Announcements

- W1 out and due Monday 4:59pm
- P2 out and due next week Friday 4:59pm

Overview

- Deterministic zero-sum games
  - Minimax
  - Limited depth and evaluation functions
  - Alpha-Beta pruning
- Stochastic games
  - Expectimax
- Non-zero-sum games

Game Playing State-of-the-Art

- **Checkers**: Chinook ended 40-year reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a list of 443,748,401,247 positions. Checkers is now solved!
- **Chess**: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Othello**: Human champions refuse to compete against computers, which are too good.
- **Go**: Human champions are beginning to be challenged by machines, though the best humans still beat the best machines. In go, b > 300, so most programs use pattern knowledge bases to suggest plausible moves, along with aggressive pruning.
- **Pacman**: unknown

GamesCrafters

http://gamescrafters.berkeley.edu/
Dan Garcia.

Game Playing

- Many different kinds of games!
- **Axes**:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Perfect information (can you see the state)?
- Want algorithms for calculating a strategy (policy) which recommends a move in each state
Deterministic Games

- Many possible formalizations, one is:
  - States: S (start at $s_0$)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: $S \times A \rightarrow S$
  - Terminal Test: $S \rightarrow \{t,f\}$
  - Terminal Utilities: $S \times P \rightarrow R$

- Solution for a player is a policy: $S \rightarrow A$

Deterministic Single-Player?

- Deterministic, single player, perfect information:
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik’s cube
  - ... it’s just search!
  - Slight reinterpretation:
    - Each node stores a value: the best outcome it can reach
    - This is the maximal outcome of its children (the max value)
    - Note that we don’t have path sums as before (utilities at end)
    - After search, can pick move that leads to best node
    - Often: not enough time to search till bottom before taking the next action

Adversarial Games

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result

- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Each node has a minimax value: best achievable utility against a rational adversary

Minimax Example

Computing Minimax Values

- Two recursive functions:
  - max-value: maxes the values of successors
  - min-value: mins the values of successors

```python
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)

def max-value(state):
    initialize max = -∞
    for each successor of state:
        compute value(successor)
        update max accordingly
    return max
```

Minimax Properties

- Optimal against a perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - $O(b^m)$

  For chess, $b=35$, $m=100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?
### Tic-tac-toe Game Tree

- **MAX** (0)
- **MIN** (1)
- **MAX** (2)
- **MIN** (3)
- **TERMINAL**

<table>
<thead>
<tr>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

### Speeding Up Game Tree Search

- **Evaluation functions for non-terminal states**
- **Pruning:** not search parts of the tree
  - Alpha-Beta pruning does so without losing accuracy, $O(b^d) \rightarrow O(b^{d/2})$

### Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone

### Why Pacman Can Starve

- He knows his score will go up by eating the dot now
- He knows his score will go up just as much by eating the dot later on
- There are no point-scoring opportunities after eating the dot
- Therefore, waiting seems just as good as eating

### Why Pacman Starves

- He knows his score will go up by eating the dot now (west, east)
- He knows his score will go up just as much by eating the dot later (east, west)
- There are no point-scoring opportunities after eating the dot (within the horizon, two here)
- Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

### Ghosts

- Game tree with 1 and multiple ghosts?
**Evaluation Functions**

- Function which scores non-terminals
- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:
  \[ \text{Eval}(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s) \]
- e.g. \( f_i(s) = (\text{num white queens} - \text{num black queens}) \), etc.

**Iterative Deepening**

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If "1" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.
   …and so on.

Why do we want to do this for multiplayer games?

Note: wrongness of eval functions matters less and less the deeper the search goes

**Speeding Up Game Tree Search**

- Evaluation functions for non-terminal states
- Pruning: not search parts of the tree
  - Alpha-Beta pruning does so without losing accuracy, \( O(b^n) \to O(b^{d/2}) \)

**Minimax Example**

**Pruning**
Alpha-Beta Pruning

- **General configuration**
  - We're computing the MIN-VALUE at \( n \).
  - We're looping over \( n \)'s children.
  - \( n \)'s value estimate is dropping.
  - \( a \) is the best value that MAX can get at any choice point along the current path.
  - If \( n \) becomes worse than \( a \), MAX will avoid it, so can stop considering \( n \)'s other children.
  - Define \( b \) similarly for MIN.

