AI Project 2 comments

Rowan Clare – 834211

Jarren Toh - 833627

The methodology used for this project involved splitting the AI into two different areas, one for approaching the placement phase, and one for the movement phase.

For the movement phase, the AI uses various data it compiles to determine priority moves, which it can use to save or kill pieces, then from those, or the rest of its moves if it doesn’t have any priority moves, it assesses the positions those moves would lead to based on 3 heuristics (described below). From there, it shortlists the best 5 moves, and proceeds to perform a basic search algorithm, which performs the same actions on the new board up until it reaches a certain depth, and uses a minimax process to ascertain the best option in each situation, up until the base situation.

The previously mentioned heuristics in use are, Board Control, which compares each players level of control over the centre, valuing centred positions exponentially more. Structure, which compares the formations of each player, valuing connections between pieces higher than spread out pieces. And aggro potential, which looks out the amount of squares on the board that each player is threatening.

Those 3 heuristics were determined largely through process of learning the game as a human player. At a high enough level, the game devolves into specifically trying to take the centre, preferably with structures that are hard for your opponent to interact with, and prevent the opponent from ever contesting your board state. This means that all 3 of these heuristics are exceptionally important, if the AI can function effectively enough, and if the heuristics are well defined enough.

While the method was unable to manage this, due to time constraints, a machine learning approach was initially planned for usage in conjunction with the heuristics. In the function that evaluates a board, the heuristics are each multiplied by a different value. This value serves as a means to actively emphasise and deemphasise heuristics as appropriate, based on their apparent value in the game at large. Eventually, values were chosen largely based off of trial and error, however, machine learning could be applied reasonably easily by collating data on the player class based off of how effectively it plays against itself at different modifier values.

The placement phase applies similar principles to the movement phase, using heuristics to determine the value of each square, and making a move to the square based off of the highest yield on a square. During the placement phase, the AI works off a simple scoring system evaluating the importance of each space on the board and then choosing the space with the greatest value. There are several factors that influence this decision. The most significant are the immediate consequences of the move, that is, any pieces that will be killed upon this move. If this move will kill opposition pieces, it will increase in value, correspondingly, if such a move were to kill the piece being placed, the value would drop dramatically. A less immediate effect is how threatening the space is, this takes into account how many of the opposition and friendly pieces will become vulnerable (within one move). As this is less significant, it has a smaller impact on the score. The final two factors influence the score less than the immediate effects but make up the strategy in placing them. These strategies continue on to influence the movement phase. They are centre control and structure quality, these factors influence the general structure, our program attempts to make and maintain without taking away from the immediate danger of enemy pieces. The placement phase contains a lot less forward thinking compared to the movement phase given the static nature of the pieces.

Despite using the term however, it is likely that the method of machine learning would’ve been something of a pseudo-machine learning method. Given that the code has guaranteed outputs, and would largely be tested against itself, the method used would most likely have involved something to the effect of trying to approximately average out values for each of the mods based off of win rates for games with those values. While it is not guaranteed for the win rates of the distribution to be consistently in line with this value, regardless of how much time is used to determine it, it is likely that, when used as an opponent for distinct players, that it has the highest probability of yielding a high win rate

Notes: the location of the Player class for part B is in Player.py