Artificial Intelligence COMP30024

Project PartB Report

Group Name: Invictus Gaming

Group Member: 1023617 Jiaqi Wu, 1067999 Chenghao Li

Overview:

IG is the name of the agent we created to play the board game ”RoPaSci360”. The structure which IG used was inspired by reinforcement learning where an agent learns by collecting reward and then constantly approximating the innate value function. However, due to our inability and lack of knowledge to implement some crucial techniques used in RL, like using Deep Neural Network to learn a value function approximator and using machine learning to find best features to represent states etc. In the end, we managed to implement a shallow paradigm which utilises the reward mechanism to make our agent perform complex actions. On the way of improvising, Li, by using the reward mechanism as an evaluation model, also developed a prediction system of the opponent which can predict the opponent's next greedy move and take advantage of it. Also temporal difference learning was also implemented to help IG to learn better value representation of each state he visited.

Keyword: Reinforcement learning, reward mechanism, prediction system, temporal difference learning.

Our approach:

Based on our understanding of the game RoPaSci360, we recognize some very interesting features of this board game. Obviously, one is the **simultaneous move** of both players, and the other is that only **one token** of each player is moved **in one turn.**

So, based on this understanding, we derived that in each turn, faced with the state, our agent can choose actions(slide, swing & throw) that aim to either **increase the value** of the current game(**Attack**) or **maintain the value** from dropping down(**Defense**).

By thinking so, we set up an **action evaluation system** to evaluate every possible action in terms of their aggressiveness and defensiveness. And IG will choose the action that collects the most reward from the action evaluation system. In doing so, IG can select much more greedy action in certain situations. For example, when being chased, IG will choose a special swing if a friendly token is nearby or slide to locations that are closer to the friendly tokens for future swing to get rid of hostile tokens.

What is more, beside using the action evaluation system to help IG choose the most appropriate move in the eyes of our tactics, we also use it as a **prediction system** to **constantly evaluate our opponent's move** and **get a sense of the opponent's play style**. After getting a hunch about the opponent's play style(like 10 rounds after the start of the game when we get enough data on our opponent), we use two statistics, **aggressive score and defensive score**, to indicate opponent's play style. And based on these two scores, we predict our opponent's move in the next round and take advantage of it.

Strategy: (defend as triangle, fight as a single warrior)

We would like to call our strategy: "**3-1**". Our strategy is very **defensive and risk-avoiding**. The tactic was inspired by Spartans. 300 Spartans managed to defend 200k Persians in the Battle of Thermopylae not only because they were well trained, but also

defense sometimes is much easier than attacking, as **weakness is shown while attacking** and the wise know how to take advantage of it.

3 refers to the three edges of the **triangle.** We use an open game strategy that deploys rock, paper and scissors each one on the board at the start of the game. **3 tokens form a very good defensive formation** because they are not only able to **eliminate any type of incoming hostile tokens** but also are able to use each other to **swing in order to do quick position reallocation**. Whenever our opponent uses any type of token to attack, we move our counter token at the top of the triangle like a shield. When hostile tokens are close, we use the swing move(like a spear) to eliminate them from a distance.

In the end, when the opponent loses enough tokens and is able to throw tokens right on the head of our token, that is 1 who starts to show.

One token is better at attacking than multiple tokens together because one player can only move one token at a time. If multiple tokens are in danger, the player can only save one. While one token is much more flexible. It takes advantage of the simultaneous feature of the game, making it hard for the opponent to catch one token. Also, one token can **hide in a hostile token of the same type and assassinate other target tokens**.

Reward mechanism:

At the start of each turn, we first generate all the valid moves that we can consider, and we use the reward mechanism to evaluate each action.

The Reward mechanism will give reward to our agent when he chooses actions that satisfy aggressive or defensive requirements. Below is a list of rewards that we would give when agents perform an action.

1. Chasing reward = +6: given to agent when he tries to move one token to eliminate opponent's token.
2. Elimination reward = +10*prob: given to the agent when his action can eliminate the opponent's token with high probability. This reward is always with the probability.(In our program it is ELIMINATION_REWARD * (1- 1/count), count = number of opponent's tokens that are in danger. In danger means the token is either in our throw range or is one-step close to our counter token)

3. Alert distance = +2: a number used to determine the perception distance of hostility
4. Approaching reward = +4: reward action that move tokens towards the position of target token
5. Avoid danger reward = +3: reward action that makes the token move when it is in the enemy's throw range in order to avoid being eliminated by throw. Or reward action that hide tokens into enemy tokens of same type
6. Intercept reward = +12: reward action that moves the counter token between hostile token and friendly token when they are 2-step close.
7. Scale = 2: used to differentiate the rank between slide and swing and between some other moves. For example, swing will get more reward when being chased.
8. Danger close flee reward +10: reward flee action when hostile token is only 1-step away.
9. Etc...

