Project 1: Ghosts in the Maze

Course: 16:198:520

Instructor: Professor Cowan

Michael Neustater

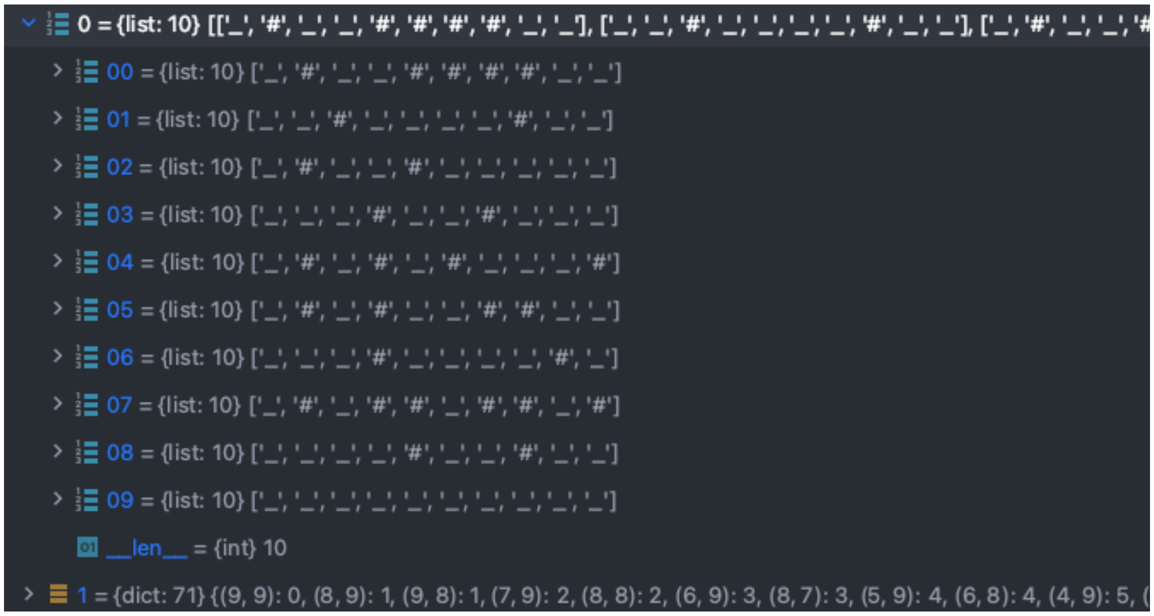
Purpose:

To gain experience implementing and using search algorithms and to highlight the differences between planning and execution. The project consists of 4 Agents (and some additional variations of these Agents), all of which use varying techniques in order to plan and solve 51x51 mazes filled with moving ghosts (which can kill the Agent) with the goal of surviving. The expectation was that each Agent would become increasingly better at surviving (with the potential increased cost of time for planning).

Design Choices:

Maze (maze.py)

The maze is generated as a 2D list and checked using BFS. Along with the maze is a dictionary that stores the length of the shortest path from a given node to the goal. (the reasoning for using BFS and storing these values will be explained later in the strategy question section) Both data structures are stored together in an array then exported to a file using pickle allowing them to be used reused. The same 100 mazes are used for a testing instance (meaning that test i for j ghosts will use the same maze as test i for j+1 ghosts). Each time main is rerun, the mazes will be regenerated and restored in the file. The generation and storing of mazes to files is handled by maze_generator.py. All testing is done with a 51x51 maze unless otherwise specified or shown. On generation, there is a 28% chance a given cell is blocked, if a maze is unsolvable according to BFS, a new maze is generated.



```
0 = {list: 10} [['_', '#', '_', '_', '#', '#', '#', '#', '_', '_'], ['_', '_', '#', '_', '_', '_', '_', '#', '_', '_'], ['_', '#', '_', '_', '#
    00 = {list: 10} ['_', '#', '_', '_', '#', '#', '#', '#', '_', '_']
    01 = {list: 10} ['_', '_', '#', '_', '_', '_', '_', '#', '_', '_']
    02 = {list: 10} ['_', '#', '_', '_', '#', '_', '_', '_', '_', '_']
    03 = {list: 10} ['_', '_', '_', '#', '_', '_', '#', '_', '_', '_']
    04 = {list: 10} ['_', '#', '_', '#', '_', '#', '_', '_', '_', '#']
    05 = {list: 10} ['_', '#', '_', '#', '_', '_', '#', '#', '_', '_']
    06 = {list: 10} ['_', '_', '_', '#', '_', '_', '_', '_', '#', '_']
    07 = {list: 10} ['_', '#', '_', '#', '#', '_', '#', '#', '_', '#']
    08 = {list: 10} ['_', '_', '_', '_', '#', '_', '_', '#', '_', '_']
    09 = {list: 10} ['_', '_', '_', '_', '_', '_', '_', '_', '_', '_']
    __len__ = {int} 10
1 = {dict: 71} {(9, 9): 0, (8, 9): 1, (9, 8): 1, (7, 9): 2, (8, 8): 2, (6, 9): 3, (8, 7): 3, (5, 9): 4, (6, 8): 4, (4, 9): 5, (
```

Example of an instance of a maze object
(a maze object is a tuple of the 2D list and the dictionary of visitable cells)

Ghosts:

The ghosts were implemented using two data structures, a dictionary, which has the key of a tuple of the x and y coordinates (two ghosts at the same position share an entry here) and a heap queue, with the sorting value being the ghost distance from the Agent (each ghost has an independent entry). The dictionary allows for O(1) lookup to see if a ghost exists at any node, which is especially useful for checking quickly for collisions and to check if the ghost at a node is in a wall (a false result means the ghost is in a wall and true means it is visible). The python implementation of a heap queue allows for a O(log n) removal of the closest ghost and O(1) to check, while reducing the need for manual sorting upon insert and deletion. In testing, the python function heapify was faster than sorted, so a heap was chosen over a sorted list for this case, however since the python implementation of heap queues does not feature peak and often the ghost does not need to be removed or inserted, but just checked, if I were to reimplement ghosts I would most likely change to a sorted list.

The initial spawning of ghosts is limited only to cells that the player can visit, however they can pass through walls into "blocked off" sections of the maze. Ghosts move in a given direction randomly and if they are in a wall, they have a 50% change of staying still. To prevent leaving the maze, the coordinates are checked to be greater or equal to 0 and less than a stored value for maze size, allowing the maze size to be changed. For testing of Agents 1-5, for a given test iteration i with j ghosts, all Agents will start with an identical pattern of ghosts in the attempt to keep testing somewhat uniform. The ghost movement varies since it is influenced by randomness factors determined in the writeup.

