Adversarial Games

- 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
  - Each node has a \textit{minimax value}: best achievable utility against a rational adversary

Minimax values: computed recursively

Terminal values: part of the game
Computing Minimax Values

- Two recursive functions:
  - `max-value` maxes the values of successors
  - `min-value` mins the values of successors

```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)
```

```python
def max-value(state):
    initialize max = \(-\infty\)
    for each successor of state:
        compute value(successor)
        update max accordingly
    return max
```

Recap: Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone
- Replanning agents:
  - Search to choose next action
  - Replan each new turn in response to new state

```
\[ \text{max} \]
\[ \text{min} \]
\[ \text{min} \]
\[ \text{max} \]
\[ -1 \]  \[ -2 \]  \[ 4 \]  \[ 9 \]
```

```python
? ? ? ?
```
Evaluation Functions

- Function which scores non-terminals

- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:

\[ \text{Eval}(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s) \]

- e.g. \( f_1(s) = \) (num white queens – num black queens), etc.

Evaluation for Pacman

\[ \text{Eval}(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s) \]
Why Pacman Starves

- 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!

Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.
   ....and so on.

Why do we want to do this for multiplayer games?

Note: wrongness of eval functions matters less and less the deeper the search goes!
Minimax Example

Pruning in Minimax Search
Alpha-Beta Pruning

- General configuration
  - We’re computing the MIN-VALUE at \( n \)
  - We’re looping over \( n \)’s children
  - \( n \)’s value estimate is dropping
  - \( a \) is the best value that MAX can get at any choice point along the current path
  - If \( n \) becomes worse than \( a \), MAX will avoid it, so can stop considering \( n \)’s other children
  - Define \( b \) similarly for MIN

---

Alpha-Beta Pruning Example

- a is MAX’s best alternative here or above
- b is MIN’s best alternative here or above
Alpha-Beta Pruning Example

Starting a/b

Raising a

Lowering b

Raising a

a is MAX's best alternative here or above
b is MIN's best alternative here or above

Alpha-Beta Pseudocode

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 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)
    v ← −∞
    for a, s in Successors(state) do
        v ← Max(v, Min-Value(s, α, β))
        if v ≥ β then return v
        α ← Max(α, v)
    return v
Alpha-Beta Pruning Properties

- This pruning has **no effect** on final result at the root
- Values of intermediate nodes might be wrong!
  - Important: children of the root may have the wrong value
- Good child ordering improves effectiveness of pruning
- With “perfect ordering”:
  - Time complexity drops to $O(b^{n/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- This is a simple example of **metareasoning** (computing about what to compute)

Expectimax Search Trees

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In pacman, the ghosts act randomly
- Can do **expectimax search**
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate **expected utilities**
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children
- Later, we’ll learn how to formalize the underlying problem as a **Markov Decision Process**

[DEMO: minVsExp]
Maximum Expected Utility

- Why should we average utilities? Why not minimax?

- Principle of maximum expected utility: an agent should choose the action which maximizes its expected utility, given its knowledge.

- General principle for decision making
- Often taken as the definition of rationality
- We’ll see this idea over and over in this course!

- Let’s decompress this definition…

Reminder: Probabilities

- A random variable represents an event whose outcome is unknown
- A probability distribution is an assignment of weights to outcomes

- Example: traffic on freeway?
  - Random variable: $T$ = whether there's traffic
  - Outcomes: $T$ in {none, light, heavy}
  - Distribution: $P(T=\text{none}) = 0.25$, $P(T=\text{light}) = 0.55$, $P(T=\text{heavy}) = 0.20$

- Some laws of probability (more later):
  - Probabilities are always non-negative
  - Probabilities over all possible outcomes sum to one

- As we get more evidence, probabilities may change:
  - $P(T=\text{heavy}) = 0.20$, $P(T=\text{heavy} \mid \text{Hour=8am}) = 0.60$
  - We’ll talk about methods for reasoning and updating probabilities later
What are Probabilities?

- **Objectivist / frequentist answer:**
  - Averages over repeated *experiments*
  - E.g. empirically estimating $P(\text{rain})$ from historical observation
  - Assertion about how future experiments will go (in the limit)
  - New evidence changes the *reference class*
  - Makes one think of *inherently random* events, like rolling dice

- **Subjectivist / Bayesian answer:**
  - Degrees of belief about unobserved variables
  - E.g. an agent's belief that it's raining, given the temperature
  - E.g. pacman's belief that the ghost will turn left, given the state
  - Often learn probabilities from past experiences (more later)
  - New evidence *updates beliefs* (more later)

Uncertainty Everywhere

- **Not just for games of chance!**
  - I’m sick: will I sneeze this minute?
  - Email contains “FREE!”: is it spam?
  - Tooth hurts: have cavity?
  - 60 min enough to get to the airport?
  - Robot rotated wheel three times, how far did it advance?
  - Safe to cross street? (Look both ways!)

