Topics

• Game playing
• Game trees
  – Minimax
  – Alpha-beta pruning
• Examples
Why study games

• Offer an opportunity to study problems involving \{hostile, adversarial, competing\} agents.
• Fun
• Interesting, hard problems
Game-Playing Agent

Diagram showing the interaction between an agent and its environment through sensors and actuators.
### Types of games

<table>
<thead>
<tr>
<th>Perfect Information</th>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperfect Information</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Types of games

<table>
<thead>
<tr>
<th></th>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Perfect info</strong></td>
<td>chess, checkers, go, othello</td>
<td>backgammon, monopoly</td>
</tr>
<tr>
<td><strong>Imperfect info</strong></td>
<td></td>
<td>bridge, poker, scrabble</td>
</tr>
</tbody>
</table>
Deterministic 2-player games

• 2-person game
• Players alternate moves
• Zero-sum: one player’s loss is the other’s gain
• Perfect information: both players have access to complete information about the state of the game. No information is hidden from either player.
• No chance (e.g., using dice) involved
• Examples: Tic-Tac-Toe, Checkers, Chess, Go, Nim, Othello
• Not: Bridge, Solitaire, Backgammon, ...
Partial Game Tree for Tic-Tac-Toe

\[ f(n) = +1 \text{ if the position is a win for X.} \]

\[ f(n) = -1 \text{ if the position is a win for O.} \]

\[ f(n) = 0 \text{ if the position is a draw.} \]
Perfect Two-Player Game

- Two players **MAX** and **MIN** take turn (with MAX playing first)
- State space
- Initial state
- Successor function
- Terminal test
- **Score function**, that tells whether a terminal state is a win (for MAX), a loss, or a draw
- Perfect knowledge of states, no uncertainty in successor function
How to play a game

• A way to play such a game is to:
  – Consider all the legal moves you can make
  – Compute the new position resulting from each move
  – Evaluate each resulting position and determine which is best
  – Make that move
  – Wait for your opponent to move and repeat

• Key problems are:
  – Representing the “board”
  – Generating all legal next boards
  – Evaluating a position
Optimal Play

This is the optimal play
Minimax procedure

• Apply the evaluation function at each of the leaf nodes
• “Back up” values for each of the non-leaf nodes until a value is computed for the root node
  – At MIN nodes, the backed-up value is the minimum of the values associated with its children.
  – At MAX nodes, the backed up value is the maximum of the values associated with its children.
• Pick the operator associated with the child node whose backed-up value determined the value at the root
Minimax Example
But in general the search tree is too big to make it possible to reach the terminal states!

Examples:
• Checkers: \(\sim 10^{40}\) nodes
• Chess: \(\sim 10^{120}\) nodes
Evaluation function

• **Evaluation function** or **static evaluator** is used to evaluate the “goodness” of a game position.
  – Contrast with heuristic search where the evaluation function was a non-negative estimate of the cost from the start node to a goal and passing through the given node

• The zero-sum assumption allows us to use a single evaluation function to describe the goodness of a board with respect to both players.
  – \( f(n) > 0 \): position \( n \) good for me and bad for you
  – \( f(n) < 0 \): position \( n \) bad for me and good for you
  – \( f(n) \approx 0 \): position \( n \) is a neutral position
  – \( f(n) = +\infty \): win for me
  – \( f(n) = -\infty \): win for you
Evaluation function examples

- Example of an evaluation function for Tic-Tac-Toe:
  \[ f(n) = [\text{# of 3-lengths open for me}] - [\text{# of 3-lengths open for you}] \]
  where a 3-length is a complete row, column, or diagonal

- Alan Turing’s function for chess
  \[ f(n) = \frac{w(n)}{b(n)} \]
  where \( w(n) \) = sum of the point value of white’s pieces and \( b(n) \) = sum of black’s

- Most evaluation functions are specified as a weighted sum of position features:
  \[ f(n) = w_1 \cdot \text{feat}_1(n) + w_2 \cdot \text{feat}_2(n) + \ldots + w_n \cdot \text{feat}_k(n) \]

- Example features for chess are piece count, piece placement, squares controlled, etc.

- Deep Blue has about 6000 features in its evaluation function
Problem spaces for typical games are represented as trees.

Root node represents the current board configuration; player must decide the best single move to make next.

**Static evaluator function** rates a board position. $f(\text{board}) = \text{real number with } f > 0 \text{ “white” (me), } f < 0 \text{ for black (you)}$

Arcs represent the possible legal moves for a player.

If it is **my turn** to move, then the root is labeled a "MAX" node; otherwise it is labeled a "MIN" node, indicating **my opponent's turn**.