Agents:

All Agents use A* for path planning. The heuristic for Agents 1-3 uses the shortest path generated at maze creation and Agents 4 and 5 also use this in addition to more data, which will be explained later. A* was chosen due to its speed and relatively good size complexity. It also allows for heuristic optimization for further speed improvement. For all Agents, an Agent moves, then the ghosts immediately after, meaning that if an Agent moves into a ghost square it dies, but also if an Agent moves, a ghost may then also move into its square. This is to prevent cases where a ghost and Agent pass through each other by swapping spaces.

Agent 1

Agent 1 calls the A* algo with no ghosts in the maze, then the ghosts are added to the maze, and it walks the path until it finishes or dies.

Agent 2

Agent 2 calls a bootstrappable Agent function, which is generic Agent 2 behavior as described in the outline. The Agent replans a new path with each step it takes always attempting to find and follow the shortest path. This bootstrappable implementation allows it to also be called by gent 3 many times for simulations. It contains a parameter called version, allowing it to have some modifications depending on if Agents 3, 4 or 5 call it. Further it has a parameter replan_fail, which if true means that if an instance cannot find a path it will return fail instead of moving away (this is used to speed up Agent 3).

Agent 3

The goal of Agent 3 is to act as a more informed version of Agent 2, trading the extra planning cost for improved performance. It is run using 8 simulations in all four directions and standing still, which simulate with the same behavior of Agent 2 by using the bootstrappable Agent function. Tie breakers are decided using the shortest average path of simulated runs. "Standing still" is simulated by standing still for a single move, while the ghosts move one step, followed by a regular Agent 2 behavior. All simulated scenarios use an exact copy of the current position of the ghosts in attempt to make them as accurate as possible. The replan_fail feature is used in order to decrease runtime. The success rate with this on or off remains similar, however the runtime decreases, since the simulations may stop sooner instead of waiting for death or for a path to open.

Agent 4

Agent 4 is like Agent 2 (and therefore also uses the bootstrappable function), in that it will run a search at each step, the major change being it does not always find the shortest path. Instead, it attempts to find a balance between shortest path and avoiding ghosts. It does this by using its own validation function, which creates a "danger" weight (variable titled offset in code) and adds it to the shortest distance to the goal as a heuristic. The danger weight is calculated by taking all ghosts that are 6 units away or less by Euclidian distance and subtracting their distance from 6 then adding the results together. Ghosts that are in walls are weighted less at 0.8 times the value of a visible ghost. Originally instead of subtracting a constant value from 6, I attempted to use a reciprocal function so that closer ghosts were significantly more "dangerous" than further ones, however in testing I found the survivability to be worse. The purpose of weighting ghosts in walls as lower is because they have a much higher chance of not moving in a single move or moving away from the player. A value of 0.5 and 0.25 were also tested but 0.8 had a constantly slightly higher success. In the case that there is no path that exists to the goal, Agent 4 will also attempt to retreat, however it will only retreat if a ghost is within 6 cells. This was chosen to avoid straying too far away from a position that may be closer to the finish, since Agent 4 already tends to not take short paths.

The current implementation of Agent 4 uses the distance of ghosts through walls, meaning that some ghosts behind walls may be weighted higher than a visible ghost. Originally, I attempted to do a BFS with a limit of 6 units from the Agent in order to only consider ghosts that the player could walk into, however this had two downsides. The first of which was running a BFS search at each walk step is relatively slow compared to the current approach and the second was that this did not consider that ghosts in walls could travel through walls and quickly become a threat.

Strategy Questions (Grey Boxes):

1.1. The algorithm I found most useful for checking the existence of paths in a newly generated maze was BFS. While BFS is more memory intensive than DFS and slightly slower than A*, by executing a complete BFS search on the entirety of the maze from the goal up to the start, the shortest path from every visitable cell to the goal can be found extremely quickly. This is useful for finding all visitable cells (needed for spawning ghosts), as well as the shortest path from each node to the end point, allowing for a heuristic that is more accurate than Manhattan distance when executing A* for Agents.

1.4. (Agent 2) When executing Agent 2, you should replan at each step since Agent 2 is intended to follow the shortest path possible from a given position at each walk step. If a previously planned path is reused, it is possible that the current path has been blocked or an even shorter path has been opened. There are some ways to avoid replanning, since the shortest path from each cell is already known (found during maze creation), you could check if the previous path planned is the same distance as the shortest path from the new cell and that that path is not blocked. I originally created a model like this and found that it does result in a much faster run time when many iterations are being run, especially in mazes with few ghosts since paths often are not blocked, however after observing the paths traveled and discussing it with the professor, I chose not to do this because it results in path biasing that makes the paths not fully representative of the A* algorithm. So, while it is possible not to replan, I have chosen not to.

(Agent 3) If there is no successful path projected in its future, the Agent attempts to "run away" from the nearest ghost that is visible. This choice was made in order to conform to the idea that Agent 3 is supposed to behave as a more informed Agent 2. In these cases where all simulations fail, success is not impossible, this is because the number of simulations is relatively low and therefore not fully representative of what will happen. This is especially true in mazes that have bottle necks, meaning that all paths must follow through one cell in order to reach the end. In these cases, especially with few ghosts, a ghost may block the bottle neck for a few turns, and the time spent retreating allows for time in which the path can open again.

Computational Bottlenecks:

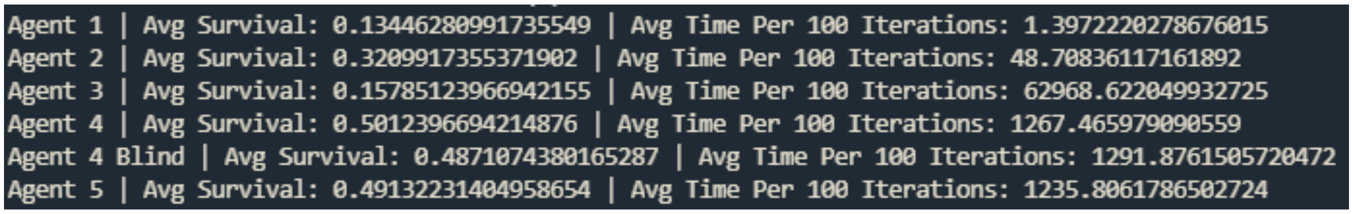
Most Agents run in a reasonable amount of time in a single threaded instance, however due to the extremely high number of searches required for Agent 3's simulations, single threading was not a viable option given Pythons slow performance. Python multithreading provided minimal speedup for the type of operations required by the Agents and many of the libraries chosen are not thread safe, so Python multiprocessing cannot be used. In order to overcome slow single threaded performance, subprocesses are used, allowing for independent instances of Python scripts. For each number of ghosts i from 0 to 119, a new subprocess is created using ag_subprocess.py, which will test all Agents sequentially for 100 mazes with I ghosts. In attempt to remain somewhat consistent, for a given maze and number of ghosts, all Agents start with the same ghost pattern. Using subprocesses, the code can be spread across 120 threads allowing for much faster processing and preventing a particularly slow Agent from being detrimental to the overall execution time. Once a thread is completed the results of a thread are stored in a pickle file in a folder corresponding to the Agent number and once all threads are completed the data is extracted from the pickle files and combined.

