CSE 3401 Assignment 3
Winter 2013

Date out: March 5. Date due: March 24, at 11:55 pm

- The submitted assignment must be based on your individual work. Review the Academic Honesty Guidelines for more details.
- This assignment constitutes 6% of your total mark for the course and is marked out of 100.
- You should adhere to the coding guidelines posted on the website; comment your code and test it thoroughly.
- You may define any auxiliary relations if that helps in defining the required predicates.
- You should Submit 3 files for this assignment:
  - a3.pl, which is the source code of your solutions for question 1 (Q1.a to Q1.e). Start from the file a3.txt (available in the resources section of this assignment) which already contains some of the code you need and fill in the missing parts. Rename this file to a3.pl before editing.
  - a3test.txt, which consists of the test cases you have used to test question 1, as well as the results of testing. Include a header with your name, student number and cse login.
  - a3.pdf which consists of your answers to Section 2. Include a header with your name, student number and cse login.

Soft Copies: Gather all the required files in a directory named `a3answers` and submit it electronically by the deadline. To submit electronically, use the following Prism lab command:

```
submit 3401 a3 a3answers
```

Alternatively, you may use web submit (https://webapp.cse.yorku.ca/submit/) and choose the correct course and assignment number to upload your files.

Question 1: Prolog Programming (60 Points)

In this assignment you are going to implement a solver to a path finding problem using three different search algorithms: A*, A* with cycle-checking and IDA*. We are providing you with the generic implementations of these algorithms in Prolog:

1. A* search with path checking (astar.pl).

2. A* search with cycle checking (astarCC.pl).

3. IDA* search with path checking (idastar.pl).

These files are available online in the resources section of this assignment. They use some common code from astarcommon.pl. Some simple examples of search spaces that show how to use these algorithms are also provided in the resources section (simpleSpace.pl, and waterjugs.pl). These files are uploaded with txt extension, and the names are all in lowercase letters. Before working with them, you need to rename them as explained in the appendix.

Your task is to formulate a path finding problem in a grid (based on a variation of the Wumpus world) as a search problem and to run experiments with the algorithms provided. Consider the following grid, which is a variation of the Wumpus world:

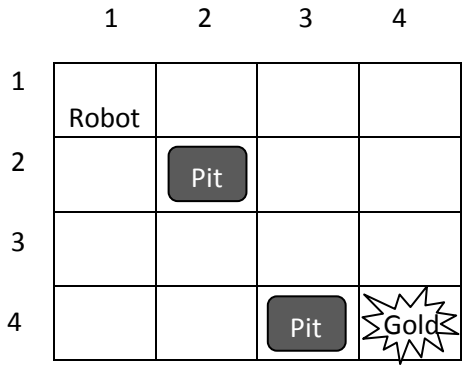

Figure 1: a starting configuration (a)

The Wumpus world is a N×N grid of squares. The world contains a robot, a pile of gold, and any number of pits. Each of these items is in a square of the grid. The robot cannot legally move into a pit (it has to avoid the squares which contain a pit). The robot may start in any square of the grid (unless it contains a pit) and it can only move one position at a time to right, left, up or down. The gold may also be in any square of the grid, except for the squares that contain a pit (the sample configurations in figures 1 and 2 show the gold at (4,4), but in general, this may not be the case). In Figure 1, the robot is located at the starting position (1,1) of the grid and has to move to the position of the gold, which is at (4,4).

The **objective** of the game is to find the shortest path the robot can take from the starting point to the goal, which is the square that contains the gold.

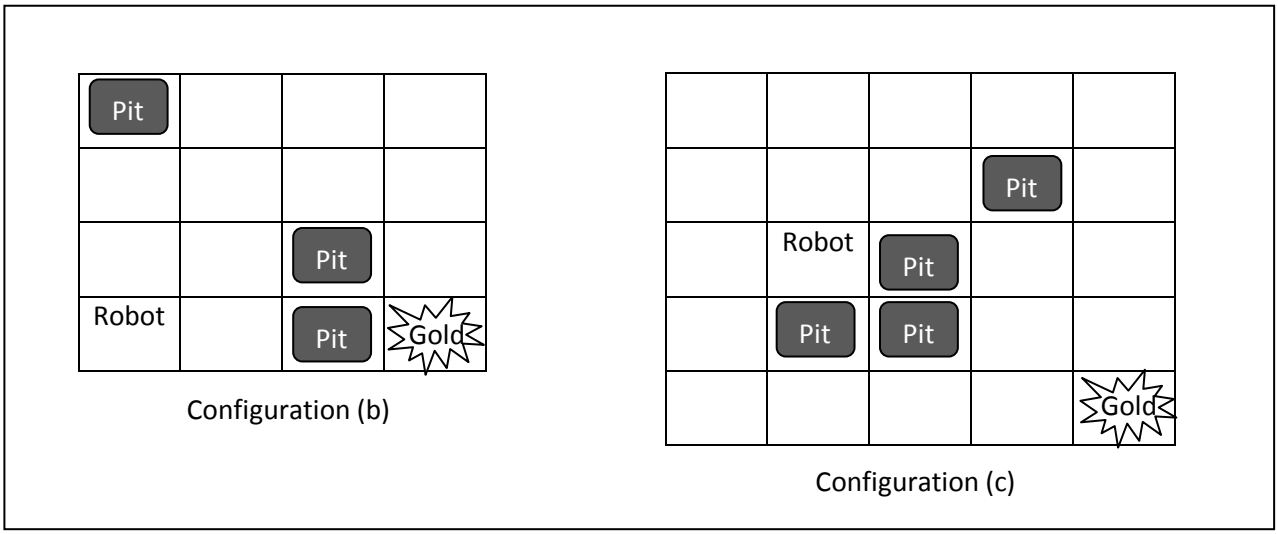

Figure 2: Sample starting configurations

Your implementation of this search problem will include the following:

Q1.a State Representation

Decide how you want to represent the configuration of the described Wumpus world. Then write down the representation for the various world configurations shown in figures 1(a) and 2(b, c). Fill the corresponding predicates in a3.pl, `init(+Name, -State)` where `Name` is the letter in the figure (i.e. a, b, c) and `State` is your representation of the configuration. In addition to the configurations in figures 1 and 2, you may define your own configurations.

Q1.b Goal Predicate

Implement the predicate `goal(+State)` that holds if and only if `State` is a goal state. Note that we require you to implement this predicate, as well as the ones for questions Q1.c, Q1.d and Q1.e, so that it works for any size of grid. This may at first seem more difficult than it is. Some form of counting will do.

Q1.c Successors Predicate

Implement the predicate `successors(+State, -Neighbors)` that holds if and only if `Neighbors` is a list of elements `(Cost, NewState)` where `NewState` is a state reachable from `State` by moving down, left, right, or up and `Cost` is the cost of doing so. Assume that the cost of every move is constant and equal to 1 in this problem (Hint: Neighbors list should not contain squares that contain a pit).

Q1.d Equality Predicate

Implement the predicate `equality(+State1, +State2)` which holds if and only if `State1` and `State2` denote the same state.

Q1.e Heuristic Predicates

In a3.pl the null heuristic `hfn_null/2` is already given. In addition, implement the following three heuristics:

- `hfn_Euclidean(+State, -V)` where `V` is the rounded Euclidean distance from the current state to the goal. The Euclidean distance between points A(X1,Y1) and B(X2,Y2) is calculated as:

$$= \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

  For example, in Figure 1, the value of rounded Euclidean distance between the starting position of the robot (1,1) and the gold (4,4) is 4. (You may use the Prolog predicate `Floor/1` for rounding the number)

- `hfn_Manhattan(+State, -V)` where `V` is the sum of horizontal and vertical distances between the current state and the goal position. The Manhattan distance between points A(X1, Y1) and B(X2,Y2) is calculated as:

$$= abs(X1 - X2) + abs(Y1 - Y2)$$

  For example, in Figure 1, the Manhattan distance between the robot's starting position (1,1) and the gold (4,4) is 6.

- `hfn_Diagonal(+State, -V)` where `V` is the maximum value of the horizontal and vertical distances between the current state and the goal position. The Diagonal distance between points A(X1, Y1) and B(X2,Y2) is calculated as:

$$= \max(abs(X1 - X2), abs(Y1 - Y2))$$

Finally, run the test cases you just implemented using the various heuristics. You can do so by calling the predicates `go/2`, `goCC/2`, `goIDA/2`. For instance, `go(a, hfn_Manhattan)` will try to solve problem (a) using A* search together with the heuristic made up from the sum of horizontal and vertical distances.

You may define any number of configurations and use them as test cases too. In your answers, do not use features of Prolog that have side effects, such as `assert` and `retract`.

Question 2: Discussion (40 Points)

Q2.a Which of the four heuristics (hfn_null, hfn_Euclidean, hfn_Manhattan and hfn_Diagonal) are admissible? For those that are not admissible, provide a counter example.

Q2.b Assume that for the action of moving to the right, the cost has changed from 1 to 0.5, and all the other moves have the same cost (1). Explain how this might affect the admissibility of the above four heuristics. For any which is not admissible, provide a counter example.

Q2.c Assume a variant of the Wumpus world (explained in question 1) that contains obstacles instead of pits. In this case, the robot does not have to avoid such squares and the cost of moving into them is 2, while the cost of moving into other squares is 1. Explain how this might affect the admissibility of the above four heuristics. For any which is not admissible, provide a counter example.

Q2.d Run your implementation on each of the 3 sample configurations using the 3 algorithms and the 4 heuristics, and fill out the table below. If you have run tests with your own sample configurations, you may add them to the table. Based on these results, provide a summary of quality of the heuristics and the search routines.

Configuration	Heuristic Used	A* - Nodes Expanded	A* Cycle Checking Nodes Expanded	IDA* - Nodes Expanded
a	Null			
a	Manhattan			
a	Euclidean Distance			
a	Diagonal Distance			
b	Null			
b	Manhattan			
b	Euclidean Distance			
b	Diagonal Distance			
c	Null			
c	Manhattan			
c	Euclidean Distance			
c	Diagonal Distance			
…	…			

Q2. e Which heuristic dominates the others (i.e. is most informative) ?

Appendix

Uploaded File:	Rename To:
a3.txt	a3.pl
astar.txt	astar.pl
astarcc.txt	astarCC.pl
astarcommon.txt	astarcommon.pl
idastar.txt	idastar.pl
simplespace.txt	simpleSpace.pl
waterjugs.txt	waterjugs.pl