Report for Coursework 2 of CS255

Student ID: u1719840

1. INTRODUCTION

The basic approach to the second gametype is to calculate which first item to take then recursively fetch all remaining items. For the third gametype, the robot needs to find the goal, fetch all items and explore if it needs too while avoiding enemies and figuring out when a move is safe or not.

2. STRATEGY DESIGN

2.1 Gametype 2

This section will explain the design choices for gametype 2.

2.1.1 Scanning The Environment

In order to figure out whether the current maze or env is doable, the robot executes dijkstra's algorithm from the goal and find all accessible items on the map. This means that the robot will not waste time trying to fetch items which are impossible to access. This has another purpose, to calculate the minimum path from the goal to every nodes and therefor items on the map which is more efficient than using A* for every path.

2.1.2 First Item Choice

The most substantial choice in this game type is the choice of the first object as after this item is dropped the order in which you take the other objects does not matter (the number of steps for the path: goal to node to goal, is constant). This part of the software changed all throughout it's implementation as better implementations came to mind. The first implemented method was to find a node which was as close as possible to a direct path: start to goal. This was a terrible method as it was not very efficient and it is fairly easy to see that you don't want the shortest path: start to object to goal. A better method is to find a node which is far from the goal and isn t too far from the start. Since if the object isn't the first collected, the robot will have to travel twice the distance from the object to the goal. Meanwhile if it is the first object collected then you have to travel from start to object to goal. Therefore, comparing these two values for each node and getting one of the extremes (depending on how you compare them) In the final version of the software, it uses a second dijkstra from the start and simply calculates the difference between start to object and goal to object. This is because to calculate the number of steps for the whole process if you chose that object as first object you remove goal to first_object from the sum of all goal to object to goal (for every object calculate that sum and add it to a total value) and add start to first_object.

2.1.3 Dijkstra

The robot only use the algorithm twice and then use the stored paths, this is because it is faster than using A* for each items. It is useful as well since it finds all the items which can be collected and if the goal is attainable.

2.2 Gametype 3

For this gametype there were two major design challenges the first was the online search and the second was avoiding enemies.

2.2.1 Online Search

In this gametype the robot stores the scope of vision it has to remember the map, locate items and detect threats. This is designed like so since it uses A* to find paths around the map. The major new feature in online search is to explore the maze when there is no available item or if the path to the goal is unknown. It doesn't use dijkstra since it finds new paths constantly and so would need to recalculate dijkstra each turn you discover new tiles. The way the robot explores is by seeing the env as a number of 7*7 tiles and checking whether they are explored, unexplored totaly or filled with walls. It is 7*7 because any smaller will be inefficient and any bigger and the robot might be stuck in an endless loop by not being able to explore fully a tile. It then chooses naively which tile to explore next by starting at the top left and going down until you reach the goal row and trying the next column. This is a part of the design which could be improved.

2.2.2 Avoiding Enemies

The robot chooses to ignore the fact that it can get some enemies' path since just adding a tries counter would effectively be the same, or better. It has also a simpler implementation. The robot only tries to get to an object through an enemy 10 times before trying to recalculate a new path since this makes it much more efficient with known/unknown path enemies. When the robot has no safe choice, it chooses to follow its normal path ignoring any enemies and thus forcing its way out of an ambush. Objects and the goal have a special property which makes enemies unable to go on that

Coursework submitted for CS255.
Report compiled January 15, 2019

tile. The robot makes use of this, and wait if there is any enemy which could collide with it after grabing the object.

3. EVALUATION METHODOLOGY

3.1 Gametype 2

There was no real unit testing during this coursework as it mostly was the often one method which did most of the work and so integration testing was the majority of the testing for this part of the coursework. The results are constant so repeating the test is useless because there is no part of random in this gametype. To evaluate where a particular solution stands, how efficient and correct it is, was mostly done by comparing results with other solutions. This was either done with the goal to attain defined next to the goal in runMaze.py or by comparing results with peers. Then either this solution gives the best possible answer by choosing the right first object to collect or it doesn't and then you have to rethink the solution.

3.2 Gametype 3

To test solutions of this gametype, it was better to put them in specific environments which forced to test some part of the solution. For example to test the logic to fetch items and use the stored map to navigate the map, smaller mazes make it so that you do not need to explore and easily test the way you store the map. To check if its valid and updates correctly. Bigger mazes test the switch between exploring and fetching items (or dropping them). Finally using different environments with different enemies test how the robot behaves and avoids the enemies. A major debugging tool was the print_state() method. This printed the state of the map, and what was discovered by the robot. It helps to see in which cases the robot gets stuck or hits an enemy. Creating a custom maps, or modifying existing maps enable us to check the robots' behaviour when confronted with specific situation.

