Chapter 5

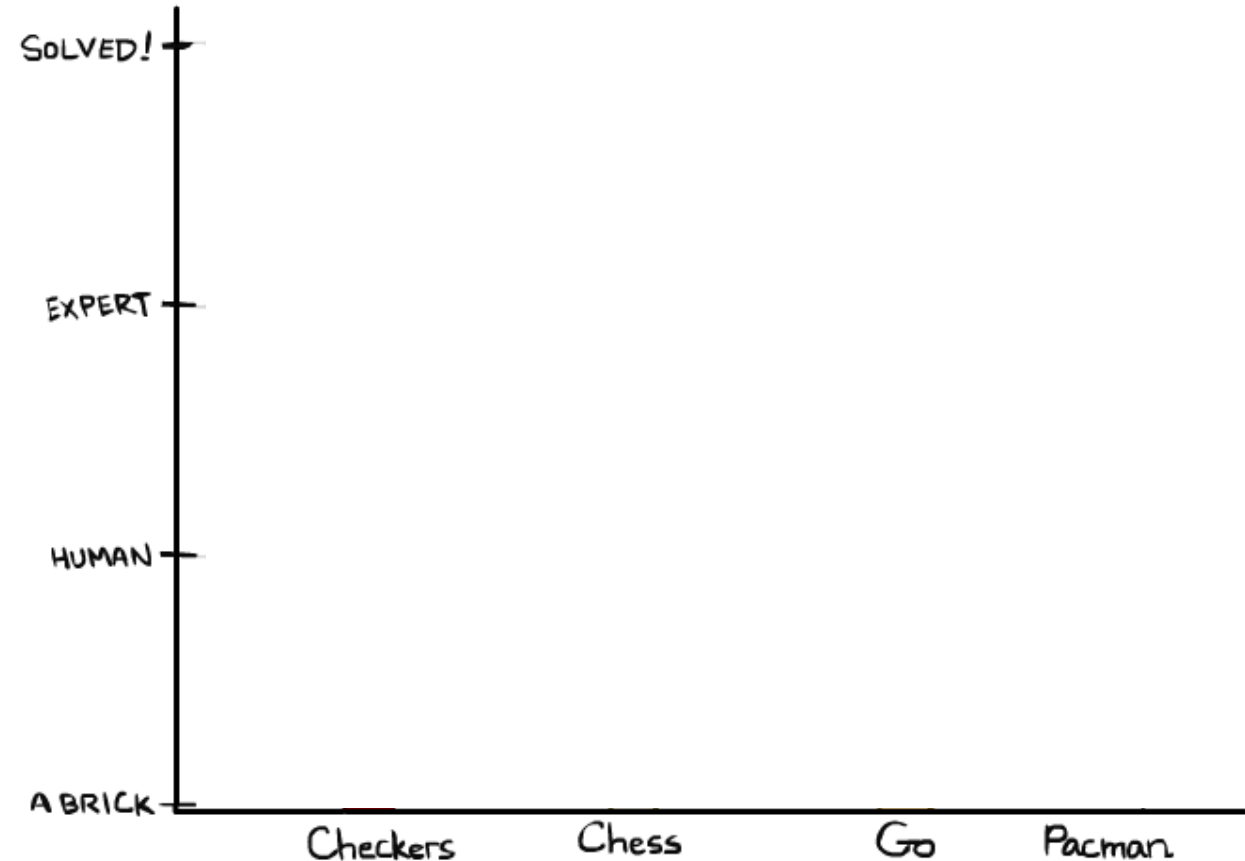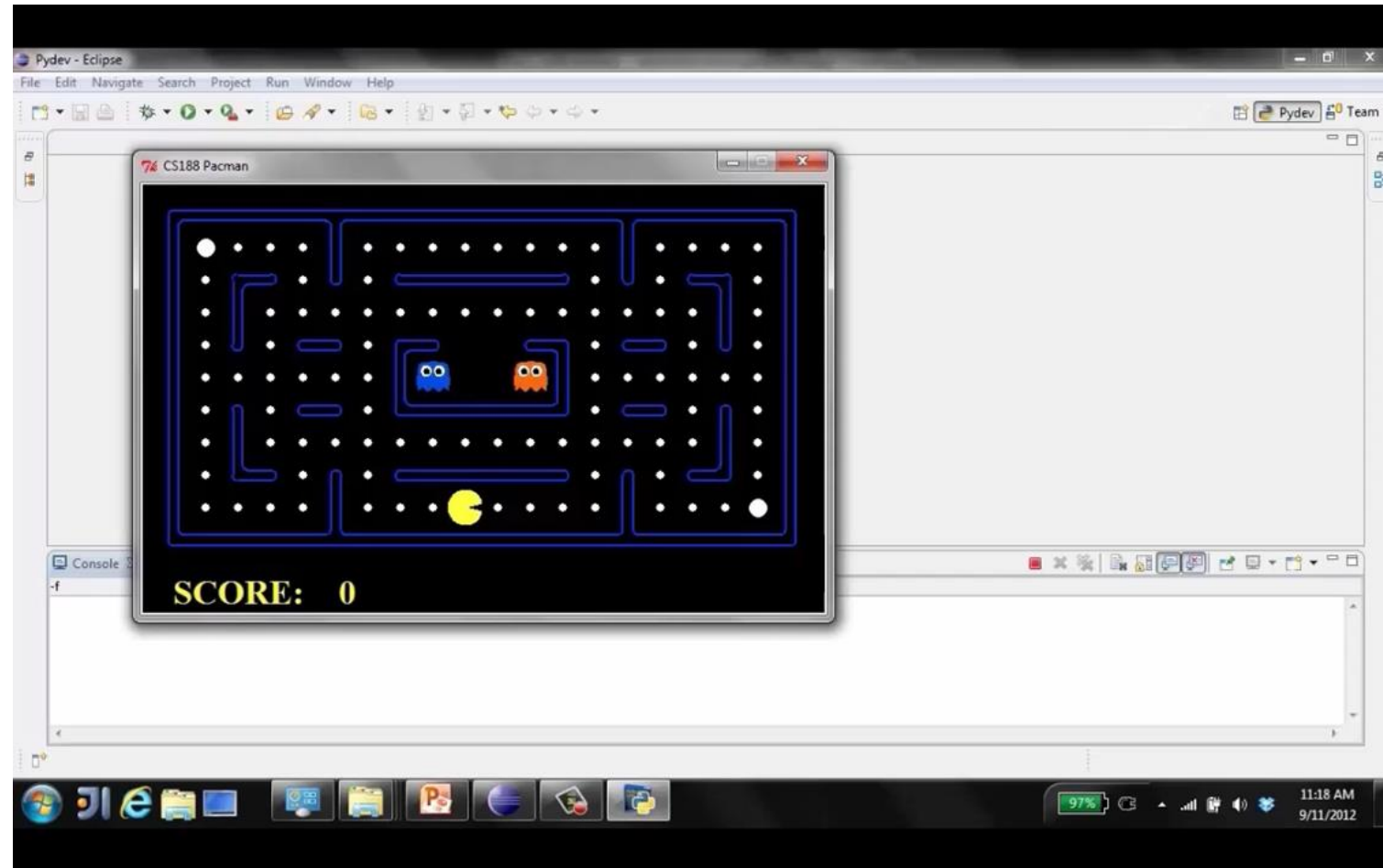## Adversarial Search and Games

Topics Covered

❖Game Theory

❖Optimal Decisions in Games

❖Minmax Algorithm

❖Alpha-Beta Tree Search

❖Expectimax Tress Search

Game Playing State-of-the-Art

❖ **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

❖ **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. AlphaGo in 2016 beats the human

❖ **Pacman**

SOLVED!

EXPERT

HUMAN

A BRICK

Checkers   Chess   Go   Pacman

Game Playing

Artificial Intelligence   (slide adapted from ai.berkeley.edu)

Types of Games

❖ Many 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 x A $\rightarrow$ S
  ➢ Terminal Test: S $\rightarrow$ {t,f}
  ➢ Terminal Utilities: S x P $\rightarrow$ R

❖ Solution for a player is a policy: S $\rightarrow$ A

Artificial Intelligence (slide adapted from ai.berkeley.edu)

Zero-Sum Games

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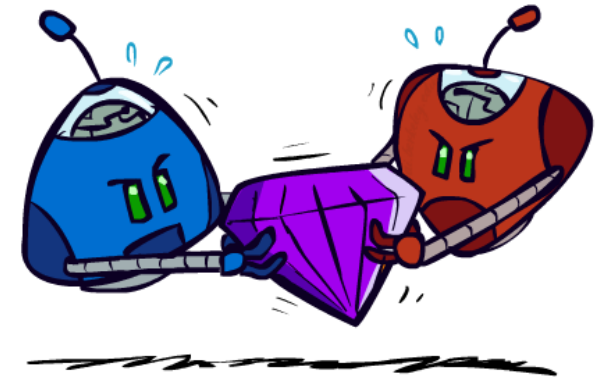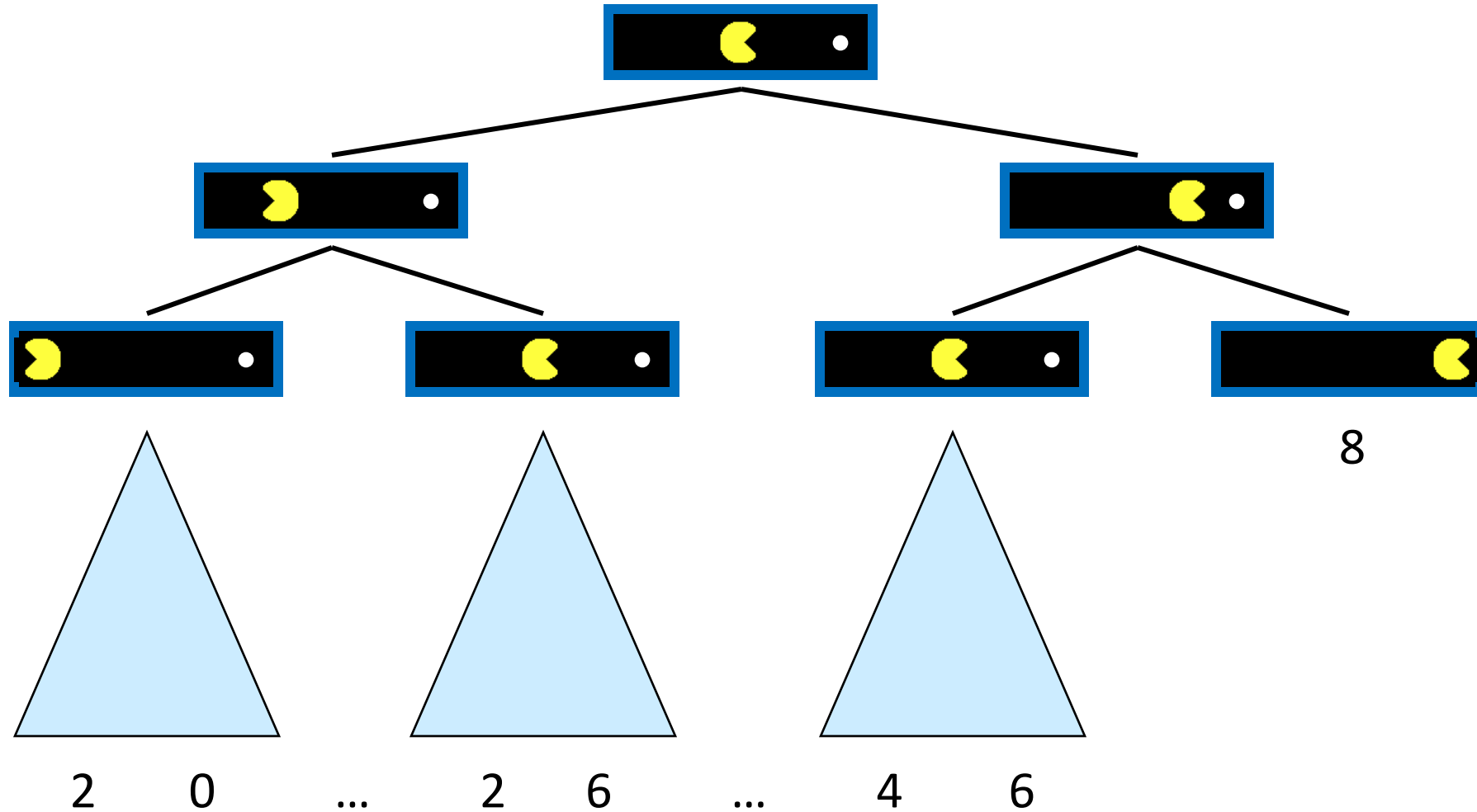
❖General Games (Non-zero sum Game)
  • Agents have independent utilities (values on outcomes)
  • Cooperation, indifference, competition, and more are all possible
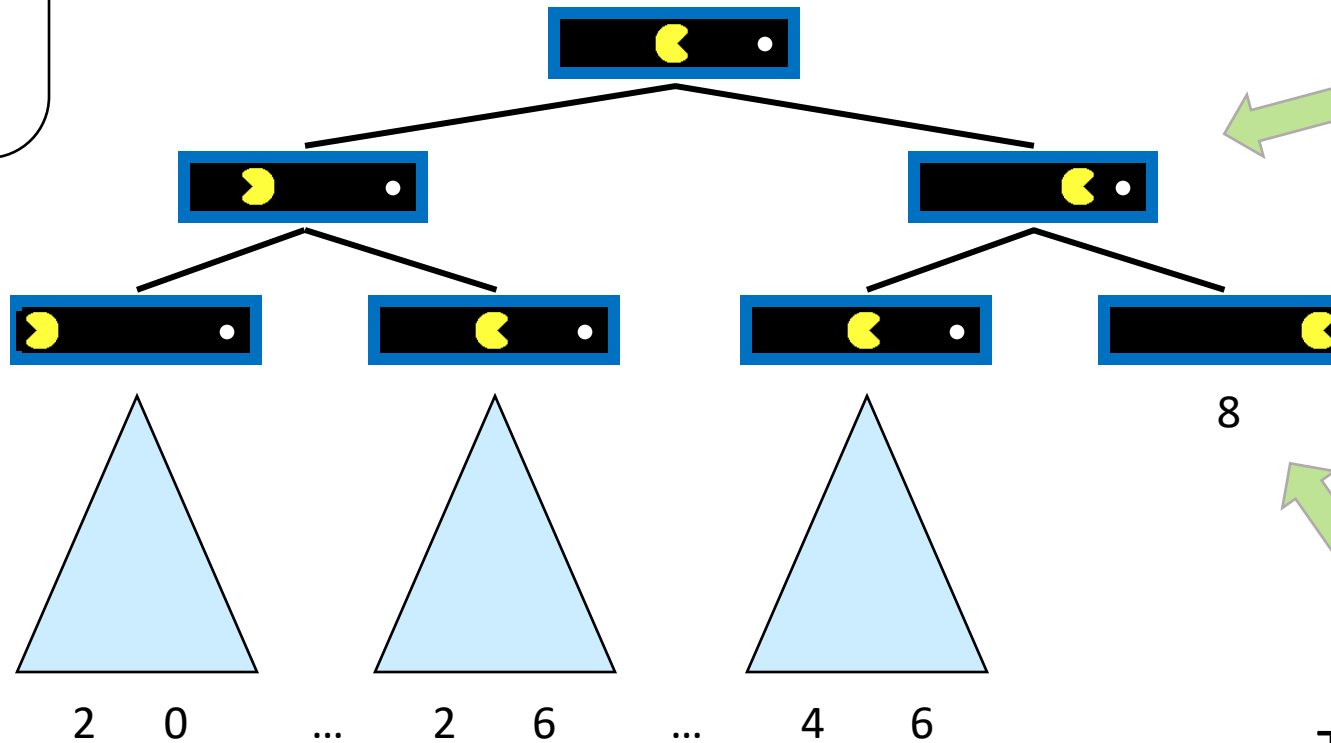 • More later, non-zero-sum games
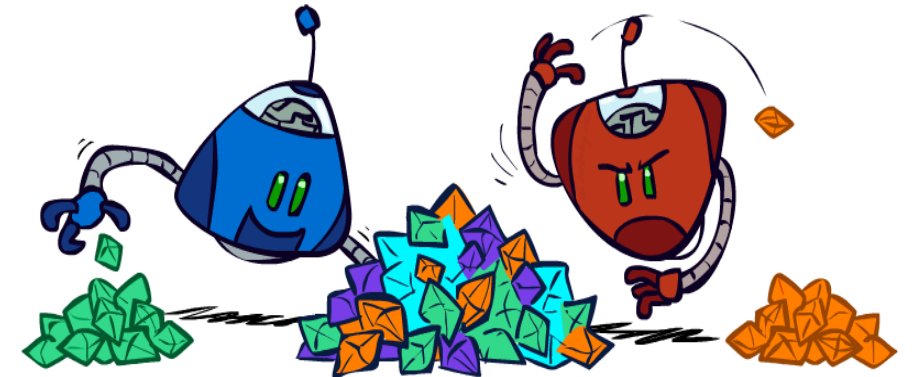
Adversarial Games

