A decision tree approach to *The Resistance*

Submitted as part of the first assignment for:

CE811

**Name**: Peter Lloyd Gent

**Date**: 9 November 2018

**Table of Contents**

1. Introduction 3
2. The Resistance 4
3. Background 5
4. Techniques Implemented 6
5. Experimental Study 11
6. Analysis 12
7. Conclusions 13
8. References 14

Appendix 15

# Introduction

This report represents an investigation into the potential of using decision trees for a simple social deduction game called the resistance. The game is set in a dystopian future where spies have been inserted into the organisation called the resistance to sabotage there missions, conversely the resistance fighters seek to finger or point out spies and to ensure that spies do not get a chance to sabotage missions by isolating them from the selection process and by voting against selections that have spies in the mission [1]. The rules for the game are quite complicated and are therefore covered in detail to highlight the challenges of the problem in the first section of the report. The important question for the solution is can it meet or surpass the performance of other bots developed without a decision tree approach, and given more time what would using a different approach such as a neural networks to learn voting and selection behaviours mean in terms of performance.

The second section looks at the program architecture and highlights of source code and algorithms used for the bot to compute behaviours for voting, sabotage and selection of players. The important question here is whether or not an architecture can be created that is easily extensible, loosely coupled and modular to be presented as a solution that can be developed further outside this report study. The results are then analysed and concluded and evaluated in the analysis and conclusion for the report. The important questions in summary for this report is firstly what are the challenges of the game presented to creating a decision tree solution? it will also be necessary to weigh up different approaches to the problem and answer the question is a decision tree is the best approach and what are the benefits and drawbacks of utilising such a solution? Finally the report will conclude upon the answer the most important point of all of whether or not a decision tree approach can match or succeed the performance of rival bots with a decision tree approach.

# The Resistance

The game is set in a futuristic dystopia of corrupt governments ruled by corporations according to Wheaton, who describes the rules of the game as follows [1]. This is the setting for a game primarily about social deduction in which two teams compete: the resistance and the spies that seek to sabotage the resistance fighters missions. When a game starts out all players close their eyes. The spies then open their eyes so that they know who the other spies are and then the resistance fighters open their eyes. This is one of the central cruxes of the game as spies need to work together to maintain their cover as they seek to fulfil their teams goals.

In the game the goals for spies and resistance fighters is different. The resistance fighters need to have three successful missions out of five and the spies need to fail three of these missions to come out on top. At the beginning of each round a new mission leader is selected that selects agents or spies for a mission based upon their affiliation with either group. Each of the players can then vote on whether or not to approve the mission based on the selected players. When the mission is executed spies may vote to sabotage or succeed a mission in order to maintain their cover. The resistance fighters on the other hand must always must vote for a mission to succeed. This raises special challenges for both teams as they vie to fail the other teams missions or permit their missions to succeed.

The challenges in writing artificial intelligence for the game depend greatly on team affiliation. The resistance fighters face a challenge because they cannot know for certain who a spy is unless an entire team votes for a mission to be sabotaged. In order to win the game they must try to finger the spies by taking an educated guess. They can do so in the case that two missions failed with the same player(s) in each mission or take a chance and pick or more out of a failed missions team. The spies on the other hand know who the other spies are and they would do well to work with the spies to get missions with one or more spies on it to fail a mission or feign a mission success to avoid being detected.

# Background

For the given problem a number of approaches to the problem were investigated and researched. This included decision trees, behaviour trees, neural networks. These can be classified into a topology of algorithms that learn their behaviour from a player; and those that do not do so. In this topology a neural network is an example of a learning algorithm and behaviour trees and decision trees are pure decision machines without any training or learning processes. A neural network is defined as consisting of a “large number of of relatively simple nodes, each running the same algorithm. These nodes are artificial neurones, originally intended to simulate the operation of a single brain cell. Each neutron communicates with a subset of the other artificial neutrons in the network” [2]. However, neural networks can be difficult to train and as more data variables become available the complexity of the system increases by many magnitudes.

