CITS3001 Project
2022

Name	StudentID	Degree
Aidan Smith	22967113	Computer science and physics
Name	**StudentID**	**Degree**
Khalin Ganeson-Hands	22736252	Computer science

Table of Contents

Introduction

Explain your understanding of the game in one paragraph.

This game is a representation of the information environment with a particular country. The population of the country are the green team members. Since the green government has

little control over the information environment over their country, they are subject to the whims of external powers. In particular, the red and the blue team are the major geopolitical players. The blue team spreads messages in order to convince the green population to vote, and the red team spreads messages in order to convince the green population not to vote. Each team can choose the strength (potency) of their message. Spreading a more potent message allow the message to convince more members of the green population. However, the more potent blue's message is, the more energy they lose, and the more potent red's message is, the more followers they lose. When blue runs out of energy, he can no longer send messages, and when red loses followers, they can no longer spread messages to those people. On blue's move, instead of spreading a message, he can release a foreign agent (a grey agent) into the population. This will spread a message for blue without the blue agent losing energy. However, there is a chance that this grey agent will be a spy. In this case, the grey agent will spread a message for red without the red agent losing followers. At the end of the game, if a majority of green team members want to vote, then blue wins, and if a majority do not want to vote, then red wins.

Please note that throughout the report, you are allowed to add screenshots, graphs, equations and code snippets.

Assumptions

State all your assumptions, including but not limited to:

1. **What is the interval of uncertainty in your project? What do-1,0,+1 represent?**
2. **How are you perceiving green nodes' opinion? Do you perceive it as vote/not vote in election, or are you perceiving it as vote for blue/vote for red?**
3. **Any other assumptions**

Each green node has two values associated with it: an opinion which is Boolean, and an uncertainty which is a real number.

The uncertainty interval in our project is [-1, 1]. -1 represents minimum uncertainty (i.e. maximum certainty) and 1 represents maximum uncertainty (i.e. minimum certainty).

We are considering a green node's opinion being 'true' to mean that they plan to vote in the upcoming election, and we are considering a green node's opinion being 'false to mean that they plan *not* to vote in the upcoming election.

We are assuming that the game ends when blue has run out of energy, and when red's number of followers has been reduced to below 10% of its initial value.

We are assuming that they players know how many grey agents there are in total, but do *not* how many of them are spies.

Selection and design of appropriate AI technology

Methodology

1. **Describe and justify your methodology for this project, including**
    a. **which parameters are hard coded,**
    b. **which parameters are to be input at the start of the game and**
    c. **what type of methods you used to make your agents intelligent.**

We decided to allow as many parameters to be input at the start of the game as possible, rather than hardcoding them. This allowed our game to be very flexible. At the start of the game, we ask the user to provide the minimum initial uncertainty of the green agents, the maximum initial uncertainty of the green agents, the total number of green agents, the percentage of green team members that initially want to vote, the number of good grey agents, the number of bad grey agents, the edges of the graph, and whether each of the red and blue agents is a human or an AI. We also give the option for the user to use files to specify some of these parameters. We also give the option to generate a graph by specifying the proportion of possible edges that should exist in the graph (such a graph is then generated randomly).

The types of methods we used to make our agents intelligent are briefly summarised below. Implementation details, specifically about the code, are left for later sections of this document, as required.

To make our agents intelligent, we used a number of different artificial intelligence techniques that we learnt from this unit, or from related material. We used Bayesian inference, training games with minimax, and probabilistic decision trees.

We used Bayesian inference to determine whether or not a grey agent should be released into the network in a given situation. Using the general structure to the Bayesian formula $P(A|B) = \frac{P(A)*P(B|A)}{P(B)}$, we computed the probability that the blue agent would release a grey agent.

Our prior probability $P(A)$ is the initial probability of releasing a grey agent, which we hardcoded at 0.15 since releasing a grey agent is quite a big risk. The posterior probability $P(B|A)$ is the probability that we should release the agent after evidence has been observed. We effectively treated this as the proportion of grey agents that have been observed to produce a positive outcome in the past. The marginal likelihood $P(B)$ is just the proportion of green agents that want to vote.

