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 hand­gun, 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 sol­diers 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.

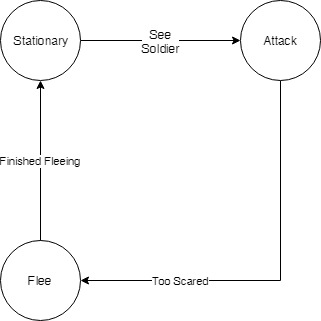


Figure 2: the final state diagram for the fugitive agents.

# UML Class Diagram

Figure : the UML class diagram for the whole program.

# Code Snippets

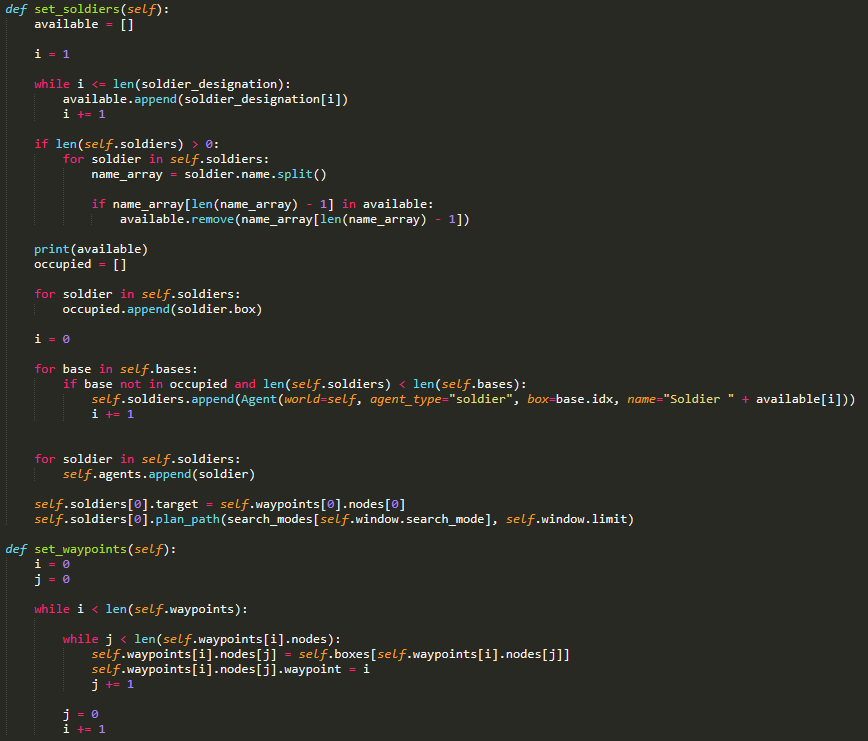


Figure : BoxWorld.FromFile(), modified to load waypoints into BoxWorld.