In the case of learning algorithms it is noted by Millington et al [3] that a common problem with these algorithms is that getting a program to learn from decisions is notoriously difficult to do so in order to outwit a human player. It can also be non trivial to select a feature space of inputs to such a learning decision algorithm [4]. Since time and complexity for the solution were major constraints, learning algorithms (neural networks) approaches have been discarded in favour of a simpler solution, found in decision trees: specifically a decision tree approach with a blackboard architecture [5]. The use of a blackboard is merely as a data structure called a "blackboard” [5]. A blackboard is simply a place to store or share variables between objects. This is often used by expert systems but it can also be used by decision trees or any algorithm that needs to swap or share data between different objects [5].

The decision tree itself is constructed of connected decision points, starting at a root node and flowing towards leaf nodes that make a decision or perform some kind of action [6]. It is noted by Millington et al[6] that although decision trees are the most basic of decision making implementations they prove in practice to be very fast at performing decision making functions and they are easily designed and implemented meaning that they can be easily customised for a particular application. This is important in any implementation of artificial intelligence for games where the fun and playability of a game for a player comes before the accuracy and precision of the bot. The accuracy being how centred results are and the precision being the tightness of a grouping of results. Therefore it’s not always the case that a very good bot would be fun for a player to play against - it may simply be too challenging or hard for the average player to play against and have fun with.

# Techniques Implemented

|  |
| --- |
| ce811diagram-2.jpeg |
| Figure 1. Class diagram for GENTP53602 |

In the class diagram above the structure of the solution is pictured. In the solution any number of CommandObjects are included in a decision tree made up of nodes. This part of the solution used two design patterns: the command pattern and the strategy pattern. The command pattern was used to send to any part of the decision tree a command to be executed that would return a number to tell the decision tree where to navigate to next and a callback to update the blackboard for the given node. The strategy pattern was used for command objects to maximise code reuse and minimise duplicate code from the subclassing of command objects such as IsSpy that makes all command objects implement a hook function executeFunction() to operate on data held in a shared data structure called the blackboard. The blackboard is coupled with each of the commands upon their instantiation when building the tree.

At the centre of the class diagram is the blackboard, as pictured in the appendix [Appendix B]. This is a data structure that is often used with expert systems, rule based systems and finite state machines. In the solution it provides a link between the bot and the blackboard with getter functions and vice versa with set functions for commands in the command tree to update. It’s worth noting that in the diagram not all of the commands could be put in the same picture as they are too numerous to do so, but they are listed here. In the solution there are two commands for each team respectively: sabotage, voting and selecting a team. This maps into commands for spies: isSpy,SpySabotage,SpyVoting,SpySelection; and the resistance equivalents SabotageResistance, ResistanceVoting, and Resistance selection. These are detailed in the following sub sections.

isSpy:

|  |
| --- |
| Screenshot 2018-11-02 at 13.48.18.png |
| Figure 2: isSpy() |

|  |
| --- |
| Screenshot 2018-11-05 at 14.49.26.png |
| Figure 3: onGameRevealed |

In the above code snippet for figure 2 the bot is either directed down the decision tree path for a resistance fighter or for a dastardly spy. It takes information from the blackboard prior to tree construction in OnGameRevealed to setup default values for things like team, suspects and identified spies as well as the bot itself in figure 3. In order to determine if a player is a spy or not it tests the length of spies because only spies have knowledge of whom is a spy and who isn’t a spy.

SabotageSpy and SabotageResistance:

|  |
| --- |
| sabotageTree.png |
| Figure 4. Sabotage |

|  |
| --- |
| Screenshot 2018-11-05 at 14.55.01.png |
| Figure 5: Sabotage spy |

In the figure 4 the sabotage function is called. The decision tree will navigate through the spy side of the tree and execute SabotageSpy in figure 5 to determine whether or not a mission should be sabotaged for a spy. In this case the GENTP53602 bot always sabotages a mission.

|  |
| --- |
| Screenshot 2018-11-05 at 15.11.04.png |
| FIGURE6: sabotageResistance |

When the game queries the sabotage function in GENTP53602 for as a resistance fighter in figure 6. The bot always returns false for the sabotage variable held in the blackboard. The functionality here for both functions is similar to the functionality of the paranoid bot [7].

