Task: Task 21

Title: Custom Project

Author: Sam Huffer, 101633177

Goals / Deliverables

- A customizable box world.
    - Corridors formed from wall tiles.
    - Patrolled by a squad of soldier agents.
        - Using group behaviours adapted for graph-based navigation.
        - Attack fugitives on sight.
        - If fugitives flee beyond soldiers' awareness, they scout the immediate area, resume attacking if they find them, resume patrolling if they don't.
        - If overwhelmed by fugitives, one soldier returns to base to lead reinforcements up to max squad size back to the rest of the soldiers.
        - Squad respawns if all soldiers die.
    - Populated with fugitive agents.
        - Stay hidden until they see the soldiers, then attack.
        - On attack, fear increases. If it gets too high, fugitives flee.
        - Respawn at random points on map when they die.
    - All agents flee explosives they're in range of until they finish detonating.

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 18: Navigation with Graphs and my custom project plan into the Task 20: Custom Project folder, stripping the spike report down to what was needed for the custom project.
- Next, I devised a larger map for the agents to wander around in, making a blank 30 x 30 map text file, designing it in the simulation itself, and copying it over to the text file. To adapt the mobile classes to it, I reduced the size of the agents, made the projectiles split their movement into several steps to ensure they don't jump over agents or walls they're supposed to hit, modified the speed of projectiles so that they don't move too fast but not so slow that the agents can overtake them and set them off, and reduced the inaccuracy margin of the hand-gun, hand grenade, and shotgun.
- I modified the soldiers' code so that while patrolling, the lead soldier would decide where the soldiers were going, and that the rest would calculate their own paths to the same location, recalculating if the lead soldier's target changed. However, I noticed that occasionally the sol-diers would move "backwards" then continue along their path when they recalculated their path to the lead soldier's target, so I also tweaked agent.plan_path() so that when a path was calculated, if the distance between the soldier and the second waypoint was less than the distance between the first and second waypoints, the first would be discarded and the soldier would just head to the second.

- I changed the read-from-file code to read the patrol waypoint box numbers from the map file and store the corresponding boxes in a list of waypoints in the box world class. While doing so, I got sick of the low framerate, I changed box.draw() to not fill the circle rendered for walls, but just do a thick outline, saving iterations of circle outline drawing and increasing the framerate.
- I altered the agent setup code so that the soldier leader would have its target set at the start and plan a path to the first waypoint, prompting the rest of the soldiers to follow it. Then I reorganised the various conditions for planning a new path into one method that could manage target selection and path planning for all of its preceding code, and replaced them with a call to that method. This way, the target selection and call of self.plan_path() is all in one location and can be more easily modified as needed.
- I ran into errors where target_enemy was None when it shouldn't have been; I put it down to a misalignment between how look() calculated if the agent could see an enemy, and how update_soldier() got the closest enemy. While testing, I also removed explosive weapons from the list of weapons, as explosives weren't working and just kept detonating behind the soldier.
- I noticed that after soldiers attacked, soldiers would sometimes patrol in the reverse direction, so I started created a directional map for soldiers to follow, where each box pointed to the next box in the patrol. However, I realised if I was going to implement it, it would require either the simulation's customizability to be disabled, or functionality to allow the changing of a tile's preferred direction for soldier agents, therefore, I decided against using a directional map. Instead, I decided to use a technique similar to how the agent bypasses redundant nodes on the path, using a path-length calculation method similar to what's used for the predictive calculations required in aiming and shooting. However, this resulted in the soldier agent somehow bouncing back and forth between the first two waypoints of the patrol, and not progressing any further. The solution I eventually arrived at was to make waypoints a line of boxes rather than single boxes, such that when the lead soldier moved backwards along the patrol path while attacking, if it triggered the boxes, it would decrement its waypoint so that when it finishes attacking, it knows which one is the next one along the patrol path. However, to retain customizability of the box world, this also required keyboard controls for customizing waypoints. To implement all of this, I first changed the code so that the box world reads in all boxes for a waypoint, then added a waypoint class that lists all boxes within the waypoint. I then added drawing code for displaying waypoints while they're being edited, inputs for editing the waypoints, and a method to handle adding and removing boxes from a waypoint.
- I added methods for switching between waypoints both when the soldiers are patrolling, and when they are not, incrementing and decrementing the index of the current and last waypoints according to which waypoints are triggered. I spotted some weird errors with the waypoint editing code in that on every number, clicking a box to add it to a waypoint would attempt to add or remove it from the correct box, but display "9" as its waypoint, and waypoints read from file would not be removable. I fixed this by reworking BoxWorld.edit_waypoint_node() to perform checked based on the waypoint number of the node passed to it and BoxWorld.active_waypoint, rather than searching through waypoints to check if the node was present in their lists of nodes or not. I also tweaked the rendering of the soldier leader's path to highlight path nodes with circles, so as to distinguish between it and other paths.
- I noticed that sometimes a soldier spots then loses a fugitive, but still pursues the fugitive despite losing it. I fixed this by altering the code for looking and checking which enemy agent in range is the closest such that it first checks if the enemy agent is within the soldier's range

as the crow flies. Then, gets a path to it and checks if the path does not extend outside the awareness range. If even a single node does, due to a wall, for example, then the soldier agent ignores it.

- I encountered a bug where if multiple soldiers are present and they spot a fugitive, they appear to just aggressively judge it rather than attack it. Looking into it, they seemed to be changing weapons repeatedly. Therefore, I modified agents to use a single weapon with ridiculously high ammo so that it effectively won't run out, and when it does it just refills automatically. Even then, I was still having issues with even two soldiers attacking at once. I added a print statement to check who they were attacking, as it looked like they were attacking each other; they were, as the enemy selection code only checked that the nearest agent wasn't themselves, rather than an agent of the same agent type. I altered that check, removing that bug.
- Here, I made some additional minor tweaks: not allowing projectiles to hit agents of the same type as the agent that fired it, rather than ignoring only their shooters until the projectile was out of the shooter's radius; stopping soldier agents from moving if doing so would overlap them with a soldier agent of higher rank (i.e. an index in BoxWorld.soldiers closer to zero), or if they're pursuing an enemy agent and moving would put them inside a wall; added inputs for toggling awareness ranges and weapon ranges on and off; and changed soldier agents to green to better differentiate between soldiers and fugitives.
- I modified fugitives to stay stationary until they spot a soldier, merely updating their heading to face the closest soldier, and *then* move and attack the soldier they spotted, using their own weapons to shoot them. I also updated fugitives and soldiers to have comparable health, and fugitives to have a fear measure that kicks in inversely proportionate to the distance between a fugitive and its attack target, causing the fugitive to flee when the fear gets too great. When it encounters a "higher-ranking" fugitive or reaches the end of its fleeing path, it sits still and starts reducing its fear over time.
- I did some reorganising and refactoring of update_soldier(), as some of its if statements seemed unnecessary and like they were duplicating functionality unnecessarily. Then I added another menu togglable by the spacebar: on clicking an empty, non-wall box, a new fugitive agent is spawned in that box. For this, I added the checks, UI elements, and functionality appropriate for spawning new fugitive agents.
- I implemented scouting behaviour for the soldiers: When they kill or lose a fugitive, they get paths to random boxes within 1.5 times the lead soldier's awareness radius, and travel there. When they get there, they stay there and look around. When the lead soldier gets to theirs, if none of them have spotted anything, it gets a new patrol path, then gets the other soldiers to do the same.
- I altered fugitives to be able to shoot while fleeing, and added a random element to their fleeing triggering, their likelihood of triggering being proportionate to their fear level. I changed soldiers to be killable, and added health bars for all agents, and removed the ability for fugitives spawned by player to respawn. I also did some minor reorganising of update_fugitive() akin to what was done to update_soldier(), and updated look() to just return the Boolean value rather than assigning it to see_target, and renamed look() as see_target().
- I added functionality for soldiers to be respawned at their starting points if all soldiers die, and for them to head to waypoint 0. I modified mouse clicks to manage agents generally rather than just spawning fugitives: if the clicked box has agents, they're destroyed. If it doesn't and it's not a wall, a fugitive spawns instead.

- I added a new box kind, “base”, to indicate where soldiers spawn, and implemented functionality for box editing to be able to turn boxes into bases and vice versa, provided the maximum number of bases hasn’t already been reached and the base box being changed isn’t the last one. With this, I also tied the number of soldiers respawned to the number of bases.
- I added the functionality for the last soldier in the squad to seek out reinforcements when the squad is overwhelmed. For this, I reconfigured soldier spawning to work when spawning soldiers at the start of the simulation, when they’re all dead, and when a soldier is seeking reinforcements, and to allocate names according to what letters are available yet nearest the start of the alphabet.
- I disabled the ability to change diagonal calculation type, search type, or search depth limit, restricting diagonal type to “max”, search type to A*, and depth limit to none.
- I noticed that the handling of waypoint triggering while soldiers weren’t patrolling had vanished, so I put that back into update_soldier(). I also added handling of waypoint changes when the lead soldier dies and its path to the new lead soldier would go through a waypoint.
- Lastly, I reorganised and split some soldier and fugitive functionality to make more readable, disabling pre-spawned respawnable fugitives to make demonstrations easier, and removed unused code, comments and discarded changes from all python files, and files leftover from past tasks.

Controls

- M: alternate between placing blocks, managing agents, and modifying waypoints.
- Managing agents:
  - Left mouse click: if the box is occupied, destroy all agents in the box. If the box is unoccupied and not a wall box, spawn a fugitive in the selected box.
- Placing blocks:
  - Left mouse click: change a box’s kind to the currently selected kind.
  - 1: mouse clicks now clear blocks (there must be at least one soldier base).
  - 2: mouse clicks now place mud.
  - 3: mouse clicks now place water.
  - 4: mouse clicks now place walls.
  - 5: moves clicks now place soldier bases, with a maximum of 9.
- Modifying waypoints:
  - [0-9]: select a waypoint.
  - Left mouse click: toggle whether a box is in the currently selected waypoint. Cannot add a box to a waypoint if it belongs to another waypoint.
- Display options:
  - B: alternate thickness of box lines.
  - C: toggles markers of the centre of boxes.
  - E: toggles displaying of movement network edges.
  - F: toggles fugitives’ awareness ranges.
  - L: toggles box labels.
  - O: toggles highlighting of agents’ optimal paths in red.
  - S: toggles soldiers’ awareness ranges.
  - W: toggles weapons’ effective ranges.
- P: (un)pause the simulation.

State Diagrams

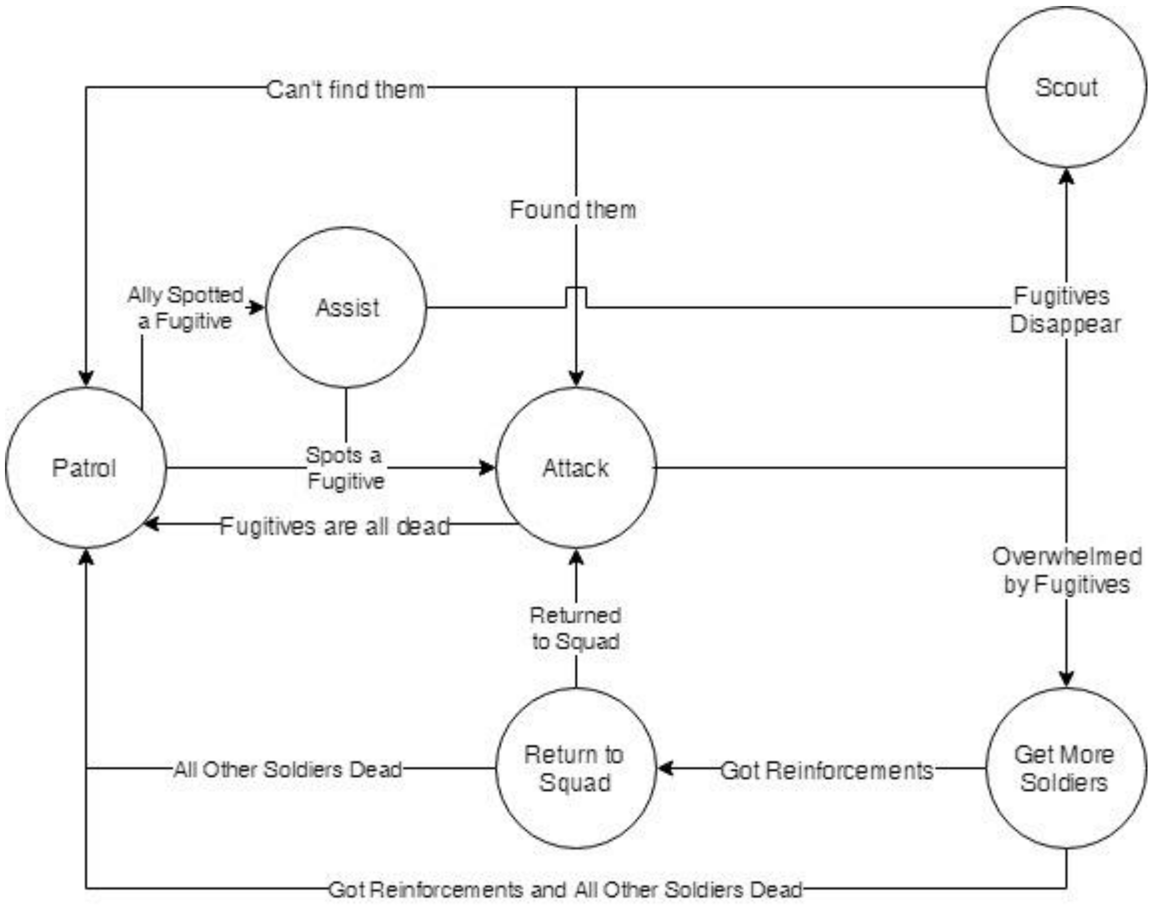


Figure 1: the final state diagram for the soldier agents.

