Artificial Intelligence

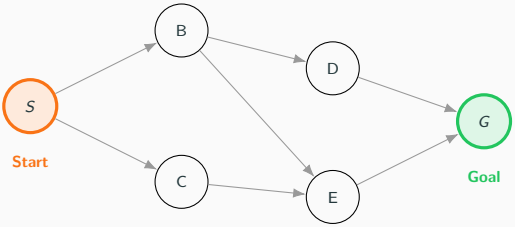
Lecture 5: Stochastic Games & Expectimax

Prof. Antonio Khalil Moretti

Week 6

SCIS 432
Spelman College

Where We've Been: Search as Graph Exploration



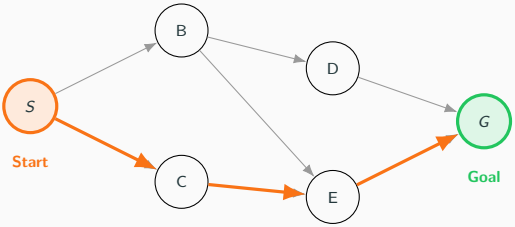
The setup we've used so far:

- **States**: nodes (board positions, map locations, . . .)
- **Actions**: edges connecting states
- **Goal**: a target state we want to reach

Search strategies:

- **Uninformed:** BFS, DFS: no knowledge of the world
- **Informed:** A* and variants: use a *heuristic* $h(n)$ to estimate distance to goal

Where We've Been: Search as Graph Exploration



The setup we've used so far:

- **States**: nodes (board positions, map locations, . . .)
- **Actions**: edges connecting states
- **Goal**: a target state we want to reach

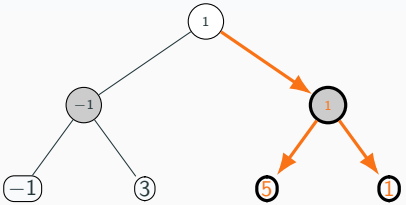
Search strategies:

- **Uninformed:** BFS, DFS: no knowledge of the world
- **Informed:** A* and variants: use a *heuristic* $h(n)$ to estimate distance to goal

Key idea: find the *best path* from S to G

Adversarial Search: Two Players, One Tree

Recap from last lecture



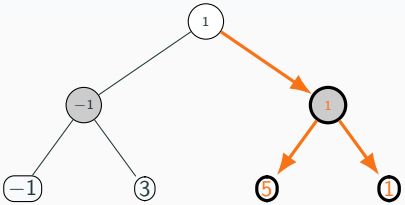
MAX picks the child with the largest value.
MIN picks the child with the smallest value.
Orange path = the move actually taken.

One-line summary of Minimax:

Choose the move that minimises the best score our opponent can achieve.

Adversarial Search: Two Players, One Tree

Recap from last lecture



MAX picks the child with the largest value.
MIN picks the child with the smallest value.
Orange path = the move actually taken.

One-line summary of Minimax:

Choose the move that minimises the best score our opponent can achieve.

Reframed with quantifiers (like in CSPs):

$\exists$ move m $\forall$ opponent response r :

"There **exists** *a move such that*
for **all** *opponent responses,*
the outcome is at least as good as v*"*

- **MAX** = existential ($\exists$): we *choose*
- **MIN** = universal ($\forall$): opponent *could do anything*
- Minimax finds the best *guaranteed* value

Minimax + Alpha-Beta: Great, But...

What alpha-beta gave us:

- Same answer as minimax
- Prunes branches that can't matter
- Best case: $O(b^{m/2})$ — doubles searchable depth

Still exponential in the worst case.

For Chess ($b \approx 35$, $m \approx 100$) we still need evaluation functions and depth limits.

Minimax + Alpha-Beta: Great, But...

What alpha-beta gave us:

- Same answer as minimax
- Prunes branches that can't matter
- Best case: $O(b^{m/2})$ — doubles searchable depth

Still exponential in the worst case.
For Chess ($b \approx 35$, $m \approx 100$) we still need evaluation functions and depth limits.

But there is a bigger assumption we haven't questioned:

Minimax assumes the game is **deterministic**:

- Every outcome is caused by a *player's choice*
- No randomness exists in the game

Is that true for real games?

- Backgammon — *dice*
- Poker — *card draws*
- 2048 — *random tile spawns*

Today we break that assumption.

The Landscape of Game-Playing AI

Where does each algorithm live?

	Fully Observable	Partially Observable
Deterministic	**Single-Agent Search** BFS, DFS, A* Maze, 8-puzzle, Tic Tac Toe, Chess *No opponent, no chance* Minimax / Alpha-Beta (adversarial)	(future topic)
Stochastic	???	???

Grey cells are topics we haven't covered yet. Let's fill them in today.

The Landscape of Game-Playing AI

Where does each algorithm live?

	Fully Observable	Partially Observable
Deterministic	**Single-Agent Search** BFS, DFS, A* Maze, 8-puzzle, Tic Tac Toe, Chess *No opponent, no chance* Minimax / Alpha-Beta (adversarial)	**Game Theory / CFR** Nash Equilibrium Poker, Battleship *Hidden information*
Stochastic	???	???

Games with hidden information need *game theory* (opponent's cards are secret).

The Landscape of Game-Playing AI

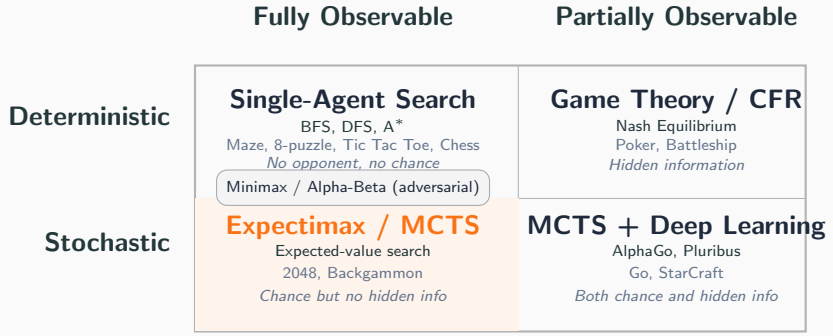
Where does each algorithm live?

	Fully Observable	Partially Observable
Deterministic	**Single-Agent Search** BFS, DFS, A* Maze, 8-puzzle, Tic Tac Toe, Chess *No opponent, no chance* Minimax / Alpha-Beta (adversarial)	**Game Theory / CFR** Nash Equilibrium Poker, Battleship *Hidden information*
Stochastic	**Expectimax / MCTS** Expected-value search 2048, Backgammon *Chance but no hidden info*	???

← **Today's focus:** fully observable games that include *chance*.

The Landscape of Game-Playing AI

Where does each algorithm live?



The hardest case combines both. Modern systems use MCTS guided by neural networks.

The World Isn't Deterministic

Minimax assumed: the only source of uncertainty is the *opponent's choice*.
We handled that by assuming the opponent plays *optimally* (worst case for us).

The World Isn't Deterministic

Minimax assumed: the only source of uncertainty is the *opponent's choice*.
We handled that by assuming the opponent plays *optimally* (worst case for us).

But many real games have a third actor: Nature.

Backgammon
Dice determine which moves are available

Poker
Cards are dealt randomly

2048
A new tile spawns at a random position

New question: how do we make *decisions* when some outcomes are *random*?
Answer: **expected value** and that requires a little bit of probability.

Probability: The Key Terms

Everything we need — nothing more

Sample space Ω: the set of *all* possible outcomes.

Event E: a subset of Ω we care about.

Probability $P(E)$: a number between 0 and 1 saying how likely E is.

Key rule: probabilities over all outcomes must sum to 1:

$$\sum_{e \in \Omega} P(e) = 1$$

(*Something* has to happen.)

Random variable X: a quantity whose value we don't know in advance; it depends on the outcome of a random experiment.

