Computer Science 5400
Artificial Intelligence

Spring 2024

Puzzle Assignment #3

Version 24.03.25

DUE DATE : Monday, April 8th, 11:59:59pm

Assignment:

Create a program that implements **Iterative Deepening Depth-First-Search** (**ID-DFS**) over the **Act-Man-II** game to find a desired sequence of game actions. You can read more about ID-DFS in section 3.7.3 [Link] of the textbook. If you own the Norvig textbook, you can also read Section 3.3 for more details on ID-DFS.
Your program's objective is to find a sequence of actions (for Act-Man) that achieves either: A **victory** condition, **or** that keeps Act-Man **alive** after **10** turns.

Specifications:

Your program shall read two **command line arguments**. The first argument will be the name of the **input file** and the second argument the name of the **output file** to which to write your output. The input file will contain an **Act-Man-II** dungeon specification, which will include complete information about the dungeon and the initial conditions. Your program shall write to the output file a sequence of actions for Act-Man that achieves the required postconditions.
Whatever is displayed to the screen during program execution will not have direct consequence on your grade.

Initial Game Board Preconditions:

- You can safely assume that there will be **at least one** way to achieve the desired postconditions from the initial input field.

Input File:

The **Act-Man-II** dungeon specification input file will have the same format as the one from *Puzzle Assignment #1*.

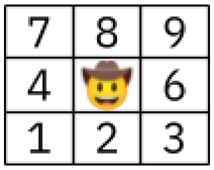
Input File Example:

```
5 11
###########
#   #  @ G#
#A#     # #
#   #D    #
###########
```

Output File:

Your output file should follow the following format:

- The first line should contain the sequence of valid actions executed by Act-Man.
- Move actions will be encoded using *numeric keypad* order.



- The firing of the magic bullet action will be encoded with the letters `'N','S','E','W'`.
- The next line should contain a single integer: The score attained by Act-Man. after executing the sequence of moves.
- The next lines should display, in a format similar to the input file, the final configuration of the dungeon after executing the sequence of moves.
- You must follow the format presented or else your submission will not be

graded

Sample Output File #0:

```
86E
38
###########
# A@#  @  #
# #     # #
#   #     #
###########
```

Note: Your output may not necessarily be the same as the sample, as your program may find another solution that satisfies the requirements.

Sample Input File #1:

```
9 12
############
#    ##G##D#
# ## ## ## #
# ## ## ## #
# ### A    #
# ## # ## ##
# ## # ## ##
#    #D##G##
############
```

Sample Output File #1:

```
412N24
45
############
#    ## ## #
# ## ## ## #
# ## ## ## #
# ###      #
# ## #@## ##
# ## # ## ##
#  A # ## ##
############
```

Sample Input File #2:

```
8 15
###############
#D         G  #
#  D#     #   #
#      A  @  D#
#   @         #
#G  #     #  D#
# G         G #
###############
```

Sample Output File #2:

```
621W17
65
###############
#             #
#   #     #   #
#         @   #
#   @      @  #
#   #A@   #   #
#             #
###############
```

Sample Input File #3:

```
13 13
#############
## D #G# D ##
# ### # ### #
##G# ### #G##
### ##### ###
## ####### ##
```

```
#A######### #
## ####### ##
### ##### ###
## # ### # ##
# ### # ### #
##  ## ##  ##
#############
```

Sample Output File #3:

```
3311364646
50
#############
##   # #   ##
# ### # ### #
## # ### # ##
### #####D###
## #######G##
# ######### #
## ####### ##
### ##### ###
## #@### # ##
#D### # ### #
## A## ##  ##
#############
```

Notes:

- As you are using **Iterative Deepening Depth-First Search**, If there is a way to achieve victory in less than **10** moves, your program should find it.

Submission:

Place all code/scripts in your git repository in the course's GitLab server, [link]
(You should have the repo open this week).
A file named '`ReadyForGrading.txt`' will be placed in your repository.
IMPORTANT: When your assignment is complete and ready, modify the content of the '`ReadyForGrading.txt`' file to '`Yes`'. This will indicate to the graders that your

assignment is ready to be graded.

Your main file shall be called "**`puzzle3`**" regardless of extension. (e.g. if you are programming in C++, your main file should be called "`puzzle3.cpp`". If you are programming in Java your main file should be called "`puzzle3.java`"). Your main file should include your **name**. Include any other necessary files in your submission.