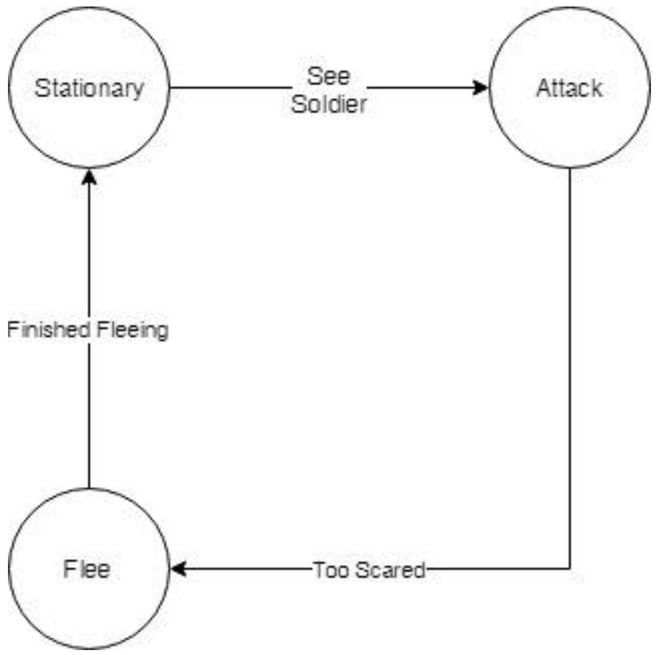


Figure 2: the final state diagram for the fugitive agents.

UML Class Diagram

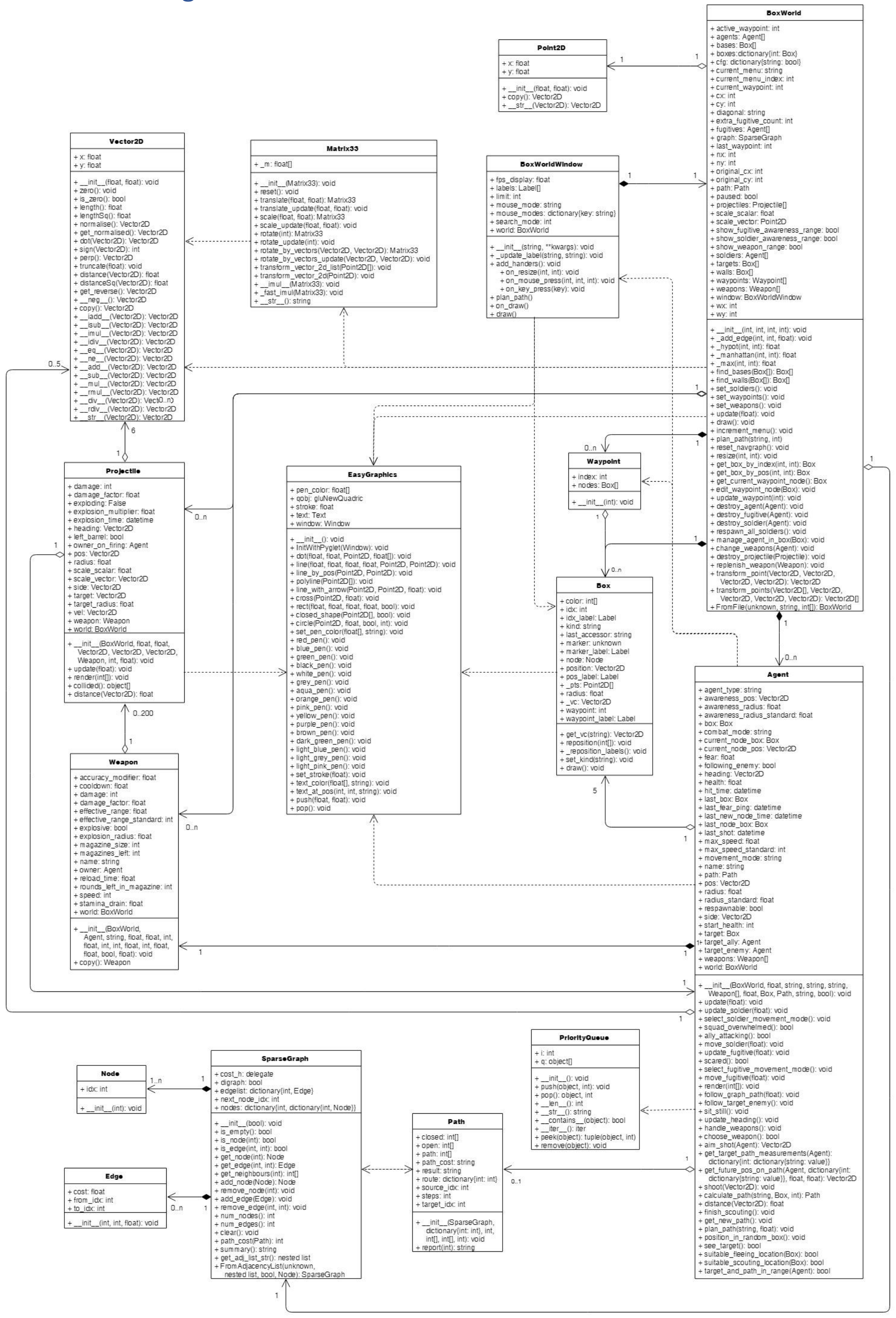
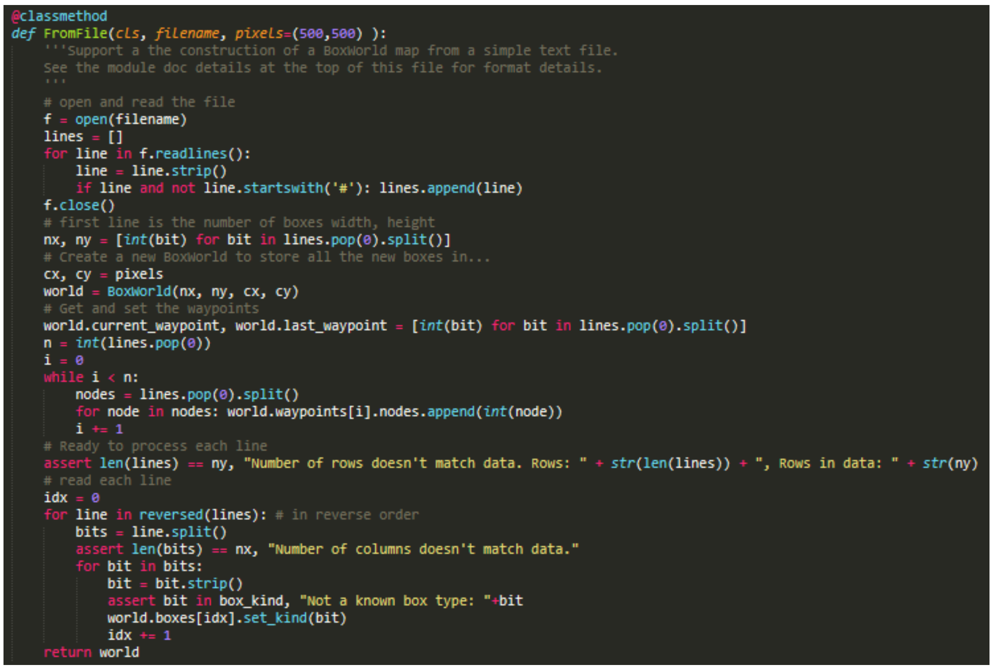

BoxWorld
+ active_waypoint: int
+ agents: Agent[]
+ bases: Box[]
+ boxes:dictionary{int: Box}
+ cfg: dictionary{string: bool}
+ current_menu: string
+ current_menu_index: int
+ current_waypoint: int
+ cx: int
+ cy: int
+ diagonal: string
+ extra_fugitive_count: int
+ fugitives: Agent[]
+ graph: SparseGraph
+ last_waypoint: int
+ nx: int
+ ny: int
+ original_cx: int
+ original_cy: int
+ path: Path
+ paused: bool
+ projectiles: Projectile[]
+ scale_scalar: float
+ scale_vector: Point2D
+ show_fugitive_awareness_range: bool
+ show_soldier_awareness_range: bool
+ show_weapon_range: bool
+ soldiers: Agent[]
+ targets: Box[]
+ walls: Box[]
+ waypoints: Waypoint[]
+ weapons: Weapon[]
+ window: BoxWorldWindow
+ wx: int
+ wy: int
+ __init__(int, int, int, int): void
+ _add_edge(int, int, float): void
+ _hypot(int, int): float
+ _manhattan(int, int): float
+ _max(int, int): float
+ find_bases(Box[]): Box[]
+ find_walls(Box[]): Box[]
+ set_soldiers(): void
+ set_waypoints(): void
+ set_weapons(): void
+ update(float): void
+ draw(): void
+ increment_menu(): void
+ plan_path(string, int)
+ reset_navgraph(): void
+ resize(int, int): void
+ get_box_by_index(int, int): Box
+ get_box_by_pos(int, int): Box
+ get_current_waypoint_node(): Box
+ edit_waypoint_node(Box): void
+ update_waypoint(int): void
+ destroy_agent(Agent): void
+ destroy_fugitive(Agent): void
+ destroy_soldier(Agent): void
+ respawn_all_soldiers(): void
+ manage_agent_in_box(Box): void
+ change_weapons(Agent): void
+ destroy_projectile(Projectile): void
+ replenish_weapon(Weapon): void
+ transform_point(Vector2D, Vector2D, Vector2D, Vector2D): Vector2D
+ transform_points(Vector2D[], Vector2D, Vector2D, Vector2D, Vector2D): Vector2D[]
+ FromFile(unknown, string, int[]): BoxWorld
Point2D
+ x: float
+ y: float
+ __init__(float, float): void
+ copy(): Vector2D
+ __str__(Vector2D): Vector2D
Vector2D
+ x: float
+ y: float
+ __init__(float, float): void
+ zero(): void
+ is_zero(): bool
+ length(): float
+ lengthSq(): float
+ normalise(): Vector2D
+ get_normalised(): Vector2D
+ dot(Vector2D): Vector2D
+ sign(Vector2D): int
+ perp(): Vector2D
+ truncate(float): void
+ distance(Vector2D): float
+ distanceSq(Vector2D): float
+ get_reverse(): Vector2D
+ __neg__(): Vector2D
+ copy(): Vector2D
+ __iadd__(Vector2D): Vector2D
+ __isub__(Vector2D): Vector2D
+ __imul__(Vector2D): Vector2D
+ __idiv__(Vector2D): Vector2D
+ __eq__(Vector2D): Vector2D
+ __ne__(Vector2D): Vector2D
+ __add__(Vector2D): Vector2D
+ __sub__(Vector2D): Vector2D
+ __mul__(Vector2D): Vector2D
+ __rmul__(Vector2D): Vector2D
+ __div__(Vector2D): Vector2D
+ __rdiv__(Vector2D): Vector2D
+ __str__(Vector2D): Vector2D
Matrix33
+ _m: float[]
+ __init__(Matrix33): void
+ reset(): void
+ translate(float, float): Matrix33
+ translate_update(float, float): void
+ scale(float, float): Matrix33
+ scale_update(float, float): void
+ rotate(int): Matrix33
+ rotate_update(int): void
+ rotate_by_vectors(Vector2D, Vector2D): Matrix33
+ rotate_by_vectors_update(Vector2D, Vector2D): void
+ transform_vector_2d_list(Point2D[]): void
+ transform_vector_2d(Point2D): void
+ __imul__(Matrix33): void
+ _fast_imul(Matrix33): void
+ __str__(): string
BoxWorldWindow
+ fps_display: float
+ labels: Label[]
+ limit: int
+ mouse_mode: string
+ mouse_modes: dictionary{key: string}
+ search_mode: int
+ world: BoxWorld
+ __init__(string, **kwargs): void
+ _update_label(string, string): void
+ add_handers(): void
+ on_resize(int, int): void
+ on_mouse_press(int, int, int, int): void
+ on_key_press(key): void
+ plan_path()
+ on_draw()
+ draw()
Projectile
+ damage: int
+ damage_factor: float
+ exploding: False
+ explosion_multiplier: float
+ explosion_time: datetime
+ heading: Vector2D
+ left_barrel: bool
+ owner_on_firing: Agent
+ pos: Vector2D
+ radius: float
+ scale_scalar: float
+ scale_vector: Vector2D
+ side: Vector2D
+ target: Vector2D
+ target_radius: float
+ vel: Vector2D
+ weapon: Weapon
+ world: BoxWorld
+ __init__(BoxWorld, float, float, Vector2D, Vector2D, Vector2D, Weapon, int, float): void
+ update(float): void
+ render(int[]): void
+ collided(): object[]
+ distance(Vector2D): float
EasyGraphics
+ pen_color: float[]
+ qobj: gluNewQuadric
+ stroke: float
+ text: Text
+ window: Window
+ __init__(): void
+ InitWithPyglet(Window): void
+ dot(float, float, Point2D, float[]): void
+ line(float, float, float, float, Point2D, Point2D): void
+ line_by_pos(Point2D, Point2D): void
+ polyline(Point2D[]): void
+ line_with_arrow(Point2D, Point2D, float): void
+ cross(Point2D, float): void
+ rect(float, float, float, float, bool): void
+ closed_shape(Point2D[], bool): void
+ circle(Point2D, float, bool, int): void
+ set_pen_color(float[], string): void
+ red_pen(): void
+ blue_pen(): void
+ green_pen(): void
+ black_pen(): void
+ white_pen(): void
+ grey_pen(): void
+ aqua_pen(): void
+ orange_pen(): void
+ pink_pen(): void
+ yellow_pen(): void
+ purple_pen(): void
+ brown_pen(): void
+ dark_green_pen(): void
+ light_blue_pen(): void
+ light_grey_pen(): void
+ light_pink_pen(): void
+ set_stroke(float): void
+ text_color(float[], string): void
+ text_at_pos(int, int, string): void
+ push(float, float): void
+ pop(): void
Waypoint
+ index: int
+ nodes: Box[]
+ __init__(int): void
Box
+ color: int[]
+ idx: int
+ idx_label: Label
+ kind: string
+ last_accessor: string
+ marker: unknown
+ marker_label: Label
+ node: Node
+ position: Vector2D
+ pos_label: Label
+ _pts: Point2D[]
+ radius: float
+ _vc: Vector2D
+ waypoint: int
+ waypoint_label: Label
+ get_vc(string): Vector2D
+ reposition(int[]): void
+ _reposition_labels(): void
+ set_kind(string): void
+ draw(): void
Weapon
+ accuracy_modifier: float
+ cooldown: float
+ damage: int
+ damage_factor: float
+ effective_range: float
+ effective_range_standard: int
+ explosive: bool
+ explosion_radius: float
+ magazine_size: int
+ magazines_left: int
+ name: string
+ owner: Agent
+ reload_time: float
+ rounds_left_in_magazine: int
+ speed: int
+ stamina_drain: float
+ world: BoxWorld
+ __init__(BoxWorld, Agent, string, float, float, int, float, int, float, int, float, int, float, float, bool, float): void
+ copy(): Weapon
Agent
+ agent_type: string
+ awareness_pos: Vector2D
+ awareness_radius: float
+ awareness_radius_standard: float
+ box: Box
+ combat_mode: string
+ current_node_box: Box
+ current_node_pos: Vector2D
+ fear: float
+ following_enemy: bool
+ heading: Vector2D
+ health: float
+ hit_time: datetime
+ last_box: Box
+ last_fear_ping: datetime
+ last_new_node_time: datetime
+ last_node_box: Box
+ last_shot: datetime
+ max_speed: float
+ max_speed_standard: int
+ movement_mode: string
+ name: string
+ path: Path
+ pos: Vector2D
+ radius: float
+ radius_standard: float
+ respawnable: bool
+ side: Vector2D
+ start_health: int
+ target: Box
+ target_ally: Agent
+ target_enemy: Agent
+ weapons: Weapon[]
+ world: BoxWorld
+ __init__(BoxWorld, float, string, string, string, Weapon[], float, Box, Path, string, bool): void
+ update(float): void
+ update_soldier(float): void
+ select_soldier_movement_mode(): void
+ squad_overwhelmed(): bool
+ ally_attacking(): bool
+ move_soldier(float): void
+ update_fugitive(float): void
+ scared(): bool
+ select_fugitive_movement_mode(): void
+ move_fugitive(float): void
+ render(int[]): void
+ follow_graph_path(float): void
+ follow_target_enemy(): void
+ sit_still(): void
+ update_heading(): void
+ handle_weapons(): void
+ choose_weapon(): bool
+ aim_shot(Agent): Vector2D
+ get_target_path_measurements(Agent): dictionary{int: dictionary{string: value}}
+ get_future_pos_on_path(Agent, dictionary{int: dictionary{string: value}}, float, float): Vector2D
+ shoot(Vector2D): void
+ calculate_path(string, Box, int): Path
+ distance(Vector2D): float
+ finish_scouting(): void
+ get_new_path(): void
+ plan_path(string, float): void
+ position_in_random_box(): void
+ see_target(): bool
+ suitable_fleeing_location(Box): bool
+ suitable_scouting_location(Box): bool
+ target_and_path_in_range(Agent): bool
PriorityQueue
+ i: int
+ q: object[]
+ __init__(): void
+ push(object, int): void
+ pop(): object, int
+ __len__(): int
+ __str__(): string
+ __contains__(object): bool
+ __iter__(): iter
+ peek(object): tuple(object, int)
+ remove(object): void
Node
+ idx: int
+ __init__(int): void
SparseGraph
+ cost_h: delegate
+ digraph: bool
+ edgelist: dictionary{int, Edge}
+ next_node_idx: int
+ nodes: dictionary{int, dictionary{int, Node}}
+ __init__(bool): void
+ is_empty(): bool
+ is_node(int): bool
+ is_edge(int, int): bool
+ get_node(int): Node
+ get_edge(int, int): Edge
+ get_neighbours(int): int[]
+ add_node(Node): Node
+ remove_node(int): void
+ add_edge(Edge): void
+ remove_edge(int, int): void
+ num_nodes(): int
+ num_edges(): int
+ clear(): void
+ path_cost(Path): int
+ summary(): string
+ get_adj_list_str(): nested list
+ FromAdjacencyList(unknown, nested list, bool, Node): SparseGraph
Path
+ closed: int[]
+ open: int[]
+ path: int[]
+ path_cost: string
+ result: string
+ route: dictionary{int: int}
+ source_idx: int
+ steps: int
+ target_idx: int
+ __init__(SparseGraph, dictionary{int: int}, int[], int[], int[], int): void
+ report(int): string
Edge
+ cost: float
+ from_idx: int
+ to_idx: int
+ __init__(int, int, float): void
1
1
1
1
0..5
6
1
0..n
1
0..n
1
1
0..n
0..n
1
1
0..200
1
0..n
5
1
0..n
1
1
1
1
1
1
1..n
1
0..n
1
0..1
1
1

