Ethan DiPilato
Brett Grossman
Abriana Nash

Tetris Played by AI

The work started out with formulating a plan a finding source code for tetris that could be used already so that time was not spent also coding tetris itself. The source code used can be found at this repository (https://gist.github.com/silvasur/565419). The plan formulated was to create three agents which would play the game. The performance of these three agents could then be compared to each other. The three agents designed were a random agent, a “greedy” agent and an “optimal” agent. In addition to these agents, a secondary search was necessary, as the agents would only produce a final position for the piece in play to be moved to. This secondary search created is a breadth first search adapted from the classwork done with Pacman.

The exact form of tetris used is as followed. The agent is presented with a grid of size 10 wide and 22 tall. The goal is to not lose which means ideally playing forever. Lose conditions are met when the tower of pieces on the board surpasses the height of the board. Actions the agents are presented with are move the piece left, right, down a line, or rotate. In order to continue playing, the agent must place pieces in a manner that allows lines to be cleared in order to keep the tower of blocks as low as possible. As lines are cleared, the game speeds up, leaving less time for the agents to make decisions.

The agents required input of something to perform a search on. This was done through the function of “finalPositions”. This function’s job is to look at the game board and find all potential legal positions for the given game piece to sit as low as possible on the board. The amount of looking the function does at the board depends on the shape of the piece it is given to look at. The result of this function being called is a list from which the agents then choose a final position for the piece to be played to.

The first and simplest agent we made was the random agent. This agent generates a list of all legal positions the given pieces can be placed and chooses a random one. We made this agent to use as a baseline for the actual AI agents. Our other agents would be expected to perform at a bare minimum at least as well as the random agent.This would be useful for tuning the other agents, seeing what kind of scores would be “good” or not.The agent would be expected to perform poorly, and lose the game in a small number of moves.

The second agent we made was a “greedy” agent. This agent takes the current state of the game and generates a list of legal moves with the current piece. It then evaluates the best move to be made based on immediate results. Simply it will take whatever move will keep the max height of the board’s pieces the lowest, or will raise the score of the game. This agent’s behavior is expected to be similar to that or a novice or intermediate player, making logical decisions on what to do with the given pieces, but without any foresight. The greedy agent should perform significantly better than the random agent, while keeping runtime low. Without any foresight however, it should easily be beaten by an experienced Tetris player, or a more complex agent.

Our final agent was our attempt at creating an “optimal” agent, which we did using an expectimax search. This agent started by again generating a list of legal positions a given piece could be placed. It would then take the maximum of a set of recursions of the function with each legal move. This step would repeat with the second layer, using the “previewed” piece that most versions of Tetris features. At lower layers, the agent would generate a list of each shape of the pieces, and then get the average of the maximum of a set of recursions for each piece. At the final layer, an evaluation function would be run on each legal move, instead of recursing again. This evaluation function would be the same as the one used in the greedy agent. Our expectimax agent was expected to perform better than most human players, and last significantly longer than the other agents. The main concern of this agent would be runtime, which would exponentially increase as further layers were generated. For this reason we wanted to switch to the greedy agent once the game was sped up to the point where the expectimax agent was too slow.

After an agent is run and chooses a final position for the piece to be played to, a solution of how to get the piece there is still required. This is where the graph search comes in. The graph search was adapted from the pacman work and implemented as a breadth first search. It takes the position given by an agent and finds the solution of moves to get there, those moves being left, right, rotate, and move down a line.

The final portion of the work was to find a way to translate the list of actions produced by the solution search into commands to give to the game and actually move the piece without user input. At this point was where we ran into major difficulties. We could not determine how to translate the actions in a fully and consistently working manner. As such the project is not in a state where it can be executed and have the game be played “by itself”.