The Duy Nguyen Student no. 1100548
Ramon Javier L. Felipe VI Student no. 1233281

COMP30024 – Project A Report

P1:

We used a hybrid informed search strategy, where A* is utilized for the majority of cases, preferably for sparse to medium-density boards, and a greedy search strategy for special cases of dense boards where every enemy piece has a power of 1. This is because A* generally does not work well with very dense boards (high density also implies a large number of power-1 pieces), as this is more likely to produce worst cases where many states after the same number of moves have equal heuristic. However, our greedy search ultimately trades efficiency for optimality. We used various different data structures for A*, namely dictionary for fast accessing the states, and priority queue, particularly min-heap, for efficiently obtaining the state, or node, with minimal evaluation cost (f). The max-heap is also used in greedy search to obtain the player piece with highest power.

TIME COMPLEXITY:

For A* search algorithm:

- The worst-case scenario would be $\boldsymbol{O(b^d)}$, where:
  - $b = 6p$, where b is the maximum branching factor, and p is the maximum number of player pieces (red pieces) of a generated configuration s_i after i number of moves made from the initial state.
  - d is the minimum number of moves to reach a goal state.
  - For s_i, the time complexity would correlate to generating the node, including its evaluation f-cost if required. This means it has the same time complexity as the heuristic function:

$$h(x) = \begin{cases} h2(x), & if\ n \geq \text{density\_threshold} \\ h1(x), & otherwise \end{cases}$$

where x is a state / node
n is the number of pieces on the board

    - The complexity of $h1$ would be $O(n \log n + ne)$ where n is the maximum number of pieces of the given state, and e is the maximum number of enemies. This is because we first sort every piece on the board by its power, from largest to smallest. Then, we iterate over every piece and check it against each enemy that has not been marked captured (see P2) in the heuristic.
    - The complexity of the modified heuristic $h2$ for dense boards is $O(n \log n + ge)$, where g is the number of pieces with a power greater than 1 (except for enemy pieces with power of 6).
- The average time complexity of the search strategy would be $\boldsymbol{O(b'^d)}$, where b' is the improved branching factor as a result of heuristic h and d is the minimum number of moves to finish a game.
- The best case requires a unique goal state. Additionally, it implies that for each level, we only need to expand a single node; meaning the heuristic would have to be very close, or equal to true costs. However, since the latter condition is not always applicable, or even at all, to our heuristic, we can only conclude that the best case requires that the initial state produces a unique optimal goal state.

For Greedy search algorithm:

The time complexity is $\boldsymbol{O(nd * p \log p)}$ where n is the number of pieces on a given board state, p is the number of player pieces, and d is the derived number of moves to reach the goal state. This is because the strategy checks until it reaches goal state, and each goal state check has $\Theta(n)$ time complexity. Within each iteration, it must continually pop from a heapified player list, and in worst case until the heap is empty. Therefore, in an iteration, the time complexity would be $O(p \log p)$.

SPACE COMPLEXITY:

For A* search algorithm:

In A*, we use 2 dictionaries to keep track of the different costs, and another dictionary and a min-heap (open set) to keep track of the leaf nodes and said nodes and its cost, respectively. The space complexity of former

two is $O(b^d)$, because they keep track every node's costs. Whereas the complexity of the latter two is $O(l)$, where l is the number of unexpanded states. Notably, however, the open set needs to store actual states, which requires more space as other dictionaries only keep track of hashed values. The space complexity for a state includes the board, which is $\Theta(n)$, where n is the number of pieces on the board; and list of moves from initial state that leads to the current one, which is $O(d)$ where d is the number of moves in optimal path. So, the complexity of the open set is $O\big(l * (n + d)\big) = \boldsymbol{O(b^d(n + d))}$ and is also the overall space complexity.

The heuristic function space complexity is only temporary, meaning after a node has been generated, the space complexity of the heuristic will become 0. During execution, the heuristic has space complexity $O(e)$, where e is the maximum number of enemies on a given board.

For Greedy search algorithm:

Since at any given time, we only keep track of a single max-heap of the player pieces, the space complexity would be $\boldsymbol{O(p)}$ where p is the maximum number of players on a given board.

P2:

Generally, our heuristic approach makes two important assumptions. Firstly, all pieces have the potential to be a power of 1 greater than its current power, and this is most applicable to enemy pieces since when spread onto, their power will be incremented by 1. However, we do not apply this to two cases. The first case is when a player (red) piece has power of 6, since adding 1 to it would eliminate an ally. And the second is when there is only a single ally on the board, meaning we cannot possibly increase its potential power any further. Secondly, all enemy pieces are candidates to spread to other enemies since ultimately, once captured, they will become a player piece.

With that, we increment all pieces per the first assumption, and sort all pieces by their incremented power value (we call this **potential pieces**). Then, from highest to lowest power, we check for each potential piece, which direction would have it captured the most enemies. Those enemies will now be stored as captured. For subsequent potential pieces, disregarding whether said piece is captured or not, we also check which direction would capture the most enemies that have yet to be captured. For our heuristic, as long as said potential piece manages to capture an enemy, the estimated number of moves to reach goal state (**EM**) will be incremented by 1. Specifically, if a piece cannot capture any enemy, either because no enemy is in any spread direction of said piece, is out of range, or all enemies have been previously captured, then EM will not be incremented. It needs to at least be able to capture an enemy for EM to be incremented by 1. If there are remaining enemies uncaptured, then EM will also be incremented by 1.

That said, our heuristic is dynamic. Iterating through every potential piece, which also means iterating through the entire state of the board, is only applied when doing so is worthwhile ($h1$). When the board is dense, there exists a large number of power-1 pieces. So, for dense boards, we use an almost identical heuristic ($h2$), but we iterate only until we reach potential pieces with power of 2 (or lower), because these potential pieces have power of 1 originally, and/or enemies with original power of 6. Then we let EM be incremented by the least possible number of moves to capture all power-1 enemies left, which is equal to the number of enemy pieces left, minus 1 if said amount is an odd number.

P3:

If SPAWN action were possible, there would on average be a larger number of configurations on each depth level. This is because the number of child nodes now are not only dependent on the number of player pieces, but also on the number of empty cells. Another difference would be that SPAWN can sometimes shorten the distance between a player and an enemy piece, allowing higher flexibility and possibly leading to, on average, lower number of moves for optimal path. While this may reduce the depth required to find the goal, we believe it is still on average a lot more expensive due to the increased branching factor.

To accommodate this change, we would think of an in-built greedy algorithm that would ignore generating a child node beforehand. This is because SPAWN action results in significantly higher branching factor. Thus, it would perhaps make sense if we could think of states that would somehow be assuredly inherently worse than others to not be considered at all.