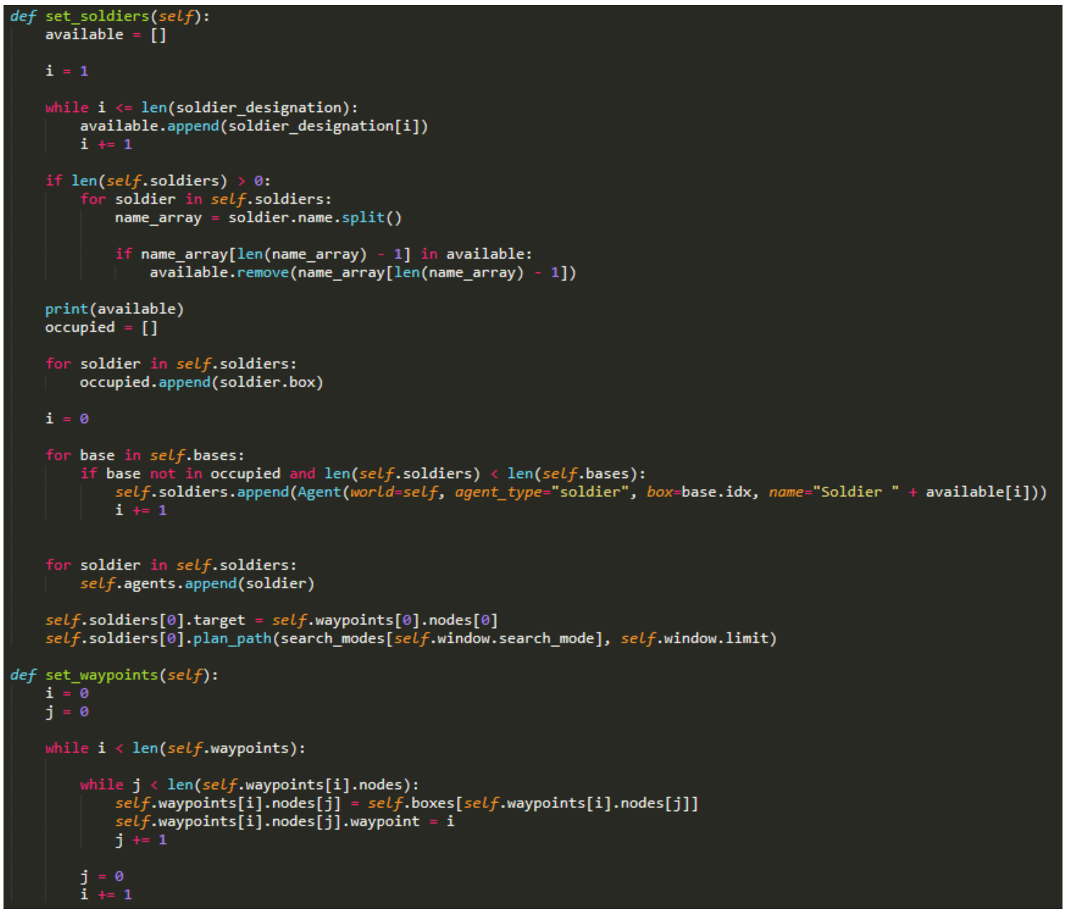
Figure 3: the UML class diagram for the whole program.

Code Snippets



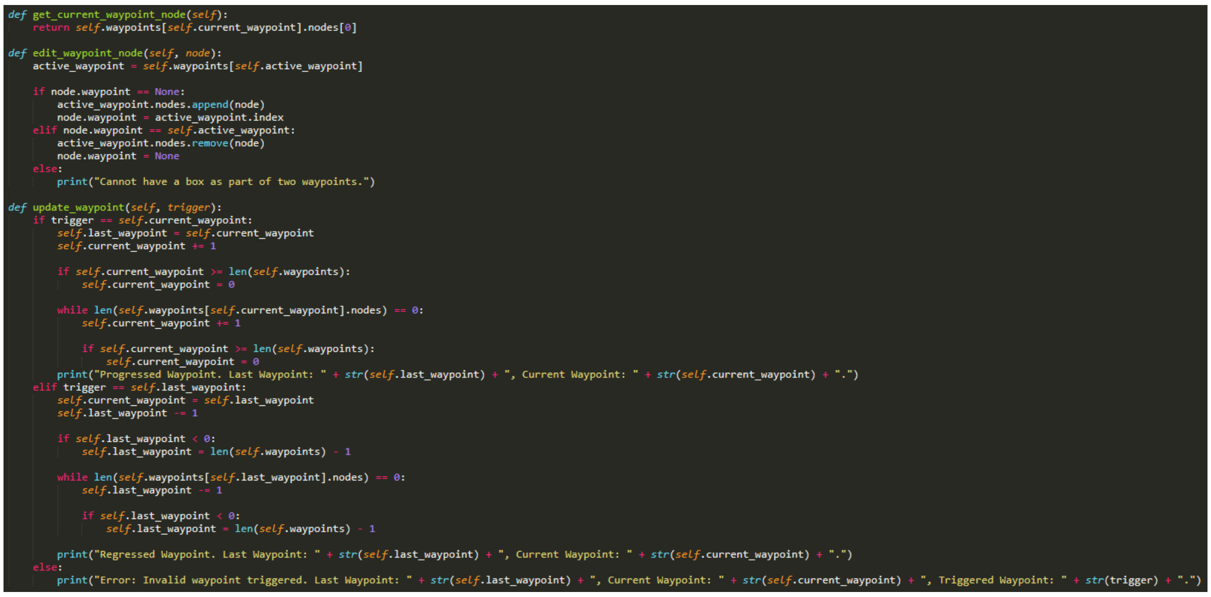
```python
@classmethod
def FromFile(cls, filename, pixels=(500,500) ):
    '''Support a the construction of a BoxWorld map from a simple text file.
    See the module doc details at the top of this file for format details.
    '''
    # open and read the file
    f = open(filename)
    lines = []
    for line in f.readlines():
        line = line.strip()
        if line and not line.startswith('#'): lines.append(line)
    f.close()
    # first line is the number of boxes width, height
    nx, ny = [int(bit) for bit in lines.pop(0).split()]
    # Create a new BoxWorld to store all the new boxes in...
    cx, cy = pixels
    world = BoxWorld(nx, ny, cx, cy)
    # Get and set the waypoints
    world.current_waypoint, world.last_waypoint = [int(bit) for bit in lines.pop(0).split()]
    n = int(lines.pop(0))
    i = 0
    while i < n:
        nodes = lines.pop(0).split()
        for node in nodes: world.waypoints[i].nodes.append(int(node))
        i += 1
    # Ready to process each line
    assert len(lines) == ny, "Number of rows doesn't match data. Rows: " + str(len(lines)) + ", Rows in data: " + str(ny)
    # read each line
    idx = 0
    for line in reversed(lines): # in reverse order
        bits = line.split()
        assert len(bits) == nx, "Number of columns doesn't match data."
        for bit in bits:
            bit = bit.strip()
            assert bit in box_kind, "Not a known box type: "+bit
            world.boxes[idx].set_kind(bit)
            idx += 1
    return world
```

Figure 4: BoxWorld.FromFile(), modified to load waypoints into BoxWorld.



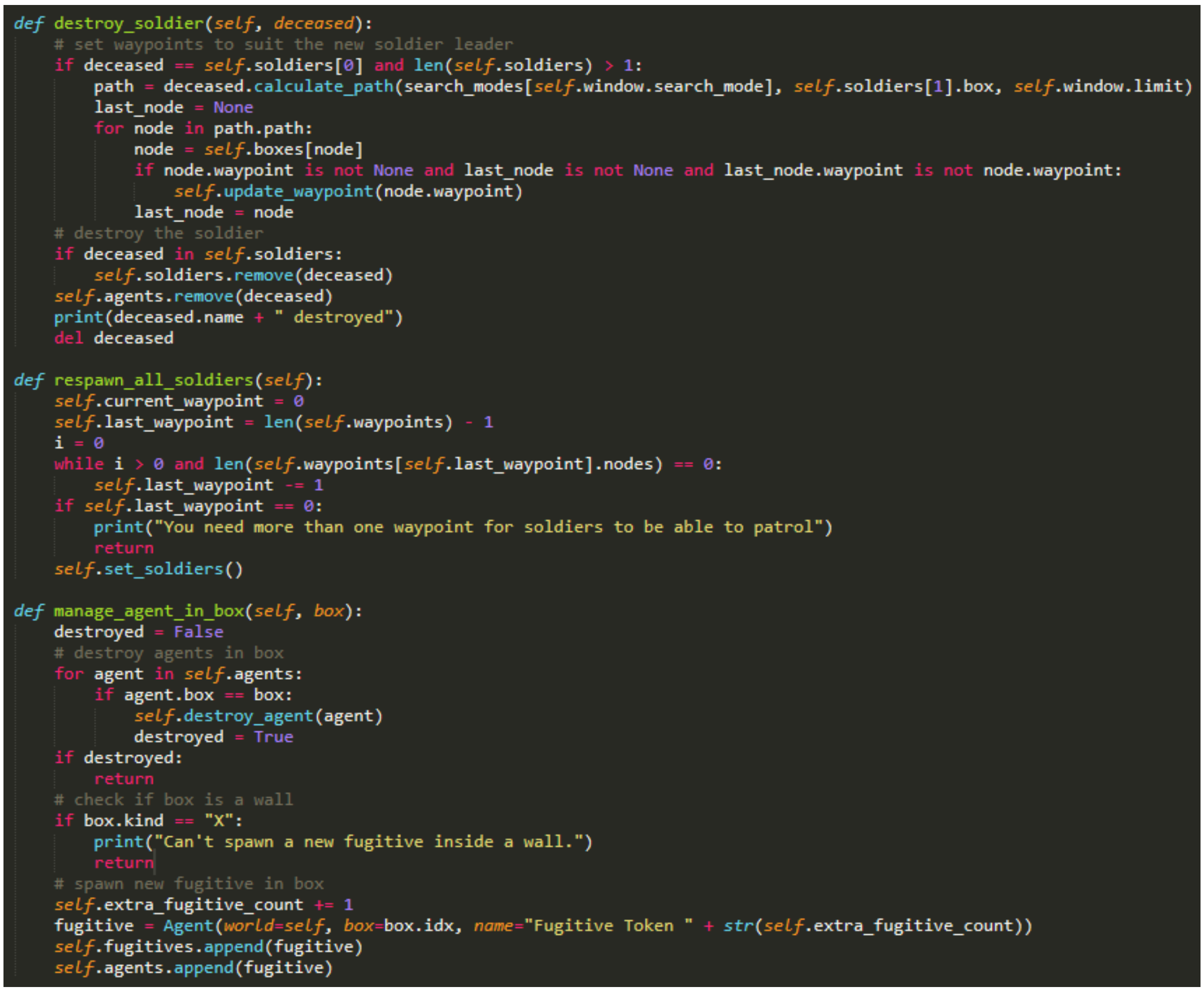
```python
def set_soldiers(self):
    available = []

    i = 1

    while i <= len(soldier_designation):
        available.append(soldier_designation[i])
        i += 1

    if len(self.soldiers) > 0:
        for soldier in self.soldiers:
            name_array = soldier.name.split()

            if name_array[len(name_array) - 1] in available:
                available.remove(name_array[len(name_array) - 1])

    print(available)
    occupied = []

    for soldier in self.soldiers:
        occupied.append(soldier.box)

    i = 0

    for base in self.bases:
        if base not in occupied and len(self.soldiers) < len(self.bases):
            self.soldiers.append(Agent(world=self, agent_type="soldier", box=base.idx, name="Soldier " + available[i]))
            i += 1


    for soldier in self.soldiers:
        self.agents.append(soldier)

    self.soldiers[0].target = self.waypoints[0].nodes[0]
    self.soldiers[0].plan_path(search_modes[self.window.search_mode], self.window.limit)

def set_waypoints(self):
    i = 0
    j = 0

    while i < len(self.waypoints):

        while j < len(self.waypoints[i].nodes):
            self.waypoints[i].nodes[j] = self.boxes[self.waypoints[i].nodes[j]]
            self.waypoints[i].nodes[j].waypoint = i
            j += 1

        j = 0
        i += 1
```

