**CITS3001 Project**

**2022**

|  |  |  |
| --- | --- | --- |
| **Name** | **StudentID** | **Degree** |
| Cormac Sharkey | 22983427 | Computer Science and Data Science Double major |
| **Name** | **StudentID** | **Degree** |
|  |  |  |

Table of Contents

[Introduction 1](#_Toc115329643)

[Assumptions 2](#_Toc115329644)

[Selection and design of appropriate AI technology 2](#_Toc115329645)

[Game Play 2](#_Toc115329646)

[Implementation of the Agents 2](#_Toc115329647)

[Agent Design 2](#_Toc115329648)

[Green Agents 2](#_Toc115329649)

[Red and blue agents 3](#_Toc115329650)

[Grey Agents 3](#_Toc115329651)

[Validation of Agents 3](#_Toc115329652)

[Performance of Agent when playing with a human 3](#_Toc115329653)

[Visualisation 4](#_Toc115329654)

# Introduction

Explain your understanding of the game in one paragraph.

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

* There exists a population of green nodes, each with its own opinion and uncertainty associated with it. Additionally, there exists two governments, red, and blue, as well as unknown grey nodes. Both governments want to manipulate the population so that the majority of them are aligned with a certain opinion (for blue, voters, and for red, non-voters). The grey nodes are inactive unless blue activates one on their turn.
* To model this, the green, red, blue, and grey (if any are active) take turns, in that order.
* On green’s turn, each node will talk to each of their neighbors depending on a probability based on the edge weight between the two nodes. The node with the lower uncertainty will convince the other node to change it’s mind and swap over to their opinion.
* On red’s turn, they may send out a message of 5 levels of potency. High potencies affect more nodes, but will raise uncertainty for affected nodes, and high potency messages can even backfire and cause some nodes to swap back to the other opinion.
* On blue’s turn, they can also choose between 5 levels of potency, but it will cost energy based on the level used. However, they do not have to worry about raising uncertainties or backfiring. Blue can also choose to activate a grey node, of unknown alignment.
* On grey’s turn, if a node is active, it will broadcast one max potency message of its alignment, with none of the downsides.

# 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

The interval is set by the user upon launching the game. The default used is (0,100), with lower values representing lower uncertainty.

Green node’s opinions are either 0, or 1, which represent not voting and voting respectively.

List of game mechanics which are assumed:

* Grey will only be active for one turn before deactivating themselves.
* Green nodes of the same opinion will lower each others uncertainty when talking to each other.
* The game is bounded by an amount of rounds (specified by user, default 10)
* Nodes which are swayed by the Blue government will have their uncertainty lowered.
* A green node convinced by another node will take the uncertainty halfway between their own uncertainty and the other nodes uncertainty.

# Selection and design of appropriate AI technology

## Methodology

1. Describe and justify your methodology for this project, including
   1. which parameters are hard coded,
      1. My methodology here is to leave as many parameters open to the user to tweak, with sensible defaults for all of them. The hardcoded parameters usually represent internal game mechanics, such as how likely it is for a red message to backfire, the amount of uncertainty blue can reduce when it sways a node, and how much nodes of the same opinion can reassure each other.
   2. which parameters are to be input at the start of the game and
      1. Parameters which can be tweaked at the start of the game:
         1. N: Number of nodes in the graph
         2. P: Probability of an edge between any two nodes when the graph is generated (we are using erdos-renyi for generation)
         3. num\_grey: Number of grey agents
         4. percent\_grey\_bad: What percentage of them are red-aligned?
         5. uncertainty\_interval: An interval representing the min and max uncertainty.
         6. percent\_green\_voters: What percentage of green nodes are already voters?
         7. red\_is\_human: Is red human controlled? You can also insert various strings here for different AI’s.
      2. Depending on what value you select for red\_is\_human, the user will have to input some parameters for the more advanced AI’s.
   3. what type of methods you used to make your agents intelligent.
      1. The human player takes control of Blue.
      2. Green nodes take into account a lot of environmental information and affect each other very dynamically. More information on this can be found in other sections of this document such as the game play section.
      3. Red has a total of 5 different control schemes:
         1. Human: Human controlled
         2. Random: Sends a random potency every turn

