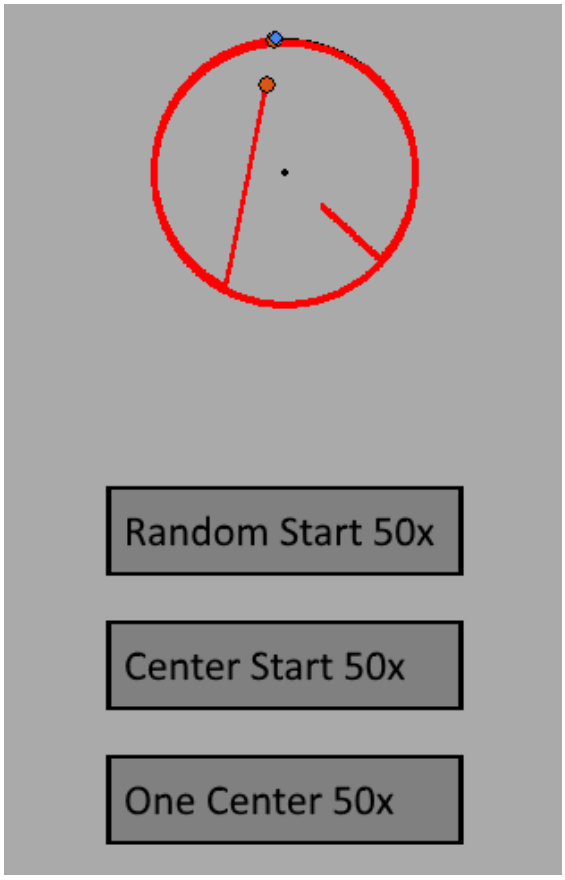


Robot Evac

Luke Harper Colin Zeidler

100886836 100881359

COMP 4001 Project

Evangelos Kranakis

Carleton University

13/12/2016

Contents

Implementation

Robot Evac was implemented using Python 2.7 and PyGame version 1.9.2rc1. The Python language was chosen for its ability to quickly create experiments and test conditions. When evaluating between Java and Python the PyGame framework was the key factor in our decision.

Pygame is actually a set of python modules that were created to develop video games in Python. It is highly portable being able to most graphic languages such as OpenGL, DirectX, windib, X11 and even the linux frame buffer. This allows our Evacuation simulation be run in many different setups. Pygame's calls make use of C-built functions that allow the framework to make use of the c language's speed instead of being restricted by python's interpreter. Pygame also abstracts away most graphical code, graphical calculation and graphical pipeline allowing code footprint to be small and simple.

Main Loop

With PyGame setting up a window, displaying a GUI, and capturing user input is very straightforward. RobotEvac's main.py handles the execution of our simulation loop. It firsts loads our button resources and creates our room circle.

The loop initially sets the simulation to not running, draws the buttons on the screen (using screen.blit) and listens for user input. Pygame provides an event object that is generated on user input. This event object contains an event type, and the data associated with that event type. The main loop checks for mouse events which supply the location of the mouse and the button the mouse was pressed. If mouse button 1 was pressed the main loop launches the appropriate simulation based on the mouse location.

Pygame's framework revolves around surfaces which can be considered like a blank sheet of paper. You can have many surfaces, you can colour, add more surfaces to other surfaces or draw different shapes or lines. The main surface is called the screen, and it is this surface at which everything is displayed. When the user clicks on a button the main loop runs the simulation; it sets the state to running and sets up the botnet object according to the simulation selected.

When running the main loop updates the bots state, redraws the screen. When a simulation round has completed it displays the time. When the entire simulation has completed it prints the output to a file.

Classes

There are four state objects in Robot Evac. These objects contain the data that is represented on the screen. These objects include the robots, the evacuation point, the robot trails and the network of robots. The room circle is drawn directly onto the screen in the main loop and is not represented by an object.

Robot Class

The Robot class represents the autonomous robots searching for the evacuation. It contains data on its position, the network the robot is in, its speed, its trail and other properties. The class includes functions to update itself where it moves to search for a location, and marking where it has been with a trail. It has a function to set its destination which is used to set its initial start position. The Robot class also contains a draw function which handles rendering of itself as a circle and drawing new members of its trail.

Note: a Robot's trail is meant a visual representation of the path the robot took to find the evacuation, and it is not considered as part of the algorithm.

BotNet Class

The Botnet class handles all the communication between robots, setting up the robots and coordinating the search. The botnet contains properties like the list of running bots, the list of finished bots, the location of the evacuation point and the time the bots have been searching. The initialization function of the botnet takes in a mode which determines the type of simulation. The simulation may be "none" in which case the bots start at the center of the circle or "random" for one bot or both.