Figure 5: BoxWorld's setup methods for soldier agents and the waypoints for their patrols.



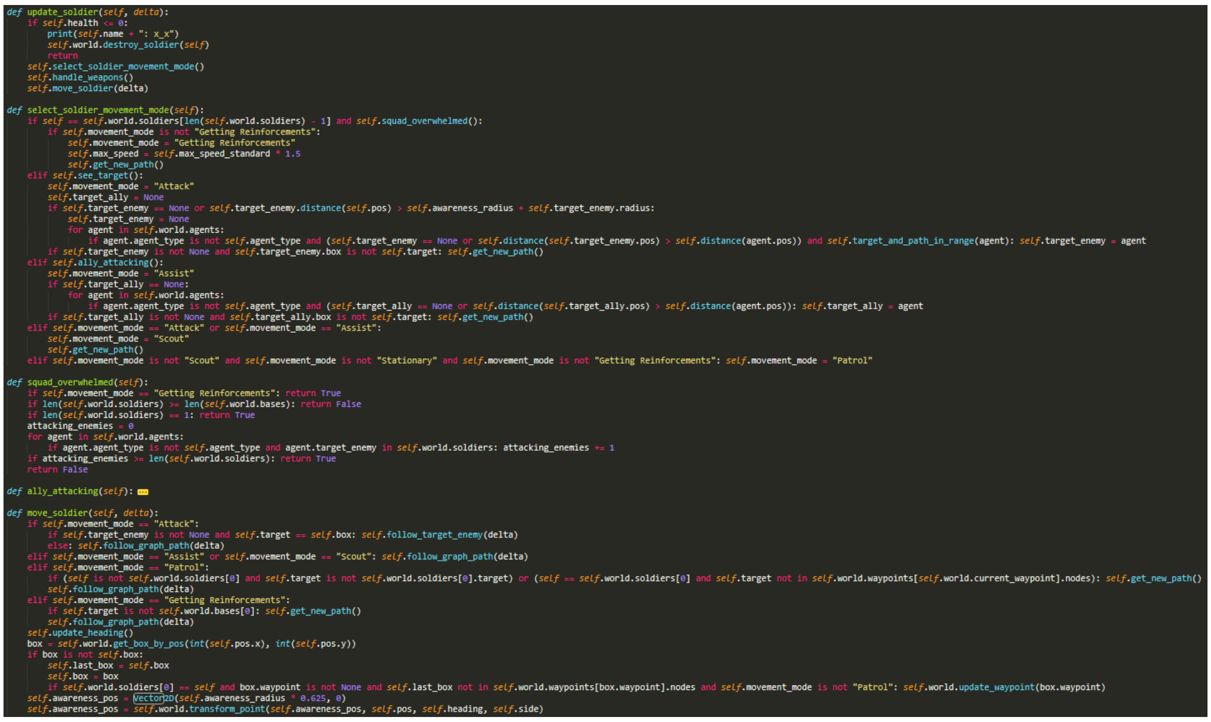
```python
def get_current_waypoint_node(self):
    return self.waypoints[self.current_waypoint].nodes[0]

def edit_waypoint_node(self, node):
    active_waypoint = self.waypoints[self.active_waypoint]

    if node.waypoint == None:
        active_waypoint.nodes.append(node)
        node.waypoint = active_waypoint.index
    elif node.waypoint == self.active_waypoint:
        active_waypoint.nodes.remove(node)
        node.waypoint = None
    else:
        print("Cannot have a box as part of two waypoints.")

def update_waypoint(self, trigger):
    if trigger == self.current_waypoint:
        self.last_waypoint = self.current_waypoint
        self.current_waypoint += 1

        if self.current_waypoint >= len(self.waypoints):
            self.current_waypoint = 0

        while len(self.waypoints[self.current_waypoint].nodes) == 0:
            self.current_waypoint += 1

            if self.current_waypoint >= len(self.waypoints):
                self.current_waypoint = 0
        print("Progressed Waypoint. Last Waypoint: " + str(self.last_waypoint) + ", Current Waypoint: " + str(self.current_waypoint) + ".")
    elif trigger == self.last_waypoint:
        self.current_waypoint = self.last_waypoint
        self.last_waypoint -= 1

        if self.last_waypoint < 0:
            self.last_waypoint = len(self.waypoints) - 1

        while len(self.waypoints[self.last_waypoint].nodes) == 0:
            self.last_waypoint -= 1

            if self.last_waypoint < 0:
                self.last_waypoint = len(self.waypoints) - 1

        print("Regressed Waypoint. Last Waypoint: " + str(self.last_waypoint) + ", Current Waypoint: " + str(self.current_waypoint) + ".")
    else:
        print("Error: Invalid waypoint triggered. Last Waypoint: " + str(self.last_waypoint) + ", Current Waypoint: " + str(self.current_waypoint) + ", Triggered Waypoint: " + str(trigger) + ".")
```

Figure 6: BoxWorld's methods for managing waypoints, whether through users editing them (edit_waypoint_node()), or soldier agents moving through the BoxWorld (update_waypoint()).



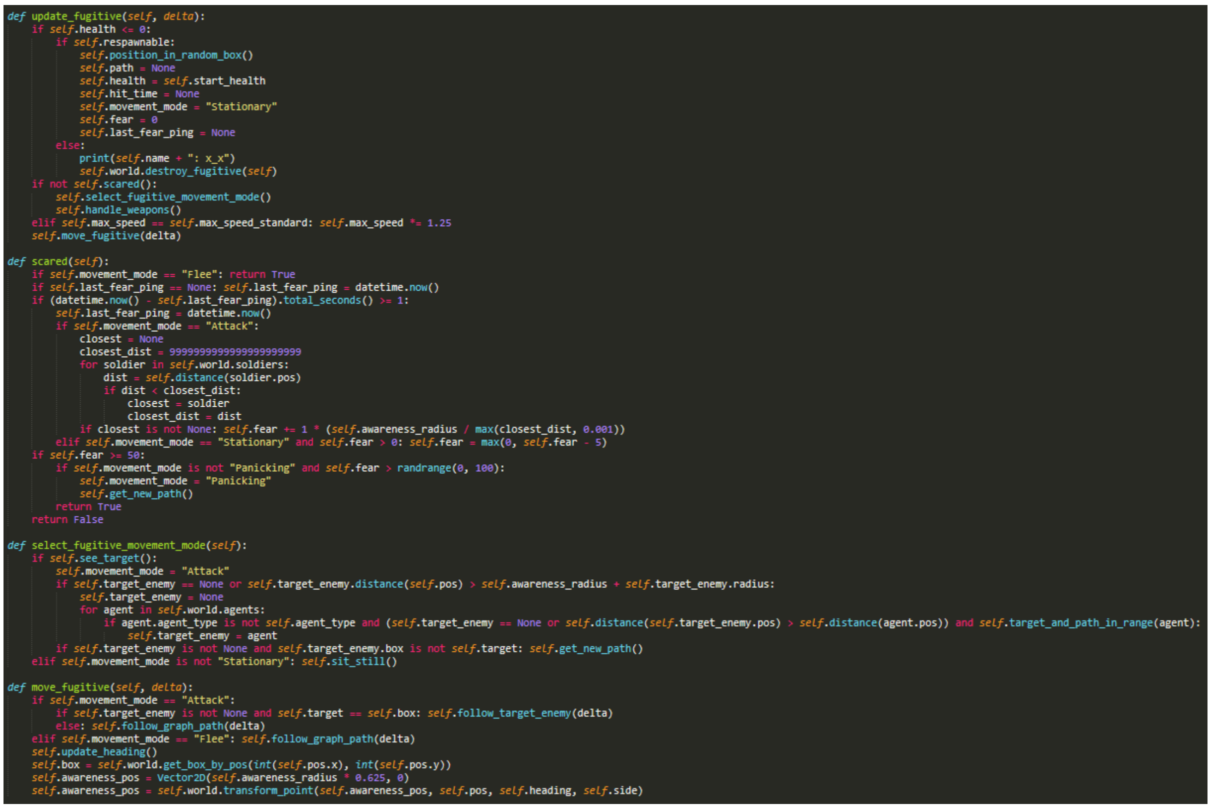
```python
def destroy_soldier(self, deceased):
    # set waypoints to suit the new soldier leader
    if deceased == self.soldiers[0] and len(self.soldiers) > 1:
        path = deceased.calculate_path(search_modes[self.window.search_mode], self.soldiers[1].box, self.window.limit)
        last_node = None
        for node in path.path:
            node = self.boxes[node]
            if node.waypoint is not None and last_node is not None and last_node.waypoint is not node.waypoint:
                self.update_waypoint(node.waypoint)
            last_node = node
    # destroy the soldier
    if deceased in self.soldiers:
        self.soldiers.remove(deceased)
    self.agents.remove(deceased)
    print(deceased.name + " destroyed")
    del deceased

def respawn_all_soldiers(self):
    self.current_waypoint = 0
    self.last_waypoint = len(self.waypoints) - 1
    i = 0
    while i > 0 and len(self.waypoints[self.last_waypoint].nodes) == 0:
        self.last_waypoint -= 1
    if self.last_waypoint == 0:
        print("You need more than one waypoint for soldiers to be able to patrol")
        return
    self.set_soldiers()

def manage_agent_in_box(self, box):
    destroyed = False
    # destroy agents in box
    for agent in self.agents:
        if agent.box == box:
            self.destroy_agent(agent)
            destroyed = True
    if destroyed:
        return
    # check if box is a wall
    if box.kind == "X":
        print("Can't spawn a new fugitive inside a wall.")
        return
    # spawn new fugitive in box
    self.extra_fugitive_count += 1
    fugitive = Agent(world=self, box=box.idx, name="Fugitive Token " + str(self.extra_fugitive_count))
    self.fugitives.append(fugitive)
    self.agents.append(fugitive)
```

Figure 7: BoxWorld's methods for destroying soldiers (whether on-click or at the hands of a fugitive), respawning them all if they all die, and managing what happens when they are clicked (destroy_agent() redirects to destroy_soldier() and destroy_fugitive() as appropriate, the latter just removing the fugitive agent and nothing more).



