Optimizing Tetris with Deep Reinforcement Learning

Author: Haroon Shahzad
December 15, 2023

Abstract

Tetris is a simple game that has evolved to have a consistent ruleset with tools and techniques that allow for setups to maximize in-game scores. Deep Neural Networks have been able to learn to play iterations of Tetris in the past, though approaches usually prefer techniques that stack into a one-wide well where an I tetromino can be placed to score four lines at once, the highest possible score. Now with new scoring mechanisms like t-spins, this paper explores how new strategies can be compared to traditional approaches to optimizing Tetris. The first agent that will be tested will be the normal Tetris agent, the second agent will be a modified version of a normal Tetris agent that penalizes itself whenever only one single line is cleared, and the final agent will prioritize t-spin clears. The game state and information are fed into a convolutional neural network, which uses deep reinforcement learning to optimize the gameplay and updates its variables through a value-iteration algorithm within a Q-learning algorithm that plays the game. After each agent uses its final generation's learnings and adjustments to play a 300-piece game, the scores and line clear distributions will be analyzed. The metric used to analyze the performance of the agents will be through how optimally they can maximize their score within the tetrominoes given to them. Finally, the outcomes will be discussed on what work can be done to incentivize the agent to play the game more effectively while increasing the difficulty, which would increase the chance of the agent losing the game.

1 Introduction

Tetris started the revolution of gaming as it was one of the first computer games to be made by legendary creator Alexey Pajitnov. At its core, the game mechanics are simple. The player is given a random tetromino, which is a shape with 4 blocks that are conjoined on adjacent edges of the blocks. There are a set amount of shapes that tetrominoes come in, so the player needs to determine which is the best piece to play to fill lines and clear them off the 10 by 20 matrix. When a line is filled, it is cleared from the board. The issue is that the board isn't always perfect as the user can play a piece and create imperfections on the boards, and these are called holes and bumps. The player must maintain a smooth and clean board to ensure optimal lifespan while also obtaining the maximum amount of score possible until they reach the top, prompting the game over screen.

Tetris has evolved over the years since its first iteration in 1984 [1], with a new generation called "Modern Tetris". Modern Tetris includes many new features that the older version of the game, called "Classic Tetris", does not include. One of the main features added to modern Tetris was the hold function [2]. The hold slot allows the player to save a piece in

the slot which can be used later on. When wanting to use the held tetromino, the tetromino currently being played gets swapped with the one that is in the hold slot. This is incredibly useful as one major issue that occurs in Tetris is when the player is unable to clear lines with the long I tetromino, which often results in a game over situations due to a drought in I tetromino occurrences. With the hold slot, any tetromino can be saved for when the time is right, allowing the player to efficiently execute the perfect move to clear the most lines possible. Another feature added to modern Tetris was t-spins. Tetris would give the player more points based on the number of lines cleared with one tetromino being placed. Players wanted to use the long I tetromino to clear four lines at once to get the optimal amount of points, granting the maneuver the name "Tetris clear" as it was the best move in the game. Modern Tetris puts a twist on this by allowing for the T-appearing tetromino to be spun into a gap that it previously would not have been able to fit in, doubling the lines cleared as a bonus for doing the risky maneuver. Hence, a t-spin that clears just two lines is rewarded the same amount of points as a regular Tetris clear would have netted, allowing for higher points per piece played and faster score progression once one learns how to set up t-spins with some finesse. One final change that modern Tetris offers is better visibility for tetrominoes that are to come. In classic Tetris, tetrominoes are given at random, and only the next tetromino is visible in the queue of pieces that are yet to come. Modern Tetris allows for the next five tetrominoes to be viewed, and when considering the hold slot having the ability to store a tetromino the total goes up to 6 tetrominoes to consider when wanting optimal piece placement.

In this paper, the focus will be on training a deep reinforcement learning algorithm to play modern Tetris with the approach of emphasizing the use of modern tools like holds and t-spins to gain the highest possible score in the shortest amount of time. With the holding slot being available, the agent needs to learn across its generations how to effectively utilize its extra tetromino option to have optimal tetromino placement. However, the real challenge arises when the concept of how to train the artificial intelligence to go for t-spins as it has to intentionally make "bad moves" to find a t-spin opportunity. T-spin setups utilize different gaps of certain pieces when stacked on top of one another that create the shape for the T-shaped tetromino to fit in, which increases the number of holes and bumps on the board. The agent that plays Tetris will want to minimize the number of holes and bumps to have smoother gameplay, so the challenge is to incentivize the agent to go out of its way to net itself with the bonus points from the t-spin. Finally, the agent will need to learn how to not just play the current tetromino optimally but to also look forward by using the queue that is available to itself to play tetrominoes effectively and set itself up for success.

