Runner Chaser Game

By Zach Bearinger and Tom Hadlaw.

November 30th 2014

Abstract

This paper presents a multi-agent system, which looks to evaluate the interplay between two agents who have opposing goals and have an effect on the environment. Using the Madkit framework we developed a simulation that has one agent chasing a different agent throughout an interconnected graph. This is to represent someone trying to find someone else by going town to town to follow their path. Each node of the graph contains a separate AI which represent a population in a location. This agent will be informed of the movements of the other agents. The chaser agent will interrogate the agent at each node as to the next location of the runner. The agent at each node doesn't always want to cooperate with the chaser agent.

Introduction

In our simulation there is a combination of passive, active and complex agents moving through a map. For our map we chose to hard code the connections and locations of the cities. Each city is connected through edges which restrict some agents' movement. This ensured that the map would be effective and complete for our testing. Our cities are populated by passive agents who will answer the chaser with either the correct or incorrect next destination. The agents will choose whether or not to help the chaser depending on their loyalty. This loyalty is randomly generated based on a set of modifiable parameter in order to provide a shifting test environment that we can easily manipulate. The running agent will transfer between cities and declare its next destination as it progresses. The agent moves randomly to prevent any exploitable patterns from emerging that the chaser could use. The chaser will poll everyone in each city it visits and try to determine where the runner went. Depending on the simulation a portion of the population will provide the chaser with faulty data. It is the chasers job to choose the most likely destination the runner went to try and catch them.

Related Work

Our project will utilize several different types of AI and their interactions. One key aspect which has had a lot of research is having AI traversing a graph. There are two algorithms which focus on fast path finding cooperation push and swap, and push and rotate. Push and Swap has two distinct operations. The push operation will move the input agent towards its going until its path is obstructed by another agent. The swap operation will direct the AI's towards the nearest intersection to allow one to pass another. Push and Swap only provides an incomplete solution and so work began on

push and rotate. Push and rotate works on the same principals as push and swap. However the rotate function cycles segments of the graph to maneuver agents. Using this method from tests provides a complete solution for maneuvering agents around a graph. However as our simulation will allow multiple agents at the same position, we will use a maneuvering system similar to the wumpus world problem.

The chasing AI will poll each location it visits akin to IBM's Jeopardy competitor. Watson's end result is an AI that will answer if its leading solution passes a confidence interval. The important factors to this project is how Watson determines its confidence interval. Researchers determined that for a quiz game like Jeopardy there was a correlation between how precise each answer was against how many questions were attempted. From this conclusion they developed a method which checks the information it knows with supporting evidence. Initially the system will deduct a number hypothetical solutions to the question. From there depending on the supporting evidence for each solution the system will rank the solutions and if the leading solution surpasses the others by a confidence interval. To develop the confidence interval IBM used machine-learning by applying a series of training questions with known answers.

Environment

Our environment is a map containing multiple cities as its nodes. Each city contains a list of adjacent cities through which they are connected and serve as the edges the agents can move along. We developed a set map as opposed to a randomly generated map means that we can vary how the landscape is set. For future testing we can create maps with different characteristics. Each city contains a population of passive agents who are arranged as communities within Madkit. These communities interact with both the runner and chaser. The runner can broadcast to each population in their separate community to alert them as to its next destination city. Akin to the broadcast of the runner the chaser can broadcast demanding all members of the community provide it with the information of where the runner went to. After this call the community will respond individually according to their loyalty.

Agent Architecture

Passive Agents

All of our passive agents in the simulation are NPC's in each city. These agents have a limited perception of the world. They are part of a community which can receive commands from both the runner and chaser. They also know if either agent is in their city. The agents' only logic is when the chaser asks for the runner to provide the next location of the runner. By this point the runner will have passed through providing each NPC with the true value of where the runner went. When the chaser calls upon the NPC the NPC will use its set loyalty value to either respond or wait. The reason for this is that with our small testing map frequently the chaser would cut off the runner, if the chaser merely waits for NPC response it lead to more interesting simulations. The loyalty is based upon a Gaussian distribution with a preset mean and standard deviation which we can modify to the loyalty of NPC agents on the map.