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

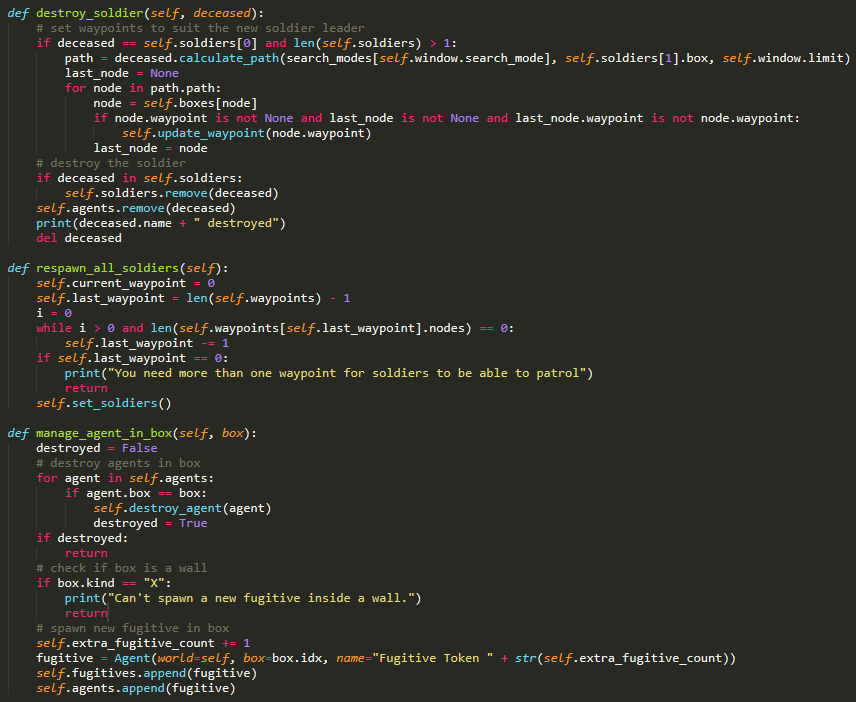


Figure 7: BoxWorld’s methods for destroying soldiers (whether on-click or at the hands of a fugitive), respawning them all if they all die, and managing what happens when they are clicked (destroy\_agent() redirects to destroy\_soldier() and destroy\_fugitive() as appropriate, the latter just removing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

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()).

# 

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

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

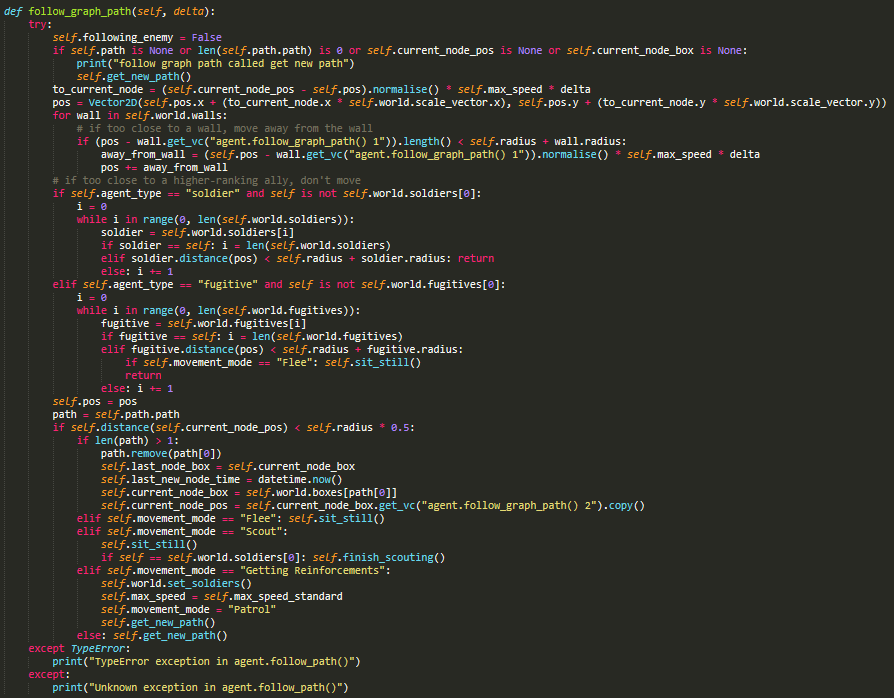
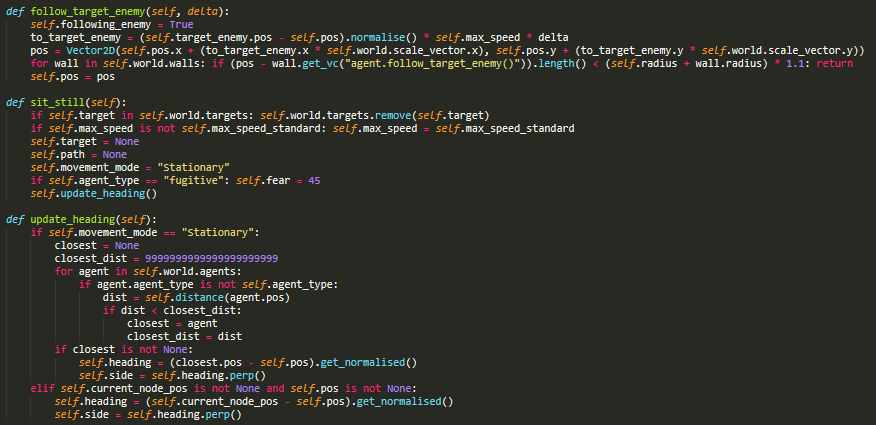


Figure 11: additional movement-related methods for following an enemy when in the same square as them (follow\_target\_enemy()), rendering an agent motionless (sit\_still()), and orienting them in the appropriate direction (update\_heading()).

