# CSC Problem Set 02 +X Report

**Name:** John Cheek
**Email:** jwcheek2@ncsu.edu
**Section:** 001
**Collaborators:** N/A

## Summary:

In my original implementation, map 3 had a very long run time. I wanted to explore this for my +X percent, so I decided to create a few more maps to test what exactly was the cause of the long runtimes. I had many different ideas on what was causing them but ultimately through the creation of additional environments, I learned that the agent had trouble navigating environments where large amounts of open space were present and where the goal is blocked by a few restrictive ‘walls’ of impassable tiles.

## "+X" Concept:

I noticed that on this problem set and the first one the agent struggled with Map 3. In this problem, Map 3 had the longest run time. I wanted to figure out why that was and test the limits of my algorithm. For my +X percent, I wanted to create more maps like Map 3 with the goal of learning more about the agent’s goal finding ability with my iterative deepening DFS algorithm. I noticed the main obstacle on Map 3 is the constricted pathway as there is a wall of impassable tiles with a 2x2 hole in between and the target is in the opposite corner. I figured tight pathways like these would be difficult for the agent to navigate, so I tried to add those into my maps. Additionally, the position of the goal seemed to matter greatly, so in some maps (Namely map 6, 8, and 10) I put the goal blocked by many tight walls. In the technical implementation, I go into my thought process behind creating each map.

# Technical Implementation of +X:

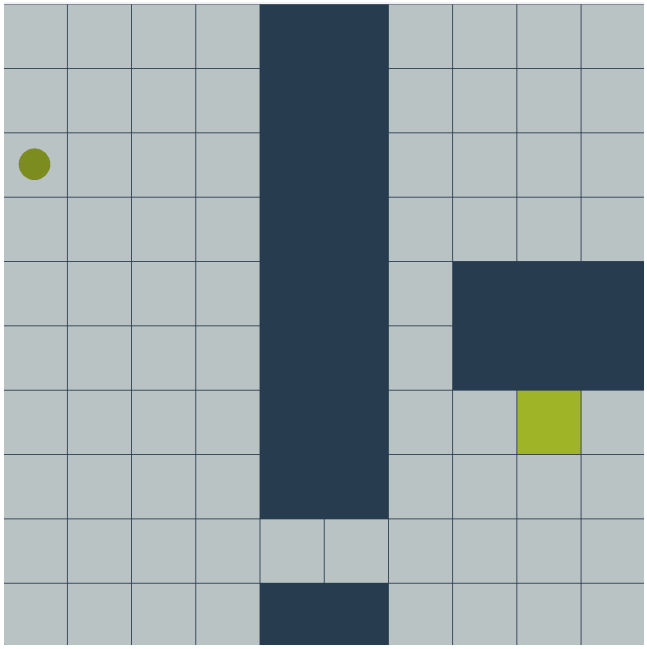
To test these maps, I simply modified the original test case to include the new maps for example:

```
@Test
public void testEnvironment06 () {
    final String map = "maps/public/map06.txt";

    for ( int trial = 0; trial < NUM_TRIALS; trial++ ) {
        final RunSimulation sim = new RunSimulation( map, ITERATIONS );
        try {
            // CompletableFuture will run an iteration's simulation
            // asynchronously for 5000 milliseconds (5 seconds) before
            // timing out.
            // This is to prevent infinite loops or inefficient
            // implementations
            // that are brute forcing solutions.
            final Runnable simulation = () -> sim.run();
            CompletableFuture.runAsync( simulation ).get( DURATION, TimeUnit.MILLISECONDS );
            if ( sim.goalConditionMet() ) {
                successfulTrials++;
            }
        }
        catch ( final InterruptedException e ) {
            if ( displayExceptions ) {
                System.out.printf( error, trial, e.getClass() );
            }
        }
        catch ( final ExecutionException e ) {
            if ( displayExceptions ) {
                System.out.printf( error, trial, e.getClass() );
            }
        }
        catch ( final TimeoutException e ) {
            if ( displayExceptions ) {
                System.out.printf( error, trial, e.getClass() );
            }
        }
    }

    final String msg = String.format( line, 5, successfulTrials / ( NUM_TRIALS * 1.0 ) * 100, NUM_TRIALS );
    System.out.println( msg );
    assertTrue( successfulTrials / ( NUM_TRIALS * 1.0 ) >= 0.7, msg );
}
```