Due to the extremely high thread count, this program should be run on the 80 – 96 thread servers on iLabs. They will utilize approximately 100% of the CPU at the start and will decrease as threads finish. Doing so allows the total execution to take a little over 29 hours, there is a variable in main allowing threads to be decreased to a more reasonable amount for local running. If these values are changed, thread value must be less than the total number of ghosts. In this case a single thread will handle 100 iterations for more than one ghost (ie: it will run 100 iterations for i ghosts, then i + 1 ghosts etc). The result is a much slower execution; however, it is still much faster than true sequential.

Displaying Data/ Modeling:

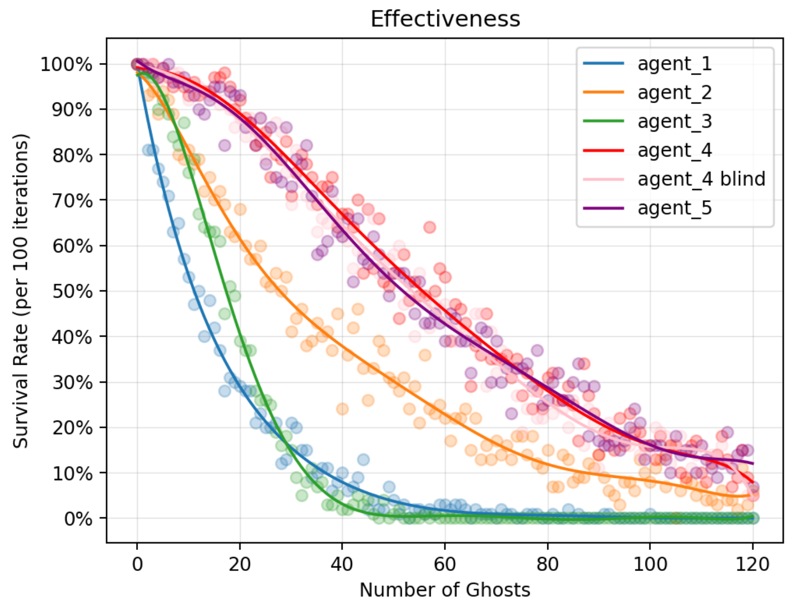
Due to the way the results are stored, display_data.py can parse the pickle files data and re-output the results of a past run if there are valid stored results. It also uses matplotlib in order to create an overall survival graph and performance graphs for each Agent. The best fit line is modeled with a 7th degree polynomial in order to more accurately display the approximate survival rate for a given number of ghosts and a 5th degree polynomial for displaying the runtime graph. The display_data.py uses thread and ghost values set in main in order to determine the naming of the files it will pull data from, meaning if these values are modified in main after the run completes, display_data.py will not be able to properly parse the data unless these values are reverted, or a new run completes.

Results:



```
Agent 1 | Avg Survival: 0.13446280991735549 | Avg Time Per 100 Iterations: 1.3972220278676015
Agent 2 | Avg Survival: 0.3209917355371902 | Avg Time Per 100 Iterations: 48.70836117161892
Agent 3 | Avg Survival: 0.15785123966942155 | Avg Time Per 100 Iterations: 62968.622049932725
Agent 4 | Avg Survival: 0.5012396694214876 | Avg Time Per 100 Iterations: 1267.465979090559
Agent 4 Blind | Avg Survival: 0.4871074380165287 | Avg Time Per 100 Iterations: 1291.8761505720472
Agent 5 | Avg Survival: 0.49132231404958654 | Avg Time Per 100 Iterations: 1235.8061786502724
```

Results of avg survival and time per 100 iterations for all fully tested Agents (time per iterations in seconds)



