Computer Science 5400
Artificial Intelligence

Spring 2022

Puzzle Assignment #3

Version 03.12

DUE DATE : Monday, March 14th, 11:59:59pm

Assignment:

Create a program that implements **Iterative Deepening Depth-First-Search** (**ID-DFS**) over the Pengu 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 Pengu) that achieves a score of at least **16**.

Specifications:

Your program will follow the same input/output structure from *Puzzle Assignment #1* and *Puzzle Assignment #2*. Two **command line arguments** indicate the **input** and **output** files. The **input** file contains an Pengu game initial board specification in the same format used in previous assignments. The **output** file will also follow the same format used in previous assignments although this time the first line shall contain a sequence of valid moves that if executed by Pengu in the input file initial

game board achieves a score of at least **16**.
As before, your output files are what is relevant to your grade, whatever is displayed to the screen during program execution will be ignored.

Initial Game Board Preconditions:

- You can safely assume that there will be **at least one** way for Pengu to achieve the desired score from the initial input game board.

Input File:

The Pengu specification input file will have the same format as the one from *Puzzle Assignment #1*.

Input File Example:

```
6 20
####################
# * S# ** S # U  #*#
#*U*U*U    P  S*U*#*#
#0  ***0UUUUU0S *#*#
#S**SS* *****0*S   #
####################
```

Output File:

The output file follows the same format specification as the one from *Puzzle Assignment #2*. The only change is the desired target score.

Output File Example:

```
741842616
16
####################
#    S#     S # U  #*#
```

```
# U U U       S*U*#*#
#0      0UUUUU0S *#*#
#S* SS        P*S    #
##########################
```

In the above example output file the first line is the sequence of moves executed by Pengu, the next line shows that a score of **16** is attained, and the next 6 lines display the final configurations of the game board after the execution of the shown sequence of moves

Submission:

Same system as before. Place all code/scripts in **this assignment**'s git repository in the course's GitLab server, [link] .
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. Additionally, add to "`ReadyForGrading.txt`" the language of which you are coding in: **one** of the strings `c` , `c++` , `python` , `javascript` , `java` , `typescript` , `rust` , `c#`
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 *.cpp -o hw1.ex
./hw1.ex $1 $2
```

An example '`run.sh`' script for Python3 if the program is called '`hw1.py3`' :

```
#!/bin/bash
python3 hw1.py3 $1 $2
```

An example '`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.

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 Pengu 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 an Pengu game, including location of the walls, snow cells, hazards and fish, the score, the location of Pengu.
>
> **TransitionFunction** (s, p) → s'
> INPUT: s : a GameBoard
> p : A sequence of Pengu actions
> OUTPUT: s' : the GameBoard that results from executing p in s.

Refining ID-DFS for Pengu:

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 Pengu-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
    ENDW
    RETURN limit_hit
END.
```

```
PROCEDURE Pengu-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 ← Pengu-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 may 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!