This is an effective formula for several reasons. Firstly, when the vast majority of green agents do not want to vote, the result is very high. This makes sense, since when we seem to be losing, it makes sense to play a high-risk, high-reward strategy such as introducing a grey agent. Also, you can see from this formula that when grey nodes have produced a positive result in the past, the resulting probability increases. In this way, our agent is learning that there may be only a small proportion of spies, and is adjusting accordingly.

For choosing the potency of message to spread, we used quite a complex scheme:

We made a blue AI play many training games against a red AI in order for them to learn about what moves tend to be optimal in each specific situation. In the initial training games, we pretend that the agents have full knowledge of the game, and allow them to use a minimax algorithm with a basic initial evaluation function to determine what they think are good moves. Since these training games involve full knowledge of the game state, we record what moves were optimal in each game state.

We then group together the game states that are similar (i.e. have similar parameters). Now for each group of similar game states, we have recorded the optimal move that was played in each of them. We take an average of those moves (potencies) in similar game states, and use that to estimate the most accurate move in that particular 'bucket' of similar game states. We create a decision tree from these buckets of game states and their associated chosen moves (each leaf of the decision tree corresponds to a bucket of similar game states, and their associated chosen move).

However, when the agents are actually playing the real game, they of course do *not* have full information. So the agents maintain a probability distribution of their unknown parameters, which is updated when information comes in. We then use these probabilities to navigate the decision tree. We weight each potency in the decision tree by the agent's estimated probability that the true game state corresponds to that game state. This probability-weighting allows us to select what the agent believes to be the optimal potency.

The implementation of the agents is explained later in this document.

Game Play

1. **Explain in detail how the game is played?**
2. **How turns are organised?**
3. **How opinions and uncertainties are updated?**
4. **Etc.**

See all the assumptions that we have made about the game (in the section 'Assumptions' above).

The blue agent has 10 different potencies from which to choose, and there is a known fixed energy loss for each of these potencies. The red agent also has 10 different potencies from which to choose, but there is a known fixed *proportion* of followers lost for each of these potencies.

The game begins with blue's turn, then red's turn, and then a green-interaction round. These three rounds then repeat until the game ends, which is when the blue agent has run out of energy and the red agent has fewer than 10% of their initial followers.

In an interaction between two nodes a, b their opinions and uncertainties are updated as follows:

Let a be the node with higher uncertainty, and let b be the node with lower uncertainty.

If a and b have the same opinion, then only node a's uncertainty changes. In particular, $a.uncertainty$ decreases by $\frac{a.uncertainty - b.uncertainy}{2}$.

If node a and node b have a different opinion, then there is a probability that node a will change their opinion (recall that node a is the one with a higher uncertainty), but no probability that node b will change their opinion. In particular, the probability that a will change their opinion is:

$P(a\ changing\ opinion) = \max(a.uncertainty - b.uncertainty, 0.9)$.
If a does indeed change their opinion, then:

$a.uncertainty = 1 - (a.uncertainty - b.uncertainty)$
$b.uncertainty$ remains unchanged.

However, if a does *not* change their opinion, then:

$a.uncertainty$ increases by $\frac{1 - b.uncertainty}{2}$.
$b.uncertainty$ increases by $\frac{1 - a.uncertainty}{2}$.

These uncertainties are prevented from ever increasing above 1.

Implementation of the Agents

1. **State main points about the implementation of agents. This heading is more focused on the code and how you made it efficient.**
2. **How long the program takes to run a single turn with variable number of green agents. Report for both small and large number of green agents.**
3. **Which programming language you used? Whether you followed an Object Oriented approach or not.**
4. **Which libraries you have used?**
5. **Etc.**

The Bayesian inference section is described in detail in the 'Methodology' section. The training games with minimax, and the probabilistic decision trees had complex implementations, and we will now give more detail regarding their implementations.

