Spike: 8
Title: Navigation with graphs

Author: Ryan Chessum - 102564760

Goals / deliverables:

Create a GOAP simulation that demonstrates the effectiveness of the technique in considering long-term outcomes of actions (related to side-effects and/or time delays) and can plan and act intelligently.

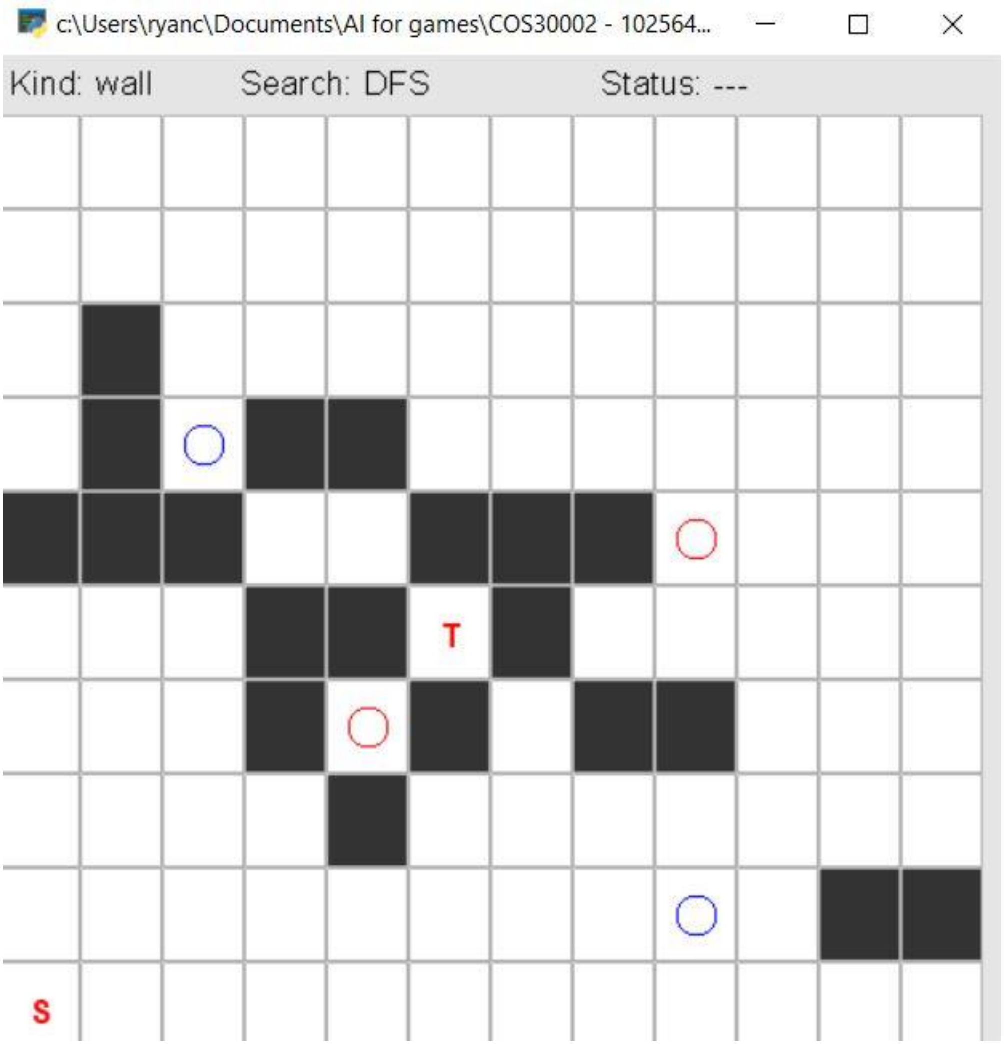
Technologies, Tools, and Resources used:

- Visual Studio Code
- Python 3.8.0
- Lecture Notes
- Code from previous spikes
- Code from previous labs

Tasks undertaken:

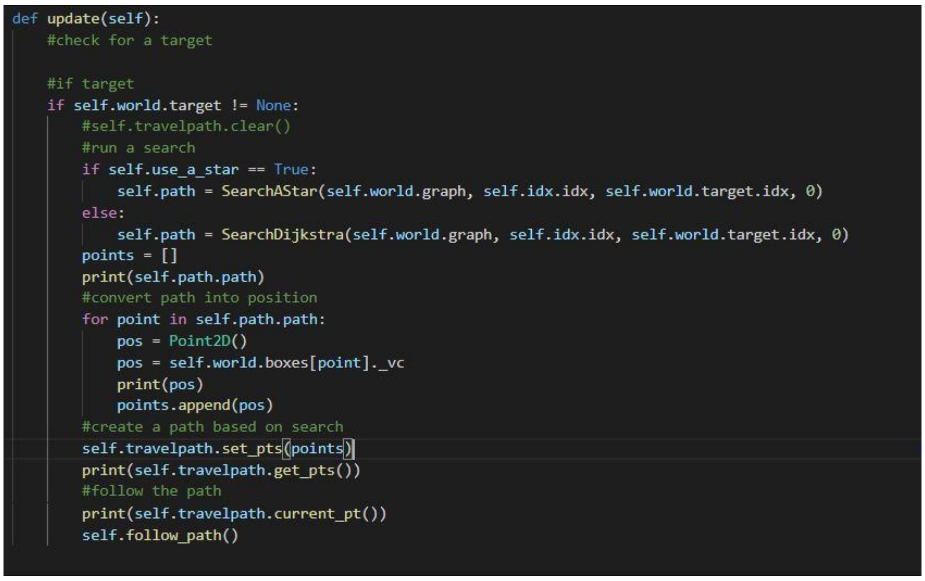
- Copy code from Lab 15
- Add agent class and related into folder
- Remove force based agent behaviour
- Add agent update and render functions into box world update and render functions
- Add new behaviour to agent update function
- Use searching algorithm to create a path for the agents to follow
- Convert that path to a list of points that the agent can move to
- Add into update function that when called the agents position is updated to the next point in the path

What we found out:


c:\Users\ryanc\Documents\AI for games\COS30002 - 102564...

Kind: wall Search: DFS Status: ---

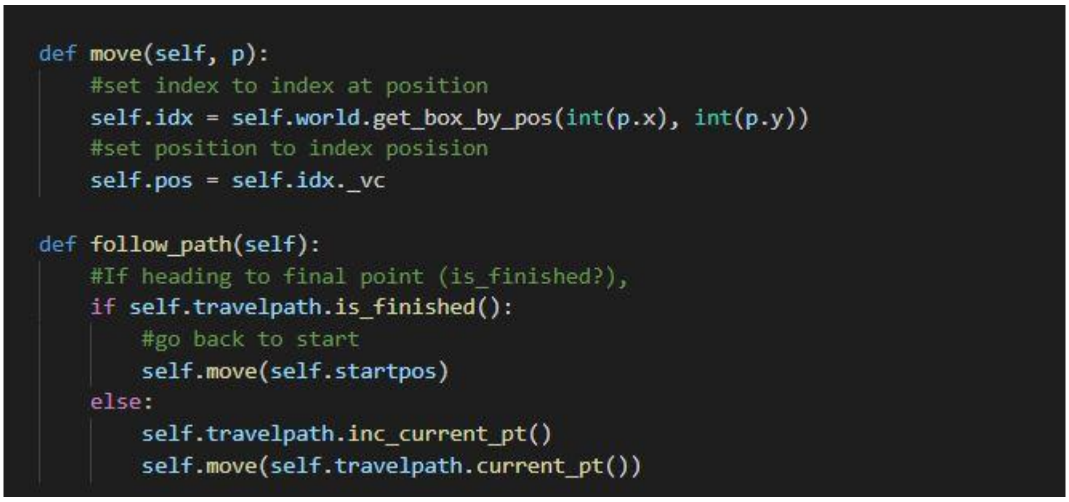
Each agent starts in one corner of the map. When the space bar is pressed update is called and the agents move to the next space in their path. Each agent moves along their own path towards the target, when they reach the target they go back to the place they started. The blue agents use the Astar algorithm while the red ones use dijkstra's algorithm.



```python
def update(self):
    #check for a target

    #if target
    if self.world.target != None:
        #self.travelpath.clear()
        #run a search
        if self.use_a_star == True:
            self.path = SearchAStar(self.world.graph, self.idx.idx, self.world.target.idx, 0)
        else:
            self.path = SearchDijkstra(self.world.graph, self.idx.idx, self.world.target.idx, 0)
        points = []
        print(self.path.path)
        #convert path into position
        for point in self.path.path:
            pos = Point2D()
            pos = self.world.boxes[point]._vc
            print(pos)
            points.append(pos)
        #create a path based on search
        self.travelpath.set_pts(points)
        print(self.travelpath.get_pts())
        #follow the path
        print(self.travelpath.current_pt())
        self.follow_path()
```

