RBE-550 Motion Planning
Daniel Montrallo Flickinger, PhD

FLATLAND

DUE: 2025-09-15 @ 11:00 UTC

Abstract

Take a journey to the 2nd dimension with these awesome gridworld point robots. Set up obstacle courses of doom, and implement various search algorithms to guide your hero through the unrelenting gauntlet. Does it have what it takes to survive the onslaught?

1 Introduction

As an introduction to graph-based discrete motion planning techniques, we're guiding our brave point robot (Figure 1) through a version of Chase. (or Robot Minefield)

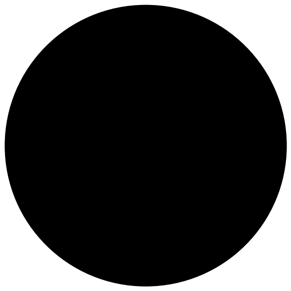


Figure 1: A point robot (actual size)

2 Development Environment

Implement an obstacle-filled grid world that models a graph for robots to navigate through. Create path planners to control all agents, and run game logic. This could be a custom-built system using C, Python or MATLAB, or utilize existing 3rd party libraries available in a variety of languages. A few examples follow:

2.1 Python

The most straightforward implementation is to write a 2D grid simulation from scratch using Python. Consider 2D grid libraries, such as minigrid, pyGame, and others.

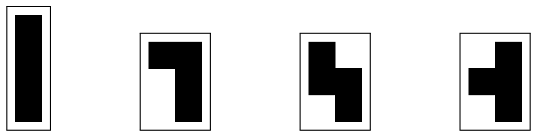


Figure 2: Example free tetromino obstacle shapes (non-standard shapes cost extra).

2.2 MATLAB

There exist options such as the Grid World model in the MATLAB Reinforcement Learning Toolbox.

2.3 ROS

Another option is to utilize ROS. Install the Grid Map library Fankhauser and Hutter [2016], following the installation instructions. Costmap 2D could also work, with some modifications.

2.4 Others...

Otherwise, implementations in any programming language may be submitted for this assignment.

3 the Obstacle Course of Doom

Create an obstacle course of doom, similar to the depiction in Figure 3. Fill the space with obstacles at a density of 20%, and instantiate a goal location at a random unoccupied cell.

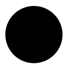
The environment must have the following attributes:

- 64 x 64 two dimensional grid
- multiple randomly placed obstacles, using tetrominoes (Figure 2)
- single hero robot, occupying a single cell at each time step
- 10 enemy robots, each occupying a single cell at each time step
- single immutable goal location
- wumpus not included (or is it?)

Consider Figure 3. The obstacle course of doom is represented as a 2D grid. Black cells are immutable obstacles. The red triangles (▲) are the enemy robots, ready to dish out pain, or fumble into rubble (either option a likely outcome). The green cell denotes the goal location. And finally, our hero robot, ●, makes an appearance.

In modeling the state of the graph, consider the following discrete states for each cell: *{ free, obstacle, junk, goal, hero, robot }* .

4 the Hero



Our hero robot (Figure 1) starts at a random unoccupied location in the world. It's guided by a path planner, using your choice of discrete algorithm. It also has omniscient perception of the world, knowing the location of all obstacles, boundaries, enemies, and the goal.

At the initial state, the robot plans a path to the goal, avoiding obstacles and enemies. It may execute this plan through as many iterations as needed, and has the option to replan at any timestep. It also may teleport to a random location if bored or threatened. Be aware that its teleporter fails after five activations. Use depth first search, breadth first search, Dijkstra's, or any other discrete algorithm, e.g. as discussed in Section 2.2.2 of LaValle [2006].

5 Robot Enemies

Enemy robots move to a neighboring cell at every time step, in the direction of the hero robot. They are not aware of obstacles, or each other. The only thing they are aware of, is the hero robot. The hero used to be one of them, but that power of omniscient perception now sets it apart. This drives the other robots into a literal blind rage. If one of these robots comes in contact with anything, it breaks down and its hulk becomes a permanent obstacle. If an enemy robot contacts the hero, it destroys the hero and the game is over.

