Augie Muenzenmay, Daniel Osafo, Miles Pei
May 5, 2022
CS:4420:0001
Adhikari

Artificial Intelligence Final Project Report – Tetris AI

Part 1: Introduction/Motivation

The primary motivation for this project was to take the knowledge of the concepts that we have learned throughout this past semester and apply them to source code that we find ourselves. Although there are many interesting and powerful project ideas in the world of Artificial Intelligence, our group found Reinforcement Learning for the game of Tetris, to be what we go forward with, for this project. Reinforcement learning is a machine learning/artificial intelligence technique that rewards an agent for choosing the correct actions leading towards a goal and uses the knowledge from sampled outcomes to improve its decisions. The significance of reinforcement learning is manifested as the more trials an agent undergoes, the better the general performance of the agent is. This notion is instrumental for understanding how to solve problems in complex domains, particularly, domains in which the agent is unsure of how the environment works or the consequences of its actions. We wanted to apply reinforcement learning principles to Tetris, one of the most popular arcade games in history. We hoped that by the end of the time allotted for our project, we would have a fully functioning and high-scoring Tetris reinforcement learning agent.

Part 2: Problem Definition

Regarding Tetris, the agent is given a 10 x 20 board and 7 unique shapes (I, J, S, Z, O, L, and T) made of 4 blocks each, commonly referred to as “tetrominoes,” that are selected at random to descend from the top of the board to the bottom of the board. The general contour of the grid is defined as the placement of shapes at the base layer of the grid. As the blocks approach the bottom, they merge with the shapes at the base layer, thus changing the general contour. The shapes can rotate clockwise or counterclockwise and move left, right, down, or choose not to move. The scoring is based on the number of blocks landed and this is helped by the elimination of rows when a full row has a block in each x-position. The given inputs for the game are the grid, the shape that is descending, as well as the next shape that will descend. Furthermore, the best possible output would be a grid in which the shapes are being dropped in ways that rows can be eliminated so more blocks can be placed. In Tetris, a player wants to avoid a situation in which the blocks are stacked to the maximum vertical dimension of the grid, in which no more blocks can be stacked, and the game terminates. In terms of a reinforcement learning agent, the Tetris agent must learn how to place shapes in a way that reduces the possibility of the game terminating, without knowledge of which rotations and translations an agent should take for its current shape, given the contour of the grid.

Part 3: Proposed Method

Reinforcement Learning for Tetris is an ideal technique as the Tetris environment is stochastic, thus, the policy determines the produced action. Whereas other techniques that we have learned in class such as the informed, uninformed, and adversarial search algorithms, rely on deterministic environments and MDPs use a formal representation of the environment, instead of developing their own representation, which would be less efficient for our project. With this problem, we used Q-Learning to implement our reinforcement learning technique. This was an obvious choice as the state space for Tetris is enormous, set to be around 2^{200} possible

states. However, to use Q-Learning, we had to figure out how to decrease the state space. One idea that we had to accomplish was to keep track of the index of the highest block and the number of holes in a given column. (A hole, meaning a situation in which there is a cell in the grid that is left free of a block, but is surrounded by other cells full of blocks). However, this idea was unable to reduce the state space because each state would still have to keep track of all possible state and action combinations for the entire grid.

After a lot of research, we stumbled across a research publication from Rhodes University, of which we based much of our solution. Instead of keeping track of the tallest blocks and number of holes, we realized that it was more important to break down each state to individual wells, (the individual contours of smaller intervals of the grid), and keeping track of the relational differences between the highest block of each successive column in a tuple. Each of these wells would have a width of 5, due to the maximum shape width being a width of 4, and have 4 relational height differences listed so only the height of an individual column is examined. Furthermore, only one of the shapes can spread across a height difference of more than 3, of which it is also able to spread across any height difference, we normalized the height differences to range from -3 to 3 and adjusted that range to be from 0 to 6, so the differences to only include non-negative values. Thus, we were able to assign proper weights to our relational difference feature multiplying each difference by 7 (the number of possible values a tuple can have in each index) to the power of their index relative to the well (1, 7, 49, 343 respectively). This calculation gives each state a distinctive value and is added to a state key that will also keep track of the current shape being dropped, the placement position, and movements taken. Thus, the state space is able to adjust to states with extremes in height difference, focus on individual transitions, and rule out any information below the highest blocks of each column.

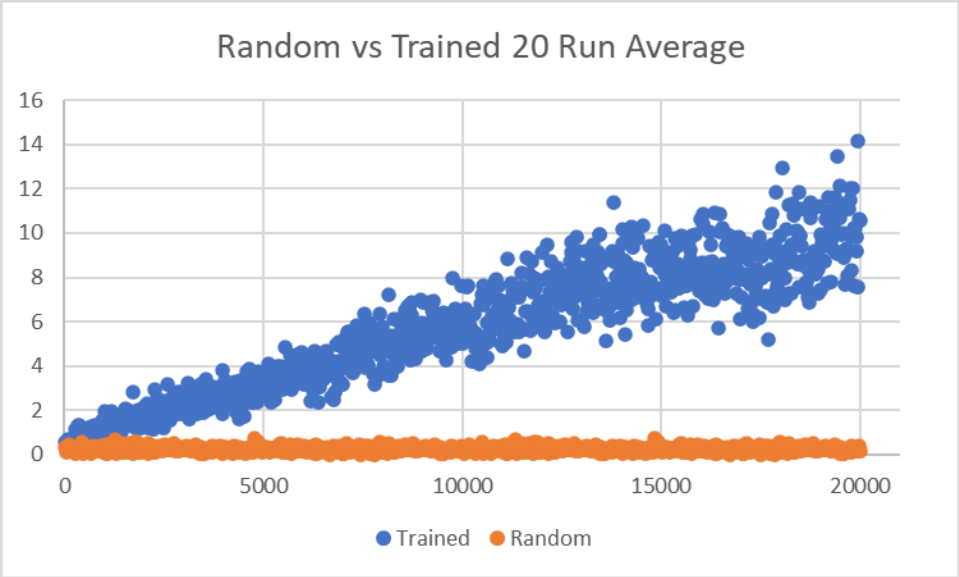
Following this, our next move was to implement Q-Learning with our newly reduced state. We decided to base our reward function off the Rhodes article's suggestion, and we penalized the agent by a score of -100 for every time the height of the well was increased, -40 points for every new hole that was introduced, and a positive score of 1 for every new line that was cleared. We then also used Rhodes' suggestion of doing Q-Learning through a SARSA(λ) implementation in which each value is associated with each action leaving each state, which improves the decision-making process, however, also increases the necessary amount of storage and time for discovery. We based the learning off

