Student No's: 22260157 & 22243849

CITS3001 Research Report:

Love Letter – AI Tournament

Reviewal of Suitable Techniques

Love Letter is a social deduction game of cunning, bluffing and deceit. Players are required to eliminate all other players or to hold the most valuable card by the end of the round to secure a point. The first player to gather four points wins the heart of the princess, and thus, the game. Given the random nature of a card game with a shared, shuffled deck, a player's objective is malleable. If a point seems unattainable in the current round, a player should aim to instead keep victory from higher scoring players, cutting their losses until the next round. The game being built upon gathering information as most of it is initially hidden from the players. They must consider its between one another, doing their best to wield and manipulate it. With it, randomness can be reduced, certainties can be derived, and deceptions may be perpetrated. This report will investigate some of the different possible gameplaying agent implementations, and their compatibility with Love Letter.

The inherent randomness in Love letter makes it difficult to measure the overall effectiveness of an agent's algorithm. The results produced are not necessarily stable, especially against agents with their own strategy (Omarov, Aslam, Brown, Reading, 2018). This factor means it is also difficult to decipher whether one agent implementation outperforms another. As such, a heuristic-based agent using a human-implemented strategy can be of a similar effectiveness as other, more involved agents. The absence of a tree search means it can perform quickly, simply selecting the highest priority move given its legal options. As it also does not require any training, it is effective from the get-go, not requiring anything more than well-implemented, and well-devised heuristics. This will be discussed further in the Rationale section of this report.

Genetic algorithms (GA) are a type of adaptive heuristic search which follow a similar premise to the human concepts of natural selection and genetics. It is analogous to genetics in that the population of potential solutions is divided into different chromosomes comprised of many genes, each of which regards the handling of a specific variable (Lin, Ting, 2011). Game states of strong "fitness scores" are made to "mate", producing "offspring" to see if the advantages of each can be combined into better alternatives. Once enough possibilities have been explored that the offspring no longer perform better than their parents, the population is made into a set of solutions for the problem.

While a GA does present a strong candidate for implementation, it is also particularly time-intensive in its inception. Love Letter in an inherently random game, making even the most effective agent

bend to the whim of its haphazardness. It is important to consider the time available to complete this project. Given that no agent can completely overcome the randomness of the game, it may be unwise to dedicate efforts to devising a GA solution. It would bear little to no benefit over another, more simply implemented agent (Koza, 1997).

Artificial neural networks (ANN) are learning computing systems inspired by the brain's neural networks. Inputs are processed in an ANN through a series of connected units called neurons. They are passed down connections weighted by their favourability for processing and transformation. Once an output is produced, the expected and actual results of the output are compared. The ANN then backpropagates through the relevant neurons, adjusting connection weights to either encourage or discourage their use based on the outcome. This adjusts the decision-making process of an ANN as priorities are shifted. This forms the learning component of the implementation (Dacombe, 2017).

ANNs present a viable option for the implementation of a Love Letter AI, though appear to have a weakness in a tournament setting. An ANN requires time and many games competing against a competent agent until it can learn to beat it. In a tournament, there will be many different types of agents, each likely with a different strategy and implementation. It would not have the chance to adapt before losing the tournament. This effectively defeats the purpose of an ANN as its ability to learn is its most significant advantage. It is possible to train the ANN before beginning the tournament, though this would require developing another agent competent enough to match potential opponents. This would take time, forcing sacrifices in other areas of this project for what is essentially just a competent agent. As such, an ANN should not be prioritised, especially considering how the simplicity and randomness of Love Letter presents little need to.

Monte Carlo Tree Search (MCTS) is a heuristic driven search algorithm that makes use of reinforcement learning when searching through a tree. It has been used in many instances of AI, including video game AI. MCTS explores the branches of a tree, analysing each route's effectiveness and updating the value of states within that route as the search deepens. There are four steps to the process; selection, expansion, simulation and update. This process essentially boils down to simulating gameplay scenarios played out by different moves and deriving their ultimate outcomes until a time or resource quota is met. The move in the current game state which has the most proportionately favourable outcomes in the future is returned for play. The more time and/or resources the MCTS can utilise, the more accurate and effective is it's decision-making. As such, MCTS is an any-time algorithm which will always have an answer to provide once it's time or resource allotment has been consumed (Ai and Games, 2018). This makes it quite competent in

video games since it can function in time constricted situations and the love-letter game is no exception.

