Love Letter: Project report

Contributors: 22289267 Hongfeng Wang
22289211 Hanoran Zhang

Literature review

1) Love Letter
   Love Letter (Kanai 2012) is a card game for two to four players. There are 16 cards in one game. There are a number and some words for explaining the effect of a card. When the game starts, cards in the deck would be shuffled automatically, and each player would hold a card which cannot be seen by other players. In the turn of each player, the player draws one card and plays one of the cards and follows the effect of this card. For the four players game, one card is removed from the deck and would never be used in this round.
   When all the cards running out but there are still two players or more alive, the alive players should show their card in hand and the player who show the card with the biggest value win this round. (Omarov et al.,2018) The goal of the game is to kick the other players out in a round and win enough rounds.

2) Simple Reflex Agent
   Simple reflex agent only acts on the current information from the state without considering the percept history. Russell & Norvig (2003) offer a figure “showing how the condition-action rules allow the agent to make the connection from percept to action.” When a certain situation arises, the agent will know how to react with minimal reasoning. The knowledge-based agent algorithm does not need any iterations and it runs faster than the rest of the algorithms with the time complexity O(1). The full set of checks and conditions can be found in the code repository.

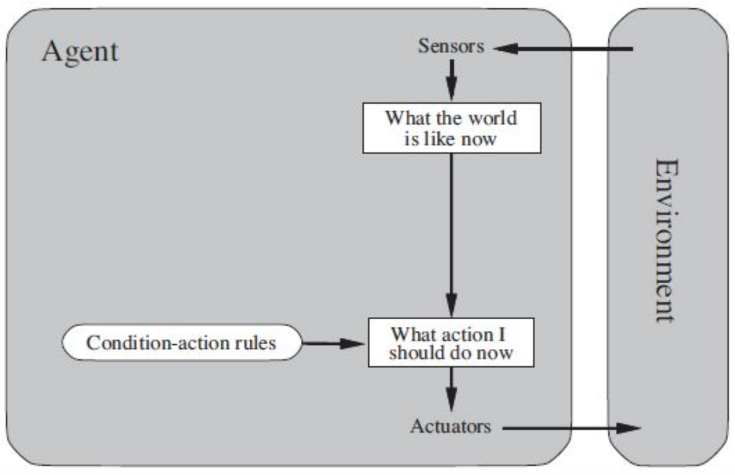


Figure 2.9 Schematic diagram of a simple reflex agent.

3) MCTS

Two fundamental concepts are included in Monte Carlo Tree Search:

I. *there is a way which can estimate an action's expected reward through a series of random simulation.*

II. *the search can use these expected reward to adjust itself toward a best-first search*

The algorithm uses the former explored result as an instruction to build a part of the tree continually. The tree nodes indicate different states while the gameplay and the link between two nodes indicate an action from the parent node to the child node. The goal of building the tree is to estimate the reward of an action. In general, with the development of the tree, the estimation will become more and more accurate. A time, memory or iteration constraint can be used as an indication of the end of the iteration. In this way, MCTS can extend the most promising areas of the tree, thereby avoiding wasting most of the computational budget and time on less interesting moves. We wish that whenever the computer stops searching, it can return the current best performing root action. There are 4 basic steps included in the MCTS algorithm, as shown in below.(Stefano,2013-2014)

