The Resistance Game New Agent Implementation

Yuxi Zhou 2004457

yz1804488@essex.ac.uk

# Abstract

Artificial intelligence is now increasingly changing our life with a unbelievable speed, automotive, manufacturing, gaming as so on. In this project, we are working on building a new agent with artificial intelligence techniques such as decision trees and neural network, who should be capable of playing the Resistance game smartly by recognizing spies with a pre-trained neural network model.

In order to implement this, we first created a new bot that inherit with the basic functionality from class Bot. And we updated some basic logic for our bot to make relatively wise choices while selecting team and voting for missions. As we created a bot who can generate player’s logs throughout the game, we made fully use of these logs to train some Keras neural network models as a classifier to help our bot to recognize spies and performed better than ever before, by reaching 83.7% wining rate as a spy which is higher than all the other players.

# Introduction

## **1.1 The Resistance Game**

The Resistance[[1]](#footnote-1) is a social deduction game with secret identities. The game's premise involves a war between government and resistance groups, and players are either members of the resistance attempting to overthrow a malignant government, or spies trying to thwart the Resistance. The structure of the Resistance game is a party game that a small group of people who knows each other fighting against a large group of people. This game is different from other similar games because there is no elimination, hence it provide more information to the player when making corresponding decision.

## **1.2 Game Rules**

At the start of the game, one third of the set of players (rounded up) is randomly and secretly chosen to be government spies infiltrating the rest of the group (the Resistance). One of the players (either a spy or Resistance member) is selected to be the Mission Leader. The government spies are made aware of each other without the Resistance knowing – the only thing the Resistance knows is how many government spies exist, not who they are. This process is conducted by the first Mission Leader, who instructs the group to close their eyes, for the spies to open their eyes and see each other, for the spies to close their eyes again, and then for everyone to open their eyes and begin the game.

The Resistance wins the game if three Missions are completed successfully. The Spies win if three Missions fail. The Spies can also win if the Resistance is unable to organize the Mission Team at any point in the game, this usually happened when there are 5 failed votes on a single mission. The game ends immediately after either three successful or three failed missions. The Resistance wins if three missions are successful. The spies win if three missions fail.

A fundamental rule of the game is that players may say anything that they want, at any time during the game. You are allowed to say anything, to any one, at any time as long as it is said publicly. information in The Resistance comes at multiple levels. First are players' voting patterns, second are Mission results, and third are cues that you can discern from player interactions. Resistance Operatives must use all the information at hand to root out the spy infestation. It's not easy to overthrow a powerful government. You can expect the spies will win a significant proportion of the time with the core game rules, particularly when played with more than 7 players.

## **1.3 Implementation with Python**

The Resistance game is implemented in Python, its framework and implementation can be found [here](https://cseegit.essex.ac.uk/ce811/assignment1) in the CSEE GitLab. To run the game, we can run the competition.py pass a parameter of game rounds (1000 e.g.) to it, and then along with players(beginners.py or intermediate.py e.g.) , when successful, this will run a competition with 1000 games, with each of the bots in the beginners or intermediate file, besides, it will output a table of results, including each players wining rate in both spy team and resistance team, in addition to a series of other parameters such as percentage of vote, select and extra. Additionally, based on this Python implementation of the Resistance game, we can use what we have learned to make much more improvement. On the on hand, we can improve the performance of the players by modifying and updating their logics. On the other hand, it is also a good idea to implement neural network models as classifiers to help players recognizing spies, this is done by training the neural network models with different input dataset, in this case we are using logs recording through the game. Besides, there are a lot more techniques that we can implement in this game to make it better and better.

In this project, we are using the technique we learned to implement new players in the Resistance game, who should give better performance while running the competition file. Specifically, the new player will be implemented during this project, not only using predefined action with relatively strong logic, but also implemented with neural network which can help the player to easier recognize spies not just by logic but also a “brain” learned each other players behaviour from previous game rounds.