Active Agents

Our active agent is the runner. The runner can only perceive the current city they are in and which cities are adjacent. The runner can influence the community of NPC's in the city he is currently in. The runner will randomly select a new route to take once entering the town. After a set period of time the runner will progress to a new city after alerting residents of the current city of their new destination. Having the runner rush from city to city in this fashion allows unpredictable patterns to develop that will force the chaser to only catch up by following the trail of the runner. The only restrictions that the runner has on its movement is that the runner will never go to the location of the chaser, furthermore the runner will never return to a previous city as this could lead to very short simulations and uninteresting runner paths.

Complex Agents

Within the simulation we only have one complex agent who exhibits reasoning and that is the chaser. The chaser can perceive when the user is in the city, the adjacent cities to its current location and the response of the community of the current city. Unlike the runner, the chaser is unable to influence or modify its surroundings. The chaser will progress from city to city polling the NPC. From there the chaser will either have their next destination or be forced to wait to ask again. From the result the chaser will proceed to its next destination. The chaser will continue until it catches the runner and puts an end to the simulation.

Implementation

For our test simulation we implemented the following table of cities which appears as the map seen in Figure 1. Within each city contains the following amount of NPC's with loyalties based on a normal distribution, and a list of adjacent cities. Our map isn't a mapping of any real world geometry, it is merely a visualization of a random map. We applied names to provide a more relatable visualization. Our simulations have one NPC per city who represents the collective population. Their loyalty is represented by a Gaussian distribution with a mean of 30 and a standard deviation of 20.

Table 1 Simulation Map Cities

City	*Connecting Cities*
Halifax	Toronto, San Francisco, New York
San Francisco	Las Vegas, Halifax, New York, Toronto, Los Angeles
Wolfville	Miami, Toronto, New York,
Toronto	Halifax, Wolfville,
Los Angeles	Miami, Las Vegas, New York, San Francisco
New York	Halifax, San Francisco, Wolfville, Los Angeles
Las Vegas	Los Angeles, San Francisco
Miami	Los Angeles, Wolfville

