

**Tetris: A Heuristic Study**

**­**A study of heuristic scoring methods for playing Tetris

Max Bergmark

Degree Project in Computer Science, SA104X  
Supervisor: Örjan Ekeberg

**Abstract**

A heuristic technique is an approach to problem solving which results not in the optimal or perfect solution, but a solution that is sufficient for the immediate goals. For some problems, finding an optimal solution is either impossible or impractical, and heuristic methods can provide an answer quicker.

In this paper, a heuristic approach to playing Tetris is discussed, and results in an AI which can play about 90000 lines before losing. This is about 10% of the score of the best AI currently.

1. Introduction

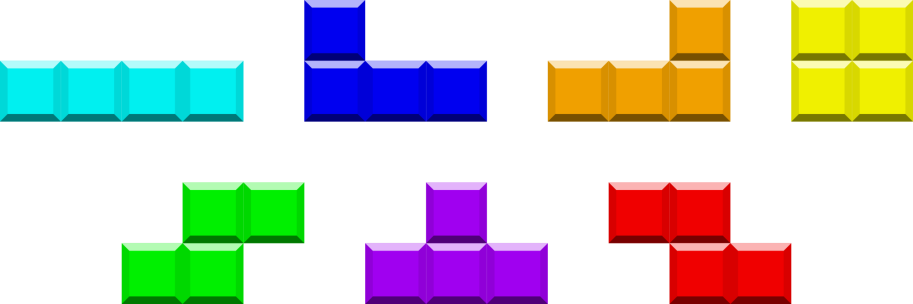
In order to create an artificial intelligence (AI) for Tetris, we need to analyze the game board. Each cell of the board can be filled or empty, and a standard Tetris board is 10 cells wide and 20 cells high. Since we have seven tetrominoes, the total number of possible game boards becomes. We cannot possibly analyze every one of these states, and thus we apply a heuristic model to the problem.

1. Background

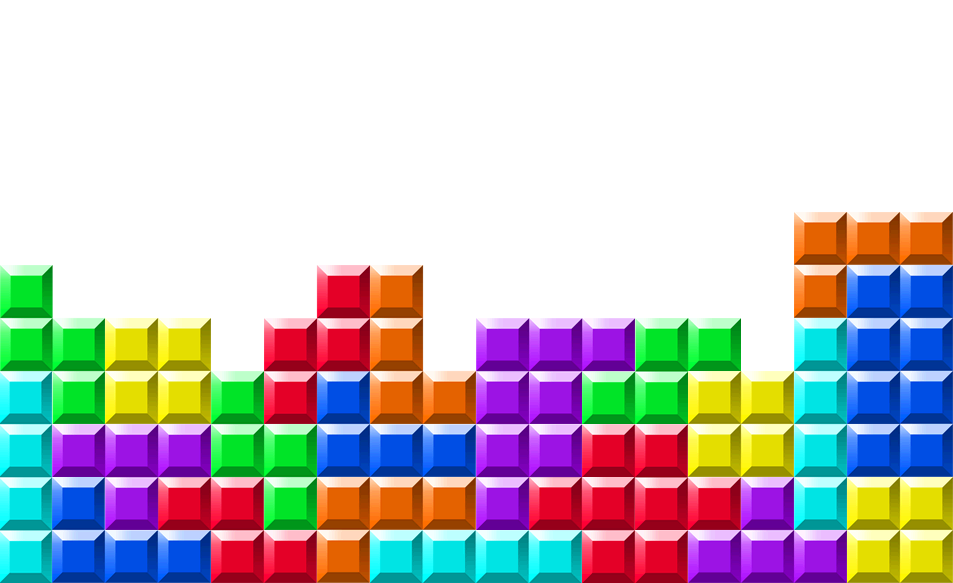
Tetris is a game in which the goal is to place falling pieces onto a game board. These pieces, also known as tetrominoes, are to be placed in a way such that horizontal lines are filled. When a horizontal line is filled, it is cleared from the board, and the player is rewarded with points. Playing Tetris can be both fun and challenging, and most of us lose after a short while. But how well can a computer perform?

1. Heuristic model

What we want to do is to find a heuristic that is able to analyze the current game board quickly, and from the answer given from the heuristic, be able to place the current piece in the correct place.