The next two aim to win by counteracting Blue’s moves in different ways, without overusing high-potency messages (and therefore making it easy for Blue to reclaim nodes with low potency messages)

* + - 1. Percentage: Uses a heuristic of percentage total nodes which are voters, and tries to counteract that with an appropriate potency message.
      2. Smart: Attempts to infer what potency Blue just used by comparing the distribution of voters and non-voters from the previous round and the current one. It then tries to match it on their turn. If it cannot infer this (perhaps a grey node was released) it will fall back to Percentage for that particular turn.
      3. Learning: This AI is a scaled down version of a reinforcement learning algorithm. It attempts to learn from the player as the two play, and tries to maximise the amount of non-voters after the players next move. If this AI is used, the user has to input some additional parameters:
         1. Amount of bins: Each bin represents a fraction of 100, in which the AI will learn and store results in. Two bins would mean the agent only learns moves for voter percentages below and above 50%, whereas 10 or 15 will provide much finer results.
         2. Gamma decay: The agent will have no information at the start of the game. To start learning, it will try a random move and store it in the table, if gamma is high enough. Gamma decay affects how fast gamma is reduced as the AI learns moves, and therefore how quickly the AI tries to exploit its knowledge. A high value will cause the AI to adapt faster, but with more incomplete knowledge, whereas a lower value causes the AI to adapt slower, but build up a far more precise knowledge on how to act.
         3. Gamma minimum: This is the minimum amount of gamma the agent will have. In other words, this amount represents the minimum chance for the AI to try learning from a random move. A high value will cause the AI to learn very often, even when it may have good knowledge of the state. A low value will cause the AI to explore less often, but could result in the AI’s knowledge getting “stale” once gamma has decayed.

In general, this AI is best used with a very high amount of rounds in a game in order to let it adapt to the player. The parameters could be tweaked to get it to show adaptation in a smaller game, but it’s behaviour would not be as refined.

## 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.

1. The game is played by inputting starting parameters (or just using the defaults) and then inputting the potency of message you want to send (or to release a grey agent, should you be blue). The graph will update automatically to reflect each turn. A good player must reflect on the state of the graph to make good decisions: Blue needs to manage their resources very carefully in order to make it to the end of the game without running out of energy. Red needs to make this as difficult as possible, by trying to counteract Blue’s actions without overusing high potency messages (as they make it easy for the game to swing back into Blue’s favor even if they use a very small amount of energy). Additionally, a player needs to consider the placement of their nodes on the graph. A very certain node, with a good amount of connections of good weight is worth trying to flip, as on green’s turn it will convince its neighbors. For Blue, that means it could be worth using more energy. For Red, it is important not to lose well positioned red nodes by using high potency messages too often. Players need to pay good attention to disconnected clusters, as they can become very difficult to flip if left unchecked. Finally, Blue can get great benefit out of activating grey nodes as they allow for high potency messages without losing energy – which can put Red on the back foot. However, activating a Red node can be devastating to Blue’s position, as it will be hard to retake flipped nodes which have not had their uncertainties raised as Red typically does.

2. Turns are organised in the order of Green, Red, Blue, Grey. This order prevents Red sending out a max potency message at the end and winning without giving Blue a chance to counteract. Grey goes last as it can act as a lifeline towards the end of the game for Blue. Green goes first so that the two governments can act on an updated population.

3. Opinions of green nodes talking will result in the node with higher uncertainty swapping to the opinion of the other node. For government interaction, the uncertainty threshold in which nodes above it will swap uncertainty is calculated as follows:

***( minUncertainty - maxUncertainty ) \* (potency / maxPotency) + maxUncertainty***

In layman’s terms, the potency selected is represented as a percentage of the maximum percentage, and then that percentage of the uncertainty interval is affected.