Example: rolling one fair die

$\Omega = \{1, 2, 3, 4, 5, 6\}$

Probability: The Key Terms

Everything we need — nothing more

Sample space Ω: the set of *all* possible outcomes.

Event E: a subset of Ω we care about.

Probability $P(E)$: a number between 0 and 1 saying how likely E is.

Key rule: probabilities over all outcomes must sum to 1:

$$\sum_{e \in \Omega} P(e) = 1$$

(*Something* has to happen.)

Random variable X: a quantity whose value we don't know in advance; it depends on the outcome of a random experiment.

Example: rolling one fair die

$\Omega = \{1, 2, 3, 4, 5, 6\}$

$E =$ "roll is even" $= \{2, 4, 6\}$

$$P(E) = \frac{|E|}{|\Omega|} = \frac{3}{6} = 0.5$$

Check:

$P(1) + P(2) + \cdots + P(6) = \frac{1}{6} \times 6 = 1$ ✓

$X =$ "the number showing" is a random variable.

X can be 1, 2, 3, 4, 5, or 6 (we don't know until we roll).

Expected Value: The Long-Run Average

Definition.

$$E[X] = \sum_i P(x_i) \cdot x_i$$

"Multiply each possible outcome by its probability, then add them all up."

Example: one fair die. Each face has probability $\frac{1}{6}$.

$$E[X] = \tfrac{1}{6}(1) + \tfrac{1}{6}(2) + \tfrac{1}{6}(3) + \tfrac{1}{6}(4) + \tfrac{1}{6}(5) + \tfrac{1}{6}(6)$$

Expected Value: The Long-Run Average

Definition.

$$E[X] = \sum_i P(x_i) \cdot x_i$$

"Multiply each possible outcome by its probability, then add them all up."

Example: one fair die. Each face has probability $\frac{1}{6}$.

$$E[X] = \frac{1}{6}(1+2+3+4+5+6)$$

Expected Value: The Long-Run Average

Definition.

$$E[X] = \sum_i P(x_i) \cdot x_i$$

"Multiply each possible outcome by its probability, then add them all up."

Example: one fair die. Each face has probability $\frac{1}{6}$.

$$E[X] = \frac{1}{6}(1+2+3+4+5+6) = \frac{21}{6}$$

Expected Value: The Long-Run Average

Definition.

$$E[X] = \sum_i P(x_i) \cdot x_i$$

"Multiply each possible outcome by its probability, then add them all up."

Example: one fair die. Each face has probability $\frac{1}{6}$.

$$E[X] = \frac{1}{6}(1+2+3+4+5+6) = \frac{21}{6} = 3.5$$

Expected Value: The Long-Run Average

Definition.

$$E[X] = \sum_i P(x_i) \cdot x_i$$

"Multiply each possible outcome by its probability, then add them all up."

Example: one fair die. Each face has probability $\frac{1}{6}$.

$$E[X] = \frac{1}{6}(1+2+3+4+5+6) = \frac{21}{6} = 3.5$$

Key insight

3.5 is **not** a possible outcome of a single roll. It is the *average* you would get if you rolled the die thousands of times. Expected value tells us what to *expect* over the long run.

Expected Value: Blackjack

A real decision, driven by expected value

The scenario: your hand totals **15**.

You can **hit** (draw one card) or **stand** (keep 15).

If your total goes over 21 you *bust* and lose.

Which cards bust you? Anything worth 7 or more.

	Cards	Count
Safe (2–6)	5 ranks $\times$ 4 suits	20
Bust (7–A)	7,8,9 + 10,J,Q,K + A	32
	Total	**52**

Expected Value: Blackjack

A real decision, driven by expected value

The scenario: your hand totals **15**.

You can **hit** (draw one card) or **stand** (keep 15).

If your total goes over 21 you *bust* and lose.

Which cards bust you? Anything worth 7 or more.

	Cards	Count
Safe (2–6)	5 ranks $\times$ 4 suits	20
Bust (7–A)	7,8,9 + 10,J,Q,K + A	32
	Total	**52**

$$P(\text{safe}) = \frac{20}{52} = \frac{5}{13} \qquad P(\text{bust}) = \frac{32}{52} = \frac{8}{13}$$

Expected Value: Blackjack

A real decision, driven by expected value

The scenario: your hand totals **15**.

You can **hit** (draw one card) or **stand** (keep 15).

If your total goes over 21 you *bust* and lose.

Which cards bust you? Anything worth 7 or more.

	Cards	Count
Safe (2–6)	5 ranks $\times$ 4 suits	20
Bust (7–A)	7,8,9 + 10,J,Q,K + A	32
	Total	**52**

$P(\text{safe}) = \frac{20}{52} = \frac{5}{13}$ $P(\text{bust}) = \frac{32}{52} = \frac{8}{13}$

Simplify: win $= +100$, bust $= -100$.

$EV(\text{hit}) = \frac{5}{13}(+100) + \frac{8}{13}(-100) = -23.1$

$EV(\text{stand}) = 0$ (no risk, no change)

EV says: stand.

Expected Value: Blackjack

A real decision, driven by expected value

The scenario: your hand totals **15**.

You can **hit** (draw one card) or **stand** (keep 15).

If your total goes over 21 you *bust* and lose.

Which cards bust you? Anything worth 7 or more.

	Cards	Count
Safe (2–6)	5 ranks $\times$ 4 suits	20
Bust (7–A)	7,8,9 + 10,J,Q,K + A	32
	Total	**52**

$P(\text{safe}) = \frac{20}{52} = \frac{5}{13}$ $P(\text{bust}) = \frac{32}{52} = \frac{8}{13}$

Simplify: win $= +100$, bust $= -100$.

$EV(\text{hit}) = \frac{5}{13}(+100) + \frac{8}{13}(-100) = -23.1$

$EV(\text{stand}) = 0$ (no risk, no change)

EV says: stand.

Expected Value: Blackjack

A real decision, driven by expected value

The scenario: your hand totals **15**.

You can **hit** (draw one card) or **stand** (keep 15).

If your total goes over 21 you *bust* and lose.

Which cards bust you? Anything worth 7 or more.

	Cards	Count
Safe (2–6)	5 ranks $\times$ 4 suits	20
Bust (7–A)	7,8,9 + 10,J,Q,K + A	32
	Total	**52**

$P(\text{safe}) = \frac{20}{52} = \frac{5}{13}$ $P(\text{bust}) = \frac{32}{52} = \frac{8}{13}$

Simplify: win $= +100$, bust $= -100$.

$EV(\text{hit}) = \frac{5}{13}(+100) + \frac{8}{13}(-100) = -23.1$

$EV(\text{stand}) = 0$ (no risk, no change)

EV says: stand.

Expected Value: Blackjack

A real decision, driven by expected value

The scenario: your hand totals **15**.

You can **hit** (draw one card) or **stand** (keep 15).

If your total goes over 21 you *bust* and lose.

Which cards bust you? Anything worth 7 or more.

	Cards	Count
Safe (2–6)	5 ranks × 4 suits	20
Bust (7–A)	7,8,9 + 10,J,Q,K + A	32
	Total	**52**

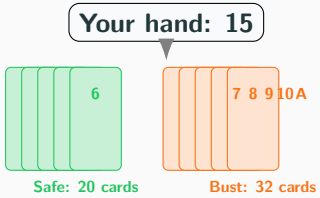
$$P(\text{safe}) = \frac{20}{52} = \frac{5}{13} \qquad P(\text{bust}) = \frac{32}{52} = \frac{8}{13}$$

Simplify: win $= +100$, bust $= -100$.

$$EV(\text{hit}) = \frac{5}{13}(+100) + \frac{8}{13}(-100) = -23.1$$

$EV(\text{stand}) = 0$ (no risk, no change)

EV says: stand.



More bust cards than safe ones → hitting is *negative* EV.

Expected value gave us the answer.

Expected Value: Two Dice

Sum of two fair dice

Two dice $\rightarrow 6 \times 6 = 36$ equally likely pairs.