- **Sources of uncertainty in random variables:**
  - Inherently random process (dice, etc)
  - Insufficient or weak evidence
  - Ignorance of underlying processes
  - Unmodeled variables
  - The world’s just noisy – it doesn’t behave according to plan!

- **Compare to fuzzy logic, which has degrees of truth, rather than just degrees of belief**
Reminder: Expectations

- We can define function $f(X)$ of a random variable $X$
- The expected value of a function is its average value, weighted by the probability distribution over inputs

- Example: How long to get to the airport?
  - Length of driving time as a function of traffic:
    - $L($none$) = 20$, $L($light$) = 30$, $L($heavy$) = 60$
  - What is my expected driving time?
    - Notation: $E[L(T)]$
    - Remember, $P(T) = \{\text{none: 0.25, light: 0.5, heavy: 0.25}\}$
    - $E[L(T)] = L($none$) \times P($none$) + L($light$) \times P($light$) + L($heavy$) \times P($heavy$)$
    - $E[L(T)] = (20 \times 0.25) + (30 \times 0.5) + (60 \times 0.25) = 35$

Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent’s preferences
- Where do utilities come from?
  - In a game, may be simple ($+1/-1$)
  - Utilities summarize the agent’s goals
  - Theorem: any set of preferences between outcomes can be summarized as a utility function (provided the preferences meet certain conditions)
- In general, we hard-wire utilities and let actions emerge (why don’t we let agents decide their own utilities?)
- More on utilities soon…
In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state:

- Model could be a simple uniform distribution (roll a die)
- Model could be sophisticated and require a great deal of computation
- We have a node for every outcome out of our control: opponent or environment
- The model might say that adversarial actions are likely!

For now, assume for any state we magically have a distribution to assign probabilities to opponent actions / environment outcomes.

**Expectimax Pseudocode**

```python
def value(s):
    if s is a max node return maxValue(s)
    if s is an exp node return expValue(s)
    if s is a terminal node return evaluation(s)

def maxValue(s):
    values = [value(s') for s' in successors(s)]
    return max(values)

def expValue(s):
    values = [value(s') for s' in successors(s)]
    weights = [probability(s, s') for s' in successors(s)]
    return expectation(values, weights)
```

Having a probabilistic belief about an agent's action does not mean that agent is flipping any coins!
Expectimax for Pacman

- Notice that we’ve gotten away from thinking that the ghosts are trying to minimize pacman’s score
- Instead, they are now a part of the environment
- Pacman has a belief (distribution) over how they will act
- Quiz: Can we see minimax as a special case of expectimax?
- Quiz: what would pacman’s computation look like if we assumed that the ghosts were doing 1-ply minimax and taking the result 80% of the time, otherwise moving randomly?
- If you take this further, you end up calculating belief distributions over your opponents’ belief distributions over your belief distributions, etc…
  - Can get unmanageable very quickly!

Expectimax for Pacman

Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Minimizing Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimax Pacman</td>
<td>Won 5/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: 493</td>
<td>Avg. Score: 483</td>
</tr>
<tr>
<td>Expectimax Pacman</td>
<td>Won 1/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: -303</td>
<td>Avg. Score: 503</td>
</tr>
</tbody>
</table>

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman
Expectimax Search

- Chance nodes
  - Chance nodes are like min nodes, except the outcome is uncertain
  - Calculate expected utilities
  - Chance nodes average successor values (weighted)
- Each chance node has a probability distribution over its outcomes (called a model)
  - For now, assume we’re given the model
- Utilities for terminal states
  - Static evaluation functions give us limited-depth search

Expectimax Quantities

![Diagram of a decision tree](image)
Expectimax Pruning?

Expectimax Evaluation

- Evaluation functions quickly return an estimate for a node’s true value (which value, expectimax or minimax?)
- For minimax, evaluation function scale doesn’t matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this **insensitivity to monotonic transformations**
- For expectimax, we need **magnitudes** to be meaningful
Mixed Layer Types

- E.g. Backgammon
- Expectiminimax
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax

\[
\text{ExpectiMinimax-Value}(\text{state}):\n\begin{align*}
\text{if } \text{state} & \text{ is a MAX node then} \\
& \text{return the highest } \text{ExpectiMinimax-Value of Successors}(\text{state}) \\
\text{if } \text{state} & \text{ is a MIN node then} \\
& \text{return the lowest } \text{ExpectiMinimax-Value of Successors}(\text{state}) \\
\text{if } \text{state} & \text{ is a chance node then} \\
& \text{return average of } \text{ExpectiMinimax-Value of Successors}(\text{state})
\end{align*}
\]

Stochastic Two-Player

- Dice rolls increase \( b \): 21 possible rolls with 2 dice
  - Backgammon \( \approx 20 \) legal moves
  - Depth \( 2 = 20 \times (21 \times 20)^3 = 1.2 \times 10^9 \)
- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier…
- TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play
- 1st AI world champion in any game!
Multi-Agent Utilities

- Similar to minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own utility
  - Can give rise to cooperation and competition dynamically…