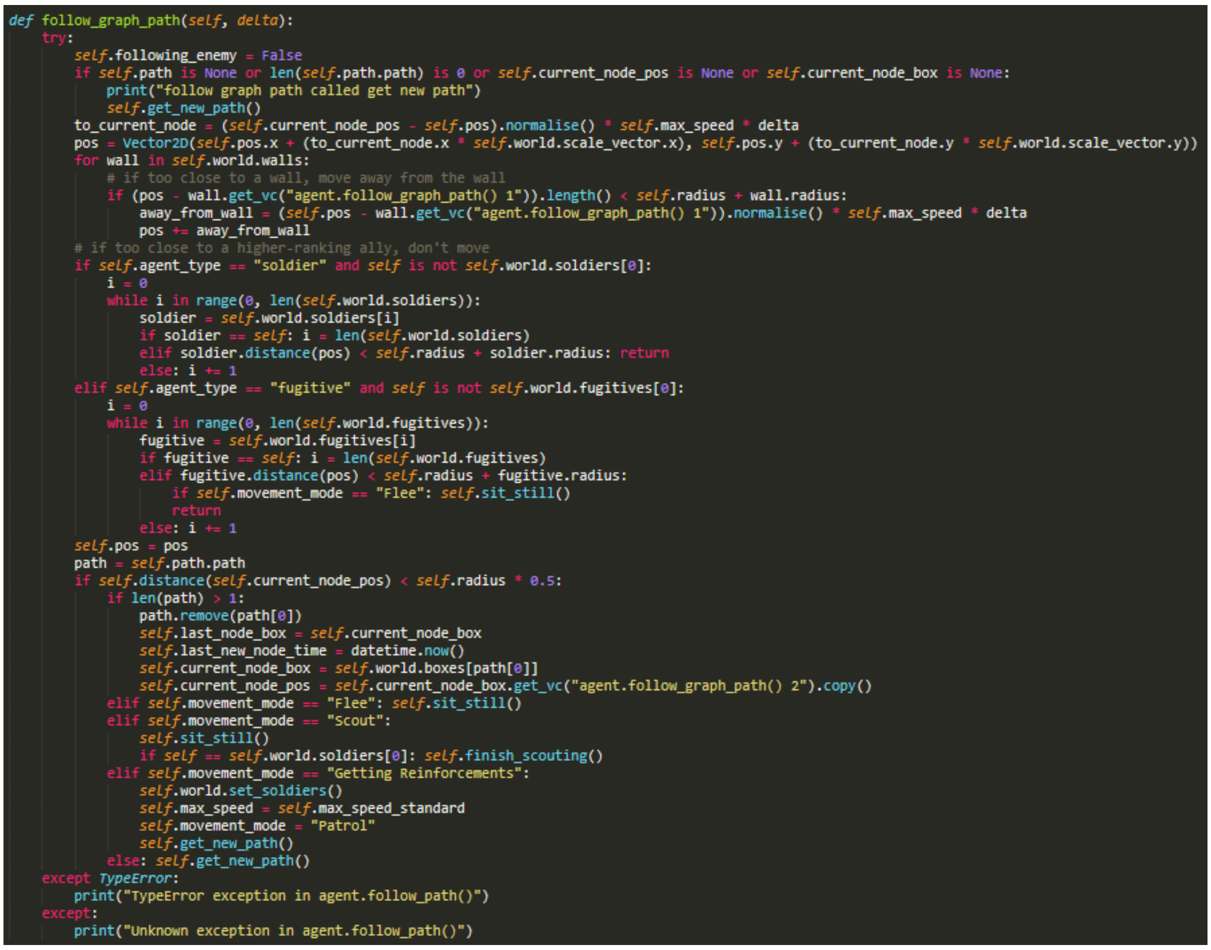
```python
def update_soldier(self, delta):
    if self.health <= 0:
        print(self.name + ": x_x")
        self.world.destroy_soldier(self)
        return
    self.select_soldier_movement_mode()
    self.handle_weapons()
    self.move_soldier(delta)

def select_soldier_movement_mode(self):
    if self == self.world.soldiers[len(self.world.soldiers) - 1] and self.squad_overwhelmed():
        if self.movement_mode is not "Getting Reinforcements":
            self.movement_mode = "Getting Reinforcements"
            self.max_speed = self.max_speed_standard * 1.5
            self.get_new_path()
    elif self.see_target():
        self.movement_mode = "Attack"
        self.target_ally = None
        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.agent_type is not self.agent_type and (self.target_enemy == None or self.distance(self.target_enemy.pos) > self.distance(agent.pos)) and self.target_and_path_in_range(agent): self.target_enemy = agent
        if self.target_enemy is not None and self.target_enemy.box is not self.target: self.get_new_path()
    elif self.ally_attacking():
        self.movement_mode = "Assist"
        if self.target_ally == None:
            for agent in self.world.agents:
                if agent.agent_type is not self.agent_type and (self.target_ally == None or self.distance(self.target_ally.pos) > self.distance(agent.pos)): self.target_ally = agent
        if self.target_ally is not None and self.target_ally.box is not self.target: self.get_new_path()
    elif self.movement_mode == "Attack" or self.movement_mode == "Assist":
        self.movement_mode = "Scout"
        self.get_new_path()
    elif self.movement_mode is not "Scout" and self.movement_mode is not "Stationary" and self.movement_mode is not "Getting Reinforcements": self.movement_mode = "Patrol"

def squad_overwhelmed(self):
    if self.movement_mode == "Getting Reinforcements": return True
    if len(self.world.soldiers) >= len(self.world.bases): return False
    if len(self.world.soldiers) == 1: return True
    attacking_enemies = 0
    for agent in self.world.agents:
        if agent.agent_type is not self.agent_type and agent.target_enemy in self.world.soldiers: attacking_enemies += 1
    if attacking_enemies >= len(self.world.soldiers): return True
    return False

def ally_attacking(self): ...

def move_soldier(self, delta):
    if self.movement_mode == "Attack":
        if self.target_enemy is not None and self.target == self.box: self.follow_target_enemy(delta)
        else: self.follow_graph_path(delta)
    elif self.movement_mode == "Assist" or self.movement_mode == "Scout": self.follow_graph_path(delta)
    elif self.movement_mode == "Patrol":
        if (self is not self.world.soldiers[0] and self.target is not self.world.soldiers[0].target) or (self == self.world.soldiers[0] and self.target not in self.world.waypoints[self.world.current_waypoint].nodes): self.get_new_path()
        self.follow_graph_path(delta)
    elif self.movement_mode == "Getting Reinforcements":
        if self.target is not self.world.bases[0]: self.get_new_path()
        self.follow_graph_path(delta)
    self.update_heading()
    box = self.world.get_box_by_pos(int(self.pos.x), int(self.pos.y))
    if box is not self.box:
        self.last_box = self.box
        self.box = box
        if self.world.soldiers[0] == self and box.waypoint is not None and self.last_box not in self.world.waypoints[box.waypoint].nodes and self.movement_mode is not "Patrol": self.world.update_waypoint(box.waypoint)
    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)
```

Figure 8: Agent.update_soldier(), as well as the soldier-specific methods it requires to function.



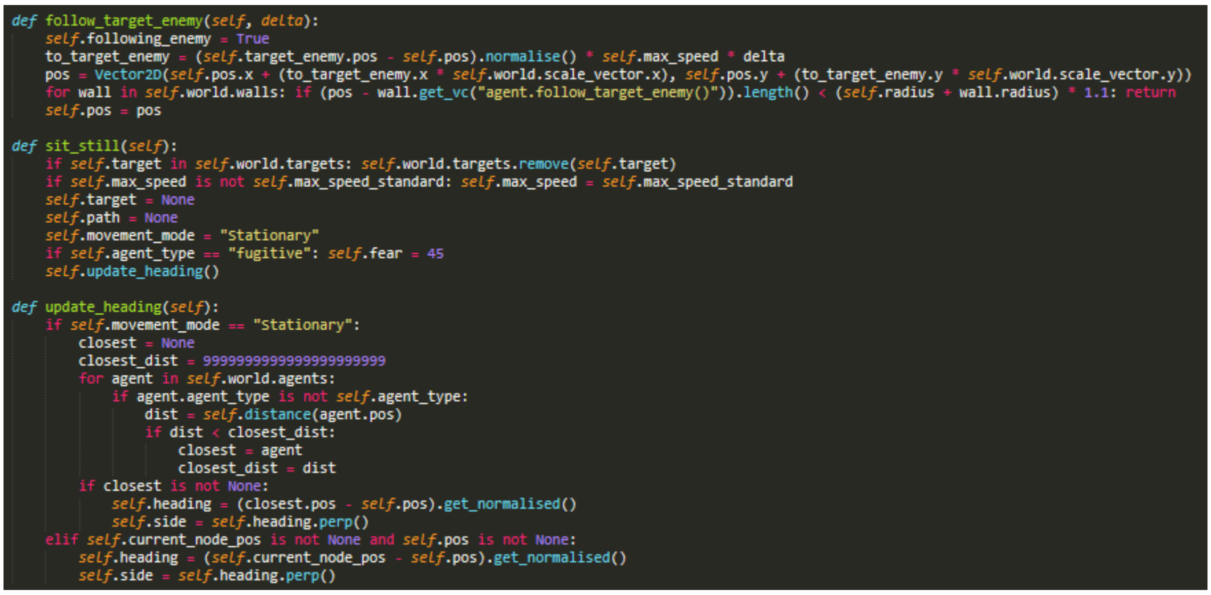
```python
def update_fugitive(self, delta):
    if self.health <= 0:
        if self.respawnable:
            self.position_in_random_box()
            self.path = None
            self.health = self.start_health
            self.hit_time = None
            self.movement_mode = "Stationary"
            self.fear = 0
            self.last_fear_ping = None
        else:
            print(self.name + ": x_x")
            self.world.destroy_fugitive(self)
    if not self.scared():
        self.select_fugitive_movement_mode()
        self.handle_weapons()
    elif self.max_speed == self.max_speed_standard: self.max_speed *= 1.25
    self.move_fugitive(delta)

def scared(self):
    if self.movement_mode == "Flee": return True
    if self.last_fear_ping == None: self.last_fear_ping = datetime.now()
    if (datetime.now() - self.last_fear_ping).total_seconds() >= 1:
        self.last_fear_ping = datetime.now()
        if self.movement_mode == "Attack":
            closest = None
            closest_dist = 9999999999999999999999
            for soldier in self.world.soldiers:
                dist = self.distance(soldier.pos)
                if dist < closest_dist:
                    closest = soldier
                    closest_dist = dist
            if closest is not None: self.fear += 1 * (self.awareness_radius / max(closest_dist, 0.001))
        elif self.movement_mode == "Stationary" and self.fear > 0: self.fear = max(0, self.fear - 5)
    if self.fear >= 50:
        if self.movement_mode is not "Panicking" and self.fear > randrange(0, 100):
            self.movement_mode = "Panicking"
            self.get_new_path()
        return True
    return False

def select_fugitive_movement_mode(self):
    if self.see_target():
        self.movement_mode = "Attack"
        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.agent_type is not self.agent_type and (self.target_enemy == None or self.distance(self.target_enemy.pos) > self.distance(agent.pos)) and self.target_and_path_in_range(agent):
                    self.target_enemy = agent
        if self.target_enemy is not None and self.target_enemy.box is not self.target: self.get_new_path()
    elif self.movement_mode is not "Stationary": self.sit_still()

def move_fugitive(self, delta):
    if self.movement_mode == "Attack":
        if self.target_enemy is not None and self.target == self.box: self.follow_target_enemy(delta)
        else: self.follow_graph_path(delta)
    elif self.movement_mode == "Flee": self.follow_graph_path(delta)
    self.update_heading()
    self.box = self.world.get_box_by_pos(int(self.pos.x), int(self.pos.y))
    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)
```

Figure 9: Agent.update_fugitive(), as well as the fugitive-specific methods it requires to function.



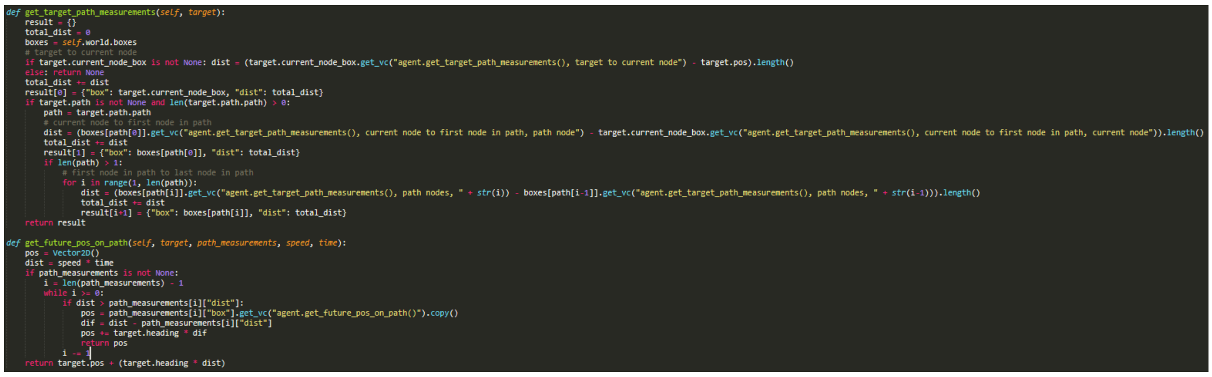
```python
def follow_graph_path(self, delta):
    try:
        self.following_enemy = False
        if self.path is None or len(self.path.path) is 0 or self.current_node_pos is None or self.current_node_box is None:
            print("follow graph path called get new path")
            self.get_new_path()
        to_current_node = (self.current_node_pos - self.pos).normalise() * self.max_speed * delta
        pos = Vector2D(self.pos.x + (to_current_node.x * self.world.scale_vector.x), self.pos.y + (to_current_node.y * self.world.scale_vector.y))
        for wall in self.world.walls:
            # if too close to a wall, move away from the wall
            if (pos - wall.get_vc("agent.follow_graph_path() 1")).length() < self.radius + wall.radius:
                away_from_wall = (self.pos - wall.get_vc("agent.follow_graph_path() 1")).normalise() * self.max_speed * delta
                pos += away_from_wall
        # if too close to a higher-ranking ally, don't move
        if self.agent_type == "soldier" and self is not self.world.soldiers[0]:
            i = 0
            while i in range(0, len(self.world.soldiers)):
                soldier = self.world.soldiers[i]
                if soldier == self: i = len(self.world.soldiers)
                elif soldier.distance(pos) < self.radius + soldier.radius: return
                else: i += 1
        elif self.agent_type == "fugitive" and self is not self.world.fugitives[0]:
            i = 0
            while i in range(0, len(self.world.fugitives)):
                fugitive = self.world.fugitives[i]
                if fugitive == self: i = len(self.world.fugitives)
                elif fugitive.distance(pos) < self.radius + fugitive.radius:
                    if self.movement_mode == "Flee": self.sit_still()
                    return
                else: i += 1
        self.pos = pos
        path = self.path.path
        if self.distance(self.current_node_pos) < self.radius * 0.5:
            if len(path) > 1:
                path.remove(path[0])
                self.last_node_box = self.current_node_box
                self.last_new_node_time = datetime.now()
                self.current_node_box = self.world.boxes[path[0]]
                self.current_node_pos = self.current_node_box.get_vc("agent.follow_graph_path() 2").copy()
            elif self.movement_mode == "Flee": self.sit_still()
            elif self.movement_mode == "Scout":
                self.sit_still()
                if self == self.world.soldiers[0]: self.finish_scouting()
            elif self.movement_mode == "Getting Reinforcements":
                self.world.set_soldiers()
                self.max_speed = self.max_speed_standard
                self.movement_mode = "Patrol"
                self.get_new_path()
            else: self.get_new_path()
    except TypeError:
        print("TypeError exception in agent.follow_path()")
    except:
        print("Unknown exception in agent.follow_path()")
```

Figure 10: Agent.follow_graph_path(), the primary method for moving agents, advancing them along their paths until they require a new path.



