Comp30024 - Artificial Intelligence Project - part A

Chuan Yang, Odin Wong

Formulating the game

For the single player variant of Chexers, turning the game into a search problem was straightforward.

- A state of the game includes the following information: a list of coordinate pairs of player pieces, a list of coordinate pairs of any blocks, and the player colour, from which we could derive the destination coordinates.
- A action represents a single valid move or valid jump, and each action updates the state of the game.
- Goal test for the game is completed when the pieces list is empty, which means all pieces have reached the destination.
- Path cost of a state is represented by the sum of all the action steps for each piece to move to its destination.

The basic rule is, we generate all children state according to all valid actions from a parent state, and update the state of children when they are generated, when one piece reaches the destination, it is removed from the piece list. As we were trying to minimize the number of actions taken, and each state is one move away from its parent, the path cost between nodes is always 1.

Algorithm:

We chose simple BFS initially, it is optimal and complete but it turns out to be too slow. Because too many useless children are generated and we have to generate their children again as they are stored in a queue.
In the end we decided to use A*, which only required some small modifications: changing the queue to a priority queue and adding a heuristic function. From the initial state we generate the tree by considering all the possible actions for every piece. For every direction we check both MOVE and JUMP actions, if the move is possible (consider blocks, edges etc.), generate the child state and add it to a priority queue. The priority queue is ordered on

$$totalCost(state) = currentCost(state) + heuristicCost(state)$$

Such that the least total cost node is expanded first. A* shows great improvement than BFS, it costs much less time and space as it does not expand those non-optimal children states. It is always optimal because the heuristic cost is always smaller than real cost.

For the heuristic algorithm, we take half of the estimated direct distance as the estimated minimum distance. This is to take into account all the possible jumps on the path to destination, as they reduce the distance by one each. This remains admissible as the distance calculation provides the actual direct distance, and any blocks/pieces are assumed to always be in the path and be able to be jumped over. Thus it will never be greater than the actual distance, however it it has error up to$0.5 \times (real\ path\ cost)$in the worst case.

Besides, we also considered using the minimum between half that distance and the number of block pieces for the consideration that a piece can not jump if there is no block on its path. However, there exist special cases like two pieces can jump over each other and reduce the total cost by half. So, we chose the simpler one to avoid making the heuristic algorithm too complicated and mistaken estimation (larger than real cost), to make sure we kept admissibility.

Feature and Complexity:

Each piece on board has 6 directions to move in, but we reserve a parent state of the current state, and current state can not generate a child which is the same as its parent (avoid the smallest circle), so each state can generate 5 children states, which means the branch factor is $5 \times (number\ of\ pieces)$. As we are using A*, we expand the predicted optimal state node and end it when reaching the game-over state, so the depth of tree is the actual minimum cost of the game (assuming the goal is reachable).

In the best input case, If all pieces can take nothing but jump actions on the path to their destination, the estimated cost is equal to the real cost, then we expand the optimal node, thus less space and time are needed. This case requires that blocks or pieces should be in the paths of all pieces such that pieces are always given chances to jump in every move until the last exit move.

Inversely, if no one piece can take a jump action but has to move step by step, or even worse, the blocks block their straight ways to destination and pieces have to move in a curve, which means the estimated cost is far less than real cost, we will expand many non-optimal nodes, and that costs more time to compute and space to store useless nodes. I practice we will likely get something in between these two cases.