Game Playing

An AI Favorite

- structured task
- not initially thought to require large amounts of knowledge
- focus on games of perfect information

Slide CS472 – Game Playing 1

---

Game Playing

**Initial State** is the initial board/position

**Operators** define the set of legal moves from any position

**Terminal Test** determines when the game is over

**Utility Function** gives a numeric outcome for the game

Slide CS472 – Game Playing 2
Game Playing as Search

Slide CS472 – Game Playing 3

Partial Search Tree for Tic-Tac-Toe

Slide CS472 – Game Playing 4
Simplified Minimax Algorithm

1. Expand the entire tree below the root.

2. Evaluate the terminal nodes as wins for the minimizer or maximizer.

3. Select an unlabeled node, $n$, all of whose children have been assigned values. If there is no such node, we’re done — return the value assigned to the root.

4. If $n$ is a minimizer move, assign it a value that is the minimum of the values of its children. If $n$ is a maximizer move, assign it a value that is the maximum of the values of its children. Return to Step 3.
Another Example

Minimax

function MINIMAX-DECISION(game) returns an operator

for each op in OPERATORS[game] do
    VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
end
return the op with the highest VALUE[op]

function MINIMAX-VALUE(state, game) returns a utility value

if TERMINAL-TEST(game)(state) then
    return UTILITY[game](state)
else if MAX is to move in state then
    return the highest MINIMAX-VALUE of SUCCESSORS(state)
else
    return the lowest MINIMAX-VALUE of SUCCESSORS(state)
The Need for Imperfect Decisions

**Problem:** Minimax assumes the program has time to search to the terminal nodes.

**Solution:** Cut off search earlier and apply a heuristic evaluation function to the leaves.

---

Static Evaluation Functions

Minimax depends on the translation of board quality into a single, summarizing number. Difficult. Expensive.

- Add up values of pieces each player has (weighted by importance of piece).
- Isolated pawns are bad.
- How well protected is your king?
- How much maneuverability to you have?
- Do you control the center of the board?
- Strategies change as the game proceeds.

Design Issues of Heuristic Minimax

**Evaluation Function:** What features should we evaluate and how should we use them? An evaluation function should:

1. 
2. 
3. 

Slide CS472 – Game Playing 11

Slide CS472 – Game Playing 12
Linear Evaluation Functions

- $w_1f_1 + w_2f_2 + \ldots + w_nf_n$
- This is what most game playing programs use
- Steps in designing an evaluation function:
  1. Pick informative features
  2. Find the weights that make the program play well

Design Issues of Heuristic Minimax

**Search:** search to a constant depth

Problems:

- Some portions of the game tree may be “hotter” than others. Should search to *quiescence*. Continue along a path as long as one move’s static value stands out (indicating a likely capture).
- *Horizon effect*
- Secondary search. (*singular extension heuristic*)
Improving Minimax — $\alpha - \beta$ pruning

Two More Examples

Slide CS472 – Game Playing 15

Slide CS472 – Game Playing 16
**Algebraic Solution**

Let $g' = e(g)$. Then $c' = \min(-.05, g')$.

The value assigned to the root node $a$ is

$$a' = \max(.03, \min(-.05, g')) = .03$$

because $\min(-.05, g') \leq -0.05 < 0.03$.

The value assigned to $a$ is independent of the value assigned to $g$.

---

**Slide CS472 – Game Playing 17**

---

**A deep $\alpha - \beta$ cutoff**

---

**Slide CS472 – Game Playing 18**
If $m$ is better than $n$ for Player, never get to $n$ in play.
\(\alpha - \beta\) Search

c = search cutoff  
\(\alpha\) = lower bound on Max’s outcome; initially set to \(-\infty\)  
\(\beta\) = upper bound on Min’s outcome; initially set to \(+\infty\)

We’ll call \(\alpha - \beta\) procedure recursively with a narrowing range between \(\alpha\) and \(\beta\).

Maximizing levels may reset \(\alpha\) to a higher value; Minimizing levels may reset \(\beta\) to a lower value.

Slide CS472 – Game Playing 21

\(\alpha - \beta\) Search Algorithm

1. If the limit of search has been reached, compute \(e(n)\) and report the result.

2. Otherwise, if the level is a \textbf{minimizing} level,
   - Until no more children or \(\alpha \geq \beta\),
     - Use \(\alpha - \beta\) search on child with current values of \(\alpha\) and \(\beta\); note the value, \(v\), returned.
     - If \(v < \beta\), reset \(\beta\) to \(v\).
   - Report \(\beta\).
3. Otherwise, the level is a **maximizing** level:

- Until no more children or $\alpha \geq \beta$,
  - Use $\alpha - \beta$ search on child with current values of $\alpha$ and $\beta$; note the value, $v$, returned.
  - If $v > \alpha$, reset $\alpha$ to $v$.
- Report $\alpha$.
Search Space Size Reductions

**Worst Case:** In an ordering where worst options evaluated first, all nodes must be examined.

**Best Case:** If nodes ordered so that the best options are evaluated first, then what?

---

Slide CS472 – Game Playing 25

---

Backgammon – Board

Slide CS472 – Game Playing 26
Backgammon – Rules

- Goal: move all of your pieces off the board before your opponent does.
- White moves counterclockwise toward 0.
- Black moves clockwise toward 25.
- A piece can move to any position except one where there are two or more of the opponent’s pieces.
- If it moves to a position with one opponent piece, that piece is captured and has to start its journey from the beginning.

