The Prisoner's Problem

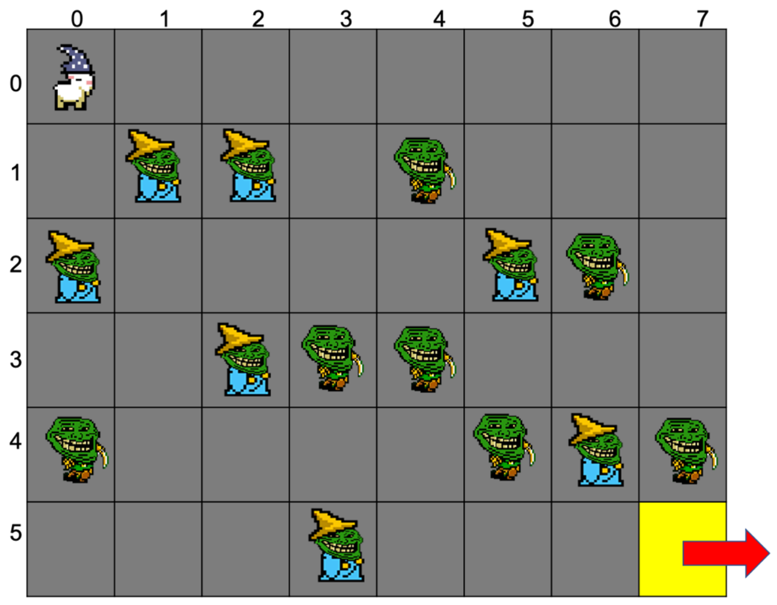
Goals of the Assignment

Practice using graphs and searches to find a path in a graph. This assignment will require the `Graph` interface and the `Vertex` and `AdjacencyGraph` classes implemented during class, as well as the implementations of the BFS and DFS path algorithms. As always, you are expected to practice good software engineering, including the Git workflow. JUnit unit testing is not expected for this assignment. Read this document ***in its entirety*** before asking for help from the course staff.

The Prisoner's Problem

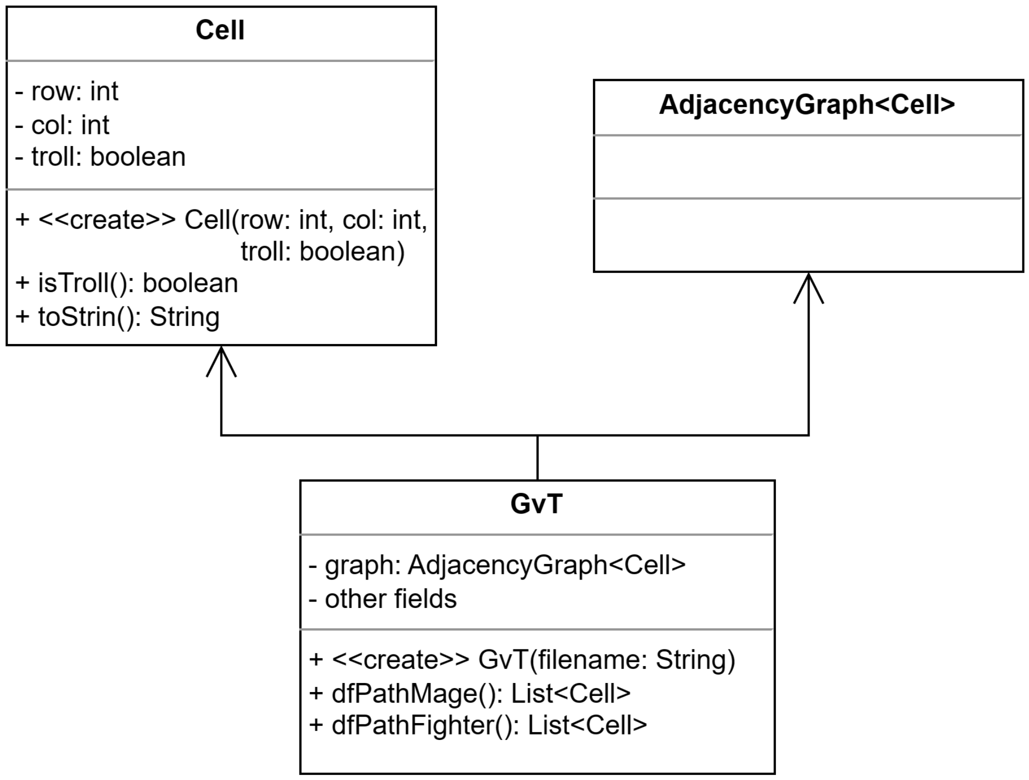
An infamous federal correctional facility consists of a grid of R x C jail cells, each devoid of natural light or fresh air. Lurking within some of these cells are ferocious trolls, the big beasts that wait for prisoners to come and then pounce and devour their prey. A prisoner named Mage is confined in the top leftmost cell of the correctional center.

