Yiyang Gao (21263128 )
Tzi Siong Leong (20753794)

Resistance Project Report

## Introduction

"The Resistance" is a board game of five to ten players with concealed identities where the nominated subset of players are sent on missions. Depending on the player's identities, they will either sabotage the mission or succeed the mission. Due to the incomplete information nature of the game, it is a good challenge to implement an AI agent for it. Several techniques have been explored such as the Bayes Rule, Monte Carlo Tree Search and several expert rules to development a sophisticated agent. It was found that the best techniques for the agents were the ones that implemented Bayes Rule and Monte Carlo Search.

## Literature review

## Monte carlo search

Monte Carlo Tree Search (MCTS) consists applying the Monte Carlo Search (MCS) heuristics with a tree searching algorithm. The MCS method is characterized by repeated random sampling of the search space to obtain the expected value of an average (Taylor 2014). For the Monte Carlo Tree Search, the search tree is incomplete so the MCS method is used to expand the tree by randomly choosing unvisited decisions/nodes to expand from (Taylor 2014). Indeed, the MCTS process consists of the selection, expansion, simulation and back propagation phases to get try to achieve the expected utility of the selected state (Taylor 2014). The selection phase is characterized by traversing through the tree randomly picking each node until it reaches an unvisited node (Taylor 2014). This selected node is expanded by adding the child nodes associated with the selected node. The third step requires playout simulations from each of the child nodes as it goes through to the terminal node while choosing random decisions (Taylor 2014). The simulations should be able to obtain the expected value of the selected node's children utility scores as well as updating or backpropagating the utilities of the nodes above the selected node up to the root node of the tree (Taylor 2014).

These processes are what make MCTS great for big games with large game trees like Go in which the partial game tree can be developed with a bias for promising states instead of search the whole game tree (Whitehouse 2014). MCTS is also described as an anytime algorithm since it is an iterative algorithm that can update the states' values following every iteration of the algorithm (Whitehouse 2014). This feature is beneficial for game simulations where it can simulate as many games as it can to obtain an expected value of the results.

## Bayes Rule

Yiyang Gao (21263128 )
Tzi Siong Leong (20753794)

Resistance Project Report

A good agent will gather as much information from surrounding as possible and perform actions based on these information. However, for a game with hidden information such as the Resistance, information available alone is not enough to make decisions, as the true state of the game simply cannot be determined. Therefore, a good agent should consider multiple contingencies and their likelihood in order to make good decisions. Bayesian Probability has long been studied in the area of artificial intelligence, when reasoning under uncertainty is required (Poole & Mackworth 2010). In Bayesian probability, probability is regarded as a belief about a certain event happening in the future or a statement being true, based on the observed history (Poole & Mackworth 2010).

The most important formula within Bayesian probability is (Poole & Mackworth 2010):

$$P(h \mid e) = \frac{P(e|h) * P(h)}{P(e)}$$

h is the proposition of some kind of statement, while e represent the history that have been observed by the agent (Poole & Mackworth 2010).
$P(h|e)$ is the posterior probability of proposition h, namely, the belief agent have in proposition h given all the history e that it has observed (Poole & Mackworth 2010).

In order to evaluate the belief in proposition h, the agent must know the likelihood of proposition h, which is represented as $P(e|h)$, that is to say the agent need to know the probability of history e happening assuming proposition h is true (Poole & Mackworth 2010). $P(h)$ is also called the prior probability, which is the probability of proposition h being true when no history have been observed (Poole & Mackworth 2010).

The denominator of Bayesian equation is P(e), which is the probability of history e happening, which is the summation of probability of all proposition that could possibility lead up to the history e observed (Poole & Mackworth 2010).

$$P(e) = P(e|h_1) * P(h_1) + P(e|h_2) * P(h_2) + P(e|h_3) * P(h_3) + \text{******} + P(e|h_n) * P(h_n)$$

Where $h_1, h_2, h_n$ are mutually exclusive propositions that can lead to the observed history e.

## Expert Rules

The Expert.java bot was influenced by some of the Opponent Modelling (OM) bots as seen in (Taylor 2014). Opponent modelling is the method of building statistical and probabilistic models of your opponent to predict their moves and strategies (Taylor 2014). The opponent modelling bots were seen as some of the more successful expert bots in the competition as stated by Taylor (2014) so it is in the

intersest of the Expert.java to be implemented as such to be competitive (Taylor 2014). Although the opponent modelling competition bots specified by Taylor (2014) were able to model players after each game, the Expert.java was not implemented with that feature in mind, just some of the heuristics mentioned from some of the expert bots. For example, the Expert.java bot is loosely based on the ScepticBot as mentioned by Taylor (2014) where it keeps track of the frequencies of certain actions from the opponents and increases their suspicion scores.