## **1.4 Framework of the Report**

This project follows the logical structure of the project from beginning to the end, starting from introducing the Resistance game along with its game rules and the some important techniques when implementing this game with Python. It is a very interesting game involving a series of game theory, data structure, decision tree, path finding algorithms and neural networks technique, which will be discussed with more details in Technique section. Additionally, in the next section of experimental study, after achieved basic logics and a trained neural network, which all implemented on our bot, we turned to do some experiments to push the bot performance even more. Besides, we analysed the results from the experiments we did. And at the end of this report, we will explain the conclusion and have a small brainstorm about the future development scope of this project.

# Background

There have been plenty of attempts to applying artificial intelligence techniques in a game. As for the Resistance game, however, only few studies did close investigate with available AI techniques for this game.

Taylor provide an overview of the game and the mechanics which make it an interesting subject [1], a review of related games and AI techniques, descriptive analysis of existing agent implementations, and original investigation focussing on the analysis of “suspicion inaccuracies” and opponent modelling techniques. And they conclude with a summary of our analysis and specific suggestions for the direction of future research.

Jack and his team from MIT developed Deep Role, a multi-agent reinforcement learning algorithm that addresses the challenge of learning who to cooperate with and how [2]. These innovations enable Deep Role to scale to the full Avalon game. Empirical game-theoretic methods show that Deep Role outperforms other hand-crafted and learned agents in five-player Avalon. Deep Role played with and against human players on the web in hybrid human-agent teams. We find that Deep Role outperforms human players as both a co-operator and a competitor.

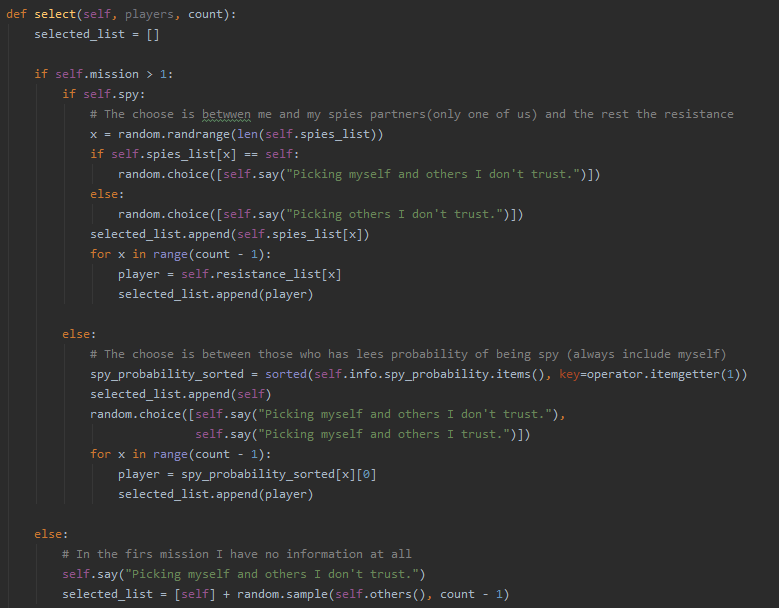
# Techniques Implemented

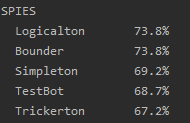
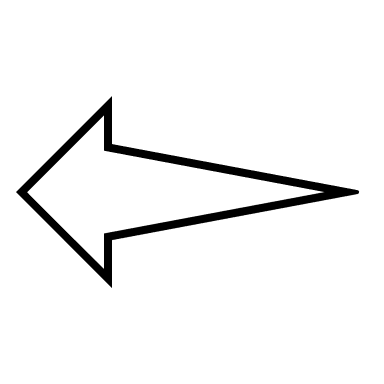
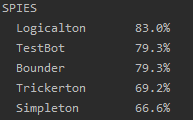
## **3.1 Applying Logic**

According to the Resistance game rule, instead of letting the bot play the game randomly, we can apply some basic logic to the bots when they are in a game. These logics will help the bot to make smarter decisions when selecting a team leader or vote for a mission, in order to win more games.

For an example, a basic logic here is, spy would better not let other players recognize itself as a spy, similarly, resistance would not select a team which possibly including spies. Hence, when the player itself is a spy, it would not choose others as leader but itself, on the other hand, if the player itself is a resistance, choose only between those who has less probability of being a spy.