Adversarial Search

Single-Agent Trees

2   0   ...   2   6   ...   4   6

Artificial Intelligence (slide adapted from ai.berkeley.edu)

Value of a State

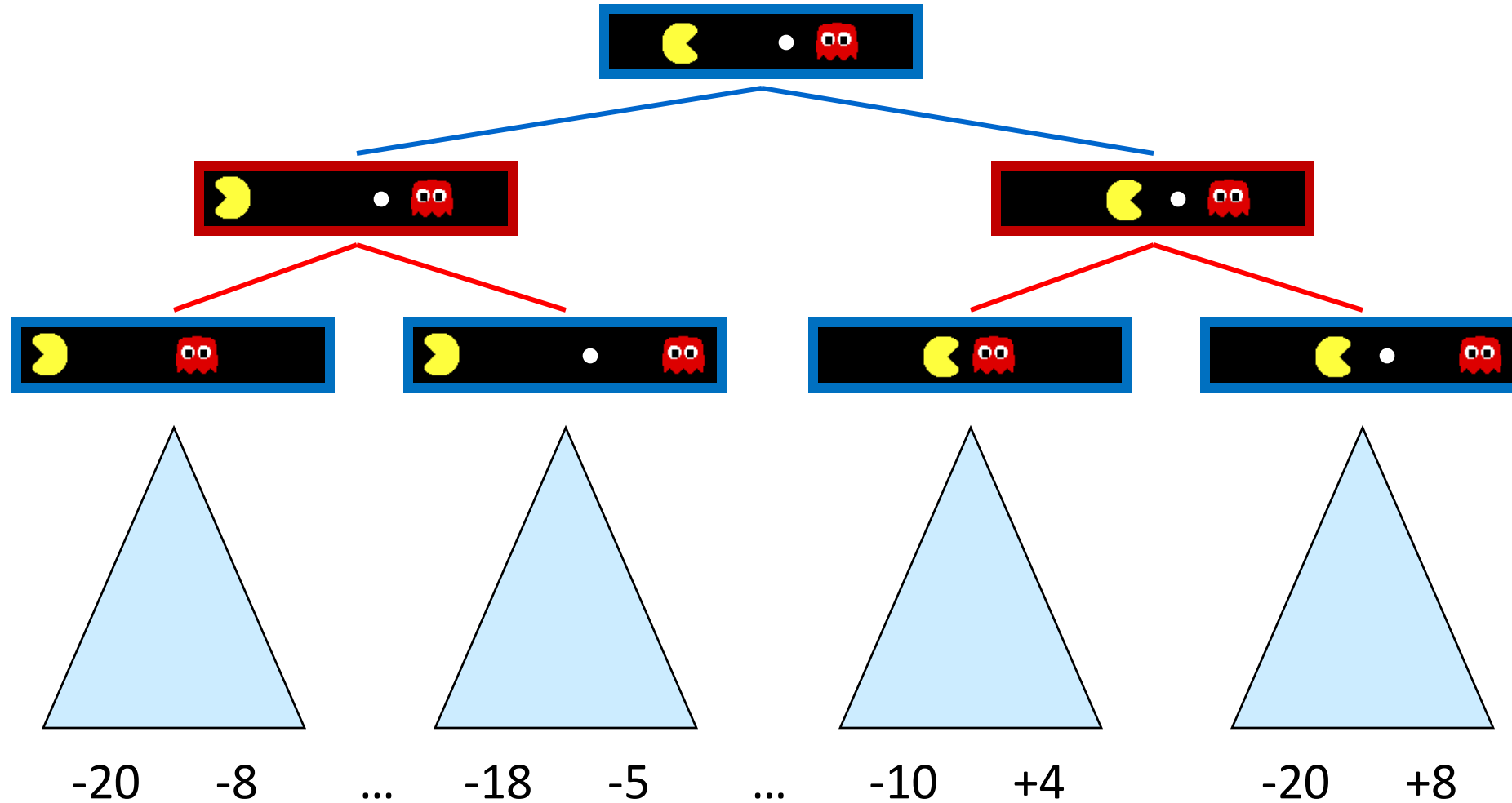Value of a state:
The best achievable
outcome (utility)
from that state

Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$
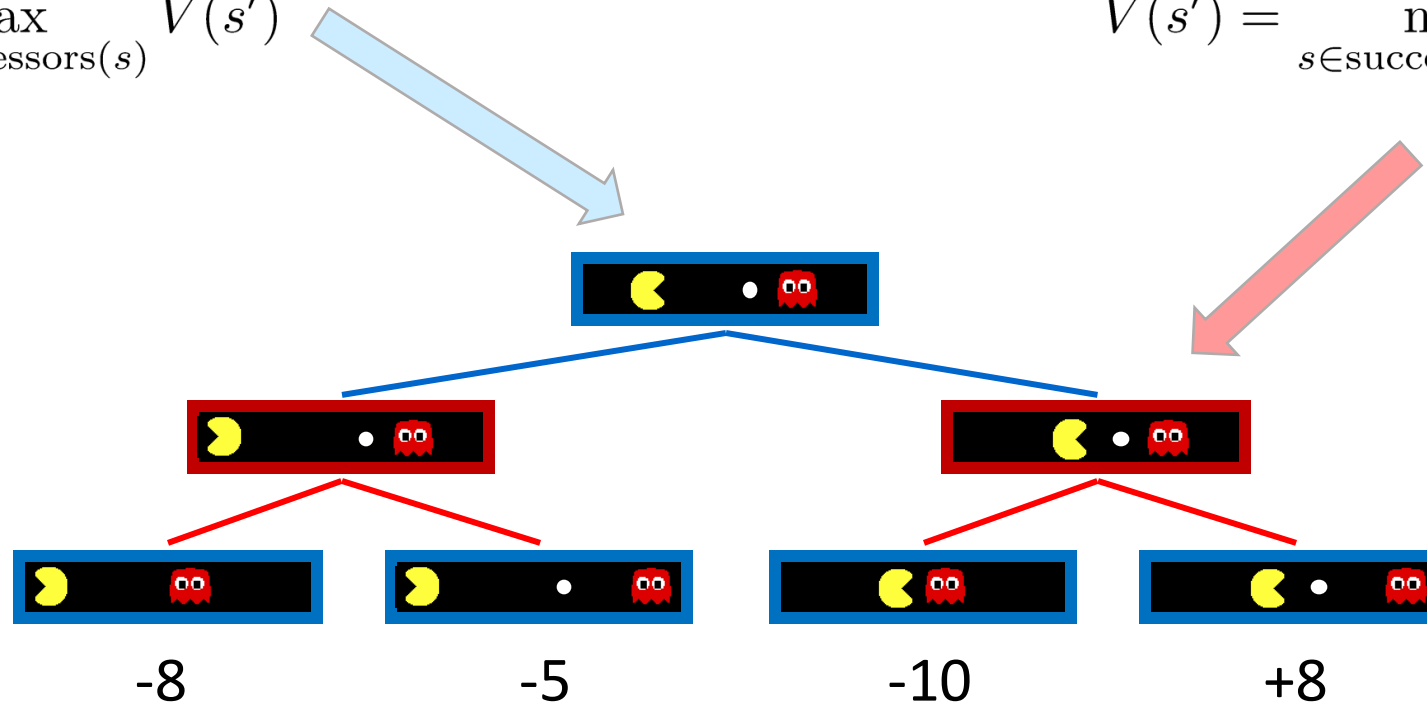

8

Terminal States:

$$V(s) = \text{known}$$

2   0   …   2   6   …   4   6

Artificial Intelligence (slide adapted from ai.berkeley.edu)

Adversarial Game Trees

Artificial Intelligence (slide adapted from ai.berkeley.edu)

Minimax Values

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

-8   -5   -10   +8

Terminal States:

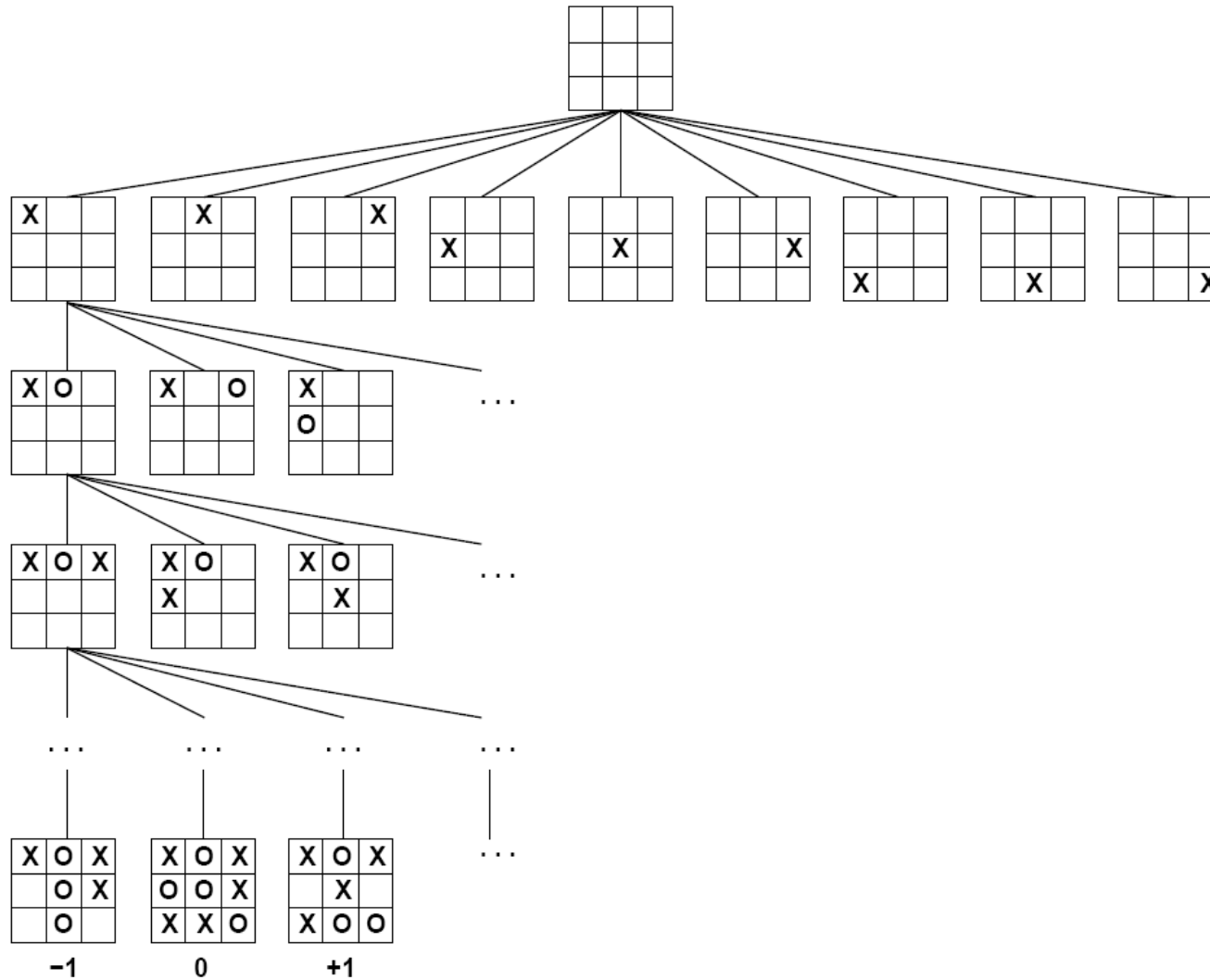$$V(s) = \text{known}$$

# Tic-Tac-Toe Game Tree

MAX (X)

MIN (O)

MAX (X)

MIN (O)

TERMINAL

Utility   −1   0   +1

Adversarial Search (Minimax)

❖ 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
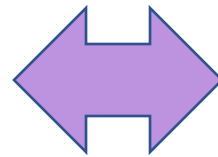
Minimax values:
computed recursively

5 **max**

2   5 **min**

8   2   5   6

Terminal values:
part of the game
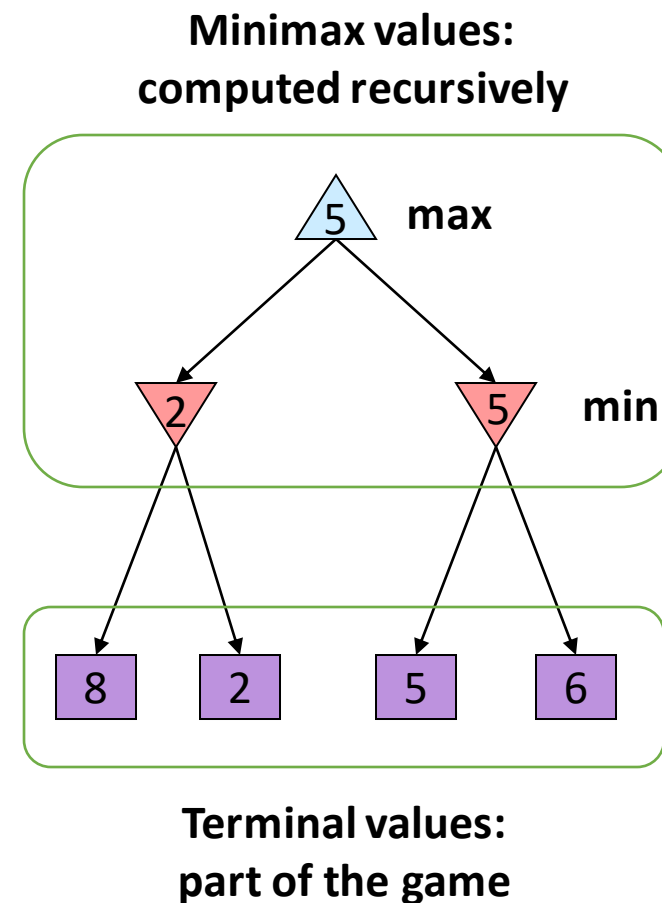
Minimax Implementation