Bayesian networks are a computational implementation of Bayesian reasoning. It is a probabilistic graphical model that represents a set of variables and their conditional dependencies through a directed acyclic graph. Algorithms can infer and learn with Bayesian networks as they are designed to find the probability of symptoms linking to causes and vice versa. In theory this can be used within the love letter AI implementation to allow for the ability to predict a player's choice after many game simulations (Soni, 2018). However, this does not seem useful for Love Letter, given that many players will play adaptively based on what cards they receive, only ever holding one card outside of their turn. What may seem like a player's pattern of play would likely just be them playing based on their circumstances, making identifications from Bayesian reasoning unreliable. It is more consistent to form predictions based on probabilities derived from unseen cards during that round, a calculation which does not require Bayesian reasoning.

Description/Rationale of Selected Technique

There are a variety of moves an agent can make with varying levels of viability and optimality. However, an agent is limited by the information it possesses, to the cards it holds, and, of course, to the realm of the rules. A large proportion of variability in Love Letter comes from its random nature. While players can obviously sway the game in their favour, this randomness demands turn-by-turn adaptability from them, rather than an elaborate plan. For our project, we decided to implement an MCTS agent for one submission, and a logic-based, reflex agent for the other. The MCTS agent looks at many different options, and the possibilities which stem from each of them. These possibilities are then explored and expanded upon until a set point or time when it returns the most proportionately successful move in leading to the desired outcome. The reactive agent analyses only the current state of the game, scanning for and playing moves of a pre-programmed priority in a human-like strategy. Each of these agents analyse the current state of the game, though one considers the future, while the other does not.

Effective agents will typically adapt to their environment, embracing the randomness, rather than try to force an elaborate plan. Learning algorithms may pick up on patterns of play that enemies would not consistently use, nor techniques it could, itself, utilise. This makes them less viable in a game with a high degree of randomness. MCTS does not need to learn, working based on the assumption that its rivals will play optimally. It will play conservatively like a minimax algorithm, opting for moves which present as little risk for as much reward as possible. The presence of many unknowns can prove rewarding for this conservative play, as MCTS will rarely be taken off guard.

Given that it is an any-time algorithm, it may easily stop its tree search when needed to fit within the restrictions of the assignment.

There are also downsides to using this implementation. MCTS may be reluctant to exploit opportune moments in play if it is overly concerned with the possible responses of opponents. Working under the assumption that opponents will always play optimally, as regarded by the MCTS, can present problems. The opponent may discover an avenue to victory distinct from the one that MCTS considers optimal, one which MCTS does not block as efficiently. Also, if the other agents also implement MCTS, the victor will be decided by the advantages afforded by randomness and turn of play, as each algorithm will perfectly counter the other. If our MCTS competes against another, deeper-searching algorithm, our MCTS may suffer from the horizon effect, making it almost strictly worse. Despite all this, MCTS is still a solid implementation choice given its independent strength and consistency.

The logic-based agent is our other submission, one which also does not learn, instead only reacting to the current state of the game and exploiting available information. It reacts with the highest priority move which it is programmed with based on human-like logic and strategy. Using information such as its knowledge of other players' hands and unseen (i.e. non-discarded) cards, the agent selects moves of certain advantage or a heightened probability of advantageousness. It does not create a tree, expand it, nor simulate the game state, making it quick and memory efficient. There is little to no risk of the program exceeding the preallotted time limit restriction. The agent does not necessarily follow an optimal route which an MCTS would expect of it, making it capable of circumventing the defensive play of an MCTS. While this unorthodox route can confuse MCTS, it is also hard to trace by a neural network, since there is no real pattern of play.

