Tetris and the Idea of Artificial Intelligence

Susmita Mondal (20310071)
Department of Computer Science and Engineering
Indian Institute of Technology Gandhinagar
Gujarat, India
susmita.mondal@iitgn.ac.in

***Abstract*—Tetris is a tile-matching puzzle game that is among the most well-known video games, worldwide. In this game, players complete lines by pushing Tetrominoes (differently designed pieces) that fall onto the playing area. If a player manages to fill a line, it vanishes and the player earns a point. The player continues to fill in the voids with falling pieces again. When the pieces touch the top of the screen, the game is over. Tetris is a well-known benchmark for research in the machine learning and artificial intelligence (AI). This report talks about two versions of this game. In the first version, the user can play according to the rules of the game. Whereas, in the second version, the computer can play by itself utilizing the four heuristics: aggregate heights, holes, complete lines, and bumpiness. The computer tackles the puzzle intelligently to give the best possible result using the genetic algorithm.**

***Index Terms*—Genetic algorithm, Tetris, Artificial intelligence (AI), The heuristics**

I. Introduction

Tetris has been used as a domain of research in sequential decision-making under uncertainty for more than 20 years. It is widely considered a challenging domain. Alexey Pajitnov invented it in the Soviet Union in 1984. It soon became a popular culture, igniting copyright battles during the final years of the Cold War.

Tetris is a board game that is both stochastic and open-ended. From the top of the board, a piece of the block falls. The piece is selected at random from a set of seven which falls in every stages. To put the current piece in the correct position, the player can move and rotate it. After the current piece touches the ground, a new one appears at the top of the board. The blocks above it would immediately fall one move when a filled row is cleared. The objective of the game is to create as many of these rows as possible.

The standard Tetris game is often used as a test-bed for artificial intelligence research. Further research into the game has the ability to lead to important topics such as function discovery, sample-efficient reinforcement learning, and autonomous learning of action hierarchies.

II. Method

A regular Tetris board has ten rows and twenty columns. Figure 1 shows the seven blocks that make up the game usually called Tetrominos. Fall one step at a time from the top of the board to the bottom. A block is placed as it hits the bottom of the board. Tetrominos are divided into seven categories: I, J, L, O, S, T, and Z. Tetrominos can be rotated 90 degrees clockwise or counterclockwise in addition to being moved left and right. During a certain action, if all cells in a row are filled, the row is removed, and the player is awarded some points. Tetris rules are straightforward. However, a player would need a lot of practice to achieve high scores.In general, players are aware of the present and next Tetromino.

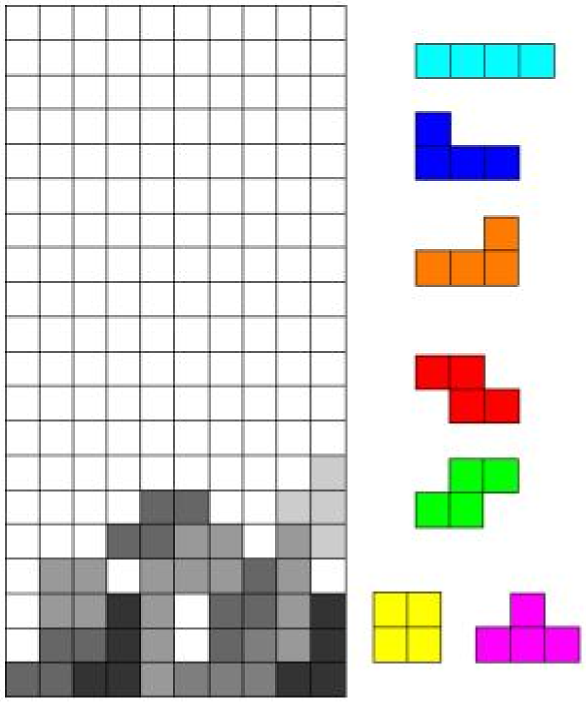

Fig. 1. A sample game board can be found on the left. The seven Tetrominoes are seen to the right. ”I”, ”J”, ”L”, ”Z”, ”S”, ”O”, ”T”

A. Applicability of the game board in AI

Artificial intelligence can run into the problem of having so many board states. A Tetromino can be arranged on the game board in a variety of ways, using various rotations and translations. Some of these placements are beneficial, while others are not. The AI must try every possible placement before deciding which are good and which are bad. Figure 2 depicts one of the possibilities of how this is obtained.