* Nodes which are flipped by Blue have their uncertainty halved.
* Voting nodes affected by Red have their uncertainties raised by 75% of the potency percentage.
* Non-voting nodes affected by red have a scaling (backfireChance = potency percentage \* 0.25) chance up to 25% of a backfire.

# 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.
   1. Each agent is a class. Each agent will only perform the calculations necessary to figure out what it wants to do, and set the action in motion. The graph, visualisation, human handling, turn scheduling, and node enumeration are all handled by different classes. (Visualisation class handles plotting and GUI. Model class handles graph creation and node enumeration, as well as activating agents on their turn. A communications class handles communication with all the nodes and recalculating their uncertainties)
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.
   1. Tests performed with these params:
      1. self.P = 0.0330
      2. self.num\_grey = 2
      3. self.percent\_grey\_bad = 0.50
      4. self.uncertainty\_interval = (0,100)
      5. self.percent\_green\_voters = 0.75
      6. Both agents will send a message of random potency.
   2. 7 nodes:
      1. model initiation: 0.234375 seconds
      2. green execution time: 0.09375 seconds
      3. red execution time: 0.078125 seconds
      4. blue execution time: 0.0625 seconds
      5. grey execution time: 0.078125 seconds
   3. 12 nodes:
      1. model initiation: 0.234375 seconds
      2. green execution time: 0.109375 seconds
      3. red execution time: 0.09375 seconds
      4. blue execution time: 0.078125 seconds
      5. grey execution time: 0.09375 seconds
   4. 50 nodes:
      1. model initiation: 0.28125 seconds
      2. green execution time: 0.109375 seconds
      3. red execution time: 0.109375 seconds
      4. blue execution time: 0.125 seconds
      5. grey execution time: 0.125 seconds
   5. 250 nodes:
      1. model initiation: 1.21875 seconds
      2. green execution time: 0.75 seconds
      3. red execution time: 0.75 seconds
      4. blue execution time: 0.75 seconds
      5. grey execution time: 0.75 seconds
   6. 1000 nodes!
      1. model initiation: 18.03125 seconds
      2. green execution time: 10.25 seconds
      3. red execution time: 10.359375 seconds
      4. blue execution time: 10.140625 seconds
      5. grey execution time: 10.453125 seconds
      6. It’s worth noting that the interactive graph is quite painful to use at this node count, so I would rule this as unplayable.
3. Which programming language you used? Whether you followed an Object Oriented approach or not.
   1. I used Python with an object oriented approach.
      1. PopulationModel
         1. Instantiates and maintains the graph and the agents. Calls methods on each agent when requested. Handles enumeration of different types of agents for other classes.
      2. GreenAgent, RedAgent, BlueAgent, GreyAgent
         1. The four agent classes. Handles intake of information from PopulationModel and calculates the next move based on its current parameters. RedAgent is of particular note, housing its 4 different AI’s.
      3. PopulationModelVis
         1. Launches the interactive visualisation, handles user input, plots the graph which is taken from PopulationModel.
         2. In general, takes user input and processes it, before feeding it back into PopulationModel (which feeds it into required agents etc.)
      4. Communications
         1. This isn’t a class, but a utility library I wrote. Agents can interact with it to push messages to the population. Communications will then ask PopulationModel and each node for any required information and update nodes based on the formulas listed above.
4. Which libraries you have used?
   1. Mesa
      1. An ABM framework which houses the internal logic for calling agents and exposes methods for different agent types, passing of information, creation of arbitrary amounts of agents, and maintaining them.
   2. NetworkX
      1. Combined with Mesa, this houses the actual graph in which you can place Mesa agents into. Also stores edge weights, handles creation of the graph via Erdos Renyi, and handles node positioning via a variety of layouts (we use a spring layout, but it can easily be changed to other layouts such as circular, shell, or Kamada Kawai.)
   3. Matplotlib
      1. Handles plotting of the graph, as well as GUI and the interactive viewer.
   4. Time
      1. Handles benchmarking as above.
   5. Random
      1. Handles certain AI functions as well as edge probabilities.
5. Etc.

