Spike: Task 18

Title: Navigation with Graphs

Author: Sam Huffer, 101633177

Goals / Deliverables

- Create a navigation graph simulation demonstrating:
  - A game world divided into tiles, and that supports a navigation graph structure.
  - Agent path planning based on current environment, using cost-based heuristics.
  - Four or more agents of at least two types that can follow their own independent paths.

Technologies, Tools, and Resources Used

- SublimeText (for editing, executing and testing the code)
- Learning materials on Canvas (for instructions and sample code)

Tasks Undertaken

- I started by copying the project from Task 17: Graphs, Paths and Search and the agent, weapon and projectile classes and the spike report from Task 16: Goal-Oriented Action Planning into the Task 18: Navigation with Graphs folder, stripping the spike report down to what was needed for this task.
- Next, I made the box world render a stationary agent, and got it calculating a path to the target rather than the start box doing so. I also added an input for the R key, having it randomise the position of the agent, and added the box world's resize code to ensure the agent stayed in its current box.
- I then got the agent to start moving along its path, having it move to the next node in the path, checking which box it is in every time it moves, and updating it when it enters a new box. I scaled the rate of movement along each axis by the ratio between the original width and height of the window to the current width and height of the window, in case the window gets resized.
- Next, I removed the start box and added more agents and had them seek out separate targets, with the R key now randomising the positions of the agents and the targets. While doing this, I noticed a bug where sometimes path nodes seem to get strapped to agents, making the path go all over the place.
- I modified the agents to wander, selecting a random box to move to and moving along the path to it until they reached their target, then moving onto a new target. I changed AStar to be the default search method, growing tired of having to change it manually. I also fixed a bug where I couldn't switch path lines off, and fixed the bug where sometimes path nodes seem to get strapped to agents. I didn't figure out what was causing it (the error output said it was a couple of methods in agent, but they were merely accessing the boxes' positions to read them, and shouldn't have been able to overwrite them since I had everything taking copies and reading those instead), but I did implement an effective countermeasure: boxes now have both _vc and position recording their positions. If something needs to access the position, it accesses _vc. If it doesn't match position, it will be overridden with a copy of position. I also tweaked the resize code so that it handles the agent's current node position that it's travelling to as well.

- I changed wall blocks to render as circles, make them look like agents aren't intersecting with them when moving across boxes diagonally. I added the weapons back in and got the radii of their effective ranges scaling according to screen size.
- I started reworking the field of view code to render properly in the new context, but ended up abandoning it since it was no longer working adequately. Soldier agents now use a circle centred 65% of its radius in front of the soldier. Anything within that circle is considered within the soldier's awareness. Soldiers now pursue the closest fugitive they're aware of, wandering if they can't see any. When pursuing, they follow a path to the fugitive. If the fugitive moves into a new box, the soldier recalculates the path. If the fugitive is in the same box as the soldier, the soldier moves directly towards it.
- Next, I modified the soldier's aiming of weapons to use the target enemy's path rather than its current velocity, since velocity is no longer being used. I copied the projectile processing code from the original world class to the box world class and tweaked it to suit the new scenario. I also made some minor tweaks to shooting so that it would work in the current context.
- I noticed that agents were not recalculating their paths if I placed new wall boxes in the box world, instead treating them as if they were the same box type as they were when the simulation began. I found the error (code was trying to "add" the wall box to box world's list of walls rather than "append" it) and fixed that. I also changed that section of code so that boxes' kinds could not be changed while an agent was inside that box, and removed some leftover code pertaining to the original target for calculating paths using the search algorithms.
- Lastly, I noticed that very occasionally, for seemingly no apparent reason, the soldier would get stuck trying to move to the centre of its current box at the end of a path, but be unable to do so. If a fugitive would come past, that would break it out of its being stuck. So that it wouldn't have to wait for a fugitive, I added several checks: if the soldier has been in the same box for 3 seconds, it automatically gets a new path for wandering; when calculating the wander path, it cannot pick its current box as the wander target; and when following the path, if the path or path node it's moving to are None, it automatically gets a new path for wandering. I haven't observed this bug since and the automatically resets to a wandering path are working, which is encouraging, but I don't know that the bug isn't still there.