The scoring system is the most important aspect of AI. The game would be lost quickly if the system is selected improperly. The AI must be capable of predicting future events in people to sustain as long as possible.

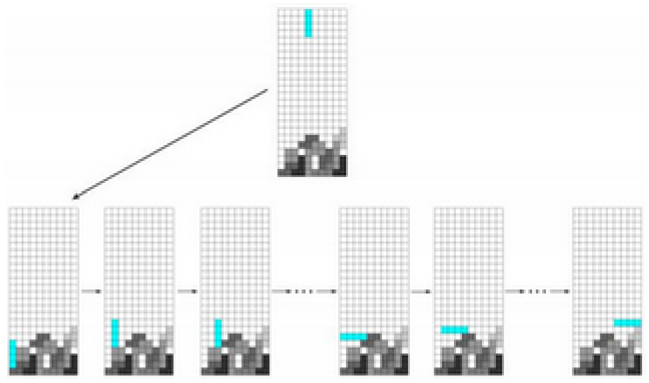

Fig. 2. The AI iterates over all of the different rotations and translations before it finds the best one

III. Related Work

Tetris brings several research challenges. To begin with, minor variations in the game's implementation causes significant differences in results. It is indeed impossible to compare scores from various review papers because of this. Second, the game's result has a wide range of possibilities. To reliably measure average results, a significant number of games must be completed. In addition, games will take a long time to finish. Researchers who have created algorithms that can play Tetris fairly well have had to wait for days to finish a single game. As a result, working with grids smaller than the traditional grid-scale of 20x10 is common. Using a grid size of 10x10 is a practical way to make the game shorter without losing its essence.

A. Initial Attempts

- Tetris was used as a test-bed for large-scale feature-based dynamic programming by Tsitsiklis et al. (1996). They used two basic state characteristics: the number of holes and the highest column's height. On a 16 x 10 grid, they got a score of about 30 cleared lines [1].
- Bertsekas et al. (1996) introduced two new variables: the height of each column and the height differential between adjacent columns. They got a ranking of about 2,800 lines using lambda-policy iteration. However, their method of finishing the game effectively shrinks the grid to 19 x 10 [2].
- Using the same features as Bertsekas et al. (1996), Kakade (2002) used a policy-gradient algorithm to clear 6,800 lines on average [3].
- Ramon et al. (2004) obtained a score of about 50 cleared lines using relational reinforcement learning with a Gaussian kernel [4].
- For a linear programming solver, Farias et al. (2006) explored an algorithm that samples constraints in the form of Bellman equations. Using Bertsekas et al. (1996) features, the solver finds a policy that clears about 4,500 lines [5].
- Romdhane et al. (2008) used patterns of small sections of the grid to merge reinforcement learning and case-based reasoning. Their totals were also about 50 lines cleared [6].

