# Development of a Tetris Playing Agent in Java

**Ömer Nebil Yaveroğlu**

Department Of Computer Engineering
Middle East Technical University
nebil@ceng.metu.edu.tr

**Abstract**

In this paper, an artificial intelligence algorithm is developed for implementing a Tetris playing agent. Tetris is a well-known board game in which the aim is to place randomly falling blocks of different shapes on a game board in such a way that the placed blocks form a line. When a line is formed, the blocks in this line are removed giving some points to the player. An agent playing this game is developed in this study by seeing the problem as a constraint satisfaction problem. By computing a score for some possible placements of a given block and then performing a search on the resulting scores, an action decision is made. In order to achieve simplicity on the number of possible placements, an ordering of possible actions to be performed is made. As a result, an agent playing Tetris in a reasonable way is implemented. On average, 4471 blocks can be placed without filling the board in the final version of the agent.

## 1. Introduction

Tetris is a falling blocks game in which the player aims to prevent filling the game play area. Pieces of seven different orientations (Figure 1) appear randomly in the game play area [1]. You aim to form lines by these pieces on game board by rotating and moving the pieces left and right while they are falling. When a line is filled, it is cleared from the game play area and all blocks above it fall one level down. The longer you are able to play by this way without filling the game play area, the better scores you can achieve. So the aim is to maximize the game duration trying to remove lines of blocks from the game board.

**Figure 1: Seven different orientations of pieces**

There are many competitions about game playing in the areas of artificial intelligence and machine learning. Tetris is a popular game for these competitions because of its simple nature and the huge number of possibilities that it includes. RL-Competition [2] is one of these competitions which require its competitors to implement a reinforcement learning algorithm for game playing. The framework that they provide for competition purposes, RL-Viz, is used during the implementation of the Tetris playing agent. The only necessary thing to implement the Tetris playing agent with this framework is to develop the agent. All the other issues like game logic, graphical interface are handled by the framework.

The approach followed during the implementation of the agent is a straight-forward search algorithm. After considering different possible placements, the best placement is chosen. A score is computed for each possible placement by considering some scoring parameters. Namely these parameters are pile height, number of holes, surface area, altitude difference, number of blocks, weighted block number, number of removed lines and row-column transition number. A linear function is formed with these parameters to find out how good a placement is.

There are many different methods in literature proposed for Tetris playing. Even some very complex algorithms such as genetic algorithms are used in these studies. It is not possible to compare my algorithms directly with other studies since our scoring mechanisms differ most of the time. But I can easily say that even with a basic algorithm very promising results can be achieved. I have used an algorithm evaluation metric which takes the number of blocks entering into the game play area in a single game. This scoring also gives some direct idea about the length of a single game duration. After the optimizations performed, 4471 blocks can enter the game play area on average in the best case. This is a good score since performing random actions result with an average score of 90.

## 2. Related Work

Tetris is a well-known game since mid-1980s. Because of this, there have been many efforts in order to find an optimal solution to the game. The game also forms a very competitive area to work on since it is rather simple and there is no limit of improvement. Because of these reasons, there are many papers related to Tetris playing. Among this variety of papers, the most promising papers were the

Breukelaar *et al.*'s [3], Böhn *et al.*'s [4], Hoogeboom *et al.*'s [5], Szita *et al.*'s [6] and Farias *et al*'s [7] papers. The focus of Breukelaar *et al.*'s paper [3] is proving the NP-Completeness of Tetris goals and NP-Hardness of making approximations about the goals in Tetris. In Hoogeboom *et al.*'s paper [5], some decidability questions related to Tetris is tried to be answered. These two papers answer some questions about the theory of Tetris.