This +X implementation revolved around creating multiple 10x10 maps which consist of text files with the following key:

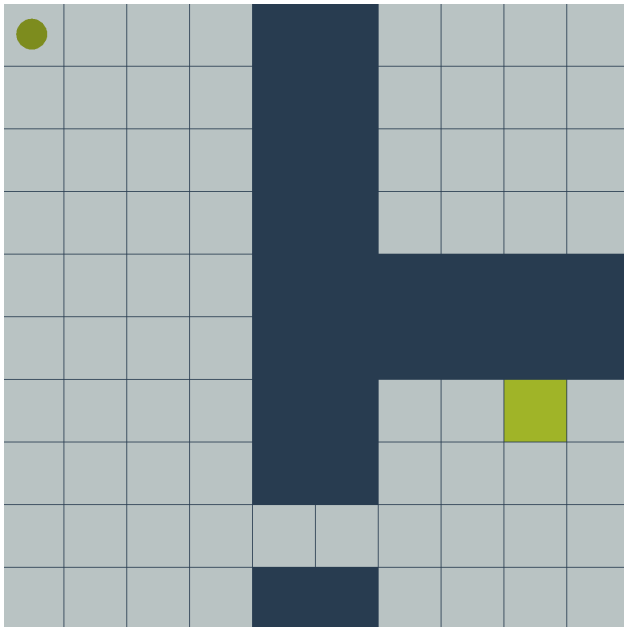
C = Clean Tiles
W = Wall Tiles
T = Goal Target

## Map 6:



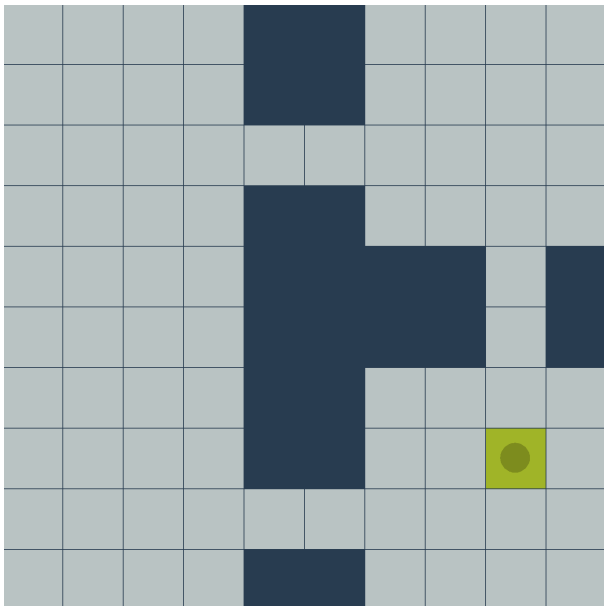
The idea behind this map has a single tight passage with the goal on the other side. There is also a decoy tunnel leading up to an empty chunk of blocks in an attempt to trick the pathfinding algorithm.

## Map 6.5:



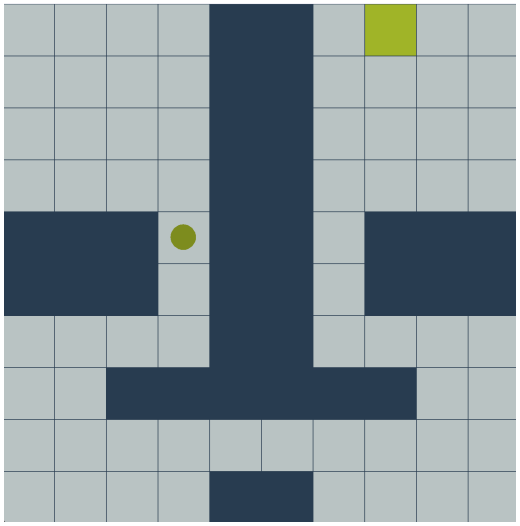
Just for fun, here is map 6 with the passage closed off. I wanted to see if this affected the test runtime. Surprisingly, it made it worse.