SpyVoting and ResistanceVoting:

|  |
| --- |
| Screenshot 2018-11-02 at 13.52.16.png |
| Figure 6: call from GENTP53602 bot |

|  |
| --- |
| Screenshot 2018-11-05 at 15.18.29.png |
| Figure 8: SpyVoting |

|  |
| --- |
| Screenshot 2018-11-05 at 15.22.50.png |
| Figure 9: Resistance Voting |

In figure six the hook function is called for voting and then the bot executes a traversal of the decision tree. This happens in order to update the getVoting value and to send the team composition to the blackboard. In the case of spies, a spy will always vote for a team composition if it includes at least one spy (see figure 8). In the case of resistance fighters the command depends on quite a lot of code found in the bot and attached to the appendix of this report due to its verbosity [Appendix A]. In the command pictured in figure 9 the decision tree command essentially looks to see if any spies have been identified and then votes false if a spy has been identified to reject the team selection.

SpySelection and ResistanceSelection:

|  |
| --- |
| paranoid select.png |
| Figure 10. Select for paranoid bot |

|  |
| --- |
| Screenshot 2018-11-03 at 16.24.36.png |
| Figure 11: Use of paranoid bot code in solution for ResistanceSelection and SpySelection |

In the case of select function for resistance fighters and spies the same code is used from the paranoid bot found in the git source repository for the resistance (see figure 2 and 3)

|  |
| --- |
| select.png |
| Figure 12: executing tree search and returning search variable |

In order to set up the select process for the decision tree the blackboard variables for player and count are set on the first call to select. In the same call the decision tree has either a ResistanceSelection or SpySelection node inserted. In practice the decision tree object has a method to insert a node respective of the team affiliation, with a check to make sure that it is only inserted on the first call to the select function found in GENTP53602. The tree then executes a search to update all of the required variables in the blackboard.

|  |
| --- |
| Blank Diagram.jpeg |
| Figure N: The structure of a GENTP53602 decision tree |

The tree structure defined in DecisionTree is effectively a wrapper class for a Node that acts as the root node to the binary decision tree. This makes a composition pattern for nodes defined with other Nodes in the Node class. The Node class itself has some helper functions such as execute which returns a positive or negative number to tell the decision tree which way it should navigate. It also provides helper functions for getting or setting Nodes to the left or right, which corresponds with the DecisionTree’s addYes or add No functions. The decision tree then has nodes attached to it via addRight and addLeft and sub nodes are added to the second tree depth with the Nodes addLeft or addRight. In effect the decision tree takes the form pictured in figure N, with either a SpySelection or ResistanceSelection node being inserted in the mutually exclusive conditions of a GENTP53602 bot being on the side of the resistance or on the side of the government.

The use of a decision tree means that the problem of designing decisions for the bot can be subdivided into decisions for spies and resistance respectively. It can be argued that this gives more prior knowledge to the problem, and the splitting performed in IsSpy means that the problem of resistance and spy behaviours are divided and conquered into more simple decisions that are more manageable to solve.

# Experimental Study

|  |
| --- |
|  |
| Figure n: highest performance results from 100 games of the resistance. |

The success rate of the GENTP53602 bot varies greatly. It is measured overall in terms of the per cent of that time that the bot is on the winning team for spies or for resistance fighters. In a test run out of a 100 the bot was found to be more than 48% effective, coming on top of all other beginner bots. This proves that the bot is more effective than any of the other bots, but not by a clear or substantial margin that would be desirable in such a study.

|  |
| --- |
|  |
| figure n: lowest performance results from 100 games of the resistance. |

Since the bot groups tightly around the 34-50%+ success rate over 200 games the bot could be said to be performing with medium precision but with low accuracy, as in figure . It would be desirable to for the performance of the bot to be 60% or above for instances where the bot plays as the resistance or as a spy. However the code submitted is a first pass at resistance and spy behaviours respectively, and there is plenty of room for improvement, particular with the behaviour of the bot for resistance fighters which from the results appears to be holding the overall success rate of the bot back.

