Why Study Games?

- Many human activities can be modeled as games
  - Negotiations
  - Bidding
  - TCP/IP
  - Military confrontations
  - Pursuit/Evasion
- Games are used to train the mind
  - Human game-playing, animal play-fighting

Why Are Games Good for AI?

- Games typically have concise rules
- Well-defined starting and end points
- Sensing and effecting are simplified
  - Not true for sports games
  - See robocup
- Games are fun!
- Downside: Getting taken seriously (not)
  - See robo search and rescue

Some History of Games in AI

- Computer games have been around almost as long as computers (perhaps longer)
  - Chess: Turing (and others) in the 1950s
  - Checkers: Samuel, 1950s learning program
- Usually start with naïve optimism
- Follow with naïve pessimism
- Simon: Predicted computer chess champ by 1967
- Many, e.g., Kasparov, predicted that a computer would never be champion
Games Today

- Computers perform at champion level
  - Backgammon, Checkers (solved), Chess, Othello
- Computers perform well
  - Bridge, poker
- Computers hold back
  (because it wouldn’t be fun otherwise)
  - Some FPS, tic-tac-toe
- Computers still do badly
  (but recent breakthroughs show promise)
  - Go

“Solved” Games

- A game is solved if an optimal strategy is known
- Strongly solved = solved for all positions
- Weakly solved = solved for some (e.g. starting) positions
- Why bother playing solved games?

Simple Game Setup

- Most commonly, we study games that are:
  - 2 player
  - Alternating
  - Zero-sum
  - Perfect information
- Examples: Checkers, chess, backgammon
- Assumptions can be relaxed at some expense
- Economics studies case where #of agents is very large
  - Individual actions don’t change the dynamics

Zero Sum Games

- Assign values to different outcomes
- Win = 1, Loss = -1
- With zero sum games every gain comes at the other player’s expense
- Sum of both player’s scores must be 0
- Are any games truly zero sum?
Characterizing Games

• Two-player alternating move games are very much like search
  – Initial state
  – Successor function
  – Terminal test
  – Objective function (heuristic function)

• Unlike search
  – Terminal states are often a large set
  – Full search to terminal states usually impossible

Game Trees (abstracted)

Minimax

• Max player tries to maximize his return
• Min player tries to minimize max’s return
• This is optimal for both (assuming zero sum)

\[
\text{minimax}(n_{\text{max}}) = \max_{s \in \text{successors}(n)} \text{minimax}(s)
\]
\[
\text{minimax}(n_{\text{min}}) = \min_{s \in \text{successors}(n)} \text{minimax}(s)
\]
Minimax Values

Minimax in the Real World

- Search trees are too big
- Alternating turns double depth of the search
  - 2 ply = 1 full turn
- Branching factors are too high
  - Chess: 35
  - Go: 361
- Full search from start to end never terminates in non-trivial games

Minimax Properties

- Minimax can be run depth first
  - Time $O(b^m)$
  - Space $O(bm)$
- Assumes that opponent plays optimally
- Based on a worst-case analysis
- What if this is incorrect?

Evaluation Functions

- Like heuristic functions
- Try to estimate value of a node without expanding all the way to termination
- Using evaluation functions
  - Do a depth-limited search
  - Treat evaluation function as if it were terminal
- What's wrong with this?
- How do you pick the depth?
- How do you manage your time?
  - Iterative deepening, quiescence
Desiderata for Evaluation Functions

- Would like to put the same ordering on nodes (even if values aren’t totally right)
- Is this a reasonable thing to ask for?
- What if you have a perfect evaluation function?
- How are evaluation functions made in practice?
  - Buckets
  - Linear combinations
    - Chess pieces (material)
    - Board control (positional, strategic)

Search Control Issues

- Horizon effects
  - Something interesting is just beyond the horizon?
  - How do you know?
- When to generate more nodes?
- If you selectively extend your frontier, how do you decide where?
- If you have a fixed amount of total game time, how do you allocate this?

Pruning

- *The most important search control method is figuring out which nodes you don’t need to expand*

- Use the fact that we are doing a worst-case analysis to our advantage
  - Cut off search at min nodes when max can already force a better outcome (for max)
  - Cut off search at max nodes when max can already force a better outcome (for min)

Alpha-beta pruning
How to prune

- We still do (bounded) DFS
- Expand at least one path to the “bottom”

- If current node is \textit{max} node, and \textit{min} can force a lower value, then prune siblings

- If current node is \textit{min} node, and \textit{max} can force a higher value, then prune siblings

Max node pruning

Implementing alpha-beta

```
max\_value(state, alpha, beta)
if cutoff(state) then return eval(state)
for each s in successors(state) do
  alpha = max(alpha, min\_value(s, alpha, beta))
if alpha ≥ beta then return beta
end
return alpha
```

```
min\_value(state, alpha, beta)
if cutoff(state) then return eval(state)
for each s in successors(state) do
  beta = min(beta, max\_value(s, alpha, beta))
if beta ≤ alpha then return alpha
end
return beta
```

Amazing facts about alpha-beta

- Empirically, alpha-beta has the effect of reducing the branching factor by \textit{square root} for many problems

- Effectively doubles search horizon

- Alpha-beta makes the difference between novice and expert computer players
What About Probabilities?

Max nodes

Chance nodes

P=0.5
P=0.5
P=0.6
P=0.4
P=0.9
P=0.1

Min nodes

Expectiminimax

- n random outcomes per chance node
- \( O(b^m n^m) \) time

\[
\begin{align*}
\text{eminimax}(n_{\text{max}}) &= \max_{s \in \text{successors}(n)} \text{eminimax}(s) \\
\text{eminimax}(n_{\text{min}}) &= \min_{s \in \text{successors}(n)} \text{eminimax}(s) \\
\text{eminimax}(n_{\text{chance}}) &= \sum_{s \in \text{successors}(n)} \text{eminimax}(s)p(s)
\end{align*}
\]

Expectiminimax is nasty

- High branching factor

- Randomness makes evaluation functions difficult
  - Hard to predict many steps into future
  - Values tend to smear together
  - Preserving order is not sufficient

- Pruning is problematic
  - Need to prune based upon bound on expectation
  - Need a priori bounds on the evaluation function

Multiplayer Games

- Things sort-of generalize, but can get complicated

- Maintain vector of possible values for each player at each node

- Might assume that each player acts greedily, but what’s wrong with this?

- Correct treatment requires the full machinery of game theory
Conclusions

- Game tree search is a special kind of search
- Rely heavily on heuristic evaluation functions
- Alpha-beta is a big win
- Most successful players use alpha-beta
- Final thought: Tradeoff between search effort and evaluation function effort
- When is it better to invest in your evaluation function?