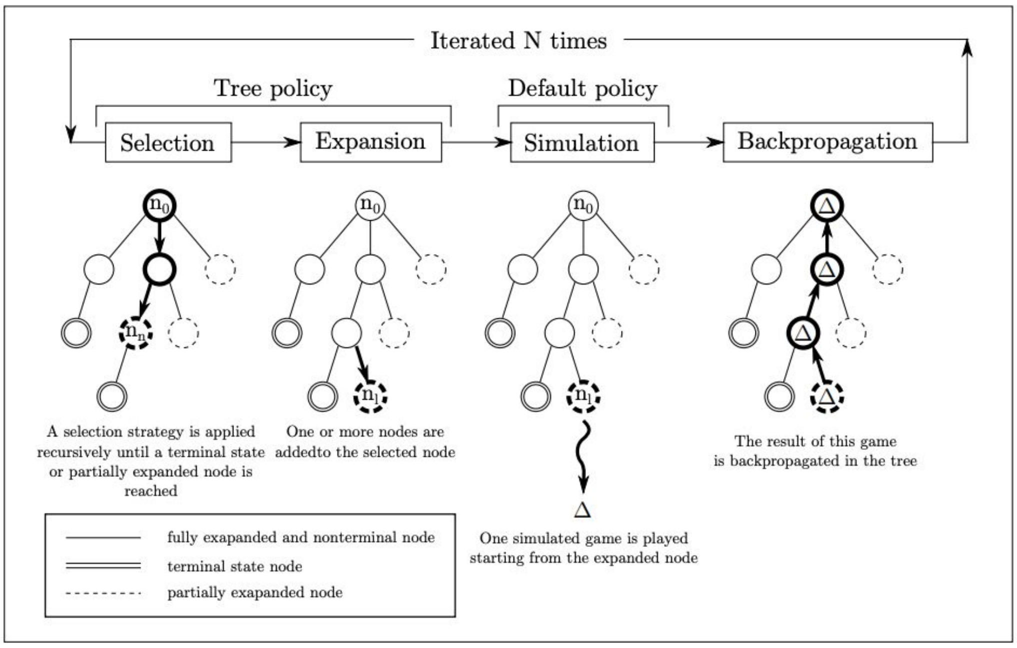

Iterated N times
Tree policy
Default policy
Selection
Expansion
Simulation
Backpropagation
n_0
n_n
n_l
Δ
A selection strategy is applied recursively until a terminal state or partially expanded node is reached
One or more nodes are addedto the selected node
One simulated game is played starting from the expanded node
The result of this game is backpropagated in the tree
fully exapanded and nonterminal node
terminal state node
partially exapanded node

Figure 2.1: Steps of the Monte Carlo Tree Search algorithm.

• *Selection*: the algorithm iteratively find a node which contains the outstanding value given by a utility function until it finds the leaf node, which means either the terminal state is reached or the tree is not fully expanded.

• *Expansion*: if a leaf node doesn't represents a terminal state, it means that there still are some actions in this state can be executed. Then, one action will produce a node.

• *Simulation*: starting from a node and then keep simulating until getting to a terminal state.

• *Backpropagation*: after simulation, update the reward value and number of visits in the node which is the start node of simulation.

Selected technique

1) Simple Reflex Agent

For the simple reflex agent, let alone the uncertainty, it considers the current state with the cards agent holds and the target we choose to take the card effect on and specifically, the card-guessing when plays the guard card. Omarov et al. (2018) suggests that the minimum value of the card would be chosen to play with. However, an exception can be found is that when the Countess card along with Prince or King in the hand, the countess must be discarded. Secondly, as we aim to win the game by earning the greatest score, the target we play card toward is the player having the most scores at the current state. When the card we play is guard, we need to guess the card using the card we have not seen.
By applying simple reflex agent, the agent will react like human with less probability playing irrationally. In addition, applying simple reflex agent can guarantee the swift response as its complexity is O(1).

2) ISMCTS(Information Set Monte Carlo Tree Search)

There are a lot of ways to implement an AI in the game.For the hidden information game, the most common way is to use determinize. The main idea in this way is to make a guess of the hidden information. For example, in our game, we don’t know what cards held by the other players, but we can guess what kind of cards they are holding depending on how many cards remaining and the cards which have been played. Then, based on a representative sample of hundreds or thousands of conjectures, we make the final decision for which card we are going to play. There is a probability of making a wrong decision by the individual guesses. However, if we have a big enough amount of guesses, we have a big chance to do the best decision based on the information caused by the guesses. There is a popular way to implement the MCTS algorithm called perfect information Monte Carlo(PIMC). However, this approach has some weaknesses and seems that it is not a good way to build an AI for Love Letter game.”PIMC has proven successful in several games, particularly Contract Bridge (Ginsberg 2001). However, the approach has two serious flaws. First, treating the game as one of perfect information means the AI will not see the value of gathering or hiding information, and will wrongly assume that it can tailor its decisions to the opponent's cards. Second, we have multiplied the amount of work our AI needs to do: each "full" decision is an average over tens or hundreds of determinization decisions, each determinization decision being as difficult as choosing a move in a game of perfect information.” (Edwardet al., 2013)
That is the reason why we used information set Monte Carlo tree search rather than PIMC. The idea of randomly guessing the hidden information is kept in ISMCTS, but we treat each guess as the instruction to guide the next iteration rather than treating them as an independent game. Unlike PIMC builds a lot of trees and averages at the end of the game, ISMCTS just grow one tree and average it at the end of the state.

Implementation

1) ISMCTS

We create three classes to help us implement the ISMCTS algorithm. What follows are the explanation for the functions we used.

