Spike: 06
Title: Navigation with Graphs

Author: Ben Holmes, 103024841

Goals / deliverables:
1. A game World that is divided into a larger number of navigation tiles, and corresponding larger navigation structure
2. 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).
3. Demonstrate multiple independent moving agent characters (at least four) that are able to each follow their own
independent paths.
4. Demonstrate at least two different types of agents that navigate the world differently

Technologies, Tools, and Resources used:

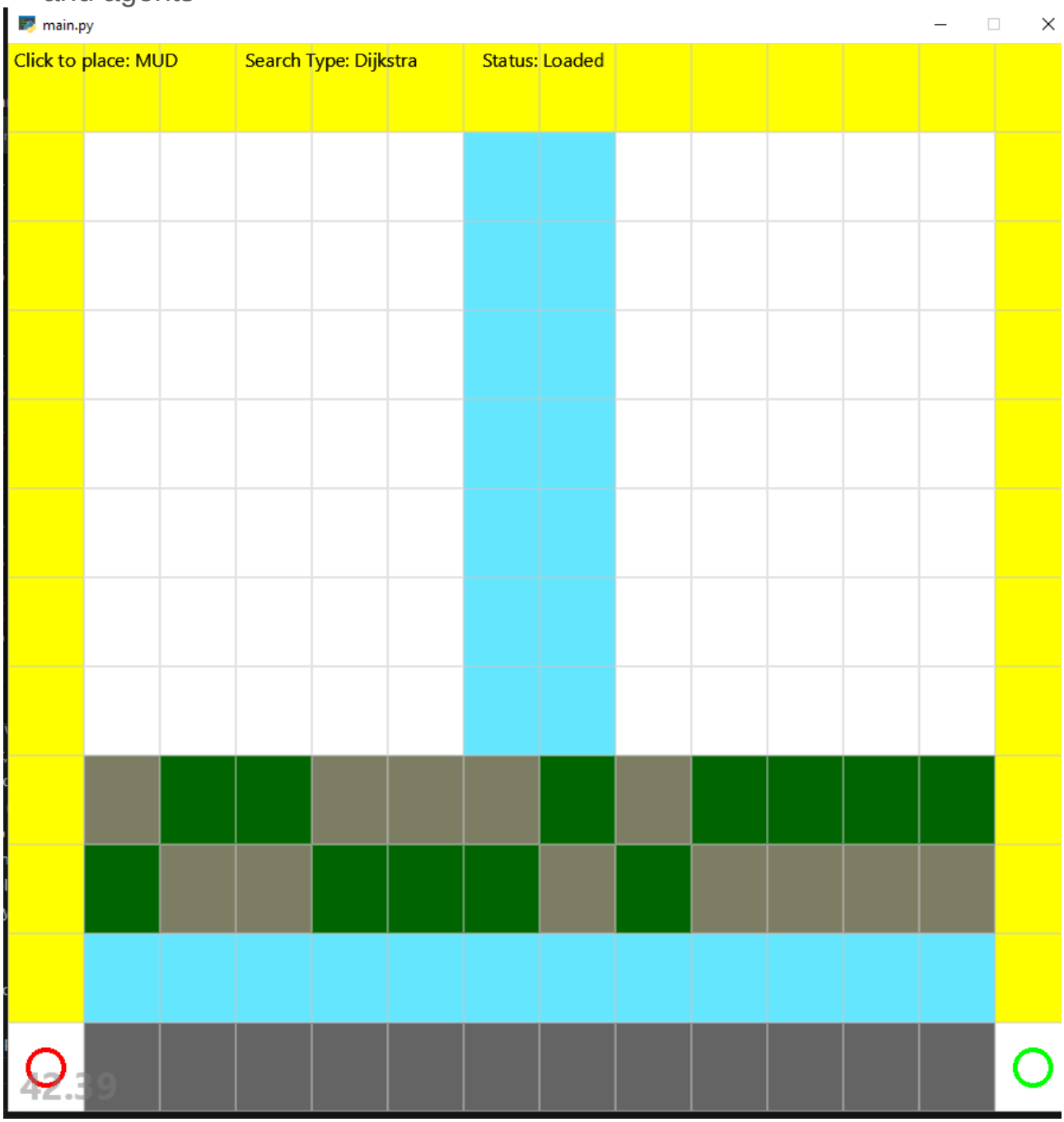
- Visual Studio Code
- Python 3.12.2
- pyglet

Tasks undertaken:

- Copied box_world code from task 5
- Adjusted it to have 6 tile types instead of 4, added Hill and Road and changed clear to Grass and changed mud to have harder traverse
- Adjusted the code to have 4 separate box_types in order to have 4 different paths generated on the same map. (these 4 are stored in box_types.py now which is attached)
- One box type is similar to the standard one given, one has better Hill traverse, one ignores terrain cost, and the last ignores terrain cost and walls
- Changed the code to create 4 separate nav graphs and paths via duplication of certain segments of code. This version is stored in Box_world OLD.py
- Added agent functionality that moved all the nav graph and path creation into the agent layer and had the box_world create 4 agents with the different box_types created earlier. The agent code is in

agent.py which was submitted along with this report (it was too many lines of code to use screenshots of)

- Created the map layout that would be used to show the working paths and agents



- Worked on the agents movement for a long time, trying to figure out how and where to set/use them
- Settled on moving 1/60 of the distance between the tiles along the path each frame (if you wish to run/test quicker you can change the value of steps in the agent.py file)

- There is an update function in game that triggers the update function in box world. The box world one removes previous stored rendered agents in its render_agents then cycles through all agents triggering their update functions and appending there render variable to the box_world's render_agents.

Box_World.py update

```
def update(self):
    for line in self.render_agents:
        try:
            line.delete() #pyglets Line.delete method is slightly broken
        except:
            pass
    for agent in self.agents:
        agent.update()
        self.render_agents.append(agent.render_agents)
```

- The agent updated function uses 2 counters, one registering the count of how many times the update has been triggered (counter) and the other counting what move its on (move). Every 60 counter increases it resets and increases move by one thus enabling it to read all of the moves in sequence. It calculates the render point by grabbing the coordinates of the start and end of the move (current and move +1) then gets the x_change and y_change, multiplies that by the counter divided by counter max steps and adds the current moves x or y to give it the offset needed to match the centre of the tiles.

Agent.py update

```
def update(self):
    steps = 60 # the value of 60 means it travels across an entire move over 60 frames/cycles (how many steps it takes between two moves)
    if self.path == None:
        return
    if self.counter == steps:
        self.counter = 1
        self.move += 1
    if self.move == self.path.path.__len__() - 1:
        return
    move_start = self.boxes[self.path.path[self.move]]
    move_end = self.boxes[self.path.path[self.move + 1]]
    x_change = move_end.center().x - move_start.center().x
    y_change = move_end.center().y - move_start.center().y
    self.x = x_change * (self.counter/steps) + move_start.center().x
    self.y = y_change * (self.counter/steps) + move_start.center().y
    self.counter += 1
    self.render_agents = pyglet.shapes.Circle(
                        self.x,
                        self.y,
                        5,
                        color=self.agent_color,
                        batch=window.get_batch("agents")
                    )
```

- This update function is triggered via the graphics on_draw function meaning it gets triggered every frame

```
@self.event
def on_draw():
    self.clear()
    self.batches["main"].draw()
    if self.cfg['TREE']:
        self.batches["tree"].draw()
    if self.cfg['PATH']:
        self.batches["path"].draw()
    if self.cfg['EDGES']:
        self.batches["edges"].draw()
    if self.cfg['NUMBERS']:
        self.batches["numbers"].draw()
    if self.cfg['AGENTS']:
        self.batches["agents"].draw()
    self.fps_display.draw()
    for label in self.labels.values():
        label.draw()
    from game import game
    game.update()
```

Game.py update:

```
def update(self):
    self.world.update()
```

- The agents have their move and counter variables reset every time the plan_path function is called meaning that each time a tile is changed or the space bar is pressed, the agents reset to their starting point.

Box_world.py:

```
def plan_path(self):
    '''Conduct a nav-graph search from the current world start node to the
    current target node, using a search method that matches the string
    specified in `search`.
    '''
    for line in self.render_path:
        try:
            line.delete() #pyglets Line.delete method is slightly broken
        except:
            pass
    for line in self.render_tree:
        try:
            line.delete() #pyglets Line.delete method is slightly broken
        except:
            pass
    for line in self.render_open_nodes:
        try:
            line.delete() #pyglets Line.delete method is slightly broken
        except:
            pass
    for agent in self.agents:
        agent.reset()
        agent.plan_path()
        self.render_path.append(agent.render_path)
        self.render_tree.append(agent.render_tree)
        self.render_open_nodes.append(agent.render_open_nodes)
```

Agent.py:

```
def reset(self):
    self.counter = 1
    self.move = 0
```

- I then updated the map to be bigger in order to satisfy the first deliverable.
- Also all agents used Dijkstra's search as the different terrains were enough to separate the agents and it was the most efficient search processing wise

What we found out:

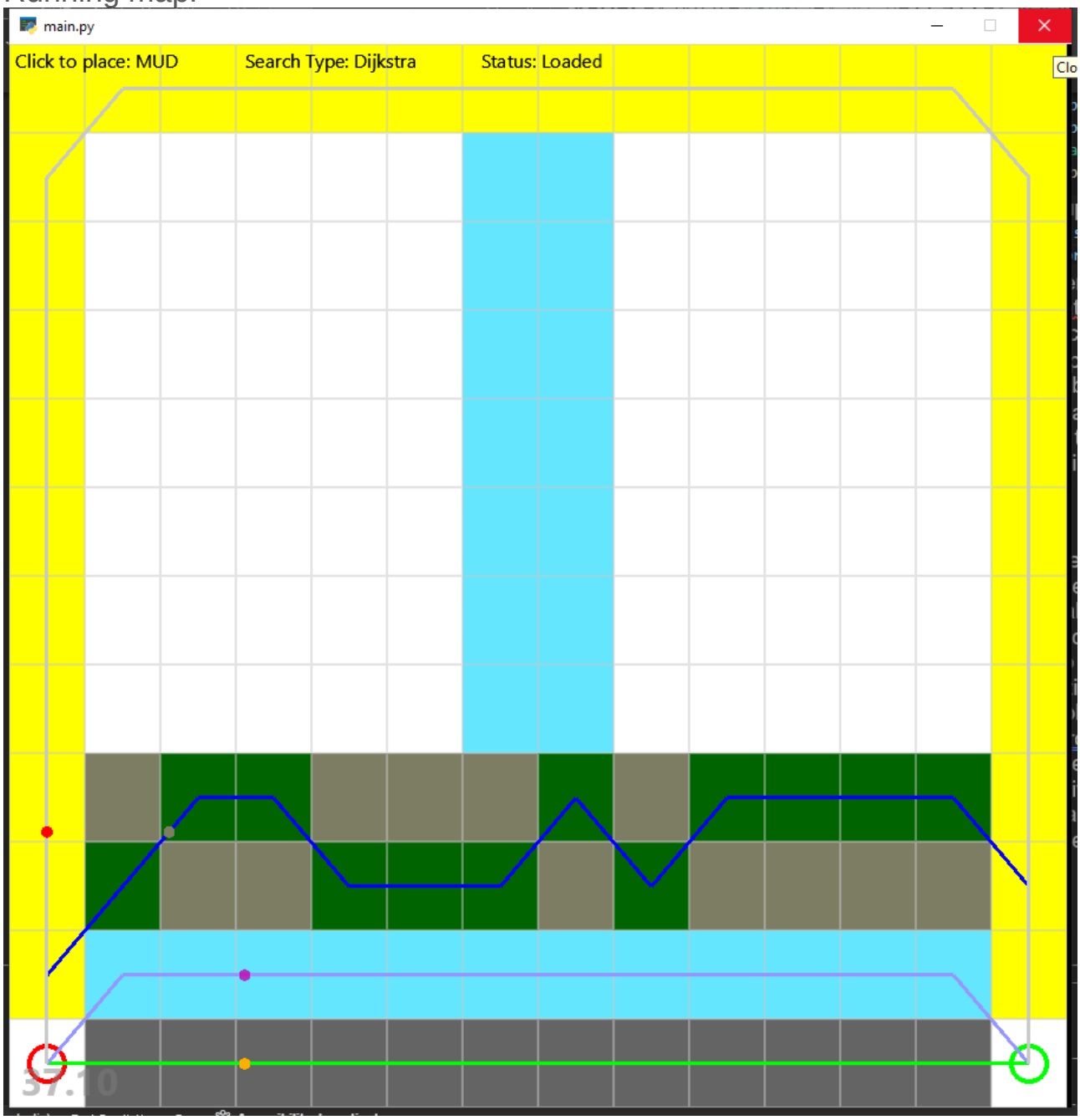
For the first deliverable the map was increased in size to 14x12 instead of 12x10, and the tiles of Hill and road were added.
In order to make those tiles work how I wanted to I had to adjust all the other tiles, such as changing clear to grass and changing grasses movement values to 2 so that road could be the cheapest value. There were a number of different iterations in order to get the values how I wanted.

This deliverable was needed for the topic as it added more complexity to the dynamic environment
For deliverable 2, each agent had a reset_navgraph and path_plan function in order to split the load and allow agents to plan their own path. This was needed as it allows for multiple different agents to start at different points or use different terrain costs enabling dynamic path planning.
For deliverable 3, each agent uses its update function to set a render position each time a frame is needed. This allows the moving agents needed by the topic.

For deliverable 4, each agent was given different box_types (this was done in the init of box_World in the agents variable) enabling differing path planning to demonstrate the dynamic environment better.

I had some difficulty with using pyglets clock function with how I had setup the agents and update methods, this is why the update function is called in the on_draw as that is called every frame so the movement is updated every frame. This can cause some minor issues around smooth movement as depending on how well the program is running its frame rate can dip down to 30 initially (at least on my machine) so the movement is not completely smooth

Running map:


main.py
Click to place: MUD
Search Type: Dijkstra
Status: Loaded