The other three papers are more focused on solutions for Tetris playing. The Böhn *et al.*'s paper [4] proposes a solution to where and how to place a falling piece. They have used a genetic algorithm for this purpose. They calculate a rating for all possible moves with the next two coming blocks and make the move which gives the best rating score. They perform some forward checking for this reason. They have removed 24311 rows with a linear heuristic and 68421 rows with an exponential heuristic which are much better than the results that could be reached in this study. It is one of the best solutions in the area of Tetris playing. It should be kept in mind that they use an advanced technique, genetic algorithms and they perform forward checking which improves their results a lot. The agent developed in my study makes decisions with just the given block and so no future predictions can be made in my implementation. It would be possible to make some comparisons between my approach and their approach if I could have some information about their performance with no forward checking version of their algorithm. I have improved some ideas given in this paper. Some parameters of my scoring mechanism match with the ones that they mentioned in their paper. So this paper was important for me because of the ideas it gave to me.

Szita *et al.*'s paper [6] uses Tetris as a benchmark for their noisy cross entropy method that they have developed for using with reinforcement learning. There were no clear performance measures of this paper as the Böhn *et al's* paper had but they mentioned that their paper's performance is worse than the Böhn *et al's* solution. This paper has also proposed some scoring parameters as Böhn *et al* did, giving me some intuition on forming my scoring parameters.

In Farias *et al*'s paper [7], an approximate dynamic programming algorithm is proposed for playing Tetris. According to this paper, Tetris can be reduced to stochastic control problem and so it is solvable by dynamic programming. They use randomized constraint sampling while applying the dynamic programming algorithm for simplifying the computation process and for reducing computational complexity of dynamic programming. The best score that they could achieve is between 4000 – 4900 cleared rows which are more or less similar to my current results. The idea of checking the possible actions in an order was a result of the randomized constraint sampling mentioned in this paper. The main advantage of my approach compared to their approach is that my algorithm is more efficient because of their dynamic programming usage.

# 3. Problem Definition and Algorithm

## 3.1. Task Definition

Tetris is a simple board game in which given a falling piece, the aim is to place this piece to the most suitable place, in order to remove lines of blocks from the board. So an agent developed for playing Tetris should be able to get the current board situation and the falling piece type as input and return the best action to maximize the number of removed lines from the board as the output. There can be seven different types of falling pieces. The piece types are named left gun, left snake, right gun, right snake, line, square and t-shape in the order of appearance in Figure 1. There are also six different actions that can be performed. These actions are going left, going right, rotating clockwise, rotating counter-clockwise, dropping one row down and dropping to the bottom. The first four moves also perform dropping one row below automatically. For example, if you perform go left action the center of your piece falls down one row and moves one column left. So placing a block to a suitable place requires some number of steps to be performed. A piece cannot be placed in a preferred location directly.

The competitive environment of a Tetris as a result of its scoring mechanism made it a popular game for over 20 years. Competitions are being held on this domain because of the great interest of people on Tetris. It is also a good area to make studies especially in artificial intelligence and machine learning since it allows probabilistic reasoning and strategy development. In fact these are also the reasons for the development of the project forming this paper.

## 3.2. Algorithm Definition

The algorithm of the developed agent performs a search on the possible placements of the falling block and chooses the best in them. A placement can be made as a result of a series of movements. The agent creates an action sequence for each newly appearing falling piece. Then it performs the created action sequence returning its values one-by-one to the framework at each of the iteration steps.

Creation of the action sequence is the most important part of the algorithm. For each falling piece, some placements are scored according to a scoring function. The action sequence giving the best score is kept to be performed. For simplifying the calculation of the possible placements, the possible actions are checked in an order. All the possible placements that can be performed as a result of first rotating, then moving left or right and finally dropping to the bottom are considered. Assume a line shape has appeared. The action sequences checked for these pieces are as in Table 1. For all seven different types of falling blocks, this order is followed and all possible action sequences with this ordering are considered. For square shape 9, for line shape 17, for left snake 16, for right snake 16, for t-shape 36, for left-gun 30 and for right

gun 30 possible action sequences are considered in this way.