Slide CS472 – Game Playing 27

Backgammon – Rules

- If you roll doubles you take 4 moves (example: roll 5,5, make moves 5,5,5,5).
- Moves can be made by one or two pieces (in the case of doubles by 1, 2, 3 or 4 pieces)
- And a few other rules that concern bearing off and forced moves.

Slide CS472 – Game Playing 28
White has rolled 6-5 and has 4 legal moves: (5-10,5-11), (5-11,19-24), (5-10,10-16) and (5-11,11-16).

Slide CS472 – Game Playing 29

Game Tree for Backgammon

Slide CS472 – Game Playing 30
State of the Art in Checkers

- 1952: Samuel developed a checkers program that learned its own evaluation function through self play.
- 1992: Chinook (J. Schaeffer) wins the U.S. Open. At the world championship, Marion Tinsley beat Chinook.

State of the Art in Backgammon

- 1980: BKG using one-ply search and lots of luck defeated the human world champion.
- 1992: Tesauro combines Samuel’s learning method with neural networks to develop a new evaluation function, resulting in a program ranked among the top 3 players in the world.
State of the Art in Go

$2,000,000 prize available for first computer program to defeat a top level player.

Slide CS472 – Game Playing 33

<table>
<thead>
<tr>
<th>Year</th>
<th>Skill Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>legal chess</td>
</tr>
<tr>
<td>1200</td>
<td>occasional player</td>
</tr>
<tr>
<td>2000</td>
<td>world-ranked</td>
</tr>
<tr>
<td>2900</td>
<td>Gary Kasparov</td>
</tr>
</tbody>
</table>

**Early 1950’s** Shannon and Turing both had programs that (barely) played legal chess (500 rank).

**1950’s** Alex Bernstein’s system, (500+ε).

**1957** Herb Simon claims that a computer chess program would be world chess champion in 10 years...yeah, right.

Slide CS472 – Game Playing 34

1967 Richard Greenblatt, MIT. First of the modern chess programs, MacHack (1100 ranking).

1968 McCarthy, Michie, Papert bet Levy (rated 2325) that a computer program would beat him within 10 years.


1973 By 1973...Slate: “It had become too painful even to look at Chess 3.6 any more, let alone work on it.”

1973 Chess 4.0: smart plausible-move generator rather than speeding up the search. Improved rapidly when put on faster machines.

1976 Chess 4.5: ranking of 2070.


1980’s Programs depend on search speed rather than knowledge (2300 range).

1993 DEEP THOUGHT: Sophisticated special-purpose computer; \( \alpha - \beta \) search; searches 10 ply; singular extensions; rated about 2600.
1995 DEEP BLUE: searches 14-ply; considers 100–200 billion positions per move.

1997 DEEP BLUE: first match won against world-champion (Kasparov).

Concludes “Search”

- Problem Solving as Search
- Uninformed search: DFS / BFS / Uniform cost search
time / space complexity
size search space: up to approx. 10^{11} nodes
special case: Constraint Satisfaction / CSPs
generic framework: variables & constraints
backtrack search (DFS); propagation (forward-checking / arc-consistency, variable / value ordering
(but incomplete)
• **Informed Search:** use heuristic function guide to goal
  
  *Greedy search*
  
  *A* search / provably optimal
  
  Search space up to approximately $10^{25}$
  
  **Local search** (incomplete)
  
  *Greedy / Hillclimbing*
  
  *Simulated annealing*
  
  *Tabu search*
  
  *Genetic Algorithms / Genetic Programming*
  
  search space $10^{100}$ to $10^{1000}$

Slide CS472 – Game Playing 39

• **Aversary Search / Game Playing**
  
  *minimax*
  
  Up to around $10^{10}$ nodes, 6 — 7 ply in chess.
  
  *alpha-beta pruning*
  
  Up to around $10^{20}$ nodes, 14 ply in chess.
  
  *provably optimal*

Slide CS472 – Game Playing 40
Search and AI

Why such a central role?

A. Basically, because lots of tasks in AI are \textit{intractable}.
   Search is “only” way to handle them.

Many applications of search, in e.g.,
   Learning / Reasoning / Planning / NLU / Vision

Good thing: much recent progress (10^{30} quite feasible;
   sometimes up to 10^{10000}). \textbf{Qualitative difference}
   from only a few years ago!