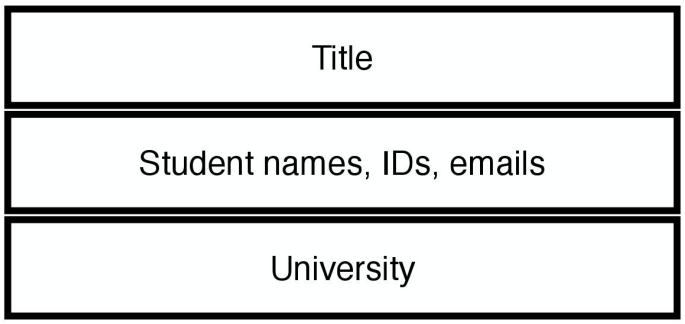


Abstract. The Resistance is a multi-agent board game with imperfect information, based heavily on trust and deception. This report summarises the development of an Artificial Intelligence Agent to play the game against four other opponents, including a literature review that explores previous research in this area. Techniques used include opponent modelling through the use of frequency-based attributes, expert rules, and machine learning through the use of a decision tree classifier. The performance of the classifier is compared with that of a neural network and nave Bayes classifier. The AI agent is compared to other agents which had previously been implemented, and results are presented which show the agent to be capable of performing comparably.

Table of Contents

List of Figures

List of Tables

1 Introduction

The problem of developing an Artificial Intelligence (hereinafter AI) for a game is often more difficult than initially anticipated. There are many approaches to solving the problem for any given game; there are multitudes of plausible algorithms that an be explored, including decision trees, neural networks, Monte-Carlo tree search, evolutionary algorithms, and even genetic programming (a subset of evolutionary algorithms). This report will detail the attempt to develop a reasonable AI for a board game known as *"The Resistance"*, a relatively unknown, team-based board game which relies heavily on deception. Such a game is interesting in terms of game AI development, as humans playing the game rely mainly on cues or "tells" from other players. An AI, on the other hand, must use only statistics and logic.

This report will outline some of the previous research into game AI development, specifically focusing on AI research surrounding board games with imperfect information. It will then detail the functionality of the bot developed alongside the report, including some performance benchmarks and comparisons with other bots. The development of the bot will be critically discussed before concluding the report and proposing future work.

2 Background

2.1 The Resistance

The Resistance is a board game for between five and ten players, which revolves around the aim of deducing the role of the other players [1]. There are two roles—"resistance members", and "spies". Each game consists of five rounds, wherein both the spies and resistance members will attempt to "win" the mission. The spies must subvert and attempt to deceive the "resistance members". In terms of categorisation of this game for Game AI purposes, it can be seen that The Resistance is a multi-agent, imperfect information, strategy board game.

At the beginning of each game, players are randomly assigned a role—either "spy" or "resistance". There are a set number of spies and resistance members for each possible number of players. The spies are informed of who the other spies are, but the resistance members remain unaware of who has the role of spy.

Each round, a new leader is selected. The leader must select a particular number of players to be on the team. Then, each player votes to either approve or reject the selected team, with the majority vote being carried out. If the team is rejected, a new leader is chosen (typically the player to the left of the current leader), and a new team is selected. If the selected team is rejected five times in a row, the spies automatically win the round.

If a selected team is accepted, the mission is "carried out", which means that any spies in the team may choose to sabotage the mission. If one or more sabotages (two or more in certain cases) are performed the mission fails and the spies win the round, else the resistance wins. Therefore, as a resistance member, the correct strategy is to reject any teams that contain a spy, and accept "clean teams". As a spy, the strategy is

to ensure that at least one spy is on the mission so that it can be sabotaged. At the end of the game, the group with the most wins (resistance or spies) wins.

2.2 Literature Review

There has been much research into the area of Game AI, particularly in recent years with exciting developments having made the headlines, for example the AlphaGo bot [2] which learned to play Go (a notoriously difficult game to program an AI bot for). However, the majority of such research relates to two player games easily represented as a tree, allowing algorithms such as minimax algorithms and other adversarial search variants to be used. Unfortunately, turn-based strategy games, a category under which The Resistance technically falls, are often typically unable to fit into such a representation as they are too complex [3].