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

# 

Figure 12: methods used by Agent.aim() to predict agents’ positions on their path at a specified point in the future.

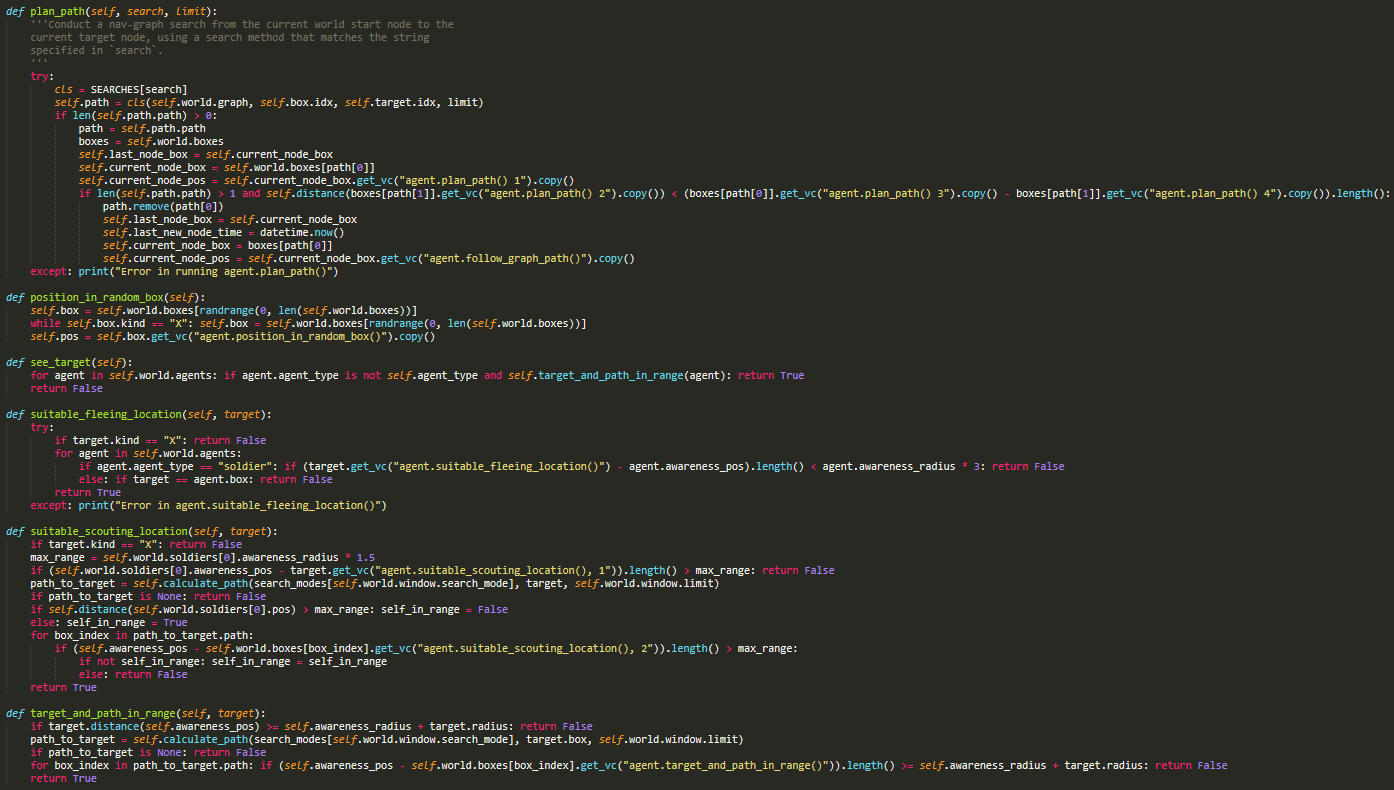


Figure13: additional supportive methods in Agent: plan\_path(), customized for agent moving in a graph-based environment,; see\_target(), which checks if an enemy is in range and the agent could reasonably get to it; suitable\_fleeing\_location() and suitable\_scouting\_location(), used by fugitives 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.

# 

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

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).