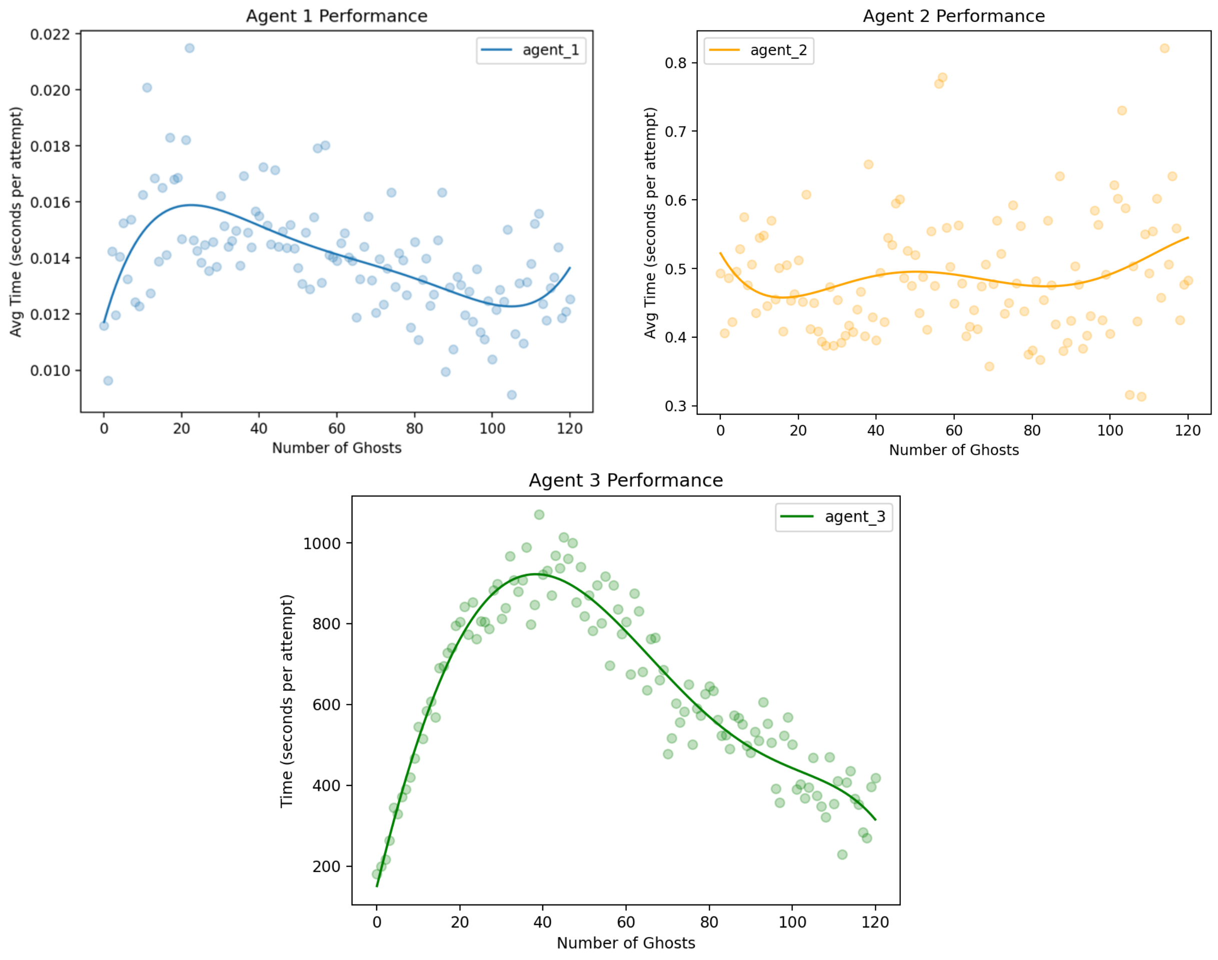
Effectiveness Scatterplot/ Graph of all fully tested Agent Runs from 0 – 119 Ghosts

Analysis:

Comparing Agents 1, 2 and 3:

Agent 2 was consistently the most successful of the first three Agents, followed by Agent 3 than 1. Agent 3, while starting out about as good as Agent 2 and usually beating out Agent 1, quickly becomes the worst of all three Agents around 30 ghosts. Agent 3 hits 0 survivability around 50 ghosts, and Agent 1 shortly after around 60. Agent 2 maintains single digit survivability much longer only approaching zero around 120 ghosts. Ultimately Agent 3 can be considered a failure since it takes much longer than Agent 1 (further explained in the next paragraphs) and performs worse than Agent 2. Agent 3's low success rate, is a direct result of the dependency on simulations. Simulations, especially those that are far from the goal, are prone to being very different than the actual movement of ghosts. This is because the ghosts move randomly on each step of the player and the player takes many steps to the goal, meaning ghost movement varies significantly between two simulations. Using successful simulations as the single factor for choosing a direction means there is a chance a "dangerous direction" is more successful due to ghosts randomly moving away or a "safe direction" is less successful because ghosts happen to move towards the Agent. This causes Agent 3 to "wiggle" meaning it that it will go back and forth between several states, this results in slower execution times and higher death rates. This issue will be explored further in the next sub-section.

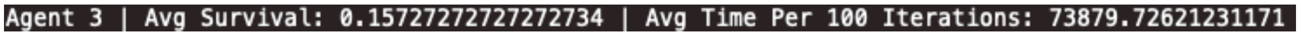
Ultimately, Agent 1 performs fastest, which is mostly due to the fact that it plans once and follows the path until completion or the goal. Additionally, the high death rate results in it often not completing runs and results in an even lower average time, meaning mazes with more ghosts often have shorter attempt times (walking further in mazes takes more time due to the increased cost of moving a large number of ghosts per step). Agent 2 performs second fastest, beating out Agent 3 even though Agent 2 also has a higher success rate. Furthermore, it has a fairly consistent time for execution, this is due to the fact that it only plans once per step and always attempts the shortest path, the increased cost of moving more ghosts is offset by the fact that Agent 2 dies more often with more ghosts in a maze making the time of execution nearly constant for all ghost amounts. Agent 3 performs the worst of the three in terms of speed taking over 1300 times longer than Agent 2 and it barely outperforms in survivability Agent 1. It is slowest around 40 ghosts, where wiggling is extremely prominent, but the number of ghosts in the maze is not high enough to result in a quick death.



Performance graphs for Agents 1-3, avg time in seconds for a singular run to complete for a given number of ghosts