4. RESULTS AND DISCUSSION

4.1 Gametype 2

For the second gametypes the results are constant, so only running the code once is sufficient to show it working. The first two maps can be solved optimally with a normal manhattan heuristic. But this isn't sufficient for the third map.

For the first environment, the "10Squares.txt" this is the number of moves it takes :

```
Robot has successfully navigated the terrain!
Moves made: 64
Collisions: 0
Retrieved 5 out of 5

real    0m0.084s
user    0m0.048s
sys     0m0.020s
```

This is the minimum number of moves and it does so almost immediately.

For the second environment, the "25Squares.txt" this is the number of moves it takes :

```
Robot has successfully navigated the terrain!
Moves made: 463
Collisions: 0
Retrieved 10 out of 10

real    0m0.095s
user    0m0.061s
sys     0m0.019s
```

This again has the most optimal solution and is very fast.

For the third environment, the "75Squares.txt" this is the number of moves it takes :

```
Robot has successfully navigated the terrain!
Moves made: 9908
Collisions: 0
Retrieved 40 out of 40

real    0m0.437s
user    0m0.338s
sys     0m0.057s
```

For this map, the robot could not find the most optimal path because the heuristic wasn't taking into account the fact that some paths weren't direct. This meant that the heuristic must be improved as it wasn't complete and using only two dijkstra calls is complete and fast enough on even big maps.

4.2 Gametype 3

For this gametype, only a couple examples are necessary as it gives the same broad results on any maze. This solution works better on smaller and more open maps. Solving the maps all the time with no enemy collisions and perfect paths is made impossible because of the randomness of some enemies. There are some mazes which are impossible due to this, for example if an aggressive enemy gets stuck in a one-way tunnel to the goal, then accessing it is impossible. In any maze it is possible to get cornered by multiple enemies or only one in a dead end, and avoiding it would require a much more complicated algorithm. This means that there will be runs with collisions and items which are taken.

The first example is with the "10Squares-ARGS.txt" map and has these results on average:

```
Played  10  games.
ENEMY COLLISIONS
Most collisions:  0  (Game  1 )
Least collisions:  0  (Game  1 )
Total collisions:  0
OBJECTS
Most objects:  5  (Game  1 )
Least objects:  5  (Game  1 )
Total objects:  50
MOVES
Most moves:  64  (Game  1 )
Least moves:  64  (Game  1 )
Total moves:  640
```

These smaller maps were easier to start with since there was

no need to explore the map since it was directly explored fully. They are also generally completed perfectly on all runs.

The second example is with the "50Squares-PDFU.txt" map and has these results on average:

```
Played  10  games.
ENEMY COLLISIONS
Most collisions:  1  (Game  1 )
Least collisions:  1  (Game  1 )
Total collisions:  10
OBJECTS
Most objects:  19  (Game  1 )
Least objects:  19  (Game  1 )
Total objects:  190
MOVES
Most moves:  3749  (Game  1 )
Least moves:  3749  (Game  1 )
Total moves:  37490
```

The known and unknown types of enemies forced an implementation of "tries" in the robot avoidance since it was most often then not very beneficial to just wait. Especially since if the robot got stuck in one place, it would get stuck again every time it tried to come back through that path.

The third example is with the "50Squares40DensityMultiMEn0.txt" map and has these results on average:

```
Played  10  games.
ENEMY COLLISIONS
Most collisions:  5  (Game  4 )
Least collisions:  0  (Game  3 )
Total collisions:  14
OBJECTS
Most objects:  19  (Game  3 )
Least objects:  17  (Game  4 )
Total objects:  184
MOVES
Most moves:  4147  (Game  6 )
Least moves:  2106  (Game  3 )
Total moves:  24598
```

This last example, shows that the robot deals well with environments with multiple different enemies. Since the logic behind it does not differentiate between robot types, there was no need to ass a special case for when enemies act differently.

5. CONCLUSIONS

Give a brief conclusion, including discussion of what worked well, and what did not work, and mention any lessons learned.

To conclude, Gametype 2 works perfectly all the time. Meanwhile Gametype 3 cannot work perfectly all the time, it needs mazes with the start at the top and the goal at the bottom or it will fail. It will avoid all contact with enemies but will try to force through an enemy if it is stuck and there is no safe move. The use of print_state() showed me it was important to spend time to help testing throughout the process of implementing the robots' logic. This made debugging the robot much easier and to find a few specific situations where the robot would fail. Trying to optimise the algorithm too far makes it impossible to implement, and without the use of good commenting it is easy to get lost in the code and forget how that logic worked.