Each level of the tree has nodes that are all MAX or all MIN; nodes at level $i$ are of the opposite kind from those at level $i+1$. 
Minimax procedure

- Create start node as a MAX node with current board configuration
- Expand nodes down to some depth (a.k.a. ply) of lookahead in the game
- Apply the evaluation function at each of the leaf nodes
- “Back up” values for each of the non-leaf nodes until a value is computed for the root node
  - At MIN nodes, the backed-up value is the minimum of the values associated with its children.
  - At MAX nodes, the backed up value is the maximum of the values associated with its children.
- Pick the operator associated with the child node whose backed-up value determined the value at the root
Issues

• Choice of the horizon
• Size of memory needed
• Number of nodes examined
Adaptive Search

- Wait for quiescence - hot spots
- Horizon effect
- Extend singular nodes /Secondary search

- Note that the horizon may not then be the same on every path of the tree
Issues

- **Choice of the horizon**
- **Size of memory needed**
- **Number of nodes examined**
Alpha-Beta Procedure

- Generate the game tree to depth $h$ in depth-first manner
- Back-up estimates (alpha and beta values) of the evaluation functions whenever possible
- Prune branches that cannot lead to changing the final decision
Alpha-beta pruning

• We can improve on the performance of the minimax algorithm through **alpha-beta pruning**

• Basic idea: “*If you have an idea that is surely bad, don't take the time to see how truly awful it is.*” -- Pat Winston

• We don’t need to compute the value at this node.

• No matter what it is, it can’t affect the value of the root node.
Alpha-beta pruning

• Traverse the search tree in depth-first order
• At each **MAX** node \( n \), \( \text{alpha}(n) = \text{maximum value found so far} \)
• At each **MIN** node \( n \), \( \text{beta}(n) = \text{minimum value found so far} \)
  – Note: The alpha values start at \(-\infty\) and only increase, while beta values start at \(+\infty\) and only decrease.
• **Beta cutoff**: Given a **MAX** node \( n \), cut off the search below \( n \) (i.e., don’t generate or examine any more of \( n \)’s children) if \( \text{alpha}(n) \geq \text{beta}(i) \) for some **MIN** node ancestor \( i \) of \( n \).
• **Alpha cutoff**: stop searching below **MIN** node \( n \) if \( \text{beta}(n) \leq \text{alpha}(i) \) for some **MAX** node ancestor \( i \) of \( n \).
Alpha-beta example
Alpha-Beta Procedure

• The alpha of a MAX node is a lower bound on the backed-up value
• The beta of a MIN node is an upper bound on the backed-up value
• Update the alpha/beta of the parent of a node N when all search below N has been completed or discontinued
Alpha-Beta Procedure

• The alpha of a MAX node is a lower bound on the backed-up value
• The beta of a MIN node is a higher bound on the backed-up value
• Update the alpha/beta of the parent of a node N when all search below N has been completed or discontinued

• Discontinue the search below a MAX node N if its alpha is $\geq$ beta of a MIN ancestor of N
• Discontinue the search below a MIN node N if its beta is $\leq$ alpha of a MAX ancestor of N
Alpha-Beta Example
Alpha-Beta Example
Effectiveness of alpha-beta

- Alpha-beta is guaranteed to compute the same value for the root node as computed by minimax, with less or equal computation

- **Worst case:** no pruning, examining $b^d$ leaf nodes, where each node has $b$ children and a d-ply search is performed

- **Best case:** examine only $(b)^{(d/2)}$ leaf nodes.
  - Result is you can search twice as deep as minimax!

- **Best case** is when each player’s best move is the first alternative generated

- In Deep Blue, they found empirically that alpha-beta pruning meant that the average branching factor at each node was about 6 instead of about 35!
Some examples…. 
Checkers

© Jonathan Schaeffer
Chinook vs. Tinsley

Name: Marion Tinsley
Profession: Teach mathematics
Hobby: Checkers
Record: Over 42 years loses only 3 (!) games of checkers
Chinook

First computer to win human world championship!

Visit http://www.cs.ualberta.ca/~chinook/ to play a version of Chinook over the Internet.
Chess
Reversi/Othello
Go: And on the Other

Gave Handtalk a 9 stone handicap and still easily beat the program, thereby winning $15,000
Perspective on Games: Con

“Chess is the Drosophila of artificial intelligence. However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophila. We would have some science, but mainly we would have very fast fruit flies.”

John McCarthy
Summary

• Two-players game as a domain where action models are uncertain
• Optimal decision in the worst case
• Game tree
• Evaluation function / backed-up value
• Minimax procedure
• Alpha-beta procedure
Additional Resources

- [Game AI Page](#)
The Game

Rules:
1. Red goes first
2. On their turn, a player must move their piece
3. They must move to a neighboring square, or if their opponent is adjacent to them, with a blank on the far side, they can hop over them
4. The player that makes it to the far side first wins.
Draw the game tree

Rules:
1. Red goes first
2. On their turn, a player must move their piece
3. They must move to a neighboring square, or if their opponent is adjacent to them, with a blank on the far side, they can hop over them
4. The player that makes it to the far side first wins.
Try this for your Game Tree

Rules:
1. Red goes first
2. On their turn, a player must move their piece
3. They must move to a neighboring square, or if their opponent is adjacent to them, with a blank on the far side, they can hop over them
4. The player that makes it to the far side first wins.