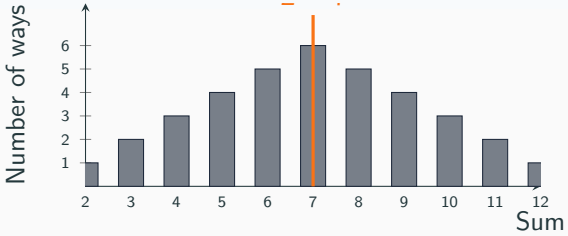
Each cell is the sum of its row and column.

Orange = sum of 7 (6 ways out of 36).

D1 \ D2	1	2	3	4	5	6
1	2	3	4	5	6	**7**
2	3	4	5	6	**7**	8
3	4	5	6	**7**	8	9
4	5	6	**7**	8	9	10
5	6	**7**	8	9	10	11
6	**7**	8	9	10	11	12

$$E[\text{sum}] = \frac{1}{36} \sum_{\text{all 36 cells}} \text{value} = \frac{252}{36} = 7$$

Alternatively: $E[\text{sum}] = \sum_{s=2}^{12} s \cdot P(\text{sum} = s)$



7 is the most likely sum *and* the expected value.

The distribution is **symmetric** around 7.

Again: $E = 7$ is not guaranteed on any single roll.

Why List All 36 Outcomes?

Key insight: Not all sums are equally likely!

- **Sum = 2:** only **one** way

$$(1,1) \qquad P(\text{sum} = 2) = \frac{1}{36}$$

- **Sum = 7:** **six** ways

$$(1,6),(2,5),(3,4),(4,3),(5,2),(6,1) \qquad P(\text{sum} = 7) = \frac{6}{36} = \frac{1}{6}$$

- **Sum = 12:** only **one** way

$$(6,6) \qquad P(\text{sum} = 12) = \frac{1}{36}$$

Different sums $\Rightarrow$ different probabilities.

Two Ways to Compute Expected Value

Both methods give the same answer — but one scales better.

1. **Enumerate the full sample space** (all 36 outcomes):

$$E[\text{sum}] = \frac{1}{36} \sum_{\text{all 36 pairs}} (\text{sum}) = 7$$

2. **Group by sum value** (compress probabilities):

$$E[\text{sum}] = 2 \cdot \frac{1}{36} + 3 \cdot \frac{2}{36} + \cdots + 7 \cdot \frac{6}{36} + \cdots + 12 \cdot \frac{1}{36} = 7$$

Why avoid approach 1 in general?

Sample spaces can grow **exponentially** (e.g., 10 dice $\rightarrow 6^{10} \approx 60$ million outcomes).

Expected Value vs. Median vs. Mode (Three Measures of "Center")

- **Mean (Expected Value):** $E[X] = \sum_i x_i \cdot P(X = x_i)$
  - Weighted average of all possible values
  - **Sensitive to outliers/extreme values**
- **Median:** The middle value when outcomes are ordered
  - 50% of probability mass below, 50% above
  - **Robust to outliers**
- **Mode:** The most likely outcome (highest probability)
  - Can have multiple modes (bimodal, multimodal distributions)

When are they all equal?

- In a **symmetric, unimodal** distribution (e.g., normal dist, symmetric dice sum)
- For the two-dice example: Mean = Median = Mode = 7

Example where they differ:

- Income distribution: Mean > Median (billionaires pull mean up)
- Median income is often more representative than mean income

Expected Value: Two Dice

Sum of two fair dice

Two dice $\rightarrow 6 \times 6 = 36$ equally likely pairs.

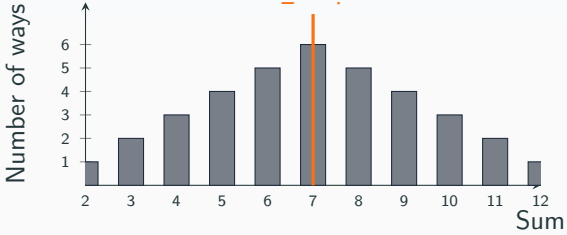
Each cell is the sum of its row and column.

Orange = sum of 7 (6 ways out of 36).

D1 \ D2	1	2	3	4	5	6
1	2	3	4	5	6	**7**
2	3	4	5	6	**7**	8
3	4	5	6	**7**	8	9
4	5	6	**7**	8	9	10
5	6	**7**	8	9	10	11
6	**7**	8	9	10	11	12

Expected value:

$$E[\text{sum}] = \frac{1}{36} \sum_{\text{all 36 cells}} \text{value} = \frac{252}{36} = 7$$



7 is the most likely sum *and* the expected value.

The distribution is **symmetric** around 7.

Again: $E = 7$ is not guaranteed on any single roll.

Why List All 36 Outcomes?

Key: Not all sums are equally likely!

- **Sum = 2:** Only **one** way: $(1,1)$

$$P(\text{sum} = 2) = \frac{1}{36}$$

- **Sum = 7: Six** ways:

$$(1,6),\ (2,5),\ (3,4),$$

$$(4,3),\ (5,2),\ (6,1)$$

$$P(\text{sum} = 7) = \frac{6}{36} = \frac{1}{6}$$

- **Sum = 12:** Only **one** way: $(6,6)$

$$P(\text{sum} = 12) = \frac{1}{36}$$

Two approaches to computing E:

1. **Enumerate sample space** (all 36):

$$E = \frac{1}{36} \sum_{36 \text{ pairs}} \text{sum} = \frac{252}{36} = 7$$

2. **Group by value**:

$$\begin{aligned} E &= \sum_{s=2}^{12} s \cdot P(\text{sum} = s) \\ &= 2 \cdot \tfrac{1}{36} + 3 \cdot \tfrac{2}{36} + \cdots \\ &\quad + 7 \cdot \tfrac{6}{36} + \cdots + 12 \cdot \tfrac{1}{36} \\ &= 7 \end{aligned}$$

Why care? Sample spaces grow **exponentially** (e.g., 10 dice $\to 6^{10} \approx 60$ million outcomes). We often avoid full enumeration when possible!

Expected Value vs. Median vs. Mode

Three measures of "center":

- **Mean (Expected Value):** $E[X] = \sum_i x_i \cdot P(X = x_i)$
  - Weighted average of all possible values
  - **Sensitive to outliers/extreme values**
- **Median:** The middle value when outcomes are ordered
  - 50% of probability mass below, 50% above
  - **Robust to outliers**
- **Mode:** The most likely outcome (highest probability)
  - Can have multiple modes (bimodal, multimodal distributions)

When are they all equal?

- In a **symmetric, unimodal** distribution (e.g., normal distribution, symmetric dice sum)
- For the two-dice example: Mean = Median = Mode = 7

Linearity of Expectation

One of the most powerful properties:

Theorem (Linearity of Expectation)

For any random variables X, Y:

$$E[X + Y] = E[X] + E[Y]$$

For any constant c:

$$E[cX] = c \cdot E[X]$$

More generally:

$$E\left[\sum_{i=1}^{n} a_i X_i\right] = \sum_{i=1}^{n} a_i E[X_i]$$

Example: Two dice sum

$$\begin{aligned} E[\text{Die}_1 + \text{Die}_2] &= E[\text{Die}_1] + E[\text{Die}_2] \\ &= 3.5 + 3.5 \\ &= 7 \end{aligned}$$

No need to enumerate 36 outcomes!

Why this is powerful:

- Break complex RVs into simpler pieces
- Avoid exponential enumeration
- Works even when variables interact

Expected Value Makes Decisions

The principle behind Expectimax

Idea: when outcomes are uncertain, compare actions by their *expected value* and pick the highest one.

Example: a carnival game

You pay $2 to spin a wheel:

- 30% chance $\rightarrow$ win $10
- 70% chance $\rightarrow$ win $0

Should you play?

Expected Value Makes Decisions

The principle behind Expectimax

Idea: when outcomes are uncertain, compare actions by their *expected value* and pick the highest one.