Instructions for Operating the Code

- Placing blocks:
    - Left mouse click: change a box's kind to the currently selected kind.
    - 1: mouse clicks now clear blocks.
    - 2: mouse clicks now place mud.
    - 3: mouse clicks now place water.
    - 4: mouse clicks now place walls.
- Search parameters:
    - D: scroll through the list of ways of calculating diagonals.
    - M: scroll forward through the list of search algorithms.
    - N: scroll backwards through the list of search algorithms.
- Display options:
    - B: alternate thickness of box lines.
    - C: toggles markers of the centre of boxes.
    - E: toggles displaying of movement network edges.
    - L: toggles box labels.
    - O: toggles highlighting of agents' optimal paths in red.
    - T: toggles display of graph options that were considered but did not end up being the optimal path.
    - U: toggles circle markers of boxes considered during the search.
- P: (un)pause the simulation
- R: randomise the position of the agents and targets.

Code Snippets

```python
def update(self, delta):
    if self.health <= 0:
        self.position_in_random_box()
        self.get_wander_path()
        self.health = self.start_health
        self.hit_time = None

        for agent in self.world.agents:
            if agent.agent_type is not self.agent_type and agent.target_enemy == self:
                agent.target_enemy = None
                agent.get_wander_path()

    if self.last_new_box_time is not None and (datetime.now() - self.last_new_box_time).total_seconds() > 3:
        self.last_new_box_time = datetime.now()
        self.get_wander_path()
        print("overriding timed out path with wander path")

    if self.hit_time is not None and (datetime.now() - self.hit_time).total_seconds() > 0.1:
        self.hit_time = None

    if self.agent_type == "soldier":
        self.update_soldier(delta)
    elif self.agent_type == "fugitive":
        self.update_fugitive(delta)

    self.update_heading()
    box = self.box
    self.box = self.world.get_box_by_pos(int(self.pos.x), int(self.pos.y))

    if box is not self.box:
        self.last_new_box_time = datetime.now()
```

Figure 1: the agents' new update method for operating in the box world.

```python
def update_soldier(self, delta):
    self.awareness_pos = Vector2D(self.awareness_radius * 0.625, 0)
    self.awareness_pos = self.world.transform_point(self.awareness_pos, self.pos, self.heading, self.side)
    self.look(self.world.agents)
    if self.path == None: self.get_wander_path()
    if self.see_target:
        self.movement_mode = "Attack"
        if self.max_speed == self.max_speed_standard: self.max_speed *= 1.25
        if self.target_enemy == None or self.target_enemy.distance(self.pos) > self.awareness_radius + self.target_enemy.radius:
            self.target_enemy = None
            for agent in self.world.agents:
                if agent is not self and agent.distance(self.pos) < self.awareness_radius + agent.radius and (self.target_enemy == None or self.distance(self.target_enemy.pos) > self.distance(agent.pos)):
                    self.target_enemy = agent
        if self.path is not None and len(self.path.path) > 0 and self.target_enemy is not None and self.target_enemy.box is not self.target:
            self.target = self.target_enemy.box
            self.plan_path(search_modes[self.world.window.search_mode], self.world.window.limit)
        if self.choose_weapon():
            if self.see_target:
                self.movement_mode = 'Attack'
                if self.last_aware_time is not None: self.last_aware_time = None
            elif self.last_aware_time == None: self.movement_mode = 'Patrol'
            elif (datetime.now() - self.last_aware_time).total_seconds() > 10:
                self.last_aware_time = None
                self.movement_mode = 'Patrol'
            else: self.movement_mode = 'Resume Attack'
            if self.weapons[0].rounds_left_in_magazine == 0 and (datetime.now() - self.last_shot).total_seconds() <= self.weapons[0].reload_time: self.combat_mode = 'Reloading'
            else:
                if self.weapons[0].rounds_left_in_magazine == 0:
                    self.weapons[0].rounds_left_in_magazine += self.weapons[0].magazine_size
                    self.weapons[0].magazines_left -= 1
                if (datetime.now() - self.last_shot).total_seconds() <= self.weapons[0].cooldown: self.combat_mode = 'Weapon is Loading Next Round'
                elif self.movement_mode == 'Attack': self.combat_mode = 'Aiming'
                elif self.movement_mode == 'Patrol': self.combat_mode = 'Ready'
            if self.movement_mode == 'Attack' and self.combat_mode == 'Aiming':
                if len(self.weapons[0].projectile_pool) == 0: self.combat_mode = 'No Projectiles Pooled'
                elif self.target_enemy is not None:
                    target = self.aim_shot(self.target_enemy)
                    if target is not None:
                        self.combat_mode = 'Shooting'
                        self.last_shot = datetime.now()
                        self.weapons[0].rounds_left_in_magazine -= 1
                        self.shoot(target)
        else: self.world.change_weapons(self)
    else:
        self.movement_mode = "Patrol"
        if self.max_speed is not self.max_speed_standard: self.max_speed = self.max_speed_standard
    if self.target_enemy is not None and self.target == self.box: self.follow_target_enemy(delta)
    else: self.follow_graph_path(delta)
```