B. Genetic Algorithms

- Bohm et al. (2005) developed a Tetris controller using evolutionary algorithms. The agent knows not only the falling Tetrimino but also the next one in their implementation. As a consequence, their outcomes are incomparable to those obtained by some current Tetrimino models. They developed a policy that was both linear and exponential. On the regular grid, they cleared 480,000,000 lines with the linear function and 34,000,000 with the exponential function. They added new features including the number of connected holes and a number of occupied cells weighted by the height. These extra characteristics were not identified in subsequent research [7].
- Szita et al. (2006) used the cross-entropy algorithm and were able to clear 350,000 lines. The algorithm looks for the linear policy that maximises the score by probing random parameter vectors [8].
- Thiery et al. (2009) improved the BCTS controller by using the cross-entropy algorithm and adding a couple of functions (hole depth and rows with holes). BCTS stands for building controllers for Tetris. They had a total of 35,000,000 lines cleared on average. BCTS won the 2008 Reinforcement Learning Competition with the addition of a new function, pattern diversity [9] [10].
- Boumaza et al.added a new evolutionary algorithm to Tetris in 2009 called the covariance matrix adaptation evolution strategy (CMA-ES). He noticed that the resulting weights were very similar to Dellacherie's and that they cleared an average of 35,000,000 lines [11].
- Salimans et al. (2017) describe the success of genetic algorithms which is the recent resurgence of evolutionary techniques as powerful competitors for reinforcement learning algorithms [12].

IV. Tetris: First Version

Regularities of various types can be seen in real-world problems. As a result, it's reasonable to predict regularities in the decision-making challenges that arise when playing video games like Tetris. Several rules have been announced since the Tetris game has been evolving for nearly three decades. In this report, the first version follows game rules of Alexey Pajitnov, which are used in Tetris Battle.Figure 3 shows the game-play of the first version. The below are the basic rules.

A. Matrix Configuration

- The playing field is rectangular in shape, with 10 columns wide and 20 rows in height.
- Starting from the beginning, each of the seven pieces includes a different kind of Tetrimino in random order. That is, a specific form of Tetrimino must appear in the following seven pieces.
- Before dropping a Tetrimino to the base of the playing area, the player can adjust the shape of the piece, switch

left and right, and use the arrow keys to make the piece drop faster.
- The Tetrimino form of the next pieces could be seen by the player.

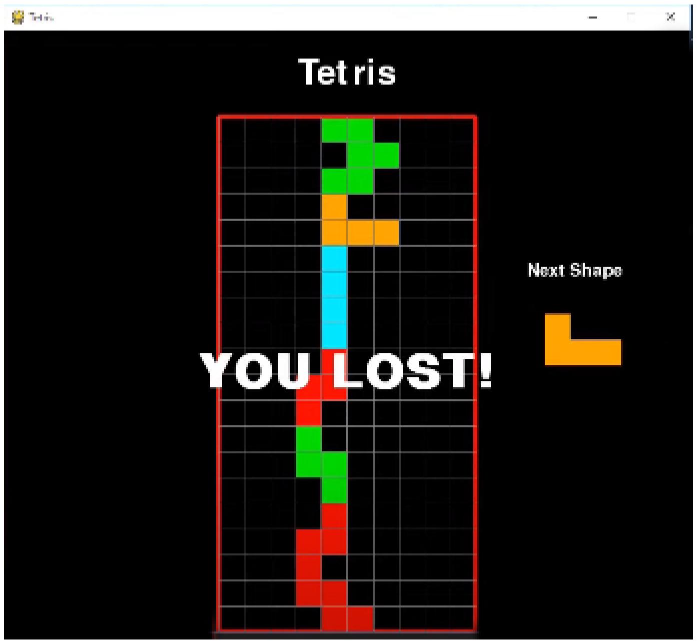

Fig. 3. The Game-play of the First Version

B. Game Configuration:

The first version is coded in Python and the library ”pygame” is used to create an interactive user interface (UI). The number of functions that are used to create this game is listed below.

- **class Piece:** The seven pieces are created, the rotated version of the shapes is also created. The shapes are given distinct colours. Figure 4 describes the possible rotations of the particular shape.
- **def create-grid:** The playing area including the borders and free space is determined.
- **def convert-shape-format:** Shape movements are made limited by this function.
- **def valid-space:** The playing area is created.
- **def get-shape:** By importing the ”random” library, shapes are drawn randomly from the top of the playing area into the grid.
- **def draw-grid:** Grid for playing area is created i.e. in 10 x 20 pixel.
- **def clear-rows:** Once a row is fully filled, it is cleared from the playing area.
- **def draw-next-shape:** The initialization of a new shape after the previous shape touches the ground or any neighboring shape. This function also helps to see the next shape for the user, in the free area of the game space.
- **def main:** This is the main function from where the game is controlled. The falling time of a shape is determined here, movement of arrow keys are specified here, lock position of a shape is made when it touches the ground or the neighboring shape.
- **def main-menu:** Text to display begin and end of a particular game.

