Choosen area: Robot Motion Planning

Project Title: Mysterious Island.

Done by: Desmond Nini

The maze is made up of a 14 by 14 grid of open cells, each of a rectangular shape. Some cells are made of walls thereby creating a complex maze. The agent is completely an autonomous robots that must find its way from a predetermined starting position to the central area of the maze unaided. The agent will need to keep track of where it is, discover walls as it explores, map out the maze and detect when it has reached the goal. Having reached the goal (first trial), the agent will typically perform additional searches of the maze until it has found an optimal route from the start to the center. Once the optimal route has been found, the mouse will run that route in the shortest possible time.

Overview:

Robots are particularly useful for performing hazardous duty services. They can easily perform hazardous work without risking human life or limb. For instance; they are used in many bomb squads around the country. Resembling small armored tanks, these robots are usually guided remotely by personnel using video cameras attached to the front of the robot (cool!). Similar robots can also help with handling or cleaning up toxic waste. They can work unimpeded in all types of polluted chemical environments, including ones dangerous in which unprotected human being would quickly die. In fact the nuclear industry was the first to develop and use robotic arms for handling radioactive materials. Currently, there is research being done in developing fire-fighting robots that can detect a fire anywhere in a building, travel to the location, and put out the fire.

In addition, robots are becoming increasingly indispensable in war and weaponry. Defense Services around the world are developing sophisticated robotics programs to keep watch on enemies and help ensure their victory if the need for war should arise. In modern warfare, drone aircraft tracks enemy movements and keeps the enemy under close surveillance.

Other examples of intelligent weaponry are **smart bombs** and **cruise missiles.**

There are also many different applications for domestic robots. While labor-saving devices such as washing machines, clothes dryers, dishwashers, ovens, and the like have been computerized for some time now, most people would not characterize them as robots but rather as machines since they do not autonomously gather the materials they need to perform their functions. Although it may be a while yet before we all have ”Rosie the Robot” maids to perform various household duties, current research in the development of domestic robots is in the area of greater automation.

Domain Background

MicroMouse is a robotics competition where small, self-contained robots, attempt to map, then solve and navigate a previously unseen maze.

The robots, or ‘agent’, have to find their way from a certain pre-chosen corner cell to the center cells of the maze, in the fastest possible time.

To achieve this, they are allowed to make as many 'runs' from the start cell to the finish cell as they can fit within a 10-minute timeslot.

This problem can be solved using the Re-enforcement learning.

From Q-learning algorithm, we can use the bellman’s equation which defines the relationship between the states and actions of a the robot

It writes the value of a decision problem at a certain point in time in terms of the payoff from some initial choices and the value of the remaining decision problems that result from those initial choices. This breaks a dynamic optimization problem into simpler sub problems, as Bellman's "Principle of Optimality" prescribes.

Robot motion planning has become a major focus of robotics.

When the Micro Mouse competition was first run, in 1979 in New York, The goal

Was to get to the opposite corner of the maze from the one where you started.

A Wall following mouse easily won this competition, and so in subsequent years the rules were changed to move the target to the center cells. This meant that an 'Island' could be created in the center of the maze, which would stop wall followers from solving the maze.

In 1980 the first MicroMouse competition in Europe was held. There were 18 competitors in total, none of whom a managed to solve the maze.

Various competitions then started springing up around the world, including Japanese, Singaporean, American and European competitions.

The UK competitions are held each year at the MINOS MicroMouse conference at Easter, the UK MicroMouse championships (UKMM) at TIC in Birmingham during June, and RoboTIC, Also at TIC in late November.

Considering my location, when it is the next project session to hold the RoboTIC competition

I will be there as these competitions are a valuable resource in finding MicroMouse information, speaking to other mouse builders, and getting technical information and advice to improve my robotic agent.

Problem Statement

Navigating through a **14 by 14** grid complex maze, the agent has to find its way from a certain pre-chosen corner cell to the center cells of the maze, in the fastest possible time and the most optimized path.

