**CITS3001 Algorithms, Agents & Artificial Intelligence**

**Project – Love Letter**

Jaydeep Gajera – 22228003

Parthvi Sheladia – 22241757

**Literature Review**

Love Letter is a game of risk, deduction, and luck and can be played between 2–4 players. A player’s goal is to get their love letter to the Princess. Two ways through which the letter can be delivered are: giving the letter to an intermediary or directly to the princess. From a deck of sixteen cards, each player starts with only one card in hand; with each card representing an intermediary. Each intermediary corresponds to an action that can affect all players in the game in different ways. Actions consists of swapping cards with another player to eliminating a player completely from the round. The game comprises of multiple rounds and each player draws a card on their turn leading to make a choice between two cards depending on what action increases their chance of winning the round and essentially the game. Powerful cards lead to early gains, but make you a target (Brown, 2018). The game is full of uncertainty and no player have a clue of other player’s cards, hence can be referred to as an imperfect information game.

Monte Carlo Tree Search (MCTS) is a decision-making algorithm that helps deal with such games. It is probabilistic and heuristic driven search which works even better when combined with a useful implementation like the classic tree search implementation. It uses the idea of building a game tree with each node representing a game state that potentially helps with making optimal decisions. The algorithm periodically exploits the best action and strategy found while also continuing to explore other alternative to see if they could replace the current best. The nodes as previously mentioned are a result of a number of simulations. The whole process can be broken down into four distinct steps: selection, expansion, simulation and backpropagation. The MCTS algorithm relies on two fundamental concepts: The expected reward of an action can be estimated doing many random simulations. These rewards can be used to adjust the search toward a best-first strategy (*Roy, 2019*).

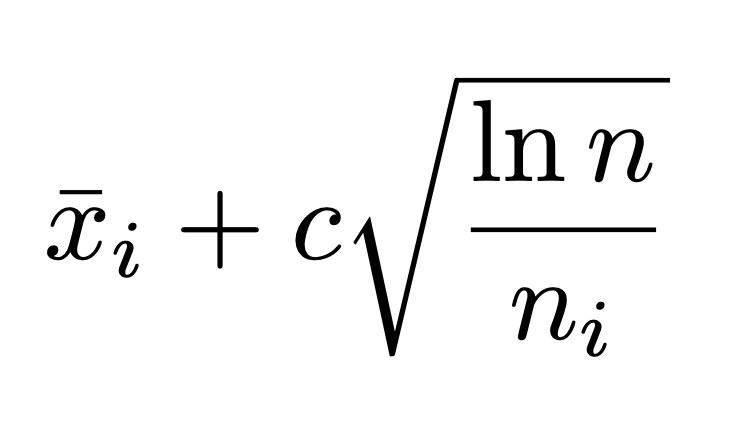
MCTS with determinization is able to handle imperfect information games much better than the basic MCTS algorithm. With determinization, probabilities of different possible states based on the information about the current game state of the player (also known as the player’s information set) is calculated. Then next steps are the same as the basic MCTS. the best moves in each state are selected assuming that players can see the entire state for the rest of the game, and it therefore it doesn’t account well for the effects of information being hidden (*GitHub, 2017*). And as cited in the Monte Carlo Tree Search for Love Letter paper, Frank and Basin (*1998*) discovered two problems of determinization, which are strategy fusion and non-locality. Strategy fusion is where the agent assumes that different decision can be made for different determinizations on the same information set and Non-locality is where certain states are considered even though they are very unlikely to be chosen. These don’t limit the results of the algorithm. The algorithm will still perform well compared to a basic MCTS.

