*Love letter*

An Implementation using Artificial Intelligence

*Nicholas Walters 22243339*

*Shikai Wang 21938451*

Love Letter – AI Implementation

**Literature Review**

Introduction

Love Letter is a game of risk, deduction, and luck for 2-4 players. The goal is to get your love letter into the Princess’s hands while deflecting the letters from competing suitors. From a deck of only **16 cards,** each player starts with a state of only one card in the hand. On each turn, you are supposed to draw one card and then remove/play one card, using the effects of that card played. Each card has different effects and values when played and will affect the other opponents depending on each card played. The goal is to expose others and knock them out of the game by knowing their cards in hand and playing moves accordingly.

Love letter is a card came with **incomplete information**. Each player is able to view their own hand, but not others (some exceptions). In order for a player/agent to win a game, the player needs to know or guess the hidden cards of the other opponents first. Once an idea is created as to what an opponent may have, a game state is created so that the player can try and win the game, by knocking out the players with known/guessed cards. There is some reasoning made when forming a guess of the other players card. This paper discusses some of the strategies that you can apply to deal with the limitations of incomplete information.

Ways to figure out what players may have, would be to play cards (such as the guard or priest etc) and figure out what the players do and don’t have. Also, as there are only 16 cards in a deck, a realistic probability of how many of each specific card that each player may have can be calculated (card counting).

If the agent has complete access to view all the opponent’s cards, then there would need to be less moves played to reach a desired goal. However, with incomplete information about the game state and opponents’ cards, there is some uncertainty applied and there could be many possible game worlds.

* At the start of the game, everything is based on luck, because there is no knowledge of other players cards and their values. There are too many different game scenarios to be played extrapolating from the start.

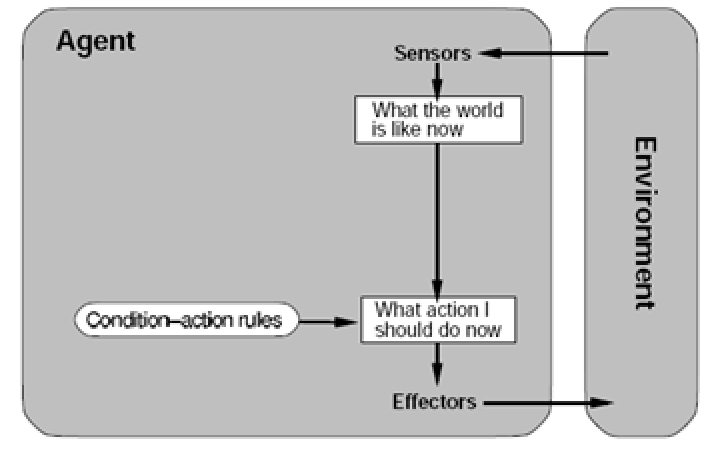
However, we can assume that at least 1 player has a high value card, as we start from a 16-card deck.

* As more cards are played, some data can be recorded as to what cards each member may have. For example, in each turn, as players show their cards, become immune to some plays, or holds onto cards for a long period of time (such as high value cards which shouldn’t be used, like the Princess) then more of an idea of the game states and cards held can be formed. Every turn that is played, it could give one or two new pieces of information for the agent.

Humans are able to alter their perception of their hand and note which cards are seemingly worthless or have high value, based on the behaviour of others. An agent does not necessarily have this sense.

**Rule Based Approach**

Makes pre-defined decisions based on a set of rules to follow, depending on the game state/situation (current environment). If the agent can guess another player’s unseen card with high probability, thanks to the discarded deck, it will execute that decision. This is a simple reflex-based approach, as shown in (Jeffrey Bradshaw, 1997):



