Chapter 6
Section 1 - 4

Adversarial Search
Outline

- Optimal decisions
- $\alpha$-$\beta$ pruning
- Imperfect, real-time decisions
Games vs. search problems

- "Unpredictable" opponent $\Rightarrow$ specifying a move for every possible opponent reply

- Time limits $\Rightarrow$ unlikely to find goal, must approximate
Game tree (2-player, deterministic, turns)
Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value = best achievable payoff against best play
- E.g., 2-ply game:

![Game Tree Diagram]
Minimax algorithm

\[
\text{function Minimax-Decision(state) returns an action}
\]
\[
v \leftarrow \text{Max-Value(state)}
\]
\[
\text{return the action in Successors(state) with value } v
\]

\[
\text{function Max-Value(state) returns a utility value}
\]
\[
\text{if Terminal-Test(state) then return Utility(state)}
\]
\[
v \leftarrow -\infty
\]
\[
\text{for } a, s \text{ in Successors(state) do}
\]
\[
v \leftarrow \text{Max}(v, \text{Min-Value}(s))
\]
\[
\text{return } v
\]

\[
\text{function Min-Value(state) returns a utility value}
\]
\[
\text{if Terminal-Test(state) then return Utility(state)}
\]
\[
v \leftarrow \infty
\]
\[
\text{for } a, s \text{ in Successors(state) do}
\]
\[
v \leftarrow \text{Min}(v, \text{Max-Value}(s))
\]
\[
\text{return } v
\]
Properties of minimax

- **Complete?** Yes (if tree is finite)
- **Optimal?** Yes (against an optimal opponent)
- **Time complexity?** $O(b^m)$
- **Space complexity?** $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games $\rightarrow$ exact solution completely infeasible
α-β pruning example
α-β pruning example

MAX

MIN

3
12
8
2

≥3

≤2

X

X
α-β pruning example
α-β pruning example
α-β pruning example
Properties of α-β

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$ → doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)
Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for max
- If v is worse than α, max will avoid it
  → prune that branch
- Define β similarly for min
The α-β algorithm

function Alpha-Beta-Search(state) returns an action
   inputs: state, current state in game
   \( v \leftarrow \text{Max-Value}(state, -\infty, +\infty) \)
   return the action in Successors(state) with value \( v \)

function Max-Value(state, \( \alpha, \beta \)) returns a utility value
   inputs: state, current state in game
   \( \alpha \), the value of the best alternative for \( \text{Max} \) along the path to state
   \( \beta \), the value of the best alternative for \( \text{Min} \) along the path to state
   if Terminal-Test(state) then return Utility(state)
   \( v \leftarrow -\infty \)
   for \( a, s \) in Successors(state) do
      \( v \leftarrow \text{Max}(v, \text{Min-Value}(s, \alpha, \beta)) \)
      if \( v \geq \beta \) then return \( v \)
      \( \alpha \leftarrow \text{Max}(\alpha, v) \)
   return \( v \)
The α-β algorithm

**function** \( \text{Min-Value}(state, \alpha, \beta) \) **returns** a utility value

**inputs:** \( state \), current state in game

\( \alpha \), the value of the best alternative for \( \text{MAX} \) along the path to \( state \)

\( \beta \), the value of the best alternative for \( \text{MIN} \) along the path to \( state \)

if Termial-Test(\( state \)) then **return** Utility(\( state \))

\( v \leftarrow +\infty \)

for \( a, s \) in Successors(\( state \)) do

\( v \leftarrow \text{Min}(v, \text{Max-Value}(s, \alpha, \beta)) \)

if \( v \leq \alpha \) then **return** \( v \)

\( \beta \leftarrow \text{Min}(\beta, v) \)

**return** \( v \)
Suppose we have 100 secs, explore $10^4$ nodes/sec $\rightarrow 10^6$ nodes per move

Standard approach:

- **cutoff test:**
  e.g., depth limit (perhaps add quiescence search)

- **evaluation function**
  = estimated desirability of position
Evaluation functions

- For chess, typically **linear** weighted sum of features
  \[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

- e.g., \( w_1 = 9 \) with \( f_1(s) = \) (number of white queens) \(-\) (number of black queens), etc.
MinimaxCutoff is identical to MinimaxValue except  
1. Terminal? is replaced by Cutoff?  
2. Utility is replaced by Eval  
3.  

Does it work in practice?  

$$b^m = 10^6, b=35 \rightarrow m=4$$  

4-ply lookahead is a hopeless chess player!  

- 4-ply $$\approx$$ human novice  
- 8-ply $$\approx$$ typical PC, human master  
- 12-ply $$\approx$$ Deep Blue, Kasparov
Deterministic games in practice

- **Checkers**: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.

- **Chess**: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

- **Othello**: Human champions refuse to compete against computers, who are too good.

- **Go**: Human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
Summary

- Games are fun to work on!
- They illustrate several important points about AI
- Perfection is unattainable $\rightarrow$ must approximate
- Good idea to think about what to think about