Instead, other techniques must be used for more complex games. Opponent modelling is a common technique used to assist in the development of a suitable AI for many games. For instance, it is commonly used in the development of bots for Poker games [4,5,6]. Broadly, an opponent model is an abstract representation of a player or the behaviour of a player in a given game [7]. Opponent modelling can be used for several purposes: to inform a human player of a game, to provide an AI opponent for a human player of a game, or to produce an AI capable of participating in a multiplayer game. The last case, producing an AI for a multiplayer game is particularly relevant for this project, and therefore opponent modelling was a concept carefully considered before commencing development of the AI for this project.

Schadd et al. explored the task of making an AI behave realistically in a real-time strategy game [8], using opponent modelling to produce believable behaviour. The researchers viewed opponent modelling as a classification problem, using it to classify the behaviours of opponents into one of a few classes. This is similar to the requirement of the AI in The Resistance to determine whether each other player is a spy or a resistance member. In the study, a fuzzy model classifier was used as a top-level classifier, while a discounted rewards scheme was used as a bottom level classifier.

Ganzfried and Sandholm combined game-theory and opponent modelling in their 2011 study [9]. The researchers were able to develop an algorithm that observed the frequencies of each action performed by opponents, and combined these with a precomputed strategy, in order to discern the best possible action. This study was novel in its use of game-theoretic reasoning, which meant it was able to avoid relying on large sets of historical gathered data. The technique of observing the frequencies of each action is simple, yet effective, and so this is a mechanism that was used in the development of the AI in this project.

There is little previous research or literature available on The Resistance. However, a 2014 paper by Taylor investigated techniques for developing AI for The Resistance, or more specifically the analysis of such AI to determine relative performance [10]. Over the course of the paper, many previous AI implementations were examined, of which a large proportion used Opponent Modelling. The paper sought to find which techniques could be excised from these implementations, possibly combining them to produce a better performing AI. The result was a combination of Opponent Modelling and Monte-Carlo Tree Search (MCTS). MCTS is a very common technique used in

game AI development, especially for simple board games [11]. Taylor also discussed some of the bots' use of expert rules in certain situations to provide logic-based decisions.

3 Techniques Implemented

The final state of the AI agent developed during the course of this project relies on a few core concepts mentioned previously. It is important to note that the behaviour of the AI is markedly different depending on the role that has been assigned to it, i.e. whether it is a spy or a resistance member. This is due to the fact that the different roles have different objectives, and additionally, when the AI is assigned the role of spy, it has perfect knowledge of which opponents are also spies. Therefore, it is only necessary to deduce the roles of the opponents when assigned the role of "resistance member". It can hence be concluded that the more difficult role to design an algorithm for is "resistance member". Opponent modelling is used in the classification of opponents as either "Spy" or "Resistance" players (through a large decision tree), when the player is assigned the role of "resistance member". In addition, expert rules are used in certain circumstances as a resistance member, whilst they make up the entirety of the spy behaviour implementation.

This section will detail the techniques used in the final AI agent, and give a brief explanation of such techniques. A discussion of how these techniques were chosen can be found in the Experimental Study section.

3.1 Opponent Modelling

Opponent modelling is perhaps the most important aspect of the final AI, as it is what allows the bot to distinguish between "resistance" and "spy" opponents, when given the role of "resistance". In a technique similar to that used in Ganzfried and Sandholm's study [9], the opponent modelling is done based on frequencies of particular events on a per-agent basis. However, this is not limited to actions performed by the agent. Instead, 13 attributes were isolated, including actions and attributes of the game state, which allowed opponents to be classified as either "spy" or "resistance". The attributes are listed below. Attributes which are not specific to each opponent, but rather are game variables, are denoted with an asterisk (*).

- Game Turn*—The current turn, or "round", of the game
- Tries*—The number of attempts that have been made to select a team this round (incremented every time a team is rejected, and if it is the fifth turn and the team is rejected the spies automatically win the round)
- Total wins*—The number of rounds which have been won by the resistance
- Total losses*—The number of rounds which have been won by the spies
- Selected—The number of times this opponent has been selected for a team that was approved
- Sabotages—The total number of sabotages that have occurred while this opponent was part of a team. This counts the individual number of sabotages, so if two players sabotage one round, each member of the team has this counter incremented by two.