Figure 2: the soldier-specific update method. Much of it is identical to update_shooter() from Task 16, barring the parts that were changed so it worked in the box world.

```python
def update_fugitive(self, delta):
    if self.path == None:
        self.get_wander_path()

    self.follow_graph_path(delta)
```

Figure 3: the fugitive-specific update method. Currently, fugitives just wander.

```
def get_wander_path(self):
    target = self.world.boxes[randrange(0, len(self.world.boxes))]

    while target.kind == "X" or target == self.world.get_box_by_pos(int(self.pos.x), int(self.pos.y)):
        target = self.world.boxes[randrange(0, len(self.world.boxes))]

    if self.target is not None and self.target in self.world.targets:
        self.world.targets.remove(self.target)

    self.target = target
    self.world.targets.append(target)

    self.plan_path(search_modes[self.world.window.search_mode], self.world.window.limit)
    print("Agent path: " + self.path.report(verbose=3))
def plan_path(self, search, limit):
    '''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`.
    '''
    cls = SEARCHES[search]
    self.path = cls(self.world.graph, self.box.idx, self.target.idx, limit)

    if len(self.path.path) > 0:
        self.current_node_box = self.world.boxes[self.path.path[0]]
        self.current_node_pos = self.current_node_box.get_vc("agent.plan_path()").copy()
```

Figure 4: methods for getting a random path, and calculating the path to any target.

```
def look(self, agents):
    self.see_target = False

    for agent in agents:
        if agent.agent_type is not self.agent_type and agent.distance(self.awareness_pos) < self.awareness_radius:
            self.see_target = True
            return
```

Figure 5: the soldier's look method for checking if a fugitive is within its awareness range.

```
def follow_graph_path(self, delta):
    if self.path == None or len(self.path.path) == 0 or self.current_node_pos is None:
        self.get_wander_path()
        print("overriding none path with wander path")

    path = self.path.path

    to_current_node = (self.current_node_pos - self.pos).normalise() * self.max_speed * delta

    self.pos.x = self.pos.x + (to_current_node.x * self.world.scale_vector.x)
    self.pos.y = self.pos.y + (to_current_node.y * self.world.scale_vector.y)

    if self.distance(self.current_node_pos) < self.radius * 0.5:
        if len(path) > 1:
            path.remove(path[0])
            self.current_node_box = self.world.boxes[path[0]]
            self.current_node_pos = self.current_node_box.get_vc("agent.follow_graph_path()").copy()
        else:
            self.path = None
            self.current_node_box = None
            self.current_node_pos = None
def follow_target_enemy(self, delta):
    to_target_enemy = (self.target_enemy.pos - self.pos).normalise() * self.max_speed * delta

    self.pos.x = self.pos.x + (to_target_enemy.x * self.world.scale_vector.x)
    self.pos.y = self.pos.y + (to_target_enemy.y * self.world.scale_vector.y)
```

