Logan Romero, Jake Kaplan, Matt Schueler

1 February 2018

**meme\_machine Report**

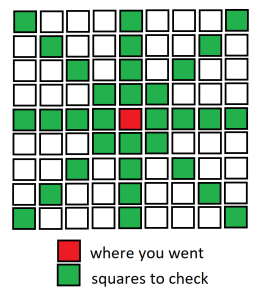
**Compiling and Running the Meme Machine**

First, compile our machine by running the makefile, via the “make” command, in the root of the folder for our submission (found within the zip file). This script will fully compile our program, entitled “ai.c”, and create the necessary links and binaries required to run the program from a Linux shell. Second, to run our program simply use the “./arena/ai” command followed by a space and the team name. This will create an image of our bot capable of interfacing with the referee and taking on any opponent. Last, to remove the files of our bot from the location it is installed simple input the command “make clean”, this command will scrub the directory of the files created by the compilation script. From there simply delete the remaining files from the directory.

**Utility Function**

Our utility function of choice is incredibly simple and effective. The function ranks actions based on their ability to build players toward a sequence of five tiles; the closer it gets the player to five, the more it’s worth. This function is easily implemented as a form of pattern recognition, where the bot will scan the board looking for combinations of tiles that could provide the player with greater utility.

**Evaluation Heuristics & Strategies**

Building upon our utility function, our heuristic dictates that the best measure toward winning is through the construction of intersecting potential winning patterns. That is, the best way to win is to construct two or more winning sequences such that the opponent cannot block you from using one or the other. Based on that assertion, we updated our utility function to parse through the board and search for multi-dimensional patterns about an index. Once a pattern is found a separate board of move weights would be updated and the value of the moves within the pattern would be added to their respective tiles. This allows the meme\_machine to consider both its own and the opponents ability to string together win states. In theory, this should allow our bot to perfectly avoid the opponent succeeding in attaining a win state, as its tiles would have very large weights if they are a part of multiple patterns.

The way our heuristic functions is that we start by parsing through the board state given to us by the referee and in turn our minimax function. Whenever we reach an empty tile we call the first part of our heuristic function that collects the strings built by the adjacency from this tile in a radius of five tiles (see right). From this process our bot now possesses four strings, one for each of the lines formed by the graph. These strings are then piped into our pattern recognition function that will scan the strings for patterns and assign a weight to the tile at the intersection of all four strings. The weight given is determined by the number of possible patterns that coincide on this one tile, and as a result favors building toward intersectional patterns rather than standalone structures. Below you will find the results of our testing trials on the bot and our heuristic.

**Tests and Results**

To test the effectiveness of our machine we had it play 35 games against itself, the results of which can be seen below:

|  |  |  |
| --- | --- | --- |
| **Trials** | **Win Rate** | **Average # of Turns** |
| 35 | 53% | 31 |

**Discussion**

Based upon our research into the game and the results of our tests, we believe that we have picked exceptionally strong heuristics for our evaluation function. From a strategic standpoint, evaluating our moves on multiple fronts run like the strategy of most Go players. From a technical standpoint and assuming a perfect minimax opponent, if we were only to value moves in a vacuum and not how they function with the other tiles on the board the game would likely never progress. Instead the two contestant bots would likely just keep shutting each other down and a forcing a draw. However, giving our bot the ability to think ahead (via minimax) and the ability to determine which moves work best via the heuristic as it moves ahead, it is capable of forcing moves that avoid these situations. When corroborating this assertion with our data it seems that we can safely assume that our heuristic is a good guide for our bot to choose the right moves. In terms of expanding the bot in the future, it would be best to run many more simulations to properly work out the weights for each type of pattern. Or, in the most optimal case allow to the bot to learn and change the weights on its own to get the best possible fit of weights to success.