Each jail cell is connected to its neighbors by a door, providing the only means of movement throughout the facility. The only way out is through the bottom rightmost cell, known as the "escape room," which opens up to the outside world. Each cell is equipped with a sensor that detects the departure of a prisoner. Upon detection, an automated mechanism promptly initiates a lockdown of all access points surrounding the chamber. Your task in this assignment is to use DFS to find a safe path for Mage to reach the escape room.



Activity Part 1 - Mage

The following (incomplete) UML diagram is provided for you to use as a guide. Depending on your implementation, you may need to add additional fields and/or methods.

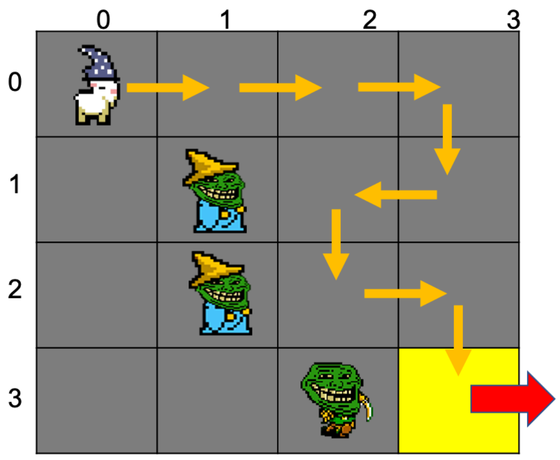

Cell
- row: int
- col: int
- troll: boolean
+ <<create>> Cell(row: int, col: int, troll: boolean)
+ isTroll(): boolean
+ toStrin(): String
AdjacencyGraph<Cell>
GvT
- graph: AdjacencyGraph<Cell>
- other fields
+ <<create>> GvT(filename: String)
+ dfPathMage(): List<Cell>
+ dfPathFighter(): List<Cell>

1. Create a new package called "`gvt`” in your repository for this assignment. Use the UML as a guide to create a class named `Cell` that represents a jail cell. A cell has a string representation such as “(<row>,<column>)”.
2. Copy and paste `AdjacencyGraph.java` into the `gvt` package and modify the class according to the suggestions provided below so that they can be used in the assignment.
    a. Add `getVertex(E value)` that returns the vertex containing the specified value.
    b. Change the access modifier of `visitDFPath()` to `protected`.
    c. While it is not mandatory, it is acceptable to make modifications or additions to the provided classes, Vertex and AdjacencyGraph.
3. Create a file named “`GvT.java`”.
    a. dfPathMage() uses the modified AdjacencyGraph to find a DF path (if it exists).
    b. Add a constructor that initializes fields by parsing a text description of a prison structure. Take a moment to examine the provided text files in the **data/dfs** folder.
        - The first line contains the number of rows and columns.

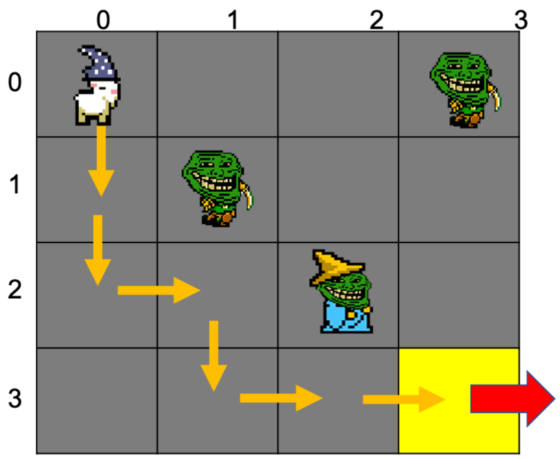
- Each of the remaining lines shows the content of cells in a row, where ‘E’ indicates the absence of a troll and ‘T’ indicates the presence of a troll.