We have seven tetrominoes, named (in order from the picture) I, J, L, O, S, T, Z.



We also have a game board, which we want to analyze. Given that we know which type the current piece is, can we find out where to place it on an arbitrary game board?

* 1. The decision function

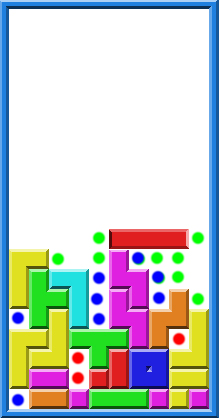
In order to find out where to place the piece, we need to try every possible placement, and every possible rotation, in order to find out which placement is optimal. We note that we can find the optimal placement for the current piece in accordance with our heuristic, but it is very unlikely that the program finds the optimal placement for a sequence of pieces.

However, we find that if we check every possible placement and every possible rotation for the current piece, and place it accordingly, the AI only manages to complete a few lines before losing. Why is that? It is because the algorithm is too greedy. We need to modify the decision function and make it able to consider the next pieces, and make sure that the pieces that come after can be placed in a good way.

Thus, we still check our heuristic model for every possible placement and rotation for the current piece, but we extend the model even further by checking what will happen after. For every possible placement and rotation for the current piece, we also check every possible placement and rotation for the next two pieces. This means that the program tries to place the current piece in every position, and for every position it tries to place all of the seven tetrominoes in every possible position, and lastly it tries to place four of the tetrominoes in every possible position. Then we calculate the average heuristic score for every placement of the current piece, and choose the position with the best score.

Doing this improves the performance of the AI greatly. From clearing <100 lines, we can now clear over 10000 lines with ease.

* 1. Scoring of positions

The heuristic model needs to be able to analyze a given game board, and decide how good it is.  

We can intuitively tell that the left board is ”worse” than the right one, but how do we make the program see that? We do it by defining “holes” in the game board. A hole is defined as follows:

1. An empty cell under the topmost filled cell in a column.
2. An empty cell in a column next to a column where we have found a filled cell.

We see from these rules that some cells are defined as holes by both rules, and some cells only by one of them. The cells which fulfill both rules are worse, and thus we simply count them twice or thrice, depending on their position. In the images above, green cells are 1-hole, blue cells are 2-hole, and red cells are 3-hole. This way of scoring a game board turns out to be quite good, resulting in scores around 250-2500 lines played before losing. In order to make it even better, we need to calculate the score from both how many holes we have, and how the holes are placed. Holes higher up are to be avoided, since they quickly make the game board higher. Thus we use the following function to determine the score of a game:

public int calcHolesConverted(int[] grid) {

int underMask = 0;

int lneighborMask = 0;

int rneighborMask = 0;

int foundHoles = 0;

int minY = 0;

while ( minY < ysize && grid[minY] == EMPTY\_ROW ) {

minY++;

}

for (int y = minY; y < ysize; y++) {

int line = grid[y];

int filled = ~line & EMPTY\_ROW;

underMask |= filled;

lneighborMask |= (filled << 1);

rneighborMask |= (filled >> 1);

//(ysize-y)

foundHoles += (ysize-y)\*setOnes[underMask & line];

foundHoles += (ysize-y)\*setOnes[lneighborMask & line];

foundHoles += (ysize-y)\*setOnes[rneighborMask & line];

}

return foundHoles;

}

This function calculates the number of holes for a given Tetris grid. We can see that we have added a factor of (ysize-y) to every hole calculation. (ysize-y) is equal to 1 at the very bottom of the grid, and is equal to 20 at the very top of the grid. Thus we can clearly see that holes higher up contribute more to the total score of the grid.

Now the program loops through every position and every rotation for the current piece and the two extrapolated pieces, and checks the score using this function for every possible combination. Using this function we get over 10000 lines cleared before the algorithm fails to play the game, and thus loses.

1. Conclusions

The Tetris AI discussed here is not optimal, and there exists other AI that can complete 10 times as many lines before losing on average. However, many of the more advanced AI use genetic programming, and have a learning curve of several days to several months. Using a scoring function which is simple and fast enables the AI to extrapolate future pieces, and make a better guess.