1. *Class ISMCTS*: for implementing the five steps in ISMCTS
   1.1. *processing()*: the function exists in class ISMCTS. It is used to do the five steps in ISMCTS(determinization, selection, expansion, simulation, backpropagation)

2. *Class Node*: for implementing the tree made in the process of ISMCTS. Each node represents a state.
   2.1. *GetUntriedMoves(ArrayList<Action> allPossibleMove)*: this function requires all possible actions which can be executed in the current state (different card and different target) and check which action has not been tried. It will return an arraylist which contain these actions.
   2.2. *UCBSelectChild(ArrayList<Action> allMoves)*: this function also requires all possible actions which can be executed in the current state. When the tree is fully expanded and still have some iterations needed to be done. We use UCB formula (figure 2) to gain a value based on total reward and number of visits, to decide which node is worth to do more simulation.

$$\frac{\text{total reward}}{\text{number of visits}} + c\sqrt{\frac{\log(\text{number of parent visits})}{\text{number of visits}}}$$

Figure 2. UCB1 formula

   2.3. *AddChild(Action act, int lastplay)*: we use this function to build the tree. In order to add a new node in the tree, we need to know who is the parent (lastplay) of this node and what action (act) parents do will link to this node.
   2.4. *Update(Mystate state)*: when a node is a leaf node, which means that we reach the terminal state(all the cards in deck running out or only one player alive), we need to update the information (number of visits and the reward value) of this node and its parent node.

3. *Class MyState*: a clone of the current state. We need this clone to help us simulate the state of the game behind and make the best choice according to the result of simulation.

2) Simple ReflexAgent

The agent follows the following process to determine the action:

a) Find out whether all the players are protected by the handmaid
b) Build a ***Hashmap*** to store the visible card from the player
c) Card selection: play the lower value card
d) Build a target ***List***

Sorts the list with the player's score in descending order and puts the protected player at the end.

e) Build guessing card list

The list initialises with the unseen card from the current state. It then deletes the guard card in the list, which we cannot guess with the guard card.

f) Switch the behaviour with the card chosen

i) Guard

• find the know player in the hashmap
• otherwise choose randomly in the unseen card

ii) Baron

If we know someone's card value lower than the agent's card, we target the player to eliminate him

iii) King

Put the card in my hand to the hashmap with the target player as the key, so that we record the card in the target hand

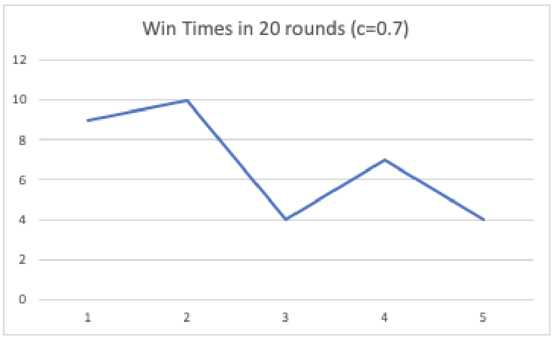
g) Return the action

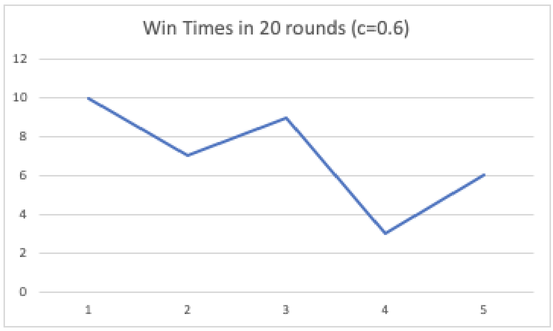
The data structure we used in this agent are hashmap to store the known cards and an array list to store the sorted target player. The hashmap make sure the known card is the latest as each player index as the key can correspond to one card only. The ArrayList offers the method to delete the card object in the list and easy way to swap the card in the list.

