Love Letter: AI Agent

Bruce How 22242664

October 23, 2019

Abstract

There has been major advancements with Artificial Intelligence agents in the game playing scene with its ability to demonstrate a high level of intelligence, and it's capability of defeating even the best players in the world. In this report, we will discuss some suitable techniques that can be utilised by an Agent to approach the card game, Love Letter. We will highlight the differences between these techniques and the rationale behind the selected technique as well as provide an analysis on it's performance.

1 Literature Review on Suitable Techniques

Love Letter is a card game for 2-4 players. The goal for each player is to get their love letter to the princess by passing it to an intermediary, or directly to the princess. Each player has only 1 card in their hand at a time but draw an additional card on their turn. Players can choose to play between one of the two cards on their turn, each of which have a different action that will affect the state of the game in it's own way. Without knowing the hands of every player and cards remaining in the deck, we can understand intuitively the difficulty for one to determine the most optimal play at any stage in the game.

Each card in Love Letter is assigned a numerical value. At the end of the game, the player with the highest numerical value, assuming that they are not eliminated, wins the round. We can extend this idea to implement a distributed greedy algorithm based agent which are agents that make decisions in a distributed and asynchronous fashion utilising only information from the current state in a game [1]. Each turn, the agent would be prompt with a simple local maximisation problem [1] such as optimising its hand to the highest numerical value i.e. by always playing the lower valued card when possible. However, it is imperative to understand that a card's numerical value does not necessarily represent its beneficial value. For example, the Guard, although has the lowest numerical value, has the potential to eliminate another player given a correct guess while the King, with a reasonably high numerical value, may result in an unfavourable hand trade. In any case, playing any of these card's actions may be more or less favourable depending on the state of the game.

We can extend this idea to implement a knowledge-based agent which is one that uses a heuristic evaluation function to determine the most optimal move [6]. The developed method for generating the heuristic evaluation functions are based on the rules of the game, how close an action progresses the agent to its goal state, and the probability of an action being correct. At every iteration, the agent takes into account the current state of the game where it has more knowledge as the game progresses, hence it's name, and is able to make more logical decisions.

The Monte Carlo tree search (MCTS) algorithm is a computerised mathematical approach for making globally optimal decisions in many decision problems. One notable example for the use of the MCTS algorithm is DeepMind's AlphaGo which proved to be successful after managing to defeat the world's best human Go players. AlphaGo uses the Monte Carlo tree search and is guided with artificial neural networks (NN) to maximise it's performance and efficiency [3]. However, due to the time frame of this project, the implementation of a random simulation cut-off value would suffice.

The basic idea behind MCTS is simple. The algorithm repeatedly performs four steps including selection, expansion, simulation and back-propagation. In the selection step, the agent selects the node to begin the search. This is based on the state of the game and is done with a controlled balance between exploitation and exploration [4]. During the expansion step, if the selected node is not a decisive node (determining the winner/loser), the agent will generate a list of child nodes representing the possible states that could occur upon performing any legal actions at the current node. Due to the generation of states, it is necessary for there to be a finite number of actions at any given state [4]. This is the case for Love Letter where there is only a finite set of actions that can be performed at any given stage, e.g. playing the hand or drawn card, targeting different players and different arguments (card guesses) with specific cards (such as the Guard and King). During the simulation step, the agent simulates a random state to the very end to determine the player's probability of winning upon choosing a particular action. In the final step, the agent propagates from the leaf node of the simulated game and through the previously traversed nodes up until the root node. The result of the action is calculated positively and negatively depending if the simulated game resulted in a win or loss respectively [4].

Artificial Neural Network (ANN) is an information processing system that is derived from the mathematical models of biological neural networks [5]. An ANN is a large composition of nodes (neurons) with an input and output layer and in some cases, several hidden layers between (known as a Multi-layer ANN) [5]. Each node takes a particular input and simulates several actions leading to other nodes until it reaches the output layer. The link between each node is assigned a weight which is altered during the training phase to simulate learning. An agent implementing an ANN would require training to learn (by tweaking weights) the outcomes of its action. Because of how open-ended the technique is, it is been widely employed to solve many difficult problems and has been frequently applied to map a given set of input patterns to the target outputs [5].

We can use this idea to mould an agent for our game by creating nodes for every action that can be played. The agent will need to be trained (playing games) against other agents where it will learn to distinguish what actions played at any particular states is more favourable towards winning.

2 Rationale of Selected Technique

With the distributed greedy algorithm approach, we could easily determine the best locally optimal action in almost constant time. This algorithm could be guided by a series of conditions to always optimise a player's hand at every given state such as playing the card with the lowest value and/or always playing the handmaiden card when possible. However, as this technique only considers the player's current hand and disregards all surrounding information at every state, such as the action of each card and the potential known cards of the other players, the algorithm may not necessarily make the most globally optimal decisions towards winning. A simple greedy based agent would outperform a random agent because playing the most locally optimal action has a better chance towards winning than randomly performing an action with no concept of the game state. However, when playing against agents that optimally utilise the time limit to consider several actions and possible states in advance, the algorithm falls behind.

The idea behind an ANN-based agent is to be able to thoroughly train the agent to a point where it can learn from its mistakes. There are two main forms of training including unsupervised and supervised learning. With supervised learning, the agent is trained with real input and expected output data. However, due to the high uncertainty and variations with Love Letter, such data is difficult to obtain. In unsupervised learning, only input data is given. This could represent the series of actions that could be performed at any given stage where it's goal is to model the underlying structure for that action. The agent could then learn based on the outcome (winning or losing the game) the weights of particular actions. This technique was not selected due to the difficulty of completing the required tasks in the given time frame. This included the finishing the required coding, and training the agent which proved to be a difficult task. An attempt was made to train the model using an external server utilising `tensorflow` but was not very fruitful.

