializa el genoma de manera aleatoria. Al detener y comenzar nuevamente el algoritmo se utiliza la población previamente guardada.

2. **Tamaño de la población:** se decidió utilizar una población de 500 individuos puesto a que se consideró que era suficientemente representativa para el problema.

3. **Cantidad de sobrevivientes:** de la población actual se escogen los 50 mejores individuos (10%) y son copiados a la siguiente generación, ya que es necesario que los individuos que posean un "buen" *fitness* prevalezca para mejorar la evolución, sin embargo una tasa muy alta impide que la población se adapte.

4. **Método de selección:** se utiliza el método de la ruleta, es decir, los individuos son seleccionados con una probabilidad que es proporcional a su *fitness*.

5. **Operador de Cruce (*Crossover)*:** se utiliza el valor por defecto de la librería, Cruce de un Punto, el cual consiste en seleccionar aleatoriamente una posición en el genoma [2] de los padres y copiar su material genético a los hijos. Se selecciona el 90% de la población para realizar los cruces.

6. **Mutación:** se utiliza el valor por defecto de la librería, intercambio, es decir, se escoge dos posiciones del genoma y es intercambiado. La probabilidad de mutación es del 5%.

7. ***Fitness:*** cada individuo realiza 50 juegos hasta perder o llegar a 100 líneas completadas (puntuación). La suma de las líneas completadas de dichos juegos es la función de *fitness*.

   Se decidió 50 juegos para cada individuo, de manera que cada uno pueda jugar una cantidad razonable de juegos en un tiempo aceptable, ya que más juegos implican más tiempo de cómputo. De la misma manera se seleccionó un máximo de 100 líneas completadas, con la finalidad de colocar una cota que permita terminar la corrida, de lo contrario podría durar horas impidiendo la evolución del siguiente individuo. El mayor *fitness* que un individuo puede tener es 5000.

8. **Criterio de convergencia:** hasta que la población converja, es decir, todos los individuos tengan el mejor *fitness* posible

Para guardar las estadísticas se utiliza un archivo csv con la función *setDBAdapter* disponible en *pyevolve* que, entre otras funcionalidades, permite obtener los resultados del mejor y peor *fitness* y el promedio, cada cinco generaciones.

4.2 La Inteligencia Artificial.

Para realizar el cálculo del *fitness* es necesario que la máquina realice movimientos en el juego, por lo que es necesario disponer de un algoritmo que se encargue de esto.

Fue desarrollada una Inteligencia Artificial que, dado un instante del juego de Tetris y una ficha a colocar, realiza una búsqueda sobre todos los posibles movimientos (incluyendo las rotaciones) y obtiene aquel que sea de mayor utilidad según los pesos de las heurísticas. Luego realiza el movimiento.

4.3 El juego

La implementación del juego [7] fue modificada para que la IA, explicada en la sección anterior, pudiese realizar los movimientos tanto en la fase de entrenamiento como en la demostración. Para el entrenamiento fue necesario agregar un mecanismo para dejar caer las piezas de manera instantánea y el reinicio instantáneo del juego, ya que de lo contrario la evolución podría tardar más de los necesario o se hubiese requerido de un humano que esté constantemente reiniciando el juego. Adicionalmente se incluyó el cálculo del *fitness* que no es más que la sumatoria de los 50 juegos jugados.

El juego presenta un error de un tablero inválido debido a un acceso erróneo a la matriz que lo representa, sin embargo esto no afectó los resultados obtenidos.

5. RESULTADOS Y ANÁLISIS

El algoritmo entrenó durante 4 días (apróximadamente 40 horas) en una computadora con un procesador Intel Core i5, 8GB de memoria RAM y una tarjeta de video integrada Intel HD Graphics 500 y fue interrumpido cada cierto tiempo evitar las sobrecarga de la misma.

Tabla generadas de la corrida del algoritmo genético. Donde está reflejado la fecha en la cual fueron realizadas y el mejor *fitness* obtenido de la población. Las siguientes estadísticas fueron tomadas cada 5 generaciones y utilizando la librería.

Fecha de la corrida	*Fitness*
23/03/17-08:54	40.0
23/03/17-15:44	6.0
23/03/17-16:34	92.0
23/03/17-18:59	79.0
23/03/17-19:02	104.0
23/03/17-21:39	3852.0
25/03/17-10:10	3794.0
26/03/17-14:25	4145.0

Luego de 40 generaciones el algoritmo genético fue capaz de llegar a un *fitness* 4145, el cual representa una mejora de aproximadamente el 80% con respecto a la primera generación.

Gráfica de Fitness contra generación.