After updating a series of logics for select and vote function, the Test Bot wining rate comes up from 68.7% to 79.3% when in the spy team.



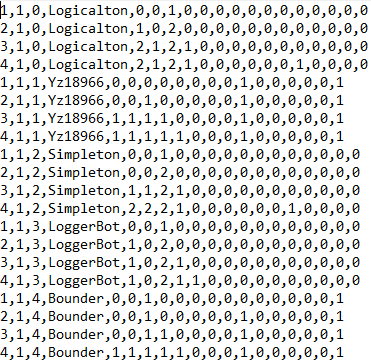
  

## **3.2 Neural Network Model**

After we have some basic logic for the bots to make relatively smarter decisions, we shall now implement some more advanced techniques to improve them even more. By applying a neural network, to analysis the game and players, our bot will have better and better performance as if they have a brain.

### 3.2.1 Generate the Data for Training the Model

In order to implement neural network, we have to get some useful data to train the model, the data can be in various kinds, lists, text, images, videos or even signals. In the Resistance game, with the aim of recording the game status, we are making fully use of the game logs, generated throughout the game.



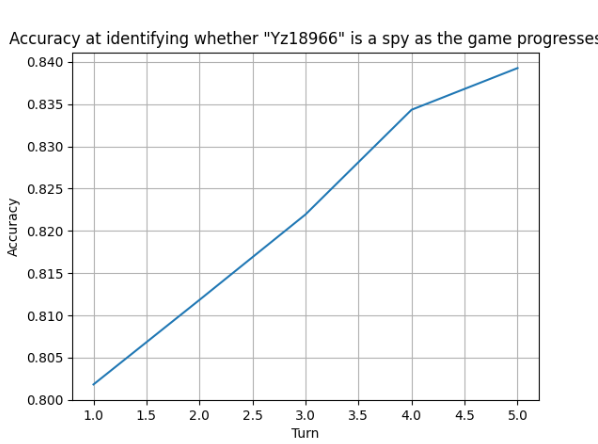
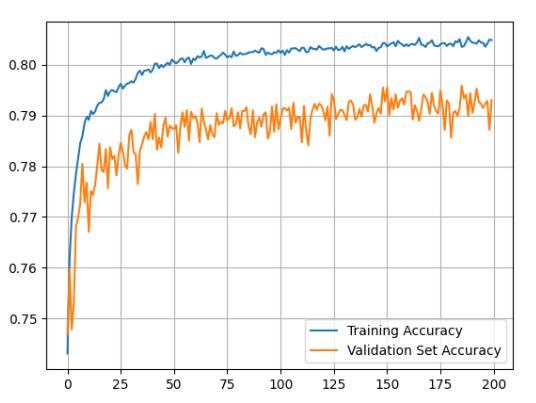
### 3.2.2 Keras Neural Network with TensorFlow

In this project, we are using TensorFlow as a framework to build our neural network. First we us pandas to load the log file as in csv format, so that we can well organize our log file into 18 clear and neat columns along with its labels in order. And then we use Numpy package toolkit to filter out only the columns we want to use as input vector for our neural network. Besides, we also split the dataset into a training data set and a validation dataset, choosing 70% of the data for training and 30% for validation. Afterwards, we define the Sequential model from Keras with 3 layers and set our optimizer as Adam, as well as the loss function by using Sparse Categorical Cross Entropy, as we are doing classification here.

### 3.2.3 Results

After setting our model well, we started training and validating our model, and then we plot out the training curves as well as the accuracy of identifying spies. At the end, we saved our classifier in a local folder.