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

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Minimax Implementation (Dispatch)

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, min-value(successor))
   return v

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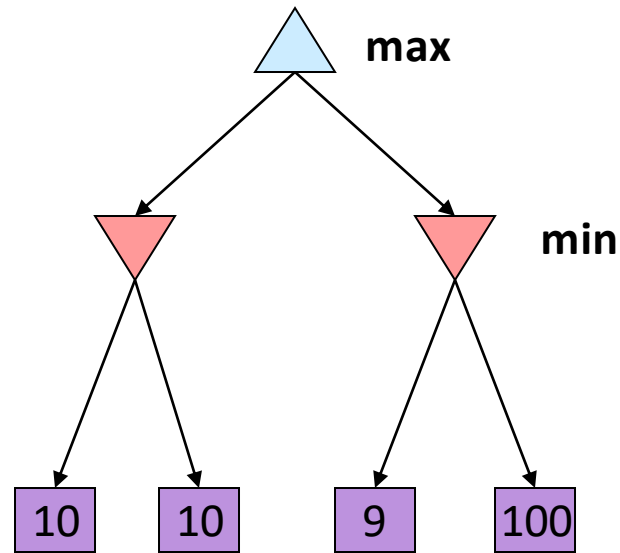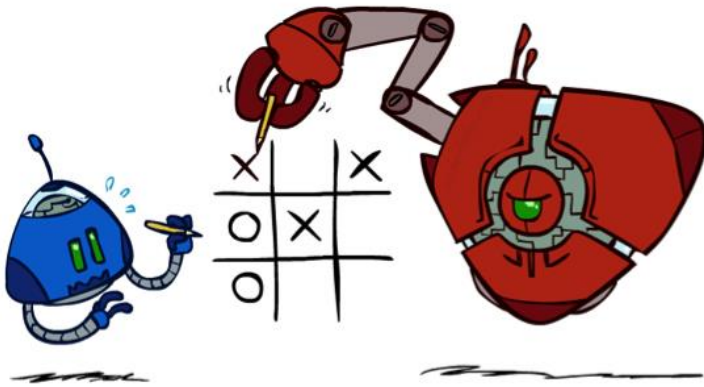   initialize v = +∞
   for each successor of state:
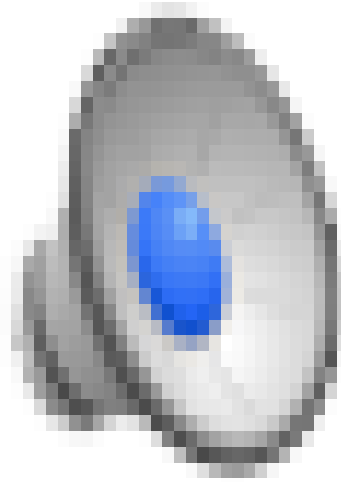   v = min(v, max-value(successor))
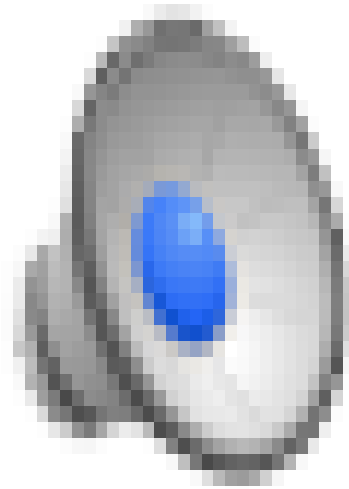   return v
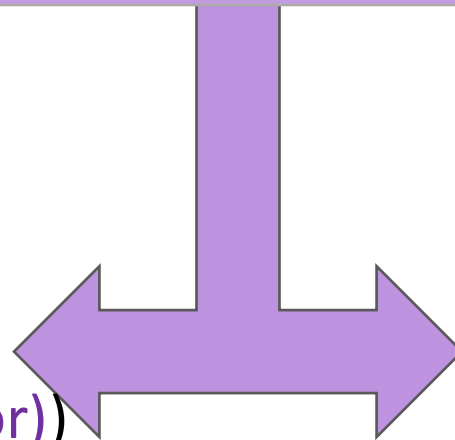
Minimax Example
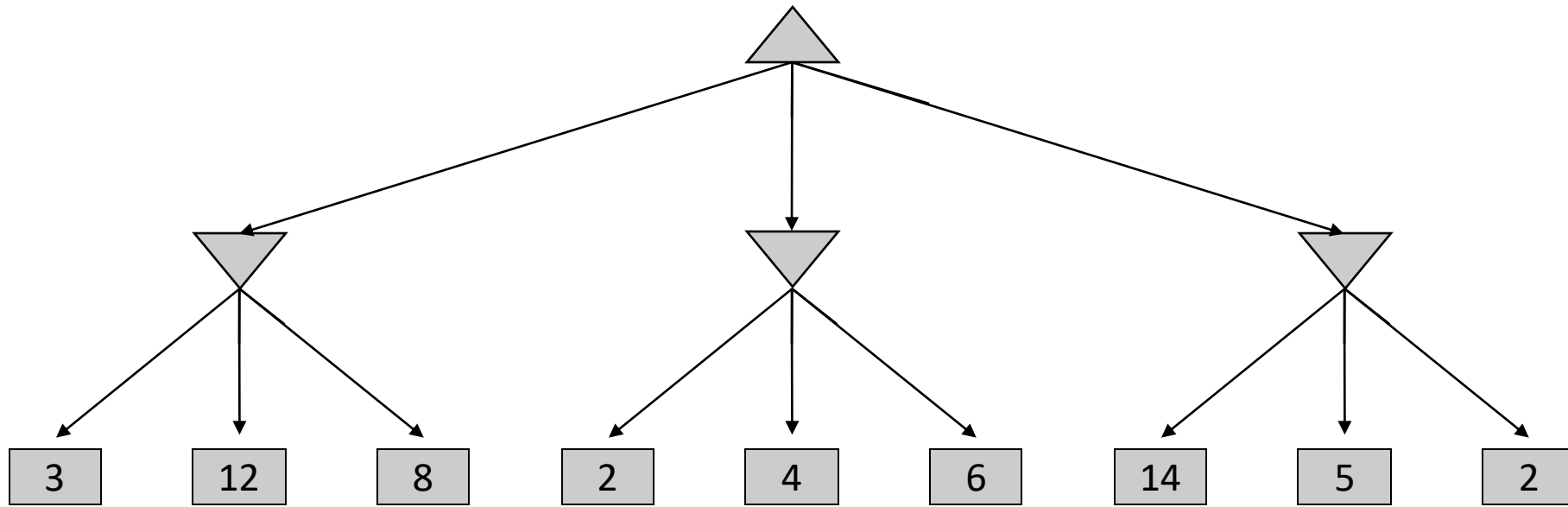
Minimax Properties

max

min

10   10   9   100
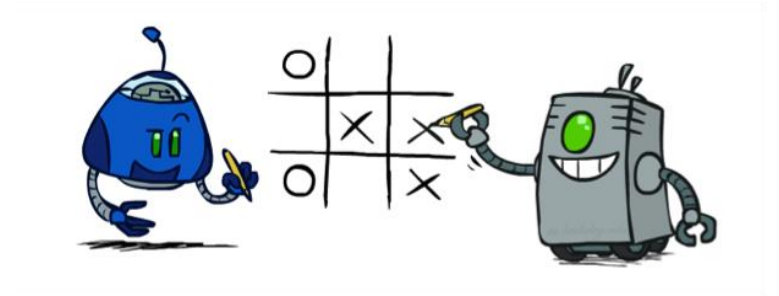
Optimal against a perfect player. Otherwise?

Video of Demo Min vs. Exp (Min)

Video of Demo Min vs. Exp (Exp)

Artificial Intelligence   (slide adapted from ai.berkeley.edu)

Minimax Efficiency

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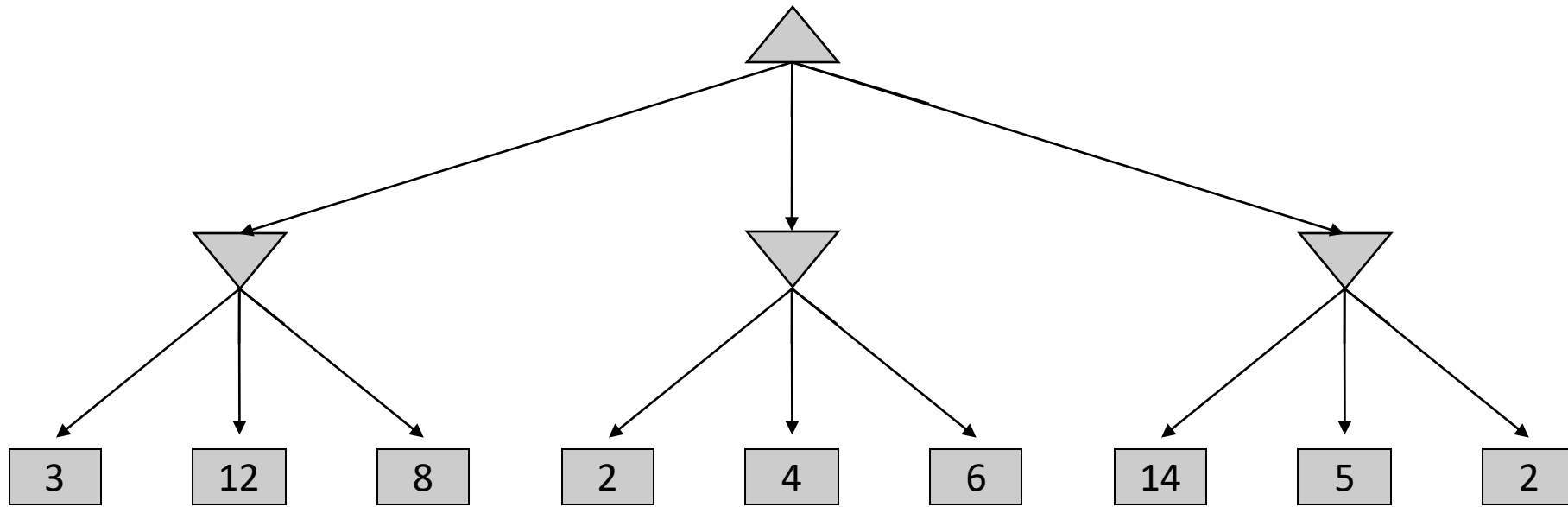➢ Time: $O(b^m)$
➢ Space: $O(bm)$

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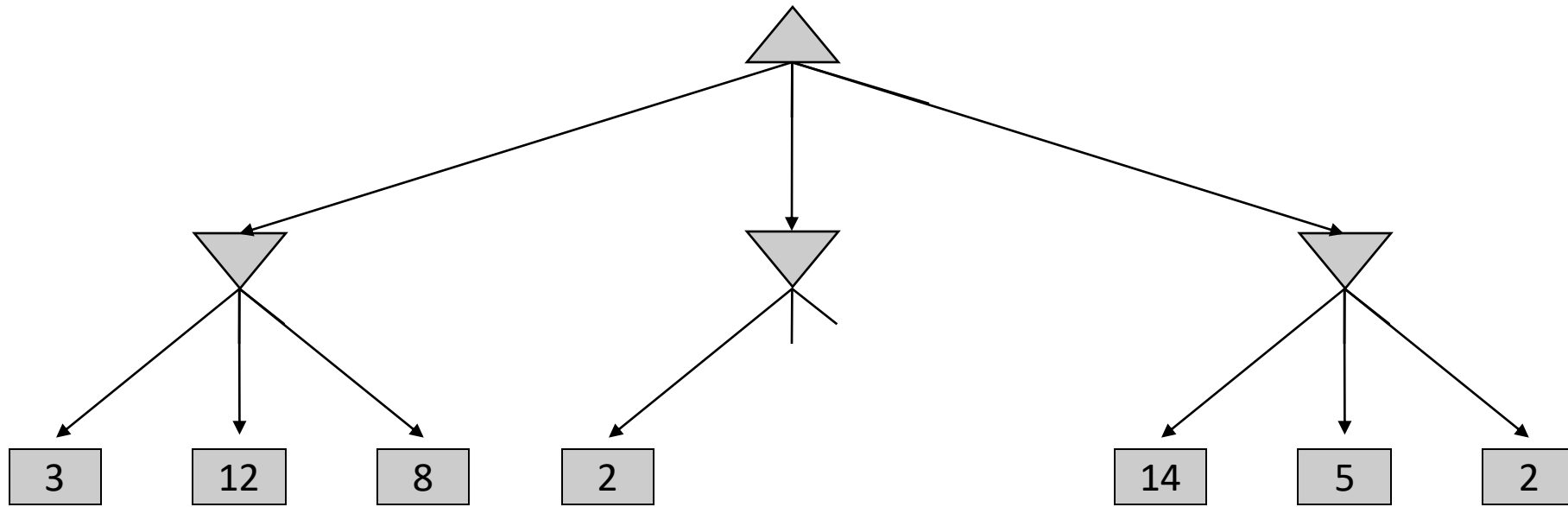➢ Exact solution is completely infeasible
➢ But, do we need to explore the whole tree?

