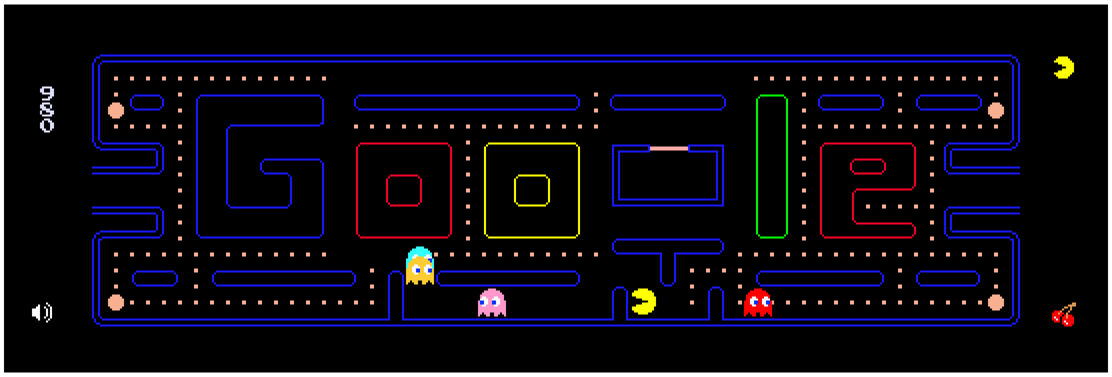
ARTIFICIAL INTELLIGENCE COURSE

Assignment 1

Searching - PACMAN Games

VERSION 1.0



Objective

You will have deeper understanding of heuristic search algorithms, and practice your programming skill in Java programming language.

Introduction

In the past, there was a famous video game that interested many people around the world. And many contests have been inspired by that game. In this semester, that game comes to our first AI assignment. That is the PACMAN game. You are required to implement some searching algorithms that you have learned in Java to help the Pacman finding the way to eat the food in a maze.

Requirements

You need to implement some Java methods in the **“Pacman.java”** file that is given to you. You should read this document carefully and follow some guidelines in the “Pacman.java” file. You should remember that, you can change anything in that file, but following the instructions in that file is a good choice.

- You can use any implementation or version of Java. Then, you **have to test** your program with **Oracle JDK version 1.6**.
- You **must use only the standard library** in JDK, **without** any external library. If you want to use some library outside, implement it in the “**Pacman.java**”.
- At the beginning of your file, you need to declare your package as **default**, which means that you don’t declare any package.

References

- http://en.wikipedia.org/wiki/Depth-first_search
- http://en.wikipedia.org/wiki/Breadth-first_search
- http://qiao.github.com/PathFinding.js/visual/
- http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
- http://en.wikipedia.org/wiki/Best-first_search
- http://en.wikipedia.org/wiki/A*_search_algorithm

Files and directories

Unzipping the Assignment1.zip that you download from Sakai, you will see the following files:

- **Assignment1.pdf** describes your assignment and guides you to complete it.
- **pacmanSim.jar** is a runnable simulator of Pacman games. It is very helpful to demonstrate your implemented algorithms.
- **maze.txt** is the input file. It depicts the maze of your Pacman game.
- **path.txt** is a sample output file. It includes the path that helps the Pacman finding the way to eat the food.
- **pacman.java** is the implementation of your algorithms. When you submit your assignment to Sakai, **only submit that file**.

The "maze.txt" file:

The **"maze.txt"** file describes a maze, the input of your program. In your program, you must **read the input** from that file in the same directory of your **"pacman.java"** file. To test your program, you can change the content of that file freely. For example, a "**maze.txt**" file could be as follows:

```
%%%%%%%%%%%%%%%%%%%%
% *            %   %
% %% %% %% %% %% % %
%        P       % %
%%%%%%%%%%%%%%%%%% %
%                  %
%%%%%%%%%%%%%%%%%%%%
```

The "path.txt" file

The "**path.txt**" file describes a path, the output of your program. In your program, you must **write the output** to that file in the same directory of your "pacman.java" file. For example, a **"path.txt"** file could be as follows:

```
[r,l,u,d]
[r,l,u,d]
[]
[]
[]
[]
[]
```