2 Related Work

Stevens and Pradhan [4] have done extensive research on the reinforcement of AI by giving rewards for desired actions. For example, they mention that "every time the agent

clears lines, the reward is the number of lines cleared, squared". That way, clearing lines like a Tetris line clear will award 4^2 points which gives 16 points compared to doing a single line clear which gives only 1^2 points, or 1 point. By building up and taking a risk to make a Tetris clearing well, the AI is rewarded with many more points than if it were to play safe and do single-line clears 4 times. This also used an epsilon-greedy policy where the agent makes an educated guess on what the "optimal" move is within its range of possibilities, incentivizing the clearing of lines based on good piece placement. With all of this contained within the deep reinforcement learning algorithm, Stevens and Pradhan [4] decided to make the algorithm approximate a function to determine the future discounted sum of possible rewards in the future with the following function below.

$$Q^*(s,a) = max\, E(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s, a_t = a, \pi)$$

Here, s_t is the time at t, a_t is the action that is done at t, π is the policy function that indicates what the action should be done with the knowledge of the current state from $t+1$ and onwards, r_t is the reward given at t when doing the action a_t with state s_t, and γ is the discount factor for how much the agent should care about future rewards. Knowing how the function continues as t (or time) grows, the Q function expands on the Bellman equation given as below.

$$Q^*(s,a) = E_{s'}\left[r + \gamma\, max\, Q^*(s', a') \mid s, a\right]$$

This equation has s' which is the possible state that results from action a taking place given state s, and a' is the action that takes place after s' is given. Having this network set up to receive the best possible action a after time t, the Q function can train the neural network as it obeys the Bellman equation and is in the form of the equation above. This training can now be cached into the neural network through the equation below.

$$Q(s,a) = r + \gamma\, max\, Q'(s', a')$$

In this equation, the Q' is the cached neural network that allows for updates to occur infrequently to prevent any instability, and self divergence from the network's q values. Finally, the network finds the priorities within each experience to learn from with the equation below.

$$Priority = \left|Q(s,a) - r - maxQ'(s', a')\right|$$

This setup for the neural network led to an average game score for the best possible model to be about 200 per game, and the relative game length to about 21.7 moves per game. This didn't produce the greatest results but still did show a proof of concept for how the neural networks could learn how to optimize the game score for classic Tetris. In contrast, the article by Lundgaard and McKee [5] on Tetris artificial intelligence through the usage of "Greedy AI", a policy in which eliminating rows is the most highly prioritized action. Prioritizing line clears immediately over anything else helps the AI keep itself alive and in the game. Here, the creation of valleys for the I tetromino is prioritized to clear the maximum amount of lines possible to get the best possible score, which was also implemented with the reinforcement learning agent with Q-Learning in the form of *Q(s,a)* and attempting to reach an optimal $Q^*(s,a)$ to find future optimal moves for whatever possible state and actions are given. $Q(s,a_i)$ is modified with the formula below.

$$\mathrm{T}Q(s,a_i) = Q(s,a_i) + \alpha\left[r + \gamma max_{a'} Q(s_{t+1}, a') - Q(s,a_i)\right]$$

The advantage of the above formula is that the available actions a' for state s_{t+1} learned from the largest value of *Q*, which then changes what the γ value is going forward to advance the training weights until it converges to a more optimal state. After this, The Q-learning network uses back-propagation to predict what the best action is and what value it will give to the *Q(s, a)* state through the formula below.

$$Q(s_t, a_t)_{observed} = r_{t+1} + \gamma max_a Q(s_{t+1}, a_{t+1})$$

With gamma (γ) value of .1, the following state would be able to calculate what the highest value action would be given the current state and its action. Additionally, the weights are updated within the neural network through the use of the formula below.

$$W_{ij} = W_{ij} + \eta\delta_j x_{ji}$$

Here, the η is a constant with the value of 0.1 that is used to adjust how fast the neural network wants to change itself, controlling the adjustment and mutation rate of the weights within the neural network itself. Lundgaard and McKee [5] attempted to implement this neural network by feeding the network the positional data of the board, indicating what cells on the matrix were filled or not, and then showing the shape of the tetromino that was currently in play. Different nodes within the network would observe how the rotation of the piece would alter the scores upon placement, which was then used to make the final decision and adjust every node. This process resulted in slow learning and did not net results that were