Game Tree Pruning

Minimax Example

Artificial Intelligence (slide adapted from ai.berkeley.edu)

Minimax Pruning

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 childrens' 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)
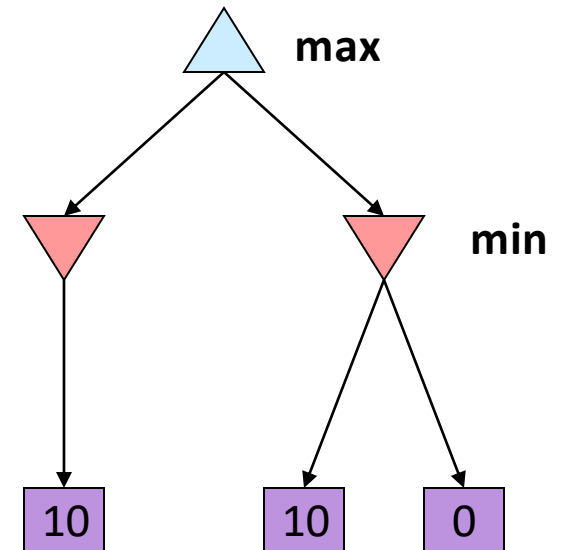
❖MAX version is symmetric

MAX

MIN a

MAX

MIN n
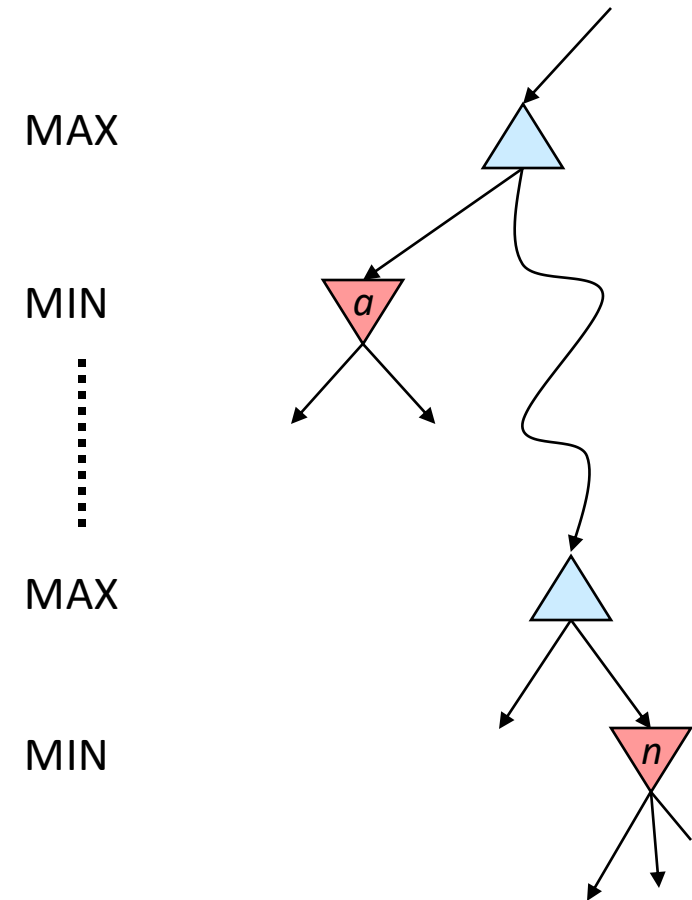
Alpha-Beta Implementation

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

def max-value(state, α, β):
 initialize v = -∞
 for each successor of state:
  v = max(v, min-value(successor, α, β))
  if v ≥ β return v
  α = max(α, v)
 return v

def min-value(state , α, β):
 initialize v = +∞
 for each successor of state:
  v = min(v, max-value(successor, α, β))
  if v ≤ α return v
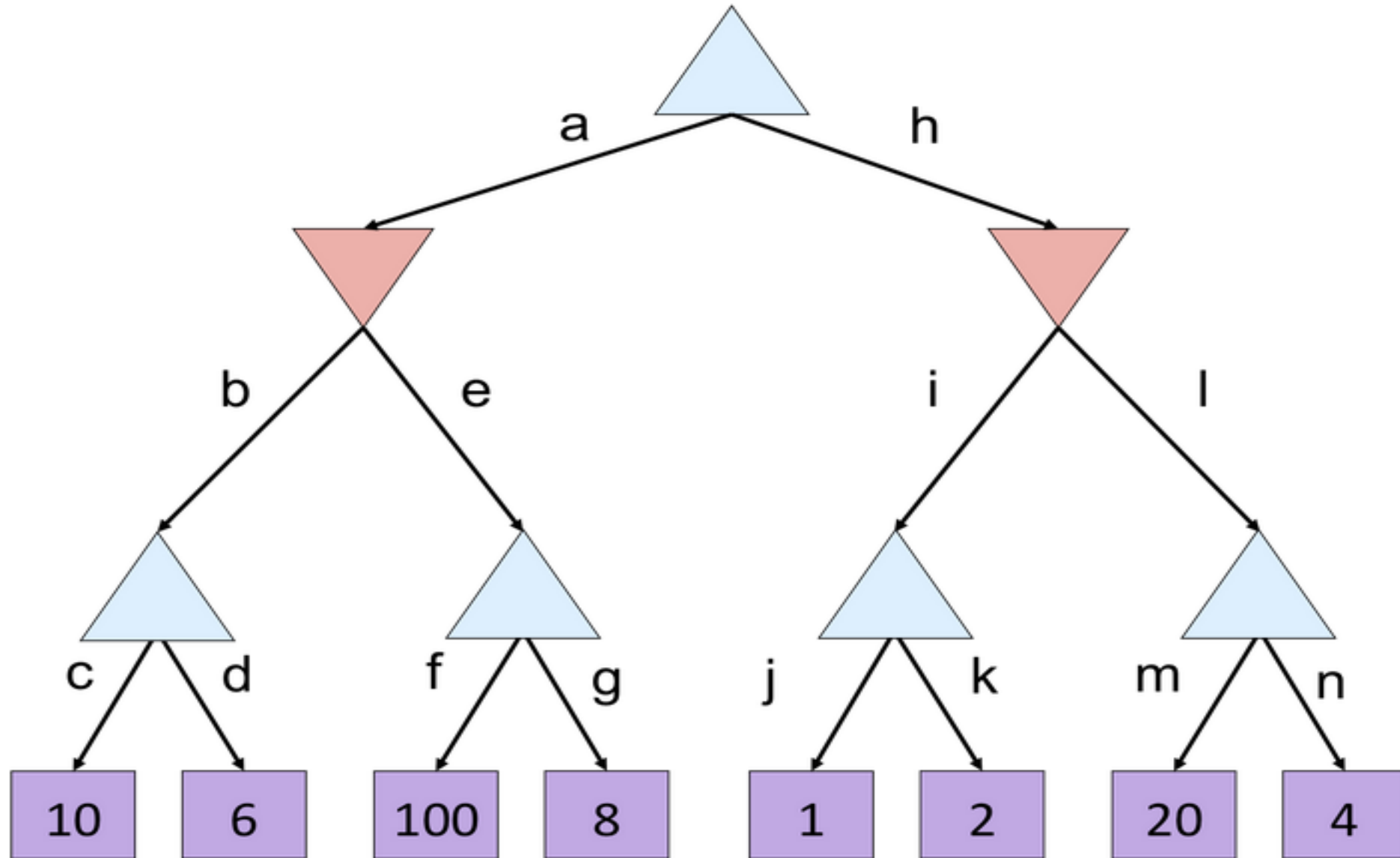 β = min(β, v)
 return v

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 naïve 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)

max

min

10   10   0

Alpha-Beta Quiz

Artificial Intelligence (slide adapted from ai.berkeley.edu)
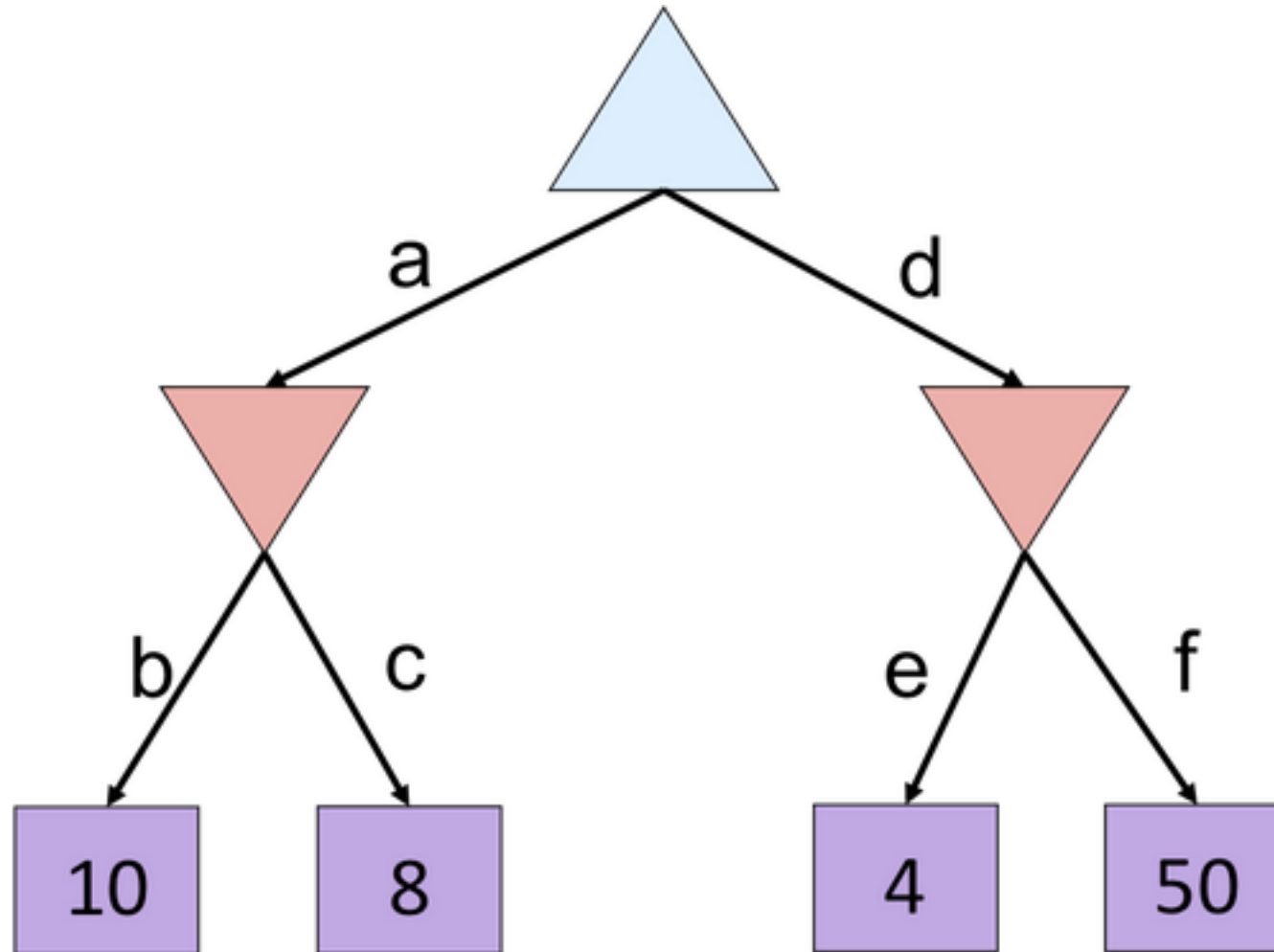
Alpha-Beta Quiz 2

Resource Limits

Artificial Intelligence   (slide adapted from ai.berkeley.edu)

Resource Limits

❖ 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

❖ Use iterative deepening for an anytime algorithm

Video of Demo Thrashing (d=2)

Artificial Intelligence (slide adapted from ai.berkeley.edu)
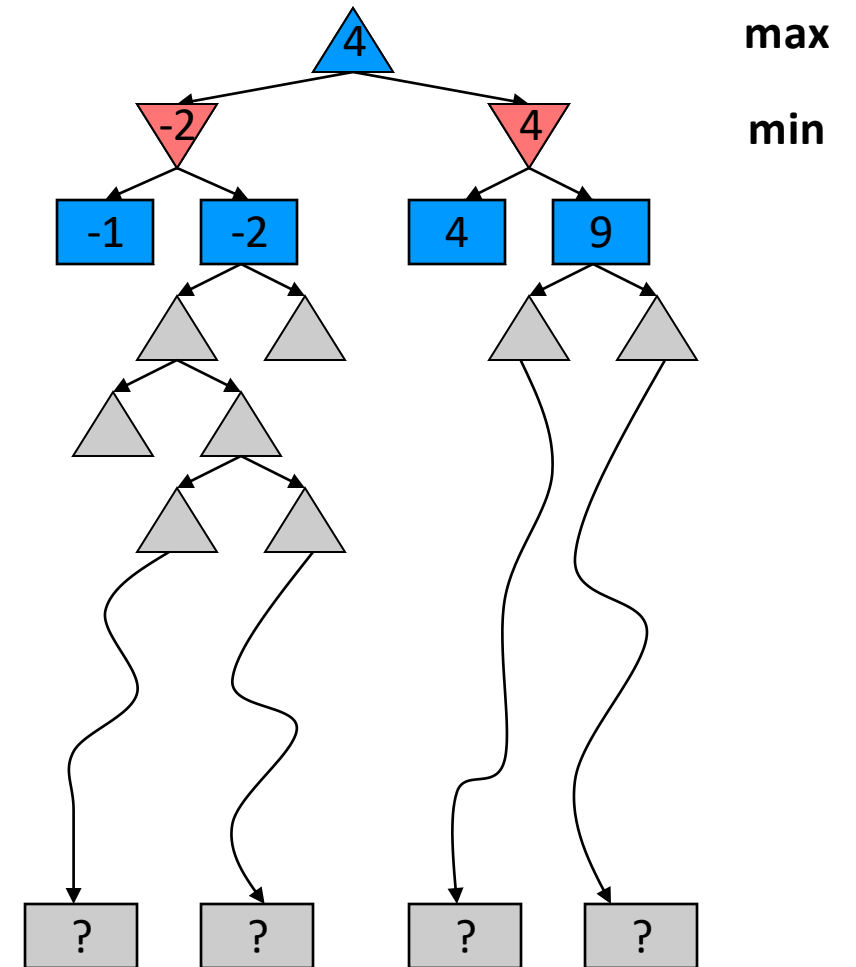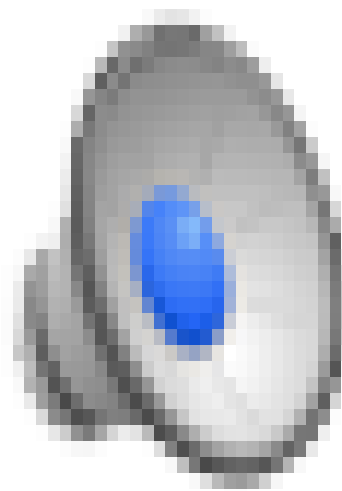
Why Pacman Starves
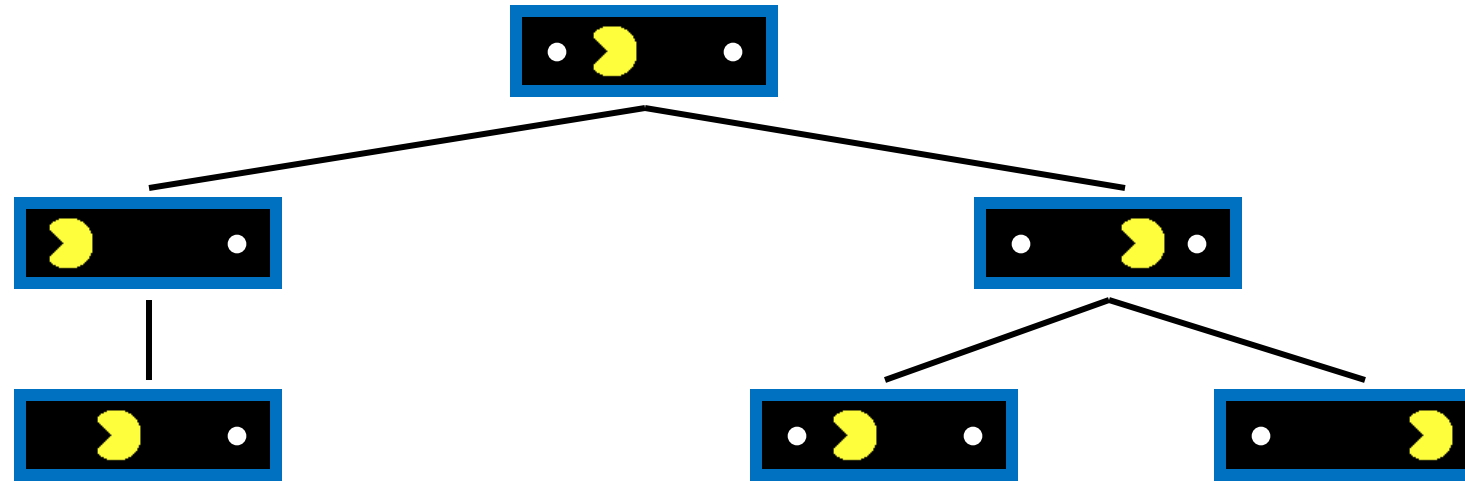
❖A danger of replanning agents!