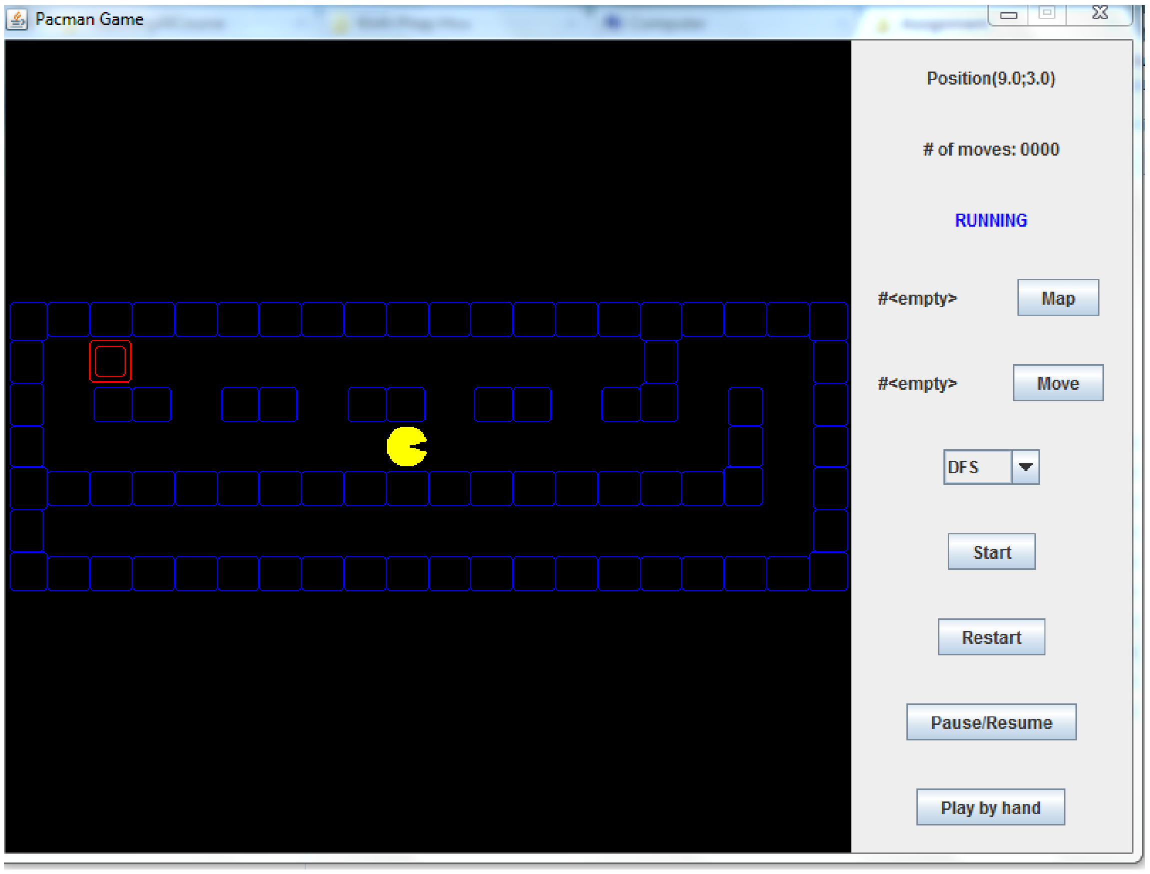
- **‘r’: right, ‘l’: left, ‘u’: up, ‘d’: down**
- There is 7 lines, each line begin with ‘[’ and end with ‘]’, separate elements by ‘,’ character (**having space after comma**). This is called a **“list”**. There is no line after the 7th line.
- From top to bottom is the result after executing the following algorithms (in order): DFS, BFS, BestFS, A*, Hill Climbing, Steepest Hill Climbing, Simulated Annealing.
- An empty list [], means there is no way to get to the food.
- Your program must output in that format. Otherwise, you will get zero for this Assignment.

The “pacmanSim.jar” file

The “**pacmanSim.jar**” file is a simulator to demonstrate your program. How to use this simulator? First, you should put your “**maze.txt**” and “**path.txt**” files in the same directory of this “**pacmanSim.jar**” file; then, double click on that file to run the simulator. If nothing appears, you should open the command line (on Windows) or terminal (on Linux and Mac OS) and **cd** to that directory and type the following command:

```
java -jar pacmanSim.jar
```

With the input above, the simulator should look like this:



Position(9.0;3.0)

# of moves: 0000

RUNNING

#<empty>

#<empty>

Notes:

- Position (9.0; 3.0): the current position of your Pacman. The (0, 0) point is at the upper left corner.
- # of moves: current number of moves that the Pacman has made.
- Running/Pausing: the current state of your game.
- Pressing "**Map**" Button to locate the input maze file. The default file is "**maze.txt**" at the same directory of the "pacmanSim.jar".
- Pressing "**Move**" Button to locate the output path file. The generated file after running your program is "**path.txt**".
- Dropdown list to choose the Algorithm to be simulated.
- Pressing "**Start**" Button to start the game. If you want to change the maze, you just need to change your "maze.txt" file's content and press "Start" Button.
- Pressing "**Restart**" Button to restart your game to initial a state.
- Pressing "**Pause/Resume**" Button to pause or resume your game.
- Pressing "**Play by hand**" or "**Play by file**" Button to change your game mode, by hand or by the "**path.txt"** file. If you choose to play by the file, you need to choose your path file, choose the algorithm you want to simulate and press "Start" Button.

The "pacman.java"

The "**pacman.java**" is the implementation of your algorithms. Again, when you submit your file to Sakai, **only submit this file**. You need to complete these tasks:

- Read the **"maze.txt"** input file at the same directory of your "**pacman.java**" file.
- Write out the result in the file "**path.txt**" file according the instruction above. You can refer to any resource on the internet, BUT make sure you understand the code clearly!!!
- Implement 7 functions of your program, including:
    - (1) DFS: Depth First Search Algorithm
    - (2) BFS: Breadth First Search Algorithm
    - (3) BestFS: Best First Search Algorithm
    - (4) AStar: A* Algorithm
    - (5) HillClimbing: Hill Climbing Algorithm

- (6) SteepestHillClimbing: Steepest Hill Climbing Algorithm
- (7) SimulatedAnnealing: Simulated Annealing Algorithm

The details of each function are described in the later part. Remember that you can add any additional method/function/class but in the same “**pacman.java**” file.

Main functions

The Pacman can move only 1 step at a time; there are 4 directions that Pacman can move. They are Up (u), Down (d), Left (l), Right (r). For each function, you need to output a **sequence of movement** (path) to help the Pacman finding the way to the Target.

- Choose a direction to get to a cell that is closer (smaller distance) to the Target, using the **Manhattan distance** between the Target Target(x0, y0) and current point (x, y):

  ```
  d = abs(x0 – x) + abs(y0 – y)
  ```

- **Choosing direction rule**: If there are two or more directions that have the same Manhattan distance, choose one according the order U -> D -> L -> R

- **If there is no way to reach the Target, then output an empty list '[]'.**

- **If the algorithm needs Evaluation Function, you MUST use Manhattan distance as described above and If there are many directions at a step, you MUST choose the direction according to the choosing direction rule above.**

DFS

This function will generate the path according to Depth First Search Algorithm, which you have learned from Data Structure and Algorithm course (http://en.wikipedia.org/wiki/Depth-first_search). Or we can simply call it **backtracking algorithm. You have to show each backtracking step**.

BFS

This function will generate the path according to Breadth First Search Algorithm, which you have learned from Data Structure and Algorithm course (http://en.wikipedia.org/wiki/Breadth-first_search). The result of this function will find the **optimal solution** (the shortest solution) to move from current position to the Target, but it is very **slow**.

BestFS

This function will generate the path according to Best First Search Algorithm. Please refer to slide 33 to slide 38 of "Chapter 3 - Heuristic Searching" for more information. (http://www.cse.hcmut.edu.vn/~tru/AI/ai.html)

A-Star

This function will generate the path according to A* Path finding Algorithm. Please refer to this website http://en.wikipedia.org/wiki/A*_search_algorithm to find out its details.

