Computer Science 5400
Artificial Intelligence

Spring 2024

Puzzle Assignment #1

Version 2024.02.23

DUE DATE : Monday, March 4th, 11:59:59pm

Assignment:

Create a program that **implements** and **plays** the Act-Man-II game. This means that your program will not only move the monsters according to the rules of the game, but will also determine how to move Act-Man. For this first assignment, your program will play the game by choosing **random**, but **valid**, moves for Act-Man to perform.

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 game specification, which will include complete information about the dungeon and the initial conditions. Your program shall write to the output file a **random**, but **valid**, sequence of actions for Act-Man-II to perform and the final conditions of the dungeon. A valid action is either the firing of a magic bullet (only **once** per game). or one that moves Act-Man from its current location cell to an adjacent open space cell and does not attempt to move Act-Man into a wall.

A **valid** action does not need to be a **good** action. In this assignment we are only concerned with correctly computing the consequences of actions.

Your program shall generate a sequence of actions that either ends in a **game-over**, ends in **victory**, or is of length **7**. (moves beyond a game-over condition will be considered invalid). Whatever is displayed to the screen during program execution will not have direct consequence on your grade. There is no need for a gui, text alone is fine, though we are not going to stop anyones artistic expressions. The specific format of the input and output files is described below.

Input File:

The Act-Man specification input file will be formatted as follows:

- The first line contains two integers **`R`**, **`C`** specifying the number of rows and columns of the game board respectively.
- The next **`R`** lines of **`C`** characters each represent the game board dungeon, where:
    - `' '` indicates an open space.
    - `'#'` indicates a wall.
    - `'A'` indicates Act-Man's initial location.
    - `'D'` indicates a demon's initial location.
    - `'G'` indicates an ogre's initial location.
    - `'@'` indicates a monster's corpse location.

Input File Example:

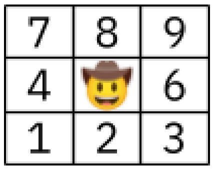
```
5 11
###########
#   #  @ G#
#A#     # #
#   #D    #
###########
```

In the above example input file, the game board is 5 rows by 11 columns. The next 5 rows show the layout of the dungeon and the initial locations of Act-Man and the monsters.

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.
- If Act-Man is **dead** at the end of the sequence of moves, indicate Act-Man's final location with an 'X' character
- You must follow the format presented or else your submission will not be graded

Output File Example 1:

```
861396W
34
###########
#   #  @  #
# #@A   # #
#   #     #
###########
```

In the above example output file the first line is the sequence of moves executed by Act-Man, the next line shows that a score of 34 is attained, and the next 5 likes display the final configurations of the game board after the execution of the shown sequence of moves

Output File Example 2:

```
96
0
###########
#  X#  @  #
# #     # #
#   #     #
###########
```

In the above example output file the first line is the sequence of moves executed by Act-Man. These actions lead to Act-Man dying because he is reached by a monster, so the next line shows that a score of 0 is attained, and the next 5 likes display the final configurations of the game board after the execution of the shown sequence of moves.

.

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 "**hw1**" regardless of extension. (e.g. if you are programming in C++, your main file should be called "hw1.cpp". If you are programming in Java your main file should be called "hw1.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 hw1.cpp -o hw1.ex
./hw1.ex $1 $2
```

A sample 'run.sh' script for **Python3** if the program is called 'hw1.py' :

```
#!/bin/bash
python3 hw1.py $1 $2
```

A sample 'run.sh' script for **Java** if the program is called 'hw1.java' :and the main class is called 'hw1'.

```
#!/bin/bash
javac hw1.java
```

```
java hw1 $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:

Description	Weight
Correct Action and State Generation	80%
Good Programming Practices (adherence to coding standards, modular design, documentation etc.)	20%