A learning agent may be able to decipher the order of prioritised moves performed by the logic agent, narrowing the possibilities of the second card held. In the given tournament environment, it is unlikely a learning agent would have time to devise a counter to this logic agent in particular. This is an especially valid assumption given that it is also competing against two other, likely differently implemented agents, which will do their best to hinder each other agent. As such, this logic agent could prove to be a bane to MCTS and a foreign opponent for learning algorithms, making it a simple, yet strong implementation.

Student No's: 22260157 & 22243849

Description of Validation Tests and Metrics

The first table below shows the performance of the reflex agent against three random agents playing 100,000 games with randomised decks. These games were simulated using the BotTester.java file. The process in using the BotTester.java file and reproducing these tests is described in the Agent22260157_ReadMe.txt file.

Name	Wins	Win Rate
LogicAgent	65928	65.928%
RandomAgent	34072	34.072%
RandomAgent		
RandomAgent		

Results were not produced for the MCTS agent as it was not fully functional by the submission due date. The instructions for testing should it have worked can be found in the Agent22243849_ReadMe.txt file.

Analysis of Agent Performance

The reflex agent performed significantly better than the random agents. From this, it can be inferred that strategy can be used to somewhat overcome the random nature of Love Letter. There is no state of the game unaccounted for with the wide spread of implemented heuristics. These heuristics are organised by human-determined priority of advantageousness, returning the highest priority move currently legal to the agent.

Different data structures were used for optimal interactivity with useful information. Data on unseen cards were stored in an ArrayList as it can be dynamically size and is easily iterated. This was used to exploit probabilities to better guess what card an opponent may hold. Highest frequency cards were prioritised for obvious reasons, while a higher value meant settling a tie, as players would typically hold onto higher value cards if they can. HashMaps were used to pair players with their cards if they were known for straightforward retrieval for certain guard guesses.

Tracking the previous moves of opponents is another potential heuristic which presents an area for improvement. Given the due date of the project, time was not dedicated towards implementing this with the given interface. Having an approximate 66%-win rate in a game with an average of 25% is a solid result. Interfering with the heuristics more could have taken progress to a point of diminishing returns.

Regrettably, the MCTS agent was not fully functional at the time of submission and thus, could not be properly tested for analysis. The agent was originally intended to be implemented as a traditional MCTS. However, the implementation was made to be simpler than perhaps it should have been. This led to the agent to perform a variety of simulations, to produce a series of results. It would then make a move based on these results rather than properly exploring game states. Analysing which move would most likely yield the desired outcome would ideally be done by selecting the most proportionately successful move by comparing desired end states to undesired end states.

There was also the issue in that MCTS would not be very effective for the love letter card game. The state of the game can vary greatly depending on the cards randomly dealt to players which leads to the search tree of an MCTS being possibly unrepresentative which in turn affects the quality of MCTS. Considering the effect of agent decisions on the state of the game through multiple simulations of a round to their outcomes would have made for a more functionally representative implementation of MCTS.

References

- Ai and Games. (2018, February 11). AI 101: Monte Carlo Tree Search [Video File]. Retrieved from https://www.youtube.com/watch?v=lhFXKNyA0QA
- Arzt, S. [Samuel Arzt] (2017, September 2). *Explained In A Minute: Neural Networks* [Video file] Retrieved from https://www.youtube.com/watch?v=rEDzUT3ymw4
- Dacombe, J. (2017). An introduction to Artificial Neural Networks (with example). Retrieved from https://medium.com/@jamesdacombe/an-introduction-to-artificial-neural-networks-with-example-ad459bb6941b
- deeplizard. (2017, November 22). Artificial Neural Networks explained [Video file]. Retrieved from https://www.youtube.com/watch?v=hfK_dvC-avg
- Koza, J. R. (1997). Genetic programming.
- Lin, C. S., & Ting, C. K. (2011, November). Emergent tactical formation using genetic algorithm in real-time strategy games. In 2011 International Conference on Technologies and Applications of Artificial Intelligence (pp. 325-330). IEEE.
- Omarov, T., Aslam, H., Brown, J. A., & Reading, E. (2018). Monte Carlo Tree Search for Love Letter.
- Soni, D. (2018). Introduction to Bayesian Networks. Retrieved from https://towardsdatascience.com/introduction-to-bayesian-networks-81031eeed94e