Example: a carnival game

You pay \$2 to spin a wheel:

- 30% chance $\rightarrow$ win \$10
- 70% chance $\rightarrow$ win \$0

Should you play?

$EV(\text{play}) = 0.3(10) + 0.7(0) = \$3$
$EV(\text{skip}) = \$0$

Pay \$2, expected gain \$3 $\Rightarrow$ net $EV = +\$1$.
Yes, play!

Expected Value Makes Decisions

The principle behind Expectimax

Idea: when outcomes are uncertain, compare actions by their *expected value* and pick the highest one.

Example: a carnival game

You pay \$2 to spin a wheel:

- 30% chance $\rightarrow$ win \$10
- 70% chance $\rightarrow$ win \$0

Should you play?

$EV(\text{play}) = 0.3(10) + 0.7(0) = \$3$
$EV(\text{skip}) = \$0$

Pay \$2, expected gain \$3 $\Rightarrow$ net $EV = +\$1$.
Yes, play!

Expected Value Makes Decisions

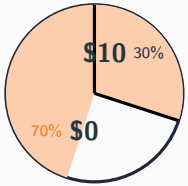
The principle behind Expectimax

Idea: when outcomes are uncertain, compare actions by their *expected value* and pick the highest one.

Example: a carnival game

You pay \$2 to spin a wheel:

- 30% chance → win \$10
- 70% chance → win \$0



Should you play?

$EV(\text{play}) = 0.3(10) + 0.7(0) = \$3$
$EV(\text{skip}) = \$0$

Pay \$2, expected gain \$3 $\Rightarrow$ net $EV = +\$1$.
Yes, play!

Expected Value Makes Decisions

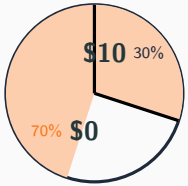
The principle behind Expectimax

Idea: when outcomes are uncertain, compare actions by their *expected value* and pick the highest one.

Example: a carnival game

You pay \$2 to spin a wheel:

- 30% chance → win \$10
- 70% chance → win \$0



Should you play?

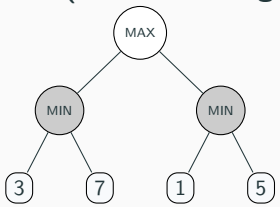
$EV(\text{play}) = 0.3(10) + 0.7(0) = \$3$
$EV(\text{skip}) = \$0$

Pay \$2, expected gain \$3 $\Rightarrow$ net $EV = +\$1$.
Yes, play!

From Minimax to Expectimax

Replace MIN with CHANCE when nature (not an opponent) decides

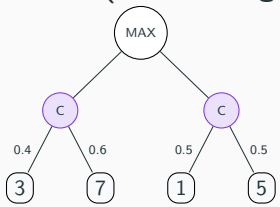
Minimax (deterministic game)



MIN = *opponent* picks the worst outcome for us.

MIN value = min(children).

Expectimax (stochastic game)



CHANCE = *nature* picks randomly.

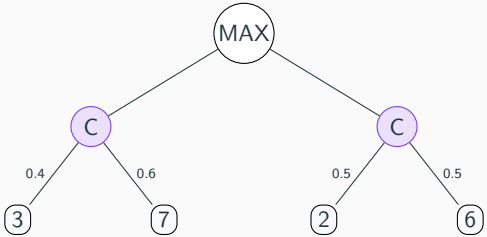
CHANCE value = $\sum P_i \cdot v_i$.

Node type	Who decides	Value
MAX	The agent	max(children)
MIN	The opponent	min(children)
CHANCE	Nature (random)	$\sum P_i \cdot v_i$

A game can have **all three** types (e.g. Backgammon: MAX → CHANCE → MIN → CHANCE).

Expectimax Walk-Through

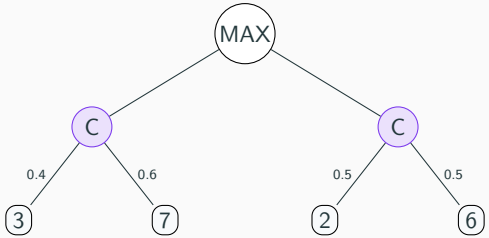
Computing values bottom-up, one step at a time



Leaves already have utility values. We propagate upward.

Expectimax Walk-Through

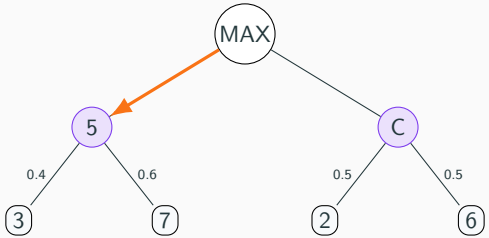
Computing values bottom-up, one step at a time



Left CHANCE node: outcomes are 3 (prob 0.4) and 7 (prob 0.6).

Expectimax Walk-Through

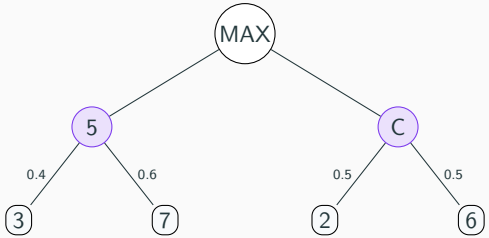
Computing values bottom-up, one step at a time



$$E = 0.4 \times 3 + 0.6 \times 7 = 1.2 + 4.2 = 5$$ ← left CHANCE value

Expectimax Walk-Through

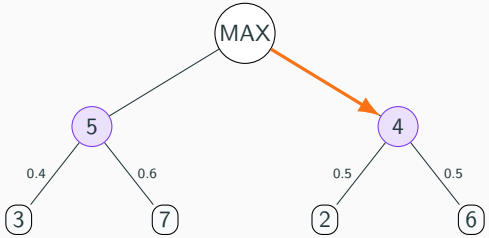
Computing values bottom-up, one step at a time



Right CHANCE node: outcomes are 2 (prob 0.5) and 6 (prob 0.5).

Expectimax Walk-Through

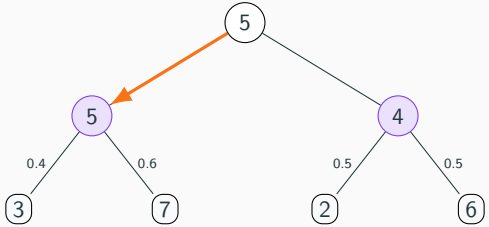
Computing values bottom-up, one step at a time



$$E = 0.5 \times 2 + 0.5 \times 6 = 1 + 3 = 4 \quad \leftarrow \text{right CHANCE value}$$

Expectimax Walk-Through

Computing values bottom-up, one step at a time



MAX picks the larger: $\max(5, 4) = 5 \rightarrow$ **choose the left action**.

Game Rules: “Closest Number”

Setup

Both players hold the same two cards:

$$\{3,\ 8\}$$

You play **first**.

A die is rolled:

- target $= 5$ with probability $\frac{1}{3}$
- target $= 10$ with probability $\frac{2}{3}$

Opponent **sees your card**, then responds adversarially.

- Closest card wins \$6
- Tie $\Rightarrow$ \$0
- You win: $+$\$6, opponent wins: $-$\$6

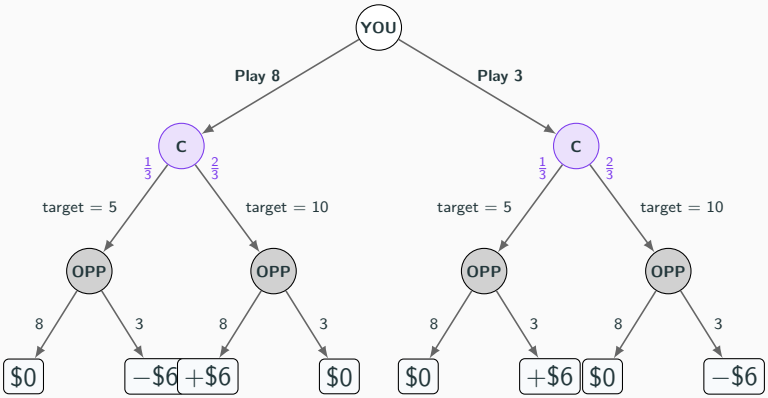
This game has both an adversary and randomness.

