Announcements

- Assignments
  - Project 0: In glookup – make sure yours registered
  - Project 1: Due 9/14 (Wednesday!)
  - Written 1: Out, Due 9/21
  - Project 2: Up in the next day or so!

- Autograder:
  - You can submit to the autograder multiple times (… but it won’t debug your code)
  - The autograder isn’t perfect, and it is only a lower bound on your score (… though the autograder is quite good, and if your code autogrades as wrong, the autograder is almost always correct)

CS 188: Artificial Intelligence
Fall 2011
Lecture 6: Adversarial Search
9/13/2011
Dan Klein – UC Berkeley
Many slides over the course adapted from either Stuart Russell or Andrew Moore

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 total 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 just beginning to be challenged by machines, though the best humans still beat the best machines. In go, b > 300!
  - Classic programs use pattern knowledge bases, but big recent advances using Monte Carlo (randomized) expansion methods.
  - Pacman: unknown

GamesCrafters

http://gamescrafters.berkeley.edu/

Adversarial Search

Many different kinds of games!

- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - 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 min value)
  - Note that we don’t have path sums as before (utilities at end)
- After search, can pick move that leads to best node

### 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

### Computing Minimax Values

- Two recursive functions:
  - \( \text{max-value} \) maxes the values of successors
  - \( \text{min-value} \) mins the values of successors

#### Code Snippet

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

def max-value(state):
    Initialize max = -\( \infty \)
    For each successor of state:
        Compute value(successor)
        Update max accordingly
    Return max
```

### Minimax Example

#### Tic-tac-toe Game Tree

![Tic-tac-toe Game Tree](image-url)
Minimax Properties

- Optimal against a perfect player. Otherwise?
- Time complexity?
  - \(O(b^m)\)
- Space complexity?
  - \(O(bm)\)
- For chess, \(b = 35\), \(m = 100\)
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?

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
- More plies makes a BIG difference
  - (DEMO: limitedDepth n)
- Example:
  - Suppose we have 100 seconds, can explore 10^6 nodes/sec
  - So can check 1M nodes per move
  - \(nB\) reaches about depth 8 – decent chess program

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!

Evaluation Functions

- Function which scores non-terminals

  - Ideal function: returns the utility of the position
  - In practice: typically weighted linear sum of features:
    \[\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)\]
  - e.g. \(f_i(s)\) = (num white queens – num black queens), etc.

Evaluation for Pacman

\[\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)\]

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!
**Minimax Example**

- **Pruning in Minimax Search**

**Alpha-Beta Pruning**

- **General configuration**
  - We're computing the MIN-VALUE at node $n$.
  - 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**

```plaintext
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return Utility(state)
    $\alpha = -\infty$
    for $a$, $a$ in Successors(state) do $\alpha = \max(\alpha, \text{MIN-VALUE}(a))$
    return $\alpha$

function MIN-VALUE(state, $\alpha$, $\beta$) returns a utility value
    inputs: state, current state in game, $\alpha$, the value of the best alternative for MAX along the path to state
    $\beta$, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return Utility(state)
    $\gamma = +\infty$
    for $a$, $a$ in Successors(state) do $\gamma = \min(\gamma, \text{MAX-VALUE}(a, \alpha, \beta))$
    if $\gamma < \beta$ then return $\gamma$
    $\alpha = \min(\alpha, \gamma)$
    return $\gamma$
```
Alpha-Beta Pruning Properties

- This pruning has **no effect** on final result at the root
- Values of intermediate nodes might be wrong!
  - Important: children of the root may have the wrong value
- Good child ordering improves effectiveness of pruning
- With “perfect ordering”:
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless...
- This is a simple example of **metareasoning** (computing about what to compute)