Figure 6: the agent's movement methods.

```
def get_target_path_measurements(self, target):
    result = {}
    total_dist = 0

    boxes = self.world.boxes

    # target to current node
    if target.current_node_box is not None:
        dist = (target.current_node_box.get_vc("agent.get_target_path_measurements(), target to current node") - target.pos).length()
    else:
        return None

    total_dist += dist
    result[0] = {"box": target.current_node_box, "dist": total_dist}

    if target.path is not None:
        path = target.path.path

        # current node to first node in path
        dist = (boxes[path[0]].get_vc("agent.get_target_path_measurements(), current node to first node in path, path node") - target.cu
        total_dist += dist
        result[1] = {"box": boxes[path[0]], "dist": total_dist}

        # first node in path to last node in path
        for i in range(1, len(path)):
            dist = (boxes[path[i]].get_vc("agent.get_target_path_measurements(), path nodes, " + str(i)) - boxes[path[i-1]].get_vc("agen
            total_dist += dist
            result[i+1] = {"box": boxes[path[i]], "dist": total_dist}

    return result

def get_future_pos_on_path(self, target, path_measurements, speed, time):
    pos = Vector2D()
    dist = speed * time

    if path_measurements is not None:
        i = len(path_measurements) - 1

        while i >= 0:
            if dist > path_measurements[i]["dist"]:
                pos = path_measurements[i]["box"].get_vc("agent.get_future_pos_on_path()").copy()
                dif = dist - path_measurements[i]["dist"]
                pos += target.heading * dif

                return pos

            i -= 1

    return target.pos + (target.heading * dist)
```

Figure 7: the methods aim() uses to predict the target's future position in the box world environment. The cut lines of code are "dist = (boxes[path[0]].get_vc("agent.get_target_path_measurements(), current node to first node in path, path node") - target.current_node_box.get_vc("agent.get_target_path_measurements(), current node to first node in path, current node")).length()" and "dist = (boxes[path[i]].get_vc("agent.get_target_path_measurements(), path nodes, " + str(i)) - boxes[path[i-1]].get_vc("agent.get_target_path_measurements(), path nodes, " + str(i-1))).length()".

```
def resize(self, cx, cy):
    self.cx, self.cy = cx, cy # world size
    self.wx = (cx-1) // self.nx
    self.wy = (cy-1) // self.ny # int div - box width/height

    self.scale_vector = Point2D(cx / self.original_cx, cy / self.original_cy)
    self.scale_scalar = (self.scale_vector.x + self.scale_vector.y) / 2

    for i in range(len(self.boxes)):
        # basic positions (bottom left to top right)
        x = (i % self.nx) * self.wx
        y = (i // self.nx) * self.wy
        # top, right, bottom, left
        coords = (y + self.wy -1, x + self.wx -1, y, x)
        self.boxes[i].reposition(coords)

    for agent in self.agents:
        agent.radius = agent.radius_standard * self.scale_scalar
        agent.avoid_radius = agent.avoid_radius_standard * self.scale_scalar
        agent.awareness_radius = agent.awareness_radius_standard * self.scale_scalar
        agent.pos = agent.box._vc

        if agent.current_node_box != None:
            agent.current_node_pos = agent.current_node_box._vc

    for weapon in self.weapons:
        weapon.effective_range = weapon.effective_range_standard * self.scale_scalar
```

Figure 8: box world's modified resize() method, which now accounts for agents and weapon ranges in addition to boxes.

```
@self.event
def on_mouse_press(x, y, button, modifiers):
    if button == 1: # left
        box = self.world.get_box_by_pos(x,y)
        if box:
            for agent in self.world.agents:
                if box == self.world.get_box_by_pos(int(agent.pos.x), int(agent.pos.y)):
                    print("Illegal change. That box has agents in it. Try again when they've all moved to new boxes.")
                    return

            start_kind = box.kind
            box.set_kind(self.mouse_mode)

            if box.kind != start_kind:
                print("box change")
                if start_kind == "X":
                    self.world.walls.remove(box)
                elif box.kind == "X":
                    self.world.walls.append(box)

            self.world.reset_navgraph()
            self.plan_path()
            self._update_label('status','graph changed')
```

Figure 9: the main class's modified on_mouse_press() method, which now checks if a box is a legal target for changing it's kind.