## Running the game

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

* Running the game is easy. Simply run the PopulationModelVis.py file, and it will guide you through setup and launch the game for you. The user will be prompted to change parameters, but I have added sensible defaults, so if the user just wants to play, they can just use them. Once the game is launched, users can interact with the GUI, and input values into the console.

# 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?
   1. I am using a static network generated at runtime based on given parameters. It has weighted edges and each node houses an agent.
2. Can we generate a network when the game start?
   1. Yes. A network can be generated using default settings or parameters given by the user.
3. What type of underlying network model you are using?
   1. I am using networkX’s implementation of a random Erdos Renyi graph with given amount of nodes and edge probabilities. I then add a random weight to each edge from 1 to 10.
4. Other properties of the network, e.g., is it weighed, can links be added or removed during the play?
   1. The graph is weighted. I think of the edge weight as a representation of the environment between two nodes: a low probability to interact could represent two nodes which are geographically far apart, who have bad internet connections, or even don’t speak each other’s language very well. Conversely, neighbors who are highly likely to interact could represent good friends, family members, or coworkers, depending on how likely the interaction is.
5. Green nodes of the same opinion will lower each others uncertainty when talking to each other.
   1. This aims to model an echo chamber, when all you hear is the same opinion from your neighbors, you are going to slowly become more and more certain of that view. This also means that smaller, disconnected clusters of nodes become important to keep track of, as they will echo an opinion around until they are all incredibly certain of it. A real life analogue of this could be extremist online communities, or, closer to home: https://en.wikipedia.org/wiki/Perth\_Prohibited\_Area
6. Green nodes which are swayed by the Blue government will have their uncertainty lowered.
   1. This can be thought of as Blue using official channels which the population is already familiar with (state TV, government material, government speeches), whereas Red uses more underhanded techniques, which don’t cost as much, but can raise uncertainties (fake news, sensationalised stories, fear mongering)

## Red and blue agents

1. Describe your design of the message potency (a.k.a. uncertainty of red and blue nodes. )
   1. Potency selection is equivalent to selecting a percentage of the uncertainty interval to affect. A potency of 1/5 is equivalent to affecting the top 20% of the uncertainty interval. For example, with an interval of 0,10, a potency of 1/5 would affect nodes on 9 and 10 uncertainty. The amount of potency levels can be increased for finer control ( say, selecting 10ths for a potency interval of 0,10)
   2. For Red, potency is also linked to increasing amounts of uncertainty for nodes effected. High potencies will flip a lot of nodes immediately, but the increased uncertainties and high risk of backfire can end up making it a worse position in the long term.
2. Describe your method for changing the followers' number in case of red agent
   1. The backfire risk is a scaling chance from 0-25% depending on how potent the message is. This risk affects all nodes above the threshold selected who are already non voters. This means that for a max potency message, *all* pre-existing non-voters will be at a 25% risk to swap over to voters.
3. Describe your method for changing the energy level of blue nodes (a.k.a. lifeline)
   1. Blue nodes have a fixed starting amount of energy. Potency is directly subtracted from this. Activating a grey node, however, doesn’t have an energy cost, so it can be a lifeline or a major advantage if Blue is running low on energy. However, if a red node is activated, it will spread a high potency red message – without raising any uncertainties or backfires.