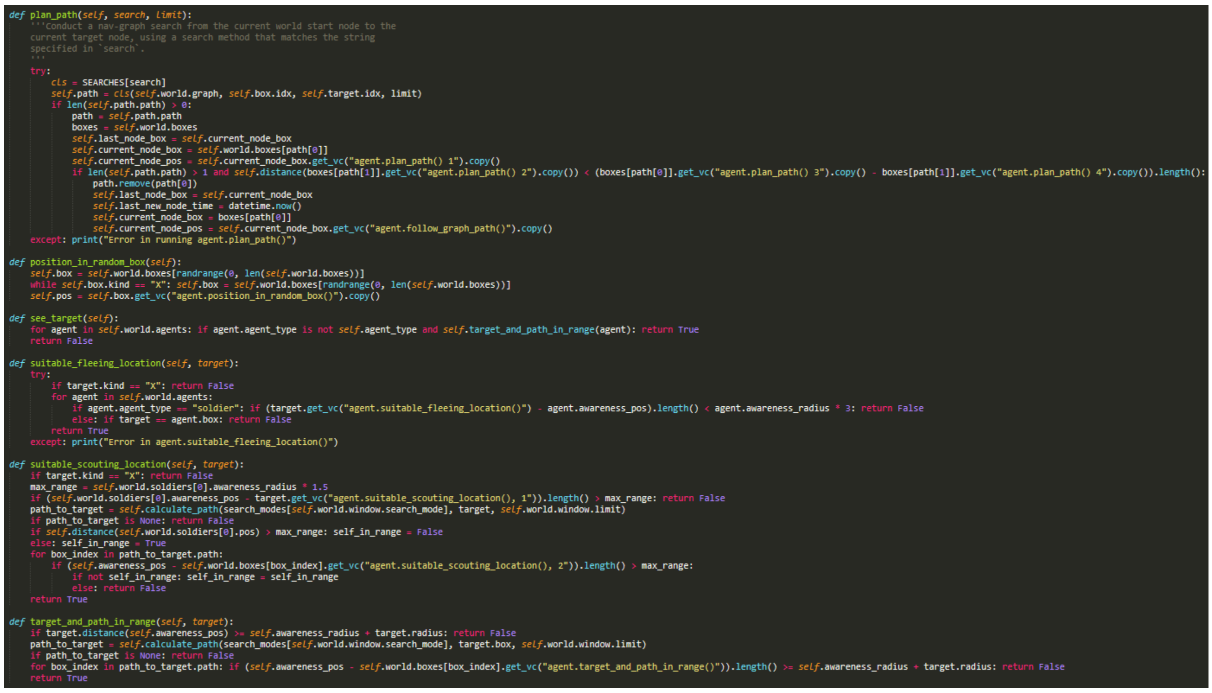
```python
def follow_target_enemy(self, delta):
    self.following_enemy = True
    to_target_enemy = (self.target_enemy.pos - self.pos).normalise() * self.max_speed * delta
    pos = Vector2D(self.pos.x + (to_target_enemy.x * self.world.scale_vector.x), self.pos.y + (to_target_enemy.y * self.world.scale_vector.y))
    for wall in self.world.walls: if (pos - wall.get_vc("agent.follow_target_enemy()")).length() < (self.radius + wall.radius) * 1.1: return
    self.pos = pos

def sit_still(self):
    if self.target in self.world.targets: self.world.targets.remove(self.target)
    if self.max_speed is not self.max_speed_standard: self.max_speed = self.max_speed_standard
    self.target = None
    self.path = None
    self.movement_mode = "Stationary"
    if self.agent_type == "fugitive": self.fear = 45
    self.update_heading()

def update_heading(self):
    if self.movement_mode == "Stationary":
        closest = None
        closest_dist = 999999999999999999999
        for agent in self.world.agents:
            if agent.agent_type is not self.agent_type:
                dist = self.distance(agent.pos)
                if dist < closest_dist:
                    closest = agent
                    closest_dist = dist
        if closest is not None:
            self.heading = (closest.pos - self.pos).get_normalised()
            self.side = self.heading.perp()
    elif self.current_node_pos is not None and self.pos is not None:
        self.heading = (self.current_node_pos - self.pos).get_normalised()
        self.side = self.heading.perp()
```

Figure 11: additional movement-related methods for following an enemy when in the same square as them (follow_target_enemy()), rendering an agent motionless (sit_still()), and orienting them in the appropriate direction (update_heading()).



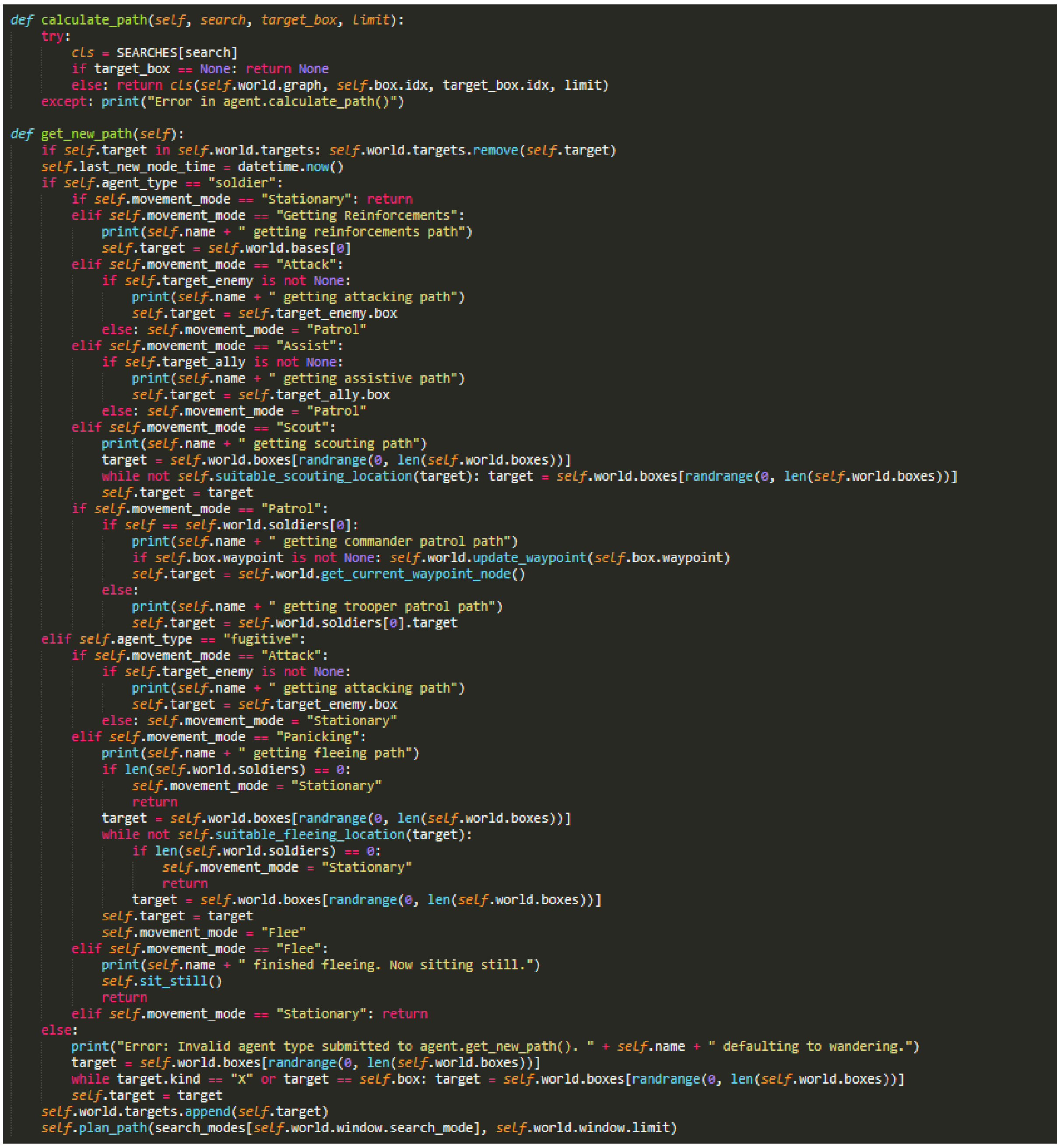
```python
def get_target_path_measurements(self, target):
    result = {}
    total_dist = 0
    boxes = self.world.boxes
    # target to current node
    if target.current_node_box is not None: dist = (target.current_node_box.get_vc("agent.get_target_path_measurements(), target to current node") - target.pos).length()
    else: return None
    total_dist += dist
    result[0] = {"box": target.current_node_box, "dist": total_dist}
    if target.path is not None and len(target.path.path) > 0:
        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.current_node_box.get_vc("agent.get_target_path_measurements(), current node to first node in path, current node")).length()
        total_dist += dist
        result[1] = {"box": boxes[path[0]], "dist": total_dist}
        if len(path) > 1:
            # 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("agent.get_target_path_measurements(), path nodes, " + str(i-1))).length()
                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 12: methods used by Agent.aim() to predict agents' positions on their path at a specified point in the future.



```python
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`.
    '''
    try:
        cls = SEARCHES[search]
        self.path = cls(self.world.graph, self.box.idx, self.target.idx, limit)
        if len(self.path.path) > 0:
            path = self.path.path
            boxes = self.world.boxes
            self.last_node_box = self.current_node_box
            self.current_node_box = self.world.boxes[path[0]]
            self.current_node_pos = self.current_node_box.get_vc("agent.plan_path() 1").copy()
            if len(self.path.path) > 1 and self.distance(boxes[path[1]].get_vc("agent.plan_path() 2").copy()) < (boxes[path[0]].get_vc("agent.plan_path() 3").copy() - boxes[path[1]].get_vc("agent.plan_path() 4").copy()).length():
                path.remove(path[0])
                self.last_node_box = self.current_node_box
                self.last_new_node_time = datetime.now()
                self.current_node_box = boxes[path[0]]
                self.current_node_pos = self.current_node_box.get_vc("agent.follow_graph_path()").copy()
    except: print("Error in running agent.plan_path()")

def position_in_random_box(self):
    self.box = self.world.boxes[randrange(0, len(self.world.boxes))]
    while self.box.kind == "X": self.box = self.world.boxes[randrange(0, len(self.world.boxes))]
    self.pos = self.box.get_vc("agent.position_in_random_box()").copy()

def see_target(self):
    for agent in self.world.agents: if agent.agent_type is not self.agent_type and self.target_and_path_in_range(agent): return True
    return False

def suitable_fleeing_location(self, target):
    try:
        if target.kind == "X": return False
        for agent in self.world.agents:
            if agent.agent_type == "soldier": if (target.get_vc("agent.suitable_fleeing_location()") - agent.awareness_pos).length() < agent.awareness_radius * 3: return False
            else: if target == agent.box: return False
        return True
    except: print("Error in agent.suitable_fleeing_location()")

def suitable_scouting_location(self, target):
    if target.kind == "X": return False
    max_range = self.world.soldiers[0].awareness_radius * 1.5
    if (self.world.soldiers[0].awareness_pos - target.get_vc("agent.suitable_scouting_location(), 1")).length() > max_range: return False
    path_to_target = self.calculate_path(search_modes[self.world.window.search_mode], target, self.world.window.limit)
    if path_to_target is None: return False
    if self.distance(self.world.soldiers[0].pos) > max_range: self_in_range = False
    else: self_in_range = True
    for box_index in path_to_target.path:
        if (self.awareness_pos - self.world.boxes[box_index].get_vc("agent.suitable_scouting_location(), 2")).length() > max_range:
            if not self_in_range: self_in_range = self_in_range
            else: return False
    return True

def target_and_path_in_range(self, target):
    if target.distance(self.awareness_pos) >= self.awareness_radius + target.radius: return False
    path_to_target = self.calculate_path(search_modes[self.world.window.search_mode], target.box, self.world.window.limit)
    if path_to_target is None: return False
    for box_index in path_to_target.path: if (self.awareness_pos - self.world.boxes[box_index].get_vc("agent.target_and_path_in_range()")).length() >= self.awareness_radius + target.radius: return False
    return True
```

Figure13: additional supportive methods in Agent: plan_path(), customized for agent moving in a graph-based environment,; see_target(), which checks if an enemy is in range and the agent could reasonably get to it; suitable_fleeing_location() and suitable_scouting_location(), used by fugitives and soldiers respectively to pick appropriate destinations; and target_and_path_in_range(), which verifies that a target agent and every node in a path to it would be within the agent's awareness range.



