Game Playing State-of-the-Art

- Chess: 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- Go: Human champions are now starting to be challenged by machines, though the best humans still beat the best machines. In Go, b > 300! Classic programs use pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods.
- Pacman

Video of Demo Mystery Pacman

Adversarial Games
### Types of Games

- Many different kinds of games!
- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?
- Want algorithms for calculating a strategy (policy) which recommends a move from each state

### Deterministic Games

- Many possible formalizations, one is:
  - States: S (start at \( s_0 \))
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: \( S \times A \rightarrow S \)
  - Terminal Test: \( S \rightarrow \{t,f\} \)
  - Terminal Utilities: \( S \times P \rightarrow R \)
- Solution for a player is a policy: \( S \rightarrow A \)

### Zero-Sum Games

- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

### Adversarial Search
### Single-Agent Trees

- Non-Terminal States: Value of a State
- Terminal States: Value of a State

The best achievable outcome (utility) from that state

### Adversarial Game Trees

- States Under Agent’s Control: Minimax Values
- States Under Opponent’s Control: Minimax Values

Terminal States: $V(s) = \text{known}$
Deterministic, zero-sum games:
- Tic-tac-toe, chess, checkers
- One player maximizes result
- The other minimizes result

Minimax search:
- A state-space search tree
- Players alternate turns
- Compute each node’s minimax value: the best achievable utility against a rational (optimal) adversary

Minimax Implementation

```python
def max_value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min_value(successor))
    return v

def min_value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max_value(successor))
    return v
```

Minimax Implementation (Dispatch)

```python
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max_value(state)
    if the next agent is MIN: return min_value(state)

def max_value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def min_value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```
Minimax Example

Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?

Minimax Properties

Video of Demo Min vs. Exp (Min)
Problem: In realistic games, cannot search to leaves!

Solution: Depth-limited search
- Instead, search only to a limited depth in the tree
- Replace terminal utilities with an evaluation function for non-terminal positions

Example:
- Suppose we have 100 seconds, can explore 10K nodes/sec
- So can check 1M nodes per move
- α-β reaches about depth 8 – decent chess program

Guarantee of optimal play is gone

More plies makes a BIG difference
- Ply = turn taken by one player
Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search
- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:
  \[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]
- e.g. \( f_1(s) = \text{(num white queens – num black queens)} \), etc.
Evaluation for Pacman

Video of Demo Thrashing (d=2)

Why Pacman Starves

- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

Videos

- Video 1: Thrashing
- Video 2: Cooperation (1)
- Video 3: Cooperation (2)
**Game Tree Pruning**

- **Minimax Example**
  - Alpha-Beta Pruning
  - General configuration (MIN version)
    - We’re computing the MIN-VALUE at some node $n$
    - We’re looping over $n$’s children
    - $n$’s estimate of the children’s min is dropping
    - Who cares about $n$’s value? MAX
    - Let $a$ be the best value that MAX can get at any choice point along the current path from the root
    - If $n$ becomes worse than $a$, MAX will avoid it, so we can stop considering $n$’s other children (it's already bad enough that it won’t be played)
  - MAX version is symmetric
Alpha-Beta Implementation

```
def min_value(state, a, b):
    initialize v = +\infty
    for each successor of state:
        v = min(v, value(successor, a, b))
    if v ≤ a return v
    a = max(a, v)
    return v
```

```
def max_value(state, a, b):
    initialize v = -\infty
    for each successor of state:
        v = max(v, value(successor, a, b))
    if v ≥ b return v
    b = min(b, v)
    return v
```

**α**: MAX's best option on path to root
**β**: MIN's best option on path to root

---

Alpha-Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naive version won’t let you do action selection
- Good child ordering improves effectiveness of pruning
- With “perfect ordering”:
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless...
- This is a simple example of metareasoning (computing about what to compute)

---

Alpha-Beta Quiz

Alpha-Beta Quiz 2