➢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!
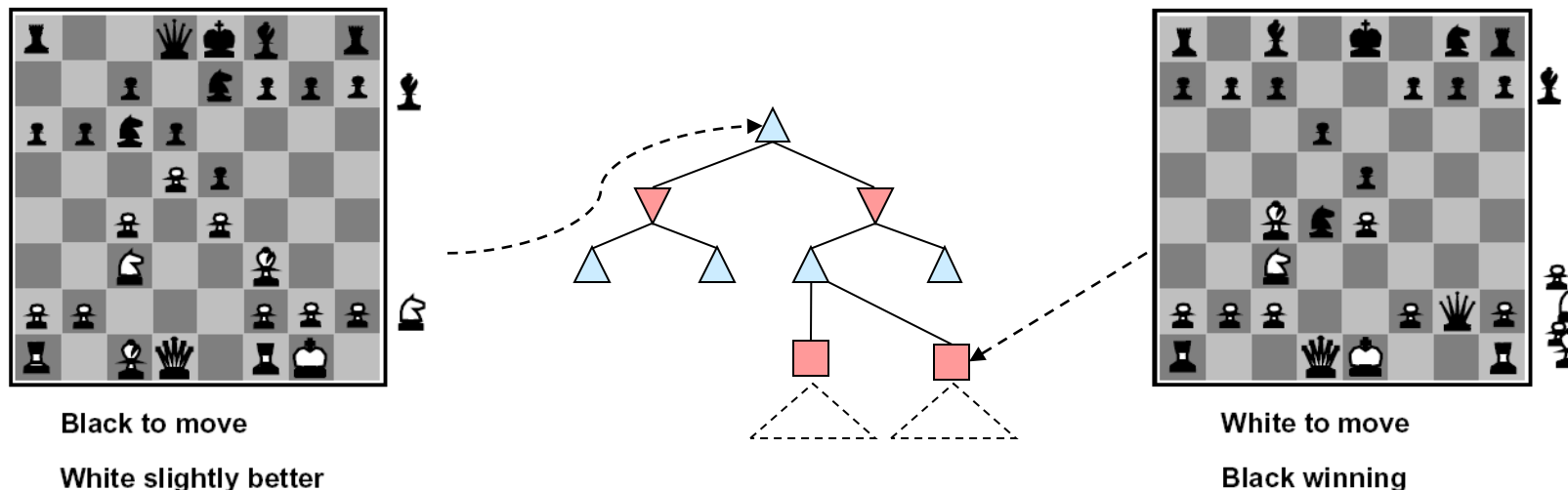
Video of Demo Thrashing -- Fixed (d=2)
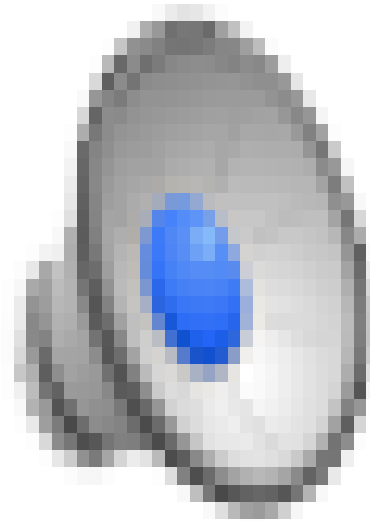
Evaluation Functions

Estimate of the expected utility of the game from the given position

Evaluation Functions

❖ Evaluation functions score non-terminals in depth-limited search

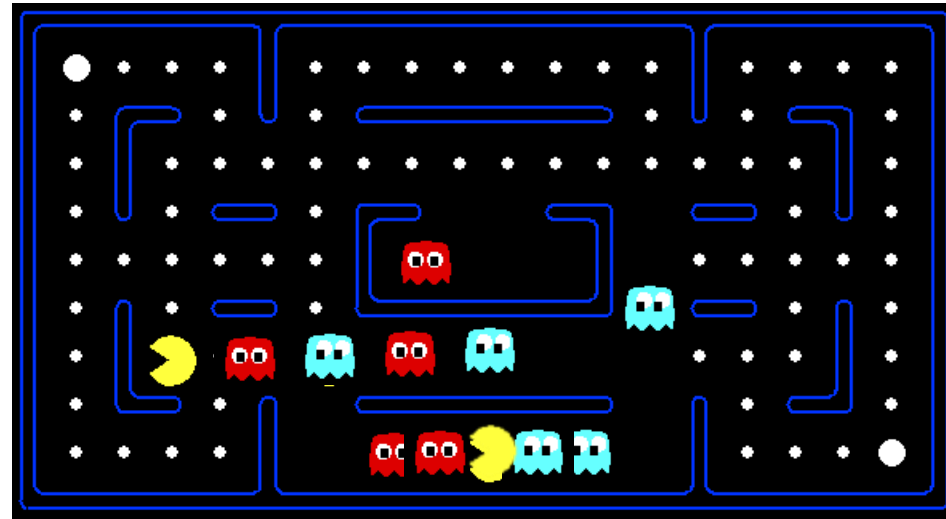Black to move

White slightly better

White to move

Black winning

❖ Ideal function: returns the actual minimax value of the position
❖ In practice: typically weighted linear sum of features:
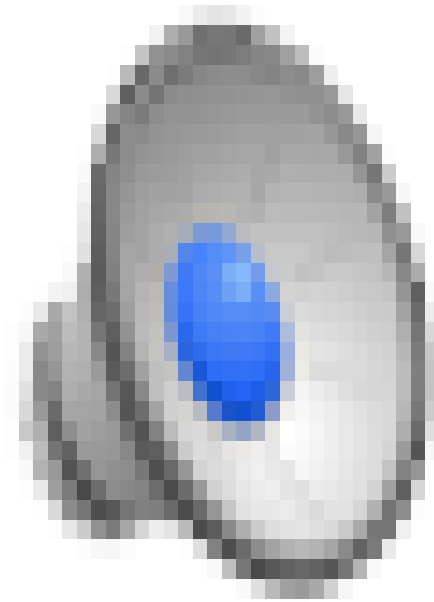
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

❖ e.g. $f_1(s)$ = (num white queens − num black queens), etc.

Evaluation for Pacman

Video of Demo Smart Ghosts (Coordination) – Zoomed In

Artificial Intelligence   (slide adapted from ai.berkeley.edu)

Depth Matters

❖Evaluation functions are always imperfect

❖The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters

❖An important example of the tradeoff between complexity of features and complexity of computation
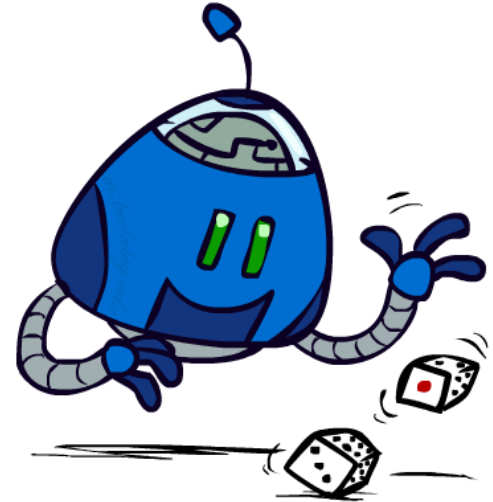
Video of Demo Limited Depth (2)
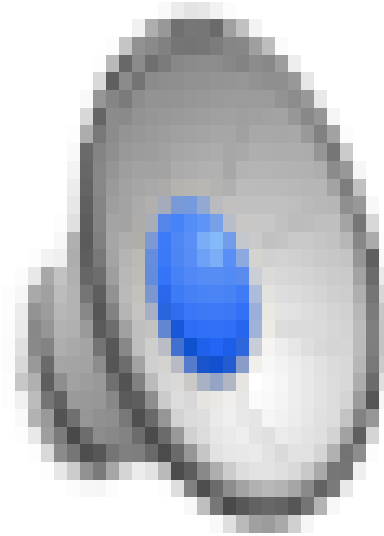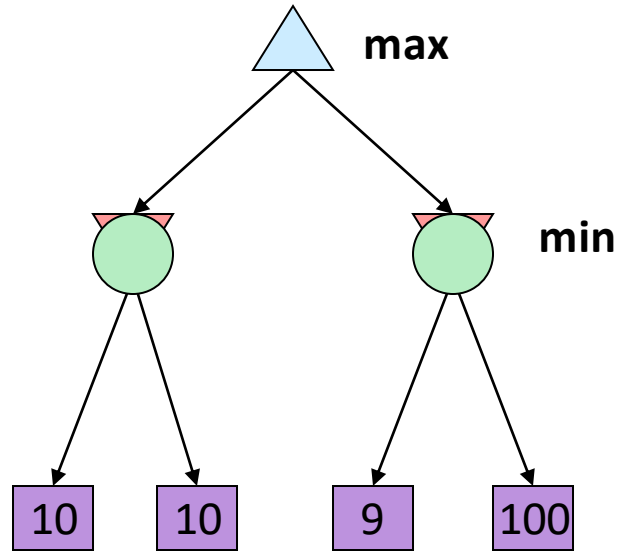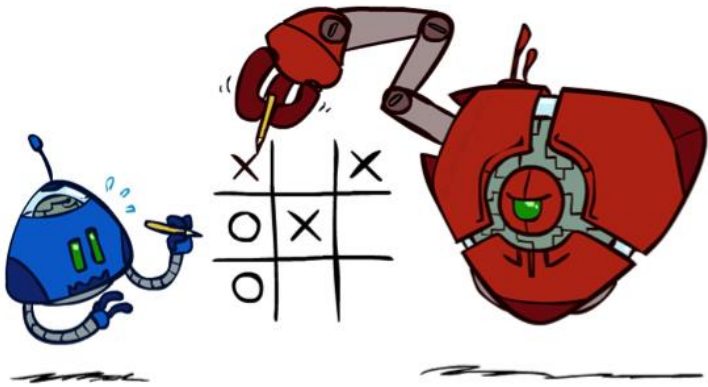
Video of Demo Limited Depth (10)
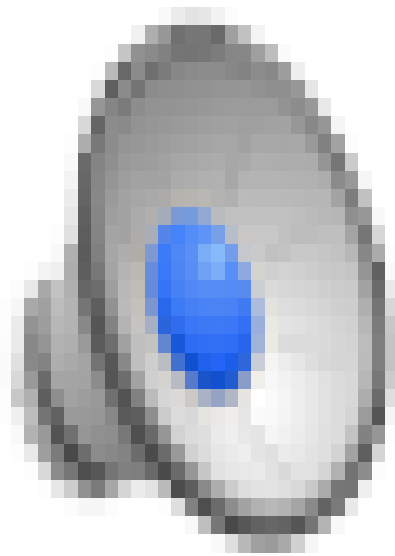
Uncertainty and Utilities

Uncertain Outcomes

# Worst-Case vs. Average Case

Idea: Uncertain outcomes controlled by chance, not an adversary!

Expectimax Search

❖ Why wouldn't we know what the result of an action will be?
  ➢ Explicit randomness: rolling dice
  ➢ Unpredictable opponents: the ghosts respond randomly
  ➢ Actions can fail: when moving a robot, wheels might slip

❖ Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes

❖ Expectimax search: compute the average score under optimal play
  ➢ Max nodes as in minimax search
  ➢ Chance nodes are like min nodes but the outcome is uncertain
  ➢ Calculate their expected utilities
  ➢ I.e. take weighted average (expectation) of children

❖ Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes
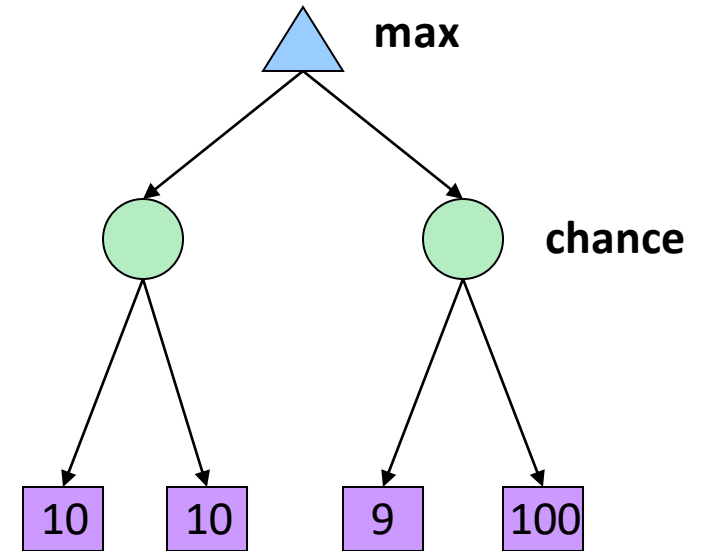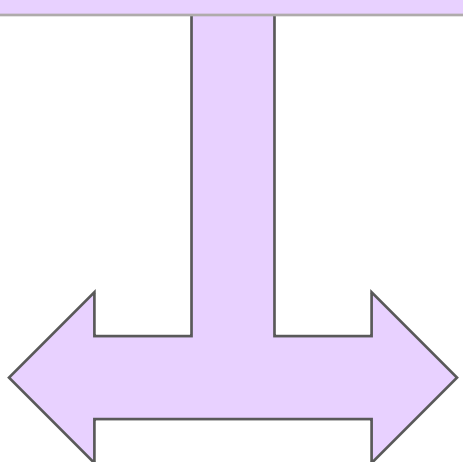
Video of Demo Minimax vs Expectimax (Min)
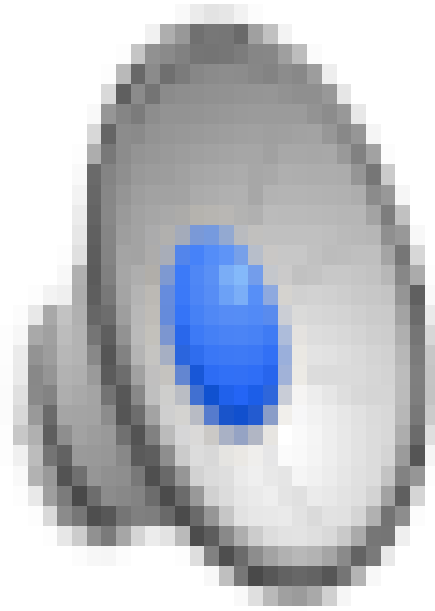
Video of Demo Minimax vs Expectimax (Exp)
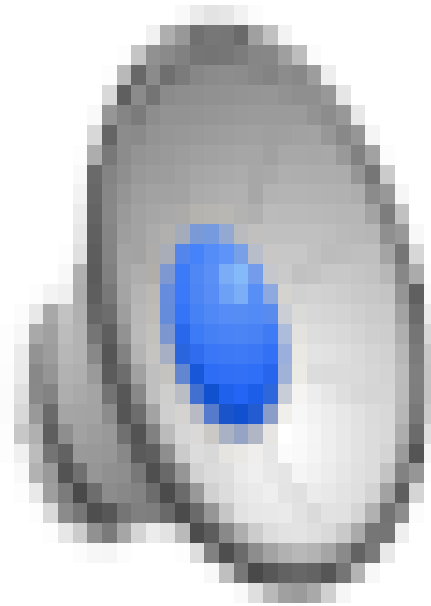
Expectimax Pseudocode

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 EXP: return exp-value(state)