Regardless of the bots' implementation of AI, they follow common expert rules that are very common with the real life game of resistance (Taylor 2014). If some bots do not follow these common rules, they would be considered highly suspicious by other bots (Taylor 2014). Some of these rules consist of the nomination leader always elect itself and not rejecting the first voting attempt for the first mission (Taylor 2014). Although an automatic spy victory after the fifth voting attempt for a mission is not implemented in the Game.java, some resistance bots do follow the expert rules where they would accept the fifth voting attempt regardless of any other factors because it would be an automatic win for the spies if they do not accept the mission (Taylor 2014). Some other rules consist of sabotaging the mission if it is the leader, avoiding double sabotages and automatic sabotages in the third mission if two missions have already been seen as sabotaged (Taylor 2014). Overall, these rules are very logical for any bot to follow to prevent the bots themselves from exposing their identities to the other bots and to ultimately win the game.

## Selected Technique

### Bayes Rule

Bayesian probability is selected for the completion of this project due to the fact that the Resistance board game is a non-deterministic game with incomplete information game where the true state of the game can only be inferred, never be truly determined. The behaviour of the other players would influence the outcome of the game, introducing more uncertainty. Therefore, Bayesian probability's ability to reason under uncertainty can be utilised in the context of the game, Resistance. As the game progress, more and more observations are obtained, this gives the agent playing this game a perfect opportunity to make use of this new observation to make an inference of the true state of the game. Subsequent decisions can be made based up this inference.

### Monte Carlo

In the case of the Resistance board game, each decisions or moves would be repeatedly randomly sampled to get a true value of its utility of the decision. The larger the amount of repeated samplings there are, the more accurate the utility of

the decision/state gets by approaching to its expected value as the number of experiments reaches infinity (Taylor 2014). In the StatAgent.java case, Monte Carlo Search was applied when the agent is a spy and chooses its nominations for his turn in the game. The different possibilities of nominations are evaluated to test each nomination for its viability to win the game through the end. Each nomination is simulated with the current state of the game all the way to the end of the game to obtain its utility value. The utility value of each nomination/node is the number of more failures obtained from the initial number of failures that the state originally had. This utility value gets added up by each simulation for that nomination and it is an indicator of how much more effective is one nomination compared to others. The nomination that has the best utility from the list of nominations gets chosen as a move for the StatAgent.java spy player. The reason why Monte Carlo Search was used instead of the MCTS was because the nomination decision only need to go to a depth of one level in a game tree so it is not so much as a MCTS but rather a MCS. Furthermore, MCS was used for picking the nomination because it was found that all the nomination choices should be simulated in the game and also simulated for a significant amount to find a more accurate expected value of its utility

One of the biggest problems of using the MCS was the time it takes to simulate the games for each nomination. Although the number of simulations is constant for each nomination choice, the number of nominations that is generated varies with the number of players: that is, more nomination possibilities are generated the greater number of players. The increasing amount of simulations needed would therefore increase the runtime needed to run MCS so the number of simulations need to be set at an optimum level try to keep the nomination function call under second. On the other hand, the number of simulations can not  be a small due to the fact of the Law of Large Numbers states that the utility reaches its expected value as the number of simulations increases (Taylor 2014). With these constraints in mind, we believe that an optimal number of simulations can be reached and the benefits of the MCS overweighs the disadvantage of a longer runtime.

## **Expert Rules**

Expert Rules were seen as quite effective in creating a formidable Resistance playing agent. Many different rules can be used to produce an Expert Rules bot, the Expert.java focuses on suspicion tracking as seen in OM bots. Although the Expert.java has a basic suspicion score tracking for the resistance player in Expert.java, it may not be as thorough as the Bayes Rule model implemented in the StatAgent.java. Likewise, conditional betrayal statements in Expert.java may not be as extensive as the Monte Carlo Search method used by the spy player in StatAgent.java. So therefore, the performance difference is of interest between the bots to observe how impactful the sophisticated models can be in winning more

games. Thus, the Expert.java was implemented as a basic expert rules bot to compare how well the sophisticated bots perform compared to a non-sophiscated expert rules bot.