promising within the time frame as this was a low-level neural network with over 202 input nodes, 40 output nodes, and 40 hidden nodes, and the result after 6,500 games had no changes after the 250 game mark, and the average score staying at a stagnant 350. Rather than the neural network approach, a generally easier strategy that would guarantee results was implemented by Wei-Tze et al. [6]. This strategy proposes the use of breadth-first-search, or BFS, to find the best possible action given the board state of the game. The agent was able to see the next three tetrominoes that were queued for play, so the goal was to stack enough I tetromino wells and see whenever an I tetromino was coming to place for optimal points. The agent also followed a strategy where back-to-back Tetris line clears would net bonus points, also called B2B rewards. When chaining B2B Tetrises, the score would be higher than if Tetris line clears were to be scored individually, leading to the agent to stack up 9 columns on the board and leave one single wide well for the B2B Tetris line clears to occur. This method also includes the hold functionality where a piece can be held in a slot that can be called upon later, so the agent would consider future moves alongside the tetromino inside of the hold slot. The result of this method led to lower overall time complexity for figuring out where each tetromino should be placed compared to other approaches like greedy choice which would prioritize cleaning lines at any points possible, and another model which prioritized a two-wide well instead of one wide well. In the end, the BFS model and the two wide well models outperformed the greedy choice model.

3 Approach

My approach to having an agent play modern Tetris optimally with different mechanics like t-spins would be to create a deep reinforcement learning algorithm that trains and learns across each generation to improve upon what it had learned. This would be done through a Q-learning algorithm that would incentivize learning the placement of pieces while using a value-iteration algorithm to train the variable values within the neural network. This procedure is outlined by Rex L [7] which is the repository I used for this project, and the article written about it discusses the use of such learning methods. First, an outer loop collects samples for *{(s, s', r)}* through the formula below.

$$1.\ \left\{\left(s_i,\ s'_i,\ r_i\right)\right\},\ s'_i\ =\ argmax_{si}\left[r_i\left(s'_i(s_i)\ +\ v_\phi(s'_i)\right)\right]$$

Then, the inner loop of the algorithm runs the formulas below to set the values of the next outer loop to change the values within argmax.

$$2.\ y_i\ =\ \delta\left[r_i\left(s'_i \mid s_i\right) + v_\phi(s'_i)\right]$$

$$3.\ \phi\ =\ argmin_\phi\ \frac{1}{2}\Sigma_i\left|\left|v_\phi(s_i)\ -\ y_i\right|\right|^2$$

Thus, the outer loop iterates every time the game environment *s* occurs given the possible states and the rewards of possible actions. Using this, the neural network can find the best possible score for the future states given the action, it will choose the best possible state to iterate from for future states. Every time the epsilon value for the mutation rate of the AI occurs, which in the case of the repository given by Rex L [7] is .05, then the agent has a random mutation to encourage growth through randomly exploring every once in a while. Meanwhile, the inner loop that updates the scores of the outer loop observes the future state to get the future reward given the current action and tetromino at play. The neural network takes this loop and trains off of it, giving a 2D array of the game board matrix, and is put into a convolutional neural network, otherwise known as a CNN, in which the CNN is fed the tetromino at play as the input node. The CNN has 7 possible input nodes as there are a total of 7 tetrominoes, and the output of the neural network would be the position that the input piece needs to be at to attain the best possible score that it desires after adjusting weights and learning how to play the pieces optimally. This includes the tetromino that is currently in the holding slot as well, as it is a possible input that the agent could choose to use during the game if it finds it more beneficial than the piece given by the game through the five tetromino queue. The CNN also knows about the game board state, as it is notified about what holes exist on the board, as well as the heights of each column to determine the bumpiness of the board as well. It also can know if the holes and bumps are intentional or allowed as it will use it to its advantage to score techniques like t-spins as they require the use of intentional holes.

4 Experiment Design and Results

The experiment was done using Python version 3.11.0, specifically using the Tensorflow 2.15 library to create and run the neural network. Within Tensorflow, Keras was used as the higher-level API that would take in the different parameters and adjust the weights of the nodes to give proper inputs and outputs from the deep reinforcement learning neural network. This project code was created by Rex L [7] from the GitHub repository called “Tetris_ai” [8]. The Tetris agent played a version of Tetris on Pygame in which the game was read through a series of Numpy matrices that contained the information of the game state, tetrominoes, rotations, and other game-related items. The agent uses a five-step search with the five tetrominoes that it has in the queue to determine the best places to put the pieces down on the board [9].

