**COMP30024-Project-B Report**

Written by ThePinkCoder

**Gameplay Approach**

To compete in RoPaSci 360 by artificial intelligence, the main applied strategy is “Finding the equilibrium strategy” which is cooperated with payoff matrix, evaluation function, cut-off and pruning. Since the game is playing simultaneously, the opponent’s best next action and the situation of next board are unpredictable and the turn is indivisible, minimax algorithm would not be used in this project. On the other hand, to handle simultaneously-play nature of this game, the equilibrium strategy is preferred because it considers all possible next actions whether our or their actions and it would offer the next move choice which has the highest probability to get the best move on board. In general, the provided solve game function assists the agent to select an average and conserved action to ensure the outcome is acceptable and not too risky. In the following section, the three key parts in this project: evaluation function, single-stage and multi-stage would be discussed. The additional applied cut-off and pruning strategy would be introduced in Performance Evaluation part.

***Evaluation function*:**

Since the solve game function has been well designed, building up a competent evaluation function has become the prior goal in this task. To generate an evaluation value, it would analyse the board situation. It takes two parameters: old board and new board, so the difference between the board before and after the next move can be detected. In this case, by comparing the new and old board, the win or loss of token can be told, and the result would count toward to the evaluation score. To let the agent being aggressive, defeating enemy token is encouraged which means that winning a token would be more valuable than losing a token. In addition, the distance between ally and enemy tokens would also be concerned. The relationship between distance score and distance is inverse proportion. It means that the two closed tokens would have more influence than two apart tokens in the calculation of evaluation score.

\**effective opponent token* is the token which can defeat or be defeated by ally token.

Conclusively, the situation with high evaluation would be the winning ally token is closing to the opponent token and the losing ally token is running away from the opponent token. Furthermore, if the ally token defeats opponent token, the evaluation score would also increase.

***single-stage*:**

To implement the single stage, the updating board and getting next actions (including swing, slide and throw) are the primary designs. Fortunately, these two functions have been done in comp30024-project-a, thus only small modification is required for pruning, and it would be explained in detail in Performance Evaluation. Firstly, all ally next actions and opponent next actions would be generated via getting actions function. The pruning strategies are applied to remove some unnecessary or sub optimal actions (more details in Performance Evaluation). Afterward, a double for loop would create a matrix which the row represents enemy next actions, and the column contains ally next actions. By going through the matrix, the current board would be deep copied, and then the copied board would be updated by one ally next action and one enemy next action. Hence, an original board and a possible new board could be used by the evaluation function to see the performance of the new board. At the end, by utilizing the payoff matrix and solve game function, the greatest ally next action would be the one corresponding to the highest number in the probability vector which is returned by solve game function.

***multi-stage*:**

The multi-stage strategy in this project is Backward induction algorithm. To look an increasing number of turns ahead, the best next action would not be chosen immediately in the single-stage. Alternatively, the new possible board would do the single-stage process again to gain the turn ahead expected value and return it back to the first-round payoff matrix. In general, the evaluation score in payoff matrix would be updated through the repetition of doing single-stage step. Ideally, if the evaluation and solve game function are capable. The performance of multi-stage would be greater than single-stage. Yet, the increasing of executed time is expected. The trade-off between time and optimal solution is worth to be concerned.

**Performance Evaluation**

Due to the constraints of program computation, the maximum computation time limit of 60 seconds and maximum memory usage of 100MB are critical to the program before any cut-off and pruning, especially for the multi-stage. Since cut-off and pruning are necessary to be used to pass the time and space constraints, the solution of next action may not be guaranteed to be optimal. Furthermore, additional guides are introduced to select the prefer action directly, so the behaviour of agent may be more likely to act as expected. The following section would illustrate how the single-stage and multi-stage being cut-off and pruned, and the reason of doing that. Also, it would talk about the pros and cons of the implementation, and the balance between time/space and optimal solution.

***single-stage*:**

To reduce computation time, some next actions would be filtered before entering the payoff matrix. Since the number of actions decreases, the time of calculating evaluation score would also decrease, and this is our main focus for passing the time constraint. To prune the next actions array, finding the most significant move is important. In the implementation, only the token which has the shortest distance between itself and “effective” opponent token would be reserved. It means that other tokens would not be urgent to attack, or doge, and therefore they would not be considered in this turn. For instance, if there is a rock ally token and there is no paper or scissors opponent token, the move of the rock ally token would not be in the next action array. In addition, the throw actions can be pruned as well. When an ally token is going to be thrown, it aims to attack enemy tokens. Therefore, it is expected to be thrown at the front row so it can tackle down enemy in a short time. Also, not all token types (r, p, s) would be concerned in one round, only the needed type would be considered. For example, if there is one rock enemy token and no paper ally token on board, only throwing a paper token would stay in the next actions array.