6 Tips and Miscellaneous

To get started, implement the following components:

- graph implementation
- game state and instrumentation for results
- obstacle generator
- goal placement
- collision checker

- visualization
- planner logic for enemy robots (move to cell closest to hero)
- AAA level planner implementation for the hero robot

Other topologies could be an option, such as toroidal. You may also include fixed teleport locations, a.k.a portals.

Your planner implementation should gracefully handle failures. There is no guarantee that a valid path exists to the goal. Also be sure to include logic that checks the overall game state, and ends the simulation when the hero robot is destroyed.

Also consider heuristics, weighting, and other cost functions when implementing an algorithm such as Dijkstra's method. The planner could perform better if it could predict the motion of the enemies, or at least avoid regions near them.

7 Results

Complete the components discussed in the following sections to receive full credit on this assignment. Refer to Section 8 for submission guidelines, and Section 9 for a summary of the grading rubric. Consider submitting your work as you complete each component, as partial credit is preferred to no credit.

7.1 10% Full Source Code

Submit the full source code, either directly to Canvas, or as a link to a revision control system. Include links to any third party libraries utilized in your solution.

7.2 20% Report

Write a brief report detailing the design, implementation, and results of the simulation. Include flowcharts, diagrams, parameter tables, and plots of the path planner state. Discuss how you approached the problem, and any interesting or unexpected results.

7.3 20% Simulation Environment

Create a simulation environment to run all of the game logic and planner functions. Submit source code, figures, and animations as articles of proof. (Specifically discussed in the report.)

7.4 25% Path Planner Implementation

Implement the planner for the hero robot. (Included with your source code submission.) Discuss your implementation in the report, and include any relevent details such as design concepts, heuristics, pseudocode, or flow charts.

7.5 25% Animation

Produce a video of an interesting run of the simulation. This could show the hero robot successfully defeating all the enemies and reaching the goal in a minimum number of moves, or it could show a hopeless run where it gets backed into a corner and succumbs to multiple robots.

Either way, post this animation to the discussion forum, and discuss why you think this result is significant, and what is interesting about it. Especially include insight into how the planner operates and makes decisions, and its actual versus expected performance in the game.

8 Submission

This assignment is due 2025-09-15 @ 11:00 UTC. *Late submissions are not accepted.* Upload completed assignment components (as individual files, **not** a single ZIP or TAR file) to the course site on Canvas.

Submit work early, as partial credit for incomplete work is preferred to earning a zero for missing work. Submissions may be ammended up until the deadline.

9 Grading Rubric

Refer to Table 1 for the grading rubric.

Table 1: Grading rubric

Weight	**Type**	**Component** (See Table 2)
10%		Full Source Code
20%		Report
20%		Simulation Environment
25%		Path Planner Implementation
25%		Animation

Table 2: Submission legend

📄	report or brief writeup
📄	source code (or link to revision control)
📄	video presentation, or simulation animation
📈	data plot or image
💬	post in the discussion forum on Canvas

10 References

Péter Fankhauser and Marco Hutter. A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation. In Anis Koubaa, editor, *Robot Operating System (ROS) – The Complete Reference (Volume 1)*, chapter 5. Springer, 2016. ISBN 978-3-319-26052-5. doi: 10.1007/978-3-319-26054-9{_}5. URL http://www.springer.com/de/book/9783319260525.

Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, May 2006. ISBN 9780521862059. URL http://lavalle.pl/planning/.

