Using AI to Develop Emergent Ways to Prevent Disease Spread

James Absolom

Robotics - Falmouth University

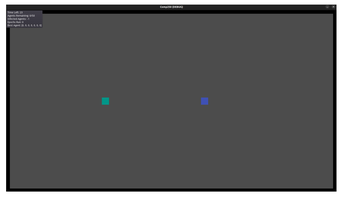
Aim of artefact

After recent human events, it's become increasingly important to come up with new ideas to prevent disease spread through communities. My AI aims to show unique emergent ways of meeting important needs while also avoiding the spread of the disease to either form a better defence against disease spread or confirm our current tactics are suitable.

Introduction

To fulfil the aim of the artefact several components have been developed which will replicate real life:

- A genetic algorithm which is a good fit for this project as it has proven its ability to replicate real-life scenarios well. One example of this is OpenAI's hide-and-seek project [1] where agents played a game of hide and seek and showed many emergent behaviours the creators did not expect. A genetic algorithm like this works by scoring each agent based on how they performed within the environment, rewarding behaviour which fits the brief and punishing behaviour that doesn't. The genetic algorithm then keeps and slightly adapts well-performing agents while getting rid of poorly performing agents.
- An series of agents to be trained by the genetic algorithm, the agents will replicate any animal which a pandemic can spread through. These agents all act independently of each other as they poll the environment around themselves and decide on an action based on these values, this helped new behaviours emerge quickly thus fulfilling the aim of the simulation.
- An environment in which the agents exist which has bounding walls to keep agents in and a green and blue square to give food and water:



The genetic algorithm is trained in *epochs* which are defined as one iteration of the training simulation; in our scenario this is defined by the 30 second timer we have for each round, though this doesn't replicate the time scales from real life it does help the algorithm learn quickly.
The project has been coded in Godot as it is an open source and lightweight game engine meaning development will be faster. Godot uses GDScript [2] which is a python-like programming language meaning it is object oriented which works well for a multi agent system.

Agents

Each agent has several parts which control the actions and movements it takes, these are:

- A kinematic body which allows for movements and collisions inside the environment.
- An action controller script which acts like a state machine swapping between different states depending on what variables it is fed, this script runs all the movement vector calculations, having this split into a different script also slightly increases the performance of the system as it can run the calculations in parallel as opposed to queuing them after the polling actions.
- A brain script which polls the environment around the agent and works out the next action based on trained values and the results from polling the environment.
- a white and a blue sprite to visualise the spread of the disease for the user.

Genetic Training

To train the agents the simulation runs a genetic algorithm which keeps and mutates well-performing agents while getting rid of agents which score poorly. The score of the agents is adjusted by multiple factors:

- +1 score for every second the agent is alive and not infected
- -500 points for getting infected
- -1000 points for dying via lack of food or water

These values can be adjusted by the user to whatever they want, however these values have been chosen to specifically punish dying over getting infected as it seems like to worst outcome of the scenario. The simulation also adds only 1 point per second survived as there is no advantage to a higher value as right now we're only tracking one thing that adds points.
The agents are then sorted by their scores with the best-scoring agents being the highest on the list. The top 1/3 of this list is then taken and copied across to the list for the next epoch. This top 1/3 is then mutated by a random amount capped by a max mutation variable and added to the next epoch list. The copy and then mutate takes up the top 2/3 of the next epoch so the remaining 1/3 is set by random values meaning we avoid over-fitting one solution and look for other ways of solving the problem.