```python
def calculate_path(self, search, target_box, limit):
    try:
        cls = SEARCHES[search]
        if target_box == None: return None
        else: return cls(self.world.graph, self.box.idx, target_box.idx, limit)
    except: print("Error in agent.calculate_path()")

def get_new_path(self):
    if self.target in self.world.targets: self.world.targets.remove(self.target)
    self.last_new_node_time = datetime.now()
    if self.agent_type == "soldier":
        if self.movement_mode == "Stationary": return
        elif self.movement_mode == "Getting Reinforcements":
            print(self.name + " getting reinforcements path")
            self.target = self.world.bases[0]
        elif self.movement_mode == "Attack":
            if self.target_enemy is not None:
                print(self.name + " getting attacking path")
                self.target = self.target_enemy.box
            else: self.movement_mode = "Patrol"
        elif self.movement_mode == "Assist":
            if self.target_ally is not None:
                print(self.name + " getting assistive path")
                self.target = self.target_ally.box
            else: self.movement_mode = "Patrol"
        elif self.movement_mode == "Scout":
            print(self.name + " getting scouting path")
            target = self.world.boxes[randrange(0, len(self.world.boxes))]
            while not self.suitable_scouting_location(target): target = self.world.boxes[randrange(0, len(self.world.boxes))]
            self.target = target
        if self.movement_mode == "Patrol":
            if self == self.world.soldiers[0]:
                print(self.name + " getting commander patrol path")
                if self.box.waypoint is not None: self.world.update_waypoint(self.box.waypoint)
                self.target = self.world.get_current_waypoint_node()
            else:
                print(self.name + " getting trooper patrol path")
                self.target = self.world.soldiers[0].target
    elif self.agent_type == "fugitive":
        if self.movement_mode == "Attack":
            if self.target_enemy is not None:
                print(self.name + " getting attacking path")
                self.target = self.target_enemy.box
            else: self.movement_mode = "Stationary"
        elif self.movement_mode == "Panicking":
            print(self.name + " getting fleeing path")
            if len(self.world.soldiers) == 0:
                self.movement_mode = "Stationary"
                return
            target = self.world.boxes[randrange(0, len(self.world.boxes))]
            while not self.suitable_fleeing_location(target):
                if len(self.world.soldiers) == 0:
                    self.movement_mode = "Stationary"
                    return
                target = self.world.boxes[randrange(0, len(self.world.boxes))]
            self.target = target
            self.movement_mode = "Flee"
        elif self.movement_mode == "Flee":
            print(self.name + " finished fleeing. Now sitting still.")
            self.sit_still()
            return
        elif self.movement_mode == "Stationary": return
    else:
        print("Error: Invalid agent type submitted to agent.get_new_path(). " + self.name + " defaulting to wandering.")
        target = self.world.boxes[randrange(0, len(self.world.boxes))]
        while target.kind == "X" or target == self.box: target = self.world.boxes[randrange(0, len(self.world.boxes))]
        self.target = target
    self.world.targets.append(self.target)
    self.plan_path(search_modes[self.world.window.search_mode], self.world.window.limit)
```

Figure 14: Agent.calculate_path(), which returns a path for analysis without assigning it to Agent.path; and Agent.get_path(), which manages getting a new path to an appropriate target based on the agent's current state.

In-Simulation Screenshots

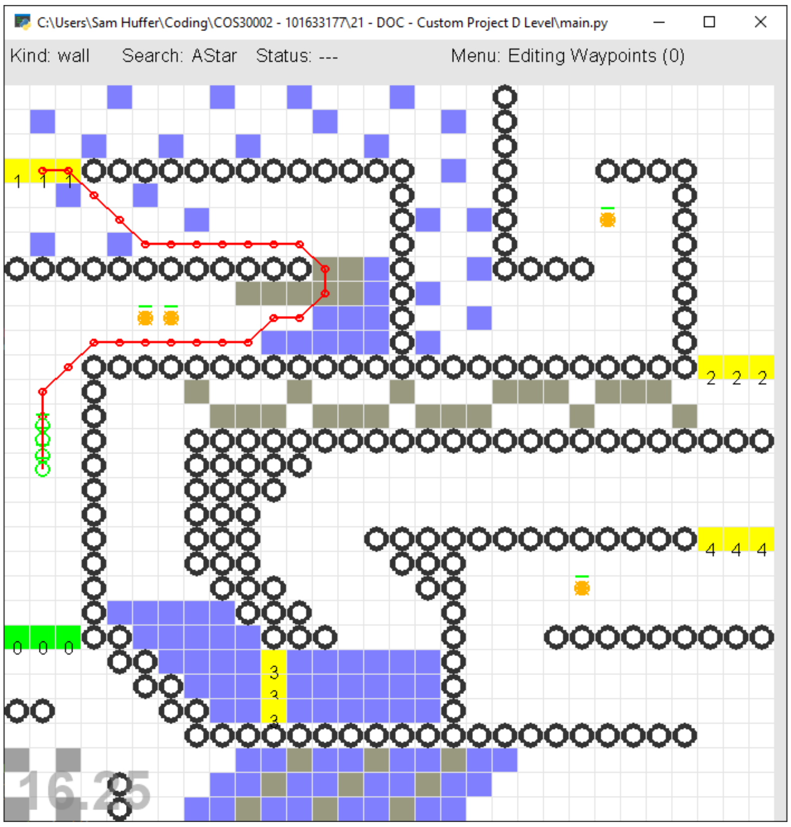

C:\Users\Sam Huffer\Coding\COS30002 - 101633177\21 - DOC - Custom Project D Level\main.py
Kind: wall
Search: AStar
Status: ---
Menu: Editing Waypoints (0)
1 1 1
2 2 2
4 4 4
0 0 0
3 3 3
16.25

Figure 15: soldier agents (green) patrolling to waypoint no. 1, with the soldiers' path (red) displayed (with the lead soldier's featuring circles around each node). Fugitive agents have (orange) have been positioned around the map, and the waypoint editing menu has been accessed to highlight where the waypoints are (green and yellow, numbered).

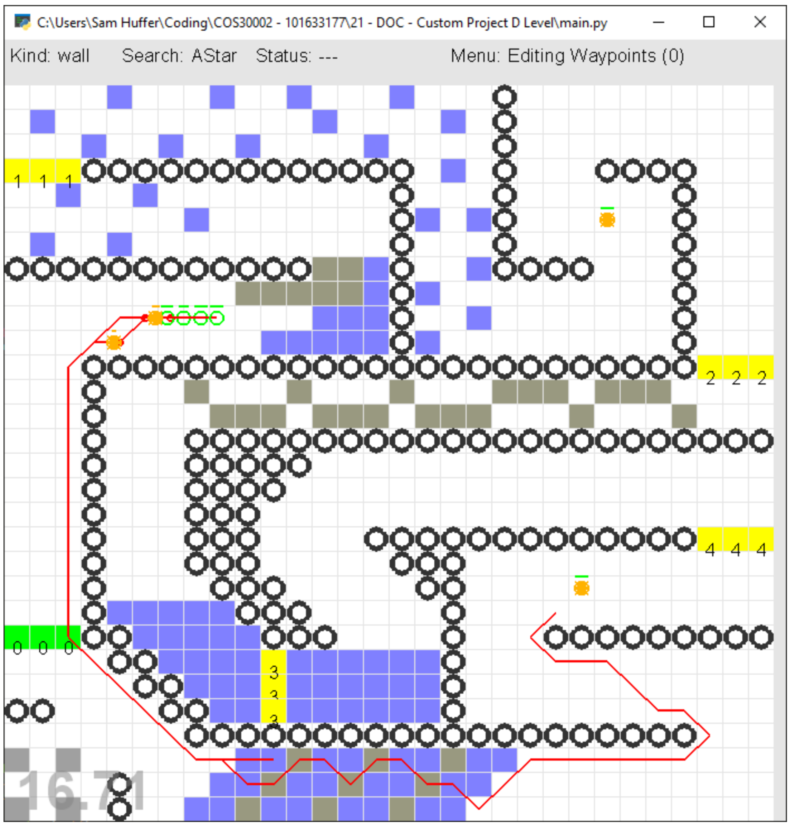
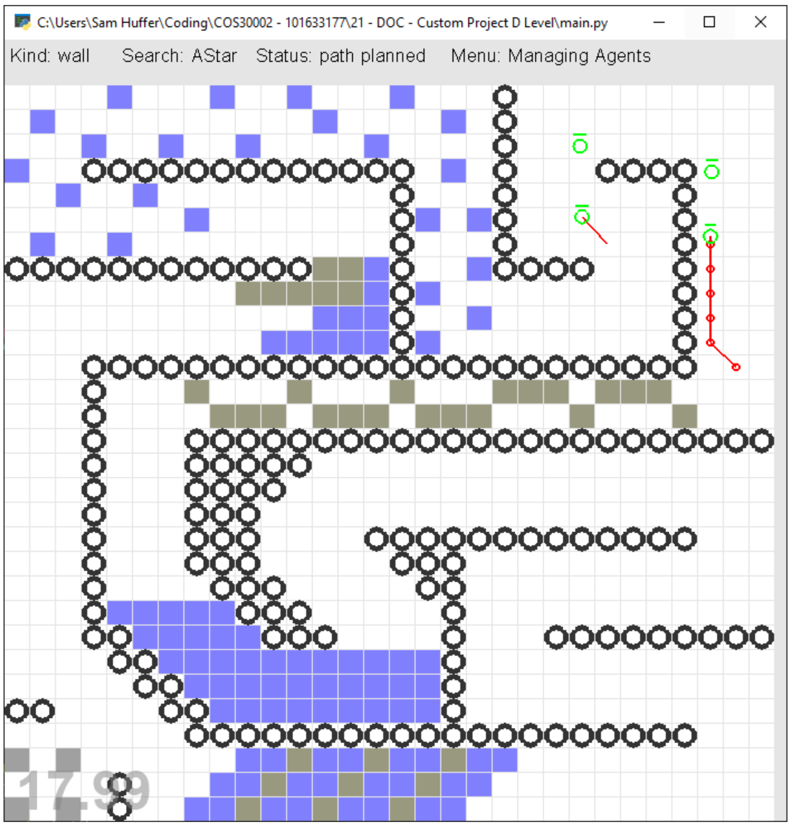
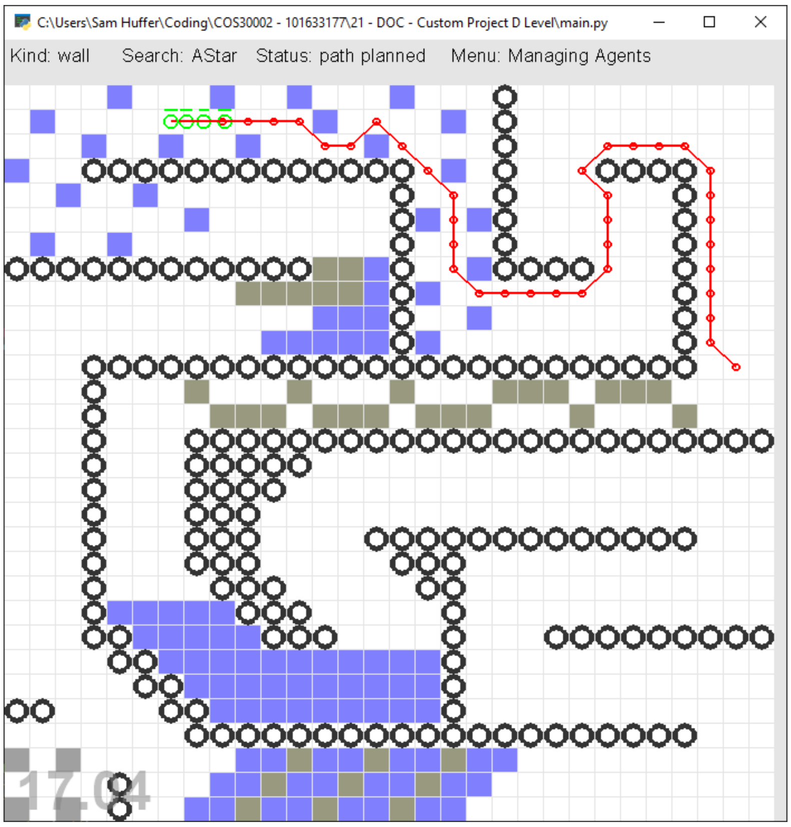

C:\Users\Sam Huffer\Coding\COS30002 - 101633177\21 - DOC - Custom Project D Level\main.py
Kind: wall
Search: AStar
Status: ---
Menu: Editing Waypoints (0)
1 1 1
2 2 2
4 4 4
0 0 0
3 3 3
16.71

Figure 16: the soldier and fugitive agents have encountered each other. One fugitive appears to still be engaged with the soldiers, yet both have panicked and are fleeing to random points on the map in an effort to escape the soldier agents.

Figure 17: soldier agents, having killed their last target, are scouting the surrounding area to check if any more fugitives are present, before resuming their patrol.

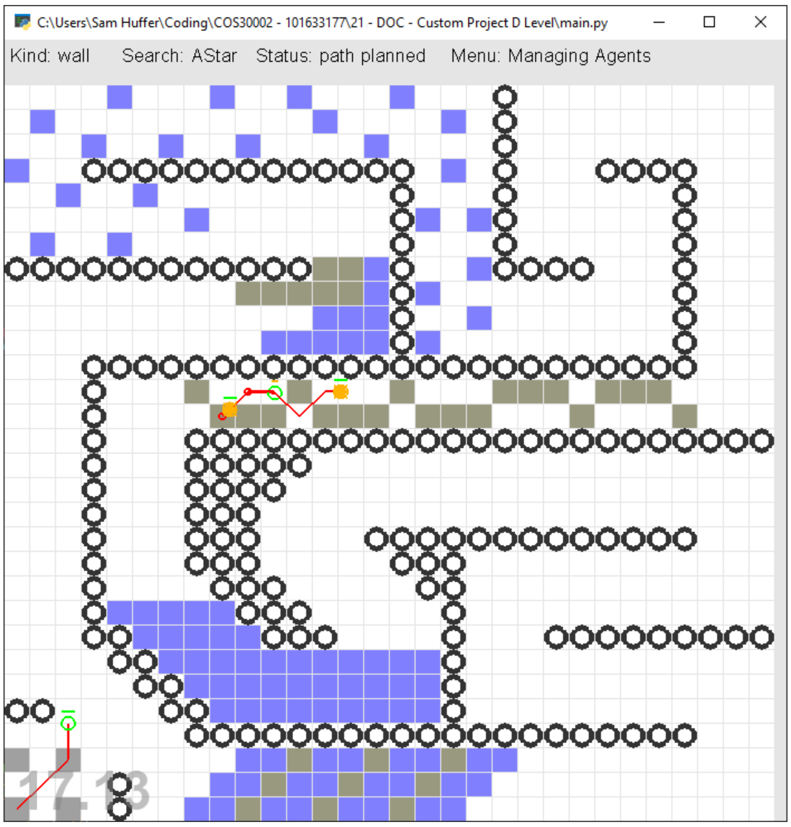

C:\Users\Sam Huffer\Coding\COS30002 - 101633177\21 - DOC - Custom Project D Level\main.py

Kind: wall Search: AStar Status: path planned Menu: Managing Agents


C:\Users\Sam Huffer\Coding\COS30002 - 101633177\21 - DOC - Custom Project D Level\main.py

Kind: wall Search: AStar Status: path planned Menu: Managing Agents

Figure 18: the soldier agents patrolling the box world, heading to waypoint no. 2



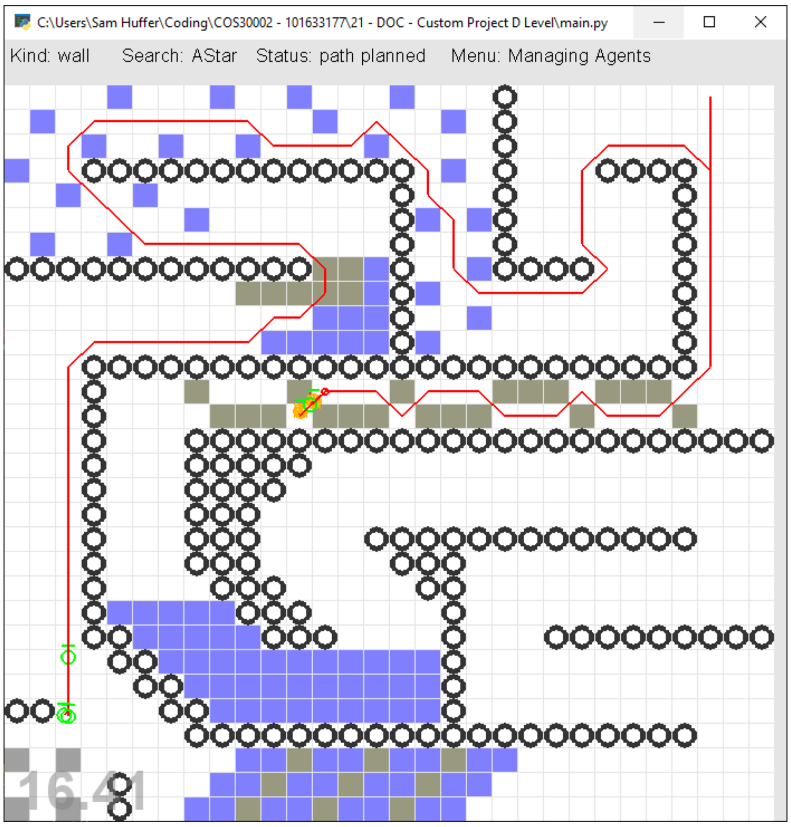
C:\Users\Sam Huffer\Coding\COS30002 - 101633177\21 - DOC - Custom Project D Level\main.py

Kind: wall Search: AStar Status: path planned Menu: Managing Agents

17.13

Figure 19: the lead soldier is engaging two fugitives (centre), while the other remaining soldier (bottom left) is heading back to their base to gather reinforcements after determining that their squad has been overwhelmed by fugitives.

Figure 20: the lead soldier continues to engage the fugitives (centre) while the other soldier has gathered reinforcements (bottom left) and is making its way back to the lead soldier via the shortest path it can calculate.



C:\Users\Sam Huffer\Coding\COS30002 - 101633177\21 - DOC - Custom Project D Level\main.py

Kind: wall Search: AStar Status: path planned Menu: Managing Agents

16.41

Specifications Met

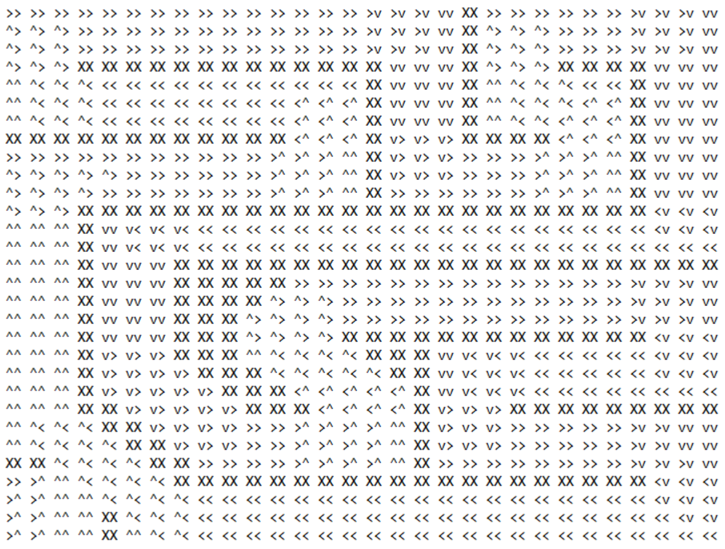
Specification	Met?	Comment
A customizable box world	Y	The box world retains the customizability of the kind of each box, and adds the kind “base” for soldiers to spawn from. Waypoints (numbered 0-9) can have boxes added and removed as well.
Corridors formed from wall tiles	Y	
Patrolled by a squad of soldier agents	Y	
Using group behaviours adapted for graph-based navigation.	Y	The lead soldier picks a target and calculates a path, the rest go to its target, waiting there if the lead soldier is tardy. “Lower ranking” soldiers give way to “higher ranking” soldiers. If a soldier spots a fugitive but another doesn’t, the latter comes to the former’s aid.
Attack fugitives on sight.	Y	
If fugitives flee beyond soldiers’ awareness, they scout the immediate area, resume attacking if they find them, resume patrolling if they don’t.	Y	
If overwhelmed by fugitives, one soldier returns to base to lead reinforcements up to max squad size back to the rest of the soldiers.	Y	Max squad size tied to the number of soldier bases (min. 1, max. 9 or the number of bases, whichever is lower).
Squad respawns if all soldiers die.	Y	
Populated with fugitive agents.	Y	Fugitive agents can be placed in the world by the user. Functionality for automatically spawning fugitives on start was removed but can be restored later.
Stay hidden until they see the soldiers, then attack.	Y	
On attack, fear increases. If it gets too high, fugitives flee.	Y	When fear is over 50/100, there is a random chance of fear/100 that the fugitive will flee each time the fear increments.
Respawn at random points on map when they die.	Y & N	Functionality for fugitives respawning remains in the code, but hasn’t been enabled for fugitives spawned manually.
All agents flee explosives they’re in range of until they finish detonating.	N	Explosive projectiles were removed, as they weren’t firing properly and didn’t seem as key to this unit’s outcomes as other parts of the custom project.

Table 1: the key specifications noted at the beginning of this report, whether they were met or not, and any comments pertaining to how they were met.

Retrospective

What I found out / What I now know

While working on the patrolling of soldiers, I found that single-node waypoints weren't enough to ensure the current waypoint in the soldiers' patrol path would be the correct one when agents can move backwards along the patrol path to attack, scout, and gather reinforcements. I considered a grid-based directional map where each box points the soldier agent to the next box in the patrol. However, I realised this wouldn't be as easy to, given that I specified that the box world would need to be fully customisable, as the ability to place and remove walls would mean the directions assigned to boxes would need to be customizable, not just the kind of the box. This would have required substantial additional work for this project, and could easily have made things messy for players tinkering around with the grid and not paying attention to what they were doing, thereby highlighting that when customizability is involved, it needs to be balanced against the game or simulation's requirement of working properly. To facilitate soldier patrols but retain customizability, I instead modified the waypoints to contain multiple boxes such that they act like trigger colliders positioned such that the player can't avoid them.