Agent 3 Flaws/ Wiggling Issue

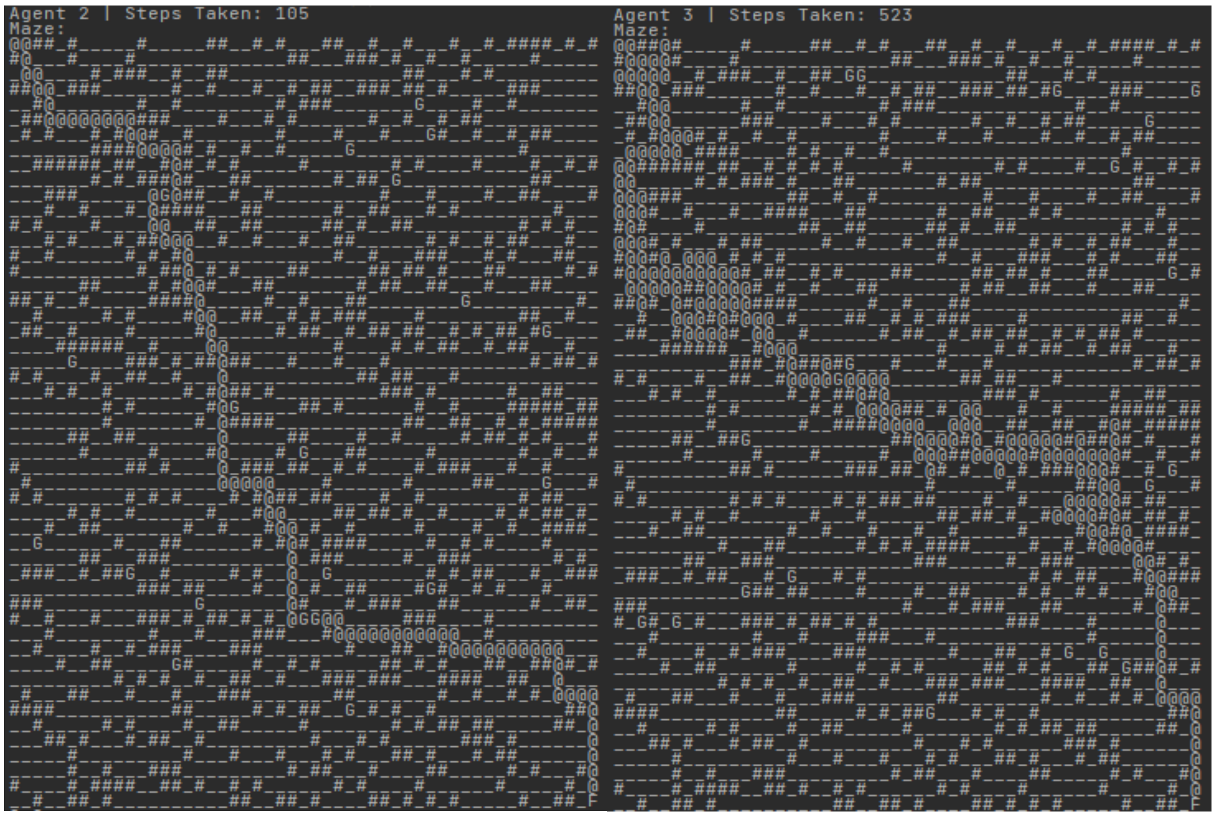
When using a low number of simulations wiggling is most prominent (ie: 3), resulting in very slow execution. Increasing the number of simulations slightly (to 8), the simulations speed up because of the reduction in wiggling, however wiggling still occurs enough to result in a slow execution and poor performance. This can be seen below, when using 3 sims instead of 8, on average 100 simulations take about an additional 1,000 seconds (16.5 minutes)



```
Agent 3 | Avg Survival: 0.15727272727272734 | Avg Time Per 100 Iterations: 73879.72621231171
```

Results of avg survival for Agent 3 with only 3 sims
(time per 100 iterations in seconds)