The function Training.trainOnGames() simulates games for the agents to play against each other for training. At each stage in the game, the function Training.run() is used to decide on what move to play. This function runs a minimax algorithm with a simple initial evaluation function called Training.initEval(). The depth used in each of these training games depends on the number of nodes in the network, since games take longer to analyse if there are more nodes in the network. Data from these training games is stored in BlueAgent.learningData and RedAgent.learningData, which are hash maps from game states to chosen moves.

The function Training.fillDecisionTreeFromLearningData() is then used convert each agent's learningData (a hash map) to a fully formed decision tree. These are called BlueAgent.decisionTree and RedAgent.decisionTree. The decision trees are built by grouping together similar game states. That is, we categorise the game states using ranges of parameters. For example, depending on each parameter's bucket-size, a single game state may fall into the collection:
10% followers remaining – 20% followers remaining
40% energy remaining - 60% energy remaining
30% green members want to vote - 40% green members want to vote.

In the function Training.fillDecisionTreeFromLearningData(), a decision tree is built such that it has a node for each group of parameters. We navigate the tree based on each game state's parameters, and update the average move stored at that node (i.e. that set of similar game states). The 'move' here is the potency, so we store an average potency at each node.

In the real game, the agents do not have full knowledge of the game state, so they maintain a probability distribution of their unknown parameters. For example, the blue node maintains a probability distribution for red's proportion of followers remaining. Updating this distribution, for example, is done from line 246 to line 291 of BlueAgent.java (these lines are in the function BlueAgent.makeAIMove()). These probability distributions are updated each turn.

Also on each turn, the agents navigate the decision tree using their probability functions for the parameters. That is, as they navigate the decision tree, they don't just follow one path through the tree, since they don't know the true game state. Instead, they weight their steps through the tree based on the probability that the game is in that state. This probabilistic weighting allows an average move (i.e. potency) to be chosen for that turn.

We observed that when there were more nodes in the network, choosing a move in the training games took longer. Because of this, we set the depth used in the training games to be lower when there are more nodes in the network, so that we can get a good balance of quantity of sample data to quality of sample data. We did this in such a way that the training games should never take longer than 15 seconds.

Although the training games take quite a long time, each turn from the blue agents and red agents is completed in a negligible amount of time, regardless of the number of nodes in the network. We made this efficient by building the decision tree in advance. The small depth of

the decision tree means that navigating the tree based (weighting by the probabilities of game states) is quick and efficient once the game has started and actual turns are occurring.

We used the programming language 'Java'. We did follow an object-oriented approach, creating separate classes for the agents, nodes, game states, etc. The class 'App' is used to run the program.

We used the external library 'GraphStream', with the 'swing' user-interface, to visualise the graph at any stage of the game. Thus, the dependencies are gs-core-2.0.jar and gs-ui-swing-2.0.jar.

Running the game

How can a layman run your game? Provide the commands, and associated parameters needed with an example workflow of the game.

To run the game, first navigate in the command-line to the directory 'CITS3001Project'. Ensure that you have the two dependencies (external libraries) mentioned above in the folder 'dir'. The command:

```
javac -cp ".;/dir/gs-core-2.0.jar;/dir/gs-ui-swing-2.0.jar"
App.java
```

can then be used to run the game.