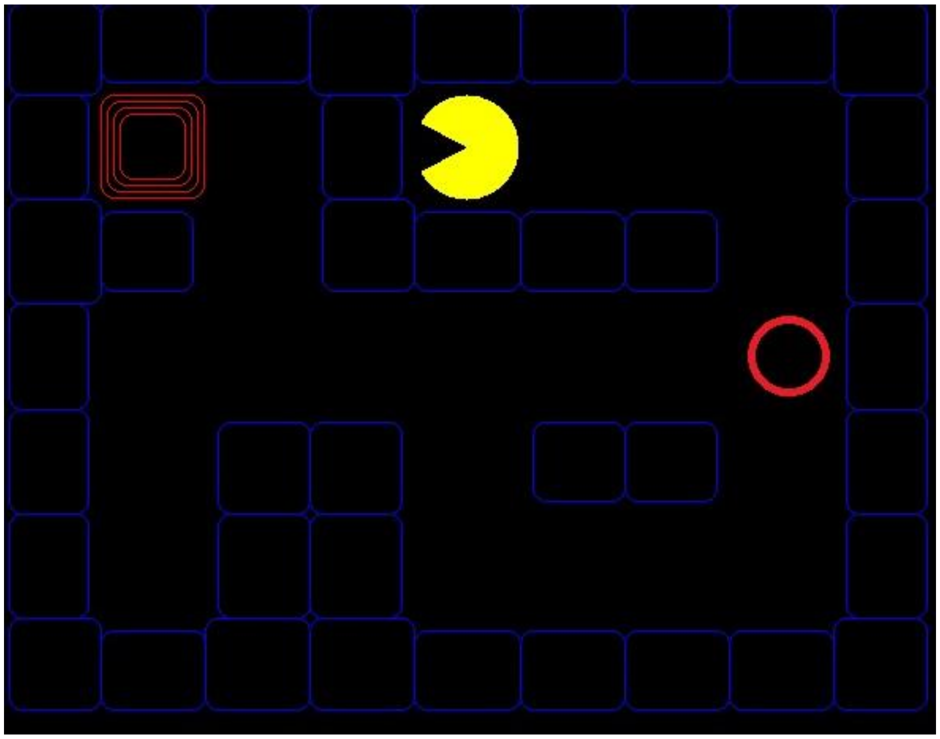
Hill Climbing

This function will generate the path according to Hill Climbing Algorithm. Please refer to slide 8 to slide 13 of "Chapter 3 - Heuristic Searching" for more information. (http://www.cse.hcmut.edu.vn/~tru/AI/ai.html)

Steepest Hill Climbing

This function will generate the path according to Steepest Hill Climbing Algorithm. Please refer to slide 14 to slide 16 of Chapter 3 - Heuristic Searching for more information (http://www.cse.hcmut.edu.vn/~tru/AI/ai.html). In the case that all possible actions on the current state lead to worse next states, the Pacman will try all possible two **consecutive** steps:

- If there is a jump of two steps that leads to a next cell that is better than current cell, the Pacman will make this jump.
- Otherwise, the Pacman backtracks to the cell that has another way to move. For example, in the following picture, the Pacman will backtrack to the cell marked by a red circle. (**You must print the backtracking step too!**)



Simulated Annealing

This function will generate the path according to Simulated Annealing Algorithm. Please refer to slide 29 to slide 32 of chapter 3 – Heuristic Searching (http://www.cse.hcmut.edu.vn/~tru/AI/ai.html).

- The initial value of ***T*** is 1,000 and ***k*** = 1.
- To calculate "new current state with probability $p = e^{(-\Delta E/kT)}$", you MUST use the **nextFloat()** function from "**java.util.Random**" to generate a random value ***p'***, then compare ***p'*** with ***p***; if $p' \leq p$, then the new state will be explored.
- After each iteration, the value of ***T*** is decreased by 1, which means $T = T - 1$.

Running & Grading

You can run your program by typing the following command. Open the command line (Windows OS) or terminal (Linux or Mac OS) and **cd** to the directory that contains the "**pacman.java**" file. Enter the following commands:

```
javac pacman.java
```

```
java pacman
```

You can use any Text Editor or IDE like Eclipse or Netbeans, but you must make sure it can run by the commands above.

Your assignment will be graded according to your implementation of "**pacman.java**" file. The below table is the percentage of points for each function. That means a test case will test all of your 7 functions. **The maximum time for running each test case is 3 seconds. Otherwise, you will get zero on that test case.**

Functions	Percentage
DFS	10%
BFS	10%
BestFS	20%
A*	20%
HillClimbing	20%
SteepestHillClimbing	10%
SimulatedAnnealing	10%

So try to get your score as high as possible.

Submission

- You submit only the "**pacman.java**" file!!!
- The deadline for this Assignment 1 is **11.55 A.M Monday 15th April, 2013**
- Where to submit: Sakai, certainly!
- Make sure that your submitted file is in plain text and do not compress it. **YOU MUST TEST YOUR PROGRAM ON LINUX OS BEFORE SUBMISSION.**

Plagiarism

You MUST do the assignment by yourself. Otherwise, for any cheating, you will get the F mark for the subject.