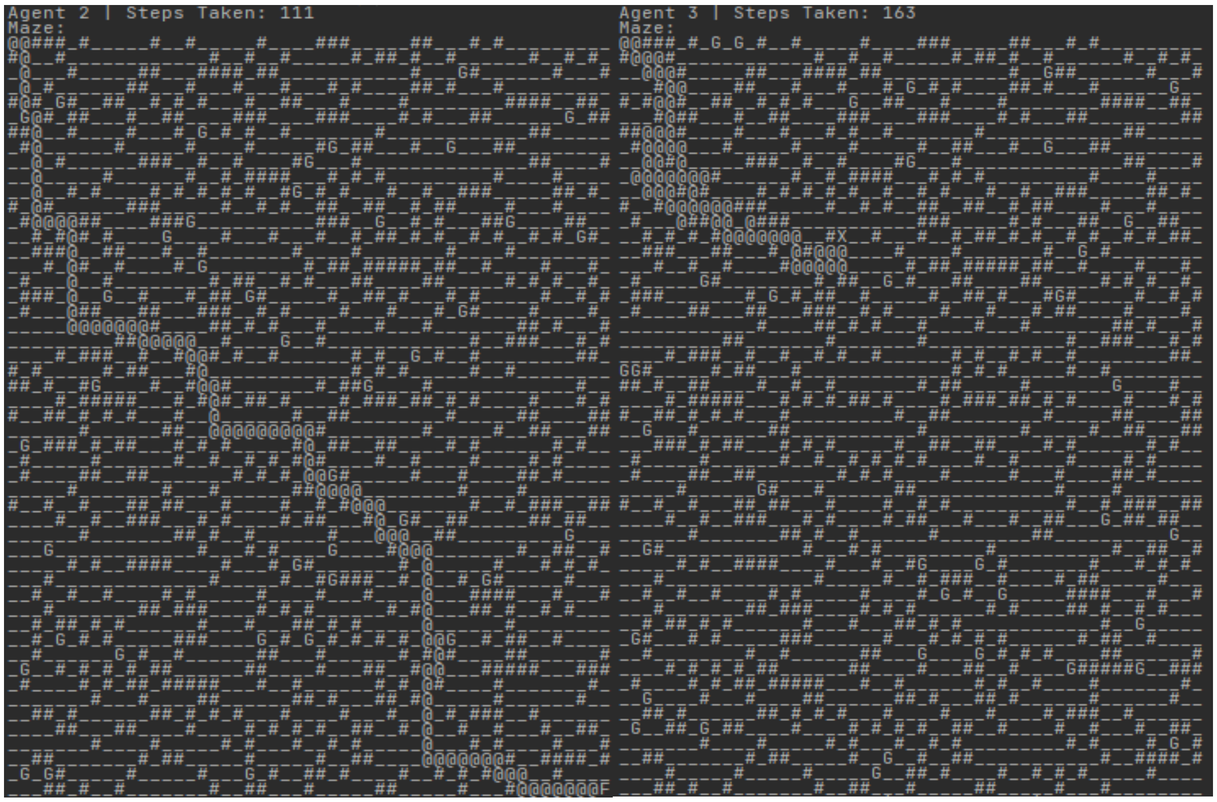
Ghosts that wiggle more tend to spend longer in the maze and as a result have a much higher chance of dying. This can be seen in the two maze examples (see below) which feature Agent 2 and 3 facing two different identical mazes with 20 and 40 ghosts respectively. In both cases Agent 3 requires significantly more steps before it either solves or dies. The thicker path indicates significant wiggling between states.



```
Agent 2 | Steps Taken: 105
Maze:
@@##_#_____#_____##__#_#___##__#__#___#__#_####_#_#
#@___#____#_____________##___###_#__#__#_____#_____
_@@____#_###__#__##_______________##___#_#_________
##@@_###______#__#___#__#_##__###_##_#_____###_____
__#@_______#__#________#_###_______G____#__#_______
_##@@@@@@@@###____#___#_#______#__#__#_##__________
_#_#___#_#@@#__#_______#____#___#___G#__#__#_##____
_______####@@@@#_#__#__#_____G______________#______
__######_##__#@#_#_#_____#_______#_#____#____#__#_#
_______#_#_###@#___##_______#_##_G___________##____
___###______@G@##__#__#_________#___#____#__##____#
___#__#___#_@#####__##______#__##___#_#________#___
#_#____#____@@__##__##_____##_#__##_________#_#_#__
___#_#___#_##@@@__#__#___#__##______#_#__#_##___#__
#__#______#_#_#@____________#__#___###___#_#___##__
#__________#_##@_#_#____##_____##_##_#___##_____#_#
______##____#_#@@#___##_______#_##__##___#___##____
##_#__#_____####@_____#__#___##________G_________#_
__#_____#_#____#@@__##__#_#_###____#________##__#__
_##__#____#_____#@_____#_##__#_##_##__#_#_##_#G____
____######__#____@@_________#____#_#_##__#_##___#__
_____G____###_#_##@##___#___#___#____________#_##_#
#_#____#__##__#___@___________##_##___#____________
___#_#__#______#_#@##_#_________###_#______#__##___
________#_#______#@G_____##_#______#__#____#####_##
________#_______#_@####___________##__##__#_#_#####
_____##__##_______@_____##____#__#_____#_##_#_#___#
______#_____#____#@____#_G___##_____#_______#__#__#
#_________##_#____@_###_##__#_#____#_###___#__#____
_#________________@@@@@____#______#_____##____G___#
#_#_______#_#_#____#_#@##_##____#__#________#_##___
_____#_#__#______#___#@@____##_##_#__#____#_#_##_#_
___#__##_______#__#___#@@_#__#_____#____#__#__####_
__G______#___##______#_#@#_####_____#__#_#____#____
______##___###__________@_###_____#__###_______#_#_
_###__#_##G__#_____#_#__@__G________#_#_##___#__###
__________###_##_____#__@_#__##____#G#__#_#___#____
###_____________G_______@#___#_###___##______#__##_
#__#___#___###_#_##_#_#_@GG@@_____###____#_________
___#________#___#____###___#@@@@@@@@@@@__#_________
__#____#__#_####___###_______#___##__#@@@@@@@@@@___
____#__##_____G#_____#__#_#_____##_#_#___##__##@#_#
_________#_#_#__#__##__#___###_###___####__##__@___
_#___##___#___#___###_______##_______#__#__#_#_@@@@
####__________##_____#_#_##__G_#_#__#___________##@
__#_____#_#____#__##_____#_______#_#_##_##_____##_@
___##_#___#_###__#________#___#_#______###_#______@
_____#_________#___#___#___#_#___##_#___#_##______@
_____#__#___###_________#_##___#____##_____#_#___#@
#____#_####__##_#__#_#______#_#____#______#_____#_@
__#__##_#__________##__##_#____##_#_#_#_____#__##_F
```

```
Agent 3 | Steps Taken: 523
Maze:
@@##@#_____#_____##__#_#___##__#__#___#__#_####_#_#
#@@@@#____#_____________##___###_#__#__#_____#_____
@@@@@__#_###__#__##_GG____________##___#_#_________
##@@_###______#__#___#__#_##__###_##_#G____###____G
__#@@______#__#________#_###____________#__#_______
_##@@______###____#___#_#______#__#__#_##_____G____
_#_#@@@#_#__#__#_______#____#___#____#__#__#_##____
_@@@@@_####____#_#__#__#____________________#______
@@######_##__#_#_#_#_____#_______#_#____#__G_#__#_#
@@_____#_#_###_#___##_______#_##_____________##____
@@@###_________##__#__#_________#___#____#__##____#
@@@#__#___#__#####__##______#__##___#_#________#___
#@#____#________##__##_____##_#__##_________#_#_#__
@@@#_#___#_##_____#__#___#__##______#_#__#_##___#__
#@@#@_@@@_#_#_#_____________#__#___###___#_#___##__
#@@@@@@@@@@#_##__#_#____##_____##_##_#___##_____G_#
_@@@@@##@@@@#_#__#___##_______#_##__##___#___##____
##@#_@#@@@@@####______#__#___##__________________#_
__#__@@@#@#@@@_#____##__#_#_###____#________##__#__
_##__#@@@@#_@@__#______#_##__#_##_##__#_#_##_#_____
____######__#@@@____________#____#_#_##__#_##___#__
__________###_#@##@#G___#___#___#____________#_##_#
#_#____#__##__#@@@@G@@@@______##_##___#____________
___#_#__#______#_#_##@#@________###_#______#__##___
________#_#______#_#_@@@@##_#_@@___#__#____#####_##
________#_______#__####@@@@__@@@__##__##__#@#_#####
_____##__##G____________##@@@@#@_#@@@@@#@##@#_#___#
______#_____#____#_____#__@@@##@@@@@#@@@@@@@#__#__#
#_________##_#______###_##_@#_#__@_#_###@@@#__#_G__
_#_________________________#______#_____##@@__G___#
#_#_______#_#_#____#_#_##_##____#__#___@@@@@#_##___
_____#_#__#______#___#______##_##_#__#@@@@#@#_##_#_
___#__##_______#__#___#___#__#_____#____#@@#@_####_
_________#___##______#_#_#_####_____#__#_#@@@@#____
______##___###____________###_____#__###_____@@#_#_
_###__#_##___#_G___#_#______________#_#_##___#@@###
__________G##_##_____#____#__##____#_#__#_#___#@@__
###______________________#___#_###___##______#_@##_
#_G#_G_#___###_#_##_#_#___________###____#_____@___
___#________#___#____###___#_____________#_____@___
__#____#__#_####___###_______#___##__#_G__G____@___
____#__##______#_____#__#_#_____##_#_#___##_G##@#_#
_________#_#_#__#__##__#___###_###___####__##__@___
_#___##___#___#___###_______##_______#__#__#_#_@@@@
####__________##_____#_#_##G___#_#__#___________##@
__#_____#_#____#__##_____#_______#_#_##_##_____##_@
___##_#___#_###__#________#___#_#______###_#______@
_____#_________#___#___#___#_#___##_#___#_##______@
_____#__#___###_________#_##___#____##_____#_#___#@
#____#_####__##_#__#_#______#_#____#______#_____#_@
__#__##_#__________##__##_#____##_#_#_#_____#__##_F
```

Two identical mazes with 20 ghosts, left using Agent 2 solves in 105 steps, right using Agent 3 solves after 523 steps (path: @, walls: #, free space: _, finish: F, death: X)