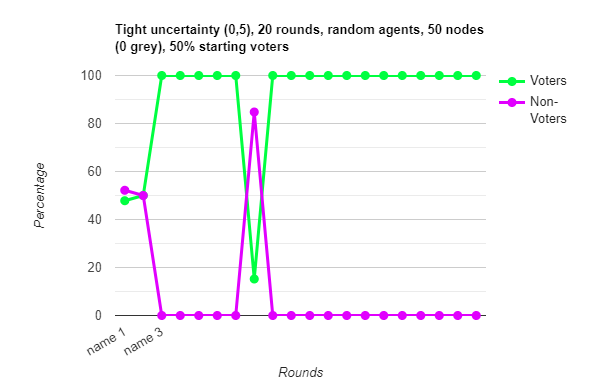
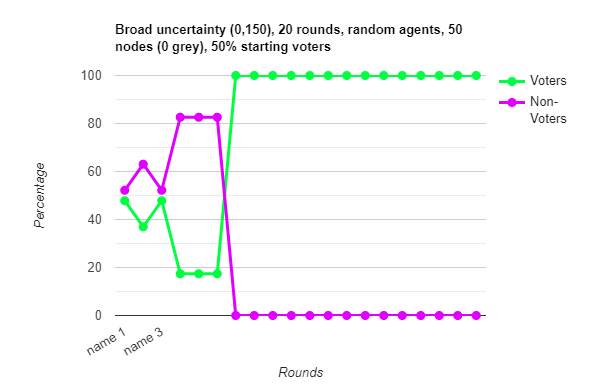
## Grey Agents

All grey agents have a hidden alignment depending on given parameters at the start of the game. A random grey agent will be activated if Blue decides to do so. Once it is grey’s turn, all active agents will spread a high potency message based on their alignment, before deactivating themselves. Grey agents actually have positions on the graph, so an interesting experiment may be to change this into more of a green node interaction, perhaps converting their neighbors with high potency!

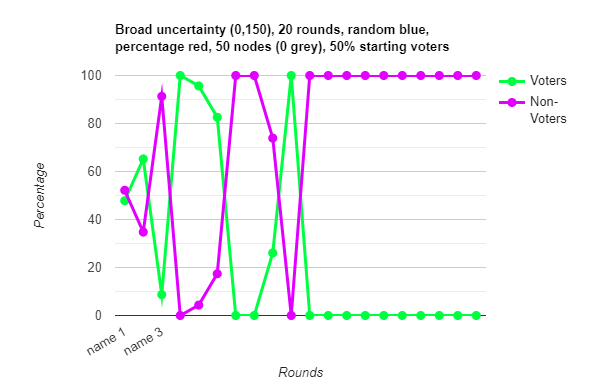
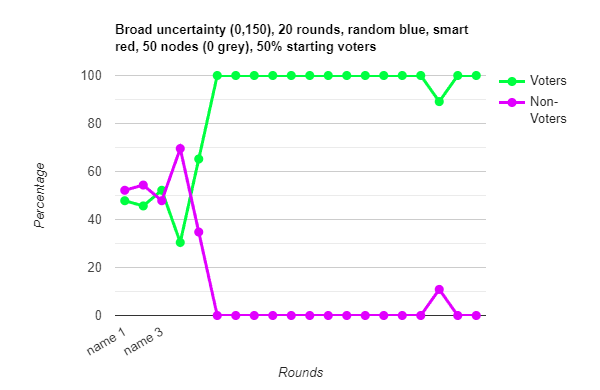
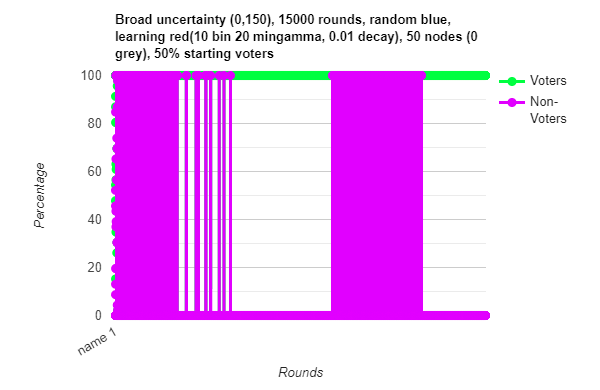
# Validation of Agents

