Hand-In for Homework 3 – Adversarial Algorithm on Pacman

Problem Description

The adversarial 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 alpha beta pruning algorithm

Alpha-beta pruning is a technique to reduce the number of nodes evaluated in the search tree by the minimax algorithm.

The benefit of alpha-beta pruning lies in the fact that branches of the search tree can be eliminated. The search time can in this way be limited to the 'more promising' subtree, and a deeper search can be performed in the same time. The optimization typically reduces the effective branching factor by two compared to simple Minimax, or equivalently doubles the number of nodes that can be searched in a given time.

The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity. As the recursion progresses the "window" becomes smaller. When beta becomes less than alpha, it means that the current position cannot be the result of best play by both players and hence need not be explored further.

Files

1) Package main.java
   Main.java – Execution begins from here
2) Package main.java.entrants.pacman.neethu.controllers :
   AlphaBetaPacMan.java – Alpha beta pruning Implementation

Instructions for compiling and running

Step 1

1) For executing Alpha beta pruning, you can uncomment the following line in Main.java
   executor.runGameTimed(alphabeta, ghosts, true);

   By default, I have used StarterGhost object and the environment is changed to fully observable by updating the 1st parameter to executor object as false in Main.java
   Executor executor = new Executor(false, true);

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

The search tree has a branching factor of 4 by Pacman and 3^4 by Ghosts alternatively. So the space complexity for a depth 4 minimax pacman tree would be = **1 + 4 + (3^4)*4 + 4*(3^4)*4 + (3^4)*4*(3^4)*4**

(there are three possible moves for the ghost).

Time Complexity

The time complexity is of exponential order.

The maximizing player (pacman) has a max branch factor of 4 while the ghosts have a max branch factor of 3^4.

The time complexity is in the order of **O(3^4)^m** as time taken will not be more than (3^4)^m.

Implementation

getMove() starts with the maximizing step for the pacman.

Maximizing step considers all the available actions for the pacman (maximizing agent)

Pacman tries to maximize the score provided by the minimizing agents (all 4 ghosts) and considers the move that maximizes its score. If the value provided by the minimizer is greater than the best already explored option along path to the root for minimizer (beta value), then we ignore that value (pruning) else update alpha as required

Minimizing step recursively gets all combination of moves for the four ghosts. Once each combination is obtained, a copy of the game is made and we advance the game. Then, we take the minimum value of all values provided by the maximizers.

Pruning is applied if the value being considered is less than best already explored option along path to root for the maximizer