**Spike:** 6

**Title:** Navigation with Graphs

**Author:** Luke Valentino, 103024456

**Goals / deliverables:**

Expand the Task 5 navigation graph simulation to demonstrate the following:

* A game world that is divided into a larger number of navigation tiles, and corresponding larger navigation graph  
  structure.
* A path-planning system that can create paths for agents, based on the current dynamic environment, using cost-  
  based heuristic algorithms that accounts for at least six types of ‘terrain’ (i.e. nodes with different costs).
* Demonstrate multiple independent moving agent characters (at least four) that are able to each follow their own  
  independent paths.
* Demonstrate at least two different types of agents that navigate the world differently.

**Technologies, Tools, and Resources used:**

List of information needed by someone trying to reproduce this work.

* Swinburne Lecture Notes
* Swinburne Maths and Physics PDF
* docs.python.org
* Code from Lab 5

**Tasks undertaken:**

Start with the code from Lab 05. This represents the world as a grid graph and implements search algorithms to create paths from point a to b:

* Create an agent class. This class renders an agent in the window, and makes it move from at a constant speed. Currently it does not move within the graph, instead it moves freely.

A computer screen shot of a program

Description automatically generated

Initial Agent Class

* + To aid in the movement of the agent, you will need to expand the point2D class to account for more operations.

A screen shot of a computer program

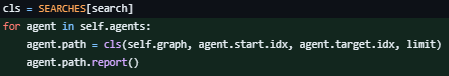
Description automatically generated

Expanded point2D class

* Next, in main.py, spawn an instance of your agent object.
* Your agent should now be spawned in the world.
* The next step is to get the agent moving along a determined path that is calculated from the box world. To do this we will need to edit some existing code.
  + First modify the agent to hold a path object.
  + Next you need to modify the box\_world.py file to hold a list of agents, with a subsequent add\_agent method.

A screen shot of a computer

Description automatically generated

* + Modify with draw function in box\_world.py. Make sure it uses loops through every agent in self.agent and uses agent.path instead of self.path.
  + Modify plan\_path in box\_world.py to loop through the list of agents. This method is to set the path for each agent in our world.
  + Modify main.py to instantiate an agent with a start, a target, and a chosen search algorithm.
  + Next we will modify our agent class to move each point at a constant speed. When the agent reaches the target point (the target point is within the proximity radius), the next point in the path is selected and set as the target. When the agent reaches the final target, it stops. Here are the methods:

Seek Method:

* def seek(self, target\_pos, dt):
* # Moves the agent towards the target position following a direct path
* direction = target\_pos - self.pos
* direction.normalize()
* self.pos += direction \* self.speed \* dt

Arrive Method:

    def arrive(self, target\_pos, dt):

        # Moves the agent towards the target position. If target is in proximity threshold,

        # sets the position to the target and stops movement

        direction = target\_pos - self.pos

        distance = sqrt(direction.x\*\*2 + direction.y\*\*2)

        if distance <= self.waypoint\_near\_dist:

            self.pos = target\_pos

            self.at\_final\_target = True

Path\_Finished Method:

    def path\_finished(self):

        # Checks if the agent has reached the end of the path.

        return self.current\_node\_index >= len(self.path.path)

Next\_waypoint:

    def next\_waypoint(self):

        # Advances the waypoint index to the next one in the path unless it's the last one

        if self.current\_node\_index < len(self.path.path) - 1:

            self.current\_node\_index += 1

        else:

            self.at\_final\_target = True

Is\_near\_waypoint:

    def is\_near\_waypoint(self):

        # Checks if the agent is near the current waypoint based on the defined near distance threshold

        current\_target = self.current\_target()

        distance = sqrt((current\_target.x - self.pos.x)\*\*2 + (current\_target.y - self.pos.y)\*\*2)

        return distance < self.waypoint\_near\_dist

Update:

    def update(self, dt):

        # Updates the agent's state every frame; checks path status and moves towards the current target or handles arrival

        if self.path and not self.path\_finished() and not self.at\_final\_target:

            if self.is\_near\_waypoint():

                self.next\_waypoint()

            if not self.path\_finished():

                self.seek(self.current\_target(), dt)

            else:

                self.arrive(self.path.end\_point(), dt)

        elif self.path\_finished() or self.at\_final\_target:

            self.at\_final\_target = True

* Now that the agent is full functional, we can create subclasses with different behaviours. For example.

class FastAgent(Agent):

    def \_\_init\_\_(self, start, target, world):

        Agent.\_\_init\_\_(self, start, target, world, speed=100, radius=10)

        self.color = (0, 1, 0, 1)

**What we found out:**

This spike was one of the more difficult ones of the semester. Although I found it quite difficult it very much did address learning outcomes 1 and 2. In particular, this task focused on ULO 2.