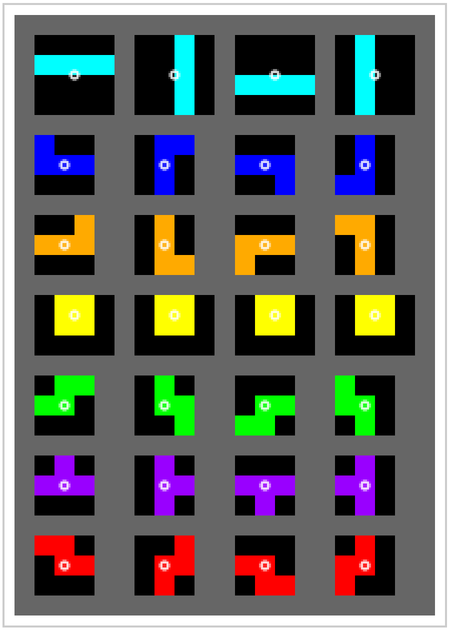

Fig. 4. All The Possible Rotations

This is the version of a single user game without AI. It can be modified in different ways. An ample amount of modifications can be made on this code like adding scores, keeping the track of the highest score, and adding high-quality graphics, etc.

V. Tetris: Second Version

The second version of this game introduces AI. The goal is to remove as many lines as possible, and therefore make as many moves as needed. To achieve this aim, the AI will test out all of the potential moves for a given Tetris piece before deciding on the right one (rotation and position). It scores on the each possible move (including the look-ahead piece) and chooses the one with the highest score as its next move.Google Chrome had been used to test the AI version of the game, which is coded in JavaScript (also works fine in other browsers).

A. The Heuristics

The score for each move is determined by examining the grid after a position is locked of a shape. This calculation is based on four heuristics: aggregate height, complete lines, holes, and bumpiness, which the AI can aim to maximize or minimize.The game-play of the second version is shown in the Figure 5.

- **Aggregate Height:** This heuristic informs about the grid’s ”height.” The aggregate height is calculated by adding the heights of each column (the distance from the