The botnet is responsible for telling the robots to update themselves. The botnet also updates the evacuation point to determine if any bots have located it. When a bot has located the evac point the Botnet class communicates that to the other robot and tells the robot to stop search and move directly to the exit.

EvacPoint Class

The EvacPoint class represents the Evacuation point that the robots are looking for. Upon its creation it generates a random location for itself along the edge of the circle. It is responsible for drawing itself, and a call to its update function checks whether or not it has been discovered by a robot, and upon discovery informs the robot it's been discovered.

Trail Class

The trail class represents the trail of the robot, it is a visual representation of the path they have traveled. When a trail has been created it sets the color size and position properties and draws itself on its draw call. It is represented by circles.

Algorithm

For theoretical calculations of these algorithms a unit circle and unit travel time are considered.

Note: difference in timings from theoretical to practice due to calculation of radial speed in python implementation

The Circle in the simulation has a radius of 75 pixels and the robot's speed is 1 pixel / 10ms

Center

In this simulation the robots both start in the center of the circle and the evacuation point is located at a random spot along the room circle's edge.

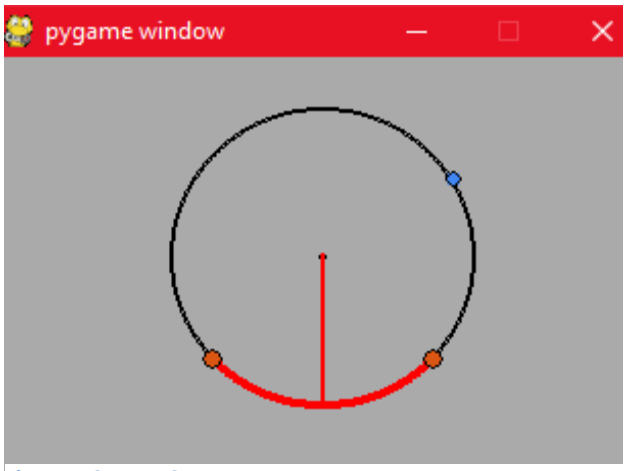


Figure 1 Center: Start

Each robot starts moving towards the bottom circle edge. They then split off one robot going clockwise the other robot going counter clockwise.

When one robot finds the exit, the other robot is informed and moved directly towards the exit's location.

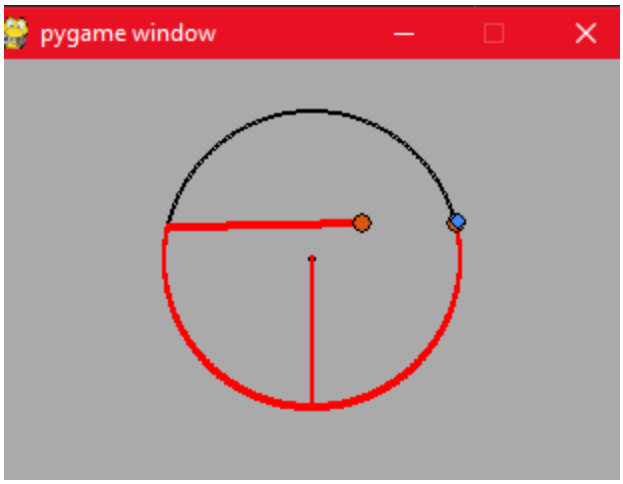


Figure 2 Center: Evac Found

Worst Case

This algorithm's worst case scenario is when the evacuation position is located at 9oclock or 3oclock (if

the room circle was a clock.). When this happens the slowest robot needs to go down to the edge, a quarter of the circumference then across the room's face to the exit which is a distance of d (Figure 2 is almost the worst case, the exit is a little high on the circle).

