Lecture 4: Minimax, Alpha-Beta pruning & expecti-minimax

AI For Traditional Games
Prof. Nathan Sturtevant
Winter 2011

Misc Notes

• Probably want a print function for debugging purposes
• Could have collaborated to make me lose the tournament
• Make sure you follow the HW instructions
• Will place my program in the course SVN

vt100 terminal commands

```c
void HopStepGameState::gotoxy(int x, int y)
{
    printf("%c[%d;%df", 27, y, x);
}

void HopStepGameState::clrscr()
{
    printf("%c[27H", 27, 27);
}

void HopStepGameState::topscr()
{
    printf("%c[27H", 27);
}

void HopStepGameState::setcolor(int color, int mode)
{
    printf("%c[%d;%dm", 27, mode, color);
}
```

RPS Competition Summary

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Against Bots</th>
<th>Against Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nathan</td>
<td>4299.5</td>
<td>2781.0</td>
<td>1518.5</td>
</tr>
<tr>
<td>Jeff</td>
<td>1752.5</td>
<td>1080.0</td>
<td>672.5</td>
</tr>
<tr>
<td>Jason</td>
<td>1493.0</td>
<td>1095.0</td>
<td>398.0</td>
</tr>
<tr>
<td>Will</td>
<td>622.5</td>
<td>592.0</td>
<td>30.5</td>
</tr>
<tr>
<td>Chris</td>
<td>361.5</td>
<td>1048.5</td>
<td>-687.0</td>
</tr>
<tr>
<td>Micah</td>
<td>141.5</td>
<td>988.0</td>
<td>-846.5</td>
</tr>
<tr>
<td>Sean</td>
<td>-1324.0</td>
<td>-238.0</td>
<td>-1086.0</td>
</tr>
</tbody>
</table>
"Solving a game"

- Classify nodes as AND/OR
  - AND/OR tree
  - Max player needs any of the children to be a win (OR)
  - Min player needs all of the children to be a win (AND)
    - Win for the max player, that is
- Only works in small games
- Bottom up example from Nim
  - Start with 7 in the stack
  - time: $O(b^d)$ space: $O(b^d)$

Minimax

- Min player and Max player
  - Max player tries to find strategy for maximum score
  - Min player tries to find strategy for minimum score
- Depth-First Search
  - Find minimax value of each state recursively
Example

Minimax Pseudo-Code

Minimax()
  GetMaxVal()

GetMaxVal()
  if (game over) return game value
  currVal ← -∞
  for each successor s in 1… # successors
    ApplyMove(s)
    currVal = max(currVal, GetMinVal())
    UndoMove(s)
  return currVal

Minimax Pseudo-Code (2)

Minimax(depth)
  GetMaxVal(depth)

GetMaxVal(depth)
  if (depth == 0) return CutoffEval();
  currVal ← -∞
  for each successor s in 1… # successors
    ApplyMove(s)
    currVal = max(currVal, GetMinVal(depth-1))
    UndoMove(s)
  return currVal

Minimax

• Analysis
  • O(d) memory
  • O(N) = O(b^d) time

• What if we don’t have time to do the whole tree?
How should we choose our eval?

- Only need an ordering on the preference of leaf values
- But, often normalize values
  - Choose set of useful features & weights
  - \( f_1 \cdot w_1 + f_2 \cdot w_2 + \ldots + f_n \cdot w_n \)
- In chess, just material value of pieces plus a good search will play somewhat reasonable chess

Can we do better?

- Don’t have to search a whole tree to know the value of the tree
  - Simple example:

Alpha Beta Pruning

- alpha is the best score achieved by the max player
  - alpha starts at \(-\infty\)
- beta is the best score achieved by the min player
  - beta starts at \(\infty\)
- If alpha \(\geq\) beta, then we can perform a cutoff
- Lots of resources on the web -- slides aren’t animated
  - Also see Korf textbook
Alpha-beta pruning

- Assuming the game is a perfectly ordered win/loss tree
- Easily show that alpha-beta expands the min nodes
- Node ordering matters!
  - $b^d$ with the wrong ordering
  - $b^{d/2}$ with perfect ordering
  - $b^{3d/4}$ with “average ordering”
- Will discuss move-ordering algorithms next week
- Is $b^{d/2}$ a big deal? Yes -- double our search depth