- Upvotes—The number of times this opponent has voted to accept a team
- Downvotes—The number of times this opponent has voted to reject a team
- Mission Fails—The number of times this opponent has been part of a team that was rejected
- Leader Fails—The number of times this opponent has been the leader of a team that was rejected
- Wins—The number of times this opponent has been part of a team where the resistance won the round
- Losses—The number of times this opponent has been part of a team where the spies won the round
- Self selection—The number of times this opponent has selected itself (when team leader) to go on a mission

The attributes above are stored and modified per-agent each game. A decision tree is used with these attributes to classify each opponent as either "spy" or "resistance". To generate the decision tree, attribute data was collected from a wide variety of bots over a very large number of games. This is discussed further in the Experimental Study section. The collection of data over a large number of games means the generalisation achieved by the decision tree can be considered to be learning over a large number of instances.

This approach is relatively simple, yet elegant—the inclusion of the *Game Turn* attribute means that not only is learning performed over many instances, but also within each game. This is because the *Game Turn* attribute allows turn-specific information to be encoded in the decision tree. This is important, as the amount of information available to the AI increases towards the end of the game, in the later rounds. So, relying on the same ruleset for every turn would be suboptimal—more complex decisions require more data which is not available at the start of the game. By using the *Game Turn* attribute, more complex decisions can be placed into lower levels of the tree and used for later turns, while earlier turns use the upper levels of the tree.

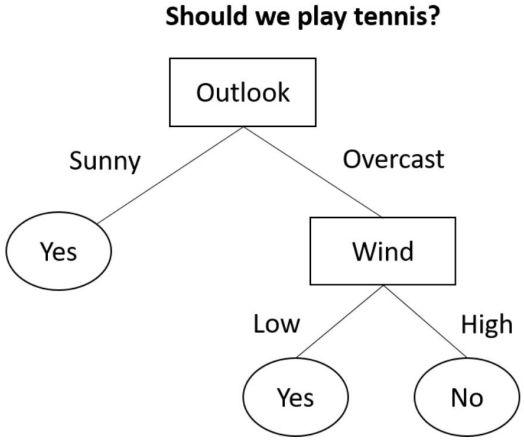
When used in conjunction these attributes provide a decent opponent model. While at first the attributes may appear unusual, upon reconsideration, it intuitively makes sense—a spy is likely to have more sabotages than a resistance player, while a resistance player will likely have more downvotes than a spy player, assuming they play with a suspicious playstyle, as is typical. There are many more such relationships which can be revealed using decision tree induction algorithms.

3.2 Decision Trees

As mentioned, a decision tree is used to provide the opponent modelling aspect of the AI. A decision tree is a very simple technique which can be used to model complex algorithms or decision making processes, and can be used as both a decision maker and a classifier. The tree takes a set of attributes, also known as a feature vector, and makes decisions based on the value of each attribute. As the name suggests, a decision tree has a tree like structure—each node has two or more possible branches based on some decision. Each decision checks one value from the feature vector, typically a numerical value or boolean condition, and based on the value, chooses one of the child branches

of the node to traverse. If the node is a leaf node, i.e. it has no child branches, it is an "action" (or in classification, a class), and is the termination of one of the decision paths through the tree. Fig. 1 gives a demonstration of a simple decision tree.

Fig. 1: A sample decision tree showing a simple decision process of whether to play tennis or not



The most frequently used type of decision tree, and indeed the type used in this project, is a binary decision tree. This means that each decision node may have at most two possible outcomes. There are alternatives to this, however: N-Ary decision trees allow as many branches as the implementer desires from each decision node, and so are potentially easier to create by hand, and Directed Acyclic Graphs are an extension of decision trees which allow nodes to be reached via multiple branches. This means that nodes do not have to be repeated, and hence the technique is more efficient. However, the implementation is far more difficult, and so binary decision trees are chosen more often.

There are many algorithms that can be used to construct a decision tree, all of which require a set of training data with inputs and desired outcomes, but the key requirement is that each decision node in the tree (when used for classification) should separate the output classes as much as possible. Or, in other words, each node should maximise the information gain from splitting the training data. For the purposes of this project, the J48 algorithm was used to build the decision tree in the AI agent. J48 is an open source Java implementation of the C4.5 algorithm for decision tree building. This is discussed further in the Experimental Study section. This project collected training data from 50000 games with a variety of other types of bot (beginners, intermediates, experts and "invalidator"). The attributes collected were those mentioned in the Opponent Modelling section above. A decision tree was then generated based on this training data.

3.3 Expert Rules

Whilst the bulk of this section has thus far focused on the "resistance role" AI, and how it deduces the role of its opponents, it is also important to consider the "spy role" AI. This operates entirely using "expert rules", which are preprogrammed, deterministic rules that do not rely on any sort of learning. It should be noted that the resistance AI uses expert rules too, in order to apply some basic logic and also to use the information produced by the opponent modelling decision tree.