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

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
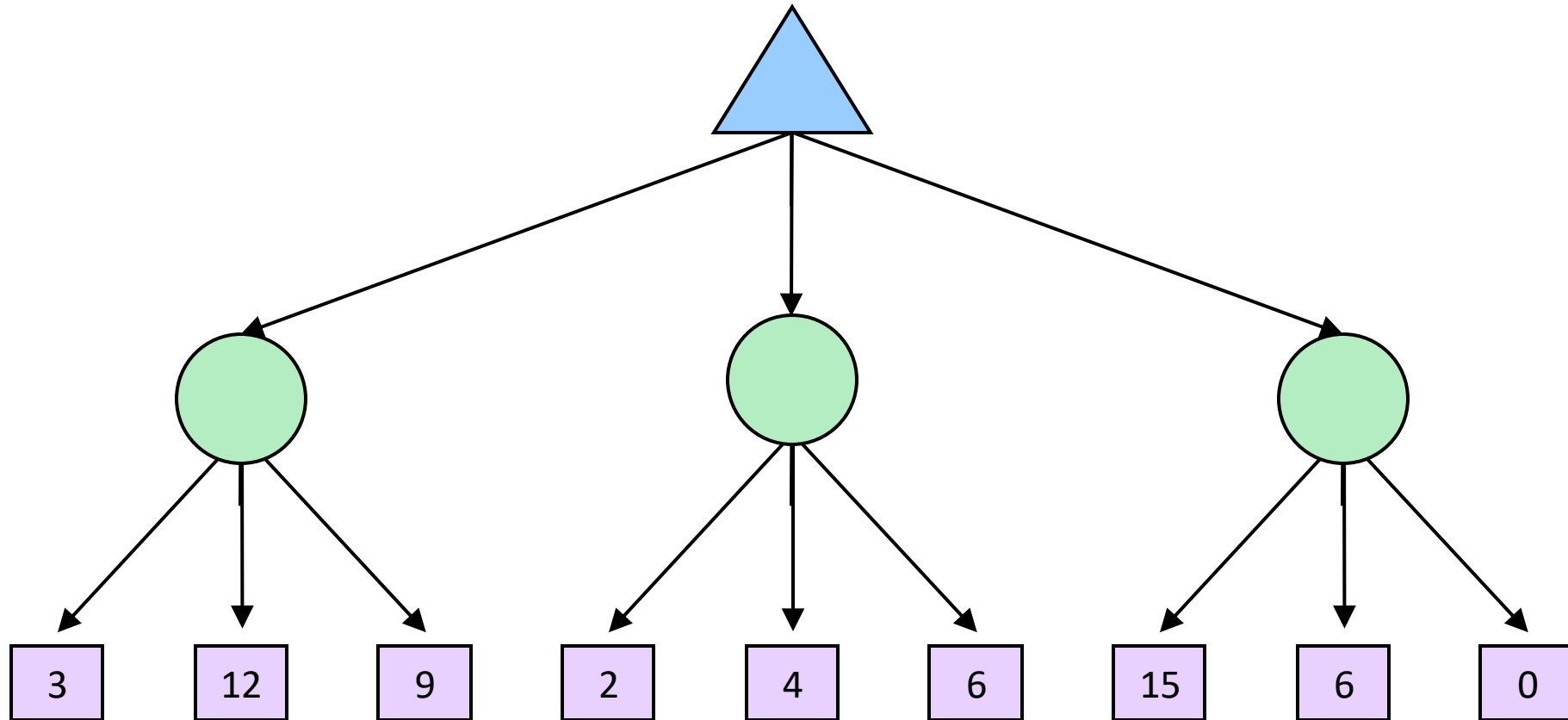   v += p * value(successor)
    return v

Expectimax Pseudocode

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
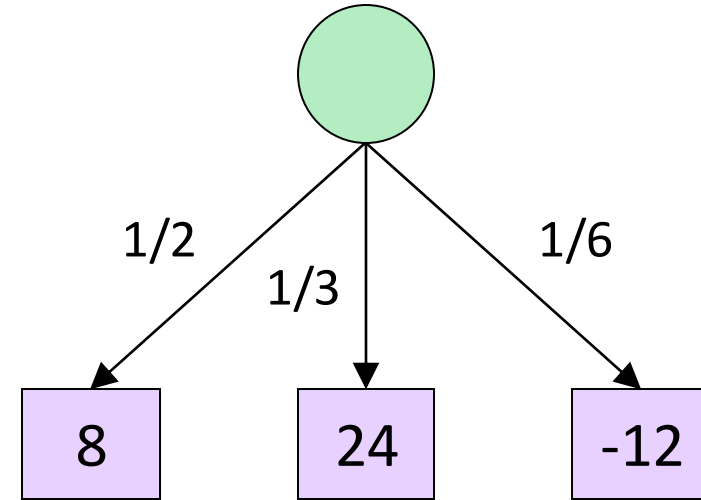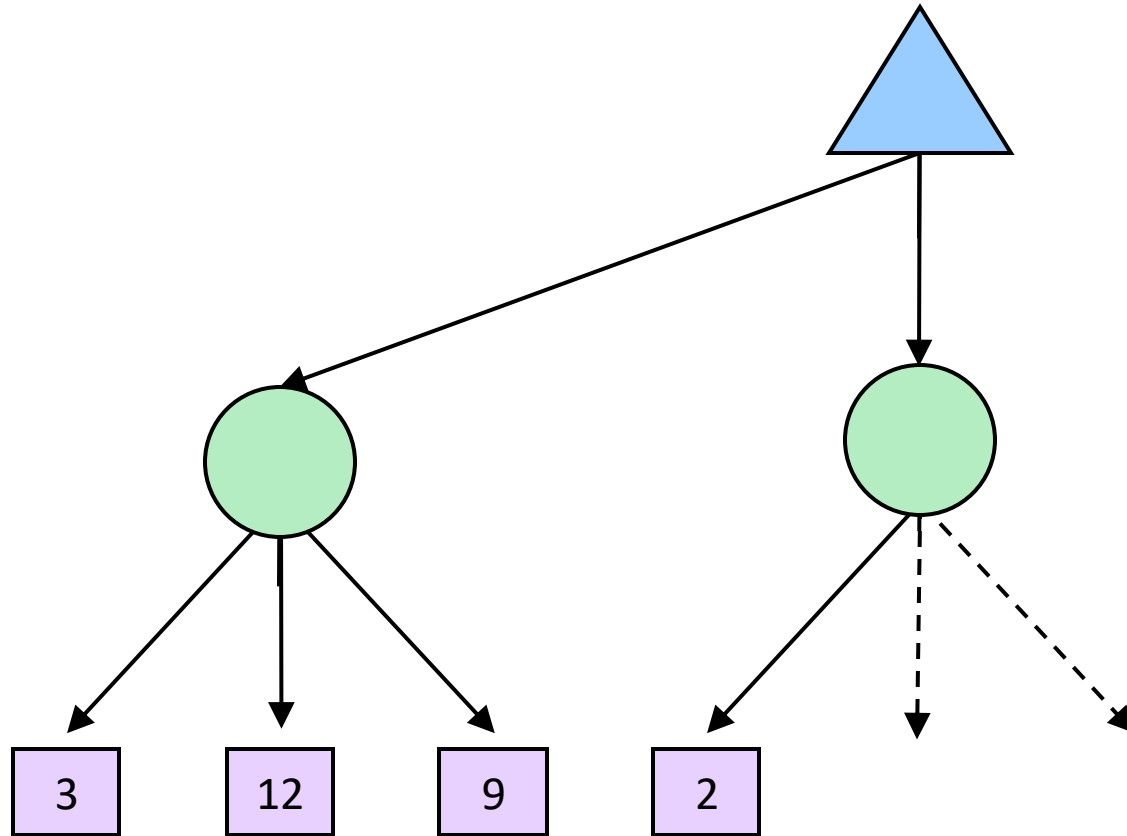

$$v = (1/2)\ (8) + (1/3)\ (24) + (1/6)\ (-12) = 10$$
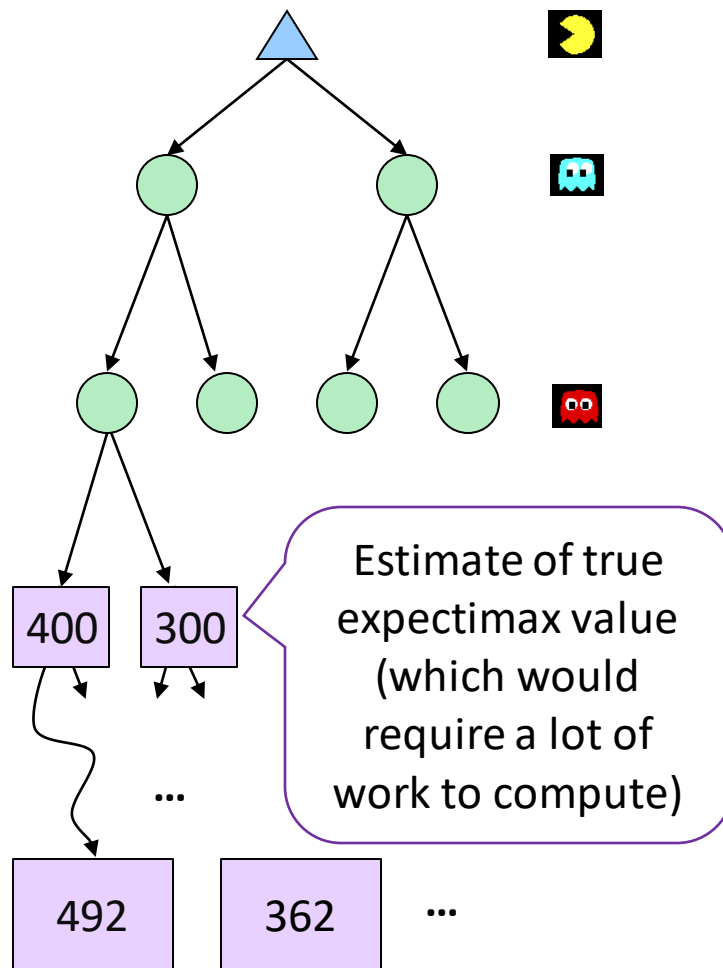
# Expectimax Example

Expectimax Pruning?

Does not work....

Depth-Limited Expectimax

Estimate of true expectimax value (which would require a lot of work to compute)

400 300

492 362 ...

Probabilities

Probabilities 101

❖ 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=none) = 0.25, P(T=light) = 0.50, P(T=heavy) = 0.25

❖ 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=heavy) = 0.25, P(T=heavy | Hour=8am) = 0.60
  ➢ We'll talk about methods for reasoning and updating probabilities later
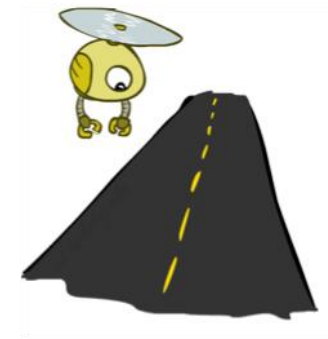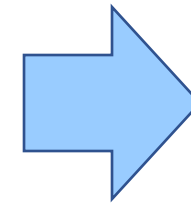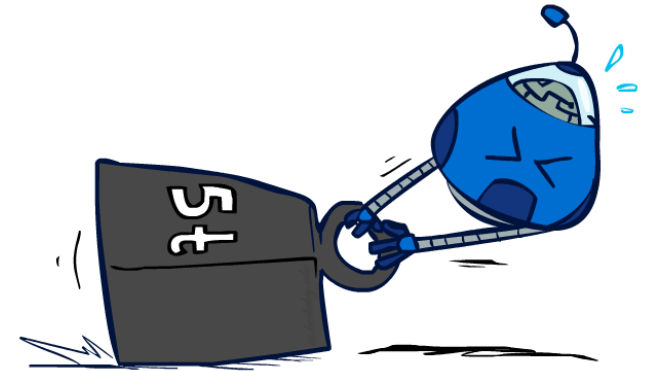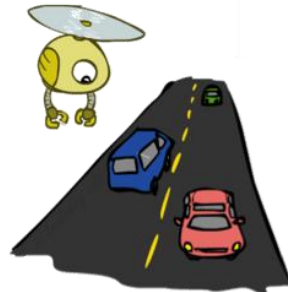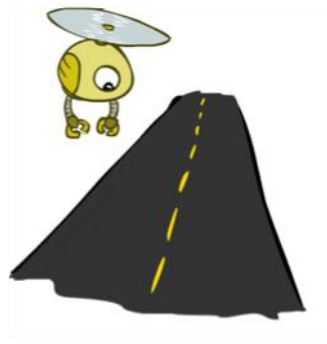
0.25

0.50

0.25

Expectations

❖The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes

❖Example: How long to get to the airport?

| Time:	20 min		30 min		60 min		
	x	+	x	+	x	→	35 min
Probability:	0.25		0.50		0.25		

What Probabilities to Use?

❖ 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 chance node for any outcome out of our control: opponent or environment
  ➢ The model might say that adversarial actions are likely!

❖ For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes
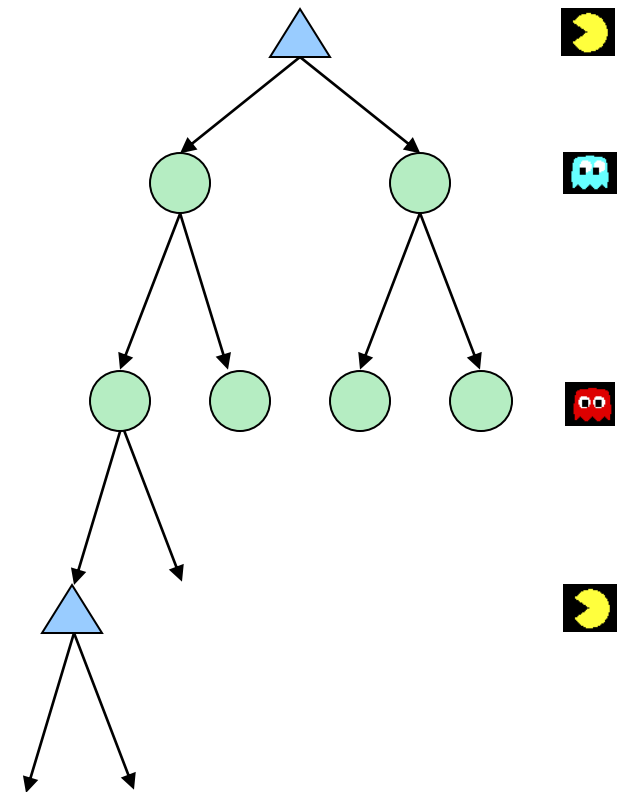
Having a probabilistic belief about another agent's action does not mean that the agent is flipping any coins!

Quiz: Informed Probabilities

❖ Let's say you know that your opponent is running a depth 2 minimax, using the result 80% of the time, and moving randomly otherwise

❖ Question: What tree search should you use?