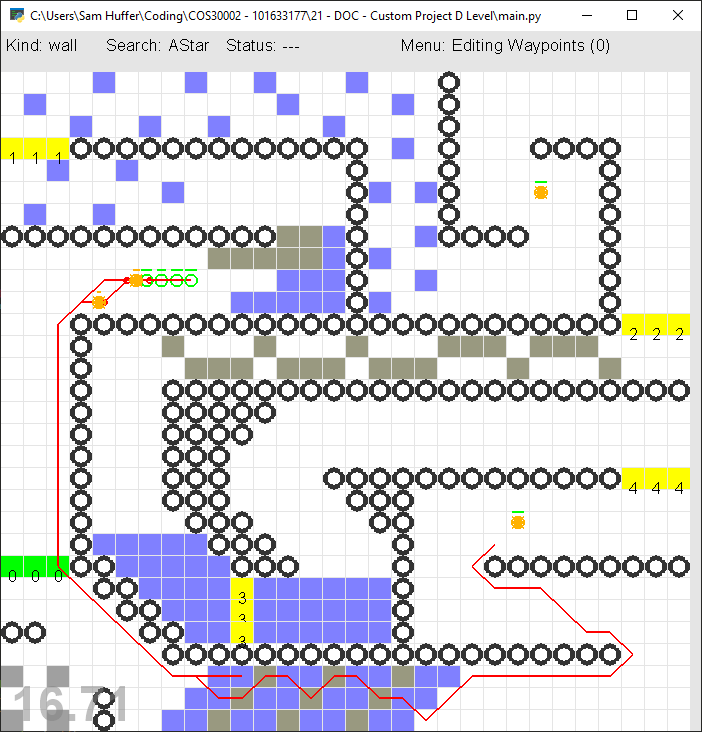
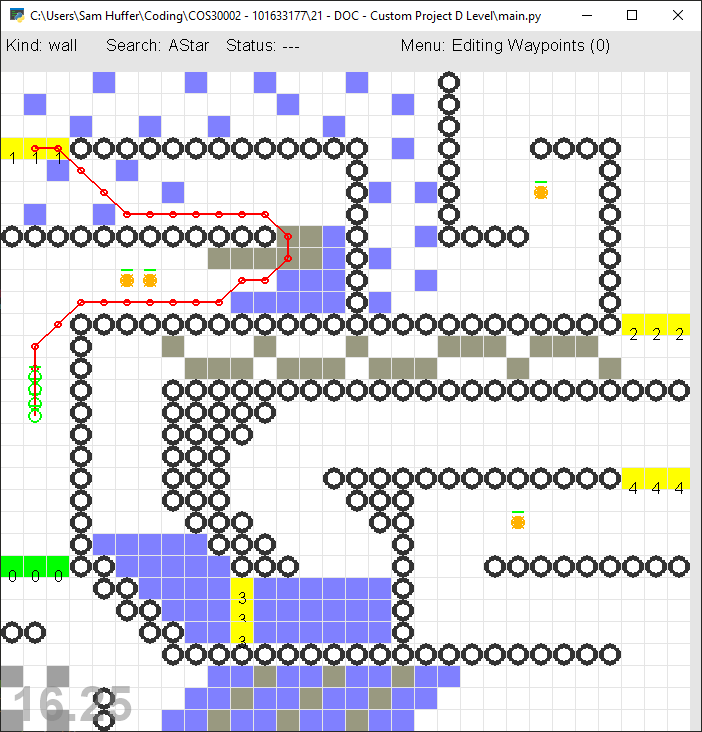


Figure 16: the soldier and fugitive agents have encountered each other. One fugitive appears to still be engaged with the soldiers, yet both have panicked and are fleeing to random points on the map in an effort to escape the soldier agents.