Each maze used in the project follows a strict specification. The maze is a fully enclosed square
At the center of the maze is a colored square (with picture trophy) enclosed by some walls and one or two entrances.
This area is the goal of the maze, and the robot must enter this space to complete a successful run.
The metrics for the test maze explains how robot can move freely through the maze in an attempt to explore and map it. It is free to continue exploring the maze even after entering the goal area. Once the virtual robot has found the goal, it may choose to end. For the second and sub sequential runs, the virtual robot is expected to traverse the maze and reach the goal as quickly as it can with Q-learning algorithm

It is expected that Q-learning technique will solve the given maze; albeit by varying degree. At the end of the project, we intend to find the best path(fastest) this algorithm can use to reach its goal. For this, we need to define a single Score Metric that will help us pick the winning path

Robot Score/ nominalized: The robot's score for the maze is set to be a negative value, which accumulates each time the agent changes state through an unconditional action. Hence, the further the agent explores, the higher the negative rewarded scores. The lower the states explored, the fewer negative points the agent earns which will all sum up with the goal reward to give a remark scores for the episode path

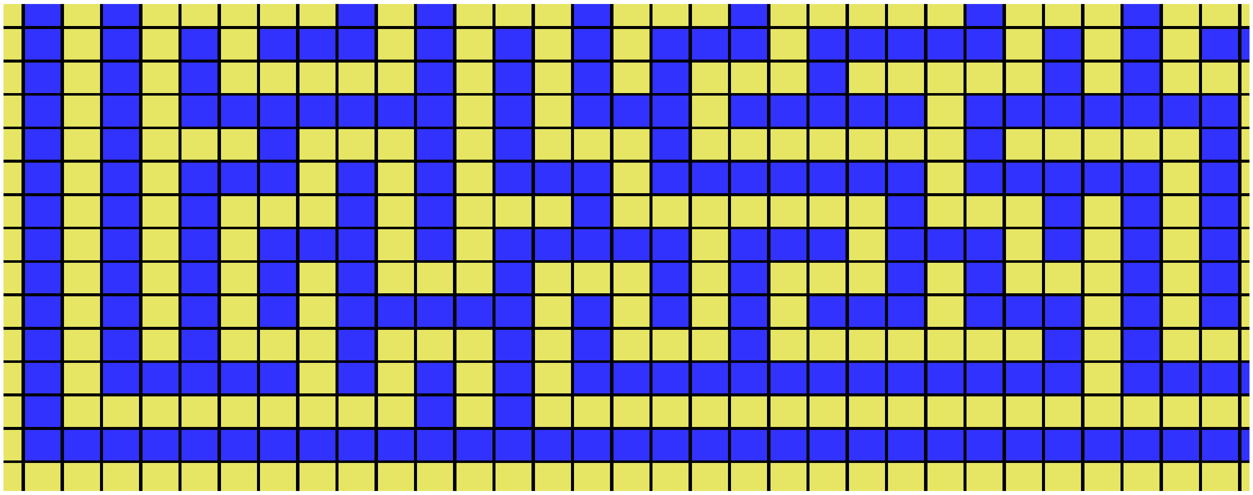
Datasets and Inputs

The ***canvas*** function from ***Tkinter*** Library in python plays a very important role in graphics.
By calling the functions, it can create various drawing elements. For example: create_rectangle(value1,value2, valueX). Some implementations also define the spatial representation and allow the user to interact with the elements.
A 14 by 14 maze can be created with walls positioned within different cells of the maze to block movement on like the borders. The x and y axis will define any of the cells to initialize the movement of the agent, probably bottom left.

It is also interesting to note that the fewer the walls in a part of the maze allows the Robot to explore that part of the maze more efficiently than it would otherwise.
If more corners were present, they would prevent the robot from seeing down long `corridors'. This shows just how much of the maze the Robot can explore with just a few steps.

On each step of the simulation the robot may choose to rotate clockwise or counterclockwise ninety degrees, and then move forwards or backwards.
After movement, one time episode has passed, and the sensors return readings for the open squares in the robots new location and/or orientation to start the next time unit.

Below is a prototype of the potential N x N maze grid



Data:

● Rules: The agent (beginning state) has to reach the goals to end the game (center)

● Rewards: The center gives a positive reward. Each step gives a negative reward

● States: Each cell is a state in which the agent can be.

