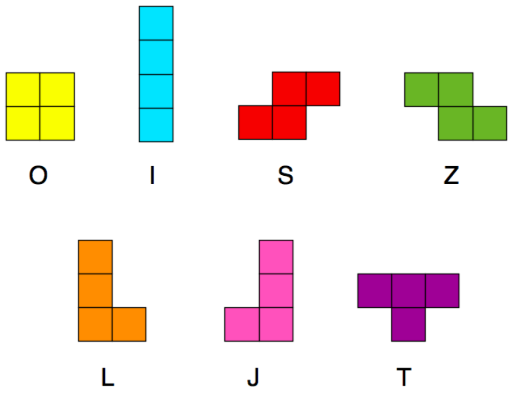
Derek Zhang (dhz2104)
Trey Gilliland (jlg2266)
Parallel Functional Programming

Tetris Position Solver

Introduction + Search Space:

Tetris is a classic tile-based puzzle game in which the goal is to insert tetrominos into an existing matrix of blocks without making the highest row go over the top of the screen. A standard Tetris board is 10 blocks wide and 20 blocks tall. There are 7 possible tetris pieces, as shown below:



The I, S, and Z blocks can be flipped horizontally and vertically. The L and J blocks can be flipped so that the long side is either on top, to the side, or on the bottom. The T block can be flipped four ways: the protruding square can either be on the top, bottom, left, or right. Only the O block cannot be flipped at all. When the squares that consist of a tetris tetromino fill an entire row, the row is cleared, giving the user points and causing the number of rows to decrease. In more modern Tetris games like Tetris Battle and Tetris 99, you are aware of the current piece and the next 5-6 pieces, so you can plan out moves ahead accordingly.

Algorithm:

The goal of our algorithm would be given a Tetris board configuration and the next N randomly generated pieces, determine the most optimal placement of these N pieces that will lead to the highest heuristic score and allow the player to continue playing. Some example heuristics for determining the optimality of a solution:

- Holes - Does this placement leave holes in the board? Holes are defined as enclosed gaps between pieces that would require the top line to be cleared to fill in. The number of holes in a placement should be minimized.
- Lines cleared - Does this placement clear any lines in the board? This should be maximized as clearing lines is key to keeping the max height away from the top.
- Aggregate height - What is the sum of all the max heights of the columns? This should be minimized as we want to keep the largest gap from the highest piece and the top.

As our search space depth is limited to the number of pieces N we know in advance, we could do an iterative deepening depth first search to search the space for the most optimal placement based on the heuristics and weights defined in this article by Yiyuan Lee[2]. This type of search can be parallelized using sparks in Haskell.

Output:

Given a current board state and a list of the next N pieces, our algorithm will calculate the optimal placement of the next piece. While the number of pieces N is given by the user, the starting board and the actual tetrominos will be randomly generated. We will print this board with the optimal placement represented as a 20x10 matrix to the command line before any placements are made, and after each tetromino is placed. The placement of each tetromino will be shown in the board after each tetromino is placed. If time allows, we will allow the user to input a board state and list of tetriminos.

References:

[1] https://markmliu.medium.com/the-tetris-proof-60a7a69a8e04
[2] https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/