Games

CPS 170
Ron Parr
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
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: 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, Chess, Othello
- Computers perform well
  - Bridge
- Computers still do badly
  - Go, Hex
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 number 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 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

Player 1

Player 2
Minimax

- Max player tries to maximize his return
- Min player tries to minimize his return
- This is optimal for both (zero sum)

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

Max nodes

Min nodes

3 2 2

32 2 3

12 2 4 15 2
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?
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
• Search from start never terminates in non-trivial games
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
  – Sometimes 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
  - Max player cuts off search when he knows min player can force a provably bad outcome
  - Min player cuts of search when he knows max can force a provably good (for max) outcome
Alpha-beta pruning

Max nodes

Min nodes

3

2

4

15

2

3

12

2

4

15

2
How to prune

• We still do (bounded) DFS
• Expand at least one path to the “bottom”
• If current node is max node, and min can force a lower value, then prune siblings
• If current node is min node, and max can force a higher value, then prune siblings
Max node pruning

Max nodes

2

4

2

4
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 the return beta
end
return alpha
```

```
min_value(state, alpha, beta)
if cutoff(state) then return eval(state)
for each s in successors(state) do
    \[ beta = \min(\alpha, \max_value(s, \alpha, \beta)) \]
    if \[ beta \leq \alpha \] the return \[ \alpha \]
end
return \[ beta \]
```
Amazing facts about alpha-beta

• Empirically, alpha-beta has the effect of reducing the branching factor by half for many problems
• This effectively doubles the horizon that can be searched
• 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 fns 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 an expectation
  – Need a priori bounds on the evaluation function
Multiplayer Games

- Things sort-of generalize
- We can maintain a vector of possible values for each player at each node
- Assume that each player acts greedily
- What’s wrong with this?
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?