**Alpha-Beta Pruning Example**

- **Alpha-Beta Pseudocode**

<table>
<thead>
<tr>
<th>function ( \text{MAX-VALUE}(\text{state}) ) returns a utility value</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ( \text{TERMINAL-TEST}(\text{state}) ) then return ( \text{UTILITY}(\text{state}) )</td>
</tr>
<tr>
<td>( \alpha \leftarrow -\infty )</td>
</tr>
<tr>
<td>for ( a, \beta ) in ( \text{SUCCESSORS}(\text{state}) ) do ( \beta \leftarrow \text{MAX}(\beta, \text{MIN-VALUE}(\beta)) )</td>
</tr>
<tr>
<td>return ( \beta )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>function ( \text{MAX-VALUE}(\text{state}, \alpha, \beta) ) returns a utility value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputs: ( \text{state} ), current state is game</td>
</tr>
<tr>
<td>( \alpha ), the value of the best alternative for MAX along the path to ( \text{state} )</td>
</tr>
<tr>
<td>( \beta ), the value of the best alternative for MIN along the path to ( \text{state} )</td>
</tr>
<tr>
<td>( \gamma \leftarrow -\infty )</td>
</tr>
<tr>
<td>for ( a, \beta ) in ( \text{SUCCESSORS}(\text{state}) ) do</td>
</tr>
<tr>
<td>( \gamma \leftarrow \text{MAX}(\gamma, \text{MIN-VALUE}(\gamma, \beta)) )</td>
</tr>
<tr>
<td>if ( \gamma \leftarrow \text{MAX}(\alpha, \beta) ) then return ( \beta )</td>
</tr>
<tr>
<td>return ( \beta )</td>
</tr>
</tbody>
</table>

**Alpha-Beta Pruning Properties**

- This pruning has no effect on final result at the root.
- Values of intermediate nodes might be wrong!
- Good child ordering improves effectiveness of pruning.
- With “perfect ordering”:
  - Time complexity drops to \( O(\sqrt{b^m}) \).
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- This is a simple example of metareasoning (computing about what to compute).

**Expectimax Search Trees**

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In pacman, ghosts act randomly
- Can do expectimax search to maximize average score
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate expected utilities
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children
- Later, we’ll learn how to formalize the underlying problem as a Markov Decision Process.
Expectimax Pseudocode

```python
def value(s):
    if s is a max node return maxValue(s)
    if s is an exp node return expValue(s)
    if s is a terminal node return evaluation(s)

def maxValue(s):
    values = [value(s') for s' in successors(s)]
    return max(values)

def expValue(s):
    values = [value(s') for s' in successors(s)]
    weights = [probability(s, s') for s' in successors(s)]
    return expectation(values, weights)
```

Expectimax Quantities

```
3.312 4.5 6.8 13.7
```

Expectimax Pruning?

```
Chance nodes
- Chance nodes are like min nodes, except the outcome is uncertain
- Calculate expected utilities
- Chance nodes average successor values (weighted)
- Each chance node has a probability distribution over its outcomes (called a model)
- For now, assume we’re given the model

Utilities for terminal states
- Static evaluation functions give us limited-depth search
```

Expectimax Search

```
One search ply
```

Expectimax for Pacman

- Notice that we’ve gotten away from thinking that the ghosts are trying to minimize pacman’s score
- Instead, they are now a part of the environment
- Pacman has a belief (distribution) over how they will act
- Quiz: Can we see minimax as a special case of expectimax?
- Quiz: what would pacman’s computation look like if we assumed that the ghosts were doing 1-ply minimax and taking the result 80% of the time, otherwise moving randomly?
- If you take this further, you end up calculating belief distributions over your opponents’ belief distributions over your belief distributions, etc…
  - Can get unmanageable very quickly!

Expectimax for Pacman

```
Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Minimizing Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimax Pacman</td>
<td>Won 5/5, Avg. Score: 493</td>
<td>Won 5/5, Avg. Score: 483</td>
</tr>
<tr>
<td>Expectimax Pacman</td>
<td>Won 1/5, Avg. Score: -303</td>
<td>Won 5/5, Avg. Score: 503</td>
</tr>
</tbody>
</table>
```

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman
Expectimax Utilities

- For minimax, terminal function scale doesn’t matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
- For expectimax, we need magnitudes to be meaningful

Stochastic Two-Player

- E.g. backgammon
- Expectiminimax (!)
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax

\[
\text{if state is a MAX node then}\n\quad \text{return the highest } \text{Expectiminimax Value of Successors}(\text{state})\n\text{if state is a MIN node then}\n\quad \text{return the lowest } \text{Expectiminimax Value of Successors}(\text{state})\n\text{if state is a chance node then}\n\quad \text{return average of } \text{Expectiminimax Value of Successors}(\text{state})\n\]

Stochastic Two-Player

- Dice rolls increase b: 21 possible rolls with 2 dice
  - Backgammon = 20 legal moves
  - Depth 2 = 20 x (21 x 20)^3 = 1.2 x 10^9
- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier...
- TDGammon uses depth-2 search + very good evaluation function + reinforcement learning:
  - World-champion level play
  - 1st AI world champion in any game!

Non-Zero-Sum Utilities

- Similar to minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own utility and propagate (or back up) nodes from children
  - Can give rise to cooperation and competition dynamically...