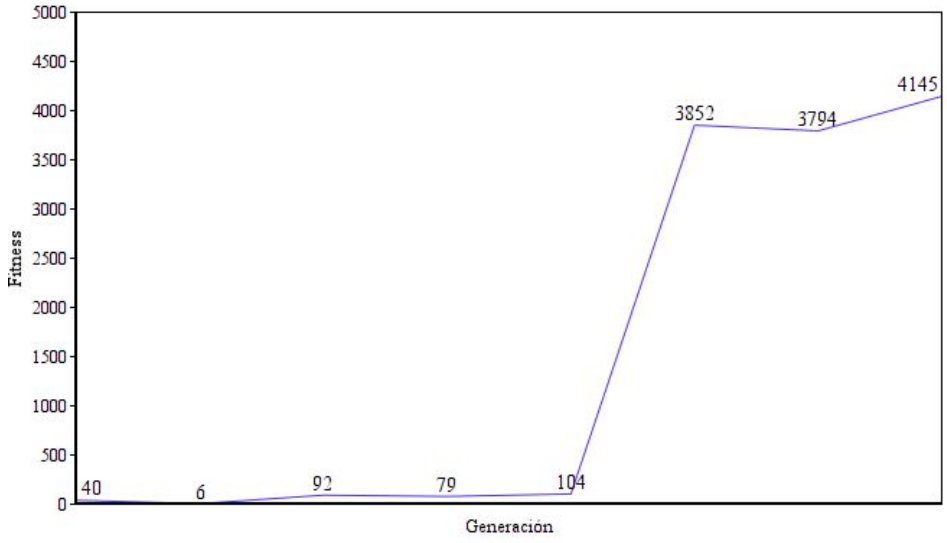


Figura 1. Evolución del *fitness*. Escala de generación: múltiplos de 5. Escala de *fitness*: múltiplos de 500.

En los resultados obtenidos se puede observar la tendencia creciente del *fitness,* lo que indica la mejora del mismo a través de las generaciones, esto concuerda con lo explicado con respecto al algoritmo genético. Resulta interesante la existencia de mesetas en donde de *fitness* no presenta variaciones significativas con el mejor de la generación (5) anterior, esto se debe a que el mejor individuo en diferentes poblaciones presentan aproximadamente las mismas características.

El mejor individuo hasta el momento posee un siguiente cromosoma: [67, 40, -71, -1].

Como fue mencionado anteriormente, el criterio de convergencia es que todas los individuos tengan el mejor *fitness* posible, sin embargo, deben realizarse muchas más iteraciones.

6. DISCUSION Y CONCLUSIONES

Dada la naturaleza aleatoria que posee el Tetris, un mismo individuo en 2 generaciones diferentes podría presentar una variación en su *fitness*, lo cual explica los puntos en los que se presenta un descenso en su valor a pesar de encontrarse en corridas posteriores; cada juego es completamente diferente y depende de las piezas generadas en el mismo.

Se logró comprobar que las heurísticas propuestas por Yiyuan Lee [2] son lo suficientemente buenas como para que el algoritmo pueda aprender a jugar mejor después de cierta cantidad de iteraciones (40), a pesar de que el experimento se realizó con la mitad de la población, de juegos para entrenar y menos de la mitad del tiempo. Sin embargo, para la heurística de altura agregada se esperaba un número negativo y se obtuvo 67, por lo que probablemente necesitaría más entrenamiento para alcanzar este valor.

Es importante destacar que estos algoritmos no buscan un resultado óptimo, sino uno lo suficientemente bueno según las métricas establecidas, además en el Tetris no existe un límite en la puntuación, por lo que siempre se podría mejorar el resultado obtenido. También se debe tener en cuenta que, aunque un individuo tuviese el mejor *fitness* posible (5000) no implica que jugará de manera perfecta, puesto que al limitar los juegos a 100 líneas completadas los pesos de las heurísticas tratarán de mejorar su desempeño hasta llegar dicha puntuación, sin embargo, no son capaces de estimar y mejorar jugadas posteriores.

Para mejorar los resultados obtenidos se podría realizar por más tiempo la etapa de aprendizaje o cambiar algunos parámetros del algoritmo, por ejemplo la cantidad de juegos y el límite de líneas completadas de los mismo, pues de esta manera un individuo particular podrá observará una muestra mayor de juegos y “aprende” una mejor estrategia para resolverlo.

7. REFERENCIAS

[1] Entertainment Software Association (2015). *Essential facts about the computer and video game industry*. Recuperado de: http://www.theesa.com/wp-content/uploads/2015/04/ESA-Essential-Facts-2015.pdf
[2] Lee, Y. (2013). *Tetris AI - The (near) perfect robot.* Recuperado de: https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/
[3] Mitchell, T. Machine Learning (1997). 249-255.
[4] Pyevolve (2009). *Pyevolve documentation*. Recuperado de: http://pyevolve.sourceforge.net
[5] *Pygame documentation*. Recuperado de: https://www.pygame.org/wiki/GettingStarted
[6] Rodriguez, P. Introducción a los algoritmos genéticos y sus aplicaciones *Universidad de Valencia.* Recuperado de: https://www.uv.es/asepuma/X/J24C.pdf
[7] Tetris. Wikipedia. Recuperado de: https://es.wikipedia.org/wiki/Tetris
[8] *Tetris Implementation in Python.* Recuperado de: https://gist.github.com/silvasur/565419