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

**Perform Various Simulations for the following set of questions.**

1. How does the game change if you have a tight uncertainty interval at the beginning of the game?
   1. Tight uncertainty (0,5), 20 rounds, random agents, 50 nodes (0 grey), 50% starting voters
   2. 
2. How does the game change if you have a broad uncertainty interval at the beginning of the game?
   1. Broad uncertainty (0,150), 20 rounds, random agents, 50 nodes (0 grey), 50% starting voters
   2. 
3. Plot distribution of uncertainties for each of the above questions.

Above

1. 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?
   1. Discuss and show with simulation results how many rounds the red agent needs in order to win.
   2. In order to consider this scenario, lets go through Red’s different AI’s against Blue, with Blue adopting the random strategy
      1. **Percentage**
         1. As expected, Red counteracts blue’s random choices, leaving them with no energy by around half way.
      2. **Smart**
         1. Unexpectedly, red fails to win in this scenario. I believe that this is because Red tries to infer the used potency by looking at percentage change from the previous state, but as there is not much change when it plateaus at the top of the interval, red is unable to gather any information.
      3. **Learning**
         1. 15000 rounds
         2. Unfortunately I think the results here are inconclusive. With learning against a more sophisticated Blue AI, I believe that the Learning agent would have far better results as it would be able to react to how Blue budgets their energy. If I were to do this project again, I would implement the learning AI on Blue instead, as it could really come up with interesting results on how to manage the energy level. There is also the possibility that perhaps my parameters could be tweaked-- more bins, less decay, less minGamma perhaps could all make a difference as to performance.
2. 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?
   1. The Blue strategy revolves directly around how resources are managed in accordance to the current graph state. Unfortunately, I did not have time to explore any strategies for this. In general, Blue wants to minimise their energy use. Some factors which affect this are:
      1. Is blue willing to risk letting in a Grey agent when at an advantage, or only if they are running low on energy?
      2. In the longer term, could a strategy of using minimal energy at the start, and then trying to use high potencies to overwhelm Red at the end work?
         1. It would depend on how fortified Red’s position is, and how well Blue can judge the graphs state.
      3. Would changes to the game allow for more skillful play for Blue?

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

# 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

The agents runs well against a human Blue. While the different AI’s are fun to play with, it’s hard to think of challenging strategies – the board state is incredibly complex for a human to comprehend, with many nodes, each with different parameters which play big parts in how the board will look in 1 or two turns. In order to introduce challenging or interesting play, I would either reduce the complexity of the game, or add different ways to interact with the graph. For example, Blue can only input a potency level based on:

* How many rounds are left in the game
  + If I expend a lot of energy now, will I have enough to last the rest of the game?
* How many nodes can I capture
  + Minimising energy use, while maximising nodes taken is a very challenging issue for Blue, especially if Red is not overusing potent messages.

Some great additions I would add if I had more time:

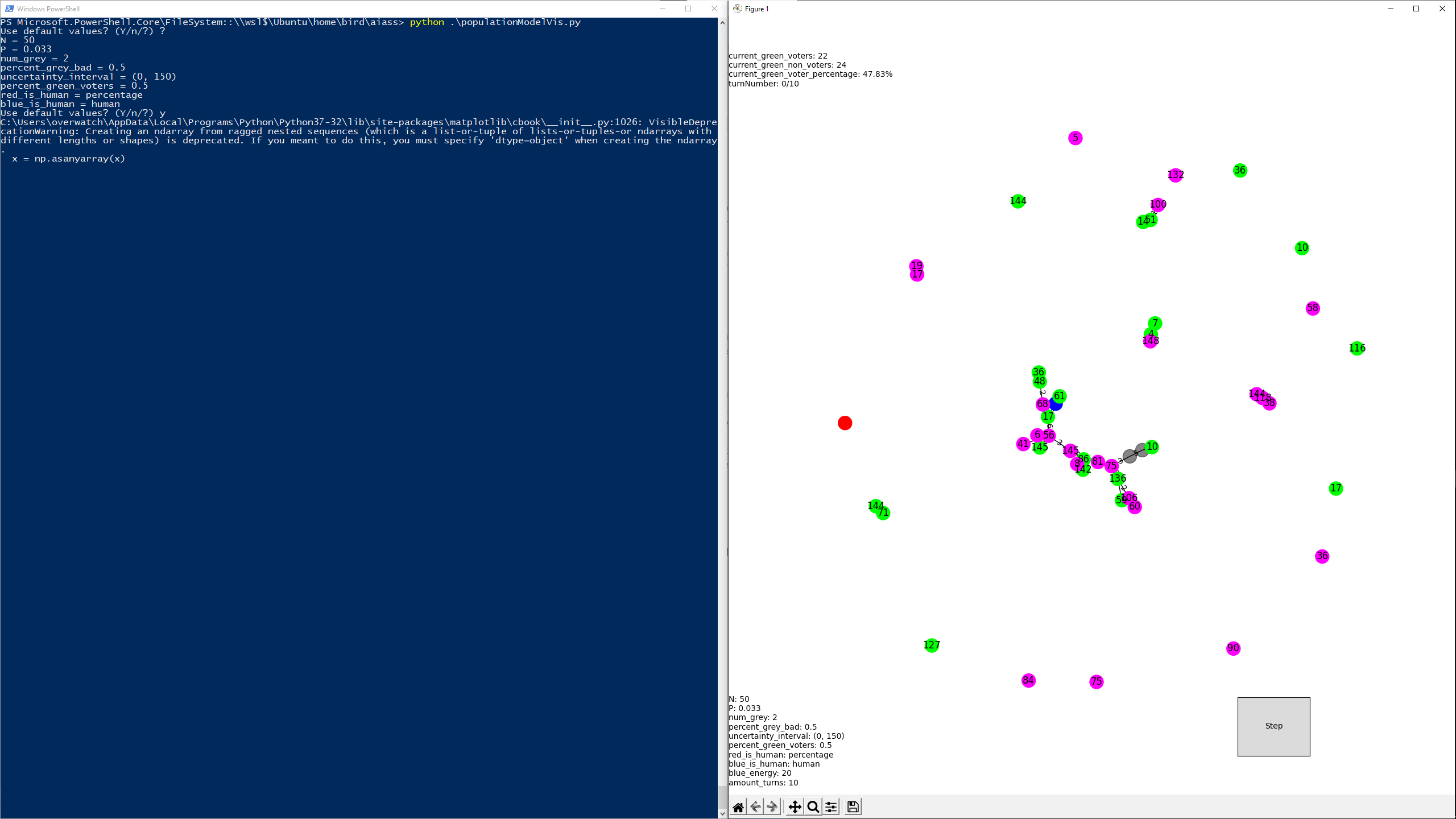
* Spacially selecting certain nodes or groups of nodes. This would allow Blue or Red to target specifically important nodes on the map. For Blue, this would add a whole new dimension to play – you don’t need to use much energy at all if powerful well placed nodes can do it all for you.
* In a similar vein, Grey agents actually have locations on the graph. It would be interesting to see how strategy evolves if the grey agents could affect their neighbors like a highly potent Green node.

