**Definition of game**

# "Tetriminos" are game pieces shaped like tetrominoes, geometric shapes composed of four square blocks each. A random sequence of Tetriminos falls down the playing field (a rectangular vertical shaft, called the "well" or "matrix"). The objective of the game is to manipulate these Tetriminos, by moving each one sideways (if the player feels the need) and rotating it by 90 degree units, with the aim of creating a horizontal line of ten blocks without gaps. When such a line is created, it disappears, and any block above the deleted line will fall. When a certain number of lines are cleared, the game enters a new level. As the game progresses, each level causes the Tetriminos to fall faster, and the game ends when the stack of Tetriminos reaches the top of the playing field and no new Tetriminos are able to enter. Some games also end after a finite number of levels or lines Below shows a table of all possible pieces under these rules.

# [Possible pieces and rotations](https://codemyroad.files.wordpress.com/2013/04/srs-pieces.png)

# Possible pieces and rotations

# The best possible move

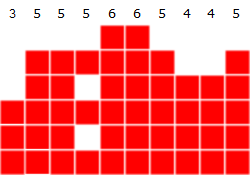
Our goal is to clear as many lines as possible, and therefore, to make as many moves as possible.

To meet this goal, our AI will decide the best move for a given Tetris piece by trying out all the possible moves (rotation and position). It computes a score for each possible move and selects the one with the best score as its next move.

The score for each move is computed by assessing the grid the move would result in. This assessment is based on four heuristics: **aggregate height**, **complete lines**,**holes**, and **bumpiness**, each of which the AI will try to either minimize or maximize.

## Aggregate Height

This heuristic tells us how “high” a grid is. To compute the aggregate height, we take the sum of the height of each column (the distance from the highest tile in each column to the bottom of the grid).

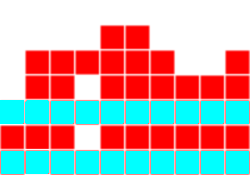
[](https://codemyroad.files.wordpress.com/2013/04/121.png)

Aggregate Height = 48

We’ll want to **minimize** this value, because a lower aggregate height means that we can drop more pieces into the grid before hitting the top of the grid.

## Complete lines

This is probably the most intuitive heuristic among the four. It is simply the the number of complete lines in a grid.

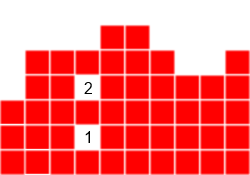
[](https://codemyroad.files.wordpress.com/2013/04/2.png)

Complete lines = 2

We’ll want to **maximize** the number of complete lines, because clearing lines is the goal of the AI, and clearing lines will give us more space for more pieces.

## Holes

A hole is defined as an empty space such that there is at least one tile in the same column above it.

[](https://codemyroad.files.wordpress.com/2013/04/3.png)

Number of holes = 2

A hole is harder to clear, because we’ll have to clear all the lines above it before we can reach the hole and fill it up. So we’ll have to **minimize** these holes.

## Bumpiness

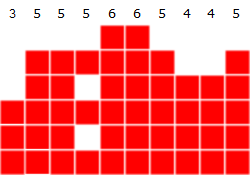
 Consider a case where we get a deep “well” in our grid that makes it undesirable:

[](https://codemyroad.files.wordpress.com/2013/04/4.png)

An unideal grid

The presence of these wells indicate that lines that can be cleared easily are not cleared. If a well were to be covered, all the rows which the well spans will be hard to clear. To generalize the idea of a “well”, we define a heuristic which I shall name “bumpiness”.

The bumpiness of a grid tells us the variation of its column heights. It is computed by summing up the absolute differences between all two side-by-side columns.

[](https://codemyroad.files.wordpress.com/2013/04/121.png)

Bumpiness = 6

In the above example,

bumpiness = 6 = |3 - 5| + |5 - 5| + ... + |4 - 4| + |4 - 5|

To ensure that the top of the grid is as monotone as possible, the AI will try to **minimize** this value.

# Putting the heuristic together

We now compute the score of a grid by taking a linear combination of our four heuristics. It is given by:

A \times (Aggregate Height) + B \times (Complete Lines) + C \times (Holes) + D \times (Bumpiness)

where A,B,C,D are constant, predefined parameters.

We want to minimize **aggregate height**, **holes** and **bumpiness**, so we can expect A, C, D to be negative. Similarly, we want to maximize the number of **complete lines**, so we can expect B to be positive.

I did some genetic tuning to produce the following optimal set of parameters:

A = -0.66569  
B = 0.99275  
C = -0.46544  
D = -0.24077

By using this set of parameters and the score formula, the AI can pick the best possible move by exhausting all possible moves and select the one with the highest score.

# Results

The AI was implemented in Java and tested