COMP30024 Assignment 2

Chuanshu Jiang, Zhengtian Lu

May 12, 2021

1 Approach

The task was broken up into two parts: designing a search algorithm to explore possible moves, and designing an evaluation function to pick the best option out of the possible moves. After repeatedly testing two contending search algorithms against each other, it was decided that the minimax searching algorithm with alpha-beta pruning was the better of the two.

1.1 Search Algorithm

Two common searching algorithms were compared to determine the most suitable algorithm for this assignment. These were:

1. Simultaneous minimax (or "maximin")
2. Pseudo-turn-based minimax

1.1.1 Maximin

The maximin algorithm is typically applied to games where all players make their moves simultaneously. This strategy presumes that the opponent will always choose the move that will result in the worst outcomes for the player, then chooses the best "response" to this move. The player "maximin" implements this.

1.1.2 Minimax

The minimax algorithm is used in turn-based games, however it seemed suitable for this purpose as we could use alpha-beta pruning, which would greatly increase the efficiency of the algorithm. To apply the minimax algorithm to a simultaneous game, each turn was broken up into two "half-turns", which were then evaluated using a minimax search tree with alpha-beta pruning.

1.1.3 Comparison

Ultimately, the minimax algorithm proved to be the most efficient, as searching was a lost faster with alpha-beta pruning.

1.1.4 Improving Efficiency

It was discovered that as the throwing range increased, the number of potential moves increased dramatically, however many of these moves were garbage throws that would not need to be considered at all. Therefore, a function was created to determine the best possible throw action out of all possibilities, and the searching algorithm would only consider this throw action.

Also, alpha-beta pruning is much more efficient if better moves are prioritised, as it can prune off more undesired branches if good moves are discovered quickly. Because of this, slide and swing actions were prioritised over throw actions.

1.2 Evaluation Formulae

Designing the evaluation formula for this game proved to be a more difficult task, as there were many parameters involved. Ultimately, we decided that the two most important factors to consider were:

1. The value of the tokens on the board
2. The distance between a token and its "target", i.e. an opponent's token that it can destroy.

The following factors were used to create the final formula:

- The value of the player's token should be proportional to the number of opposing tokens it can destroy. This will limit the number of useless throws made, and prioritise throwing when the opponent owns token that cannot be destroyed by the ones already on the board.
- The value of the player's token should be inversely proportional to the number of identical tokens already owned. This will prevent the player from continuously throwing one kind of token and wasting throws
- The value of the player's token should increase as it gets closer to an enemy. This incentivises the searching algorithm to select turns that move the player's tokens towards the closest enemy.
- The value of enemy tokens stays constant. This was originally not the case, however it was discovered that weighing opponent tokens using the same metric as the player's own tokens resulted in many strange behaviours, including: the player refusing to throw tokens as that "increased the value of the enemy tokens" and choosing moves to destroy its own tokens in order to decrease the value of the enemy tokens.

Below is the final evaluation formula used to determine the value of the player's own tokens.

$$t_{value} = \frac{\text{number of defeatable opponents}}{\text{number of identical tokens} \times (\text{distance to closest target} + 1)}$$

The formula used to evaluate the score of a whole board was

$$b_{value} = \sum(t_{value}) - 2 * \text{numer of opponent's tokens}$$

2 Performance Evaluation

2.1 Supporting work

Two players were written to test the final algorithm. The first was a random player, who would pick a random move out of all available options. The second was a greedy player, who would look at all possible moves and pick the best one. This player was named slowandgreedy, as it used exhaustive searching and was therefore very slow.

2.2 Testing strategies

The most important strategy used in testing was introducing the element of random choice. This was because when two players were pitted against each other, the games would always reach the same position. To fix this problem, algorithms were allowed to randomly choose between two options if the values of the positions were the same. This was very useful in the "opening" of the game, and introduced enough entropy to provide useable data when testing the players against each other.