COMP30024 Artificial Intelligence

Part B Report

Zheping Liu, 683781
Bohan Yang, 814642

1 Program Structure

The program we designed have three major modules, and one helper module. The three major modules are Player, GameState and GameBoard, the helper module is PriorityQueue which is majorly used when doing A* search.

1.1 Player

Player is the class that represents the players in a game. Each Player instance contains their colour and current game state, which is an instance of the GameState module.

1.2 GameState

GameState instance contains all necessary information about a game state at a given turn. These information includes current positions of all pieces for all teams, exit positions for all teams, the number of exits for all players, the last action applied, the predecessor state of the current state and a GameBoard instance.

1.3 GameBoard

This is a relatively small module. It contains all valid positions in a game, and some basic functions like computing the distance between two positions, computing jump destination and jump median, checking if two positions are next to each other, etc.

2 Approach

The basic approach our team has implemented was book learning plus modified minimax algorithm. Also, we have an own-designed evaluation function and state reward function.

2.1 Book Learning

Before choosing any action for our pieces, our team will filter all available actions, the results are two lists, *urgent actions* and *desirable actions*. Urgent actions are those actions with highest priority. Generally, they are actions that could stop us losing any piece. These actions includes moving pieces in risky away and moving pieces to stop enemy acquiring our pieces (i.e. blocking jump destination of enemies). All desirable actions include those actions will not result our pieces in danger. For example, moving to positions that can be acquired by enemies or moving away from the protecting positions. Moreover, all urgent actions needs also to be in the desirable lists, if not,

they may just move from one risky position to another risky position. When there is any urgent actions, they will always be considered first, other desirable actions are only considered when there is no urgent actions.

2.2 Modified Minimax Algorithm

In this modified minimax algorithm, our agent will first get all successors by applying the filtering from previous step. For each of these successors, it will simulate the players in sequence to choose their *optimal* action and keep iterating until the limit is reached. During each of these iteration, we compute the rewards of that successor and use them to compute the final evaluation scores for all initial successors. Finally, we will choose the action/successor with the highest evaluation score.

The evaluation function is described below.

2.2.1 Final Evaluation Function

$$\sum_{i=0}^{LIMIT} \gamma^i \times R_i \tag{1}$$

The γ above is the discount factor, R_i is the state reward at state i. We are discounting the future state rewards and sum them together to get the current state reward.

2.2.2 State Reward Function

The state reward function is separated into three components.

First component is the total rewards of number of pieces.

$$R_{pieces} = n \times r_{piece}$$

Where n is the number of my pieces in the given game state, and r_{piece} is the reward for every single piece. To avoid the situation that agent will always value capturing other pieces higher than moving to the goal positions, or situations when there is not enough pieces to exit but the player still values number of exits higher. Our team has designed the reward of a single piece as a natural logarithm function of the ratio between number of my pieces and number of enemy pieces.

$$r_{piece} = -3 \times \ln(\frac{n}{m}) + 10$$

Where n is the number of my pieces, m is the total number of enemy pieces. As the ratio increases, the marginal benefit by capturing one piece decreases. On the other hand, when the ratio decreases, the marginal benefit by capturing one enemy piece increases. This function can help encourage the agent to capture more pieces when we are outnumbered by enemy pieces, and moving towards the exit positions when we outnumber our enemies. If the number of my piece (n) or the number of enemy piece (m) is zero, it will return 0.

The second component is the total reward of the total distances to the goal state.

$$R_{distances} = d \times r_{distance}$$

Where d is the total distances between all of our pieces to their closest exit positions in respective. The $r_{distance}$ is a much simple function, it returns -2 when the sum of number of exits and

number of pieces for the given team is greater or equal than 4, else it will return 0. This is to encourage the pieces move closer to the goal positions when we have enough pieces to exit, since longer distance will bring higher penalties.

The final part is the total rewards of the number of exits.

$$R_{exits} = e \times r_{exit}$$

Where e is the number of exits at the given state, and r_exit is the reward for every single exit. When the game state is the goal state ($e >= 4$), each exit will get a reward of 100. When there is enough pieces to exit ($e + n >= 4$), each exit will get a reward of 15. At last, if there is not enough pieces to exit, each exit will have a reward of -100 (i.e. a penalty of 100). This is to encourage the pieces quickly moves towards the goal state and avoid exiting the game board when there is not enough pieces to win.