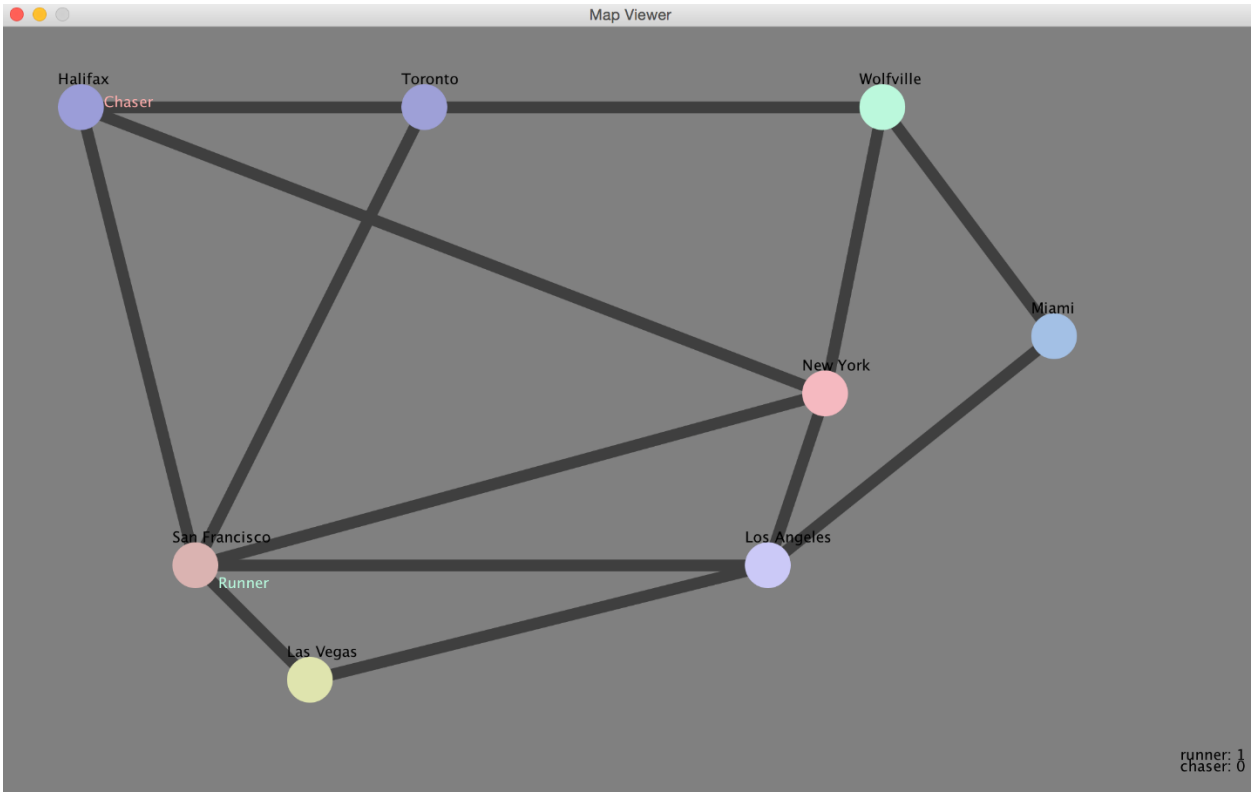

Figure 1 Simulation map

Our program will output the relevant data in a short output after the simulation is completed. An example of which is seen below in figure 2.

```
Chaser took 4 moves to catch runner!
Chaser polled npc's a total of 5 times.
Chaser recieved :4 responses.
Runner took 13 moves before it was caught.
```

Figure 2 Example Program Output

Results

We ran our simulation a total of 10 times to acquire data to represent how efficient our solution is. Our raw data is represented in Table 2 below. As you can see the chaser was always able to catch the runner with less movement. What this represents is that as the map is fairly small the runner will double over the chasers path, or run into the chaser. For future work it would be interesting to see how a more expansive map would impact the framework.

Table 2 Raw Data Response

Test	*Runner Moves*	*Chaser Moves*	*Chaser Polls*	*Chaser Responses*	*Average Response*
1	13	4	5	4	0.8
2	51	20	26	20	0.769230769
3	21	7	8	7	0.875
4	7	1	1	1	1
5	35	13	16	13	0.8125
6	45	18	21	18	0.857142857
7	9	2	2	2	1
8	12	3	4	3	0.75
9	23	8	9	8	0.888888889
10	9	2	2	2	1
Average	22.5	7.8	9.4	7.8	0.875276252

When you compile the data into a line graph as seen in Figure 3 comparing the amount of moves both the runner and chaser take as well as the difference between them it is clear that the chaser is pursuing the runner's path but is intercepting more recent trails.

