Machine Learning Engineer Nanodegree

Capstone Proposal
Robot Motion Planning Capstone Project.
Plot and Navigate a Virtual Maze

Domain Background
Robotics is very important nowadays and brings technologies which can deal with automation machines that can take the place of humans in dangerous environments or manufacturing processes, or resemble humans in appearance, behavior and cognition. One of the main challenges in robotics is finding the optimal path through in the environment they operate.

Problem Statement
A robot mouse has a task to plot a path from a corner of the maze to its center. The robot may make multiple runs in a given maze. The first run should be used not only to find the center, but to explore as much maze as possible and figure out the best paths to the center. In subsequent runs, the robot mouse attempts to reach the center in the fastest time possible, using what it has previously learned.
The problem lies in the machine learning area as it operates within the initially unknown world which needs to be explored. This prevents the classical search algorithms to perform well as it's required to do some initial exploration beforehands.
The role of Machine Learning in autonomous robots is described in the following papers: Reinforcement Learning for Autonomous Robot Navigation (see https://ieeexplore.ieee.org/document/833418/ for more details); Real-world reinforcement learning for autonomous humanoid robot docking (see https://www.sciencedirect.com/science/article/pii/S0921889012000814 for more details).

Datasets and Inputs
Here are the descriptions of the robot and an environment.

The maze exists on an n x n grid of squares, n is even. The minimum value of n is twelve, the maximum - sixteen. Along the outside perimeter of the grid, and on the edges connecting some of the internal squares, are the walls that block all movement. The robot will start in the square in the bottom-left corner of the grid, facing upwards. The starting wall will always have a wall on its right side and an opening on its top side. In the center of the grid is the goal room consisting of a 2 x 2 square. The robot must come here from its starting square as fast as possible.
Mazes are provided to the system via text file. On the first line of the text file is a number describing the number of squares on each dimension of the maze n. On the following n lines, there will be n comma-delimited numbers describing which edges of the square are open to movement. Each number represents a four-bit number that has a bit value of 0 if an edge is closed (walled) and 1 if an edge is open (no wall); the 1s register corresponds with the

upwards-facing side, the 2s register the right side, the 4s register the bottom side, and the 8s register the left side. For example, the number 10 means that a square is open on the left and right, with walls on top and bottom (0*1 + 1*2 + 0*4 + 1*8 = 10). Due to array indexing, the first data row in the text file corresponds with the leftmost column in the maze, its first element being the starting square (bottom-left) corner of the maze.
I will be using the 3 mazes provided with the files in the starter code.

The robot can be considered to rest in the center of the square it is currently located in, and points in one of the cardinal directions of the maze. The robot has three obstacle sensors, mounted on the front of the robot, its right side, and its left side. Obstacle sensors detect the number of open squares in the direction of the sensor; for example, in its starting position, the robot's left and right sensors will state that there are no open squares in those directions and at least one square towards its front. On each time step of the simulation, the robot may choose to rotate clockwise or counterclockwise ninety degrees, then move forwards or backwards a distance of up to three units. It is assumed that the robot's turning and movement is perfect. If the robot tries to move into a wall, the robot stays where it is. After movement, one time step has passed, and the sensors return readings for the open squares in the robot's new location and/or orientation to start the next time unit.

Solution Statement
Mice can use various search algorithms, such as: Dijkstra's algorithm, A* search, etc. Also I plan to give a try to Monte Carlo and Temporal Difference methods from Reinforcement Learning.

Benchmark Model
I'll use the depth-first search to compare the performance of the approaches I'm about to explore. It's known that depth-first search doesn't always provide the optimal path, so with having this in mind, I might be able to actually beat it by using techniques given in Solution Statement.

Evaluation Metrics
On each maze, the robot must complete two runs. In the first run, the robot is allowed to freely roam the maze to build a map of the maze. It must enter the goal room at some point during its exploration, but is free to continue exploring the maze after finding the goal. After entering the goal room, the robot may choose to end its exploration at any time. The robot is then moved back to the starting position and orientation for its second run. Its objective now is to go from the start position to the goal room in the fastest time possible. The robot's score for the maze is equal to the number of time steps required to execute the second run, plus one thirtieth the number of time steps required to execute the first run. A maximum of one thousand time steps are allotted to complete both runs for a single maze.

Project Design

There's a starter code already which needs to be completed. The environment is pre-defined in maze.py. In robot.py there's a starter code of the agent that needs to be implemented. The agent needs to return the next step based on the sensory information provided. To do so, I will explore 2 approaches:

- Try to build the Reinforcement Learning agent using Monte Carlo and Temporal Difference methods to explore the world and find the best possible path. The state space and action space will be discrete (according to the maze and robot specifications) which makes it easy to apply different Reinforcement Learning algorithms.
- Try to do the exploration and then use Dijkstra's algorithm and A* search algorithm to find the best path based on the explored data. The first step with data exploration is is planned to be tried with the following methods: random move, dead-end detection, use counter to visit the less visited locations, build some heuristics and try to explore this way.

These approaches will be compared with the performance of the benchmark model defined.