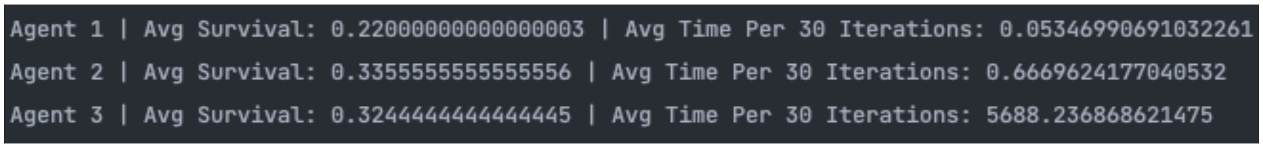


```
Agent 2 | Steps Taken: 111
Maze:
```

```
Agent 3 | Steps Taken: 163
Maze:
```

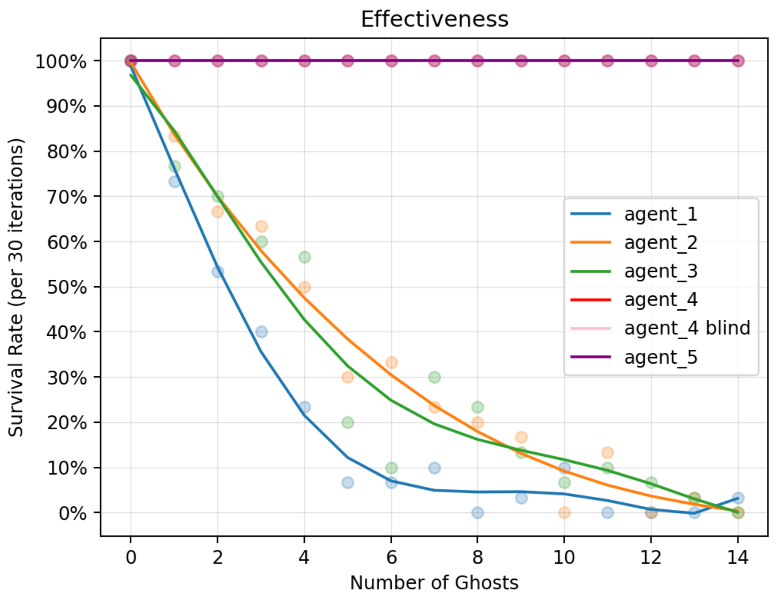
Two identical mazes with 40 ghosts, left using Agent 2 solves in 111 steps, right using Agent 3 dies after 163 steps with significant wiggling (path: @, walls: #, free space: _, finish: F, death: X)

Wiggling and ultimately survivability can be improved upon substantially by increasing the number of simulations in each direction by a very large number, with the consequence of increased planning time. In order to perform consistently like Agent 2, the number of simulations must be very high. I was able to test and confirm this theory by using a smaller scaled maze (10x10) with 1000 simulations in each direction, which resulted in Agent 3 being within 1% performance of Agent 2. The amount of time these simulations took makes this approach impractical in very large mazes. While some balance of speed and accuracy must exist, it appears any Agent 3 that executes in a practical period of time will still be much slower than Agent 2 and perform worse.



```
Agent 1 | Avg Survival: 0.22000000000000003 | Avg Time Per 30 Iterations: 0.05346990691032261
Agent 2 | Avg Survival: 0.3355555555555556 | Avg Time Per 30 Iterations: 0.6669624177040532
Agent 3 | Avg Survival: 0.3244444444444445 | Avg Time Per 30 Iterations: 5688.236868621475
```

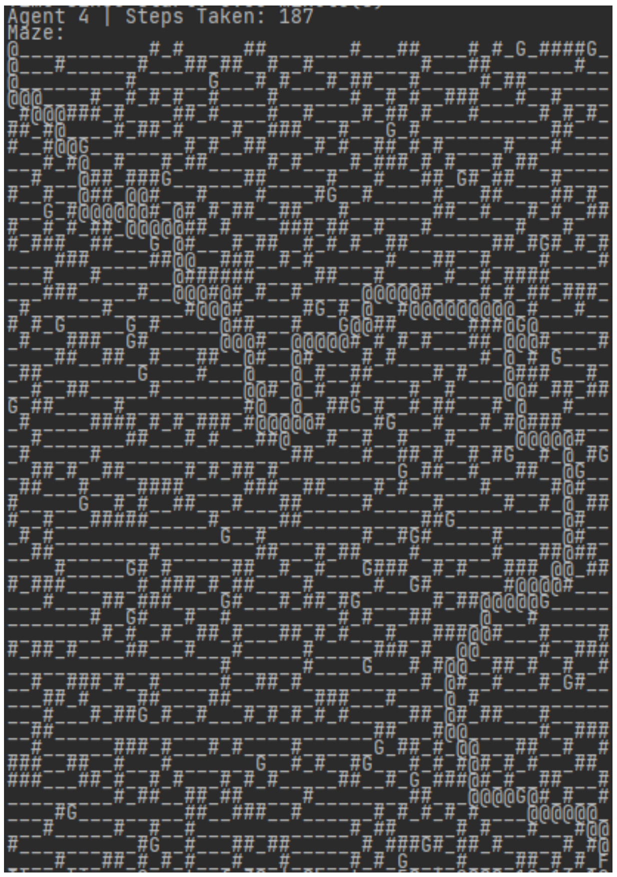
Avg Survival Rates of 30 iterations with 1000 simulations on a 10x10 grid (time per 30 iterations in seconds)



Effectiveness graph of 30 iterations with 1000 simulations on a 10x10 grid (Agents 4 and 5 show 100% since they were not being tested in this scenario)

Agent 4 Success:

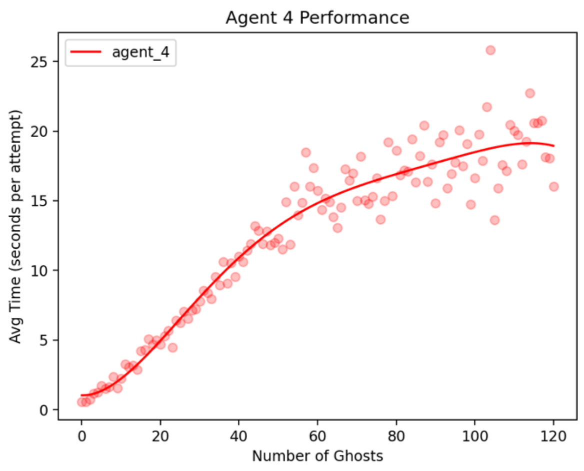
Agent 4 was much more successful than Agents 1, 2 and 3. It performed consistently better than Agent 2, by around 18 percent on average and only begins to approach similar levels of effectiveness in cases with very higher numbers of ghosts. Even then it remains above 10% survivability at 120 ghosts, only dipping below 10% beyond 120 ghosts It succeeded for two reasons, the first of which was that it often took paths that entirely avoided ghosts or at least chose to take paths with the fewest number of ghosts. Furthermore, while the retreat mechanic was kept, it was limited, preventing the Agent from retreating from unnecessarily far ghosts and spending more time in the maze. While avoiding ghosts is important, the longer an Agent is the maze the higher the chance of death. The goal of Agent 4 was to find a balance between progressing towards the goal while also avoiding ghosts. This can be seen below where Agent 4 takes a path in less crowded areas that tend to be slightly longer.



```
Agent 4 | Steps Taken: 187
Maze:
```

