COMP30024 Project Part B Report

Yang Yue Edson Li 981704

Savas Marcou 995350

• Describe your approach:

The core of our strategy is based on an A* search algorithm that we developed in part A of the project. We used this as it consistently finds the optimal path within an acceptable time limit. This search allows us to effectively find the shortest path from one node to another. We first had to modify this to properly use a hex-manhattan distance - requiring conversion of axial coordinates. After this, adding some weighting allowed us to take into account the nodes we have already placed. The optimal path search thus simply performs a search from every end of one side of the board to every end of the other. All the nodes in the paths with the lowest cost (i.e number of tiles to place) are aggregated into a selection of "best node plays." This collection guarantees shortest path progress towards the end of the board - without taking into consideration capture plays from the opponent. The motivation for this is simple - reach the end of the board before the other player does by placing the *fewest tiles*.

The first iteration of our AI simply selects at random from this collection of "best nodes" until a clear optimal path is formed. However, this was insufficient as the AI did not have any understanding of the opponent's progress, and thus the opponent could simply reach the other end of the board before the player. This led to the development of a basic "blocking" strategy that is essentially minimax with a look-ahead of one, using "optimal path nodes" as the heuristic. This functions by calculating the cost of the opponent's optimal path, 'playing' a node, and then calculating that cost again, testing if it has increased. In our implementation, the block strategy takes as input the player's "optimal path" as its set of nodes to test (i.e. heuristic), thus greatly reducing the amount of searches that need to be performed (however, this strategy can theoretically take any heuristic set of nodes). The evaluation function is the change in lowest cost path of the opponent. The motivation of this is to find the subset of nodes that make it more difficult for the opponent to reach their goal while simultaneously making progress on the player's optimal path.

• **Performance evaluation:**

Our program has gone through three iterations. At each iteration, we allowed the previous version of the algorithm to play against the upgraded version. We found that at each step, the newer version would always outperform the old algorithm. This was our main yardstick for determining performance improvements. We also tested against random and greedy algorithms to ensure consistent success. Our program can consistently perform well against random and greedy algorithms, however we have been unable to compare performance to “smart” algorithms, as our algorithm is the “smartest” we currently have available.

• **Other aspects:**

An important creative aspect was actually playing the game with each other. This allowed us to think in the perspective of our game playing agents, and required some internal thought as to what motivated our actions and line of thinking. This gave us a general direction forward as to how to develop the AI.

We have made and considered multiple algorithmic optimisations in the interest of keeping time complexity down. The first observation was that the algorithm found every node on the board to be part of the “best path” from one side of the board to the other, as any given node would take the same amount of places to reach. Simultaneously, it was also the most time consuming turn in the game. Thus, we chose to instead randomise the starting tile, making significant savings in time complexity. Furthermore, we realised that the exponential growth of our final algorithm at higher board sizes pushed us over the limit. This similarly happened in the first few turns, where the number of nodes and paths to explore is much larger than when a few pieces have been placed. As a result, on larger boards for the first few turns, we allow our algorithm to revert to the older iteration of "optimal path search" which does not consider blocking and thus does much less computation. This is not incredibly consequential as the boards are so large at $n > 12$ that odds of success are difficult to determine from so early in the game.

We eventually decided that it was optimal to always steal the other agent's starting piece, as all the tiles (besides the centre tile) were more or less equal in 'favorability' at the start of the game. This ensured that if we were playing as the blue side, we would still get first player advantage. Another observation we made was that for board sizes $n < 5$, other agents would sometimes win due to accidentally capturing our pieces on the board. As such, to minimise the chances of this from happening, we hard-coded a selection of optimal starting tiles to randomly choose from and place for the first turn for these board sizes.

• **Supporting work:**

We developed a random agent, which chooses to place tokens down randomly. While testing the agent, the game highlighted some moves which were invalid, which allowed us to improve our own agent further as we did not originally consider some scenarios. It also highlighted flaws in our player class code, which was essential in making an agent that did not throw any errors.
We at several points saved a copy of our agent as well, so that after implementing a new optimisation could test our newer agent against our older more stable version, which ensured for constant improvements and a secondary layer to version control.