I trained a few different versions of the deep reinforcement learning network to have different play styles to compare the results of the training between each network. First, I created a network that plays the game with default training as the author Rex L had intended, and as per Rex’s description in his article about it, they stated that after 20 generations there

was a final working model that could play efficiently [7]. So, I compared the 20 generations of this model's first 300 pieces to some of the other models that I tuned and the results of their first 300 pieces played. The main metrics were the score that the agents had achieved after 300 pieces were played, as well as the number of lines cleared in the process to show how efficient it was with the tetrominoes it had. After the total lines cleared, the unique amount of lines cleared was also tracked through one, two, three, or four lines cleared in one go, as each clear has a different amount of points given in the game. Finally, the number of t-spins that the agents performed were also tracked, showing the amount of t-spin single line clear that were achieved (which are point equivalent to a two-line clear) and the t-spin double line clear (which are point equivalent to a four-line clear). The results are below.

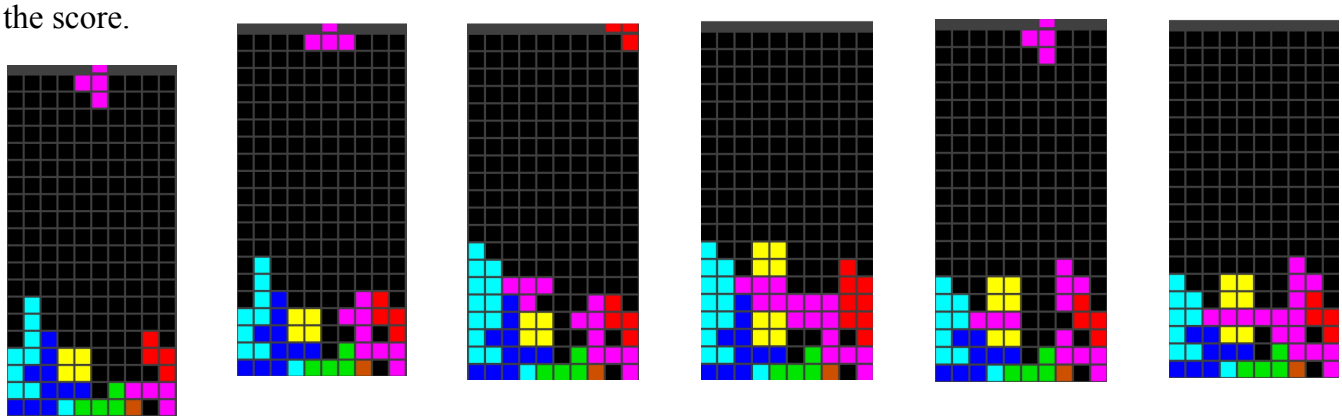
Metric \ Agent	Normal Tetris Agent	Penalize One Line Agent	T-Spin Agent
Score	2510	2770	3670
Lines Cleared	115	117	105
4 Line Clear	18	24	0
3 Line Clear	6	5	1
2 Line Clear	9	1	38
1 Line Clear	8	4	26
T-Spin Single	0	0	29
T-Spin Double	1	0	7

Table 1: Performance of the three Tetris agents with varying focuses of line clears

5 Analysis

As shown in Table 1, the results of the three different agents are obvious. The t-spin agent that focused on clearing as many t-spin lines as possible came out on top with over a 1000-point lead compared to the other two models. This is a result of the t-spins being a two-line clear and giving the same amount of points as a four-line clear, meaning that the same 100 points granted for a Tetris four-line clear is given to the t-spin double, and the 30 points granted for a two line clear is given to the t-spin single. The t-spin agent was able to get more t-spin doubles than the other agents were able to get Tetris four-line clears. The t-spin agent also utilized one and two-line regular clears as a way to initiate the t-spins, which was a good use of space to maximize the amount of points gained during the trials. Below is a series of frames from the t-spin agent that shows how the neural network was

trained to create t-spin setups using intentional holes and bumps that allowed for maximizing the score.



Here, the agent takes the holes on the board on the first frame and places a T tetromino on its tall side, creating an intentional gap on the second frame that allows for the t-spin double setup on the third frame. From there, the agent uses the pieces in the queue to fill the edge to guarantee the t-spin double and spins the T tetromino into the gap on the fourth frame. After getting the first t-spin, the agent directly takes the next opportunity to create another t-spin double setup by placing a T tetromino hanging off the top edge in the fifth frame, a common setup for hanging tetrominoes like S, Z, L, J, and T where hanging an edge off the side of another tetromino that is higher up to create the perfect t-spin gap. Finally, the agent spins the T tetromino into place, getting what would be eight lines worth of stacking up and clearing within 6 pieces of play. It is because of efficiencies like these that the t-spin focused agent can capitalize on the gaps created on the board to come up with creative t-spin setups that allow it to surpass the normal Tetris-focused agent and penalize one line agent.