In-Simulation Screenshots

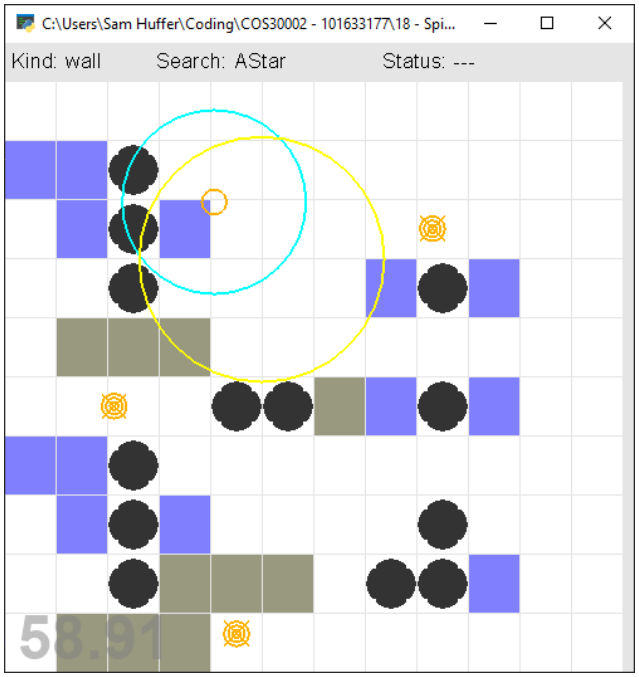

Figure 10: all agents wandering about, minding their own business.

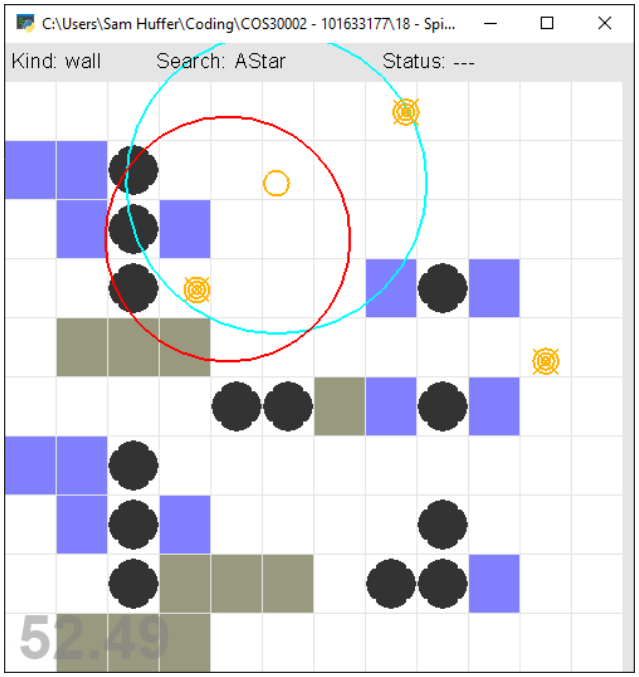

Figure 11: the soldier agent has spotted a fugitive wandering about, and is now following a path towards it.

What I Found Out

- When using boxes, I either had to scale boxes to be 1:1 upon resizing the window and to all fit within the window, or I had to scale everything separately for x and y axes where possible, and fudge it as best as I could when using something like a circle where you can't scale it on the x and y axes separately. The first was the ideal scenario, but things went weird when I tried scaling the boxes, so instead I ended up scaling everything else on the x and y axes, fudging it where necessary.
- I had several considerations to account for when having agents move using a path rather than using the earlier force-based model, such as:
    - Making the soldier pursue the target directly when they were in the same box as the target, but the target wasn't at the same coordinates as the centre of the box.
    - When not using force-based, heading and side cannot derive from velocity, as velocity isn't being used; I had to derive them from the vector to the current target instead. It looked a little jerkier and could probably be made smoother, but that wasn't strictly necessary for this task, so I left it alone.