Part 4: Experiments/Results

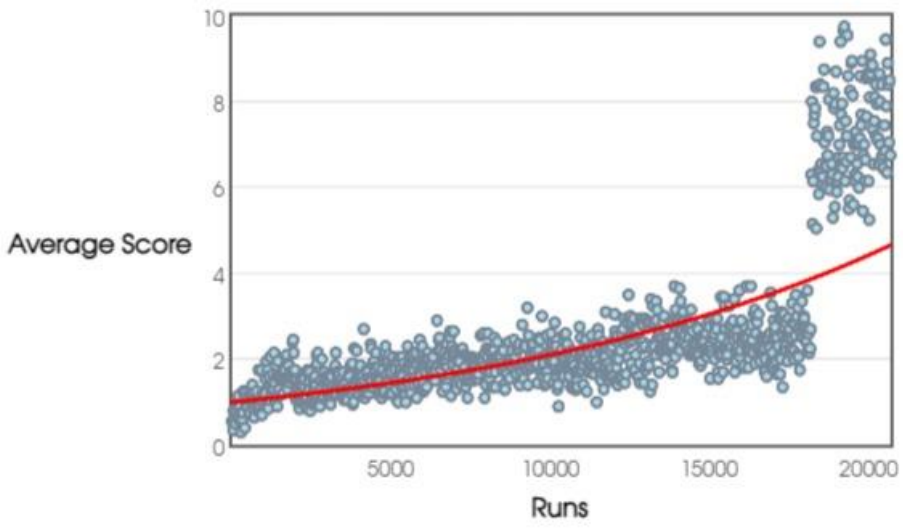
- The questions we centered our experimental observations around were: Is the Tetris agent learning and how much does the randomness of selection impact its learnability? Our project test bed was formed by using the pickle package to save our previous Q-values to save and train data, as well as the pandas package to store statistics from our training into .csv format.

- For the first question, our results demonstrated significant learning from the agent. The metrics that we used to analyze whether it was learning include the maximum score obtained, average number of shapes, average score obtained, and number of states discovered. This trained model obtained a high score of 47, whereas the random model

reached a high score of only 6. The average number of shapes seen per game more than doubled the random model by the end of the training.



- For our second question, during experimentation, all variables were kept the same except for the epsilon value which determines whether to explore or exploit what it has learned. The alpha and discount values we used were set at 0.7 and 0.8, respectively. Gradually decreased the initial epsilon value from 0.5 to 0.45 after the 2000th episode. 0.45 to 0.4 after the 4000th episode. 0.4 to 0.2 after the 6000th episode. 0.2 to 0.01 after the 8000th episode. The average score nearly doubled after the jump from 0.2 to 0.01.



Part 5: Conclusion and Discussion

Augie Muenzenmay, Daniel Osafo, Miles Pei
May 5, 2022
CS:4420:0001
Adhikari

The project goal was to apply reinforcement learning to a Tetris-playing agent, so it can learn how to obtain increasingly high scores. Our Tetris Agent was demonstrated to show effective learning. Our understanding of the state space, rules of the game, and continuous testing and debugging proved crucial for our ability to have a fully functioning learning agent. Furthermore, there are many more games and research problems that would be able to be solved by similar processes to the ones we used in Tetris. There are also versions of Tetris that could benefit from our code or other artificial intelligence techniques that we have used in class. We trained and ran our Tetris agent with a grid width of length 5. In the future, we would like to expand our algorithms to play in a full grid of length 10. Overall, reinforcement learning is one of the most promising and holistic aspects of artificial intelligence. The skills that we have acquired in class have enhanced all our abilities to take complex problems and analyze them in methodological and effective ways.

Part 6: Individual Contributions

- Augie
  - Implemented a significant portion of the Tetris reinforcement learning code. Restructured the source code to be able to work with our agent. Implemented the training and added functionality to save stats from the training data as well as save the learned values to .pickle files. Trained the model for several hours. Researched tetris models to reduce the state space to a reasonable number of states.
- Danny
  - Worked on researching project options, project research, how to conceptualize and represent the state space, helping implement reinforcement learning functionality, providing additional test and debugging information, typing up the final report and README.md, and documentation.
- Miles
  - Researched the application of Q-Learning on Tetris. Implemented the Q-Learning functions. Provided training data for the Tetris agent. Added README.md and function documentation. Worked on implementing Q-Learning for 2048 as a backup. Assisted in typing final report.

Bibliography:

Carr, D. (2005, November 7). *Adapting reinforcement learning to tetris*. CiteSeerX. Retrieved May 5, 2022, from https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.548.9414

https://github.com/LoveDaisy/tetris_game