Artificial Intelligence
Final Project

Kevin Lin, Kong Huang, Samir Mohamed

December 4, 2015

Contents

Chapter 1

Algorithm Implementations

1.1 Breadth-first Search

Breadth-first search is a simple search algorithm that starts at a root node and explores all neighboring nodes first before moving on to the next level.

1.2 Depth-first Search

Depth-first search is a simple search algorithm that starts at a root node and explores as far as possible along each branch of the tree before backtracking back to the root node.

1.3 Iterative Deepening

Iterative deepening is a search algorithm that runs a depth-first search over and over with increasing depths. This ensures that you never end up exploring along infinite dead-end branches because the length of the path is capped at a certain depth each iteration. It combines the space-efficiency of depth-first search with the completeness of breadth-first search.

1.4 A*

Do A* stuff.

1.5 Hill Climber

The most basic of the optimization algorithms, the hill climbing algorithm generates neighboring solutions until there are no better neighboring solutions. In the case of our Pacman implementation, it looks at all the possible moves around where the player currently is. Afterwards, it evaluates each move and simply picks the one with the best score. This will sometimes result in situations where the algorithm gets stuck in a local maxima.

1.6 Simulated Annealing

Simulated annealing is similar to the hill climbing algorithm with one small difference. If the next solution is not better than the previous solution, generate an acceptance probability and *maybe* move to it depending on that probability's comparison to a random number. This ensures that the algorithm will sometimes elect to keep the worse solution, allowing it to break out of local maxima. Our acceptance probability is based on the current level time, and goes down as the game level timer goes up.

1.7 Evolution Strategy

1.8 Genetic Algorithm

1.9 Alpha-beta Pruning

Alpha-beta pruning is away of finding an optimal solution to the minimax algorithm while *pruning* the subtrees of moves that won't be selected. The algorithm gets its name from two bounds that are passed during the calculation. These bounds limit the solution set based on what's already been seen in the tree. The beta, also referred to as the MinValue in our code, refers to the minimum upper bound of possible solutions. The alpha, also referred to as the MaxValue in our code, refers to the maximum lower bound of the possible solutions. The alpha and beta can be referenced as the best and worst move for Pacman respectively. Our implementation of the algorithm ignores searches in the opposite direction since they are already covered elsewhere in the tree. The MIN and MAX functions call each other to get a good reference alpha and beta value while the main function is used to decide a proper decision on what move to make depending on those values.

1.10 k-Nearest Neighbor

1.11 Perceptron

1.12 ID3 Decision Tree

1.13 Q-learning

Chapter 2

Algorithm Analysis and Comparisons

2.1 Data

2.2 Analysis

Chapter 3

Custom Algorithm

3.1 Idea

3.2 Implementation

3.3 Performance

References

[1] "Hill climbing" *Wikipedia.* Wikimedia Foundation, n.d. Web. 03 May 2015.