Spy AI There are three situations in which the AI must perform an action when given the "spy" role: voting on a team, selecting a team, and deciding whether to sabotage a mission or not. The rules are explained below, and it can be seen that, on the whole, this bot is very simple indeed. The most complex rules are in the "sabotage" function, mainly because the bot tries to minimise suspicion on itself. This is important to do because if the opponents become too suspicious, the chance of them selecting the AI for the team is decreased, and hence the chance for the AI to subvert future missions is also decreased.

- *Voting*: The rules for voting on a team are very simple—if the team contains a spy (or the AI itself), vote to approve the team, else vote to reject it.
- *Team Selection*: The rules for selecting a team are also very simple—the AI simply selects itself and a random sample of the other players.
- *Sabotaging*: The rules for deciding whether to sabotage the mission are slightly more complex.
  1. The AI will never sabotage the first mission. This is a common strategy for minimising suspicion, as there are only two players sent on the mission, so if it is sabotaged there is a 50% chance (in the eyes of the opponents) that the AI is a spy.
  2. The AI will always sabotage the fifth and final mission, because it no longer matters about suspicion, so it's essential to try to win this round.
  3. If it is not the first or last turn, the AI will examine the team and count the number of spies present. If there is another spy in the team, the AI will vote *NOT* to sabotage. The reason for this is that, when examining the rulesets of other AI implementations, it was found that almost all of them were greedy—they immediately chose to sabotage the mission if it was not the first turn. Therefore, this AI assumes that any other spies will automatically sabotage, and tries to decrease suspicion by not voting sabotage, as only one sabotage is required to let the spies win.

Resistance AI As mentioned previously, the resistance ruleset is by far the more complex part of the AI. Although the majority of the complexity is encoded in the decision tree, there are still some complexities in the expert rules, as seen below. Naturally, there are only two situations in which the AI is required to perform an action as resistance: voting and selecting.

- *Voting*: The rules for voting are more complex than those for team selection.

1. If this is the fifth attempt at making a team, the AI will always approve it, because the spies will automatically win if the team is rejected. The risk that a spy will sabotage the round is less than the certainty that the spies will win the round if the team is rejected.
2. If the team contains three members, and the AI is not one of them, then vote to reject the team. The reason for this is as follows: there are *always* five players in this framework. Two of these players are always spies. This AI is not a spy. Therefore, in a team of three without this AI, there is guaranteed to be *at least* one spy.
3. If the AI is the leader of the team, vote to approve the team. The AI will never knowingly select a spy.
4. The AI counts the number of suspected spies in the selected team, using the decision tree to classify each member. If there are any suspected spies in the team, the AI votes to reject it, else if all members are suspected resistance the AI votes to approve.

– *Team Selection*: Team selection is reasonably simple for this bot.

1. The AI always selects itself for the team.
2. The AI then iterates over the other players in the game, adding any that the decision tree indicates are *NOT* spies to the team.
3. If the team is now too large, truncate it to the required size.
4. If the team is too small, randomly select enough players from the remaining players to bring the team to the required size. Of course, this risks adding a spy, but given that the decision tree does not have 100% accuracy, this is the best compromise.

3.4 Variants on these techniques

While these techniques produced a reasonably successful AI, there are many variants or alternatives which ostensibly could provide better performance. For instance, there are alternative classification methods that operate on feature vectors that could give higher levels of accuracy. A small selection of such techniques is performed in the Experimental Study section. The expert rules used in the AI could certainly be changed, and different attributes could be chosen to use in the classification of opponents. One of the primary areas that could be improved is the performance of the AI when in the spy role, as the rules at the moment are very simple.

4 Experimental Study

The development of the AI occurred in several stages, from producing a program that could be used in the framework and environment provided, to collecting data, to finally building and implementing the decision tree classifier.

4.1 Environment

For this project, a computer simulation of The Resistance, written in Python, was used (originally designed for a competition held by AiGameDev in Vienna in 2012 [12]). The framework allows competitions of bots to be run over a specified number of games, and allows for comparison of the results of each bot, to see which was the most effective overall. The framework always plays games with five players. This is good, because it means the number of spies is constant—there are always two spies and three resistance members per game.

The framework allowed the performance of the AI to be rapidly assessed after each modification. This proved very useful for development purposes. Additionally, it was easy to test the AI against other premade bots, and to vary how many games were played at once.