# Analysis

The development of the bot aimed to complete a first pass on the bot over the course of a two week cycle. A first pass of the program was completed but extra time had to be taken on designing and building the tree and blackboard. This meant that that solutions such as a neural network couldn’t be considered to attempt to increase performance of the bot in time. In the future the bot could be developed with a neural networks solution in mind, which could be compared with the submitted solution using decision trees. This hints at the interesting possibility to provide a comparison between neural networks and decision tree solutions in terms of their accuracy for being on the winning team or not.

Further areas for improvement could also be found in implementing functions for announcing and saying a belief that a certain player is a spy to make the bot more interactive with a user that is using irc to play the game with the bot. This could be developed in tandem with improvements to fingering spies by picking a one out of three chance that a given player is a spy. However as aforementioned this would represent a greedy algorithm that could end up selecting too many players at a time and consequently finger resistance fighters as spies.

Fortunately, the architecture of the solution is a sound one to implement new commands such as say or announce. The use of the command and strategy patterns for the solution makes the commands very modular and easily extendable. It is quite possible with these patterns being implemented to create new decision nodes as simply as extending commandObject. This means that with any future development a better solution with a higher accuracy is plausible.

The peak success rate of 48% accuracy is something which could be improved on further without necessarily implementing a neural networks solution. The primary thing holding the bot back from being more accurate is making sure that fingered or identified spies are not selected for a mission if the bot is playing as resistance.

# Conclusions

In the introduction to the report a of questions were posed to the reader. The first question was whether or not a decision tree bot can match or surpass the beginner bots? The second question is whether or not a decision tree is the best choice for a social deduction game and what are the drawbacks and advantages to using a decision tree system? The third question was whether or not an architecture can be created that is easily extensible, loosely coupled and modular to be presented as a solution that can be developed further outside this report study? In conclusion we seek to answer these questions posed in the introduction.

Firstly, the bot cannot be considered a failure because it can perform better than the other bots with repeatability of up to 48% accuracy. It would be preferable to have a much larger margin by 20% for example, for the bot to stand out in terms of decision making as this would represent a bot that wasn’t too hard for a player to play against but not too easy either. This is the goldilocks comfort zone in which the bot may be rather on the easy side. The answer to the second question is that frankly speaking the use of decision trees has neither been a success or a failure, the real areas for development lye in the selection and voting behaviour for the resistance and finding more creative ways to identify spies beyond the intersection of players in two failed teams could be found in choosing a random player to classify as a spy at random. However this risks potential resistance players being misclassified as spies. The solution presented takes the pragmatic approach of just selecting bots that appear in two failed missions to be classified as spies as an educated guess. Further development here is risky but it could prove useful in increasing the win rate for the bot. Thirdly, the architecture created for the bot is a sound one. By using the strategy and command patterns together the tree is easily extensible and in the case of select easily inserted into the decision tree. The use of decision trees has in fact been a beneficial exercise in practicing how to use decision trees as binary trees in the decision process. This has been some of the more complicated code to write and has been a definite time sink that has meant that a neural network solution could not be investigated in time. The use of decision trees has, however, meant that code doesn’t require re writing for every case as the prerequisite node for isSpy for example is visited in each instance of navigating the decision tree. Further development of these nodes would give promising improvements in the bot’s performance within the goldilocks zone.

# References

[1] W. Wheaton, “The Resistance: Felicia Day, Allison Scagliotti, Ashley Clements, and Amy Okuda on TableTop SE2E02,” *YouTube*, 18-Apr-2013. [Online]. Available: https://www.youtube.com/watch?v=g\_QRczGzXqw. [Accessed: 02-Nov-2018].

[2] I. Millington and J. Funge, *Artificial intelligence for games*. Burlington: Elsevier, 2009, pp. 646-652.

[3] I. Millington and J. Funge, *Artificial intelligence for games*. Burlington: Elsevier, 2009, pp. 606.