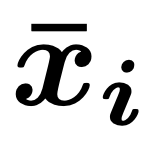
Perfect Information Monte Carlo Search with the Determinized Upper Confidence Bounds Applied to Trees is based on the idea to build a tree in an incremental and asymmetric manner by doing many random simulated games. Mańdziuk (2018) highlights the main purpose of this algorithm is to find a balance between the exploration of the less frequently simulated nodes and the exploitation of the already chosen best ones. To explain the whole process in terms of the four steps of MCTS, once a node is selected, expansion occurs by choosing an available move and adding it to the tree. Once that is done, by running a simulation on the child node and backpropagating, a result is achieved. A simulation is based on a default policy in the algorithm and the result is calculated using the UCB formula. By continuously repeating these steps, the result values are updated. This algorithm is beneficial as no prior knowledge is required for the agent (knowledge of game rules is enough). Along with this, a determinization is a conversion of a stochastic game with imperfect information to a deterministic game with perfect information, in which the hidden information and the outcomes of all future chance events are fixed and known (Lanzi and Palma, 2014).

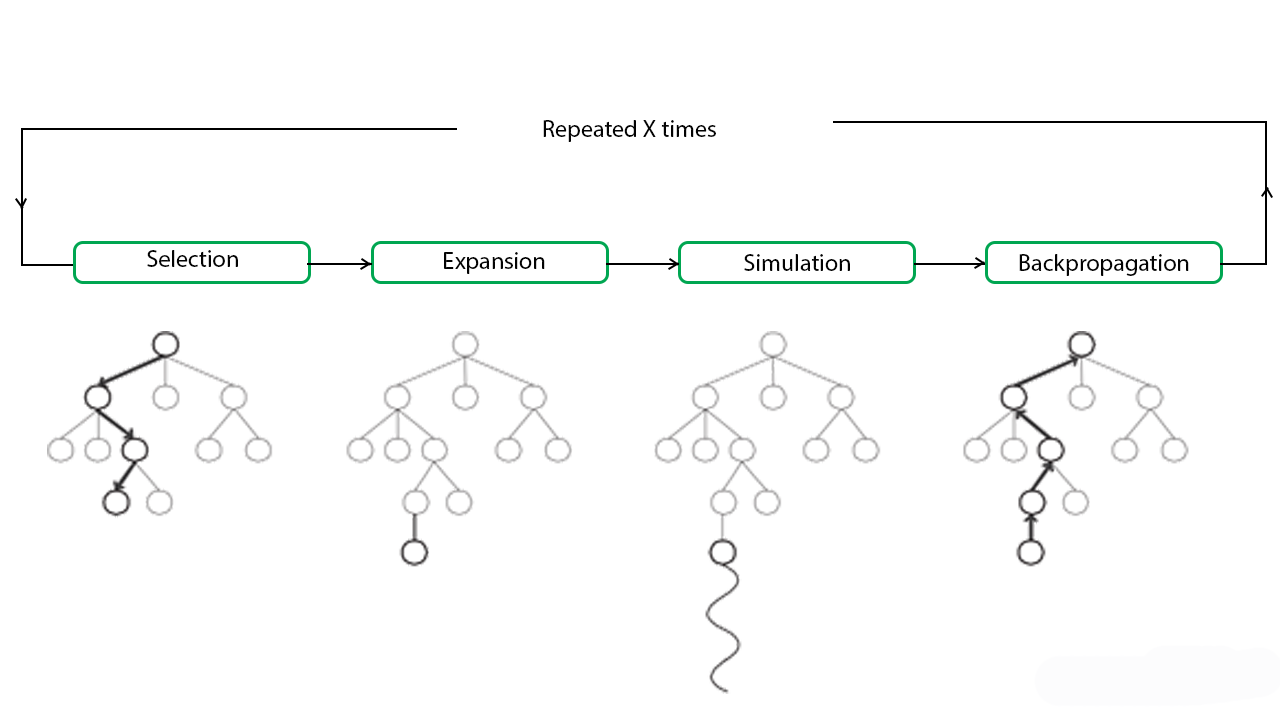
Another useful search-based algorithm that can deal with imperfect information games is an extension of MCTS called the Information Set MCTS (ISMCTS) (Mańdziuk, 2018). In this algorithm, rather than the node being represented by the state of the game/player, it is now represented by information sets. The information set is from the root player’s point of view and corresponds to moves player by the corresponding player.