● Actions: There will be only 4 actions. Up, Down, Right, Left.

Solution Statement

We have different variety of ways to solve the mysterious journey like the random mouse algorithm like:

Random run algorithm: Which is noted for revisting cells thus using more time for trainning
Follow wall algorithm: which hardly visits the inner cells
Depth first search algorithm: which select some arbitrary node as the root in the case of a graph and explores as far as possible along each branch

Q-learning: is a model-free reinforcement learning technique. Specifically, *Q*-learning can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP)
It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. A policy is a rule that the agent follows in selecting actions, given the state it is in. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state

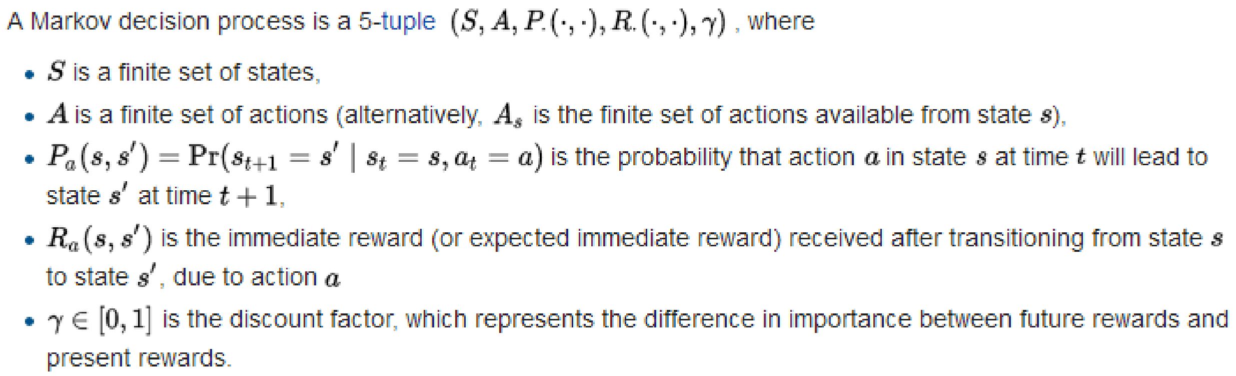
The purpose of our Q- training is to enhance the brain of our agent So as to be represented by q matrix. More training will give better Q- matrix that can be used by our agent to move in optimal ways.

The logic:

1. Initialize Q(s,a) arbitrarily
2. Repeat (for each episode)
   - Choose a from s using policy derived from Q (e.g. e-greedy i.e. always random)
   - Take action a, observe r,s’
   - update Q(s,a) using one-step Q above
   - s <– s':
3. Until s is terminal.

(Where **s** is the state of the cell, **a** is the action to be triggered, **r** is the reward obtained from an action, and **s’** is the new or next state)

By using **Markov decision processes (MDPs**, we can provide a mathematical framework for modeling



A Markov decision process is a 5-tuple $(S, A, P_\cdot(\cdot,\cdot), R_\cdot(\cdot,\cdot), \gamma)$, where

- S is a finite set of states,
- A is a finite set of actions (alternatively, A_s is the finite set of actions available from state s),
- $P_a(s,s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$,
- $R_a(s,s')$ is the immediate reward (or expected immediate reward) received after transitioning from state s to state s', due to action a
- $\gamma \in [0,1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

$$Q(s,a) = R(s,a) + y * \max[Q(s', all\ actions)]$$

Now from our initial Q matrix, we can update our values via the formulae above.
Parameter gamma(y) has range value of 0 -1 (0<= y <1). If y is closer to zero, the agent will tend to consider only immediate reward, makes the learner short-sighted giving more importance to short term rewards. But if y is closer to one, the agent will consider future rewards with greater weight

Benchmark Model

The maze the have to execute two runs, the first run called exploration run in which this gives its the opportunity to learn the environment, explore every path randomly to build a map of the maze from the initial possession. There is no reason why the optimal path through the maze cannot be found as long as the virtual robot completes its exploration of the maze. Any other distance for the final path portion of the benchmark would be entirely arbitrary so the optimal path will be used for the benchmark.