11 List of URLs

```
http://wiki.ros.org/costmap_2d ........................................ p. 2
https://archive.org/details/robotmine.qb64 ............................ p. 1
https://canvas.wpi.edu ............................................ pp. 5, 6
https://canvas.wpi.edu/ ............................................... p. 4
https://en.wikipedia.org/wiki/Chase_(video_game) ...................... p. 1
https://github.com/anybotics/grid_map#installation .................... p. 2
https://minigrid.farama.org/index.html ................................ p. 1
https://www.mathworks.com/help/reinforcement-learning/ug/
  create-custom-grid-world-environments.html .......................... p. 2
https://www.pygame.org/wiki/GettingStarted ............................ p. 1
```

Last update: July 25, 2025

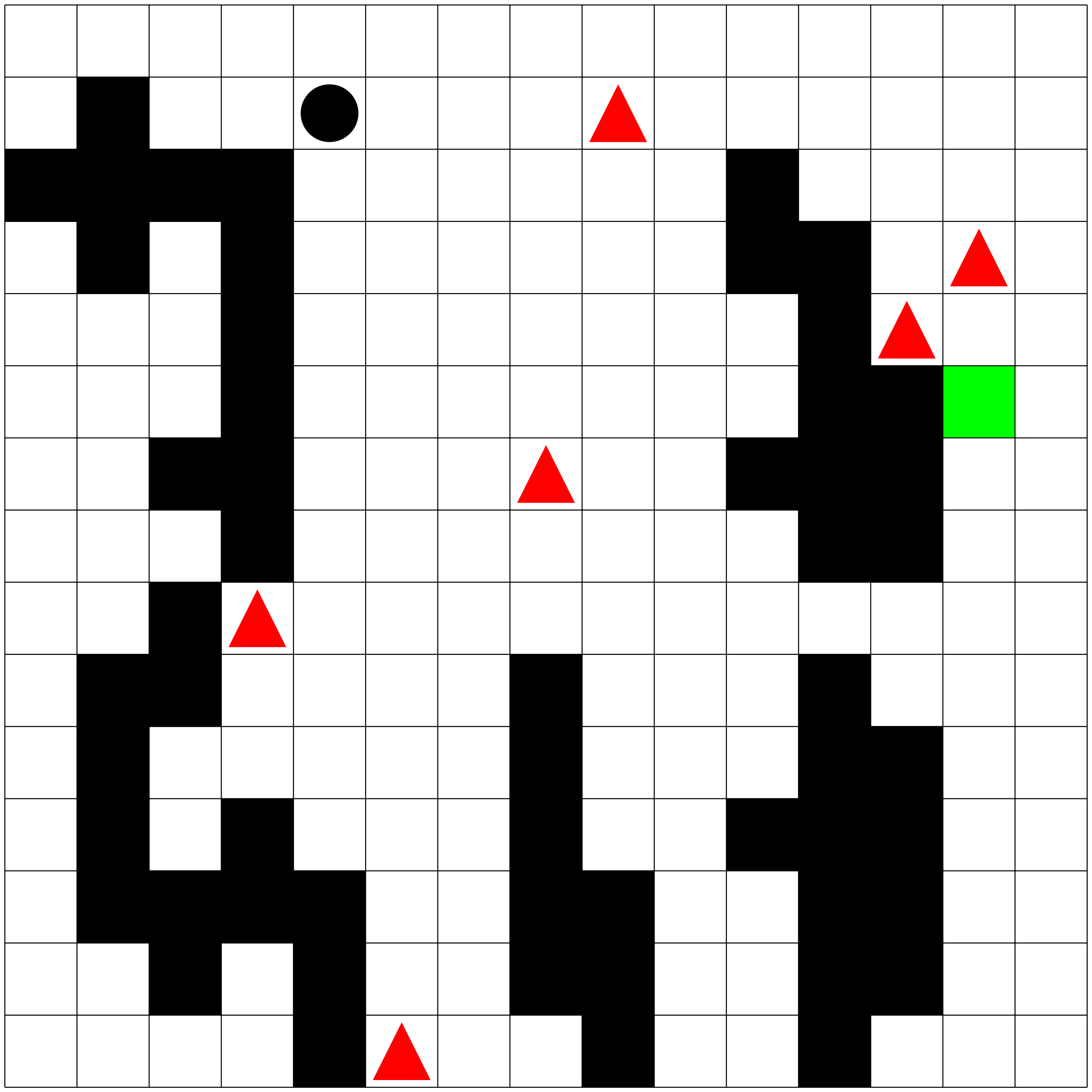


Figure 3: Example Obstacle Course of Doom.