While running the game, prompts will indicate what information is required to be entered by the user. An example of running the game, with specified parameters, is provided here:

```
Enter the minimum initial uncertainty of each green team
member (give a value from -1 to 1)
-.5

Enter the maximum initial uncertainty of each green team
member (give a value from -1 to 1)
.5

Enter 'p' to generate the node ids and teams from input
parameters.
Enter 'f' to use the node ids and teams specified in an input
file.
p

Enter the number of green team members (maximum 1000).
[RECOMMENDED: 200]
```

```
200

Enter the percentage of green team members that initially want
to vote (from 0 to 100).
50

Enter the number of good grey agents.
5

Enter the number of bad grey agents (i.e. spies).
5

Enter 'p' to generate the graph from input parameters.
Enter 'f' to use the graph specified in an input file.
p

Enter the percentage of possible edges that should be a real
edge (from 0 to 100) [For the design of this game, PLEASE USE
NO MORE THAN ABOUT 5% ON REASONABLY LARGE NETWORKS. Don't make
the network dense at all, as this is super unrealistic in a
real country, and will cause the green team members to
consolidate their opinions amongst themselves every move].
5

Enter 'h' to make the blue player a human.
Enter 'a' to make the blue player an AI.
a

Enter 'h' to make the red player a human.
Enter 'a' to make the red player an AI.
a

Training agents and building probabilistic decision trees...
This should be done within about 15 seconds.


*** Blue agent's turn ***
Blue AI is sending a message with potency 5
118 members of the green team DO want to vote.
82 members of the green team DO NOT want to vote.

*** Red agent's turn ***
Red AI is sending a message with potency 8
64 members of the green team DO want to vote.
136 members of the green team DO NOT want to vote.

Green nodes interacting with each other...
```

(and the game would continue).

Agent Design

This heading focuses on the architectural design of the agents in the game

e.g.,

Green Agents

1. **Are you using a static network or a dynamic network?**
2. **Can we generate a network when the game start?**
3. **What type of underlying network model you are using?**
4. **Other properties of the network, e.g., is it weighed, can links be added or removed during the play?**
5. **Etc.**

We are using a static network of green agents. We allow you to generate the network, by specifying the proportion of the possible edges that exist. A random algorithm will then generate the graph accordingly. Alternatively, a network can be specified via an input file. The prompts that appear when running the game explain how to input the graph using either of these methods.

The underlying network is an undirected, unweighted graph of nodes connected by edges. Links cannot be added or removed during the play (but of course, the red agent can lose followers).

Red and blue agents

1. **Describe your design of the message potency (a.k.a. uncertainty of red and blue nodes.)**
2. **Describe your method for changing the followers' number in case of red agent**
3. **Describe your method for changing the energy level of blue nodes (a.k.a. lifeline)**
4. **Other Pertinent points regarding the working of your agent**

The blue agent has 10 possible potencies, numbered from 1 to 10. Their respective corresponding uncertainties are:
0.65, 0.5, 0.35, 0.2, 0.05, -0.1, -0.25, -0.4, -0.55, -0.7
And the respective amount of energy lost from each is:
5, 10, 15, 20, 25, 30, 35, 40, 45, 50

The red agent also has 10 possible potencies, numbered from 1 to 10. Their respective corresponding uncertainties are:
0.25, 0.1, -0.05, -0.2, -0.35, -0.5, -0.65, -0.7, -0.85, -1.0
And the respective *proportion* of followers lost from each is:
0.04, 0.08, 0.12, 0.16, 0.20, 0.24, 0.28, 0.32, 0.36, 0.40

Hence, the potency that a blue or red agent chooses determines the uncertainty that it will have when it interacts with green nodes.

In particular, the followers that the red agent loses will be the followers that are *least* favourable to red. Importantly, this means that the red agent will consolidate a strong support base, while becoming completely unable to reach some of the people who don't already agree with them.

Grey Agents

Describe the working of the grey agents.

The grey agents interact in a similar way to how to blue or the red agent interacts with members of the green team. However, the grey agent, like the blue agent is always able to interact with all green nodes. Also, its uncertainty is always -0.5.

When a grey agent is released into the network, it is randomly chosen from the set of grey changes that have not yet been released. This determines whether or not it is a spy for the red team.