Search along circle: Circumference = $\frac{1}{4}(2\pi r)$

Distance to cross circle: Diameter = $2r$

Distance to edge: Radius = r

Worst Case = $\frac{1}{4}(2\pi r) + 3r$

Worse case time $\simeq$ 4.57

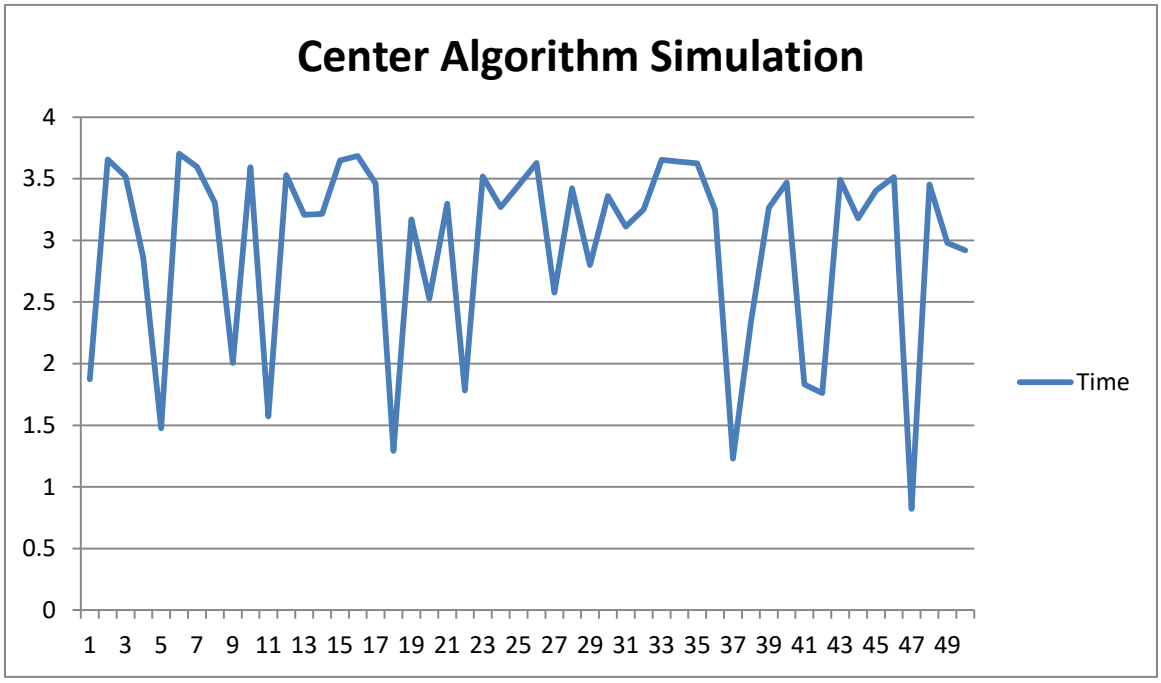
Simulation Data

After running 50 simulation of the center algorithm the following results were found:

max time: 3.70200014114

min time: 0.822000026703

average time: 2.96330000401



One Random

In this Simulation the one robot starts at the center of the room circle while the other starts at a random location inside of the circle. The evacuation point starts at a random location along the circle's edge.

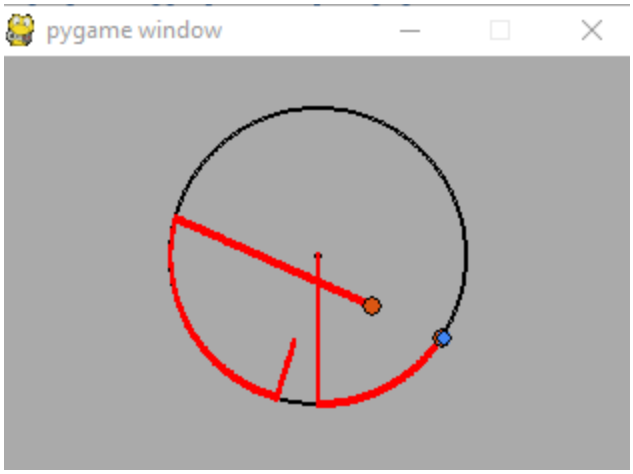


Figure 3 One Random: Evac Found

The Robots go to the nearest edge of the circle or if in the center move to the bottom of the circle (6 o'clock). The robots then chose a random direction to search and search along the edge of the circle until the evacuation point is found.

Once an evacuation point is found, the robot who did not find it immediately stops searching and travels to the evacuation point.

Worst Case

This algorithm's worst case scenario is when the randomized robot's starting position is located at the center. This would cause the randomized robot to travel to the bottom of the circle as it is its closest edge. This would result in both robots traveling towards the same point along the edge. If the robots randomly choose the same direction to search then the algorithm becomes a one robot search as they are both searching along the same path at the same time (or alternatively functions the same as the center start algorithm if they pick different directions).

If the evacuation point is directly beside the bottom of the room circle, but not in the direction the robots chose to search then the robots would need to travel the whole perimeter to reach the evacuation point.