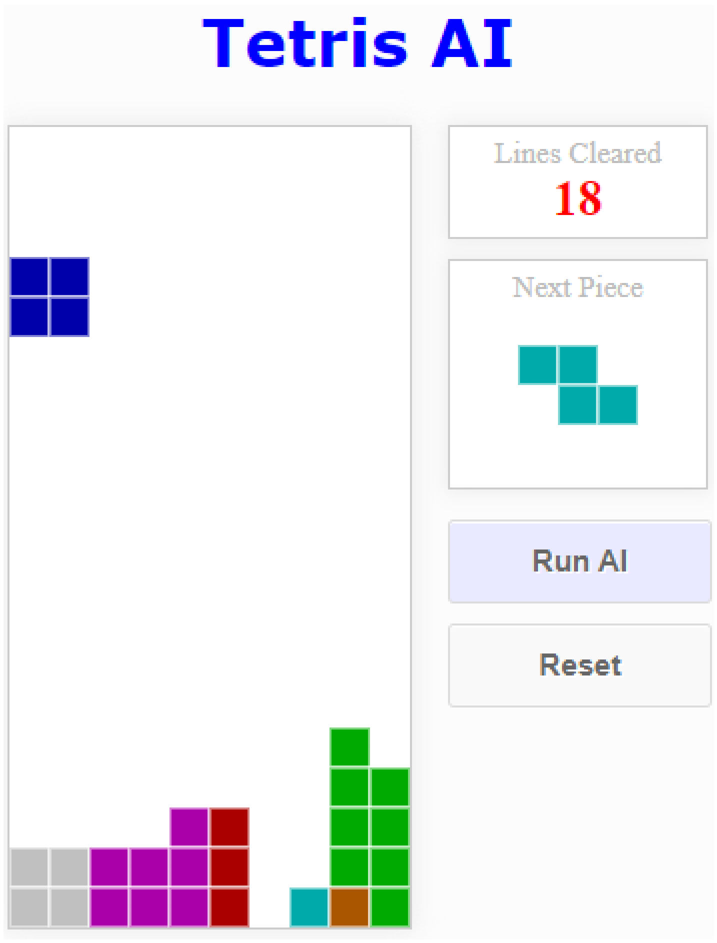

Fig. 5. Game-play of Tetris Second Version with AI

highest tile in each column to the bottom of the grid). This version tries to keep this value as low as possible because a lower aggregate height allows to drop more pieces into the grid before touching the grid's top. Figure 6 describes aggregate height of a grid. Figure 6 shows the aggregate height of a grid.

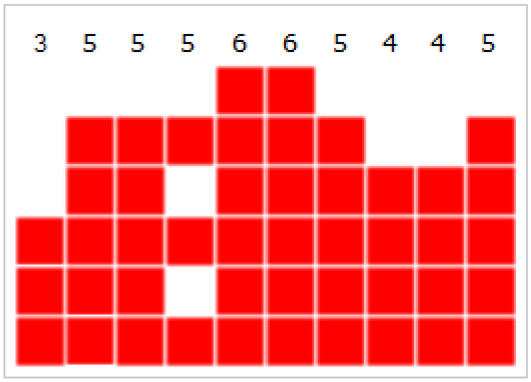

Fig. 6. Aggregate Height = 48

- **Complete Lines:** Among all of the four heuristics, this is perhaps the most intuitive. It basically refers to the total number of lines in a grid. Clearing lines is the objective of the AI, and clearing lines allow more room for more pieces. So, the number of complete lines is to be increased.Figure 7 shows complete lines in a particular grid.

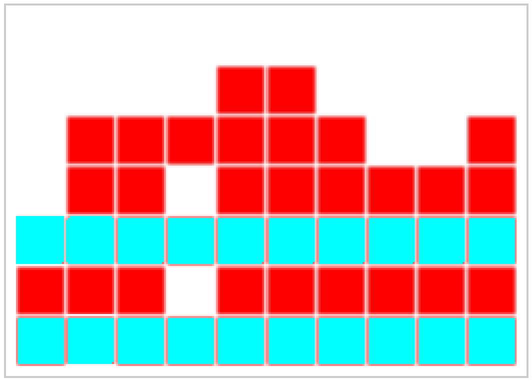

Fig. 7. Complete Lines = 2

- **Holes:** A hole is known as space over which at least one tile from the same column is missing. A void is harder to remove. The AI first needs to clear all of the lines above it before approaching a hole and fill it. As a result, AI needs to keep these gaps to a minimum.Figure 8 describes the presence of holes in a grid.

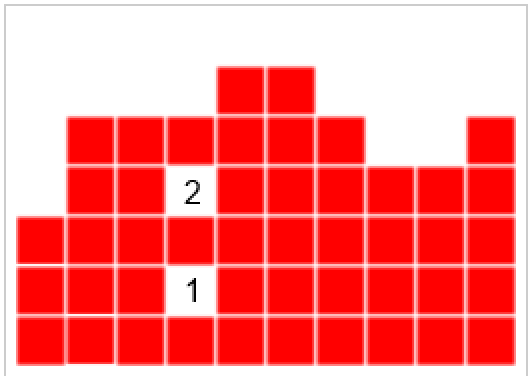

Fig. 8. Number of Holes = 2

- **Bumpiness:** Consider the situation when the grid develops a deep "well" that makes it unplayable. These "well"s mean that lines which can be easily cleared, are not being removed. If a well is needed to be covered, it would be difficult to remove any of the rows that the well spans. This version describes the heuristic called "bumpiness" to generalize the concept of a "well." The AI would aim to keep this value as low as possible to keep the top of the grid as uniform as possible. Figure 9 describes a "well" and Figure 10 shows the method by which bumpiness is measure.
- **Putting the Heuristic together:** The linear combination of the four heuristics is used for calculating the grid score. The following formula provides a score function:

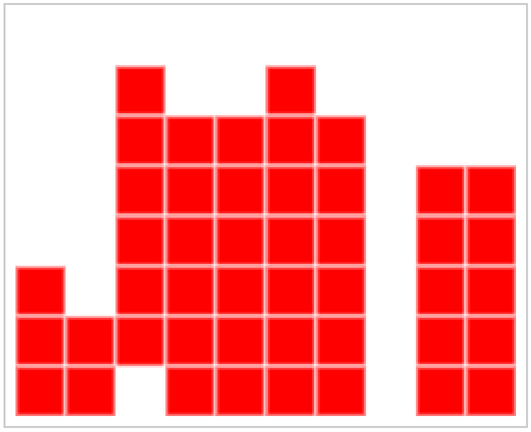

Fig. 9. An Unexpected "Well" in a Grid

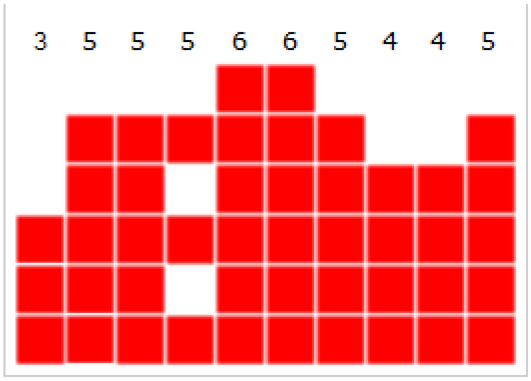

Fig. 10. Bumpiness = 6

a*Aggregate Height+b*Complete Lines+c*Holes+d*Bumpiness

where a, b, c, and d are constants. AI needs to minimize aggregate height, holes, and bumpiness, so a, c, d is to be negative. Similarly, AI needs to maximize the number of complete lines, so b is to be positive.

B. The Genetic Algorithm

A possible set of parameters (a, b, c, d) can be represented as a vector in R^4. For real-valued genes, a typical Genetic Algorithm will require searching the entire solution space for the best set of parameters. However, for this initiative, AI needs to consider points on the unit 3-sphere (i.e. 4-dimensional unit sphere). This is because the score function is linear. Figure 11 describes how to score function is represented in the context of a vector.

Both parameter vectors pointing in the same direction yield the same results. Each direction only needs to accept a single vector. This justifies restricting the search to points on the surface of the unit 3-sphere since the sphere's surface covers all potential directions.

- **Fitness function:** With 1000 unit parameter vectors, the population is first initialized. AI runs for 100 games with no more than 500 Tetromino pieces for each parameter variable (i.e. 500 at most moves).

$$f(\vec{x}) = \vec{p} \cdot \vec{x}$$

$$\vec{p} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}, \vec{x} = \begin{pmatrix} \text{Aggregate Height} \\ \text{Complete Lines} \\ \text{Holes} \\ \text{Bumpiness} \end{pmatrix}$$

Fig. 11. Equation for Parameter Set

- **Tournament selection:** Tournament collection is used to pick parent individuals for reproduction. AI randomly chooses 10 percent (= 100) of the population, and the two fittest individuals in this sub-pool produces new offspring by crossover. This cycle is replicated until the number of new offspring touches 30 percent of the population size (=300).
- **Weighted average crossover:** On the unit 3-sphere, the offspring vector lies in between the two parent vectors but tends towards the fitter parent by a factor that grows with the difference in fitness between the two parents. This represents the crossover operator's preference for the fitter parent.
- **Mutation operator:** Each freshly developed offspring has a 5 percent risk of being mutated. If it does not mutate, the offspring vector's random component is modified by a random value of up to 0.2. After that, the vector is normalized. Since dealing with unit spheres, the offspring vectors to be rotated by a certain degree. 4D rotations, on the other hand, are much more perplexing than 3D rotations. As a result, the AI uses a simpler mutation operator.
- **Delete-n-last replacement:** When the number of offspring produced exceeds 30 percent (= 300) of the initial population size, the population's worst 30 percent of individuals are deleted and replaced with newly produced offspring, creating the population's next generation. The population distribution remains unchanged (1000). The procedure will then be repeated until the population is sufficiently fit, at which point the algorithm will terminate.