## Implementation

### StatAgent.java (Resistance Player)

StatAgent employs Bayesian model when playing as a resistance member. StatAgent utilize the Bayesian formula described in the literature review section as the following:

1. Each possible composition of spies team is treated as a proposition h. The prior probability of each proposition h is just $\frac{1}{C_n^m}$, where m is the number of spies within the game, n is the number of player minus one, (minus one is because player itself is sure that itself cannot be on a spy team). $C_n^m$ is the number of picking m number of spies out of n number of people. $\frac{1}{C_n^m}$ is just the probability of each proposition being true, since no history has been observed yet.
2. Every time there is a new observation, namely, a round of game has been completed, and the players on the mission and total number of sabotages known, Bayesian formula will be utilise to calculate the probability of each proposition given the newly observed history. In order to model this, the likelihood of proposition h, $P(e|h)$, must be determined. StatAgent implements this by keeping a sabotage probability variable within the class, it represents the probability of each spy sabotages in the current round.
   For each new observation, $P(e|h)$ can be calculated as
   $$_S^F C * a^F * (1-a)^{S-F} * P(e_{old}|h)$$
   Where F is the number of sabotages from the newly observed game, S is the number of spies on the team according to each proposition, a is the probability of each spy sabotage within the round. $P(e_{old}|h)$ is the likelihood of proposition h for the history before the newest observation.
3. $P(e)$ can be obtained as described in the literature review section, it is just the summation of likelihood of each proposition.

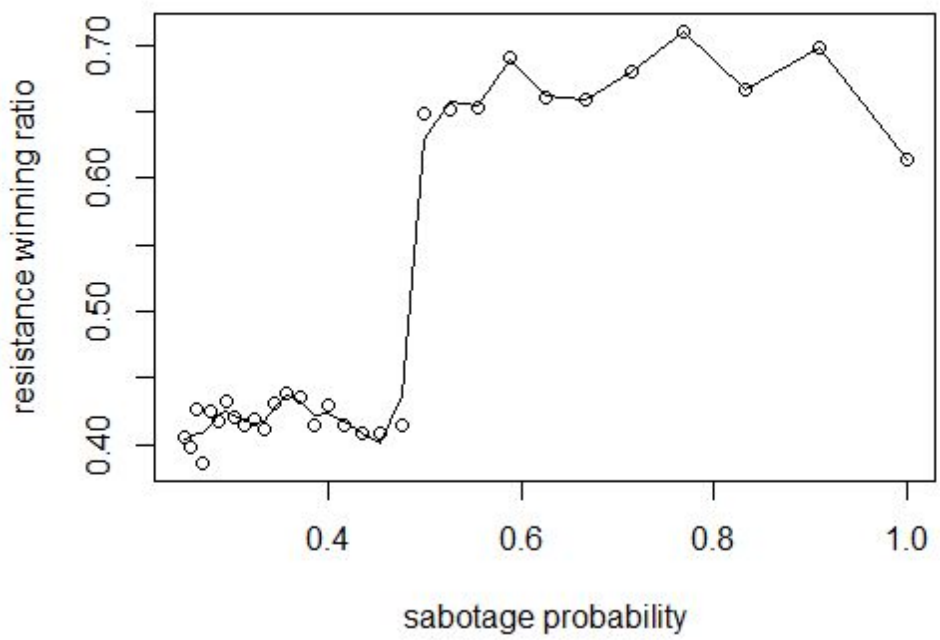
With all the key components calculated, the agent can now determine the probability of each proposition being true given all the observed history. Using this probability table of each potential set of spies, the agent will then build a suspicion table. This table is obtained by adding the probability of each proposition being true to each

players within the proposition. Then, the players will be ranked. The agent will only nominate the least suspicious players according to this table, as well as reject the proposed team if it contains any suspicious players according to the table. For example, for a five players game, the StatAgent will nominate the least three suspicious players according to the table, and reject a three players proposed team if the team contains any top two suspicious players according to the table.

The variable sabotage probability StatAgent keeps is also a function of the number of failed round the game have seen. The logic behind this is the fact that as the number of failed games increases, the spies are getting more and more closer to the game, so spies will be more prone towards sabotaging to end the game.

