**CSE 537 Project 4 Report**

By Anke Li

04/12/2015

In this assignment, I finished the first three questions. Since I don’t have enough time, I have to give up the last question. This is a tough assignment for me, and I spent a lot of time understanding how prolog works and how to program prolog. For the last question, it should be an easy modification. Theoretically just by changing prolog algorithm to support dynamic destination we could finish it. But unfortunately I couldn’t figure out how to do that before deadline.

**Modeling maze configuration:**

I generated maze.P and mazeastar.P files during initialization of PositionSearchProblem instance just like instructed in specification. Different from the specification, I put three pieces of information into connected(): current cell, next cell and movement to get there. In addition, I indexed the cells by their coordinates. The reason why I need two files is that for A\* search, we need more information than BFS and DFS, so the format of mazeastar.P is slightly different from maze.P, but they are similar in general. I fixed the destination at the end of my code by writing a goal() fact into the files, and this caused my program to be inflexible in question 4.

**Depth First Search Implementation:**

Depth First Search is relatively easy to implement. In my dfs.P file, I first defined the “member” relationship, in order to avoid revisiting cells. Then I created the root of my tree, from the destination, I could explore to the start cell. At last, I defined how to find the next valid cell from previous cell, which is by checking connected() and visited list. Prolog will automatically backtrack if it reached a dead end. In my python function, I announced a new query with the starting cell as the first cell. Prolog will search the tree and return the position information if there is one.

**Breadth First Search Implementation:**

Breadth First Search is much harder than I thought, and I spent too much time on this one. The basic idea is still construct the tree from the ending node (destination). The challenge is how to get all children before search for the next level. For this challenge I wrote a successors() to get all possible children of one path. I used setof() to find next cells that satisfy the constraints (connected with current node and not visited before) and stored them in a list. Then I defined a recursive probing mechanism. I used AllPaths to represent all possible paths. In each iteration, I used successors() to get the possible children, and put them into NewPaths. Now I split all possible paths to remaining Paths and NewPaths. For each child in NewPaths, we have a new node and continue this process. Finally, program will check if there is a path with starting cell as the first node.

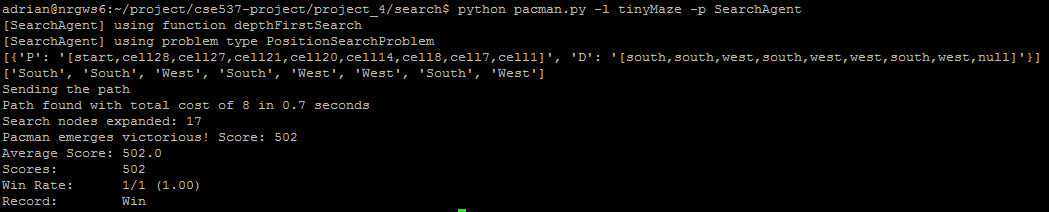
**A\* Search Implementation:**

When I implemented this algorithm, I borrowed some idea from DFS, and added new facts and rules to accommodate the need for heuristic. First I defined a new operator “#” to connect all information I put into mazeastar.P. This is just for convenience and this operator doesn’t have any real functions.

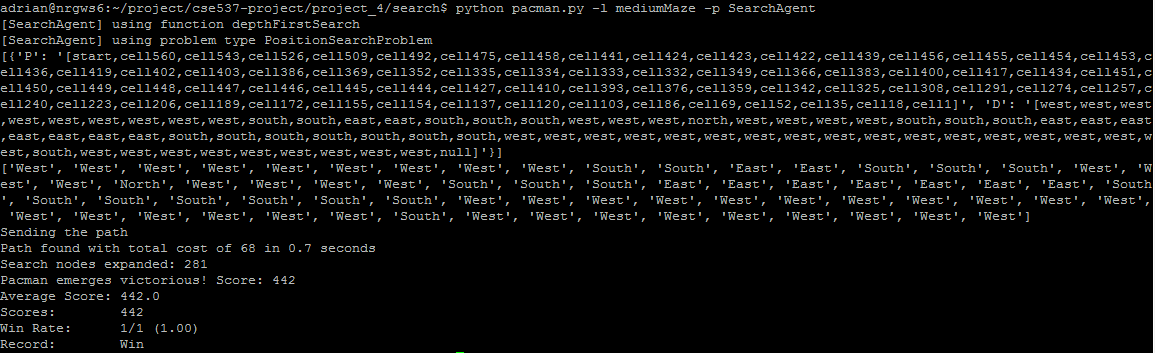
On the basis of BFS algorithm, I used sort to achieve biased selection of children. First, I need to define the characteristics of sort, and how to insert a new path to a list. Then I modified successors() to generate more information, such as the new estimation of one possible path. Then when recursively searching for the result, a new condition is added to the definition: generated children list much be sorted according to their heuristic value in ascending order. This way, prolog will always choose the path with smallest value to probe, thus achieved A\* search.