Dice roll $\rightarrow$ target number

Two cards: 3 and 8

Games With Both Opponent and Chance

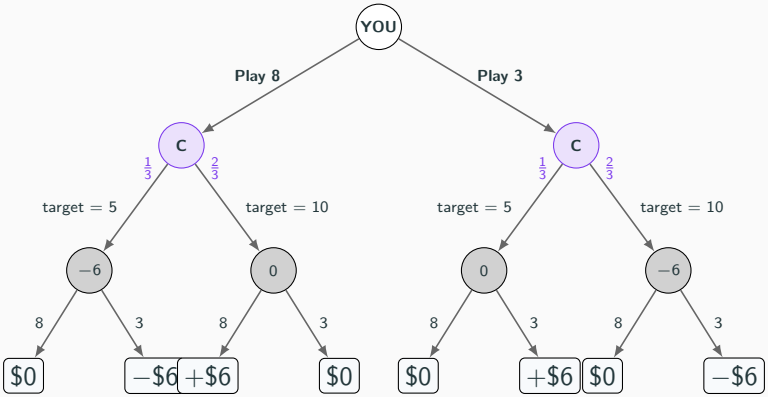
"Closest Number" — you play first, then fate, then your opponent



Both players have a **3** and an **8**. A die is rolled: 1–2 $\Rightarrow$ target $= 5$; 3–6 $\Rightarrow$ target $= 10$. Closest card wins \$6. You play first; opponent sees your card before choosing.

Games With Both Opponent and Chance

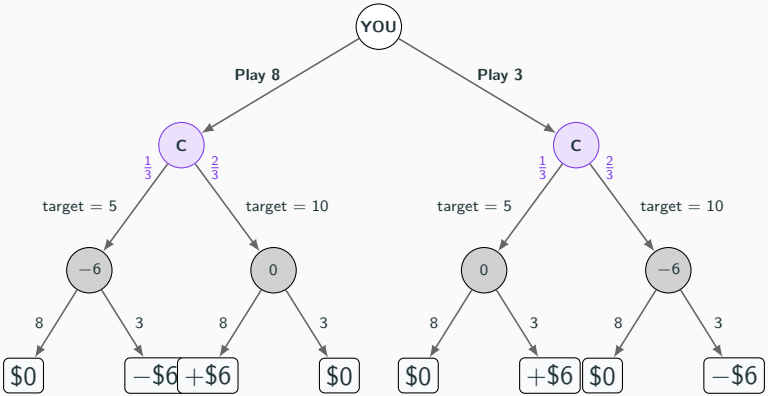
"Closest Number" — you play first, then fate, then your opponent



MIN nodes: opponent picks whichever card *minimises your payoff*. Opponent plays **3** when target is 5 (it's closer), **8** when target is 10.

Games With Both Opponent and Chance

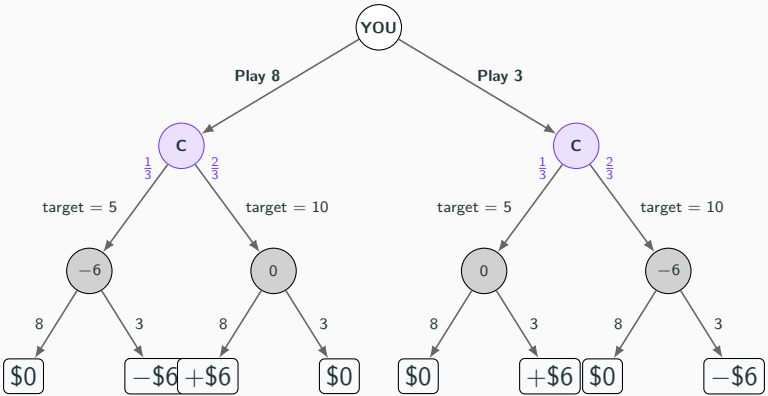
"Closest Number" — you play first, then fate, then your opponent



Leaf values: $+6$ = you win, -6 = opponent wins, 0 = tie (same distance).

Games With Both Opponent and Chance

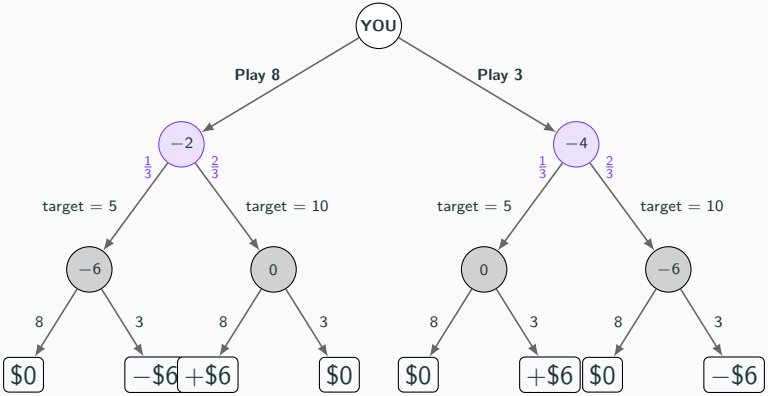
"Closest Number" — you play first, then fate, then your opponent



MIN picks the smaller child at each node. Values propagate up.

Games With Both Opponent and Chance

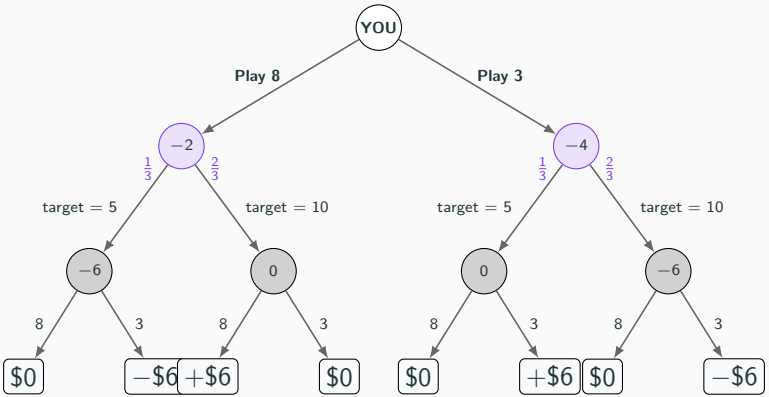
"Closest Number" — you play first, then fate, then your opponent



CHANCE averages with the die probabilities: left $= \frac{1}{3}(-6) + \frac{2}{3}(0) = -2$; right $= \frac{1}{3}(0) + \frac{2}{3}(-6) = -4$.

Games With Both Opponent and Chance

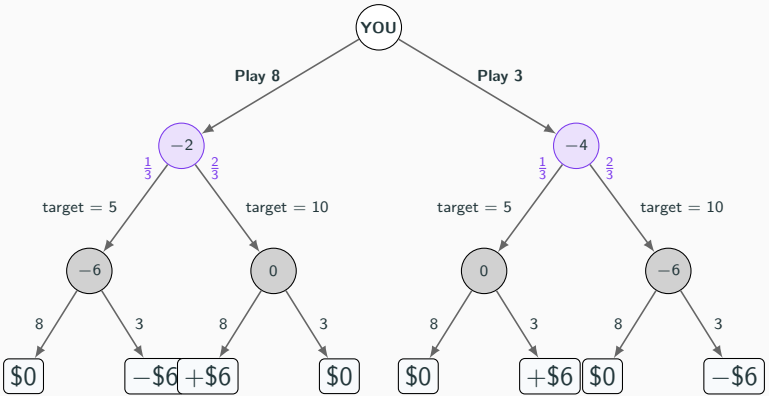
"Closest Number" — you play first, then fate, then your opponent



Playing 8 loses only when target is 5 (unlikely: $\frac{1}{3}$). Playing 3 loses when target is 10 (likely: $\frac{2}{3}$).