The exploration portion of the benchmark is much harder to define. This ambiguity suggests that two benchmark scores would be appropriate. An Upper Benchmark score will define a threshold that any robot using even a naive exploration technique should beat while a Lower Benchmark will define a much harder to achieve threshold that should require a robot to use a much more sophisticated and tailored exploration logic. Another way to compare the two scores is that the Upper Benchmark is the largest acceptable score a working robot needs to beat while the Lower Benchmark is the score that a robot should try and beat to prove the quality of the implementation.

To visit every cell of a maze, it would be impossible for a robot to traverse the entire maze and visit every cell without stopping at a cell more than once. Naive exploration logic could be implemented in such a way as to instruct the robot to navigate the maze and stop at every unvisited cell. Once all cells have been visited, then the robot will have a complete map of the maze and will be able to find the optimal path to the goal. This suggests that the Upper Benchmark needs to allow for more exploration steps than there are cells in the maze, but even naive exploration logic should allow the robot to traverse the entire maze without using as many steps as twice the number of cells and if it does, then that is a symptom that something is wrong with the exploration logic

Evaluation Metrics

To adequately evaluate the performance of the robot navigation and mapping logic, we need to raise the number of trials so as to be certain.

Considering we program quite some sequence of episodes to cover, should eventually confirms our adequacy of evaluation.

The robot will performed so well when compared to the maze benchmark outlined in the Benchmark section of this report, that a stricter benchmark was defined to better analyze the results.

The race benchmark is the same for both the Upper (less strict) Benchmark and Lower (more strict)

Benchmark, but the exploration scores differ by a single factor. The Upper Benchmark includes an ex-plor1tion score that is based on the robot stopping in each cell twice while the Lower Benchmark allows the robot to, on average, only stop at each cell once.

NB: A larger positive value may indicate that the robot outperformed the Benchmark score by that amount. This means that a larger positive number is better than a smaller number. Negative numbers indicate that the robot did not beat the benchmark.

It is clear that for a very large majority of mazes, the robot beat both of the Upper and Lower

Benchmark scores. As the maze dimension increases, the robot outperformed the benchmarks by a larger amount

Project Design

The project workflow by theory are follows the entities below

1.Start position: the agent needs an initial position ()