**Results:**

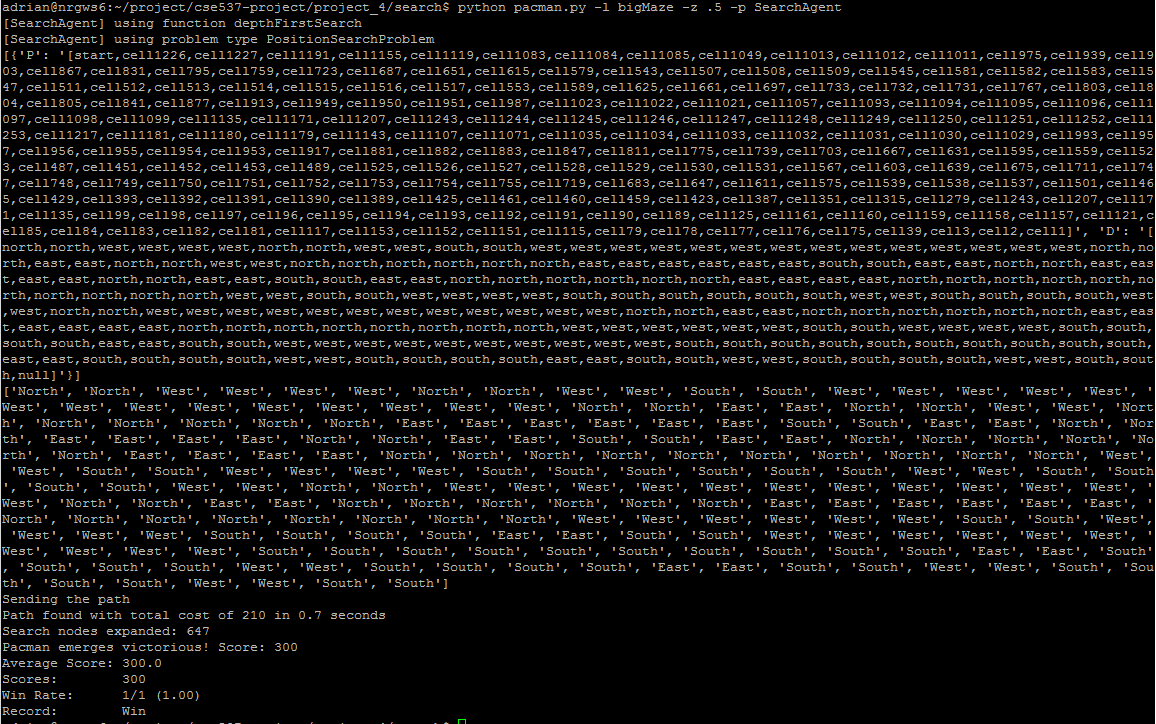
1. python pacman.py -l tinyMaze -p SearchAgent



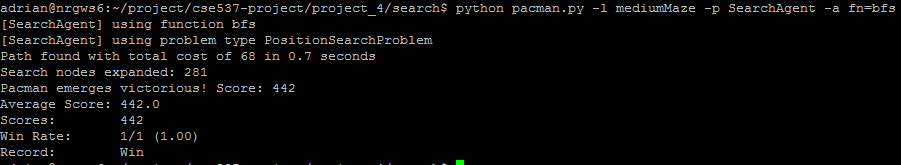
1. python pacman.py -l mediumMaze -p SearchAgent



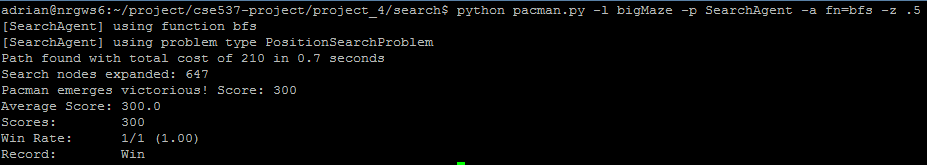
1. python pacman.py -l bigMaze -z .5 -p SearchAgent



1. python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs



1. python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5



1. python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