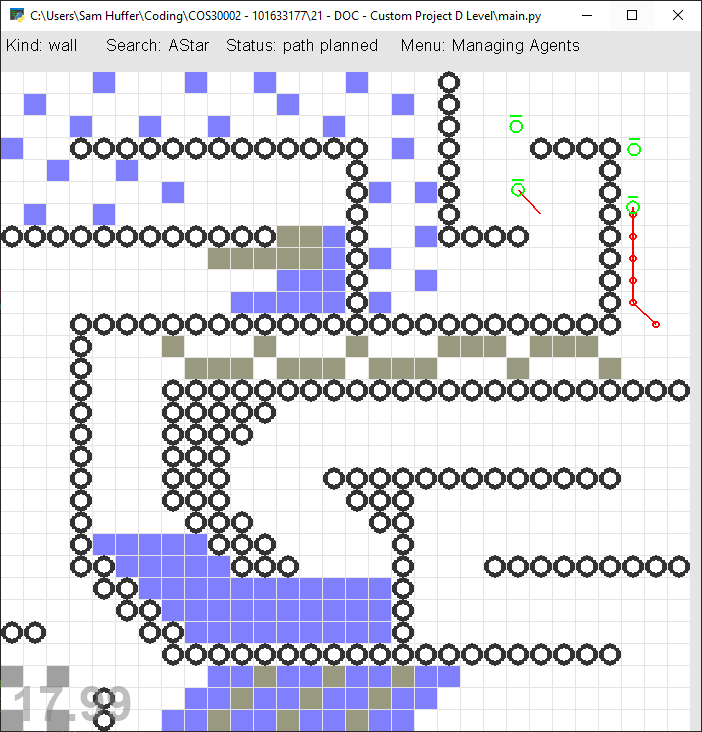
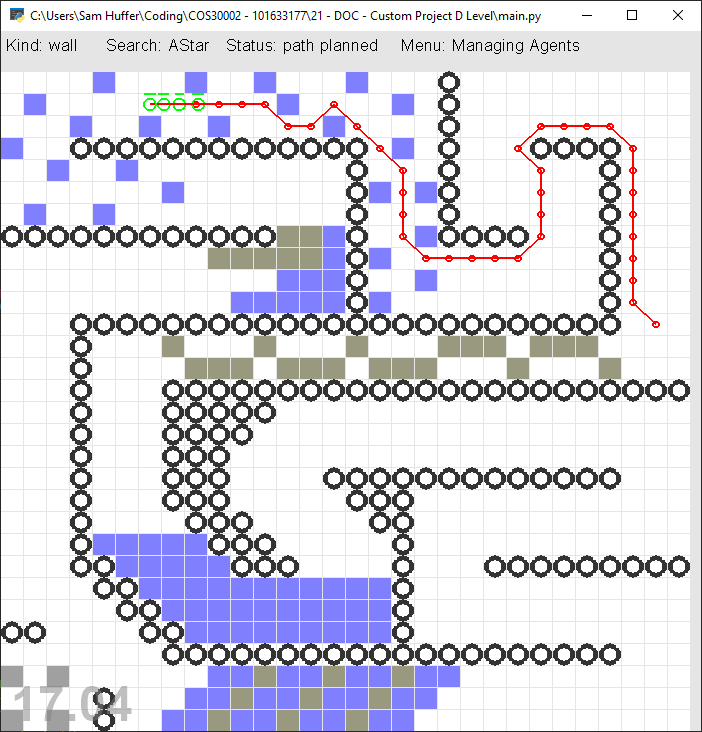


Figure 18: the soldier agents patrolling the box world, heading to waypoint no. 2

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.