4.2 Data collection

Once a basic AI had been implemented that participated in the game without any special logic, the data collection process began. In order to collect meaningful data, it was necessary to have perfect information; to learn to classify other players whilst in the role of resistance, the AI had to know who the spies were, even as resistance. Of course, the game does not ordinarily inform resistance players who the spies are, so the environment was modified, only for the data collection step, to inform this AI which players were spies. This allowed data to be produced that could be used to train a classifier.

Due to the vast number of possible permutations and combinations of bots, roles, and actions, it was necessary to collect and analyse vast amounts of data. Processing this amount of data by hand would have been tedious and error-prone, so the decision was made to use Weka, a machine learning and data mining toolkit written in Java [13]. Therefore, during data collection, the AI recorded the data to file in ARFF format. An excerpt of the final data file is presented in Fig. 2.

From the literature review, it was evident that some attributes had to be chosen in order to allow for Opponent Modelling to be performed, as well as a classification method. To begin with, just four attributes were chosen: *upvotes*, *downvotes*, *wins*, and *losses*. As more attributes were added, the data collection was rerun, to ensure that all pieces of data included all attributes.

4.3 Building the classifier

To build the classifier, Weka requires an input data file, the generation of which was covered in the previous subsection. The input file contains all the attribute values and classes, essentially represented as a set of feature vectors. Weka allows for the input data to be split into a training set and a validation set—this is important, to ensure that a model that has been generated is not overfitting the input data. Typically it is incorrect to test the performance of a classifier on the training data, instead, the validation set is used to give an idea of the performance once the training set has been used to train the classifier. For this project, a 66/34 split was used for training and validation sets. This

Fig. 2: An excerpt from the final data file, collected by the AI over 50000 games with various bots

```
@RELATION spies-voting

@ATTRIBUTE turn NUMERIC
@ATTRIBUTE tries NUMERIC
@ATTRIBUTE totalwins NUMERIC
@ATTRIBUTE totallosses NUMERIC
@ATTRIBUTE selected NUMERIC
@ATTRIBUTE sabotages NUMERIC
@ATTRIBUTE upvotes NUMERIC
@ATTRIBUTE downvotes NUMERIC
@ATTRIBUTE missionfails NUMERIC
@ATTRIBUTE leaderfails NUMERIC
@ATTRIBUTE wins NUMERIC
@ATTRIBUTE losses NUMERIC
@ATTRIBUTE selfsel NUMERIC
@ATTRIBUTE class {True, False}

@DATA
2,1,1,0,1,0,1,0,0,0,0,0,1,True
2,1,1,0,2,0,1,0,0,0,1,0,1,False
2,1,1,0,1,0,1,0,0,0,0,0,0,False
2,2,1,0,2,0,1,1,0,0,1,0,1,True
2,2,1,0,1,0,0,2,0,0,0,0,0,False
2,2,1,0,2,0,2,0,1,1,0,0,1,True
```

means that 66% of the input data was used to train the classifier, and 33% was used to in the validation set to test the performance.

The choice of classifier was largely dictated by ease of implementation—while Weka provides the facilities to generate Neural Networks, Bayesian Classifiers, Logistic Classifiers, and so on, the easiest of all of these to actually implement in an AI is most assuredly decision trees, and so decision trees were chosen for the classifier in the AI. Incidentally, Decision Trees are also arguably the decision making algorithm most easily interpreted by humans. For all of the following results, 50000 games were run to collect the data, however, it should be noted that the exact number of instances of data may vary due to the somewhat random nature of the game. Once the decision tree had been generated by Weka it was manually converted into a large Python conditional tree.

As mentioned earlier, the early stages of the AI collected just four attributes for opponent modelling and classification: *upvotes*, *downvotes*, *wins*, and *losses*. The default decision tree algorithm provided by Weka was used (J48, an open source Java implementation of C4.5), with default parameters, to generate the tree. These parameters were varied with the final test set to produce the best decision tree possible. A confusion matrix presenting the results of using just these attributes can be seen in Table 1. In total, 69199 data items were used for training, and 35648 were used for validation. The table shows that of 35648 validation samples, 23953 were correctly classified, and 11695 were incorrectly classified, yielding an accuracy of roughly 67.2%. Recall values (the proportion of correctly classified data items) for classification of spy and resistance members can be seen in equation 1. Note that the recall value for the spy class means that the decision tree misclassifies spies more often than it correctly classifies them. This is an issue, because classifying spies correctly is arguably the most important part of the AI when in the role of a resistance member. Hence, these results warranted further experimentation to increase the accuracy of the classifier, despite the relatively high accuracy of classifying resistance members.