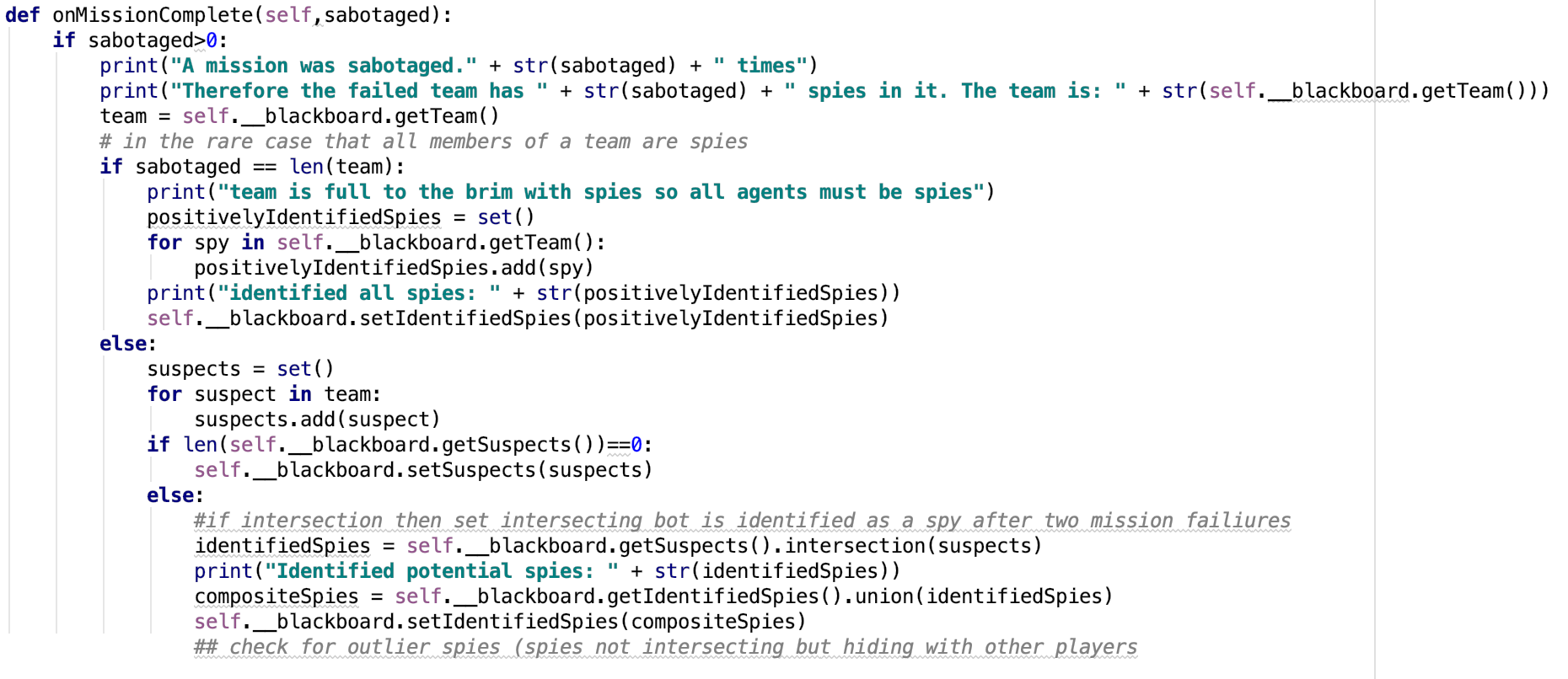
[4] I. Millington and J. Funge, *Artificial intelligence for games*. Burlington: Elsevier, 2009, pp. 607.

[5] I. Millington and J. Funge, *Artificial intelligence for games*. Burlington: Elsevier, 2009, pp. 459-465

[6] I. Millington and J. Funge, *Artificial intelligence for games*. Burlington: Elsevier, 2009, pp. 295-300

[7]J. Walto, "bots/beginners.py · master · ce811 2018 / Assignment 1", *GitLab*, 2018. [Online]. Available: https://cseegit.essex.ac.uk/ce811/assignment1/blob/master/bots/beginners.py. [Accessed: 02- Nov- 2018].

# Appendix

A.

B.