Runner Chaser Game

# Abstract

# 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 location. 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 respond with either the truth of where the runner went or another city entirely. The loyalty is based upon a bell curve that is set before the simulation begins. We can modify the global loyalty to test for interesting results.

## 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 choice the runner will not make is returning to a previous city as this could lead to very short simulations.

## 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 populous. From there the chaser will compare all the data received from the population. 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 implementation we implemented the following map. Within each city contains the following amount of NPC’s with the following loyalty.

# Examples

# Conclusion