$$\begin{aligned} recall_{spy} &= 3619 \div (3619 + 9020) = 0.286 \\ recall_{res} &= 20334 \div (2675 + 20334) = 0.884 \end{aligned} \tag{1}$$

Table 1: Confusion matrix for the initial decision tree

		Predicted	
		Spy	Resistance
Actual	Spy	3619	9020
	Resistance	2675	20334

The next attempt to produce a decision tree included eight more attributes in an attempt to increase the accuracy: *tries*, *total wins*, *total losses*, *selections*, *sabotages*, *mission fails*, *leader fails*, and *self selection*. Table 2 presents the confusion matrix for

this decision tree, with a total of 38618 validation samples (after 74964 training samples to generate the decision tree). Equation 2 shows the recall values for each class. It can be seen that the recall rate for spy classification increased considerably over the first decision tree. However, spies were still misclassified more often than not. Resistance classification decreased slightly, as a consequence.

$$\begin{aligned} recall_{spy} &= 6157 \div (6157 + 7564) = 0.449 \\ recall_{res} &= 21642 \div (3255 + 21642) = 0.869 \end{aligned} \tag{2}$$

Table 2: Confusion matrix for the second decision tree

		Predicted	
		Spy	Resistance
Actual	Spy	6157	7564
	Resistance	3255	21642

The final attribute to be added was the *Turn* attribute—the attribute that allowed the decision tree to encode turn-specific information and model opponent behaviour that changes according to turn. This attribute provided a surprising jump in accuracy, especially when combined with some fine tuning of parameters in the decision tree induction algorithm (decreased confidence value to cause more pruning), as can be seen in the confusion matrix in Table 3, and the recall values in Equation 3. This classifier was trained with 78425 data items, and validated with 40401. The total accuracy of this decision tree is 76.8%, correctly classifying 31019 items of the 40401 in the validation set. The decision tree correctly classifies spies more often than not, in 60.8% of cases. This actually provides a very good success rate when implemented in the final AI agent.

$$\begin{aligned} recall_{spy} &= 8720 \div (8720 + 5625) = 0.608 \\ recall_{res} &= 22299 \div (3757 + 22299) = 0.856 \end{aligned} \tag{3}$$

Table 3: Confusion matrix for the final decision tree

		Predicted	
		Spy	Resistance
Actual	Spy	8720	5625
	Resistance	3757	22299

4.4 Alternative methods

Despite decision trees being chosen, other methods (in particular neural networks and Bayesian classifiers) were tested whilst tuning parameters for the decision tree building process in Weka, to compare the performance of the techniques, purely for academic interest. The confusion matrices of a Nave Bayes Classifier and Neural Network trained on the final training data are presented in Tables 4 and 5, respectively. The neural network was trained over 1000 epochs. It can be seen that the Nave Bayes Classifier did not reach the same level of accuracy for classification of either class; the decision tree performed better in both cases. It can also be seen that the neural network demonstrated worse performance than the decision. This is surprising, given that neural networks typically perform very well, often above other classification methods. It is likely that training over a larger number of epochs would give better results.

Table 4: Confusion matrix for a Nave Bayes Classifier

		Predicted	
		Spy	Resistance
Actual	Spy	8333	6012
	Resistance	7045	19011

Table 5: Confusion matrix for a Multilayer Perceptron Neural Network

		Predicted	
		Spy	Resistance
Actual	Spy	8482	5863
	Resistance	3799	22257

4.5 Performance relative to other agents

Whilst the statistical performance of the decision tree classifier has been explored, there are other aspects to the AI, namely the expert rules, which determine the performance of the bot. Fortunately, the framework allows easy testing against other bots. In its final state, the AI was tested for 5000 games against a combined pool of the "intermediates" and "experts" bots provided with the framework. The results are shown in Fig. 3. The results can be a little hard to interpret, but to see an overall picture of which bot performed the best, the "TOTAL" section should be considered. It can be seen from the figure that the AI developed for this project (called "jforre") beat all the other bots. This is a very

encouraging result. It should be noted that the "vote" and "selection" columns in the "RESISTANCE" section that the bot was quite accurate at voting to accept spy-only teams (the AI voted to accept 73.7% of spy-only teams), reject teams containing spies (the AI rejected 68.9% of teams containing spies), and select spy-free teams (52% of teams selected by the AI were free of spies). Additionally, the "selected" column in the "SPIES" section shows that the AI was selected for 47.5% of games, which indicates the bot was good at avoiding suspicion.

