Today

- Adversarial Search
  - Minimax recap
  - $\alpha$-$\beta$ search
  - Evaluation functions
  - State of the art in game playing
Recall: Game Trees

From current position, unwind game into the future

Recall: Minimax Search

- Find the best current move for MAX (you) assuming MIN (opponent) also chooses its best move.

- Compute for each node $n$:

$$\text{MINIMAX-VALUE}(n) = \begin{cases} 
\text{UTILITY}(n) & \text{if } n \text{ is a terminal} \\
\max_{s \in \text{succ}(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\
\min_{s \in \text{succ}(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node}
\end{cases}$$
Example (4 ply)

Which move to choose?
Choose this move
### Minimax Algorithm

```plaintext
function Minimax-Decision(state) returns an action
    v ← Max-Value(state)
    return the action in Successors(state) with value v

function Max-Value(state) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    v ← -∞
    for a, s in Successors(state) do
        v ← Max(v, Min-Value(s))
    return v

function Min-Value(state) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    v ← ∞
    for a, s in Successors(state) do
        v ← Min(v, Max-Value(s))
    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)
Good enough?

- **Chess:**
  - branching factor \( b \approx 35 \)
  - game length \( m \approx 100 \)
  - search space \( b^m \approx 35^{100} \approx 10^{154} \)

- **The Universe:**
  - number of atoms \( \approx 10^{78} \)
  - age \( \approx 10^{21} \) milliseconds

*Can we search more efficiently?*

---

**Two-Ply Game Tree**

```
\begin{align*}
\text{MAX} & : A_1 & A_2 & A_3 \\
\text{MIN} & : A_{11} & A_{12} & A_{13} & A_{21} & A_{22} & A_{23} & A_{31} & A_{32} & A_{33} \\
 & 3 & 12 & 8 & 2 & 4 & 6 & 14 & 5 & 2
\end{align*}
```
Pruning trees

Minimax algorithm explores depth-first

At MIN node:
Current best max value $\alpha = 3 > 2$

No need to look at these nodes!! (these nodes can only decrease MIN value from 2)
Pruning trees

MIN

3
12
8
2
14

MAX

≥3

MIN

≤3

≤2

X
X

≤14

Pruning trees

MAX

≥3

MIN

≤3

≤2

X
X

≤14

≤5
Pruning trees

One more example
One more example

At MAX node:
Current best MIN value $\beta = 10 < 15$

No need to look at these nodes!! (these nodes can only increase MAX value from 15)

This form of tree pruning is known as alpha-beta pruning

alpha = highest (best) value for MAX along current path from root

beta = lowest (best) value for MIN along current path from root
The α-β algorithm
(minimax with four lines of added code)

function **ALPHA-BETA-SEARCH**(*state*) returns an action
inputs: *state*, current state in game

oncé = MAX-VALUE(*state*, −∞, +∞)
return the action in **SUCCESSORS**(*state*) with value *oncé*

function **MAX-VALUE**(*state*, α, β) returns a utility value
inputs: *state*, current state in game

α, the value of the best alternative for **MAX** along the path to *state*
β, the value of the best alternative for **MIN** along the path to *state*

if TERMINAL-TEST(*state*) then return **UTILITY**(*state*)
oncé ← −∞
for *a*, *s* in **SUCCESSORS**(*state*) do
vé ← MAX(vé, MIN-VALUE(*s*, α, β))
if vé ≥ β then return vé
α ← MAX(α, vé)
return vé

The α-β algorithm (cont.)

function **MIN-VALUE**(*state*, α, β) returns a utility value
inputs: *state*, current state in game

α, the value of the best alternative for **MAX** along the path to *state*
β, the value of the best alternative for **MIN** along the path to *state*

if TERMINAL-TEST(*state*) then return **UTILITY**(*state*)
vé ← +∞
for *a*, *s* in **SUCCESSORS**(*state*) do
vé ← MIN(vé, MAX-VALUE(*s*, α, β))
if vé ≤ α then return vé
β ← MIN(β, vé)
return vé
Properties of $\alpha$-$\beta$

- Pruning does not affect final result
- Good *move ordering* improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  \[ \Rightarrow \text{allows us to search deeper} - \text{doubles depth of search} \]
- $\alpha$-$\beta$ search is a simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*)

Good enough?

- **Chess:**
  - branching factor $b \approx 35$
  - game length $m \approx 100$
  - $\alpha$-$\beta$ search space $b^{m/2} \approx 35^{50} \approx 10^{77}$

- **The Universe:**
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{21}$ milliseconds
Transposition Tables

• Game trees contain repeated states

• In chess, e.g., the game tree may have $35^{100}$ nodes, but there are only $10^{40}$ different board positions

• Similar to explored set in graph-search, maintain a transposition table
  ➢ Got its name from the fact that the same state is reached by a transposition of moves.

• $10^{40}$ is still huge!

Can we do better?

• Strategies:
  search to a fixed depth (cut off search)  
  iterative deepening (most common)
Evaluation Function

- When search space is too large, create game tree up to a certain depth only.
- Art is to estimate utilities of positions that are not terminal states.
- Example of simple evaluation criteria in chess:
  - Material worth: pawn=1, knight =3, rook=5, queen=9.
  - Other: king safety, good pawn structure

\[
\text{eval}(s) = w_1 \times \text{material}(s) + w_2 \times \text{mobility}(s) + w_3 \times \text{king safety}(s) + w_4 \times \text{center control}(s) + \ldots
\]
Cutting off search

- Does it work in practice?
  Suppose $b^m = 10^6$ and $b=35 \Rightarrow m=4$

- 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

Game Playing State-of-the-Art

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. Checkers is now solved!
Game Playing State-of-the-Art

- **Chess**: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million positions per second, used very sophisticated evaluation function and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

![Chess Match](image1.png)

Game Playing State-of-the-Art

- **Othello**: Human champions refuse to play against computers because software is too good

- **Go**: Human champions refuse to play against computers because software is too bad.
  - In Go, \( b > 300 \), so need pattern databases and Monte Carlo search (UCT)
  - Human champions are now beginning to be challenged by machines.

- **Pacman**: The reigning champion is <your CSE 473 program here>
Next Time

• Rolling the dice
• Expectiminimax search

• To Do: Project #1 (due this Thursday!)

Exercise:
Prune this tree!
No, because \(-29 > -37\) and other children of min can only lower min's value of \(-37\) (because of the min operation)
Another pruning opportunity!

Pruning can eliminate entire subtrees!
-43 > (-46)
No need to look at this subtree!