0.1   0.9
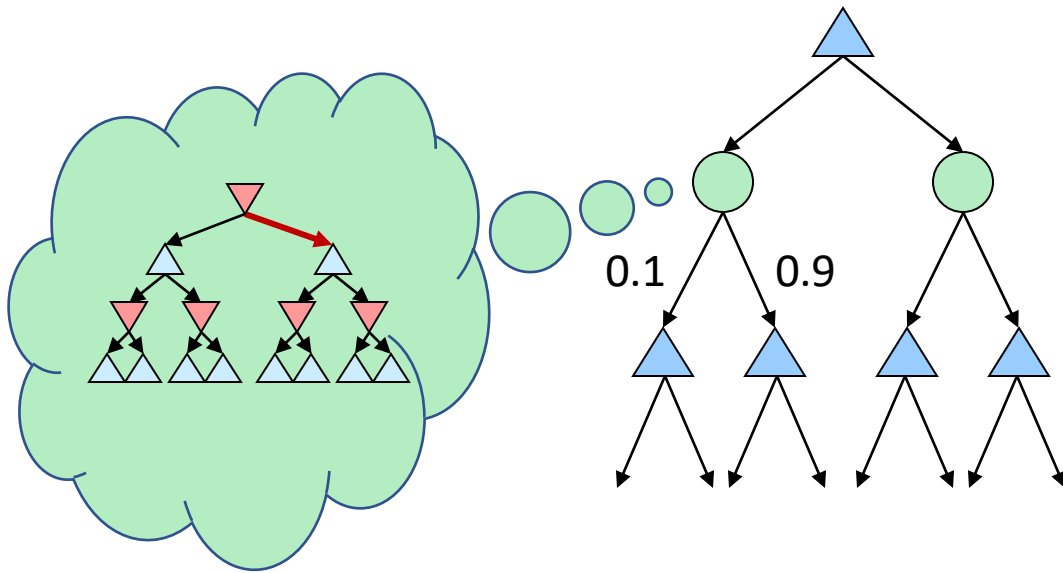
■ Answer: Expectimax!

  ■ To figure out EACH chance node's probabilities, you have to run a simulation of your opponent

  ■ This kind of thing gets very slow very quickly

  ■ Even worse if you have to simulate your opponent simulating you…

  ■ … except for minimax, which has the nice property that it all collapses into one game tree

Modeling Assumptions
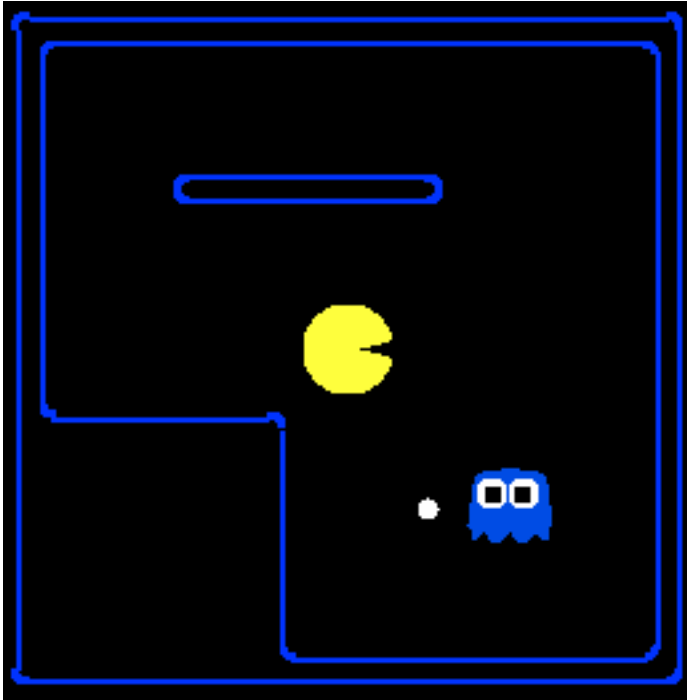
The Dangers of Optimism and Pessimism

Dangerous Optimism
Assuming chance when the world is adversarial

Dangerous Pessimism
Assuming the worst case when it's not likely

Assumptions vs. Reality

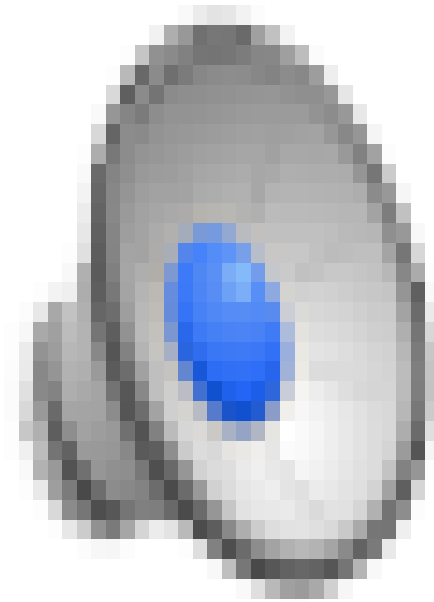|	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

Results from playing 5 games

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman
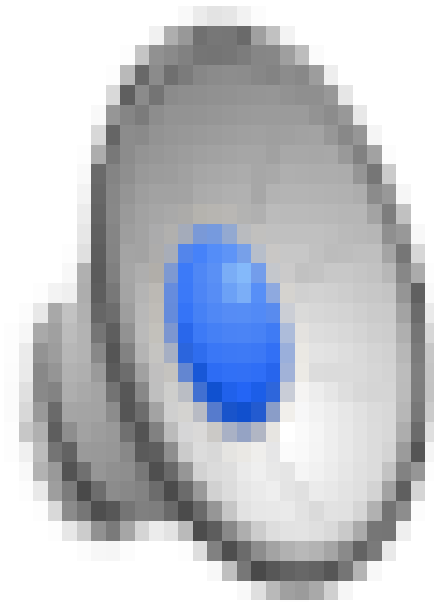
[Demos: world assumptions (L7D3,4,5,6)]

Video of Demo World Assumptions
Random Ghost – Expectimax Pacman
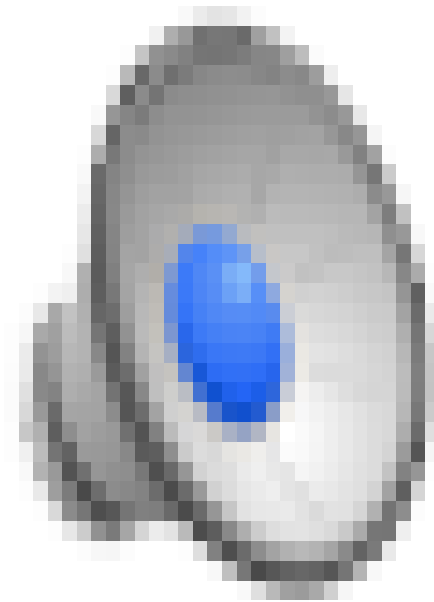
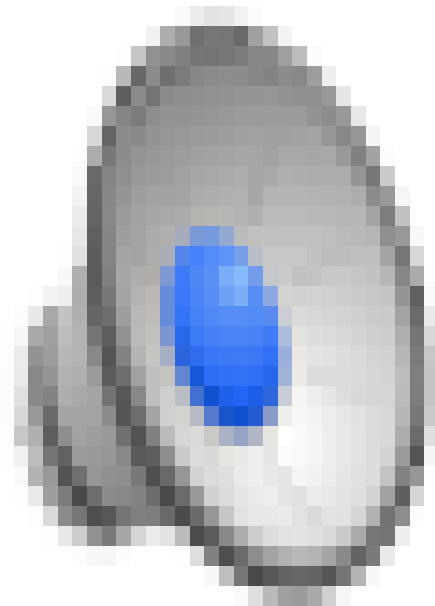# Video of Demo World Assumptions Adversarial Ghost – Minimax Pacman

Video of Demo World Assumptions
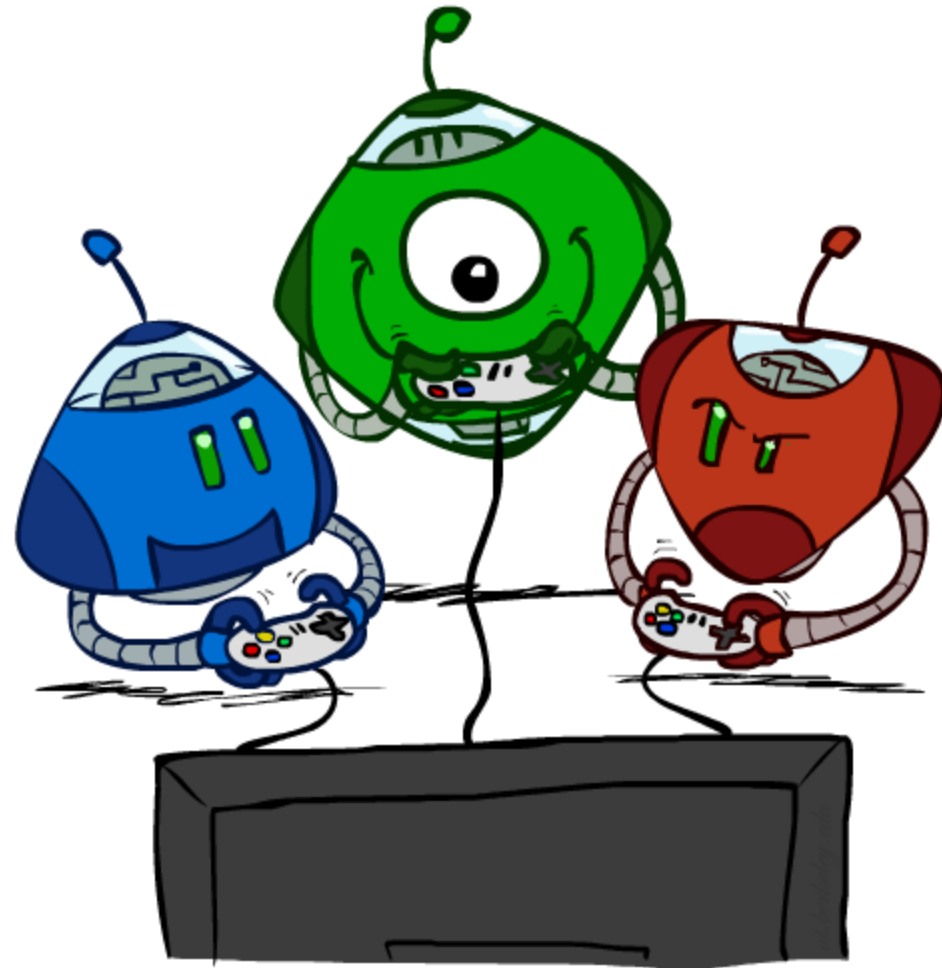# Adversarial Ghost – Expectimax Pacman

Video of Demo World Assumptions
Random Ghost – Minimax Pacman
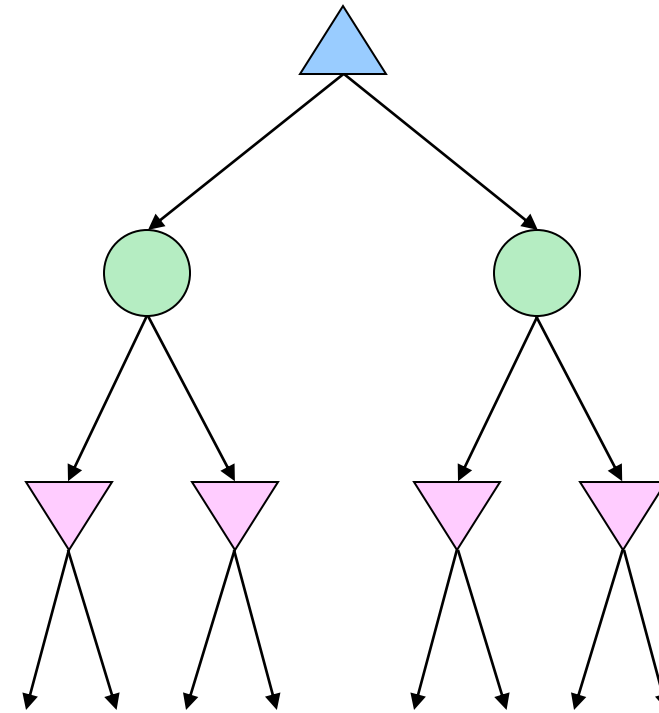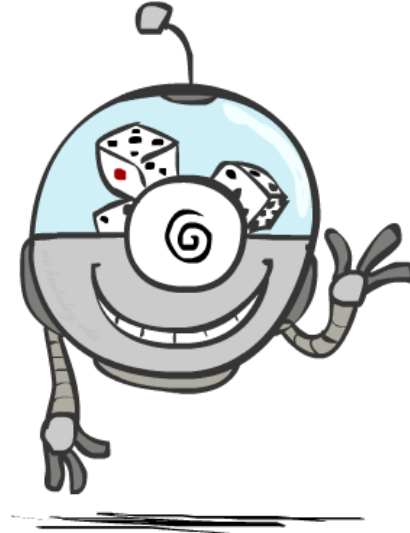
Other Game Types

Mixed Layer Types

❖ E.g. Backgammon

❖ Expectiminimax

  ➢ Environment is an extra "random agent" player that moves after each min/max agent

  ➢ Each node computes the appropriate combination of its children

Example: Backgammon

❖ Dice rolls increase b: 21 possible rolls with 2 dice
  ➤ Backgammon ≈ 20 legal moves
  ➤ Depth 2 = 20 x (21 x 20)^3 = 1.2 x 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...