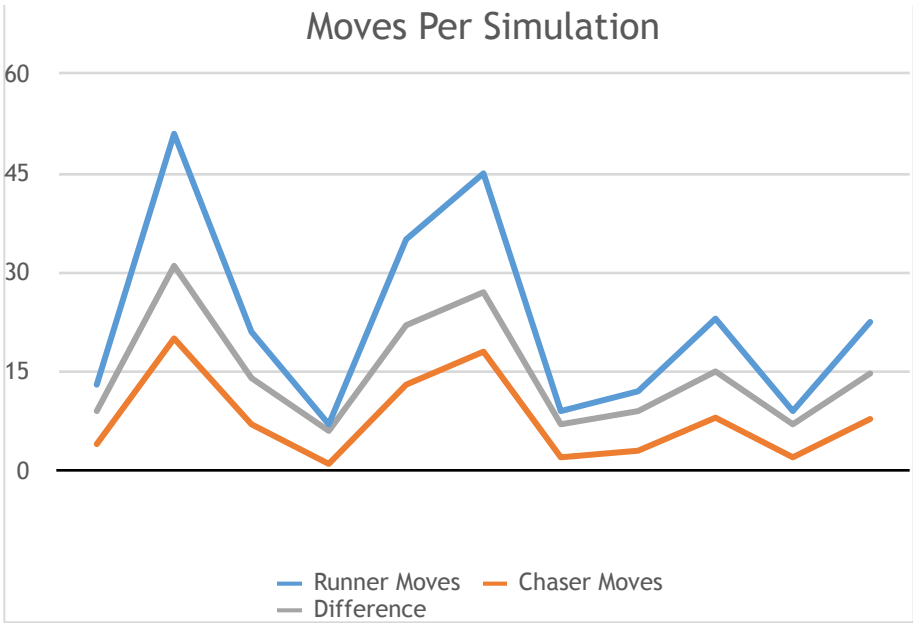

Figure 3 Moves per Simulation

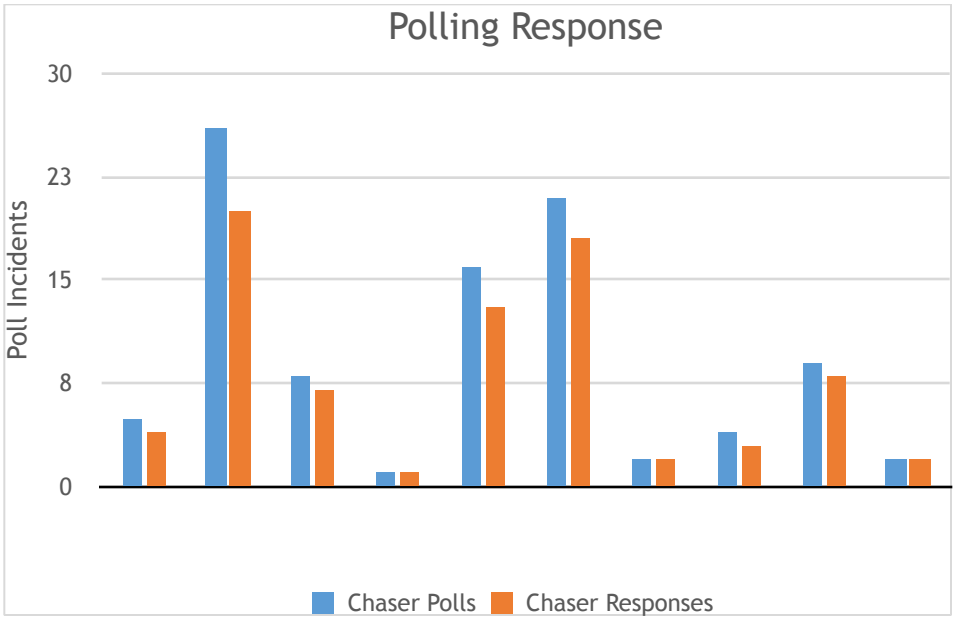

Figure 4 Polling Response

Above in Figure 4 is a visual representation of how the polling through each test proceeded. As you can see the longer the simulation the greater frequency of non-responses as you would expect. Overall the average response rate turned out to be roughly 88%. This is what was expected considering we set our Gaussian distribution to be a mean of 20 with a standard deviation of 30. As this distribution represents the likeliness of a non-response it's expected that approximately 80% would be a positive response directing the chaser.

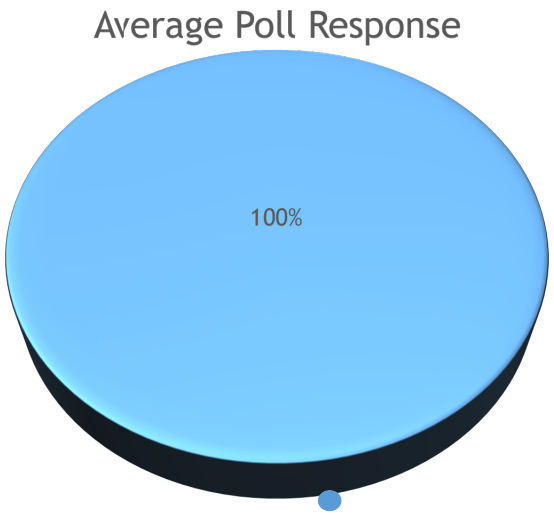

Figure 5 Average Poll Response

From the data it can be seen that experimentation on a larger map is required for greater results providing more insight into the interplay between the runner and the chaser. The data proves that the system is running correctly but isn't in-depth enough for true insight into the premise.

Conclusion

Through this project we have proven that the interplay between agents that are chasing each other. However we feel like greater testing and work would lead to more interesting results. We successfully created the software required for this kind of simulation but the system needs more refinement and testing. We find that this project is an excellent starting point that leads into very interesting interplay between contrasting agents. This system could also be integrated into other projects to provide another layer of mechanics. In particular we see a more refined version of this system being integrated into video games with great effect.