Universidad de Costa Rica

Escuela de Ciencias de la Computación e Informática

CI-0134 Investigación en Ciencias de la Computación

Proyecto:
Tetris AI

Estudiantes:

Josef Ruzicka González, carné: B87095

Karen Zamora Villalobos, carné: B37741

2° ciclo, 2022

Descripción del juego

El juego que se implementa es el de Tetris. Este según Liu & Liu (2020) es un juego clásico que se desarrolla en un tablero bidimensional, donde piezas de 7 formas diferentes, llamadas tetriminos, caen desde arriba gradualmente. Para este juego el objetivo es mover y rotar los tetriminos que caen para formar filas completas en la parte inferior del tablero. Cuando una de estas filas se llena, se elimina toda la fila y todas las celdas sobre esta fila bajarán una fila. El juego termina cuando no hay espacio en la parte superior del tablero para el próximo tetrimino. Por lo que el objetivo final es el de eliminar la mayor cantidad de posibles antes de que termine el juego.

En nuestro caso el tablero tiene una altura de 20 filas y un ancho de 10 columnas, siendo entonces su dimensión de 20x10. y se utilizan las 7 piezas del juego original, compuestas cada una por 4 bloques, las cuales se ejemplifican en la figura 1.

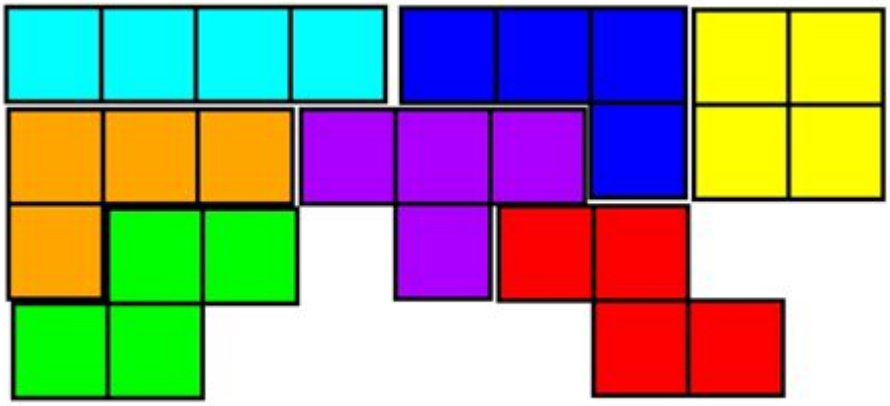
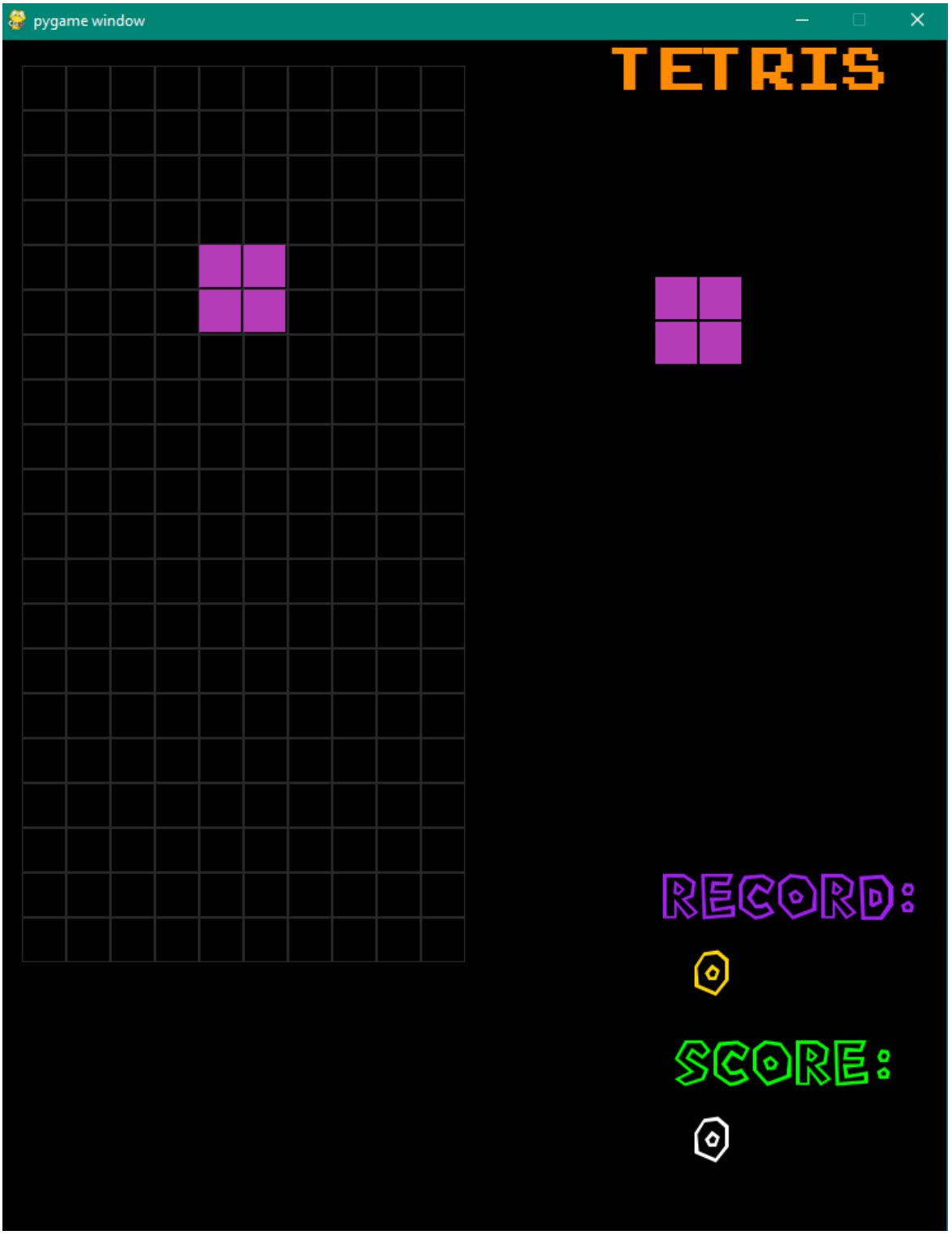


