Hand-In for Homework 2 – Uninformed and Informed Search Algorithms on Pacman

Problem Description

The uninformed and informed search algorithms are implemented on the framework provided as part of the CIG conference 2016 http://cig16.image.ece.ntua.gr/competitions/ Game AI competition.

I have used Bread First Search for uninformed and A-star algorithm for informed search.

For uninformed search I have used BFS due to the following advantages:

1) Easier to implement
2) BFS will never get trapped exploring the useless path forever.
3) Also, if there is a solution, BFS will definitely find it.

For informed search I have used A-star as it has the advantage that among all shortest-path algorithms using the given heuristic function h(n), A* algorithm expands the fewest number of nodes.

Files

1) Package main.java
   Main.java – Execution begins from here
2) Package main.java.entrants.pacman.neethu.controllers :
   BFSPacMan.java – BFS Implementation
   AStarPacMan.java – A-Star Implementation
3) Package main.java.entrants.pacman.neethu.util
   MazeNode.java – Node structure
   Util.java – create graph and extract path

Instructions for compiling and running

Step 1

1) For executing BFS, you can uncomment the following line in Main.java
   executor.runGameTimed(bfs, ghosts, true);
2) For executing A-star, you can uncomment the following line in Main.java
   executor.runGameTimed(astar, ghosts, true);

   By default, I have used POCommGhosts.
   You can pass randomGhosts object as executor.runGameTimed(astar, randomGhosts, true) if you want to use RandomGhosts

Step 2

If you are using Eclipse IDE, you can run the code by selecting Run-> Run as -> Java Application and execute the Main.java file.

Analysis

Space Complexity

Breadth First Search (BFSPacMan.java)

- ‘q’ is a Queue data structure which stores the open nodes
- ‘visited’ is a Queue data structure which stores the visited nodes
- at depth d there are b^{d+1} unexpanded nodes in the ‘q’

So space complexity is $\boldsymbol{O(b^{d+1})}$ where b is the branching factor and d is the depth at which target is present.

A-Star (AStarPacMan.java, MazeNode.java, Util.java)

Considering the number of nodes in the maze as n. A-star algorithm keeps all the nodes in the memory. Hence $\boldsymbol{O(b^{m})}$ where b is the branching factor and m is the depth at which target is present.

Time Complexity

Breadth First Search (BFSPacMan.java)

Assume (worst case) that there is 1 goal leaf at the RHS at depth d

so BFS will generate = $b + b^2 + \ldots + b^d + b^{d+1} - b$
= $O(b^{d+1})$

Time complexity is $O(b^{d+1})$ where b is the branching factor and d is the depth at which target is present.

A-Star (AStarPacMan.java, MazeNode.java, Util.java)

Time complexity is $\boldsymbol{O(b^{d})}$ in the worst case. But since the heuristics used in admissible (shortest distance), it prunes away many of the b^d nodes that an uninformed search would expand. When using the optimal heuristic, A* will be O(n) in both space and time complexity if we disregard the complexity of the heuristic calculation itself. Here n is the length of the solution path.

Implementation

BFSPacMan.java

The getMove() considers all the available pills in the maze and then takes the closest one and do BFS search from source as current position and target as the closest pill by calling getBFSPath()

getBFSPath(int s, Game game,int d) returns a BFS path from index s to index d. (the BFS path returned is an array of indices). MOVE getMove(Game game, long timeDue) considers each index in the path returned by the getBFSPath() and decides the next move. If the path is empty, getMove() goes on to consider the next target from the list of pills.

BFS algorithm is implemented as below:

A queue ‘q’ maintains a list of nodes to be visited

A queue ‘closed’ maintains a list of nodes already traversed.

Consider the first node added to queue and added it to the list of visited nodes.

Then consider each of its neighbors and add them to ‘q’ if they are not already visited.

This loop ends once the queue is empty or if the target is found.

AStarPacMan.java

The getMove() considers all the available pills in the maze and then takes the closest one and do A-Star search from source as current position and target as the closest pill by calling computePathsAStar ()

computePathsAStar(int s, int t, MOVE lastMoveMade, Game game) returns a A-Star path from index s to index t. (the A-Star path returned is an array of indices). MOVE getMove(Game game, long timeDue) considers each index in the path returned by the computePathsAStar () and decides the next move. If the path is empty, getMove() goes on to consider the next target from the list of pills.

A-Star algorithm is implemented as below:

A graph of current maze is initially created.

A Priority queue ‘open’ maintains a list of nodes to be visited.

The function takes a node from open list and consider all its neighbors and f value for each node is calculated as f =g + h, where cost of each node is initialized to 1.

h is the heuristic and is taken as the shortest distance from the node being considered to the target node.

g value for each child node is updated as initial cost + its parent’s g value.