Games With Both Opponent and Chance

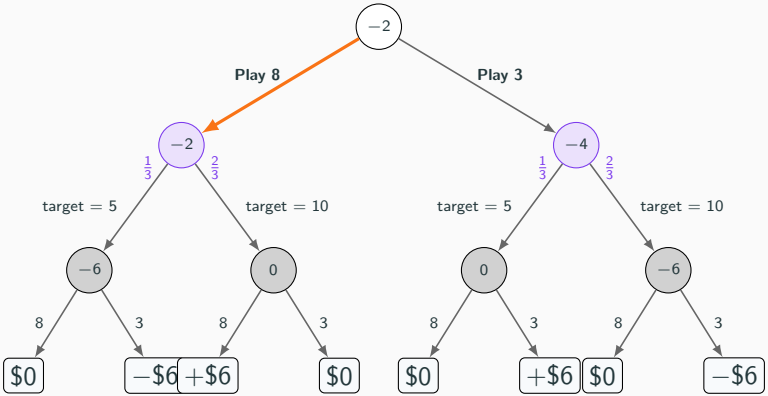
"Closest Number" — you play first, then fate, then your opponent



The die roll — pure chance — is what makes one choice better than the other.

Games With Both Opponent and Chance

"Closest Number" — you play first, then fate, then your opponent



MAX picks $\max(-2, -4) = -2 \rightarrow$ **Play 8**.

Why This Example Matters

This toy game is artificial — on purpose.

It cleanly separates the three sources of uncertainty in game search:

- **MAX:** your decision (which card to play)
- **CHANCE:** randomness (the die roll)
- **MIN:** an adversary reacting optimally

Why This Example Matters

This toy game is artificial — on purpose.

It cleanly separates the three sources of uncertainty in game search:

- **MAX:** your decision (which card to play)
- **CHANCE:** randomness (the die roll)
- **MIN:** an adversary reacting optimally

Why pair this with 2048?

- "Closest Number": **MAX** → **CHANCE** → **MIN**
- 2048: **MAX** → **CHANCE** (no opponent)

Expectimax handles both — by averaging over chance and optimizing against opponents when present.

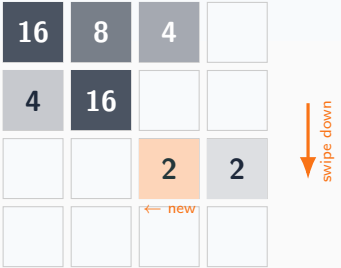
Anchor Example: 2048

How 2048 works:

- 4×4 grid of tiles
- Swipe up / down / left / right
- Matching tiles *merge* and double
- After every move, a *random* tile spawns

Mapping to Expectimax:

- **MAX node** = you choose a swipe direction
- **CHANCE node** = game spawns a tile
  - tile = 2 with probability 0.9
  - tile = 4 with probability 0.1
  - position = uniform over empty



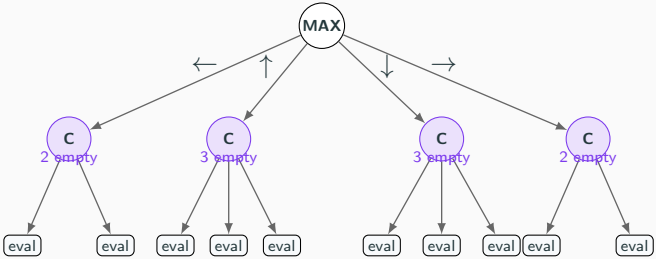
The orange "2" is the randomly spawned tile after the last move.

2048: The Expectimax Tree Example

One empty cell on the board, four legal swipes, each opens different spawn slots

Current board:

2	4	2	4
4	2	4	2
2	4	4	8
4	2	*empty*	8



each CHANCE edge: $\text{prob} = \frac{1}{\text{number of empty cells}}$; $\text{leaf} = \text{evaluate(board)}$

The two **4**s (row 2) merge on any horizontal swipe. The two **8**s (col 3) merge on any vertical swipe, giving a **16**.

More merges $\Rightarrow$ more empty cells $\Rightarrow$ more spawn outcomes for CHANCE.

Expectimax Limitation: No Pruning Possible

Recall: alpha-beta pruning works because at a MIN node, once we find a value $\leq \alpha$ we can stop — that branch can never be chosen by MAX.

Expectimax Limitation: No Pruning Possible

Recall: alpha-beta pruning works because at a MIN node, once we find a value $\leq \alpha$ we can stop — that branch can never be chosen by MAX.

At a CHANCE node that reasoning breaks down:

The expected value is a *weighted sum* of **all** children.
Skipping any child changes the answer.
$\Rightarrow$ We **must** visit every branch.

Consequence:

- Time: $O(b^d \cdot n^d)$, where n = chance outcomes per level
- In 2048: 4 moves $\times$ $\sim$10 spawn positions per

Expectimax Limitation: No Pruning Possible

Recall: alpha-beta pruning works because at a MIN node, once we find a value $\leq \alpha$ we can stop — that branch can never be chosen by MAX.

At a CHANCE node that reasoning breaks down:

The expected value is a *weighted sum* of **all** children.
Skipping any child changes the answer.
$\Rightarrow$ We **must** visit every branch.

Consequence:

- Time: $O(b^d \cdot n^d)$, where n = chance outcomes per level
- In 2048: 4 moves $\times$ $\sim$10 spawn positions per

This motivates the next algorithm:

Monte Carlo Tree Search (MCTS)

Instead of evaluating *every* outcome exactly,
sample promising branches and learn from simulations.

MCTS trades exactness for speed — and it works remarkably well in practice.

MCTS: The Thought Experiment

Imagine you are playing 2048 and it is your turn. Four possible swipes.

You can't search the whole tree — it is too deep. So instead you think:

"What if I just played each move out randomly — hundreds of times — and saw which one led to the best average score?"

MCTS: The Thought Experiment

Imagine you are playing 2048 and it is your turn. Four possible swipes.

You can't search the whole tree — it is too deep. So instead you think:

"What if I just played each move out randomly — hundreds of times — and saw which one led to the best average score?"

Swipe Left	**Swipe Up**	**Swipe Right**	**Swipe Down**
200 random games	200 random games	200 random games	200 random games
Avg score: **4 200**	Avg score: **5 800** ←	Avg score: **3 100**	Avg score: **4 500**

MCTS: The Thought Experiment

Imagine you are playing 2048 and it is your turn. Four possible swipes.

You can't search the whole tree — it is too deep. So instead you think:

"What if I just played each move out randomly — hundreds of times — and saw which one led to the best average score?"

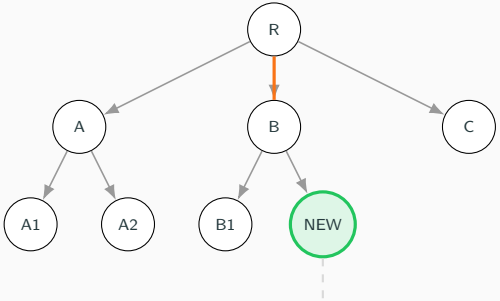
Swipe Left	**Swipe Up**	**Swipe Right**	**Swipe Down**
200 random games	200 random games	200 random games	200 random games
Avg score: **4 200**	Avg score: **5 800** ←	Avg score: **3 100**	Avg score: **4 500**

Pick "Swipe Up". That intuition — *simulate, compare, choose* — is the core of **MCTS**.