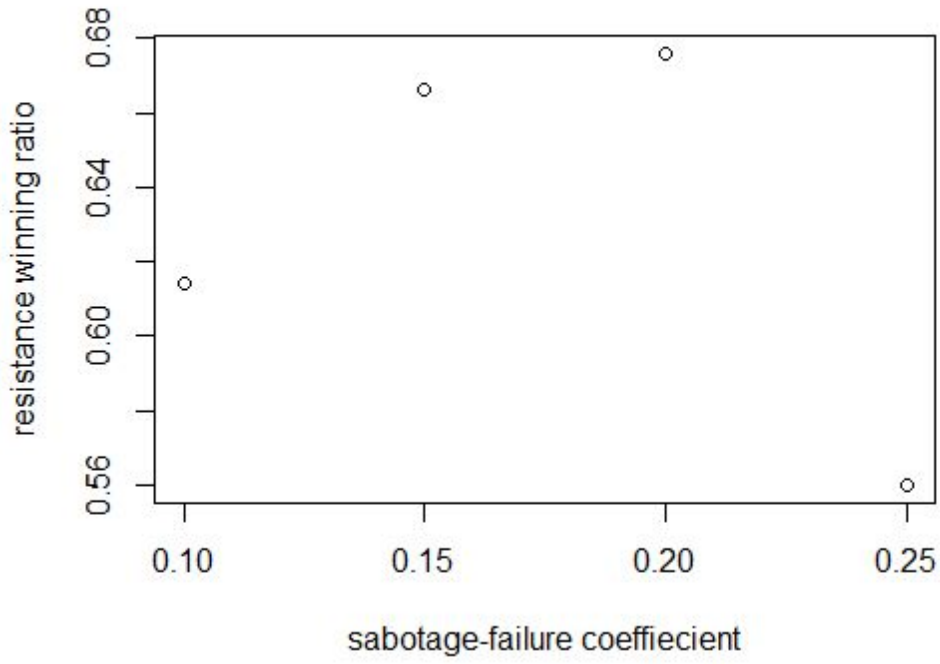
The proper value of the sabotage probability needs to found in order to optimize the performance. Therefore, many trials were conducted to obtain this key parameter. StatAgent was setup to play against the Expert Agent, and the win ratio of StatAgent as resistance member will be recorded with different value for sabotage probability for Bayesian model. The coefficient between sabotage probability with the number of failed rounds is also found in this way.

Below is the graph obtained by running 1000 games with StatAgent being the resistance and Expert Agent being the spy under different sabotage probability value.



As shown in the graph, when the sabotage probability is below 0.5, the win ratio for resistance is significantly lower than when the sabotage probability is above 0.5. When the sabotage probability is 1, the win ratio start to decline again. This is anticipated as when the sabotage probability is 1, the agent is assuming all spy sabotage all the time in every single game. This is clearly not the truth, so the suspicion ranking for all players will not reflect the true state of the game. When the sabotaging probability is too low, the agent is become too trusting, resulting unreliable suspicion ranking as well. Therefore, it is determined that sabotage probability at 0.6 is an appropriate choice.

The following graph is obtained when trying to determine the sabotage failure coefficient. A similar process is followed as above, with StatAgent playing as resistance all the time versus Expert Agent.




As illustrated in the graph, a sabotage failure coefficient of 0.2 is found out to be optimum. The actual probability for spy to sabotage in a given round is therefore calculated as

$$P = 0.5 + 0.2 * \text{number of failed round}$$

It can be observed that when the coefficient is 0, meaning when the relationship between sabotage probability and failed round is not considered, the win ratio is lower. This can be explained by considering the scenario where in a 5 player

game, there have been 2 failed round already. A logical spy agent would sabotage all the time if on a mission since spy is only one more fail away from game. With the sabotage probability increased, if the following round does not have any sabotages, then the chance of spies not being among this team is very high. Sabotage-failure coefficient take advantage of this mentality, therefore StatAgent with sabotage failure coefficient has higher win ratio.