VI. Conclusion

Two different aspects of the Tetris game are described throughout the report. one of them is without AI and another one is using AI. In the first version, the user has to control the pieces using arrow keys, whereas in the next version the computer can configure the moves intelligently by itself. This work tends to assume that, using a genetic algorithm to create an artificially intelligent agent for Tetris is a reasonable solution. This logic can be applied to any task in which a state-space search can be performed and each state can be outlined as a value. With a greater value being better - for example, in Tetris, each state can be summarized as the score at that state or the number of lines cleared to get to that state.

VII. Future Scope

Since the game only has four elements, it will be unable to handle any unique situations. To have a clearer and more realistic representation of the game, one or two more features should be included.

- Some of the heuristics were found to be redundant, while others needed to be replaced by better heuristics (for example, the column height heuristic) for better performance.
- AI was programmed to follow a specific set of rules. A particular set of rules may or may not achieve the same results.
- A naive random piece generator can instead result in a ridiculously long series of S or Z tiles. It will increase the complexity. For future work, the AI can improve if properly tuned.

VIII. Acknowledgment

The first version of the Tetris without AI is been adopted from ”https://www.youtube.com/watch?v=zfvxp7PgQ6c”. The basic code of this tutorial is taken and the scoring system is discarded. The second version of Tetris with AI is adopted from ”https://github.com/LeeYiyuan/tetrisai”.

References

[1] Tsitsiklis, John N., and Benjamin Van Roy. ”Feature-based methods for large scale dynamic programming.” Machine Learning 22.1 (1996): 59-94.

[2] Bertsekas, Dimitri P., and John N. Tsitsiklis. Neuro-dynamic programming. Athena Scientific, 1996.

[3] Kakade, Sham M. ”A natural policy gradient.” Advances in neural information processing systems 14 (2001).

[4] Ramon, Jan, and Kurt Driessens. ”On the numeric stability of gaussian processes regression for relational reinforcement learning.” ICML-2004 Workshop on Relational Reinforcement Learning. 2004.

[5] Farias, Vivek F., and Benjamin Van Roy. ”Tetris: A study of randomized constraint sampling.” Probabilistic and randomized methods for design under uncertainty. Springer, London, 2006. 189-201.

[6] Romdhane, Houcine, and Luc Lamontagne. ”Reinforcement of Local Pattern Cases for Playing Tetris.” FLAIRS Conference. 2008.

[7] Böhm, Niko, Gabriella Kókai, and Stefan Mandl. ”An evolutionary approach to tetris.” The Sixth Metaheuristics International Conference (MIC2005). 2005.

[8] Szita, István, and András Lörincz. ”Learning Tetris using the noisy cross-entropy method.” Neural computation 18.12 (2006): 2936-2941.

[9] Thiery, Christophe, and Bruno Scherrer. ”Improvements on learning Tetris with cross entropy.” Icga Journal 32.1 (2009): 23-33.

[10] Thiery, Christophe, and Bruno Scherrer. ”Building controllers for Tetris.” Icga Journal 32.1 (2009): 3-11.

[11] Boumaza, Amine. ”On the evolution of artificial Tetris players.” 2009 IEEE Symposium on Computational Intelligence and Games. IEEE, 2009.

[12] Salimans, Tim, et al. ”Evolution strategies as a scalable alternative to reinforcement learning.” arXiv preprint arXiv:1703.03864 (2017).