Additionally, the penalize one line agent performed better than the normal Tetris agent as it had more Tetris line clears and less of the other line clears with about the same amount of lines cleared. The tweaking of the rewards for the penalize one line agent shows that encouraging the agent to stay away from doing greedy line clears and preventing non-Tetris clears resulted in a higher score. The normal Tetris agent had a total of 23 different clears that were not the optimal Tetris line clear (or a t-spin double, which seems to have happened exactly once during the 300-piece trial), while the penalize one line agent only had 10 clears that were not Tetris line clears. The penalize one line agent had better stacking that allowed for more Tetris clears, which also could have been a factor as it had more generations than the normal Tetris agent as the penalized one line agent had 71 generations. The increase in generations could also have led to better knowledge by the agent to place tetrominoes more optimally, reducing the holes and bumps in the board to make I wells like discussed in [6].

Finally, lifespan was not an issue for any of the agents. It appears that the 20-generation mark that the normal agent achieved was more than enough to produce an agent that would not immediately bring itself to its demise by randomly placing pieces. This

is also reflected in the 69 generations that the t-spin agent developed through as its difficult setups do not lead to death, as it will ensure its own life to keep getting t-spins to add to the score. Death was another penalty added to the agents during their deep reinforcement learning, and it seems that all agents have familiarized themselves with the concept of staying alive to prevent the penalty from being applied to their training reward.

5 Conclusion and Future Work

In this paper, the steps to create a neural network that understood the rules and goals of Tetris were discussed to produce an agent that maximized the score that it could reach within a certain number of pieces played, in this experiment that number was 300, and different play styles were developed between agents through incentivizing and encouraging specific moves to develop habits that would be carried across their generations. Ultimately, the agent that focused on doing t-spins attained the highest score within the 300-piece playtime as it maximized the number of points scored per line cleared.

In the future, it might be beneficial to implement real-time scoring and other Tetris elements like accelerated gravity, and snap lock-to-grid to force agents to make decisions as they come down, as well as improve the learning algorithm itself. First, the real-time scoring accelerated gravity, and snap lock-to-grid features are things that will force the agent to play with time-sensitive decision-making, as the faster the pieces start to fall, the quicker the decisions will have to be made. In this research procedure, the agents never had to worry about dying as their speed in placing pieces meant that they never had to worry about anything like gravity or the pieces snapping themselves in place and locking the position when touching the other pieces on the board. If the agent is taught how to make more lightning-fast decisions through deep reinforcement learning with fast gravity, the agent may be able to improve its way of determining how optimal a tetromino is placed as it might not be able to reach a certain position on the board as fast as it would want to, encouraging other strategies to overcome such obstacles. Furthermore, adjusting the algorithm to mutate more to find more stable beginning options like openings would ensure a path for the algorithm to identify guaranteed ways to score points as the game starts. In modern Tetris, the tetrominoes given are based on a “7-bag”, which throws all seven different tetrominoes into a bag and the game pulls from the bag until empty, which will then reset the bag to full. If the agent learns how to read a 7-bag’s possible openings, it could develop strategies based on the clean board that it has and the five tetrominoes that are given in the queue, allowing for consistent beginnings as discussed in [3]. Consistent openers would allow the agent to develop proper mid-game strategies to find known setups like the ones it knows from the opening strategies.

References

[1] History of Tetris. (2018). History of Tetris®. Tetris. https://tetris.com/history-of-tetris

[2] Modern Tetris and Classic Tetris: What is the difference? (n.d.). Tetris Interest. https://tetrisinterest.com/modern-Tetris-and-classic-Tetris-what-is-the-difference.

[3] Opener - harddrop. (n.d.). Harddrop.com. https://harddrop.com/wiki/Opener

[4] Stevens, M., & Pradhan, S. (2016). Playing Tetris with deep reinforcement learning. Convolutional Neural Networks for Visual Recognition CS23, Stanford Univ., Stanford, CA, USA, Tech. Rep.

[5] Lundgaard, N., & McKee, B. (2006). Reinforcement learning and neural networks for Tetris. Technical report, Technical Report, University of Oklahoma

[6] Wei-Tze, T., Student, J., Chi-Hsien, Y., Wei-Chiu, M., & Tian-Li, Y. Tetris Artificial Intelligence.

[7] L, R. (2021, June 21). Reinforcement Learning on Tetris [Review of Reinforcement Learning on Tetris]. Medium.

[8] zeroize318. (n.d.). Tetris_ai. [GitHub repository]. https://github.com/zeroize318/tetris_ai

[9] L, R. (2021, September 5). *Reinforcement Learning on Tetris 2*. Medium. https://rex-l.medium.com/reinforcement-learning-on-tetris-2-f12f74f70788