Fig. 3: Results from the Intermediates and Experts competition

```
SPIES                    (vote,          voted,  selected,  selection)
  jforre          76.4%  100.% 100.%     55.4%    47.5%       100.%
  Suspicious      76.0%   83.9% 65.8%     54.9%    48.6%       100.%
  Bounder         68.0%   73.2% 56.1%     44.4%    40.6%       100.%
  Logicalton      60.5%  100.% 100.%      40.8%    41.2%       100.%
  Simpleton       59.1%   64.4% 70.2%     41.7%    38.9%       100.%
RESISTANCE               (vote,          voted,  selected,  selection)
  jforre          37.6%   73.7% 68.9%     63.2%    62.3%       52.0%
  Suspicious      37.3%   89.0% 61.0%     64.8%    59.5%       56.4%
  Bounder         32.0%   92.8% 46.5%     66.2%    57.0%       49.0%
  Logicalton      27.0%  100.% 35.1%      68.9%    54.4%       43.5%
  Simpleton       26.1%  100.% 32.7%      68.7%    56.5%       37.5%
TOTAL
  jforre          53.1% (e=1.38 n=5000)
  Suspicious      52.8% (e=1.38 n=5000)
  Bounder         46.4% (e=1.38 n=5000)
  Logicalton      40.4% (e=1.36 n=5000)
  Simpleton       39.3% (e=1.35 n=5000)
```

Due to the encouraging results of the previous competition, it was decided to run a final competition emulating the final round of the competition held as part of the Vienna Game AI Dev event. The top four bots in Vienna were: *PandSBot*, *ScepticBot*, *Rebounder*, and *Clymily*. Unfortunately the *Clymily* implementation provided with the framework did not work, so the *GarboA* bot was substituted instead for this competition. 62500 games were played as part of this competition, and the results are displayed in Fig. 4. It can be seen that the AI developed in this project performed quite well against these very advanced bots, coming third out of the five competitors. Of particular interest in the "RESISTANCE" section is the "voted" column which shows the AI was very well trusted when part of a team, and the "vote" column which shows that it was quite good at voting to approve teams with no spies, and performed respectably at voting down teams which contained spies. In the "SPIES" section, the AI was surprisingly well trusted—it was selected most often out of all bots in the competition, and approved with the second highest frequency of all bots. These results are surprisingly good considering the very simple implementation of the spy AI.

Fig. 4: Results from the “Vienna style” competition

```
SPIES                    (vote,        voted,  selected,  selection)
  PandSBot        60.8%  100.% 84.7%   39.4%   35.1%      100.%
  ScepticBot      55.7%  73.9% 83.5%   39.2%   37.4%      100.%
  jforre          54.3%  100.% 100.%   42.3%   44.2%      100.%
  GarboA          52.2%  100.% 99.5%   45.9%   19.7%      100.%
  Rebounder       51.5%  98.4% 100.%   32.6%   33.9%      100.%
RESISTANCE               (vote,        voted,  selected,  selection)
  PandSBot        49.0%  85.8% 64.5%   61.7%   64.6%      65.0%
  ScepticBot      45.7%  86.3% 66.0%   62.1%   64.3%      70.7%
  jforre          44.7%  81.1% 59.9%   63.9%   63.6%      55.9%
  GarboA          43.3%  81.5% 55.9%   57.7%   62.2%      51.8%
  Rebounder       42.9%  96.5% 48.4%   63.8%   61.8%      63.4%
TOTAL
  PandSBot        53.7% (e=0.39 n=62500)
  ScepticBot      49.7% (e=0.39 n=62500)
  jforre          48.5% (e=0.39 n=62500)
  GarboA          46.8% (e=0.39 n=62500)
  Rebounder       46.4% (e=0.39 n=62500)
```

5 Analysis

The steps undertaken during this project worked well given the limited timeframe. Reviewing the literature before starting intensive development meant that a head-start could be made by choosing a technique that would give good results based on previous research. In this case, opponent modelling through the use of frequency attributes, fed into a classifier in the form of a decision tree, gave surprisingly good results. Without the study performed by Ganzfried, it is possible the concept of frequency-based attributes would not have made it into the final version of the AI, which given the effectiveness of the technique would have been disappointing.