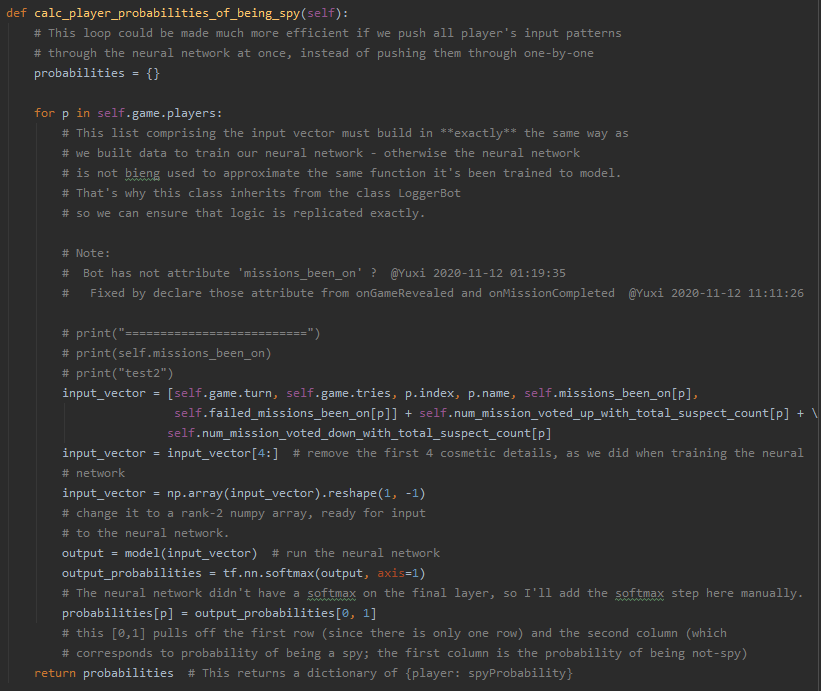
In this case, we got over 80% as training accuracy and 79% as validating, which is a problem here, as validation accuracy usually suppose to be higher than training one.



## **3.3 Appling the Classifier on a New Bot**

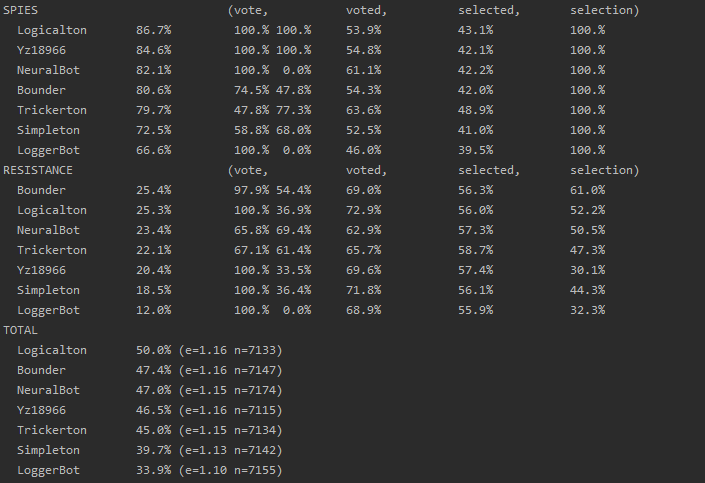
In the previous work, we built, trained and saved our neural network. It is time to apply it on our bot.

We firstly load the classifier we saved before, which is well trained by a relatively huge log with more than 32,000 KB, and then inherit the function of calculating the probabilities of being a spy from the previous neural bot, along with the logics we updated before.



## **3.4 Results**

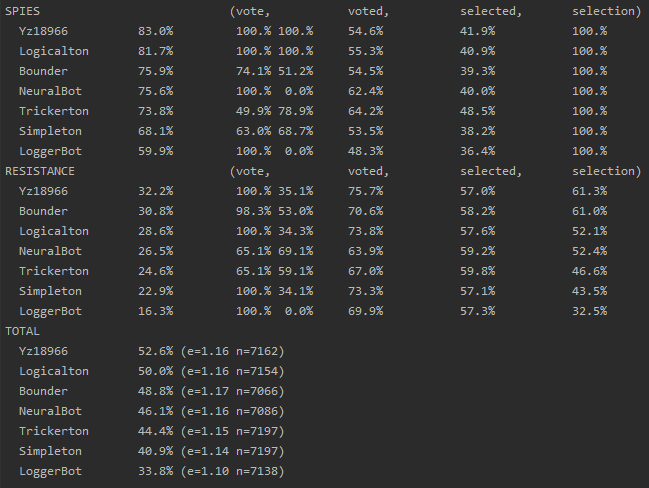
Along with both logic and neural network classifier, our bot reached 84.6% wining rate as a spy, which still gives us some scope to improve more.



