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 (default strategy), 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.

$$token\ score = \ ally\ token\ calculated\ by\ its\ closest\ effective\ opponenet\ token$$

$$distance\ score = \frac{total\ token\ score}{total\ ally\ tokens}$$

$$diff\ ally = number\ of\ ally\ \ of\ old\ board -\ number\ of\ ally\ \ of\ new\ board$$

$$diff\ oppo = number\ of\ opponent\ \ of\ old\ board -\ number\ of\ opponent\ \ of\ new\ board$$

$$evaluation\ score = 1.5 * distance\ score - 12 * diff\ ally + 20 * diff\ oppo$$

**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 pay-off 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.

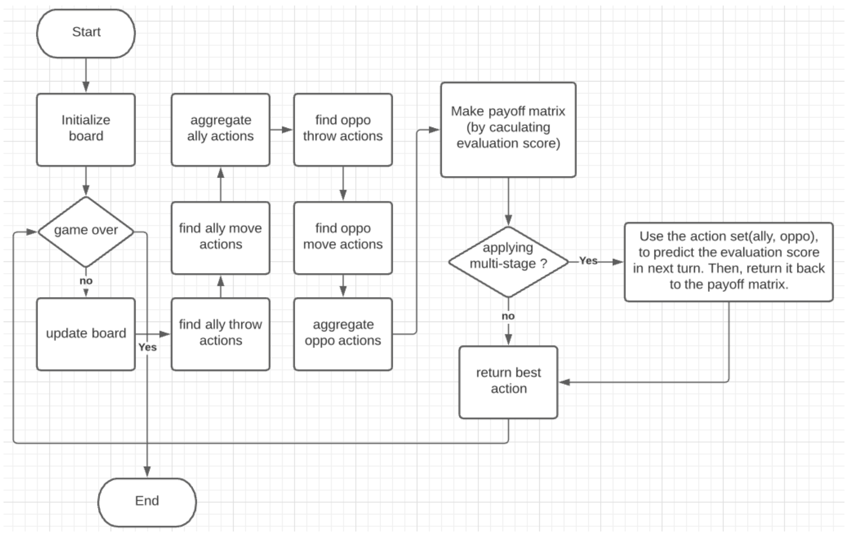

Figure 1) The flowchart for our implemented agent

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 an ally rock token and there is no opponent paper or scissors token, the move of the ally rock 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 enemy rock token and no ally paper token on board, only throwing a paper token would stay in the next actions array.

Approximate Comparison:

$$T_{ally} = total\ ally\ throws$$

$$T_{oppo} = total\ opponent\ throws$$

$$M_{ally} = total\ ally\ moves$$

$$M_{oppo} = total\ opponent\ moves$$

Before Pruning:

$$size\ of\ payoff\ matrix = (T_{ally} + M_{ally}) * \ (T_{oppo} + M_{oppo})$$

After Pruning (5th throw situation):

$$\begin{aligned} size\ of\ payoff\ matrix \\ &= \left(\frac{1}{3}T_{ally} * \left(0\ or\ \frac{9}{5+6+7+8+9}\right) + \frac{3}{5}M_{ally}\right) \\ &* \ \left(\frac{1}{3}T_{oppo} * \left(0\ or\ \frac{9}{5+6+7+8+9}\right) + \frac{3}{5}M_{oppo}\right) \end{aligned}$$

- In most of the time, T_{ally} would be 0. It would only be greater than zero when it is necessary to throw a particular type of token.
- M_{ally} 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.
- T_{oppo} and M_{oppo} would have same the pruning strategy as T_{ally} and M_{ally} (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:

$$A_{ally} = total\ ally\ actions$$

$$A_{oppo} = total\ opponent\ actions$$

$$n = number\ of\ times\ doing\ multi-stage$$

$$m = proportion\ of\ random\ selection\ of\ opponent's\ actions$$

Before cut-off:

$$number\ of\ times\ referring\ to\ evaluation\ function = (A_{ally} * A_{oppo})^n$$

After cut-off:

$$number\ of\ times\ referring\ to\ evaluation\ function = (\frac{A_{ally}}{2} * \frac{A_{oppo}}{m})^n$$

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 overconfident 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:

For the throw actions, if there is no pruning strategy has been applied, the number of actions would be three times the hexes which can be occupied directly in this turn. For example, when one token has been thrown, the available hexes can be thrown would be eleven and the throw actions would contain thirty-three different choices. At the end, when the last token is ready to be thrown, the whole board would be available for the throw action, which equivalents to one hundred and eighty-three possible throw actions. Clearly, it would definitely slow our speed of creating the pay-off matrix.

After much thought and consideration, our group decide that it is not necessary to consider throwing all three type tokens in some particular situation. For example, when there is an ally rock token, and there are one opponent scissors token and one opponent rock token, only paper would be considered to be thrown which can help to defeat the opponent rock token. Since throwing other types of token would not assist ally to win and throwing a scissors token would even increase the chance to be defeated, the actions are considered to be the sub-optimal action in this situation which would not be the candidate of next action afterward. To solve the issue, the strategy of throwing the types of ally token which are not on the current board and needed to win the current situation would be applied.

As the result, the number of throw actions in each turn would be highly decreased and might even drop to zero when three types of ally tokens on the board, the. Also, it could speed up the creation of the pay-off matrix. Since this ideal pruning is considering the urgent and necessary throw actions, removing those obvious sub-optimal throwing, and making each throw action be more influential, it is expected to improve the win rate of the agent.

In addition, based on the observation through several gameplays, the idea of attacking opponent's token by directly throw is inspired. Ideally, this is more efficient to defeat enemy. However, the potential risk is that if there is only one opponent token on board, the chance of beating the token is very small (since the token have six other possible moves). Therefore, the directly throw action may be a waste at the end. To avoid this situation, some constraints and guideline have been introduced to the agent, thus it would tackle down the opponent tokens by a wiser throwing decision.

Move Strategy:

Due to the constraint of computational time, the further demand of reducing move actions including slide and swing is required. Usually, an ally token would have six different slide moves (since the swing condition rarely happens). The total average move actions would be six times the number of ally token on board. As the game progressing, the number of tokens on board would increase. This situation certainly would extend the size of pay-off matrix which would increase the computational time eventually.

To filter the sub optimal and inefficient move actions, the strategy is that only selecting the token which has the shortest distance between itself and "effective" opponent token. It means that other tokens which are not being urgent to attack, or doge would not be considered in this turn. For instance, if there is an ally rock token and there is no opponent paper or scissors token, the move of the ally rock token would not be in the next action array. The use of shortest distance ally token is to ensure that the move action is the most influential one which can gain the best interest for our agent. As a simple example, if there are two ally rock tokens and one opponent scissors token, only the one rock which is closest to the opponent would be concerned. The implementation would eventually provide three move actions which are one rock, one paper and one scissors (if they are on the current board). It allows the size of pay-off matrix become very small and consistence.

However, there is an issue which is that too many ally tokens are not being concerned due to the move strategy, even though they are extremely close to the effective enemy tokens. To avoid this situation and improve the defective strategy, the move of ally tokens which are very close to the effective enemy tokens would also be considered.

Ideally, the move actions would be pruned in an efficient way and the impact of applying this move strategy would be significant during the middle of the game (since there would be many tokens on board). Overall, it would reduce the computation time by not wasting excess process for checking sub optimal actions.