One action may satisfy multiple requirements and therefore get more than 1 reward. And IG will in the end select the action that has the most rewards. **The magnitude of the number of each reward matters**, sometimes an action that satisfies only one condition with the largest reward may outweigh the action that satisfies multiple conditions but with small rewards or in the opposite way.

By tuning the number given to each reward, we can manage to control our agent to perform the best action we want it to perform based on our strategy.

Prediction system:

In this part, we tried to use different methods of doing prediction, for example, using normal distribution to estimate opponent player's next action....

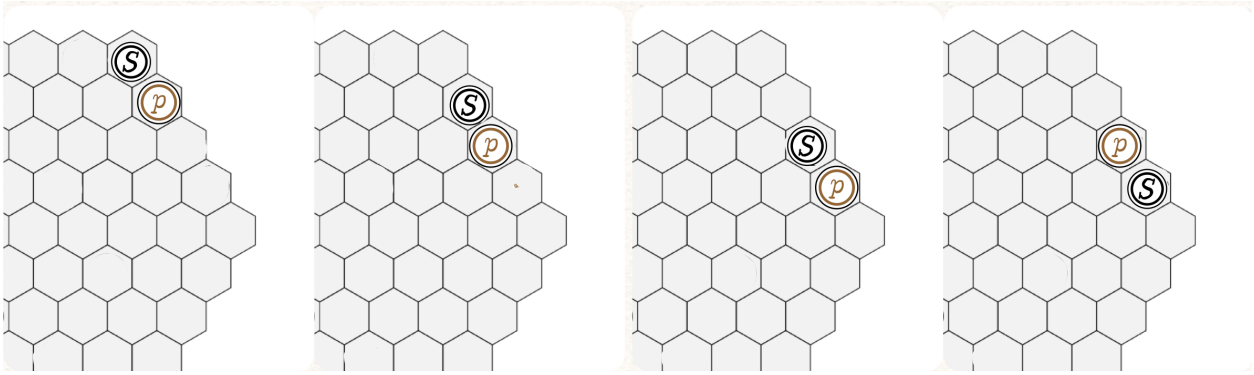
To achieve this, we build an evaluation function to use on enemy's action. The evaluation function returns a score list for each of possible action that enemy can take, the list is in sorted order from high to low, therefore, we recorded index of enemy's action in this list and get a normal distribution to do the prediction: enemy will choose on the action at mean index value of our score list.

However, as every algorithm use different evaluation function, so this prediction method may not be good. To further improve the performance, we introduced dynamic parameters for the evaluation function. Calculating each actual enemy action's aggressive score, defensive score, and whether they prefer to throw other than slides/swing. By comparing those detailed scores with our predicted action's corresponding score, we change the weight of each parameter (aggressive weight, defensive weight and prefer_throw weight in our evaluation function for enemy prediction.)

Later, since using normal distribution of their selected index along with this dynamic changing weight, the prediction may have worse performance, so in our last final submission, we replace the normal distribution of index with always choosing best actions enemy can take (index 0 at our predicted action score list using evaluation function)

We achieve average 30% accuracy using this method.

One interesting advantage brought by prediction: Position Flipping



As opponent(black) token continue to chase our token(red), the aggressive weights of opponent will increase and our prediction system will predict hostile token will follow our token, thus we do a position flip so that we could flee to friendly token or eliminate tokens in the future turn

Temporal difference learning:

We set up a value approximator to evaluate the state value. The equations is :

$$V = W * F$$

F refers to the features we generate that we think are related to the value of the state. And W represents the weights of the corresponding feature.

Below is a list of features we use:

1. Board_count: it equals the number of throws and tokens on board minus the number of throws and tokens on board of the enemy.
2. Num_throw: number of throws we have, throw become much more powerful in late game
3. Enemy_token_in_danger: it represents the number of opponent's tokens in our throw range or being chased at the moment.
4. Token_in_danger: represents the number of tokens of ours in enemy throw range or being chased
5. Token_on_board: the number of tokens we have on the board, it is quite related to the total ability on the board
6. Enemy_token_on_board: the number of tokens the enemy has on the board.
7. Mean_distance_to_attack: average steps for each token to eliminate their target token
8. Min_distance_to_attack: the least step of one token to eliminate opponent's token
9. mean_distance_to _defense: average steps for each token to defend hostile tokens
10. Min_distance_to_defense: the least step of one token to defend opponent's token, it represents the level of danger we are faced with
11. Support_distance: it represents how close each token is to each other

And in each game, we use temporal difference learning (lambda = 0) to adjust the weights of each feature.

Reference and Inspiration:

1. Reinforcement Learning Classes taught by David Silver, the guy who built AlphaGo, at UCL. https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver
2. Temporal Difference Learning at lecture slide week05