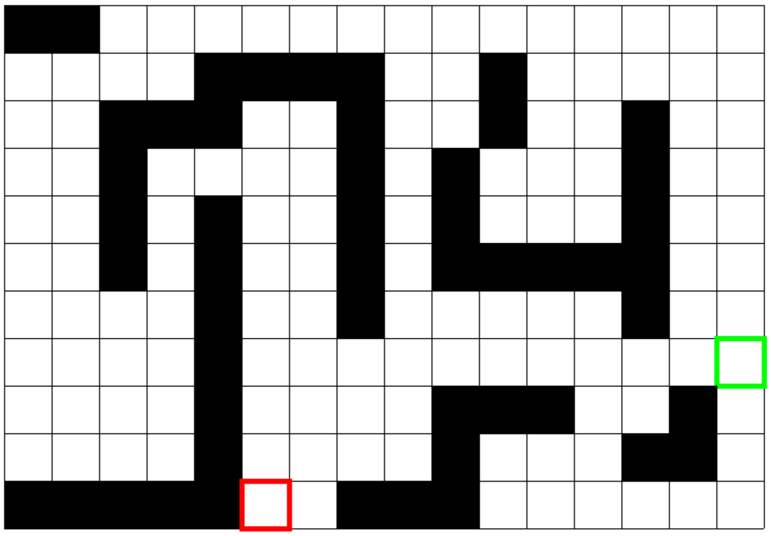
CSC 3510
Introduction to Artificial Intelligence
Florida Southern College

HW2: Uninformed Search
Due: Monday, October 30, 2023

The purpose of this assignment is to ensure that you are able to:

- understand the conceptual differences between BFS, DFS, and UCS
- program uninformed search algorithms in Python
- work collaboratively with a peer to efficiently solve problems

1. Consider the maze shown below (`maze.png`), in which the objective is to travel from the start cell (red) to the goal cell (green). For each of the following cases, number the cells in the order they are visited using the specified search algorithm. Stop numbering after you reach the goal (i.e. the goal should contain a number). You may assume the start cell has a label of 0 and that loops are prohibited (i.e. each cell should only contain one number). There is no programming required for this problem.



   (a) BFS using North-East-South-West order.

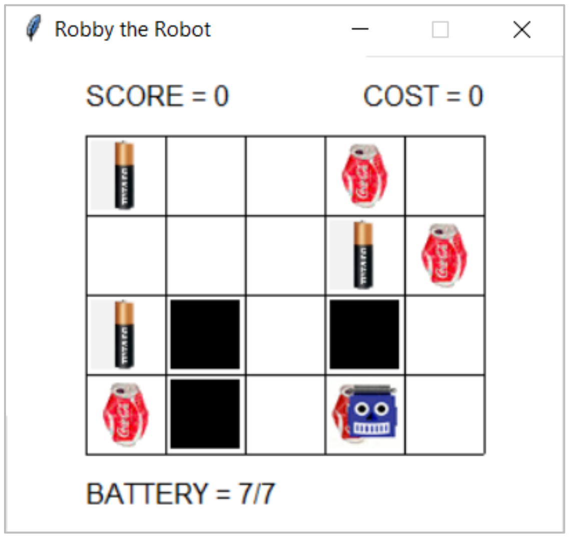
   (b) DFS using North-East-South-West order.

   (c) DFS using West-East-South-North order.

   (d) UCS using North-East-South-West order with North costing twice as much as the others.
   (HINT: Keep track of the cost to get to each cell in addition to the search index!)

2. Now that you have demonstrated a basic conceptual understanding of how uninformed search works, you will program Robby the Robot to pick up cans without running out of battery using breadth-first search (BFS). Starter code has been provided for you (see `robby_search.py`). The portions of code that you need to edit are identified by comments, e.g. `# ***EDIT CODE HERE***`. A couple sample worlds are also included (`world0.txt, world1.txt`), but your code will be tested on other worlds

as well, so it may benefit you to create your own worlds for debugging. Here is what the first sample world should look like at the start if programmed correctly:



Also, here is a summary of the programming tasks that you must complete:

(a) Use the `argparse` module to enable the user to enter command line arguments. Details are provided in the comments of the code.

(b) Read the world parameters from file. The format for the world files is:
- rows and columns in the world
- row and column of Robby's initial position
- the initial contents of the world, given row by row
  - `. = empty cell`
  - `B = battery`
  - `C = can`
  - `W = wall`

(c) Create the world using the `robby` library (v2) methods for loading contents, positioning Robby, and setting the full battery level.

(d) In the infinite loop, edit the selection that determines whether all the cans have been picked up.

(e) Complete breadth-first search in the provided function. Use whichever queue structure you prefer.

(f) Edit the `issolved` method to check whether a series of actions (a path) solves the problem.

(g) Edit the `isvalid` method to check whether a series of actions (a path) is valid for the given world. Invalid paths include:
- if Robby runs out of battery
- if Robby goes out-of-bounds
- if Robby runs into a wall
- if Robby repeats a state in memory (you do not need to edit this part at all)

(h) Write code to simulate the final path (i.e. move Robby) in the world when "`Return`" is pressed.