| Rotation Movements | Movements of left and right | Dropping |
|---|---|---|
| - | L , L , L | D |
| - | L , L | D |
| - | L | D |
| - | - | D |
| - | R | D |
| - | R , R | D |
| - | R , R , R | D |
| CRT | L , L , L , L , L | D |
| CRT | L , L , L , L | D |
| CRT | L , L , L | D |
| CRT | L , L | D |
| CRT | L | D |
| CRT | - | D |
| CRT | R | D |
| CRT | R , R | D |
| CRT | R , R , R | D |
| CRT | R , R , R , R | D |

**Table 1: Action sequences checked for line shaped falling piece (CRT: Clockwise Rotation, L: Move Left, R: Move Right, D: Drop)**

Each of the controlled action sequences are scored with a linear function. The scoring function consists of 8 parameters. These parameters are:

- Pile Height: It is the row of the highest occupied cell on the board. This value should be kept as small as possible since the game ends when the pile height becomes equal to the board height. (See Figure 2)
- Hole Count: It is the number of all unoccupied cells that have at least one occupied cell above them. This value should be kept as small as possible since holes cause an increase in the pile height and cannot be removed easily. (See Figure 2)
- Removed Lines: It is the number of lines that are cleared as the result of the last action. This value will be important in determining the outcomes of placing a block to a position. It should be maximized to increase the game duration and score.
- Altitude Difference: It is the difference between the highest occupied and lowest free cell. It is important for avoiding creation of a pile in one corner of the board and leaving other corner empty. The evaluation function will try to keep this value minimized. (See Figure 2)
- Blocks: It is the number of blocks that exist on the board. This value will be kept as small as possible to avoid filling the game board.
- Weighted Blocks: It is a score given depending on the number of occupied cells on the board. The weight of a block comes from the height it occupies. The higher blocks are penalized more when compared to lower ones. So this value is tried to be kept small. This parameter is especially useful when the board is empty since it discriminates the orientation differences of a falling piece.
- Surface Length: It is the number of borders that a falling piece can be placed on or near. In other words, it is the number of block borders that are not in a hole. (See Figure 2) Its value should be minimized in order to achieve some compactness of placement.
- Number of transitions: It is the total number of block borders. In other words, it is the number of transitions from an occupied area to unoccupied area and vice versa for each row and column. This parameter can also be thought as the total number of borders in the holes plus the surface length parameter.(See Figure 2) This value is the most deterministic parameter of the scoring function because of its' success in placing the pieces in the most compact way. This value should be minimized in order to achieve compactness and avoid holes.

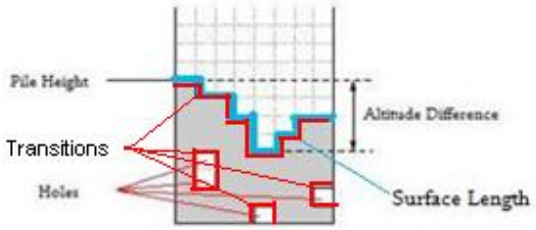


**Figure 2: Representation of Pile Height, Altitude Difference, Surface Length and Holes**

While keeping the aims mentioned for each parameter in mind, these parameters are combined to form a linear scoring function. In order to equalize their effect on the function, their scores are normalized to the range [0,100]. Then these normalized values are combined as a linear function as below.

$$f(\text{Score}) = a_1 * n_1 + a_2 * n_2 + \ldots + a_8 * n_8$$

In this formula, $a_i$ values represent the constants that define the weight of the parameter. The $n_i$ values represent the normalized parameter values. The $a_i$ values are optimized by experimenting with different values in a numerous tries. A huge amount of improvement is achieved as a result of this optimization. The resulting scoring functions are applied for all possible action sequences and the highest score is taken as the best action to be performed.

## 4. Experimental Evaluation

### 4.1. Methodology

As a performance metric, RL-Competition framework uses the number of blocks entering the game board during a game. This measure also gives some idea about the duration

of a game. This measure is directly computed by the RL-Competition framework and represented by three numbers, E/S/T. (Figure 3) The E value shows which run is being performed. The S value represents the number of blocks which entered to the game board at current run. T the value represents the total number of blocks which entered to the game board at E runs.