Validation of Agents

1. **Report any tests you conducted to ensure the agents are doing the task they have been asked to do.**

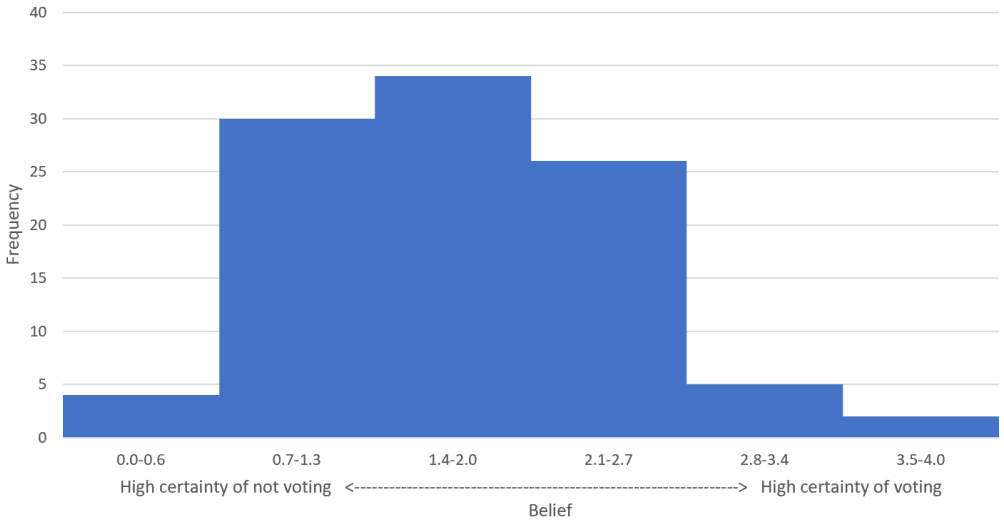
We tested our agents extensively during development and after development. By running simulations, we confirmed that when a very small proportion of green nodes want to vote, the blue agent is more likely to release a grey agent into the network, as this high-risk, high-reward strategy makes sense when it seems as though all hope is lost. We also tested to make sure that the blue agent is more likely to release a grey agent into the network if previously released grey agents were not spies.

Perform Various Simulations for the following set of questions.

2. **How does the game change if you have a tight uncertainty interval at the beginning of the game?**

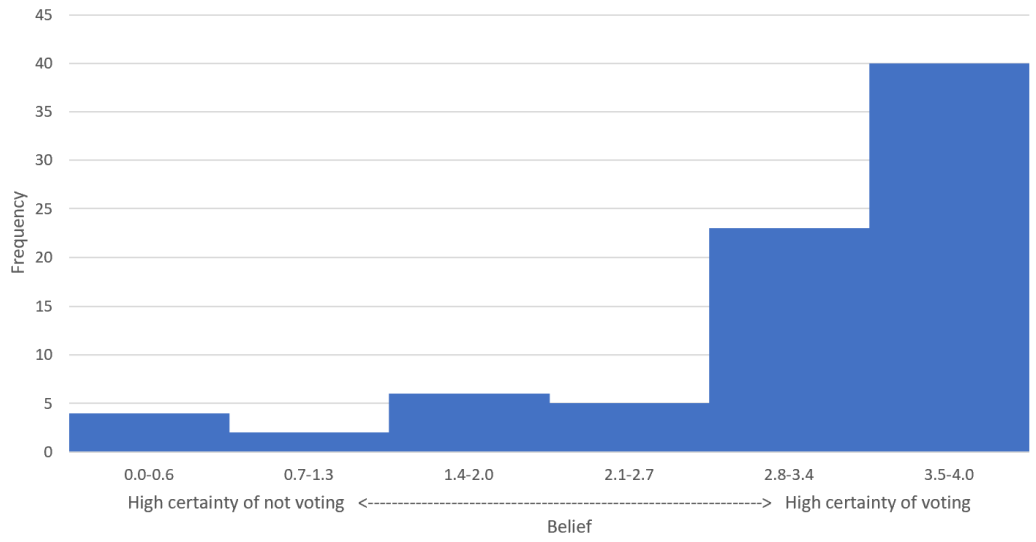
   When there is a tight uncertainty interval at the beginning of the game (e.g. -0.1 to 0.1), the opinions of the agents remain split from around 30% wanting to vote - 70% wanting to vote. They then sway gently from player to player at the start of the game. To be more specific, a typical simulation may have 50% wanting to vote, then 45%, then 53%, then 46%, etc. with subsequent turns. However, over this time period, each agent decreases the uncertainty of their voter base, making it harder to convert over time.

