**COMP30024 Part A report**

Anthony Hill 1305284

Liam Anthian 1269187

When selecting an appropriate search strategy for any given issue, the main motivating factor is always the end goal with the maximum branching factor and max depth of the state space for the problem helping to narrow down strategy. For this project, optimality is the most important goal - a solution needs to be reached with the least possible moves, and efficiency of finding this solution and everything else comes secondary. Regarding the case of tetrominoes on an 11x11 board, the number of possible moves for a given player at any point in time is highly variable, but it is especially high in the worst case scenario of a sparsely filled grid - a single red tile on an otherwise empty board has 164 possible next moves. Even from the starting point in the *test-vis1.csv* example there are 19 possible next moves. With a branching factor this massive, it is especially important that a clever search is discovered.

We ended up implementing a variant of an A\* search, queuing up states in an inbuilt python priority queue (Queue.PriorityQueue) with priority derived from a combination of current count of moves taken (g) and a heuristic (h) to indicate how much closer to the goal a current state now is. For cases where priority is equal between two states, the fresher (and in turn typically deeper) state is placed first - this allows a targeted depth-first kind of exploration towards end goal without sacrificing the optimality of solution seeking at all. It additionally allows us to reach a goal state sooner and effectively prune how many unuseful states we will need to explore.

**CALCULATION OF TIME/SPACE COMPLEXITY HERE**

Our heuristics developed were all purposefully admissible, that is, less than or equal to the minimum number of moves still needed to reach a goal state. Resultantly optimality could never be compromised - at worst, if any chosen heuristic was uninformed we would still reach an optimal end goal (greediness avoided!). Our end heuristic was a measure of the closest red tile to a most filled axis of the target tile. Walked through:

* To clear the target, either the y or x axis must be filled with tiles
* To approach (and begin filling) either axis, a pre-existing red tile must be built off of
* It follows that an optimal solution has to have more than or equal to the cheapest amount of tetrominoes placed to reach and fill an axis
* Thus the heuristic finds the best minimum sum of a red tile to axis approaching and filling for each given board state

Calculation:

| h = 15 (max possible value for h)  for red cells  a = distance to target x-axis + free cells in x-axis  b = distance to target y-axis + free cells in y-axs  If min(a,b) < h:  h = min(a,b)  return h |
| --- |

If all Blue tokens had to be removed from the board to win, the search strategy employed would have to be dramatically altered, and chosen heuristic entirely redesigned. **More text here**.

Notes:

* Writing is still pretty messy in general but just wanted to get a skeleton / draft of concept done