But doing this uniformly wastes effort on bad moves — MCTS learns where to

MCTS: The Four Phases

One iteration of the algorithm



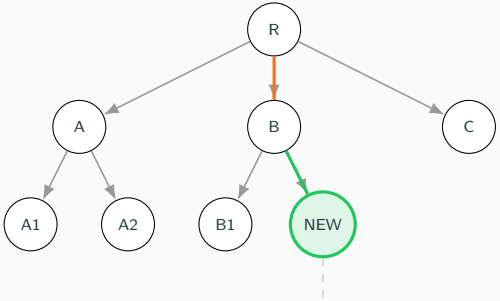
(1) Selection

Walk down the tree choosing the **most promising** child, using past results and visit counts. Stop at a node with unexplored children.

Repeat this loop thousands of times, then pick the child of the root with the most visits.

MCTS: The Four Phases

One iteration of the algorithm



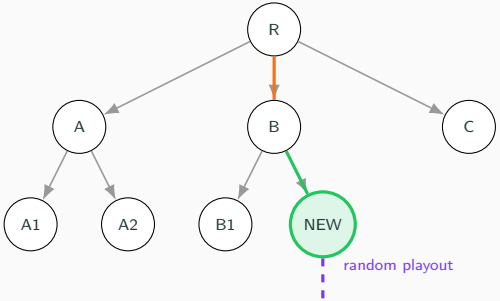
(2) Expansion

Add one new child node — a move we haven't tried from this position yet.

Repeat this loop thousands of times, then pick the child of the root with the most visits.

MCTS: The Four Phases

One iteration of the algorithm



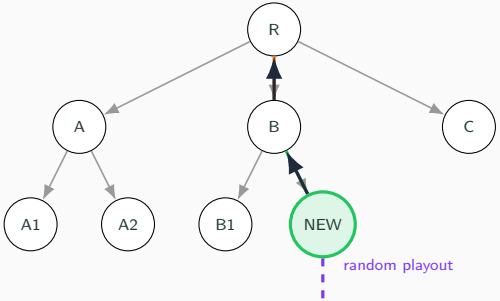
(3) Simulation

From the new node, play random moves to the end of the game. Record the result.

Repeat this loop thousands of times, then pick the child of the root with the most visits.

MCTS: The Four Phases

One iteration of the algorithm



(4) Backpropagation

Walk back to the root. At every node on the path: reward $+=$ result, visits $+= 1$. These counts guide future selection.

Repeat this loop thousands of times, then pick the child of the root with the most visits.

What Information Does MCTS Store?

At every node (state), MCTS stores **simple statistics** about each action/child:

- **Visits** (n): how many times this child was selected
- **Total reward** (w): sum of rollout returns from this child
- **Average value** ($\bar{X}$): $\bar{X} = w/n$

What Information Does MCTS Store?

At every node (state), MCTS stores **simple statistics** about each action/child:

- **Visits** (n): how many times this child was selected
- **Total reward** (w): sum of rollout returns from this child
- **Average value** ($\bar{X}$): $\bar{X} = w/n$

After thousands of rollouts, these numbers summarize what we learned.

Key decision inside Selection

Which child should we try next, given what we know so far?

Selection: Exploration vs. Exploitation

The decision problem inside Selection

The tension:

Exploitation: keep choosing actions with high average value so far.

Exploration: try actions we haven't tested much — they might be better.

Concrete example:

- Move A: 7 wins out of 10 rollouts (70%)
- Move B: 3 wins out of 5 rollouts (60%)

A looks better *so far*, but B is based on less evidence.
If B is actually stronger, we don't want to miss it.

Selection: Exploration vs. Exploitation

The decision problem inside Selection

The tension:

Exploitation: keep choosing actions with high average value so far.

Exploration: try actions we haven't tested much — they might be better.

Concrete example:

- Move A: 7 wins out of 10 rollouts (70%)
- Move B: 3 wins out of 5 rollouts (60%)

A looks better *so far*, but B is based on less evidence. If B is actually stronger, we don't want to miss it.

Goal: pick the child that is *worth sampling next*.

"Try what looks good, but also keep checking what you haven't tried."

UCT/UCB is a standard way to implement this tradeoff.

UCT (UCB in Trees): The Selection Score

During **Selection**, MCTS often uses **UCT** (UCB applied to trees):

$$\mathrm{UCT}(i) = \underbrace{\frac{w_i}{n_i}}_{\text{exploitation}} + \underbrace{c\sqrt{\frac{\ln N}{n_i}}}_{\text{exploration bonus}}$$

- w_i = total reward accumulated from child i
- n_i = number of visits to child i
- N = number of visits to the *parent* node
- c = exploration constant (often tuned; a common default is $\sqrt{2}$)

UCT (UCB in Trees): The Selection Score

During **Selection**, MCTS often uses **UCT** (UCB applied to trees):

$$\mathrm{UCT}(i) = \underbrace{\frac{w_i}{n_i}}_{\text{exploitation}} + \underbrace{c\sqrt{\frac{\ln N}{n_i}}}_{\text{exploration bonus}}$$

- w_i = total reward accumulated from child i
- n_i = number of visits to child i
- N = number of visits to the *parent* node
- c = exploration constant (often tuned; a common default is $\sqrt{2}$)

Intuition:

- $\frac{w_i}{n_i}$ favors actions that have worked well (**exploit**)
- $c\sqrt{\frac{\ln N}{n_i}}$ is large when n_i is small (**explore**)

Convention: if $n_i = 0$, treat UCT as $+\infty$ so every child is tried at least once.

Why UCT Works Well in Practice

UCT is a principled balance between **learning** and **using what you've learned**:

- **Avoids getting stuck early:** even if one move looks best after a few rollouts, UCT still checks alternatives.
- **Focuses compute where it matters:** as evidence accumulates, weak moves get sampled less.
- **Automatic annealing:** exploration pressure decreases naturally for heavily-visited moves because the bonus shrinks with n_i.

Why UCT Works Well in Practice

UCT is a principled balance between **learning** and **using what you've learned**:

- **Avoids getting stuck early:** even if one move looks best after a few rollouts, UCT still checks alternatives.
- **Focuses compute where it matters:** as evidence accumulates, weak moves get sampled less.
- **Automatic annealing:** exploration pressure decreases naturally for heavily-visited moves because the bonus shrinks with n_i.

One-line takeaway

UCT doesn't pick the best move — it picks the move most worth learning about next.

UCT Example: Step-by-Step Selection Compute UCT scores and Pick Max

Assume the **parent** has been visited $N = 20$ times, and $c = \sqrt{2} \approx 1.414$. Two children:

- Child A: $w_A = 14$, $n_A = 20$ (avg $= 14/20 = 0.70$)
- Child B: $w_B = 3$, $n_B = 5$ (avg $= 3/5 = 0.60$)

UCT Example: Step-by-Step Selection Compute UCT scores and Pick Max

Assume the **parent** has been visited $N = 20$ times, and $c = \sqrt{2} \approx 1.414$. Two children:

- Child A: $w_A = 14$, $n_A = 20$ (avg $= 14/20 = 0.70$)
- Child B: $w_B = 3$, $n_B = 5$ (avg $= 3/5 = 0.60$)

Step 1: compute $\ln N = \ln 20 \approx 2.996$.