Path traveled by Agent 4, which takes 187 steps to reach the goal, this path is similar to Agent 5 (ghosts shown in positions when maze is completed resulting in the appearance of ghost in the path)

While on average Agent 4 performs about 26 times slower than Agent 2, it only begins to dip below 10% survivability at around 120 ghosts. The slower run time is largely due to the fact that Agent 4 does not take the shortest path and therefore often walks further. It is also slowed down because it must consider ghosts that are nearby when deciding danger, which is computationally expensive. The increased run time with higher numbers of ghosts comes as a result of several factors, moving ghosts becomes more expensive as the number of ghosts increase. Furthermore, more ghosts mean a path to reach the goal may become even longer as the Agent attempts to avoid more ghosts.



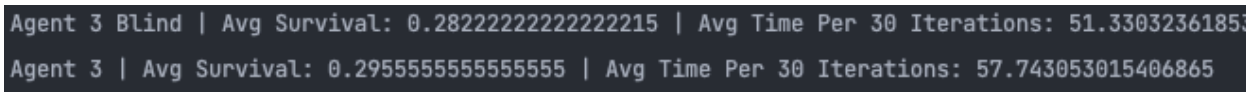
Performance graphs for Agent 4, avg time in seconds for a singular run to complete for a given number of ghosts

Agents without seeing in walls/ Agent 5:

(ghosts that cannot see in walls are sometimes referred to as blind)

-Agents 1 and 2 behave in a way in which they behave identically if they cannot see the ghosts inside of walls. This is because Agent 1 does not take ghosts during its only planning. Furthermore Agent 2 only takes ghosts into account if they block the path (meaning they are visible). Both Agent 2 and 3 also retreat only from the nearest original ghost. It can be argued that Agent 3 does see ghosts in walls since my Agent 3 copies the current state of ghosts for simulations and some of these ghosts are in walls. If the only ghosts that were copied were the ones that were visible, the survival rate remains similar for similar reasons as to why Agent 3 performs poorly. Additionally due to the implementation of how ghosts are stored, removing ghosts before executing simulations increases the complexity of Agent 3, this is negated by the fact that the simulations now have fewer ghosts and therefore occur faster. The similar behavior in addition to the extremely long runtime means I have excluded the full results of 100 iterations from 0 to

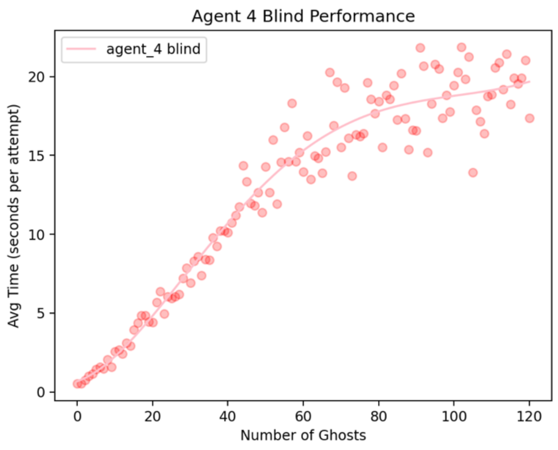
119 ghosts in the official data collected, however Agent 3 features a flag allowing this behavior to exist for testing, and the result of a small-scale test can be seen below.



```
Agent 3 Blind | Avg Survival: 0.28222222222222215 | Avg Time Per 30 Iterations: 51.3303236185
Agent 3 | Avg Survival: 0.2955555555555555 | Avg Time Per 30 Iterations: 57.743053015406865
```

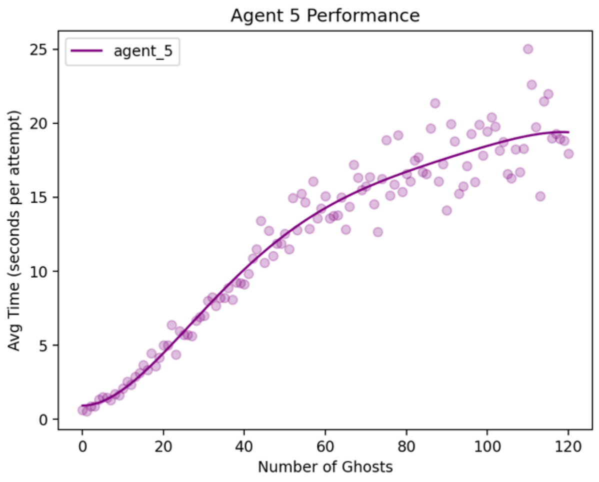
Agent 3 vs Agent 3 Blinded in a 10 x 10 maze with 3 simulations each (time per iterations in seconds)

-Agent 4 without seeing through walls appears to be similarly as successful, only performing about 1% worse in testing on average. I believed is because regular Agent 4 considers ghosts in walls to have less of a value (80%) than those that are visible. When the 80% value was changed to other values between 25% and 100% for regular Agent 4, the success rate fluctuated less than 1%. Additionally, the average run time and time per attempt for a given ghost number remains very similar to the original (see below).



Performance graphs for Agent 4 Blind, avg time in seconds for a singular run to complete for a given number of ghosts

-Agent 5, in attempt to reduce the 1% decrease in blind Agent 4, takes the “danger” weight of a given cell from the last move and averages it with the new “danger” weight, which attempts to consider cells where ghosts may have been previously visible and have moved into walls. The downside is this also means a ghost that was in a wall and has come out may cause a lower “danger” weight than reality. In testing Agent 5 on average does only slightly worse than Agent 4 (about 0.2%) and sometimes even beats it out. Since Agent 4 has more info, this suggests that a better way to weight ghosts in walls for Agent 4 could make it more accurate. Like the blinded Agent 4, the run time of Agent 5 remains fairly like the original Agent 4 and follows a similar pattern of slowing down with more ghosts (see below).



Performance graphs for Agent 5, avg time in seconds for a singular run to complete for a given number of ghosts