4. Create a `main` and use the `graph*_mage.txt` files to test the constructor and `dfPathMage()`.

One possible path from location (0,0) to location (3,3) on the 4x4 board



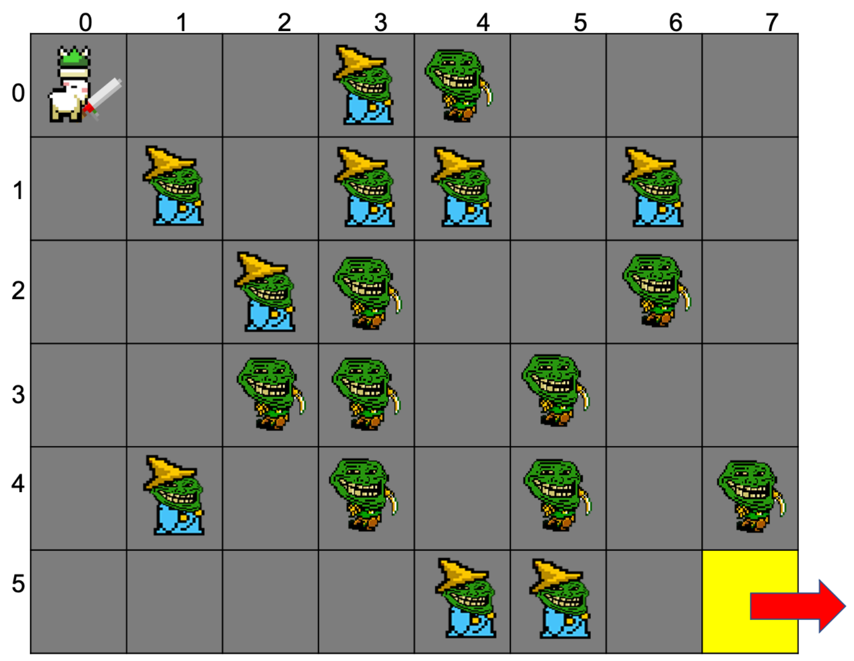
[(0,0),(0,1),(0,2),(0,3),(1,3),(1,2),
(2,2), (2,3),(3,3)]



[(0,0),(1,0),(2,0),(2,1),(3,1),(3,2),(3,3)]

Activity Part 2 - Fighter

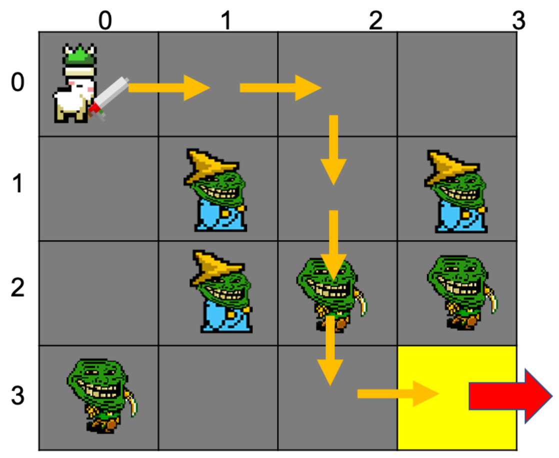
In a separate building in the infamous facility, a prisoner named Fighter is captive in the top leftmost cell. Despite the increased presence of trolls making things look gloomier than ever, Fighter was able to smuggle a weapon called Cleave. While powerful enough to bring down one troll, the knife would be shattered in the process, leaving the prisoner without a weapon for future battles. Your final task is to use DFS to find a safe path for Fighter to reach the escape room.



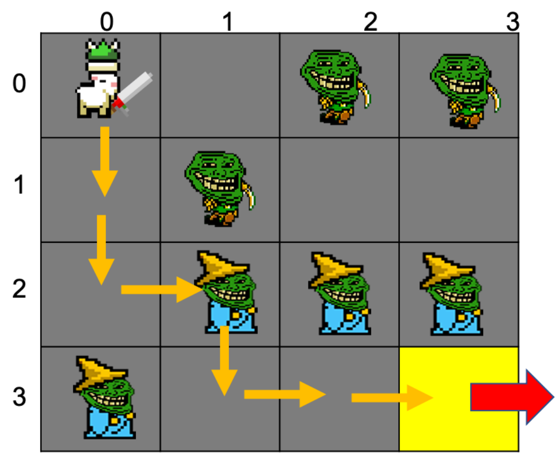
1. Add the `dfPathFighter` method to the GvT class. The method returns a secure route for Fighter.

2. In `main()`, use the `graph*_fighter.txt` files to test `dfPathFighter()`.

One possible path from location (0,0) to location (3,3) on a 4x4 board



[(0,0),(0.1),(0,2),(1,2),(2,2),(3,2),(3,3)]



[(0,0),(1,0),(2,0),(2,1),(3,1),(3,2),(3,3)]

Submission Instructions

You must ensure that your solution to this assignment is pushed to GitHub *before* the start of the next lecture period.