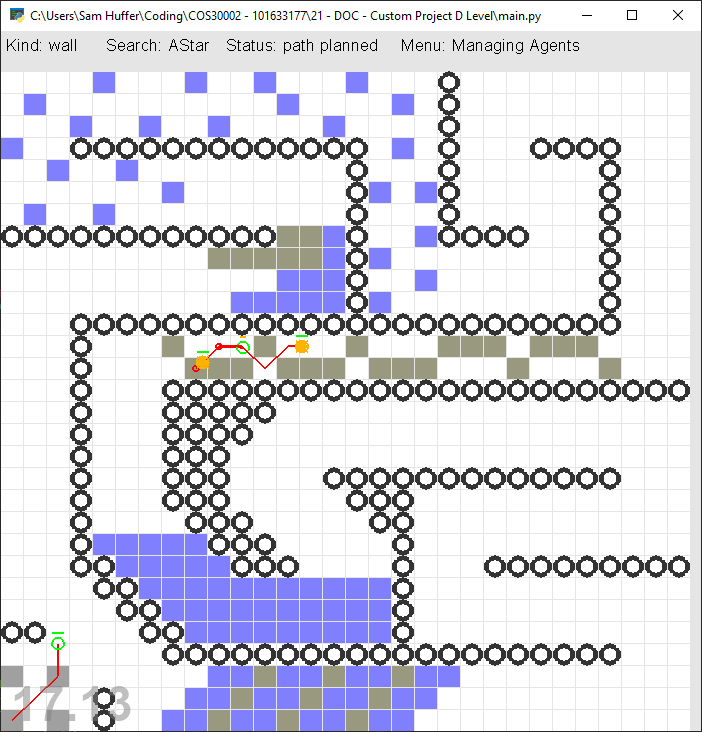


Figure 19: the lead soldier is engaging two fugitives (centre), while the other remaining soldier (bottom left) is heading back to their base to gather reinforcements after determining that their squad has been overwhelmed by fugitives.

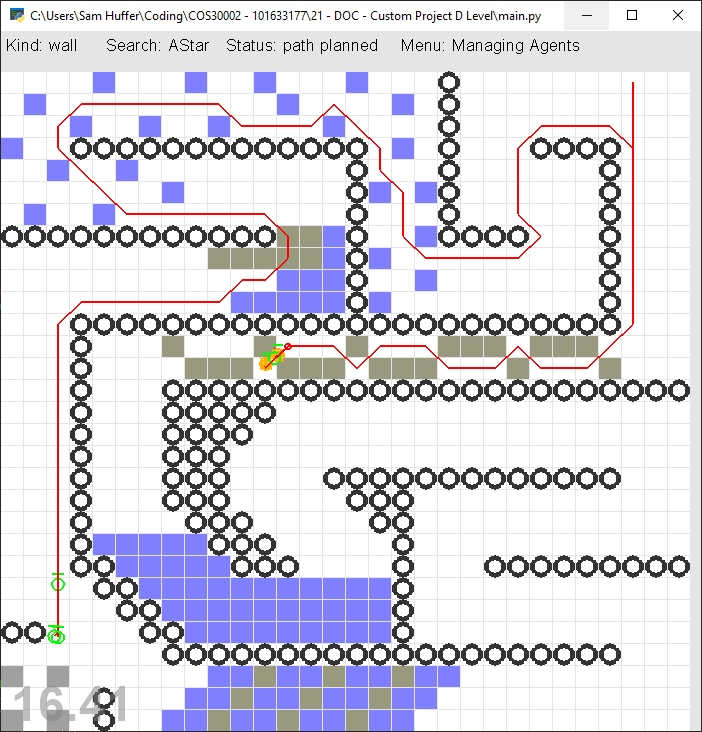


Figure 20: the lead soldier continues to engage the fugitives (centre) while the other soldier has gathered reinforcements (bottom left) and is making its way back to the lead soldier via the shortest path it can calculate.

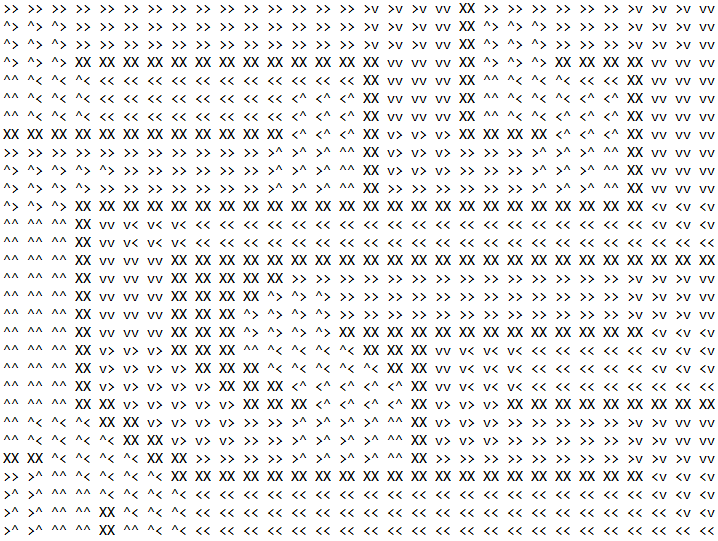
# Specifications Met

|  |  |  |
| --- | --- | --- |
| **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 : 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.

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

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

## 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.