# Experimental Study

## **4.1 Modify Logic of the Bot**

According to the game result in the above section, even with a trained neural network, our bot still cannot beat Logicalton bot with the wining rate when in the team spy, who does not have a neural network but do have a relatively well designed logic. In this case, there is definitely some scope for improvement in logical aspect.



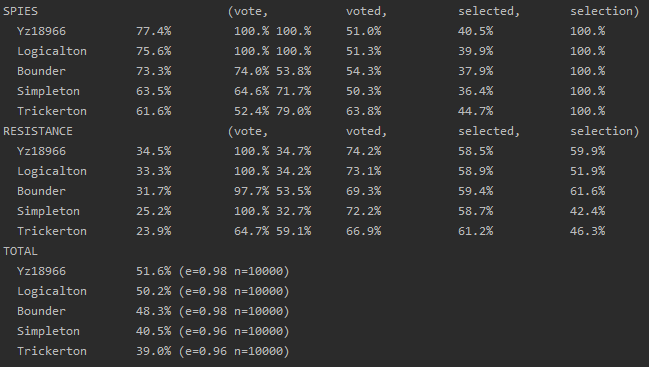
## **4.2 Use Different Logs as Training Dataset**

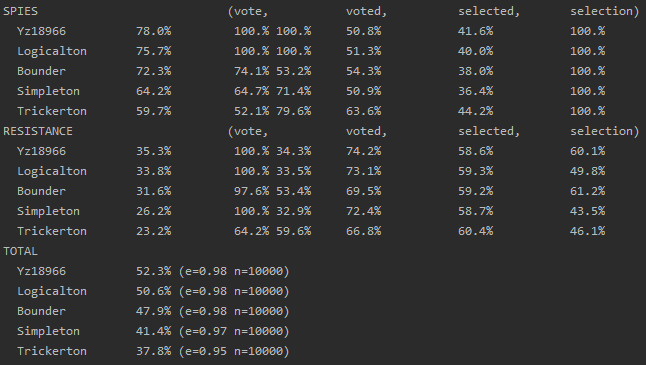
After we applied a better logic, we turn to work on our neural network more.

### 4.2.1 Use Double Size of Logs

In order to have more input data for our neural network, we run the competition over 40000 rounds, which makes the log generated twice large than our previous one. Afterwards, we use this new log as input data set to train our neural network.

