Artificial Intelligence

Kevin Lin, Kong Huang, Samir Mohamed

December 4, 2015

Contents

Chapter 1

Algorithm Implementations

1.1 Breadth-first Search

Do breadth-first stuff.

1.2 Depth-first Search

Do depth-first stuff.

1.3 Iterative Deepening

Do iterative deepening stuff.

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.

```
// if it's better take it
if (eval(newState) > currentEval){
    myMove = eachMove;
    currentEval = eval(newState);
}
```

Listing 1.1: SNIPPETS: Hill Climber Decision Making

1.6 Simulated Annealing

Simulated annealing is similar to the hill climbing algorithm with one small difference. After it calculates the neighboring solutions, if the new state has a better score, move to it. However, if the new state has a lower score, 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.

```
// if it's better take it
 if (evalScore > currentEval) {
    myMove = eachMove;
    currentEval = evalScore;
}
// if its not better, well take it anyways according to an acceptance probablity so you can escape local maxima
else if (simulatedAnnealingAcceptanceProbability(currentEval, evalScore, game) < Math.random()){
    myMove = eachMove;
    currentEval = evalScore;
}
```

Listing 1.2: SNIPPETS: Simulated Annealing Decision Making

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.

```
MAIN FUNCTION
    // calculate a move score
    moveScore = alphaBetaMinValue(newState, eachMove, alpha, beta, MAX_DEPTH - 1);
    if (moveScore > bestScore){
        bestScore = moveScore;
```

```
6           bestMove = eachMove;
7       }
8       if (moveScore < beta)
9           alpha = Math.max(alpha, moveScore);
10      else
11          break;
12  MIN
13      double v =  alphaBetaMaxValue(newState, previousMove, AX, BX,↩
            depth - 1);
14      if (v <= A)
15          return alpha;
16      if (v >= B)
17          return beta;
18      vsum += v;
19      A += U - v;
20      B += L - v;
21  MAX
22      value = Math.max(value, alphaBetaMinValue(newState, eachMove,↩
            alpha, beta, depth - 1));
23      if (value < beta)
24          alpha = Math.max(alpha, value);
25      else
26          break;
```

Listing 1.3: SNIPPETS: Alpha-beta Pruning Decision Making

1.10 k-Nearest Neighbor

1.11 Perceptron

1.12 ID3 Decision Tree

1.13 Q-learning

Chapter 2

Algorithm Analysis and Comparisons

Chapter 3

Custom Algorithm

References

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