Performance Analyse

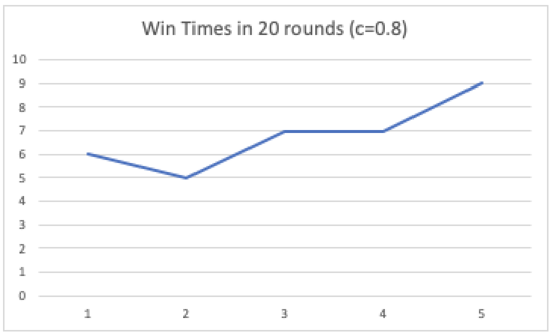
1) ISMCTS Agent

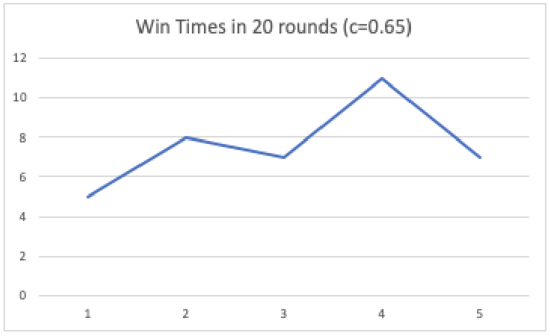
   As the iteration is time-consuming, we run the ISMCTS agent 5 times using four different values of c in UCB selection, which are 0.7, 0.6, 0.65, 0.8. We set c=0.65 because of the higher winning ratio of 0.6 compared to 0.8. With the varied iteration, the winning percentage varied from 34% to 38%, peak at c=0.65. Increase the iteration time help the winning rate a bit, however, the iteration times beyond 1000 does not do any help except the longer running time. The expectation to the win rate is around 70% when using ISMCTS AI vs the random AI. From the viewpoint of the win rate, we believe that our ISMCTS algorithm doesn't help us find the best action in current state. As a result, we think that there are some problems existing in the part of selection and tree building. In our implementation, our ISMCTS trees only have two layers. We guess whether the size of the tree causing the improper simulation. When we inspect the final tree after the last iteration, we found that the probability of winning in each child node are close, which means the final behaviour of the agent is similar to the random agent. Moreover, whether the UCB1 is not suitable for our implementation. These questions should be researched further.





Average: 34% 35%

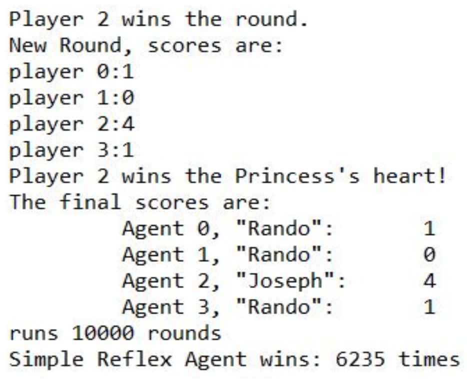




34% 38%

2) Simple Reflex Agent

   As the agent determines the actions by using the if-else statement, the game can run ten thousand round in thirty seconds, so we run the game in 10,000 times with three random agents first. The winning times count as 6235, which means the agent goes to the winning rate of 62.35%. The outcome shows as below. When placing two simple reflex agents within the game plays with two random agents, the aggregate winning rate goes up to 77.43%. We also found that placing the agent as the third player the winning rate would go up around 5%, the reason is unknown although we guess that it is to do with the game mechanics. However, the result is based on other players playing randomly. To get better result when playing with an intelligence agent, the agent should calculate the card probability in others hand using the discard pile. For example, when someone discarded the countess, the probability of this player holding king or prince on the hand is higher. In conclusion, the simple reflex agent can mostly beat the random agent, but the performance toward intelligence agent need to be improved.



```
Player 2 wins the round.
New Round, scores are:
player 0:1
player 1:0
player 2:4
player 3:1
Player 2 wins the Princess's heart!
The final scores are:
         Agent 0, "Rando":       1
         Agent 1, "Rando":       0
         Agent 2, "Joseph":      4
         Agent 3, "Rando":       1
runs 10000 rounds
Simple Reflex Agent wins: 6235 times
```

Reference

Edward Powley, Daniel Whitehouse, Peter Cowling(2013), Reducing the burden of knowledge: Simulation-based methods in imperfect information games. http://www.aifactory.co.uk/newsletter/2013_01_reduce_burden.htm

Kanai S. (2012). Love Letter : Tempest Edition.

Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2, chpt. 2

Stefano Di Palma(2013-2014), Monte Carlo Tree Search algorithms applied to the card game Scopone. http://teaching.csse.uwa.edu.au/units/CITS3001/project/2017/paper1.pdf

Tamirlan Omarov, Hamna Aslam, Joseph Alexander Brown, Elizabeth Reading(2018), Monte Carlo Tree Search for Love Letter. https://www.researchgate.net/publication/327679828_Monte_Carlo_Tree_Search_for_Love_Letter