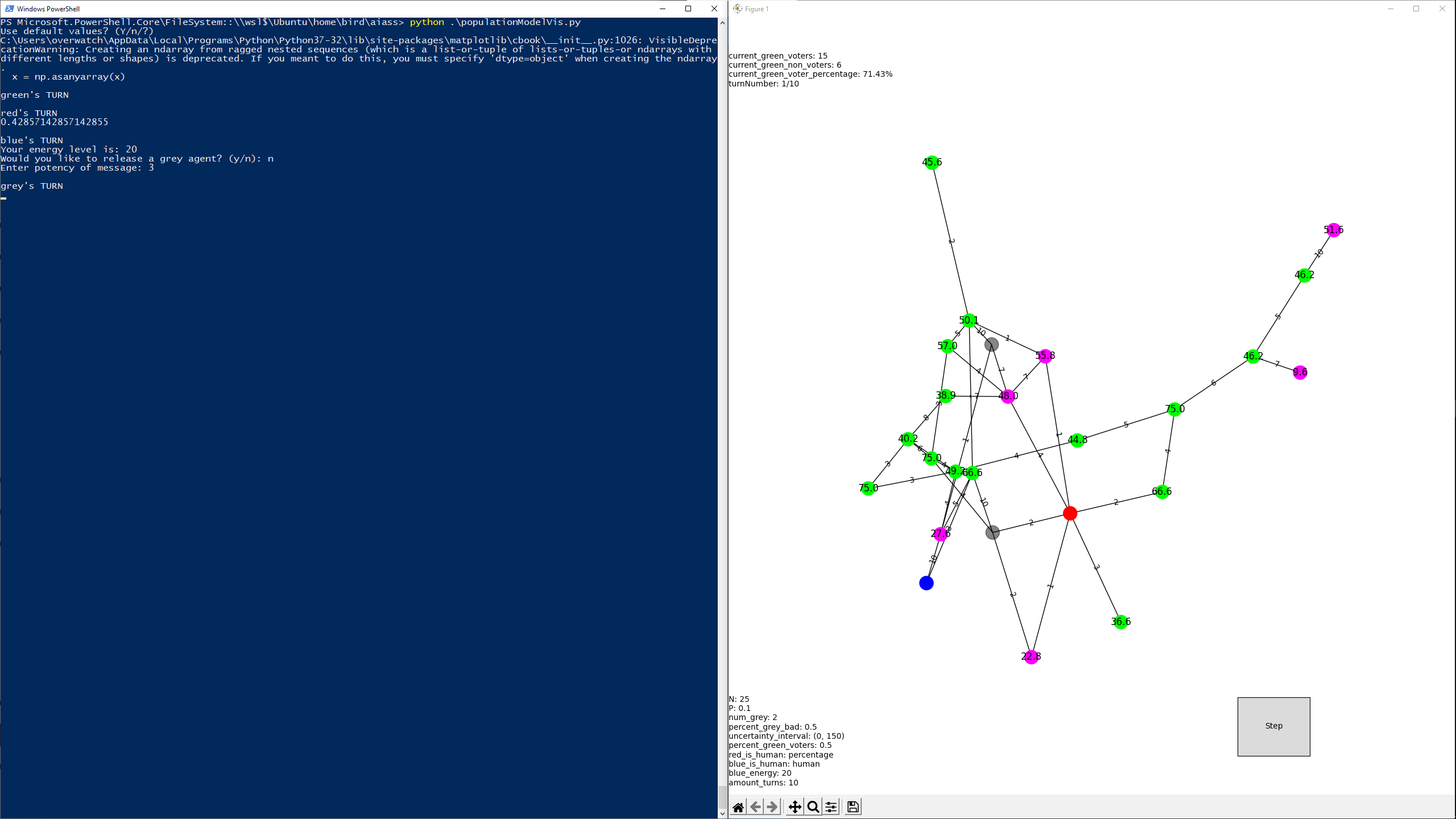
# Visualisation

Describe your visualisation methods with some screenshots

Visualisation is done through Matplotlib, with GUI also provided by it.

1: Initialised game, highly disconnected, large graph, with clusters floating around.

2: Smaller, more connected play field. We have just used a high potency message to counter Red

3: Adjusting position of the graph.

