Project Part A: Report

How have we formulated the game as a search problem?

State: A state corresponds to an arrangement of tokens on the board. The state is represented as a dictionary containing a list of stacks on the board in the format ["number_of_tokens", "x_position", "y_position"] for each player. The keys of the dictionary are "white" and "black", e.g. {"white": [[1, 0, 1], [1, 2, 0]], "black": [[2, 3, 1], [4, 7, 8]]}.

Actions: The possible actions a player could take are: move (left, right, up, down) or boom. The maximum number of possible moves for a stack of height n is $4n^2 + 1$, corresponding to 4 directions, n steps outward, where up to n tokens can be moved, plus a boom action. An action is represented as a list of data type: ["move"/"boom", stack_from, stack_to].

Goal Test: Our goal test is an implicit goal test - we check that the number of black tokens remaining is equal to zero.

Path Cost: The path cost is the number of actions executed to reach the current node.

Solution: A list of actions required to pass the goal test within the time limit for a given initial state.

What search algorithm does our program use to solve this problem? Why did we choose it?

We chose to use the A* algorithm for this project. The A* algorithm expands nodes with the lowest evaluation function $f(n)$, first, calculated as the sum of the path cost $g(n)$ and heuristic $h(n)$.

Completeness

We formulated the search problem such that the path cost is always 1 (non-negative edge weights). Additionally, we implement state-checking and do not generate children for nodes whose states have been explored to prevent cycles and avoid redundancy. The branching factor of each node is finite due to the finite number of possible actions. There are also only a finite number of board configurations. Hence, there must be a finite number of nodes that can be expanded, so A* is complete.

Heuristic Function

The heuristic implemented was the sum of the Manhattan distances between each black token and its closest white token, divided by the size of the biggest stack of white tokens. We selected Manhattan distance as the tokens are only allowed to move in the 4 cardinal directions. We divided it by the largest stack to estimate the minimum number of moves required by that stack to move that distance. The heuristic is lower when white tokens are closer to black tokens (encouraging white tokens to move towards black tokens), and when there are fewer black tokens on the board (encouraging removal of black tokens).

Optimality

A* is optimal given an admissible heuristic. Our heuristic is not admissible. It sometimes overestimates the cost when there are a high number of black tokens. Hence, our algorithm is not optimal. However, our algorithm still takes into account the path cost and outperforms a purely Greedy Best-First search. It also prevents the algorithm from strictly following a slower breadth first search behaviour by providing near-accurate estimates of the cost to a goal state. In this way, the heuristic implemented finds a balance between optimality and time efficiency.

Efficiency and Further Optimisations

We also implemented a 'loss condition' check to prevent states with no possible solution from being expanded further. This was implemented by calculating the minimum number of booms required to achieve the goal state and comparing this with the actual number of white tokens still remaining. We observed this improved search efficiency for board configurations with a large number of white tokens and those requiring complex solutions, such as trapped white tokens.

What features of the problem and your program's input impact your program's time and space requirements?

Consider a general search problem with maximal branching factor b, maximum depth m, heuristic h, true cost to goal h^* and least-cost solution depth d.

A* has a time complexity exponential in the relative error of the heuristic and the solution depth: $O(b^{\frac{|h-h^*|}{h^*}d})$.

A* stores all generated nodes in memory hence it has space complexity $O(b^d)$ in the case that the heuristic given is admissible, since only nodes with $f(n) \leq h^*$ are expanded. Since our heuristic is not admissible, we may be expanding nodes with $f(n) > h^*$, hence could expand more nodes deeper than d, depending on the accuracy of the heuristic. In the extreme case when $h(n)$ outweighs $g(n)$, A* turns into Greedy Best-First Search, which has space complexity $O(b^m)$.

For our algorithm, we observed that solutions were near optimal and found within the 30 s time limit, suggesting our heuristic is fairly accurate.

The problem

In this problem, each stack of n tokens can take $4n^2 + 1$ possible actions. The maximum branching factor is $b = 37$ when $n = 3$, corresponding to a stack of 3 tokens with all possible actions allowed. The least-cost solution depth d varies with the initial configuration. From experimentation, d could vary from 1 to around 20. The maximum search tree depth m is always finite because there are a finite number of board configurations, but grows larger with increasing number of tokens in the initial state.

Implementation

To minimise space requirements, we implemented state-checking to prevent nodes with duplicate states from being generated and prevent cycles, as well as loss-checking to prune nodes which will guarantee a loss.

To reduce time complexity for state-checking, a Frozenset was used to store explored states instead of a list of dictionaries. This significantly reduced the complexity of lookup from $O(n)$ to $O(1)$. The priority queue was implemented as a minheap, allowing $O(\log n)$ retrieval of the best node. In contrast, implementing a sorted list would result in $O(n \log n)$ complexity.