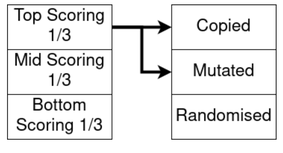


Figure 1:A diagram that shows the genetic learning

Mathematical Equation

Each action an agent can take has a priority, these priorities are compared to work out which action the agent should do next; these priorities are calculated by looking at the environment around the agent and then multiplying the values taken from the environment by a trained multiplier. The equations to work out the priority mostly look like this:

$$\frac{(1-(\frac{MaxValue_1}{CurrentValue_1}))*multiplier_1+...+(1-(\frac{MaxValue_n}{CurrentValue_n}))*multiplier_n)}{n} \quad (1)$$

Where n is the number of variables taken from the environment, Note: this is generalised, the equation changes for some parts of the code
The max values which are on top of the fractions are set as global variables and define the maximum a value can be, for example, the total amount of agents in the system or the maximum amount of food an agent can have. The bottom value on the fractions are the value of a variable inside the environment, this is got when the agent polls the environment around it. The equation then takes this value away from one to standardise all the values to between 0 and 1 and make it so higher the values correctly correspond to how they should react. The equation then gets the average by adding the multiplied values together and dividing them by how many values there are.
This means multiple values can lead to one action as it will just be an average out. Here is a diagram of the whole process to hopefully make it easier to understand:

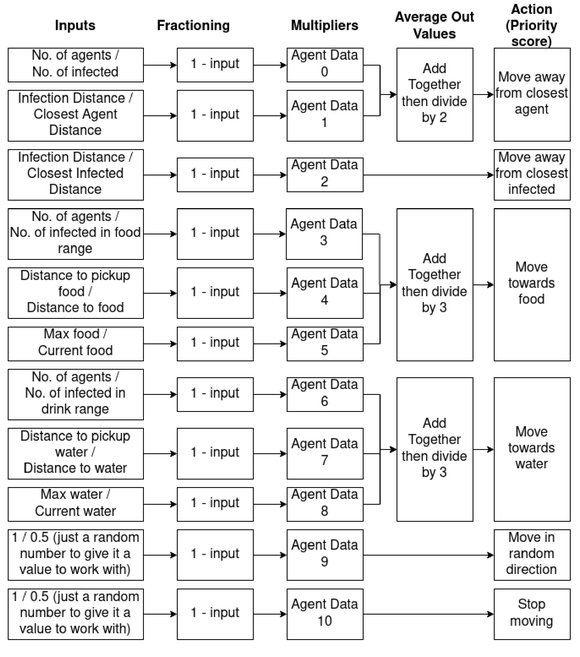


Figure 2:A diagram that shows the equations

Future changes

Potential further development of this project revolves around the addition of new deep learning methods. One implementation would be moving away from the simulation's current decision-making methodology and return to a more traditional neural net, this would allow the agents to develop more complex behaviours compared to the current system. Another change would be to move away from a traditional genetic algorithm to a NEAT algorithm, this would allow the algorithm to change the composition of the neural net meaning that the AI could become far more advanced. Both these ideas help the simulation to better fulfill its aim of developing new emergent behaviours.

Conclusion

The developed system works well enough to fulfill the original aim of the artefact however it still has flaws for example the food and water sources become overpowering contributors to the spread of the disease. This is likely because of the poll rate of the agents causes them to sit in the food and water sources while waiting for a poll. There are two potential solutions to fixing this issue; the agents could be made to poll more frequently and ignore the lag that would induce in the system, or the environment could have more food and water sources to ensure the agents have a choice in which source they go to.
Despite both these conclusions, the genetic learning system did show some unique emergent behaviours. The solution they generally converged upon was to run in for food and water and then instantly run towards the exterior wall to ensure maximum distance from every agent as possible. Though it is difficult to decipher entirely what these solutions' real-life parallels are, it may suggest stockpiling supplies at the start of a pandemic may lead to a higher chance of avoiding the disease. A similar pattern of behaviour was observed at the beginning of the COVID-19 pandemic when supermarkets experienced record sales of goods "exceeding even peak Christmas trade" [3]. This draws an interesting parallel to real life and poses the question "what if we added supply shortages into the system?".

QR code and link to Github Repo

You can follow the QR code or this link to find the project and a list of all references in "references.md" github.falmouth.ac.uk/JA244121/Comp250-JA244121.