The structure of the MCTS based agent can be easily applied in the context of Love Letter due to the idea that at any state, actions can be performed generating many different other states. The MCTS based agent was the initially selected technique that was dropped after failing to perform simulations in a reasonable time frame.

The selected technique was the knowledge-based agent. The rationale behind the selection of this algorithm was because of its balance between simplicity and performance. It is an extension of the greedy heuristic based agent that also considers information from the current state allowing it to make more logical decisions that is also based on probability. Each action is assigned a priority

where the action with the highest priority (most optimal action) is performed. The idea behind this fits very well with `Java`'s Priority Queue structure as it allows us to generate a list of actions, and `poll` the best action to be performed by the agent. Nevertheless, the use of probabilities such as determining the best cards to guess could be easily mapped using a `HashMap`. Because these data structures were already provided within Java, the implementation of this technique was not a difficult task.

3 Validation

Validation of the agent's performance was done by playing many games (100,000) against the random agent. The validation metric used was the win rate of the agent. One important heuristic implemented was that the agent should play actions with different priorities based on the probability of a certain cards appearing. Because of how widely used probabilities are within this knowledge based agent, there is a natural preference for using the `double` datatype over `float` due to its better precision.

Due to the competitive nature of the game, different cards may be differently prioritised by the agent depending on how knowledgeable and aggressive the agent is or has to be. In any case, each card has its own base priority indicating the card's natural preference and is adjusted according to the game's state. The base priority values are altered after playing many games to improve performance, which is measured by it's win rate. It is imperative that the game seed is different each game (randomised) to ensure that the agent's base priority values, when altered, are not over-fitted.

A vital part of creating a knowledge-based agent is to implement heuristics that will improve performance. In order to access the value of each heuristic, the agent's win rate is recorded with and without each heuristic. This allows us to analyse the change in performance from each heuristic (Covered in section 4).

4 Agent Performance

The implementation of simple heuristics such as never playing the Princess and always playing the handmaiden resulted in an increase in the agent's performance (measured by it's win rate) by `23.12%`.

Simple Heuristic Implementation	
Heuristic	Win rate
None (random)	24.83%
Simple heuristics	27.95%

Table 1: Agent performance with and without the implementation of simple heuristics

From the above Table 1, we can see that the implementation of a few simple heuristics can significantly increase the agent's performance. So let's implement more. At every state, the agent has access to a list of unseen cards and known cards for each player. We can use this information to make probability-based decisions when targeting players with aggressive cards such as the Baron and Guard. The agent's performance also increased slightly when prioritising actions against opposing players with a higher score. This increased aggression towards players closer to winning decreases their chances of winning (through elimination) which is beneficial to the agent. If we implemented logic-based heuristics, for example, one that is based on the rules of Love Letter where a player discarding the Countess card is likely to have a high-value card; only marginal performance increase is observed (Table 2).

Additional Heuristic Implementation	
Implementation	Win rate
Simple heuristics	27.95%
Probability Based Guesses	50.33%
Targeting winning player	56.09%
Rule based logic	58.73%

Table 2: Agent performance with and without the implementation of different heuristics. Each row includes the heuristic implementations from the previous rows.

There was a negligible performance increase when adding logic-based heuristics to the agent. This is because random agents have no concept of the game and may perform illogical actions.

Agent Performance		
Agent	Wins	Win rate
Agent22242664	31287	62.57%
RandomAgent	6489	12.97%

Table 3: Agent performance against three other random agents after 100,000 games

After performing several minor tweaks and priority value adjustments, the agent was able to maintain a `62.57%` average win rate when played against the random agent (Table 3). There is quite a significant improvement in the agent's performance after adjusting the base priority values and logic behind the priority changes. As this was done manually, it is not necessarily the most optimal combination of priority values. In an ideal case, these values should be modified automatically in a similar fashion to how an ANN modifies the weights between neurons.

References

[1] Qu, Brown & Li. (2018). Distributed Greedy Algorithm for Multi-Agent Task. 2, Retrieved from https://nali.seas.harvard.edu/files/nali/files/submodular_automatica2018.pdf

[2] Zhao, N., Guo, Y., Xiang, T., Xia, M., Shen, Y., Mi, C. (2018). Container Ship Stowage Based on Monte Carlo Tree Search. Journal of Coastal Research, 540-547. Retrieved from https://www-jstor-org.ezproxy.library.uwa.edu.au/stable/26543014

[3] Duckett, C. (2019). Google AlphaGo AI clean sweeps European Go champion — ZDNet. Retrieved from https://www.zdnet.com/article/google-alphago-ai-clean-sweeps-european-go-champion

[4] Nijssen & Winands. Monte-Carlo Tree Search for the game of Scotland Yard. In Computational Intelligence and Games (CIG), 2011 IEEE Conference on, pages 158–165. Retrieved from https://ieeexplore.ieee.org/document/6032002

[5] Nezhad, A., Mahlooji, H. (2014). An artificial neural network meta-model for constrained simulation optimization. The Journal of the Operational Research Society, 65(8), 1232-1244. Retrieved from http://www.jstor.org.ezproxy.library.uwa.edu.au/stable/24503183

[6] S. Haufe, D. Michulke, S. Schiffel & M. Thielscher. (2010). Knowledge-Based General Game Playing. Retrieved from http://www.cse.unsw.edu.au/ mit/Papers/KI11a.pdf