Fig. 1

En la figura 2 se puede observar como se ve la interfaz de nuestro juego, en la que se tiene el tablero antes mencionado al lado izquierdo y al lado derecho se muestra la próxima figura en aparecer. Además, se muestra *record* que corresponde al registro de la puntuación más alta obtenida hasta el momento y *score* que indica la puntuación obtenida en el juego hasta el momento.



Descripción del agente

El tipo de algoritmo de aprendizaje mecánico que se utilizará para el desarrollo del agente, las características del agente (entradas/salidas/interacción con el espacio).

El programa se basa en recompensar la colocación de piezas a la menor altura posible, entonces nuestros estados consisten en un arreglo con las 10 alturas máximas de cada columna, pero hacer todas las combinaciones posibles de alturas de cada columnas requeriría demasiada memoria: 10 columnas con valores entre 0 y 20 para cada una representando su altura por lo tanto ignoraremos el eje x, y nuestra tabla sólo se tratará de dónde estamos en el eje y, recompensando solo basado en altura (también habrá recompensas por eliminación de filas y castigos por perder el juego).

Leyendo la publicación de las referencias observamos que la solución que sus autores utilizaron consistía en asignar a sus estados: Posibles ubicaciones para la pieza actual, huecos en el tablero actual y altura máxima de cada columna, sabemos que con toda esa información sería posible crear una inteligencia bastante superior, sin embargo, como estamos intentando aprender poco a poco, solo utilizamos una de esas de momento (además, esa se nos ocurrió antes de leer el paper).

Con respecto a las recompensas utilizadas en el entrenamiento, se consideraron varias opciones de reconocimiento de puntos para el agente, una propuesta, por ejemplo, consistía en calcular la desviación estándar entre las alturas máximas de cada columna para así poder premiar más cuando la altura se mantenía lo más baja y pareja (para completar filas) posible, la otra propuesta (la que finalmente utilizamos) consiste en otorgar puntos por acciones concretas, como la de colocar una pieza en una altura menor a la tercera parte del mapa, y quitar puntos por colocar piezas a una altura mayor a un dos tercios del mapa. Siempre se supo que se asignarían y eliminarían puntos por completar líneas o perder el juego.

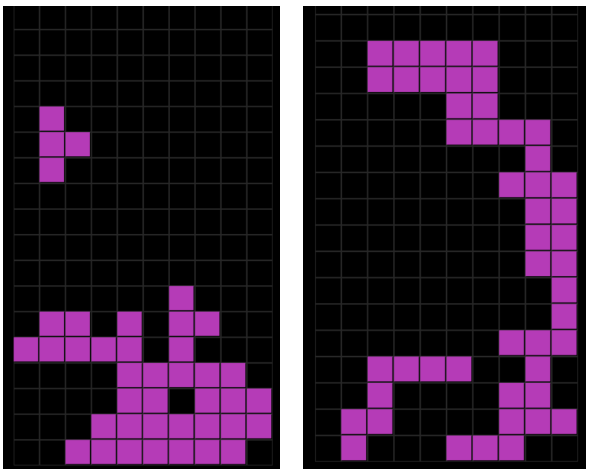
Desafíos encontrados en el desarrollo del agente

desafíos encontrados en el desarrollo de su agente, cómo se resolvieron los mismos.

Se inició el proyecto con la intención de crear un agente con una red neuronal y utilizar un modelo de deep Q learning, la red neuronal se implementó, pero no se tuvo claridad con respecto a cómo hacer el entrenamiento conectándolo con el juego, por lo que se recurrió a una Qtable y un Agente de Q learning.

Resultados obtenidos

El agente logra completar algunas líneas, y con suficientes épocas de entrenamiento parece lograr intentar hacer patrones más “aplanados” y menos asimilados a una torre.



Ejemplo de patrón aplanado vs patrón asimilando una torre.

Referencias bibliográficas

Space Coder. (2022). Let's code Tetris Game in Python. Recuperado de https://www.youtube.com/watch?v=7kGNs5R-AM8&t=300s

Liu, H., & Liu, L. (2020). Learn to Play Tetris with Deep Reinforcement Learning. Recuperado de https://openreview.net/pdf?id=8TLyqLGQ7Tg

Loeber, P. (2020, diciembre 20). Teach AI To Play Snake - Reinforcement Learning Tutorial With PyTorch And Pygame (Part 1). Recuperado de https://www.youtube.com/watch?v=PJl4iabBEz0

Moreno, J. M. (2019). Q-Learning aplicado a Tetris. Recuperado de https://www.youtube.com/watch?v=z4OomBu6kD0&ab_channel=JavierMart%C3%ADnMoreno

Schoberg, S. (2020, abril 9). Introduction to reinforcement learning (Q-learning) by maze solving example. Recuperado el 7 de diciembre de 2022, de Analytics Vidhya website: https://medium.com/analytics-vidhya/introduction-to-reinforcement-learning-q-learning-by-maze-solving-example-c34039019317