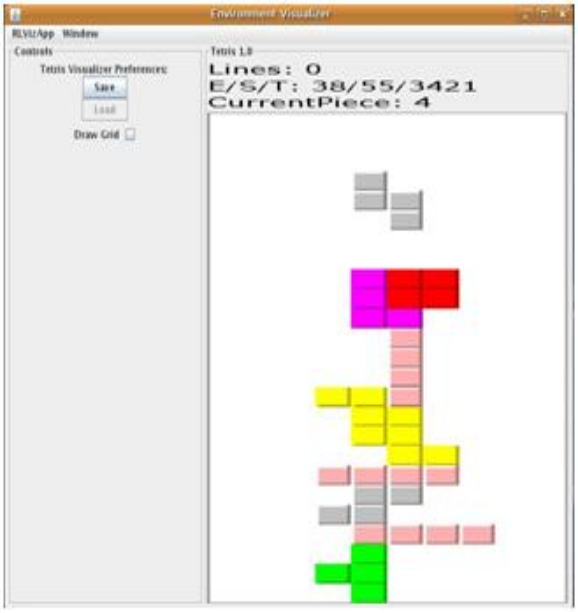


**Figure 3: A screenshot of the framework showing a run performed by a random agent**

The scoring mechanism for algorithm evaluation is extracted from these values with the following formula.

$$\text{Average Score} = (T - S) / (E - 1)$$

For example the score of the board situation given in Figure 3 is (3421 - 55) / (38 - 1) = 90.97.

This scoring mechanism is meaningful since it gives the average number of blocks that can appear on the game board without filling the board. Most of the measurements performed in other studies use the number of lines removed with a given number of blocks as the evaluation score of an algorithm. It may be parallel to the aim of the game but I don't think it is a good scoring metric for measuring the performance of an agent. That method may give good results in a case like iteratively removing one line of blocks on an empty board and filling the board to get an empty board again. But the evaluator function should measure the compactness of the solution. It should also address the duration of the game that can be played without filling the board. The scoring mechanism that is used in this study allows the measurement of these factors.

## 4.2. Results

Defining the parameter constants of the scoring function is the most time consuming part of the development process. After the optimization process, the constants of the parameters giving the highest evaluation score are defined as in Table 2.

| | |
|---|---|
| Pile Height | 3 |
| Hole Count | 40 |
| Removed Lines | 2 |
| Altitude Difference | 1 |
| Blocks | 1 |
| Weighted Blocks | 1 |
| Surface Length | 1 |
| Number Of Transitions | 35 |

**Table 2: Optimized constants for scoring parameters**

As can be seen from the table, the most dominant parameters of the scoring function are the hole count and the number of transitions. So the main aim of the algorithm is achieving compactness and minimizing the hole count. When run with these parameters, the board situation in Figure 4 can be achieved.

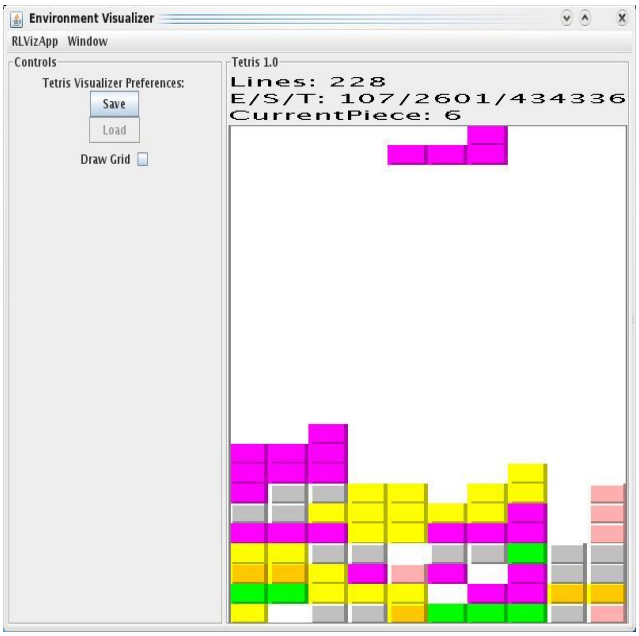


**Figure 4: A screenshot of the framework showing a run performed by the implemented agent**

As can be seen from Figure 4, the average evaluation score of my agent for this experiment is (434336 – 2601) / (107 - 1) = 4072,97. After performing some number of experiments, the highest achieved score with these parameters is found to be 4471.

