EXPECTIMINIMAX CHEATSHEET

Game Trees: Adversaries + Uncertainty | AI Search Algorithms

What is Expectiminimax?

An extension of Minimax that handles BOTH adversarial opponents AND random chance events.

Used in games like Backgammon: you roll dice (chance) and play against an opponent (adversary).

Three node types alternate in the game tree: MAX (you) -> CHANCE (randomness) -> MIN (opponent).

The Three Node Types

MAX Node

YOUR turn - pick BEST

U(s) = max Q(s,a)

"Me - I want MAX!"

MIN Node

OPPONENT picks WORST

U(s) = min Q(s,a)

"Monster - hurts me"

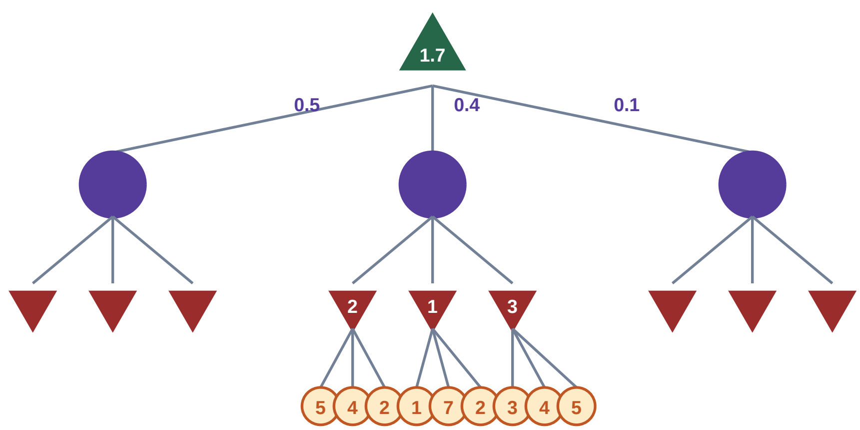
CHANCE Node

RANDOM weighted average

Q = Sum P(s') * U(s')

"Chaos - dice roll"

Worked Example (from class slide)



Step-by-Step Calculation:

Step 1: MIN nodes pick smallest child value

Left MIN: min(5,4,2) = 2

Mid MIN: min(1,7,2) = 1

Right MIN: min(3,4,5) = 3

Step 2: CHANCE node weighted average

0.5 x 2 + 0.4 x 1 + 0.1 x 3

= 1.0 + 0.4 + 0.3 = 1.7

Step 3: MAX node picks highest = 1.7

Mnemonic: "Me, Chaos, Monster"

Me = MAX **Chaos = CHANCE** **Monster =**

Layers: Me -> Chaos -> Monster -> Chaos -> Me ...

Also: "MaCMIN" = Max, Chance, MIN

Quick Algorithm Comparison

Minimax:	`MAX <-> MIN`	(Chess)
Expectimax:	`MAX <-> CHANCE`	(Dice solitaire)
Expectiminimax:	`MAX <-> CHANCE <-> MIN`	(Backgammon)

PROBLEMS, SOLUTIONS & PSEUDOCODE

Key Problems

1. Exponential Blowup
Extra CHANCE layers = much deeper tree. O((b*n)^d)

2. No Alpha-Beta Pruning
Can't skip branches - need ALL children for averages

3. Utility Scale Sensitivity
Nonlinear transforms change decisions (unlike Minimax)

4. Need Probability Model
Must know P(outcome). Wrong model = bad play

Solution Strategies

1. Depth-Limited Search
Stop early + evaluation function to estimate

2. Monte Carlo Sampling
Sample random outcomes instead of enumerating all

3. Star1/Star2 Pruning
Special pruning using bounds on chance values

4. Forward Pruning
Ignore very low-probability branches

Pseudocode

```
def value(state):
    if state is terminal: return utility(state)
    if agent(state) == MAX: return max_value(state)
    if agent(state) == MIN: return min_value(state)
    if agent(state) == EXP: return exp_value(state)

def max_value(state):          # Agent's turn
    v = -infinity
    for s' in successors(state):
        v = max(v, value(s'))
    return v

def min_value(state):          # Opponent's turn
    v = +infinity
    for s' in successors(state):
        v = min(v, value(s'))
    return v

def exp_value(state):          # Chance event
    v = 0
    for s' in successors(state):
        p = probability(s')
        v += p * value(s')
    return v
```

Formula Quick Reference

Node Type	Symbol	Operation	Formula	Analogy
MAX	Triangle Up	max(children)	U(s) = max Q(s,a)	Your best move
MIN	Triangle Down	min(children)	U(s) = min Q(s,a)	Opponent's best
CHANCE	Circle	weighted avg	Q = Sum P(s')*U(s')	Dice / luck
Terminal	Leaf	known value	V(s) = utility	Game over

Exam Tips & Key Takeaways

Always work BOTTOM-UP: leaves -> MIN/MAX -> CHANCE -> repeat upward to root.
CHANCE = expected value = Sum(probability x child). Don't pick min or max here!
Expectiminimax IS sensitive to monotonic utility transforms. Minimax is NOT.
Alpha-beta doesn't work directly here. Use Monte Carlo / Star pruning instead.