## Map 7:



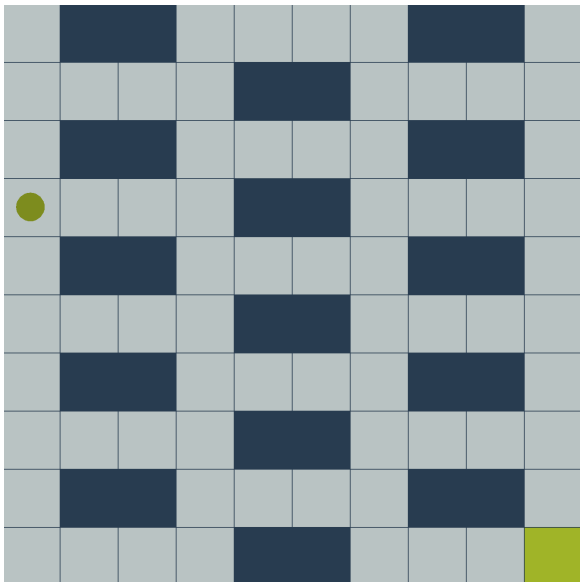
Maps 6 and 6.5 gave me an idea: Maybe their runtime was long because there was only 1 path the agent could take (through the 1x2 passageway). If I slightly altered the map to include 2 possible paths to the goal, will the algorithm perform better?

## Map 8:



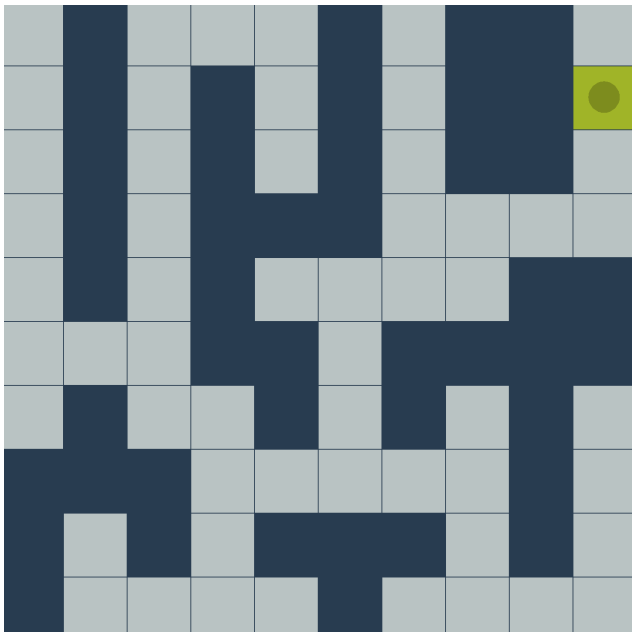
This map has multiple tight passages surrounded by wall tiles. Since my implementation skips over impassable tiles when searching, theoretically this map should perform better than some of the others. There is still only 1 path to the goal, but this time there are many impassable tiles.

**Map 9:**



This map has many tight paths through it, but there are many possible paths to the goal

**Map 10:**



This map is a maze with multiple dead ends and one path through the environment

## Evaluation and Results:

| Map | Test Success Rate | Test Runtime |
|---|---|---|
| 03 (Base) | 100/100 | 1m 13s |
| 06 | 100/100 | 3m 51s |
| 06.5 | 100/100 | 4m 04s |
| 07 | 100/100 | 0m 48s |
| 08 | 99/100 | 3m 59s |
| 09 | 100/100 | 0m 03s |
| 10 | 100/100 | 0m 01s |

From these tests, it can be seen that maps 6, 6.5, and 8 had very long test runtimes of over 3.5 minutes. My theory is that this is because of the large amounts of open space combined with the tight paths. As the other maps like 9 and 10 seemed to reinforce, the tight passageways alone did not increase the runtime, but it was the large amounts of open space with limited options to proceed that did greatly increase this runtime. Since there is lots of open space, there are many paths for the iterative deepening DFS to explore with only one or 2 paths to reach the goal

I expected Map 10 to have a very long (if not the longest) runtime but surprisingly it had the lowest. However, this makes sense. My algorithm utilizes a depth-first search approach and the paths in the maze are very linear; therefore it was able to back out of dead ends quickly and not have any additional paths to explore since it does not consider impassable tiles to the search tree. was also surprised by Map 9's performance. There is lots of open space, but I believe the large amounts of potential paths to the goal helped this.