**Selected Techniques**  
  
Utilizing epistemic rationale, we could make a lot of guidelines for how to derive knowledge and awareness about the game state. But, Love Letter, is very unpredictable as a player can have any of the 16 cards. Hence, it is very hard to distinguish between good and bad decisions (moves). For example, if the player decides to play the BARON card and no other information is known, then there may be high chances of them having a lower valued card, which would be very costly. It is vital for the player to be aware of the game state and other cards.  
  
Since Love Letter is very unpredictable game, it would be more suitable for our technique to work with probabilities and evaluation functions rather than simply making good or bad decisions. Using probabilities (or evaluation functions) won’t win us the round every time, but it will give us a good understanding of the game and help us recognise the good decisions. Therefore, keeping this in mind, a **knowledge-based agent** algorithm was implemented. It executes moves in line with the game rules and also inspects for the moves that can lead to an immediate win or lose. It tries to avoid the moves leading to an immediate lose. Also, when the information about the game state available is vague, then it calculates the probabilities of a player having different cards in their hands and chooses the player and/or card with the highest chances. The chances are calculated based on the played cards, non-played cards and the cards in the agent’s hand. The agent also plays the lower valued card out of the two, in order to keep the higher one, which can possibly result in agent winning the round. The knowledge-based agent is not time or memory costly as most of the time it implements if-else conditions to make decisions, but has a lower win rates as it may not be able to guess the correct card in opponent’s hand.  
  
Secondly, a more reliable and advantageous approach was implemented. This implementation uses **Monte Carlo Tree Search (MCTS) algorithm**. We chose to work with this algorithm as it is a feasible algorithm with no prior requirements or brute forcing of every possible action.

There are four main steps in this algorithm: selection, expansion, simulation and backpropagation.  
In the **selection** process, the algorithm traverses the current tree from the root node by selecting child nodes using a specific strategy. The strategy uses an evaluation function, Upper Confidence Bound (UCB), to optimally select nodes with the highest estimated value returned from the evaluation function. UCB formula is used to traverse the tree as it balances the exploration-exploitation trade-off (*Roy, 2019*). The nodes are selected based on the following UCB formula:



where  is empirical mean of node , which is calculated as ratio of number of simulations won passed through the node to number of all simulation passed through the node (*Brown, 2018).*  is the number of times node  was selected from its parent according to Upper Confidence Bound (UCB) formula. The constant, , controls the rate of exploitation and exploration when traversing the tree nodes and lastly, , is the number of times the parent node was visited (*Brown, 2018*). In the **expansion** process, all new unvisited child nodes are added to the node that was selected in the previous selection process. Then, in the **simulation** process, game is played by choosing moves and strategies until the round is finished, during this step no new nodes are added to the tree. Lastly, in the **backpropagation** process, the nodes are visited from the leaf (end) node to the root node, in order, and a counter is incremented for each node in the path. After all of the above steps are done, the algorithm chooses the child node of root that has the highest counter. Though this approach has higher win rates, it requires a lot more time and memory resources than knowledge-based agent. As the tree grows rapidly after the first few iterations, the memory becomes an issue. Also, this algorithm requires a large number of iterations in order to successfully find the best path. Therefore, the time also becomes an issue.  
  
The four main steps of the MCTS algorithm can be visualised in the picture shown below.

 Picture 1: Monte Carlo Tree Search steps (Roy, 2019)

**Implementation**  
**Knowledge-based Agent –** Firstly for this agent, we need to find all the possible win conditions. Therefore, if we know any card of any players through playing PRIEST or KING, then we need to save that player and also the card that they hold. We also need to store how many cards are left in the deck and the cards that the players may hold. This is done by getting all the discards and taking away them away from the total deck, we can also further prune the cards out by subtracting the cards that we already have in our hand and the freshly drawn card from the deck. At this point, the agent will have all the known cards of any players and also all the unplayed cards including the ones that the opponents currently hold.  
  