```
>> >> >> >> >> >> >> >> >> >> >> >> >> >> >> >v >v >v vv XX >> >> >> >> >> >> >v >v >v vv
^> ^> ^> >> >> >> >> >> >> >> >> >> >> >> >> >v >v >v vv XX ^> ^> ^> >> >> >> >v >v >v vv
^> ^> ^> >> >> >> >> >> >> >> >> >> >> >> >> >v >v >v vv XX ^> ^> ^> >> >> >> >v >v >v vv
^> ^> ^> XX XX XX XX XX XX XX XX XX XX XX XX XX vv vv vv XX ^> ^> ^> XX XX XX XX vv vv vv
^^ ^< ^< ^< << << << << << << << << << << << XX vv vv vv XX ^^ ^< ^< ^< << << XX vv vv vv
^^ ^< ^< ^< << << << << << << << << <^ <^ <^ XX vv vv vv XX ^^ ^< ^< ^< <^ <^ XX vv vv vv
^^ ^< ^< ^< << << << << << << << << <^ <^ <^ XX vv vv vv XX ^^ ^< ^< <^ <^ <^ XX vv vv vv
XX XX XX XX XX XX XX XX XX XX XX XX <^ <^ <^ XX v> v> v> XX XX XX XX <^ <^ <^ XX vv vv vv
>> >> >> >> >> >> >> >> >> >> >> >^ >^ >^ ^^ XX v> v> v> >> >> >> >^ >^ >^ ^^ XX vv vv vv
^> ^> ^> ^> ^> >> >> >> >> >> >> >^ >^ >^ ^^ XX v> v> v> >> >> >> >^ >^ >^ ^^ XX vv vv vv
^> ^> ^> ^> >> >> >> >> >> >> >> >^ >^ >^ ^^ XX >> >> >> >> >> >> >^ >^ >^ ^^ XX vv vv vv
^> ^> ^> XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX <v <v <v
^^ ^^ ^^ XX vv v< v< v< << << << << << << << << << << << << << << << << << << << <v <v <v
^^ ^^ ^^ XX vv v< v< v< << << << << << << << << << << << << << << << << << << << << << <<
^^ ^^ ^^ XX vv vv vv XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
^^ ^^ ^^ XX vv vv vv XX XX XX XX XX >> >> >> >> >> >> >> >> >> >> >> >> >> >> >v >v >v vv
^^ ^^ ^^ XX vv vv vv XX XX XX XX ^> ^> ^> >> >> >> >> >> >> >> >> >> >> >> >> >v >v >v vv
^^ ^^ ^^ XX vv vv vv XX XX XX ^> ^> ^> ^> >> >> >> >> >> >> >> >> >> >> >> >> >v >v >v vv
^^ ^^ ^^ XX vv vv vv XX XX XX ^> ^> ^> ^> XX XX XX XX XX XX XX XX XX XX XX XX XX <v <v <v
^^ ^^ ^^ XX v> v> v> XX XX XX ^^ ^< ^< ^< ^< XX XX XX vv v< v< v< << << << << << <v <v <v
^^ ^^ ^^ XX v> v> v> v> XX XX XX ^< ^< ^< ^< ^< XX XX vv v< v< v< << << << << << <v <v <v
^^ ^^ ^^ XX v> v> v> v> v> XX XX XX <^ <^ <^ <^ <^ XX vv v< v< v< << << << << << << << <<
^^ ^^ ^^ XX XX v> v> v> v> v> XX XX XX <^ <^ <^ <^ XX v> v> v> XX XX XX XX XX XX XX XX XX
^^ ^< ^< ^< XX XX v> v> v> v> >> >> >^ >^ >^ >^ ^^ XX v> v> v> >> >> >> >> >> >v >v vv vv
^^ ^< ^< ^< ^< XX XX v> v> v> >> >> >^ >^ >^ >^ ^^ XX v> v> v> >> >> >> >> >> >v >v vv vv
XX XX ^< ^< ^< ^< XX XX >> >> >> >> >^ >^ >^ >^ ^^ XX >> >> >> >> >> >> >> >> >v >v >v vv
>> >^ ^^ ^< ^< ^< ^< XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX <v <v <v
>^ >^ ^^ ^^ ^< ^< ^< ^< << << << << << << << << << << << << << << << << << << << <v <v <v
>^ >^ ^^ ^^ XX ^< ^< ^< << << << << << << << << << << << << << << << << << << << << <v <v
>^ >^ ^^ ^^ XX ^^ ^< ^< << << << << << << << << << << << << << << << << << << << << << <<
```

Figure 21: the directional map that I devised in map3.txt, but commented out upon abandoning it.

While building this custom project, I found that graph searches were good for more than just determining paths for agents to move along to get to a target location. I also ended up using them for: checking whether enemy agents within range can reasonably be reached or if they were on the opposite side of a wall and there wasn't a gap in the wall close enough for the agent to reach the enemy agent, by checking that each node in the path was within the agent's awareness range; and correcting the current waypoint when a lead soldier died and its successor was positioned between different waypoints to the original lead soldier, by getting a path from the lead soldier to its successor and simulating the lead soldier moving along it to their successor, triggering waypoints as was appropriate

What I'd do different / What I'd change

If I had to do this project again and had sufficient time to do so, I'd want to implement the project using a strongly-typed language like C# using a more powerful editor than SublimeText, so as to cut down the time spent fixing syntax and type errors, and make debugging and monitoring the values of variables easier. Perhaps building it in Unity and monitoring agents in the inspector would make debugging easier again, and offer a possibly better 3D view of the simulation environment. Certainly, it might have made implementing patrol waypoints and collision simulation easier with existing means for both to be achieved in Unity.

Regarding the design of the project itself, perhaps including more/better separation between agents than "defer to your superiors" would have helped clean up the appearance of the combat between soldier and fugitive agents, stopping them from becoming a jumbled mess of circles.

Further changes I'd suggest

A force-based version of this custom project would be interesting to see implemented. It would require more collision detection and avoidance code, but it would give agents smoother-looking movement than strictly following angular paths does.

If one were to continue with the current graph-based approach, one thing the current version suffers from is that agents, while following their paths, sometimes get stuck for seemingly no reason (or at least none that I could figure out), and get stuck trying to move to the next node in the path for a moment. I work around this by forcing a recalculation of the path if it gets stuck for too long, and this seems to work okay, albeit not perfectly, as it does eventually force the agents to continue on their paths, even if it doesn't look the nicest. A future iteration that addresses or bypasses this issue and results in seamless movement along paths would be something to consider.

Another change to a future graph-based iteration could be to remove diagonal graph connections immediately adjacent to a wall box, such that an agent following its path won't look like they're cutting across the external whitespace of wall boxes when moving diagonally between boxes adjacent to the wall box. If this were implemented, I'd also want to tweak the wall boxes to render as squares again rather than circles, as accommodating agents cutting across wall boxes would no longer be a concern.

Additionally, explosive projectiles were removed from this iteration because they were proving tricky to manage but weren't essential to the project and didn't demonstrate this unit's outcomes as strongly as other aspects of it. A future iteration could restore them to the simulation and refine them such that they look like they're firing and detonating properly, akin to what was achieved in the force-based spikes where explosive projectiles were present.

Lastly, a further possible extension that occurs to me is to change the simulation into a game, perhaps a twin-stick top-down shooter, whereby a player controls one of the soldier or fugitive agents, or another type entirely, and sees how long they can survive in the box world, or switches control from agent to agent on one side of the conflict and strives to eradicate all agents of the opposing side.