   In this case, the agents tend to prefer smaller uncertainties at the beginning, increasing only when the uncertainties of green agents become extreme later on.



3. **How does the game change if you have a broad uncertainty interval at the beginning of the game?**

When there is a broad uncertainty interval at the beginning of the game, the agents typically start out with more potent messages (a potency of around 7) (this varies from game to game, but over many games, this trend can be seen). The opinions of the green nodes sway wildly from turn to turn, until either someone's energy or follower count gets too low, or the green population gets almost stuck at over 95% or below 5%, as one agent has managed to pull them over and consolidate. If no consolidation occurs (which only occurs occasionally with a broad starting interval), then the agents decrease the potency of their messaging and the game concludes with a near 50%-50% split.



4. **Plot distribution of uncertainties for each of the above questions.**

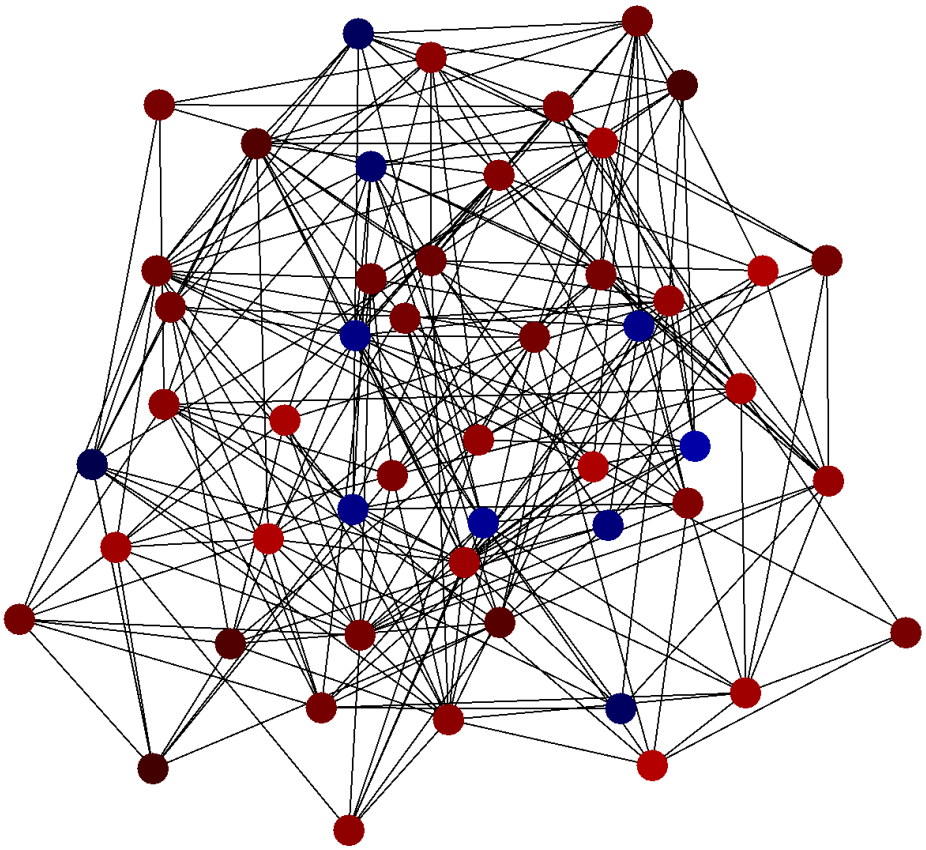
Done above.

5. **In order for the Red agent to win (i.e., a higher number of green agents with the opinion "not vote", and an uncertainty less than 0 (which means they are pretty certain about their choice)), what is the best strategy?**
    a. **Discuss and show with simulation results how many rounds the red agent needs in order to win.**

Simulations show that the best strategy for the red agent is usually to start with a small potency. With 200 nodes and some standard parameters, the win rate of the red agent was 29% when it started with a small potency, but 63% when it started with a potency below 5. Acting too early means that the red agent will gain followers very early, but then the blue agent has plenty of time to recover and take back control of the game.