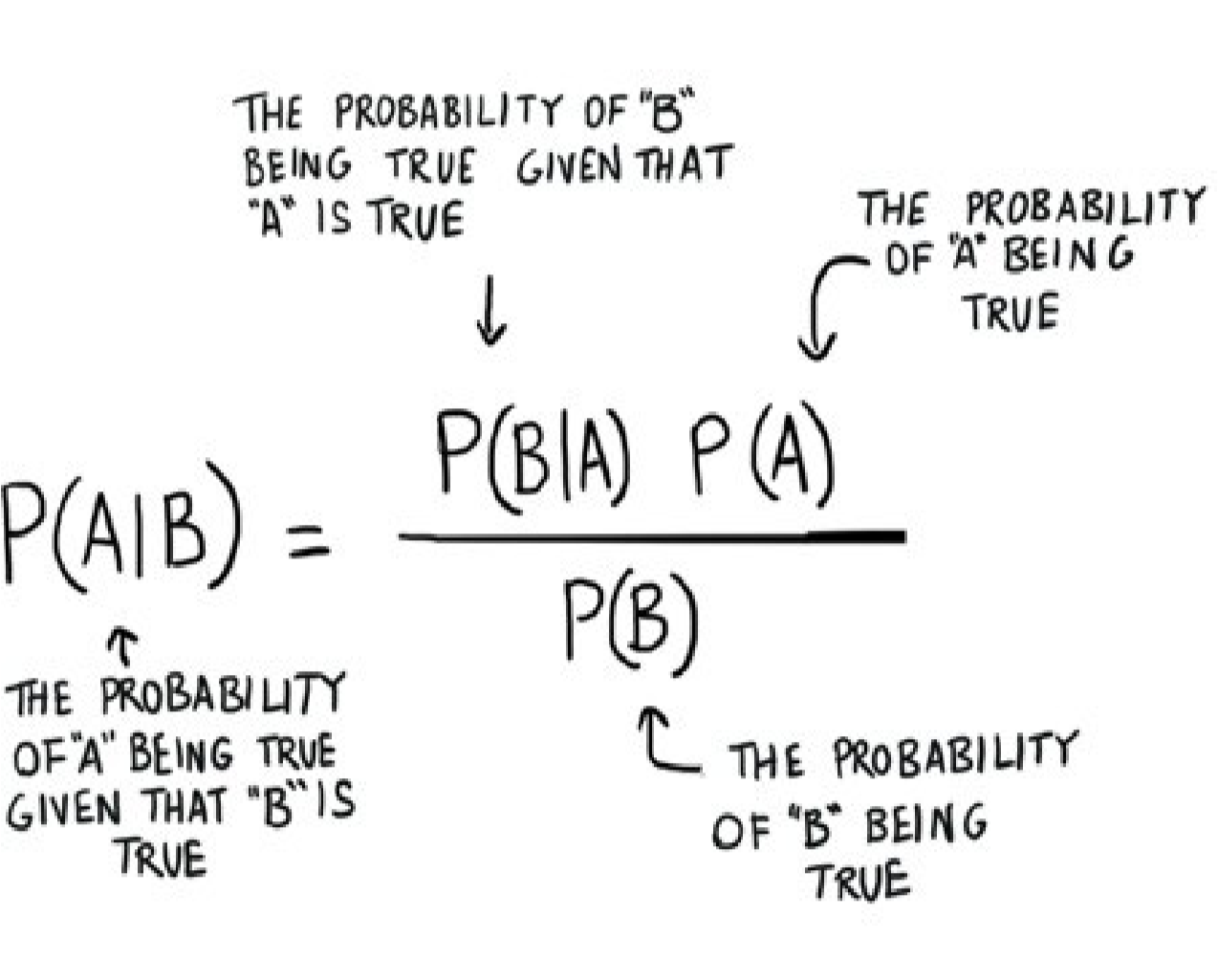
When playing ‘Love Letter’ there are a number of rules/strategies that people unconsciously follow when playing the game:

* By default, Discard/Play the lower value cards, and keep the higher value ones until the end of the round
* Play the Baron only if you have another relatively high-ranking card, and the probability that the opponent has a higher card is low (by looking at unseen deck)
* Play the King only if there are more Unseen cards which are ranked higher than your current card in hand (i.e. Don’t play the King if the probability of getting a worse card is high)
* If the player has seen another person’s card and knows their card value, you can play a guard or baron and try to eliminate them
* When taking an un-informed guess using a guard, count the discarded guard deck, and find the card which has been played least and has the greatest number of cards left (i.e. greatest number of unseen cards).
* If you know that another player has a high value card, use prince on them to make them discard theirs. If you have a low value hand, then discard your deck.
* If a player has played a countess, then most likely (not always) they have a prince or king.
* Always Target a player who has the highest number of rounds won so far (except yourself).

(These rules have been implemented in the code provided)

The above strategies were used in the implementation of this rule based, reflex agent. Some more strategies can be applied such as Bayes Theorem, which is **an extension to the rule-based agent:**

This agent extension will some similarities to the Bayes theorem, as it figures out the probabilities that an opponent will have a card, where the models of opponent strategies have been provided already.



In this selection strategy, the agent memorises the deck of unseen cards. The agent considers two scenarios to guess a player’s card. Firstly, it considers both the number of unseen cards, and secondly it acknowledges that the target player will likely keep the higher valued cards over the smaller ones. For example, if there is 1 unseen Baron and 1 unseen King in the deck, the selection criteria would be skewed towards the King. These scenarios help the agent to determine the probability of what the target may have in his hand and play a card accordingly.

If there is a choice between A (Baron) and B (King) based on values in the unseen deck, the opponent is likely to keep the higher value card B and play the lower one A (Baron).

However, this strategy will not work on the Random agent, because it discards anything regardless of value and has no strategy.

**Monte-Carlo Tree Search (MCTS)**

MCTS is a heuristic based algorithm which expands the nodes which have the highest probability of returning a good outcome first (but also explores other possibilities that don’t look favourable). MCTS involves Selection, Expansion and Simulation (and Backpropagation).

In the case of love letter, we would use a (Single Observer) SO-MCTS algorithm, as the agent doesn’t know the values of other players cards. MCTS works the best if it knows the the entire state of the game, and possible plays of the opponent’s card are known, but in love letter, this is not the case (uncertain).

A node is traversed from the root node to a leaf node, the child (move in Game) that maximises the UCB1 formula is selected.