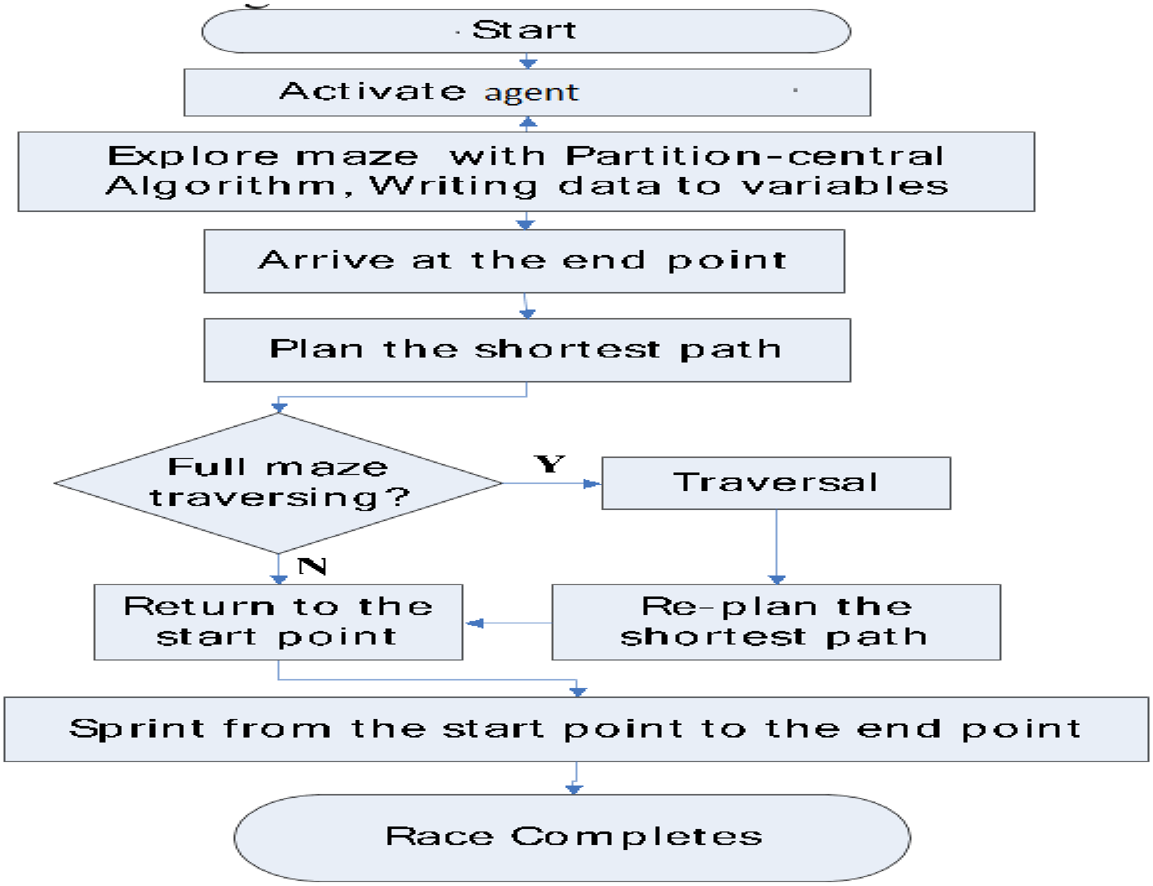
2. The agent needs to receive a small amount of information about the maze (sensor readings).
3. Combine this information with the heading and location of the robot.
4. Update the maze map with any new information inferred directly or indirectly from the sensor readings.
5. Given the current knowledge of the maze, decide where to travel to maximize the acquisition of new information.
6. Decide how to get from the current location to the desired location.
7. Move toward the desired location.
8. Repeat steps 1-7 until the optimal path from the start to goal locations has been found.
9. Start the next run of the maze.
10. Follow the optimal path from the start to the goal.

As straightforward as this sounds, the actual implementation of the robot that can consistently and successfully complete this process in an efficient manner even when faced with a large number of maze. permutations is considerably more difficult. An entirely
robust solution would likely increase the complexity of the exploration logic and might require sacrifcing the efficiency by which the robot explores the majority of mazes it faces and thus causing the robot to perform worse on the majority of mazes simply to perform better for a small number of others. Whether this would be an acceptable compromise is largely academic and is outside of the scope of this project (given the current project parameters and scoring rules).
Despite this difficulty, the algorithms used to map and navigate the maze are quite straight forward. It
is interesting to consider how a small number of reasonably simple algorithms, when combined in a coherent way, can result in quite complex behaviour and manage to solve the problem at hand so well.
.
On the note of algorithm, Q-learning is a model-free algorithm, in which the agent learns the Q-factors (i.e., the value function extended to state-action pairs, rather than only states) from its experience with the environment sound sweet able enough to control the maze navigation.
Q in the name of a reinforcement learning algorithm estimates the Q-factors, which are simply the state-action value function. Let S and A be the state and action spaces, respectively. By state-action value function, I mean that you take all the state-action (s,a) pairs, for all s in S and a in A, and it build a new Markov Decision Process with transitions between pairs in this SxA extended space does get to process going.

The strategy will covers the following entities

Something like:

```
create and initialise a queue
create a routing map as an array
fill the array with the value
fill the target cells(s) with the value 0
add the target cell locations to the queue
while there are items in the queue
  get a location from the tail of the queue
  if the cell does not contains the value  then
    for each accessible, unfilled neighbour
      put in it a value one more than the current cell
      add it to the queue
    end for
  end if
```


Start
Activate agent
Explore maze with Partition-central Algorithm, Writing data to variables
Arrive at the end point
Plan the shortest path
Full maze traversing?
Y
Traversal
N
Return to the start point
Re-plan the shortest path
Sprint from the start point to the end point
Race Completes