Please enter your name and uID below.

Name: Jason Crandall

uID: u0726408

Submission notes

- **(PS 1 specific) This is a warm-up assignment. Full points will be given for answers that have some correct elements and that clearly been given a legitimate effort.**

- *Solutions must be typeset* using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the semester, so please retain the original files.
- Please remember that for problem sets, collaboration with other students must be limited to a high-level discussion of solution strategies. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1.

The main structure of my algorithm for this problem set revolves around 2 key data structures. The first being the “bag” which I use a built-in stack in c#. and then I also utilize a 2-dimensional int array to keep track of the reachable points in the maze. In order to represent vertices of the graph, rather than create a separate object with the coordinate information, I merely push the row into the stack first, and then the column. Knowing this, anytime I pop from the stack, I know to expect the column first followed by the row. The way that I utilized WFS in this scenario is to return the amount of treasure collected from all the nodes that a player can reach. Using the 2-dimensional parallel array, I use WFS to traverse the maze from the starting point, and if the current node is an int, I add that to the found treasure, breaking early if the found treasure is already greater than a previously found better solution. This way, all the paths that are reachable from the start are marked, and the treasure found is returned.

With this logic in place, in order to find the optimal placement of a monster, I simply attempt placing the monster in every valid place in the grid, then run it through my WFS traversal algorithm. I store the lowest found treasure globally, so with each passthrough of the WFS, I am able to track if the solution is more optimal than a previous one. If such is the case, I save the coordinates used to place the monster and return them.

Using this methodology, a reasonable upper bound for this algorithm would be O((rows*columns)^2) as I attempt to place a monster on every single possible square before finishing which is bound by O(rows*columns). Then for each of these monster placements, I must traverse the maze with WFS, which could end up traversing the whole thing again, thus giving an exponential bound.

2.

3.