The next step is to play the compulsory card that we may have to play, for instance, if we have PRINCE or KING and also the COUNTESS, we must discard the COUNTESS.  
  
Then, we compare the two cards that we have and play the lowest valued card. Hence, we first check for GUARD, if we have it, then we must play it. With this card, if we know any opponents’ cards, we can target them, which will eliminate them; if we do not know any opponents’ cards, then we choose the player with the highest score and also choose the greatest number of non-GUARD cards in unplayed cards.  
  
Then, we check for the PRIEST card and target the player with the highest score.  
  
After that, we check for the BARON card and see if we know any opponents’ cards. If we do and they are lower valued, we target them to eliminate them from the round. Otherwise, if they are higher valued, then must not choose them at any cost as it will eliminate us. Therefore, in this case, we must play our other card.  
  
Next, we check for the HANDMAID card and play it if we have it.

Then, if we have the PRINCE card and we know that our opponent holds the PRINCESS card, we must target them to eliminate them. Otherwise, if our second card is PRINCESS, then we must not target ourselves, otherwise we will be eliminated.

Finally, we check for the KING and the COUNTESS card and play them.  
  
With this agent, the probability is calculated as follows:

For instance, if we are calculating the probability for the card PRIEST, no PRIEST cards are discarded by any player and there are total of 8 unplayed cards left. Then the probability equals to . This is calculated for all the cards, the card with the highest probability is chosen.

**Monte Carlo Tree Search (MCTS) algorithm –** This agent performs the four main steps as described above to choose the best possible action for the current state. It uses Tree and Node structure to save all the nodes and their correlation with the other nodes. Also, a custom NodeState class is defined that implements all of the methods defined in the original State class. A NodeState is attached to each node to keep track of the current game state.  
  
A time of 900 milliseconds is given to explore as many nodes as possible and return the best node. Inside this time loop, all of the four steps are performed.  
  
Firstly, the selection process is performed, which chooses the node according to UCB formula as mentioned above. The child node that has the highest score. The constant, , which controls the rate of exploitation and exploration of nodes is chosen to be 0.7, which has been proven to be effective via many controlled experiments before (*Brown, 2018*).

Secondly, in the expansion process, both of the cards are examined and played to give the updated child nodes. With the cards requiring a target player (and a card guess), all possible actions are considered and added as the child nodes. These actions only include the legal actions meaning possible actions pruned down in order to make the algorithm more efficient. All of these actions are performed for each node and a new game state is saved for each node.

Now that all possible nodes are explored, the simulation process is performed, where the algorithm selects random unvisited nodes and plays the game. The nodes are explored until the round is finished. Hence, the tree has maximum depth of 12 as there will be four players holding the cards and 12 cards in the new deck, which will need to be simulated. During this process, no new nodes are added to the tree.  
  
Lastly, in the backpropagation process, starting from a leaf node to the root node, a counter is incremented for each node that is in the path. In our implementation, we are adding a score of 10 each time we visit a node during this process.  
  
Then, we choose the child node of the root that has the highest score. This will definitely be the best action out of all possible actions available, that is if the algorithm is able to explore all the nodes in the time and space constraint. The main advantage of this algorithm is that any of the intermediate steps can be saved for later propagation, this saves a lot of space. It also allows for asymmetric expansion of the tree.

We tried to implement this agent and were very close to get it working, but due to not having enough time, it is incomplete and running it results in error. Parts of the implementation of this agent is sourced from an implementation of MCTS of Tic-Tac-Toe game (*Baeldung, 2019*).

**Validation**

Firstly, to test the performance of the RandomAgent provided, we ran a tournament between the RandomAgent and three Knowledge-based Agents. Then, to test the performance of our Knowledge-based Agent, we ran a tournament between three RandomAgents and our knowledge-based agent.