*Approximate Comparison:*

*Before Pruning:*

*After Pruning (5th throw situation):*

* In most of the time the, Tally would be 0. It would only be greater than zero when it is necessary to throw a particular type of token.
* Mally usually only considers the ally tokens (one token for each type: r, p, s) which have the shortest distance between effective opponent tokens. Sometimes, it would include the ally tokens which are urgent to doge.
* Toppo and Moppo would have same the pruning strategy as Tally and Mally (further detail in Observation & Result).

***multi-stage*:**

In multi-stage, not all next actions would be considered further due to the limit of computation time. Alternatively, only the actions which score is higher than median score would do the multi-stage. The actions with low evaluation score would be eliminated immediately. The agent would believe the actions with high score are more likely to perform well in the following round. Alternatively, the actions with low score would be abandoned immediately, due to their low potential. Furthermore, it is not only filtering ally next actions, but opponent next actions also would be randomly selected for the multi-stage (since opponent’s further actions are unpredictable). The opponent next actions size would be trimmed as well. Therefore, the process can be executed in an ideal time.

*Approximate Comparison:*

*Before cut-off:*

*After cut-off:*

***observation & result*:**

After applying the pruning and cut-off strategy, it saves a bunch of times from avoiding checking the meaningless actions. To prune the enemy actions, the same rule is applied. The agent would assume that the reserved actions are optimal, since the opponent would not be smarter (it would not always be true apparently). Yet, we still believe that it would not take a non-efficient action. The opponent is always treated as the agent which has similar behaviour as our agent. If the opponent is taking irrational or sub optimal move, the agent can still handle it due to its more capable algorithm and evaluation function. However, if the enemy is smarter, the agent may not win the game. Since their next actions may be pruned (by our arrogant pruning and cut-off), our agent would not forecast the outcome of the “unpredictable actions”. In general, pruning and cut-off can save the computation time and space indeed. Yet, it would take risk of not simulating all possible outcomes, and it would be dangerous if the opponent is applying a better strategy. In our final design, although the multi-stage has been implemented and it can work competently, it would still be turned off due to the constraint of computational time. Overall, the trade-off is acceptable when our agent is facing the Random and Greedy opponent. Therefore, the pruning would save the computation time and it would not affect the game result at all.

**Other Aspects**

In this section, throw and move strategy would be discussed. Although the evaluation and solve game function would pick the ideal action, our team still implement some rules to guide the agent to do the wanted actions directly. In this way, the next actions array can be smaller, and the computation time would also be lesser. Since the strategy would affect the algorithm and gameplay significantly, the implementation is an important aspect in this program.

***Throw Strategy:***

***Move Strategy:***

***For the throw action, if we do not do any pruning or filtrate on it, it would be 3 x the column we are able to throw, (3 here is the amount of type we can choose, ‘r’/’s’/’p’). For example, when we have throw 1 token, the available column we could throw would be 11 and the throw action would be up to 33 different choices. An the end, when we are throwing out last throw, we can throw any where in the board, which means we have up to 183 possible throw action. It would definitely slow our speed of creating the pay-off matrix.***

***After deep consideration, our group find that it is not necessary to considerate to throw all three types in some situation. For example, when we got token with type ‘r’ and in the current board, and the opponent have token with type ‘s’ and ‘r’, we only need to consider to throw a new token with type ‘p’, it could help to defeat those token with type ‘r’ of opponent. Throwing other types of token would not help us to win, and throwing a ‘s’ token would even increase our chance to be defeated, they are considered to be the sub-optimal action in this situation.***

***Therefore, we come out a strategy that for the throw action in each turn, we only consider to throw those types that we do not have in the current board, and it is also needed in order to beat the opponent at the end, like the example above, if we do not throw any token with type ‘p’, we would not able to win at the end.***

***As the result, the number of throw action we need to considered in each turn would highly decrease, and even when we have got all three types of tokens in the board, the throw action would be 0. On the one hand, it could speed up the creation of the pay-off matrix. On the other hand, it is a kind of pruning, only consider the urgent and necessary type to be throw, remove those clearly sub-optimal throwing, and make each throw action more useful and helper to win the game.***

***Pitfall: after over 20 games, we find that our agent only kill opponent’s token by moving (slide or swing) and never by throwing. But actually, while some opponent tokens are in our throw-able area, throwing a token which can defeat it directly to one of those column is a more efficient and faster way to kill opponent tokens.***

***Solve: Before generating the action of throwing and moving, we decided to consider if there is any throw actions, that has a higher chance to***