UCT Example: Step-by-Step Selection Compute UCT scores and Pick Max

Assume the **parent** has been visited $N = 20$ times, and $c = \sqrt{2} \approx 1.414$. Two children:

- Child A: $w_A = 14$, $n_A = 20$ (avg $= 14/20 = 0.70$)
- Child B: $w_B = 3$, $n_B = 5$ (avg $= 3/5 = 0.60$)

Step 1: compute $\ln N = \ln 20 \approx 2.996$. **Step 2: exploration bonus for A**

$$c\sqrt{\frac{\ln 20}{20}} = 1.414\sqrt{\frac{2.996}{20}} = 1.414\sqrt{0.1498} \approx 1.414(0.387) \approx 0.547$$

UCT Example: Step-by-Step Selection Compute UCT scores and Pick Max

Assume the **parent** has been visited $N = 20$ times, and $c = \sqrt{2} \approx 1.414$. Two children:

- Child A: $w_A = 14$, $n_A = 20$ (avg $= 14/20 = 0.70$)
- Child B: $w_B = 3$, $n_B = 5$ (avg $= 3/5 = 0.60$)

Step 1: compute $\ln N = \ln 20 \approx 2.996$. **Step 2: exploration bonus for A**

$$c\sqrt{\frac{\ln 20}{20}} = 1.414\sqrt{\frac{2.996}{20}} = 1.414\sqrt{0.1498} \approx 1.414(0.387) \approx 0.547$$

$$\mathrm{UCT}(A) \approx 0.70 + 0.547 = 1.247$$

UCT Example: Step-by-Step Selection Compute UCT scores and Pick Max

Assume the **parent** has been visited $N = 20$ times, and $c = \sqrt{2} \approx 1.414$. Two children:

- Child A: $w_A = 14$, $n_A = 20$ (avg $= 14/20 = 0.70$)
- Child B: $w_B = 3$, $n_B = 5$ (avg $= 3/5 = 0.60$)

Step 1: compute $\ln N = \ln 20 \approx 2.996$. **Step 2: exploration bonus for A**

$$c\sqrt{\frac{\ln 20}{20}} = 1.414\sqrt{\frac{2.996}{20}} = 1.414\sqrt{0.1498} \approx 1.414(0.387) \approx 0.547$$

$$\mathrm{UCT}(A) \approx 0.70 + 0.547 = 1.247$$

Step 3: exploration bonus for B

$$c\sqrt{\frac{\ln 20}{5}} = 1.414\sqrt{\frac{2.996}{5}} = 1.414\sqrt{0.5992} \approx 1.414(0.774) \approx 1.095$$

UCT Example: Step-by-Step Selection Compute UCT scores and Pick Max

Assume the **parent** has been visited $N = 20$ times, and $c = \sqrt{2} \approx 1.414$. Two children:

- Child A: $w_A = 14$, $n_A = 20$ (avg $= 14/20 = 0.70$)
- Child B: $w_B = 3$, $n_B = 5$ (avg $= 3/5 = 0.60$)

Step 1: compute $\ln N = \ln 20 \approx 2.996$. **Step 2: exploration bonus for A**

$$c\sqrt{\frac{\ln 20}{20}} = 1.414\sqrt{\frac{2.996}{20}} = 1.414\sqrt{0.1498} \approx 1.414(0.387) \approx 0.547$$

$$\mathrm{UCT}(A) \approx 0.70 + 0.547 = 1.247$$

Step 3: exploration bonus for B

$$c\sqrt{\frac{\ln 20}{5}} = 1.414\sqrt{\frac{2.996}{5}} = 1.414\sqrt{0.5992} \approx 1.414(0.774) \approx 1.095$$

$$\mathrm{UCT}(B) \approx 0.60 + 1.095 = 1.695$$

UCT Example: Step-by-Step Selection Compute UCT scores and Pick Max

Assume the **parent** has been visited $N = 20$ times, and $c = \sqrt{2} \approx 1.414$. Two children:

- Child A: $w_A = 14$, $n_A = 20$ (avg $= 14/20 = 0.70$)
- Child B: $w_B = 3$, $n_B = 5$ (avg $= 3/5 = 0.60$)

Step 1: compute $\ln N = \ln 20 \approx 2.996$. **Step 2: exploration bonus for A**

$$c\sqrt{\frac{\ln 20}{20}} = 1.414\sqrt{\frac{2.996}{20}} = 1.414\sqrt{0.1498} \approx 1.414(0.387) \approx 0.547$$

$$\mathrm{UCT}(A) \approx 0.70 + 0.547 = 1.247$$

Step 3: exploration bonus for B

$$c\sqrt{\frac{\ln 20}{5}} = 1.414\sqrt{\frac{2.996}{5}} = 1.414\sqrt{0.5992} \approx 1.414(0.774) \approx 1.095$$

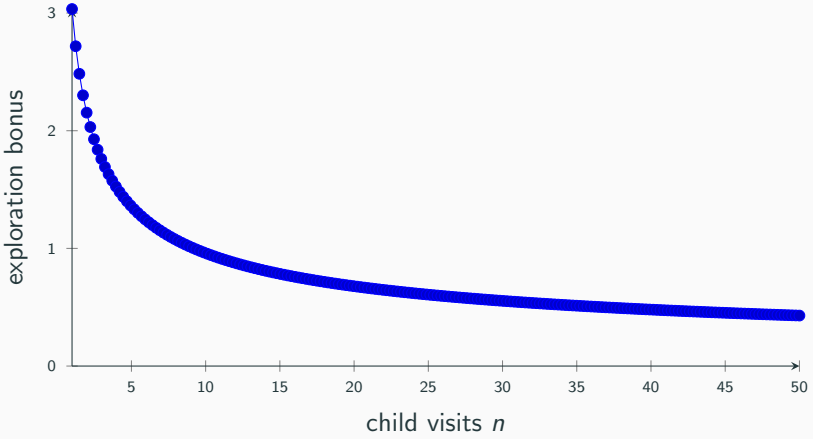
$$\mathrm{UCT}(B) \approx 0.60 + 1.095 = 1.695$$

Decision

Since $\mathrm{UCT}(B) > \mathrm{UCT}(A)$, **choose B next** (it is under-explored).

Intuition: Exploration Bonus Shrinks with Visits

For fixed parent visits N and constant c, the exploration term is $c\sqrt{\frac{\ln N}{n}}$ which decreases as n increases.



Early on (small n), the bonus is large $\Rightarrow$ explore. Later (large n), the bonus is small $\Rightarrow$ exploit.

What If We Did Not Explore? (Common Failure Mode)

If we always pick the child with the highest current average $\frac{w}{n}$:

- early randomness can make a mediocre move look best
- the algorithm can get stuck **over-sampling** that move
- truly strong moves might never get enough trials to prove themselves

What If We Did Not Explore? (Common Failure Mode)

If we always pick the child with the highest current average $\frac{w}{n}$:

- early randomness can make a mediocre move look best
- the algorithm can get stuck **over-sampling** that move
- truly strong moves might never get enough trials to prove themselves

Why UCT matters

Exploration is what prevents "premature certainty."

Expectimax vs. MCTS: When to Use Which

	Expectimax	**MCTS**
Evaluates chance outcomes	All of them (exact)	Samples (approximate)
Needs an evaluation function?	Yes	No (can use win/loss)
Prunable?	No	N/A
Time behaviour	Fixed-depth search	Anytime — improves with more time
Good when tree is...	Moderate size	Very large
Depth it can reach	Shallow (4–6)	Deep (to end of game)
Example	2048 (depth-limited)	Go, AlphaZero

Practical guideline:

- You have a *good evaluation function* and moderate branching → **Expectimax**
- Branching factor is huge or evaluation is hard → **MCTS**
- Modern systems often *combine* both: MCTS guides the search, a neural network provides the evaluation (AlphaGo, AlphaZero).

Summary

Probability basics

- Sample space, events, probability
- Random variables
- Expected value $E[X] = \sum p_i x_i$: the long-run average
- Use EV to compare uncertain actions

Expectimax

- CHANCE nodes replace (or join) MIN nodes
- CHANCE value = expected value over outcomes
- No pruning must visit all branches
- Works well with a good evaluation

MCTS

- Simulate, compare, choose
- Four phases: Select → Expand → Simulate → Backpropagate
- UCB balances exploration and exploitation
- Anytime: better with more iterations

The big picture (revisit the grid)

- Deterministic + adversary → Minimax ($\exists/\forall$)
- Stochastic + full obs → Expectimax / MCTS
- Hidden info → Game Theory (coming

Next Week

Coming up: Constraint Satisfaction Problems (CSPs)

- A different paradigm: *satisfy constraints*, not find shortest path
- Sudoku, map coloring, scheduling
- Backtracking search with constraint propagation
- Connects back to the $\exists/\forall$ quantifier view we saw today

Reading: Russell & Norvig, Chapter 6 (Constraint Satisfaction Problems)