The development of the AI highlighted the usefulness of data mining tools such as Weka. Analysing the vast amounts of collected data by hand would likely have failed to reveal the complex relationships between attributes which were so easily deduced by the automatic decision tree induction. The use of the “Turn” attribute allowed in-game learning to be encoded into the decision tree through the use of historical data. This elegant approach provided a surprising improvement to results, once implemented.

6 Conclusions and Future Work

This project has demonstrated the effectiveness of data mining when applied to historical data, with a view to creating a generalised behaviour through decision making and classification. The use of opponent modelling, a reasonably common technique in game AI development, allowed a very simple classification method—decision trees—to achieve reasonably high levels of accuracy, beyond that of a neural network and nave

Bayes classifier, when using a feature vector composed of action frequency attributes and environmental variables. The project showed that combining several techniques—opponent modelling, expert rules, and decision making through classification—allows for good performance, where one of the techniques alone would not.

The AI produced during this project was able to outperform the sample agents provided with the framework; the so-called "intermediates" and "experts". It was also able to hold its own against some of the best bots known—those that participated in the final round in the Vienna competition. This was despite a very simple "spy AI" implementation, and also despite the classifier having been trained only on the "intermediates" and "experts". The fact that it performed so well against the "Vienna bots" shows that the classifier reached a good level of generalisation.

In future work, it may be interesting to perform more research into the performance of the "spy AI", as almost all of the focus was spent on maximising performance of the resistance side. Indeed, the same conclusion was reached by the organisers of the Vienna competition [12]. Almost all bots previously created for this game focus on the resistance behaviours, ostensibly because this is the most difficult part to play. However, it is likely that improving the performance of the spy role would yield an increase in overall effectiveness.

It would also be interesting to explore other attributes that could be used for opponent modelling in The Resistance. The set of attributes used in this project are by no means exhaustive, and new attributes would likely improve the effectiveness of the AI substantially. Additionally, it would make sense to explore other methods of classification, particularly with techniques such as neural networks. While the neural network was unable to perform quite as well as the decision tree in this project, it is likely that more fine tuning of the parameters would yield more impressive results.

References

1. Indie Boards & Cards. The resistance rules. [Online]. Available: http://www.indieboardsandcards.com/resistance.php
2. J. X. Chen, "The evolution of computing: Alphago," *Computing in Science Engineering*, vol. 18, no. 4, pp. 4–7, July 2016.
3. J. F. Ian Millington, *Artificial Intelligence for Games*. Elsevier, 2009, ch. Board Games, p. 667.
4. G. Fedczyszyn, L. Koszalka, and I. Pozniak-Koszalka, *Opponent Modeling in Texas Hold'em Poker*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 182–191. [Online]. Available: https://doi.org/10.1007/978-3-642-34707-8_19
5. F. Southey, M. P. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner, "Bayes' bluff: Opponent modelling in poker," *arXiv preprint arXiv:1207.1411*, 2012.
6. A. Davidson, D. Billings, J. Schaeffer, and D. Szafron, "Improved opponent modeling in poker," in *International Conference on Artificial Intelligence, ICAI00*, 2000, pp. 1467–1473.
7. H. J. van den Herik, H. Donkers, and P. H. Spronck, "Opponent modelling and commercial games," in *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG05)*, 2005, pp. 15–25.
8. F. Schadd, S. Bakkes, and P. Spronck, "Opponent modeling in real-time strategy games." in *GAMEON*, 2007, pp. 61–70.
9. S. Ganzfried and T. Sandholm, "Game theory-based opponent modeling in large imperfect-information games," in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 533–540.
10. D. Taylor, "Investigating approaches to ai for trust-based, multi-agent board games with imperfect information; with don eskridge's "the resistance"," *Discovery, Invention & Application*, 2014. [Online]. Available: https://computing.derby.ac.uk/ojs/index.php/da/article/view/68
11. G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *AIIDE*, 2008.
12. AiGameDev.com Team. (2012) The resistance ai competition. AIGameDev. [Online]. Available: http://files.aigamedev.com/coverage/GAIC12_AiGameDevResistanceCompetition.pdf
13. Machine Learning Group. The University of Waikato. [Online]. Available: https://www.cs.waikato.ac.nz/ml/index.html