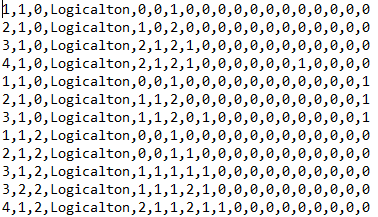
Not much improvement, 77% -78%

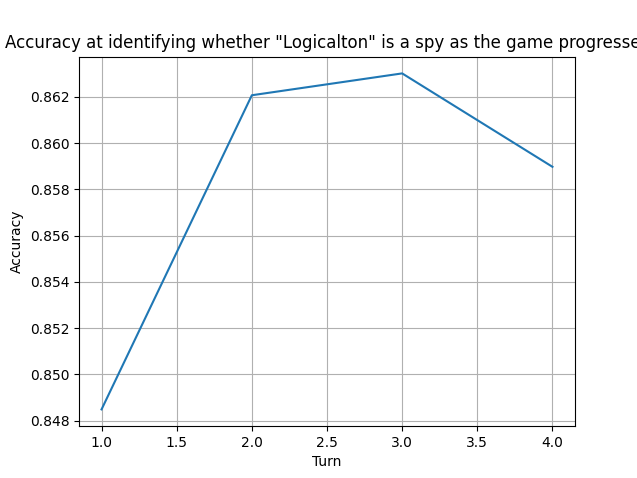




### 4.2.2 Use Certain Player’s Log as Training Dataset and Play Against this Player

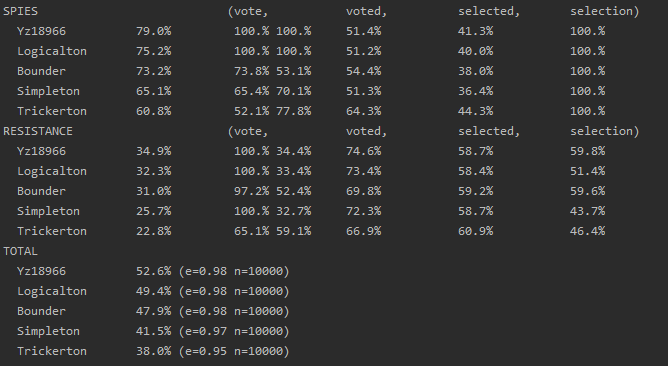
In this experiment, we had a idea of training the neural network with only certain player, and then apply this new classifier to play against this certain player, after knowing its behaviour much more than before.





Here is the result against player “Logiclaton”, this time our bot beat Logicalton with a relatively higher wining rate as a spy, from previous 78% to 79% now.

Which is not a big improvement, but according to I am using the logs selected manually by myself, the training data set is really small, so I assume the improvement will be good if I use a bigger log as input training data, which will be done in the future work with interest.

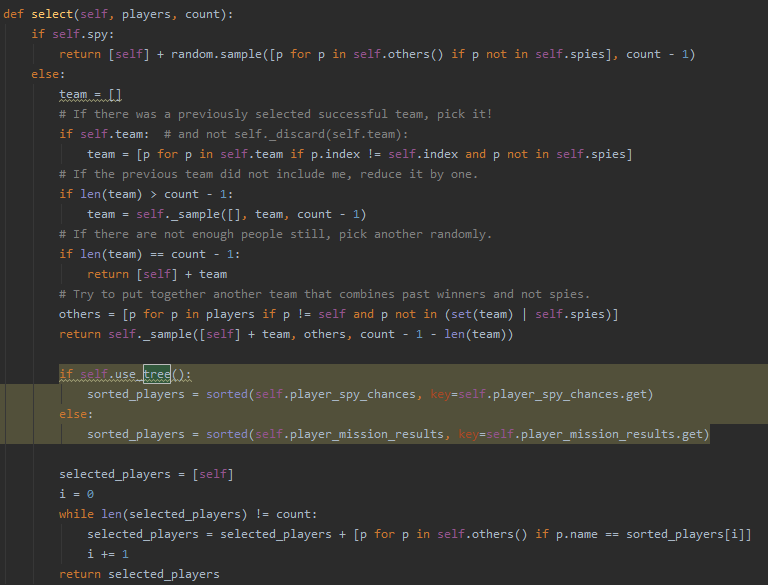


## **4.3** **Configure Hyperparameters in Model Settings**

Learning rate

batchsize

## **4.4 Apply Decision Tree As Classifier**



## **4.5 Implement Text Log Analysis by NLP**

# Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Techniques** | Wining Rate Spy | Wining Rate Resistance | Wining Rate Avg |
| Updating Logic | 79% | 32% | 47% |
| Applying NN | 83% | 35% | 52% |
| Applying Decision Tree | 86% | 37% | 53% |
| Modify Hyperparameters | 87% | 37% | 53% |

# Conclusions and Future Work

Applying Decision Tree and NN can improve the bot performance more

In the future I am think applying Natural Language Processing to analysis the Logs more.

# References

|  |  |
| --- | --- |
| [1] | Taylor, Daniel Peter, “Investigating Approaches to AI for Trust-based, Multi-agent Board Games with Imperfect Information,” in *University of Derby*, 2014. |
| [2] | Jack Serrino, Max Kleiman Weiner, David C.Parkes, Joshua B.Tenenbaum, “Finding Friend and Foe in Multi-Agent Games,” in *Advances in Neural Information Processing System 32 (NeurIPS 2019)*, 2019. |

1. https://en.wikipedia.org/wiki/The\_Resistance\_(game) [↑](#footnote-ref-1)