Both of the tournaments included 100,000 games of Love Letter with each agent being tested playing from different position in each tournament.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Position 1 Win Rate (%)** | **Position 2 Win Rate (%)** | **Position 3 Win Rate (%)** | **Position 4 Win Rate (%)** |
| RandomAgent | 3.780 | 6.211 | 15.706 | 10.723 |
| Knowledge-based Agent | 50.797 | 50.658 | 57.269 | 51.356 |

The Knowledge-based Agent performed exceptionally better than the RandomAgent. This proves that using a well-defined algorithm increases the chances of an agent’s success. Also, it can be noticed in the above results table that for some reason, the win rates of both agents for position 3 are quite high than the other positions. For RandomAgent, the win rate for position 3 increased by 315.50% than the lowest win rate. For Knowledge-based Agent, the win rate for position 3 increased by 13.05% than its lowest win rate. The reason for this increased win rate was not established. The MCTS Agent could not be tested due to it producing errors. Therefore, it is not included in the results above.  
  
If this experiment / project was to be done in future, some improvements could be made to the Knowledge-based Agent such as if an opponent has played BARON on another opponent, then we know that the opponent, that is not eliminated, has a card valued higher than the recently discarded card by the eliminated agent. This information will be helpful to us in future to possibly eliminate them. When tested our Knowledge-based Agent multiple times, it was found that our agent was eliminated from the round due to three main reasons: if the opponent played a correct card with the correct guess of the card, if we had no choice to eliminate ourselves (i.e. if only two players left, we have a BARON and PRINCESS and we know that opponent has a higher valued card than us; hence in this situation playing either card will eliminate us) or we chose a wrong target player for the BARON card. Hence, in future, some more strategies will be applied when playing the BARON card as mentioned before.  
  
Also, if our MCTS Agent was to work, then we would experiment with different values of constant,  (which controls the rate of exploitation and exploration), to make the agent more efficient. In future, to improve the MCTS Agent, the strategies of the Knowledge-based Agent could be applied to prune the tree down to a smaller size. This will definitely make the agent space efficient and possibly time efficient as well because it will take less time when backpropagating.

**References**

* Brown, J. (2018). *Monte Carlo Tree Search for Love Letter*. [online] ResearchGate. Available at: <https://www.researchgate.net/publication/327679828_Monte_Carlo_Tree_Search_for_Love_Letter> [Accessed 16 October 2019].
* GitHub. (2018). *eugenp/tutorials*. [online] Available at: <https://github.com/eugenp/tutorials/tree/master/algorithms-miscellaneous-1/src/main/java/com/baeldung/algorithms/mcts/montecarlo> [Accessed 18 October 2019].
* GitHub. (2019). *eugenp/tutorials*. [online] Available at: <https://github.com/eugenp/tutorials/tree/master/algorithms-miscellaneous-1/src/main/java/com/baeldung/algorithms/mcts/tree> [Accessed 18 October 2019].
* Lanzi, P. and Palma, S. (2014). *Monte Carlo Tree Search algorithms applied to the card game Scopone*. [ebook] The University of Western Australia. Available at: <http://teaching.csse.uwa.edu.au/units/CITS3001/project/2017/paper1.pdf> [Accessed 15 October 2019].
* Mańdziuk, J. (2018). *MCTS/UCT in solving real-life problems*. [online] ResearchGate. Available at: <https://www.researchgate.net/publication/320003615_MCTSUCT_in_solving_real-life_problems> [Accessed 12 October 2019].
* Monte Carlo Tree Search. (2016). [ebook] Swarthmore College. Available at: <https://www.cs.swarthmore.edu/~bryce/cs63/s16/slides/2-15_MCTS.pdf> [Accessed 17 October 2019].
* Baeldung. (2019). *Monte Carlo Tree Search for Tic-Tac-Toe Game | Baeldung*. [online] Available at: <https://www.baeldung.com/java-monte-carlo-tree-search> [Accessed 18 October 2019].
* Roy, R. (2019). *ML | Monte Carlo Tree Search (MCTS) - GeeksforGeeks*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/> [Accessed 16 Oct. 2019].