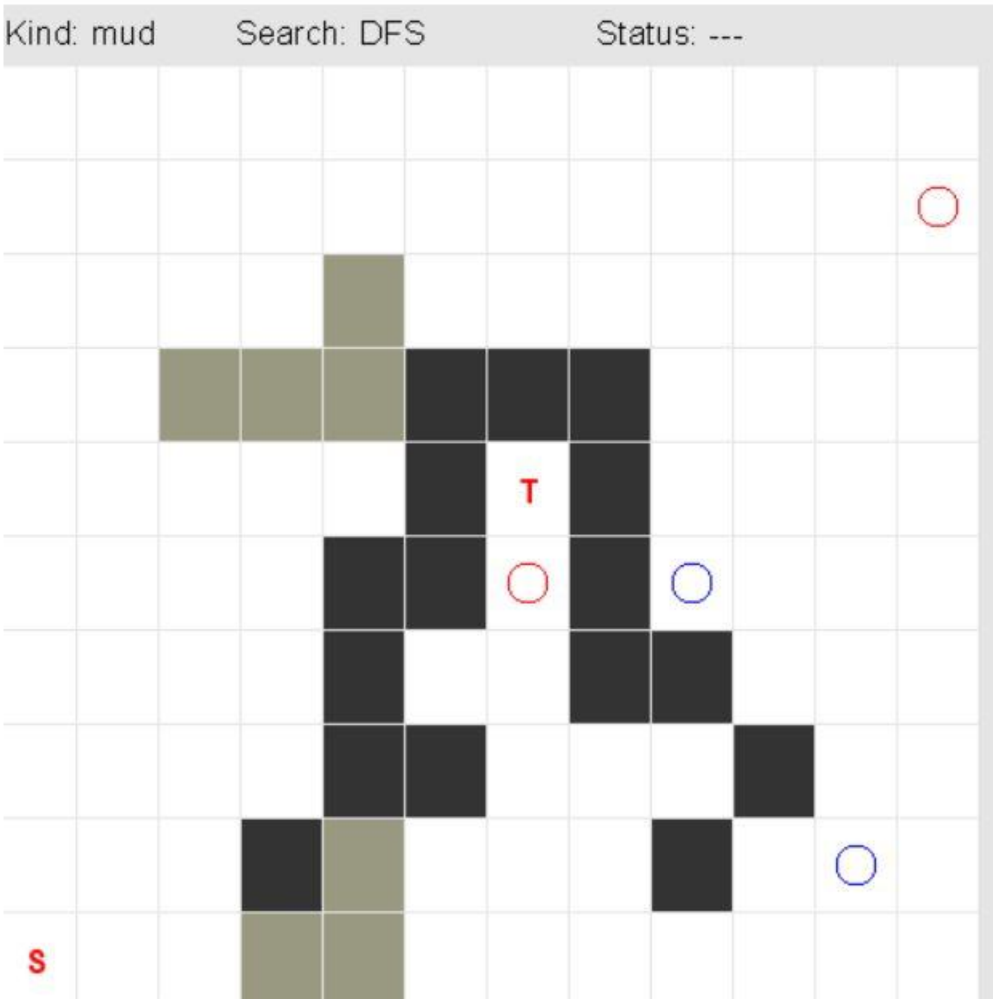
Each agent calculates their path every time they call update so they can always dynamically path find around new obstacles placed in their way.



```python
def move(self, p):
    #set index to index at position
    self.idx = self.world.get_box_by_pos(int(p.x), int(p.y))
    #set position to index posision
    self.pos = self.idx._vc

def follow_path(self):
    #If heading to final point (is_finished?),
    if self.travelpath.is_finished():
        #go back to start
        self.move(self.startpos)
    else:
        self.travelpath.inc_current_pt()
        self.move(self.travelpath.current_pt())
```

I adapted the follow path function from lab 9 to move the agents along the graph.



Kind: mud Search: DFS Status: ---