Foundations of Intelligent Systems
(Fall 2015, Prof. Zanibbi)

Project 1: Rolling Die Mazes

Due: Friday, Oct. 9, 2015 (11:59pm)
This project will be completed by groups of two students. **Email both the instructor and TA with the members of your project team ASAP. Only one member of each team should submit the project.**

Submission Instructions: Through MyCourses, submit **two files**: 1) a .zip archive of your program files as .py files and a (very brief) README file explaining how to run your program and interpret the output, and 2) a .pdf file containing the write-up for the project.

Task Description

For this project, you will write an A* search in python for solving rolling-die mazes. The objective is to "roll" a die along its edges through a grid until a goal location is reached. The initial state of the search is given by the die location and orientation. For this project, the die will always start with the 1 facing up ('visible'), 2 facing up/north, and 3 to the right/east: note that the sum of two opposite faces on a die is always 7. Rolling die mazes contain obstacles, as well as restrictions on which numbers may face 'up.' **For this project, the number 6 should never be face up on the die, and the number 1 must be on top of the die when the goal location is reached.**

You will need to implement 3 different heuristics (that are *admissible* and *consistent*) for use with your A* search function. You will create a program `sdmaze`, which takes the name of a puzzle file (text file) from the command line, and then returns the solution produced using each heuristic, along with the metrics described above.

For each maze, your program must produce output as specified below.

- **If a solution is found,** first draw the maze in the starting state, showing the location and orientation of the faces on the die, and then draw an updated maze and die location/orientation after each move in the solution path. Finally, print the number of moves in the solution, the number of nodes put on the frontier queue, and the number of nodes visited (i.e. taken off the front of the queue).
- **If there is no solution,** the search should return a message (e.g. 'No Solution'), the number of nodes put on the frontier queue, and the number of nodes visited.

Hints: Your implementation will be smaller and more flexible if you think carefully about the problem representation that you will use before you start programming. In particular, remember the components of a problem definition as discussed in the text and in class, and think about how this can be used to organize your program. Another good strategy is to start by writing code to read and print mazes, so that you can get started as you think about the structure of the problem, and have code that you can use for testing ready early on.

Mazes

Here are the puzzles that we will use for the project (you will need to copy these out to text files for use in your program). In these diagrams, '.' represents an empty location, '*' represents an obstacle, and 'S' and 'G' represent the starting and goal locations. Recall that the die always starts in the same orientation (see above), and must land on the goal ('G') with 1 on top in order to solve the maze.

```
Puzzle 1

S...G
.....
```

```
Puzzle 2

S....
..***
.*..G
.....
*....
```

```
Puzzle 3

S.....
.****.
.*....
.*....
.....G
```

```
Puzzle 4

.G*.S.
...*..
.*...*
......
...*..
```

```
Puzzle 5

...........S
...........
...........
...........
....*......
...*.......
..*........
...........
....*......
...*.......
..*........
G..........
```

Project Write-Up

In addition to your code, you will submit a 3-4 page .pdf file, `discussion.pdf`, containing the following:

1. **Problem Definition (5%):** provide the search problem definition. Make sure that the state and state space representations are clear.
2. **Heuristics (15%): Mathematical definitions** and descriptions for your heuristic functions, including what they measure, and why they are admissible. Your admissibility argument should be concrete, and technical (i.e. mathematical) - what is it about the mathematical definition of the heuristic and the structure of the problem that guarantees admissibility? *You do not need to define the A* algorithm, as we discussed this in class*.
3. **Performance metrics (15%)**: for each heuristic, run the five puzzles above and report the number of nodes generated (i.e. put on the frontier queue), and the number of nodes visited in a table. Also, provide one or two bar graphs showing the number of nodes generated and visited for the three different heuristics (**Hint:** the python 'matplotlib' library can help here).
4. **Discussion (10%):** Discuss the results, and whether they are consistent with how you expected the results to turn out. **Provide a possible explanation for why the results turned out how they did.** Also provide any other observations that you have about the experiment or the heuristics.

Grading

The project is out of 100 points. The grade for your project will be determined as follows:

Implementation and Performance

- 40% Correctness (each heuristic is admissible, searches return correct paths)
- 10% Efficiency of best heuristic (in terms of states generated; **for full points, you need to generate less than 300 nodes for puzzle 5**)
- 5% Code style (use a research programming style with short informative comments only; indicate the structure of your program, and describe any subtle details of the implementation)

Write-up

- 45% Problem definition (5%), heuristics (15%), performance metrics (15%), and discussion (10%).