Travel the whole circle perimeter: circumference = $2\pi r$

Travel to Bottom of slowest robot: radius = r

Worst case: $2\pi r + r$

Worse case time $\simeq 7.28$

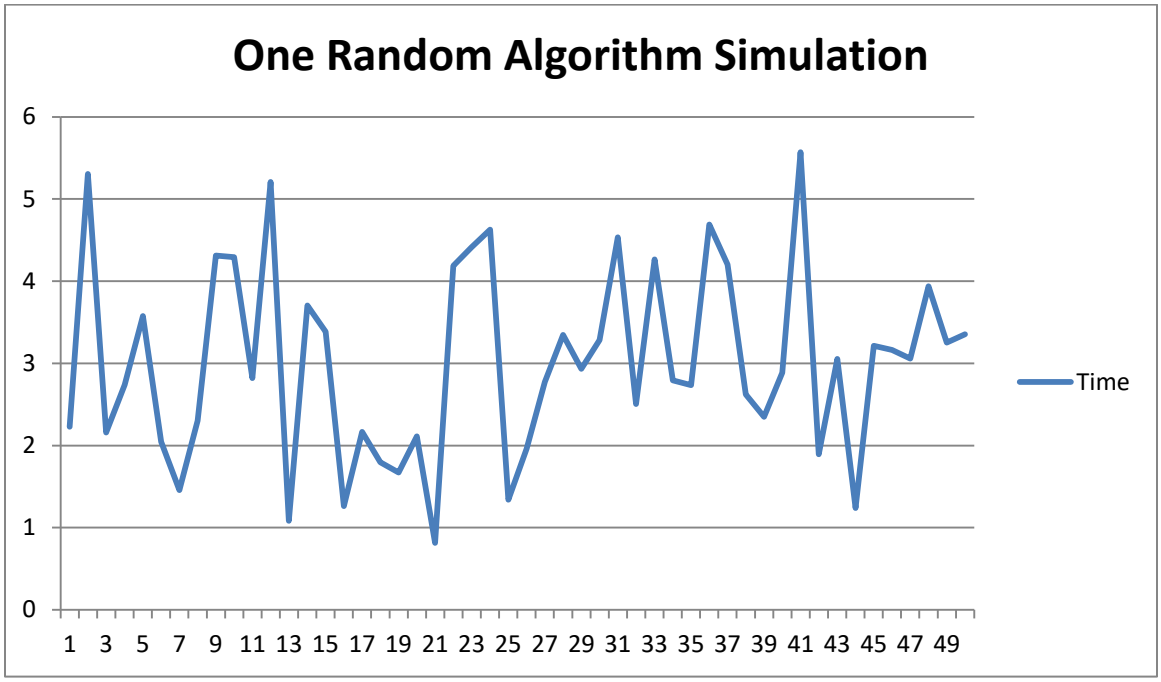
Simulation Data

After running 50 simulation of the one random algorithm the following results were found:

max time: 5.56999993324

min time: 0.81500005722

average time: 3.01279999733



Both Random

In this simulation both robots start at completely random locations within the circle and the evacuation point is located at a random spot along the room circle's edge.

The robots go to the nearest edge of the circle or if they randomly start in the center they head towards the bottom edge. Upon reaching the edge of the circle the robots chose a random direction to search in.

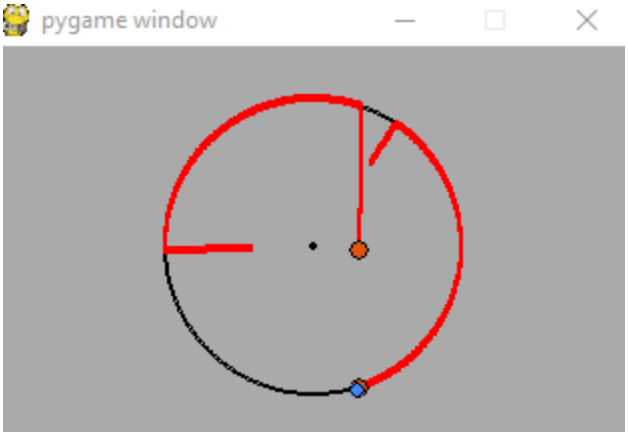


Figure 4 Both Random: Evac Found

Once the evacuation point is found, the robot that did not find the evacuation point stops searching and goes directly to the evacuation point.

Worst Case

This algorithm's worst case scenario is similar to the one random worst case. When the robots' randomized start positions both robots to start at the center of the circle and they decide to search in the same direction it becomes the same problem as one random (or alternatively functions the same as the center start algorithm if they pick different directions). The robots search along the same path and the algorithm becomes a one robot search.

Same calculations as the one random algorithm:

Travel the whole circle perimeter: circumference = $2\pi r$

Travel to Bottom of slowest robot: radius = r

Worst case: $2\pi r + r$

Worse case time $\simeq$ 7.28

Simulation Data

After running 50 simulation of the both random algorithm the following results were found:

max time: 5.06399989128

min time: 0.753000020981

average time: 2.39054000854