You can implement your assignment using one of the supported programming languages, but it must be in a way compatible with the Computer Science Department's Linux machines. In order to accommodate such different languages, your submission should include a bash script named '`run.sh`' that **compiles** and **runs** your program with all the necessary options and commands.

For example, the following is a possible '`run.sh`' script for **C++ 11**.

```
#!/bin/bash
g++ -std=c++11 puzzle3.cpp -o puzzle3.ex
./puzzle3.ex $1 $2
```

A sample '`run.sh`' script for **Python3** if the program is called '`puzzle3.py`' :

```
#!/bin/bash
python3 puzzle3.py $1 $2
```

A sample '`run.sh`' script for **Java** if the program is called '`puzzle3.java`' :and the main class is called '`puzzle3`'.

```
#!/bin/bash
javac puzzle3.java
java puzzle3 $1 $2
```

Your program will be evaluated and graded using the command:

```
./run.sh gradinginput.txt gradingoutput.txt
```

IMPORTANT: Remember to make your '`run.sh`' file executable as a script with the LINUX command:

```
chmod +x run.sh
```

Grading:

Your program will be evaluated and graded on the Computer Science department's Linux machines so your program needs to be compatible with the current system, compilers and environment.

Grading Weights:

Your program will be evaluated and graded on the Computer Science department's Linux machines so your program needs to be compatible with the current system, compilers and environment.

Description	**Weight**
Correct Actions and Final State Generation.	40%
Specified Search Algorithm.	40%
Good Programming Practices (adherence to coding standards, modular design, documentation etc.)	20%

A Primer for Puzzle Assignment #3:

Intuition:

We continue exploring the **Act-Man-II** game tree, but this time we explore it **Depth-First** up to a depth that is incremented iteratively.

Implementation:

We reuse from *Puzzle Assignment #2* the **GameBoard** data structure and the **TransitionFunction()**.

As a reminder:

GameBoard : A "snapshot" of a **Act-Man-II** game, including location of the of Act-Man and the monsters.

TransitionFunction (s, p) → s'

IINPUT:	s : a GameBoard
	p : A sequence of Act-Man actions
OUTPUT:	s' : the GameBoard that results from executing p in s.

Refining ID-DFS for the Act-Man-II game:

The main changes from before is that the frontier is now a **LIFO stack**, there is a depth limit that controls when candidate plans are tested, and two functions are used, one to explore the game tree depth first, and another one to iteratively increase the depth limit.

```
PROCEDURE Act-Man-2-Bounded-DFS
INPUT s0 : the initial GameBoard
      goal(s) : boolean function that tests if the goal is true in
                GameBoard s.
      depth_limit : integer, depth to limit search.

OUTPUT : a sequence of actions that takes the game from s0 to a state that
         satisfies the goal.
         OR true : if the depth_limit is reached
         OR false : if no plan exists

VAR
    frontier : LIFO Stack of sequences of actions.
    limit_hit : boolean

BEGIN
    limit_hit ← false
    push [] (empty sequence) into frontier
    WHILE frontier is not empty
        pop sequence of actions p = [a0, a1, a2, ..., ak] from frontier

        IF length of p = depth_limit
            sk ← TransitionFunction( s0, p )
            // sk is the GameBoard that results from executing p in s0
            IF goal(sk)
                 RETURN p
            IF p has valid moves
                 limit_hit ← true

        ELSE
            FOR every valid action a at sk
                push [a0, a1, a2, ..., ak, a] into frontier
    END
    RETURN limit_hit
END.
```

```
PROCEDURE Act-Man-2-ID-DFS
INPUT s0 : the initial GameBoard
      goal(s) : boolean function that tests if the goal is true in
                GameBoard s.

OUTPUT : a sequence of actions that takes the game from s0 to a state that
         satisfies the goal.
         OR false : if no plan exists

VAR
    depth : integer
    res : sequence of actions OR boolean

BEGIN
    depth ← 0
    REPEAT
        res ← HappyCows-Bounded-DFS( s0, goal(), depth )
        IF res is a sequence of actions THEN
            RETURN res
        depth ← depth + 1
    UNTIL res is false
END.
```

This is, of course, just a first draft.

Further refinement:

You are not required to implement pruning (avoiding cycles or repeated states) on this assignment, but it **will** help it speed up execution. Many of the components carry from *Puzzle Assignment #2*, but you need to be careful as this is a different algorithm.

Good Luck!