### 4.3. Discussion

The implemented agent tries to maximize the compactness of the placements trying to avoid holes. For this aim, the algorithm has a strong architecture. But for some extreme cases like a case in which no line shape appears, the algorithm may not perform well. Sometimes it requires line shape for the removal of lines.

Another weakness of the algorithm is the ordering of actions. The ordering of actions prevents the agent to make moves like filling a hole which is open on one side. If the algorithm performed actions without an order, compactness of placements may increase and the number of holes created may decrease. But considering all the moves with no ordering would produce unmanageable number of possible placements.

On the other hand, the main advantage of the algorithm is its' simplicity. Even with the limited amount of movements and the simple scoring function, good results can be achieved. The most important features are the number of holes and the number of transitions. The other parameters of the scoring function are useful for making decisions at non-differing cases like empty board. This shows that the key point of getting high scores in Tetris is avoiding holes.

## 5. Conclusion

In this paper, the process of developing a Tetris playing agent is presented. The algorithm makes a scoring on the possible placements and chooses the best in them to perform. An ordering is defined for the computation of possible placements. In this ordering, the rotations are considered first. Then moving the piece left or right and finally dropping the piece are considered. For each of the possible placements, a score is computed. The action sequence giving the best score is performed. An evaluation mechanism based on the number of blocks appearing in the game is used to evaluate the algorithm developed. The agent has achieved the score 4471 in the best case while a random agent could make at most 90. This shows the huge amount of improvement achieved. There are many proposed methods for Tetris playing. Some of them even use some advanced methods like genetic algorithms and dynamic programming. When compared to them, my agent produces worse results. But the results are important in the manner that it is even possible to achieve good results with simple algorithms. The development process also shows the effects of different scoring functions.

## 6. Future Work

The current algorithm is based on an ordering of actions. This ordering is necessary since it is not possible to compute all the possible placements of a falling piece if all actions are allowed. This ordering makes the number of possibilities manageable. On the other hand, the algorithm doesn't have the capability of filling unclosed holes since it doesn't have moving left or right capability. In the current version of the algorithm dropping one row below action is never used in action sequences. If this action is introduced to the algorithm and the possibilities can be increased by considering moving left or right action as the last action, an improvement can be achieved.

As far as I have observed, the variety of parameters of the scoring function is enough for computing a reasonable score for a placement. But it could be further improved by adjusting the constants of these parameters more. Some machine learning algorithms may be introduced for the optimization of the multiplication constants. These constants may also be defined depending on the falling piece type. The easiest and simplest adjustment that can be made on the agent is just playing with the parameters.

## References


[1] "The Tetris game in wxWidgets". http://zetcode.com/tutorials/wxwidgetstutorial/thetetrisgame/

[2] RL-Competition Web Site. http://rl-competition.org/

[3] Ron Breukelaar, Erik D. Demaine, Susan Hohenberger, Hendrik Jan Hoogeboom, Walter A. Kosters, David Liben-Nowell. 2003. Tetris is Hard, Even To Approximate. International Journals of Computational Geometry & Applications

[4] Niko Böhm, Gabriella Kokai, Stefan Mandl. 2005. An Evolutionary Approach to Tetris. MOC2005

[5] Hendrik Jan Hoogeboom, Walter A. Kosters. 2004. Tetris and Decidability. Elsevier

[6] Istvan Szita, Andras Lörincz. 2006. Learning Tetris Using the Noisy Cross-Entropy Method. Neural Computation 18,2936-2941

[7] Vivek F. Farias, Benjamin Van Roy. Tetris: A study of randomized constraint Sampling. Stanford University