**StatAgent.java (Spy Player)**

The spy player is partly implemented with the Monte Carlo Search method and expert rules. The expert rules were implemented for the functions do_Betray() and get_ProposedMission(). They consists of conditionals in which the spy player finds itself in to make a move during the game. The do_Nominate() function, when the agent is a spy, uses Monte Carlo Search method to search for a viable nomination for the game. It is an anytime algorithm in which it plays as many simulations as it can under the time limit for each nomination to get a better expected utility score. The function stops anytime when the time limit is reached and the results can still be interpreted, however, it would be best for the algorithm to run as many simulations as it can to generate a more truer expected utility value. Inside the do_Nominate() function, it calls the MonteCarloSearch class where it is responsible for finding the best nomination with the best utility score from the simulated games. It creates simulated games by calling the SimGame class with given game state conditions of the current game. The SimGame object is then added with simulated agent players that are also given current game state conditions as well. Namely, the ResistanceSimulationAgent and the Expert classes. The Expert bots play as spies and the ResistanceSimulationAgent bots play as resistance players. The MonteCarloSearch class is passed the list of nomination possibilities for the Expert spy to nominate from. Essentially, the Expert spy bots acts as a proxy for the StatAgent player to play the simulated games so that the StatAgent player would make its decisions based on how the Expert bot plays as a spy.

After all the players and state parameters have been added, each simulated game is played through to the end, it returns the number of more failures that have occurred in that simulated game compared to the initial number of failures that the game initially have. This score gets added up from all simulated games for each simulated nomination and it represents the utility score for that nomination. After all the expected utility scores have been obtained from each nomination possibilities, the nomination that has the highest utility score gets chosen as the decision for the StatAgent player.

Yiyang Gao (21263128 )
Tzi Siong Leong (20753794)

Resistance Project Report

## Expert.java

The Expert.java was implemented to be loosely based on the OM bot, Sceptic, which keeps track of suspicious behaviors and increases the suspicion scores of the opponents who exhibit the behaviors. The Expert.java was implemented as a basic expert rules bot to be used to compare against the StatAgent.java bot. The suspicion scores are kept as averages as AveragedValues objects in each player's PlayersSuspicion objects. These averaged suspicion values are used as an indicator of how suspicious an opponent can be for the resistance player. The resistance player can then use the suspicion score to vote for missions and to nominate players for missions. Depending on the action, a suspicion score valued between 0 and 1 is added to initial suspicion score and an average is taken again. This means that the suspicion score can fluctuate depending on what actions the opponent does.

The Expert.java spy player mostly consists of conditional expert rules such as to sabotage when it can and avoiding multiple sabotages in a mission. Since the spy has perfect information, it has the privilege to choose its own nomination with the ability to try its best to cover up its identity while sabotaging a mission. Overall, the common expert rules implemented in Expert.java try to mimic real life conditionals as closely as possible.

## Validation

In order to validate the performance of the two agents, StatAgent and Expert Agent, multiple trials are run with each agent taking on different roles and facing different opponent.

1. First, Expert Agent is tested by letting it playing against itself in 1000 games. The winning ratio for resistance and spy is found to be 0.43: 0.57.
2. Then, StatAgent is tested by letting it playing as resistance facing Expert Agent playing as spy. The winning ratio for resistance and spy is found to be 0.61: 0.39. This result shows that StatAgent's gameplay utilizing Bayesian model is stronger than Expert Agent. This is expected as StatAgent's Bayesian model is assumed to be better making decisions under uncertainty.
3. Next, StatAgent's gameplay as spy is tested by first letting is playing 1000 games against itself. StatAgent's spy gameplay utilizes Monte Carlo search, who also use StatAgent's Bayesian model to simulate the game as resistance player. Therefore, StatAgent's spy gameplay is expected to be particularly strong against its resistance gameplay, as the Monte Carlo search use Bayesian model to simulate the game and always pick the move most in favour to spy. The actual winning ratio for resistance and spy is found to be a surprising 0.92: 0.08, which confirms the expectation of StatAgent's spy gameplay being particularly good against Bayesian model.

4. Finally, StatAgent is tested by playing as spy against Expert Agent playing as resistance. This time, the actual winning ratio for resistance and spy is found to be 0.62: 0.38, much lower than its performance against itself. This is another proof that Monte Carlo Search technique exploits the weakness of Bayesian model particularly well.

**Bibliography**

Poole, D. L., & Mackworth, A. K. (2010). *Artificial Intelligence: Foundations of Computational Agents*: Cambridge University Press.

Taylor, D. P. (2014). Investigating Approaches to AI for Trust-based, Multi-agent Board Games with Imperfect Information; With Don Eskridge's``The Resistance''. *Discovery, Invention & Application*(1).

Whitehouse, D. (2014). *Monte Carlo Tree Search for games with Hidden Information and Uncertainty.* University of York.