However, the red agent quickly increased its potency after the start of the game. The simulations show that the best strategy for red to win is to use very highly potent messaging *not* at the start of the game, but shortly after the start of the game, while you still have a lot of followers. This means that in the long run, the red agent can consolidate more followers.

Red then wins if the game drags out towards the end, since red has a very strong voter base (their followers) that they can affect will reasonably high potency towards the end of the game, while at the end of the game, blue has little energy and can't do much to change nodes' opinions. Games that red wins take an average of 8.9 turns each.



This image shows a red agent on turn 12 that has waited-out its opponent, who now has low energy.

6. **In order for the Blue agent to win (i.e., a higher number of green agents with an opinion "vote", and an uncertainty less than 0 (which means they are pretty certain about their choice)), what is the best strategy? [**
    a. **Discuss and show with simulation results how many rounds the blue agent needs in order to win.**
    b. **What impact did grey agents have on the simulation? Explain how did you test the impact of grey agents?**

Please note that for answering questions use your own mental model of how you implemented uncertainties if they are different from the specs

Simulations (with 200 nodes) show that the blue agent wins by spending a moderate amount of energy at the start of the game, but then a very high amount of energy when it is running out of energy. When the agent has below 24% of its energy remaining, it typically spends it within one to two turns. When the blue agent spread out its spending towards the end of the game, the game lasts longer, but the blue agent slowly loses support amongst the green population. Hence, the blue agent tends to win shorter games. Games that the blue agent wins take an average of 5.8 turns each.

When grey agents are introduced, blue actually gained a slight advantage. By running simulations with 200 nodes, we observed that blue's win-rate increased to 66% from 57%. We observed that the average proportion of the green nodes who wanted to vote at the

moment that a grey agent was released was 26%. This shows that the blue agent favours releasing grey nodes when it's chances of winning seem low. Since blue can control when the grey agents are released, it makes sense that the introduction of grey agents would benefit the blue agent.

Also, the blue agent was more likely to release grey nodes if previously released grey nodes had not been spies.

Performance of Agent when playing with a human

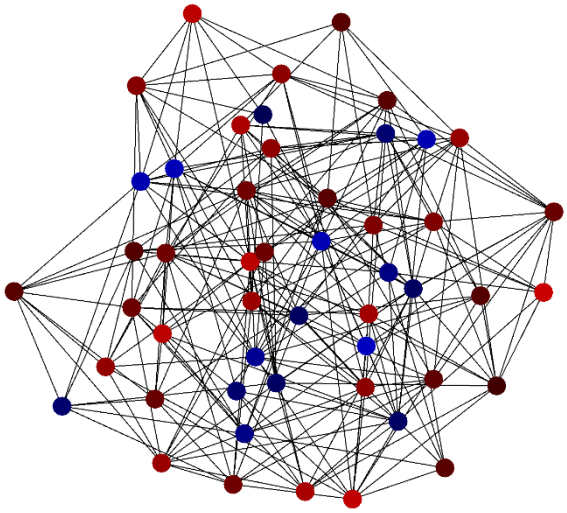
Does the agent run and performs at an excellent level with challenging play when the opponent is a human? Discuss your findings

When playing with a human with little experience playing with the agent, the agents win the vast majority of the time. However, when the human has a lot of experience with how the agent plays (e.g. the human knows when the agent is saving energy to release it in one go), the human is able to outsmart the agent about 30% - 40% of the time. So although the agents provide great insight into the game, especially when they play each other thousands of times, they cannot consistently beat humans who are experienced playing against them. However, it is unclear how much of this is due to all the inherent randomness of the game.

Visualisation

Describe your visualisation methods with some screenshots

We visualise the network as a graph of nodes and edges. Colours indicate which team the green team member supports. Brighter colours indicate less uncertainty. Edges indicate possible green-to-green interactions.



We also allow all the opinions and uncertainties of the green team member to be displayed at any point in the game.