When a leaf node is reached, the expansion stage adds all the legal and valid actions as children. Simulation stage will play out a round of the game and determine its probable result. This result is used to update information along all nodes, from node to the root (backpropagation).

MCTS is efficient in being able understand the current state of the game in ‘Love Letter’. If you can run MCTS for a long period of time, then it will give greater reliability, but it can still be stopped at any time and return its best move so far.

**Rationale, Selection and Implementation of Agents**

The probability-based reflex agent was finally selected after much testing of viable techniques.

It is easy to see that the technique of rule-based probability agent was chosen because there are not many other ways to have an edge/advantage a game which involves too much uncertainty. The only advantages gained may be in the case where you can remember the cards already played/discarded (card counting), keeping higher value cards and assuming others do the same, and finally guess using the seen cards (priest).

The probability-based agent will prioritize the elimination though the cards of higher value and consider the number of cards in the unseen deck. It will also be a ‘Conservative Agent’ as its technique is very defensive in nature. In this case, the agent will only play the risky move of baron (or King) if the probability of winning the comparison is greater than 60%. We found that 60% probability is a good trade-off, as we don’t want to discard higher value cards too often. It will not play the baron especially if its chances of losing the comparison are low.

In less risky moves such as playing the guard, or prince the agent will also make an educated guess based on what has been already played.

The probability agent used the complimentary function unseenDeck() to return an indexed array corresponding to each card. This passed information to the guess(), playKingOK() and playBaronOK() function will also calculate the probability of returning a good answer based on that deck.

While the Monte Carlo Tree Search Algorithm (MCTS) is a viable technique for this game, it was dropped after some prototyping and experimentation. MCTS method requires simulation the actions other players will take, but without knowledge of the opponents hand it cannot simulate a game. MCTS required too many new classes, such as a new State class, and graph/Node classes and did not yield any noticeable results for the effort involved.

Simulation in MCTS requires little or absolutely no uncertainty. As there is so much uncertainty around what a player may hold, it was not viable in our approach.

The Single Observer MCTS (SO-MCTS) considers the uncertainty of a game, and is designed to accommodate unknown variables, but making incomplete simulations were questionable at best, as it is no better than selecting random moves.

**Data Structures and Code Implementation – basic Agent**

Global Variable:

1. playedCard = {0, 0, 0, 0, 0, 0, 0, 0};

Record all the card that previous player played, if one of the card type is collected fully then these card would not be a target.

1. topScorer;

Record the player with the highest score and agent will keep targeting it.

1. myIndex;

check if it is my turn and get the current card from hand.

Guess Function:

Returns the biggest card value from unseen cards. Created an array called deck in order to store the card that still unplayed in the deck. Current hand card need to be deducted and always target the highest value in the deck.

countessPlayed Function:

Obviously, this function is going to check that if any opponent has played countess. Iterating all played card and check each turn if other players played countess, if yes return true.

GetHighestPlayer Function:

In this function, find the player with the highest score in each round. Each player’s score will be stored in hashMap excluding own agent, being protected player/s, or the player who is eliminated. If all players are protected by handmaid, then the agent randomly choose a player.

allUnableToEliminate Function:

This function is to count if any players are not eliminated or being protected by handmaid. Return true if not opponent can be targeted.

fillKnowCards Function:

The aim is using maps to store all cards. Using TreeMap is easy to find which player played what value of the card.

Action Function:

Basically, this function will perform an action after drawing a card from the deck. As time goes by, the agent got more information from nonknowledge-based to knowledge based, handles the selection of the guess card as. The card which is encouraged the minimum number of times in the played deck is chosen. Therefor the algorithm can reduce the number of non-played cards for each character in the game. In this function, defined a variable called handCard and drawnCard. In order to determine which is the best card to play, agent check if current hand has countess and prince or countess and king card. If agent is holding a princess, agent will always play the card with lower value. For the guard, if a player has played a countess at last turn then target the opponent and guess the opponent has a prince or king. For prince card, if other players are being protected by the handmaid then choose itself as a target.

**Data Structures and Code Implementation – Probability Agent**

* TreeMap<Integer, Integer>: stores the known cards from using the priest {PlayerIndex, cardValueInHand} a TreeMap is used because it can easily extract the value of a person’s card given their player index.
* Int [] deck stores all the unseen cards in an array. Each index represents a different card, and their associated card count (unseen count). Here is an example where there are 2 unseen barons, 1 guard and 1 unseen princess

{1,0,2,0,0,0,0,1}

* HashMap<Integer, Integer>: used for storing the players and the number of rounds that they have won. This is used to target specific players that have high win rates that we want to target first, to prevent them winning further.

{PlayerIndex, roundsWon}

**Validation**

**Analysis of the Probability Agents Performance against 3 random agents, over 100 games (4 Players Total)**

The probability agent performs above a higher standard than the random and basic agents. This can be contributed to the fact that the basic agent has no concept of its environment and relies on randomness for its next move.

References:

Rule Based Agents, Compliance, and Intention – Antonino Rotolo

https://page-one.springer.com/pdf/preview/10.1007/978-3-642-22546-8\_7