❖ Historic AI: TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play

❖ 1st AI world champion in any game!
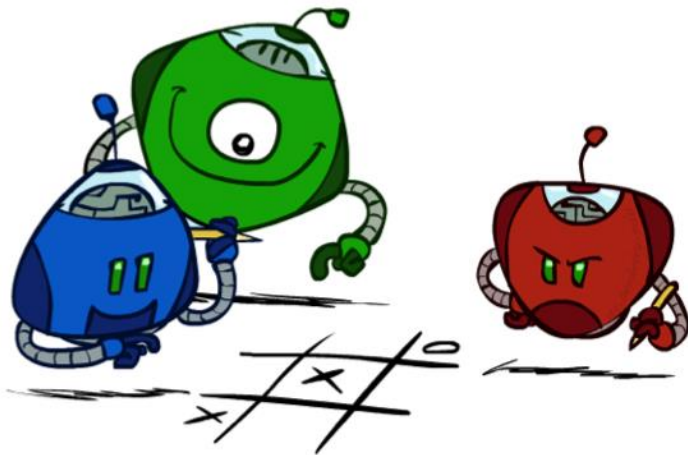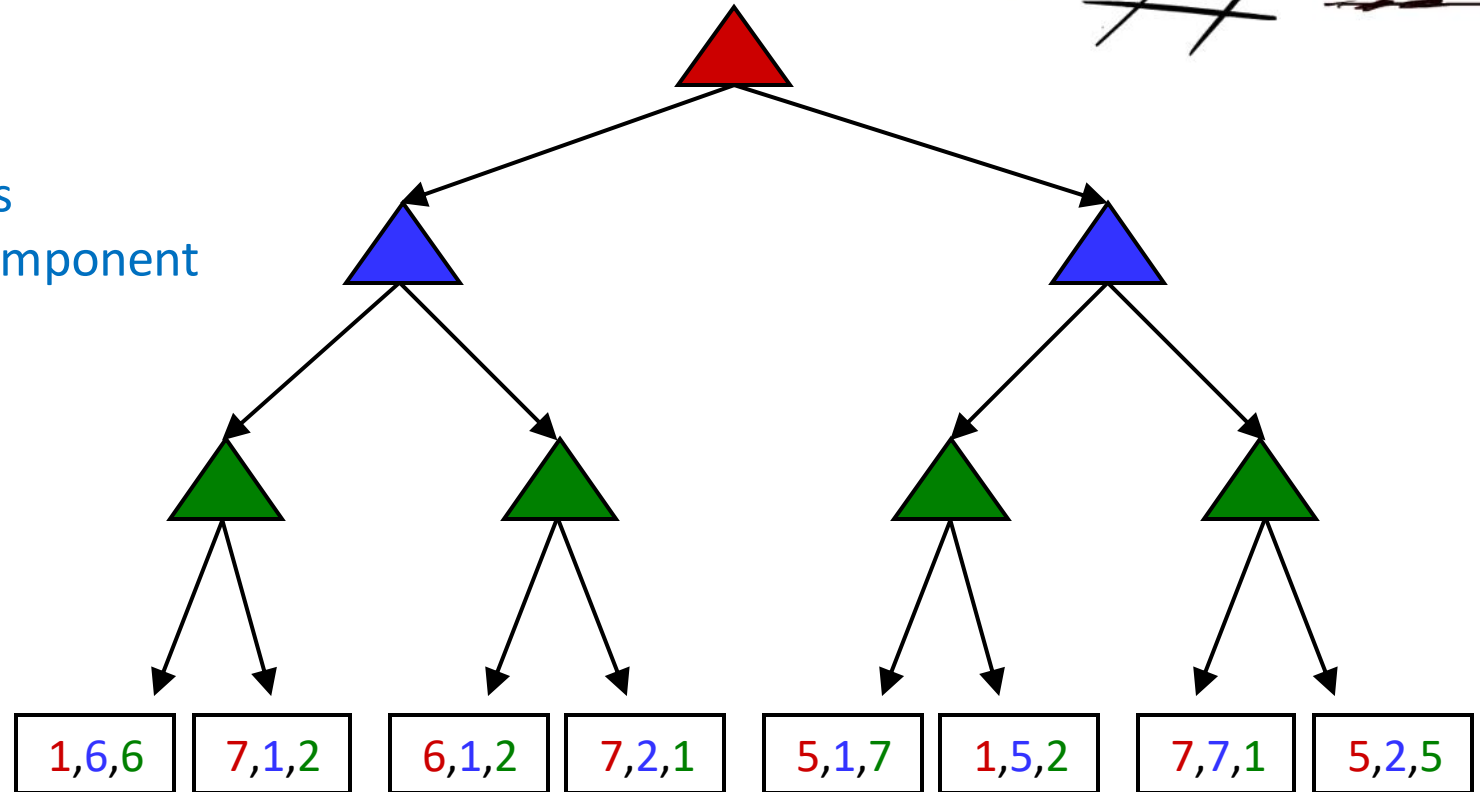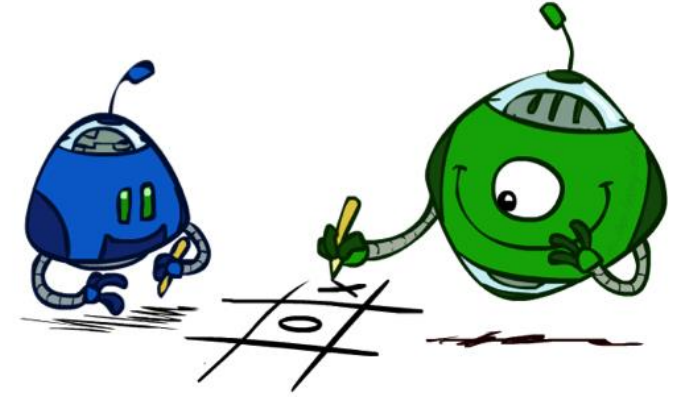

Image: Wikipedia

Multi-Agent Utilities

❖ What if the game is not zero-sum, or has multiple players?

❖ Generalization of minimax:
  ➢ Terminals have utility tuples
  ➢ Node values are also utility tuples
  ➢ Each player maximizes its own component
  ➢ Can give rise to cooperation and competition dynamically…

Utilities
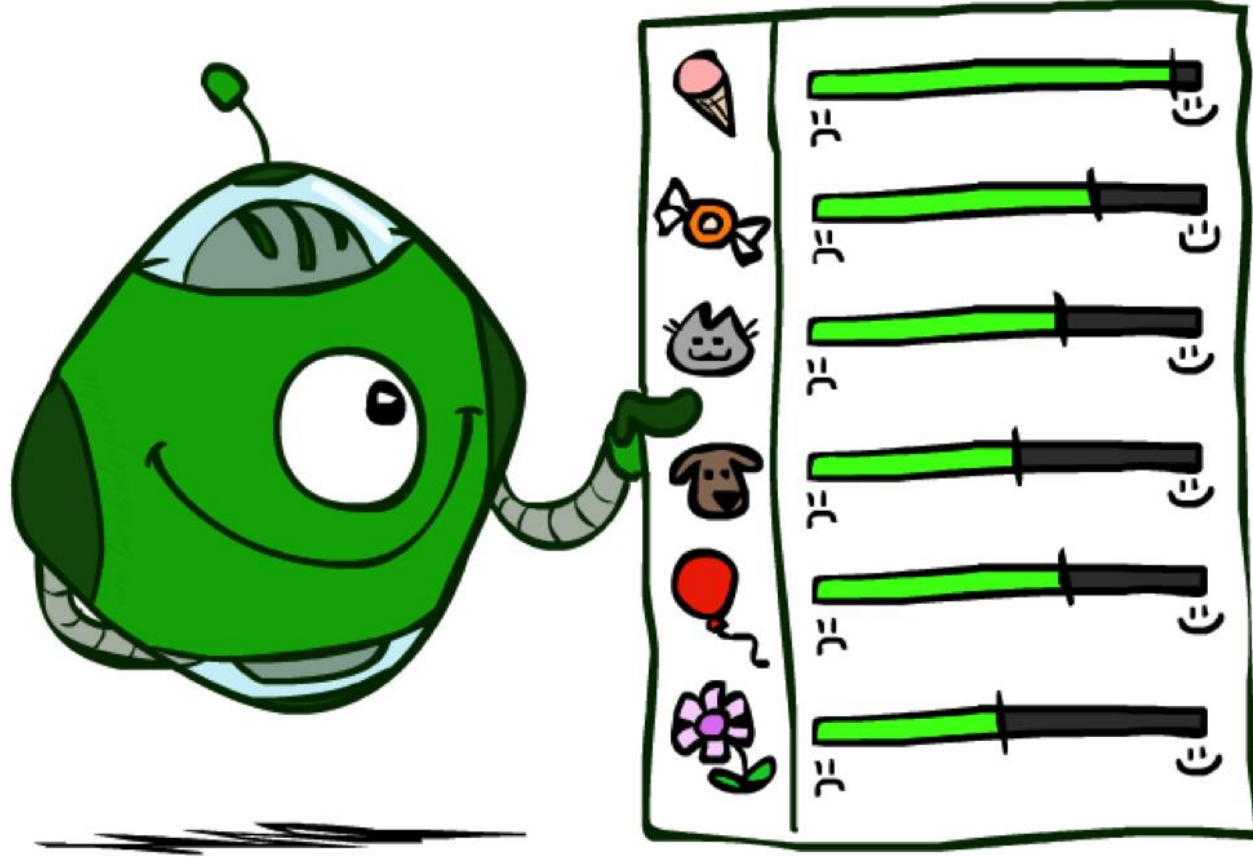
Maximum Expected Utility

❖ Why should we average utilities? Why not minimax?

❖ Principle of maximum expected utility:
  ➢ A rational agent should chose the action that maximizes its expected utility, given its knowledge

❖ Questions:
  ➢ Where do utilities come from?
  ➢ How do we know such utilities even exist?
  ➢ How do we know that averaging even makes sense?
  ➢ What if our behavior (preferences) can't be described by utilities?

What Utilities to Use?
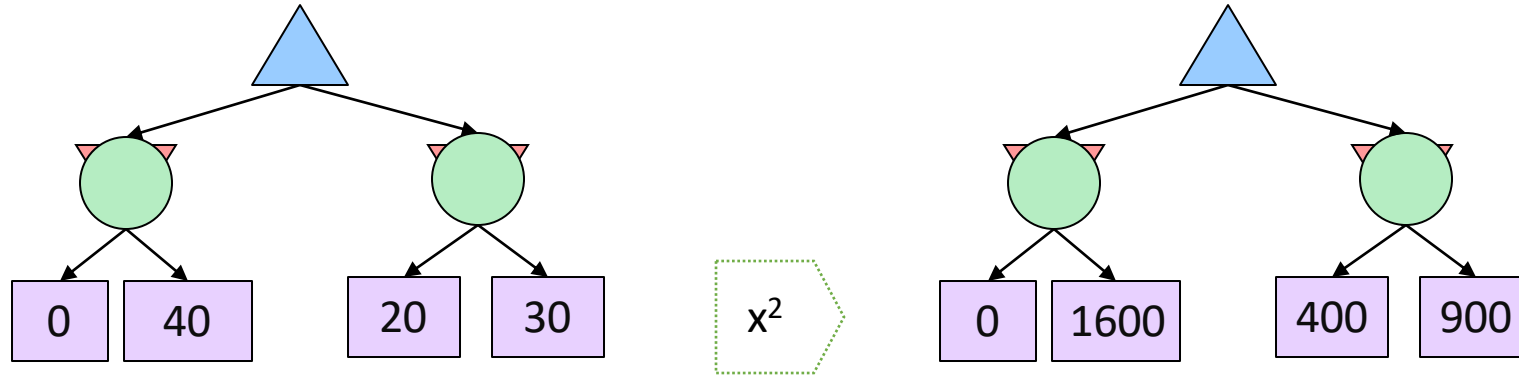
❖For worst-case minimax reasoning, terminal function scale doesn't matter
  ➢We just want better states to have higher evaluations (get the ordering right)
  ➢We call this insensitivity to monotonic transformations

❖For average-case expectimax reasoning, we need *magnitudes* to be meaningful

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 "rational" preferences can be summarized as a utility function

❖ We hard-wire utilities and let behaviors emerge
  ➢ Why don't we let agents pick utilities?
  ➢ Why don't we prescribe behaviors?

Utilities: Uncertain Outcomes

Getting ice cream

Get Single

Get Double

Oops
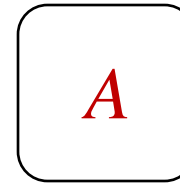
Whew!
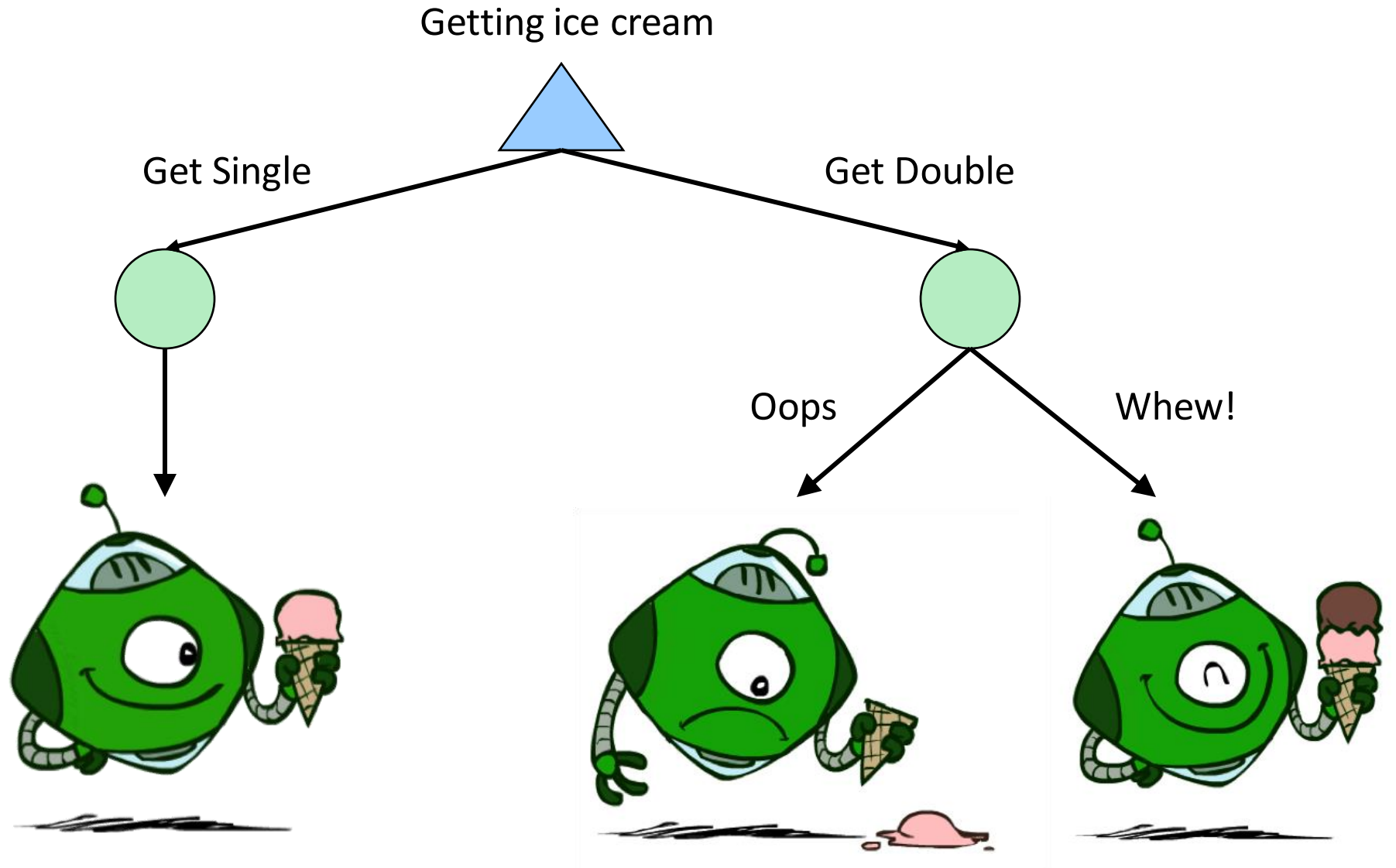
Preferences

❖ An agent must have preferences among:

➢ Prizes: A, B, etc.

➢ Lotteries: situations with uncertain prizes
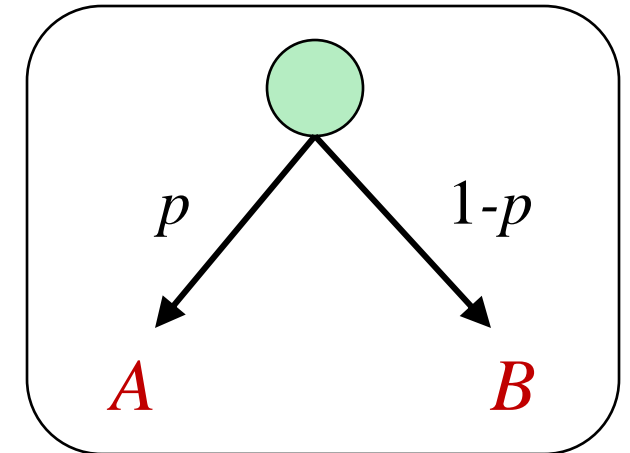
$$L = [p, A; \ (1-p), B]$$

❖ Notation:

➢ Preference:

➢ Indifference:

$A \succ B$

$A \sim B$

A Prize

A

A Lottery

Rationality
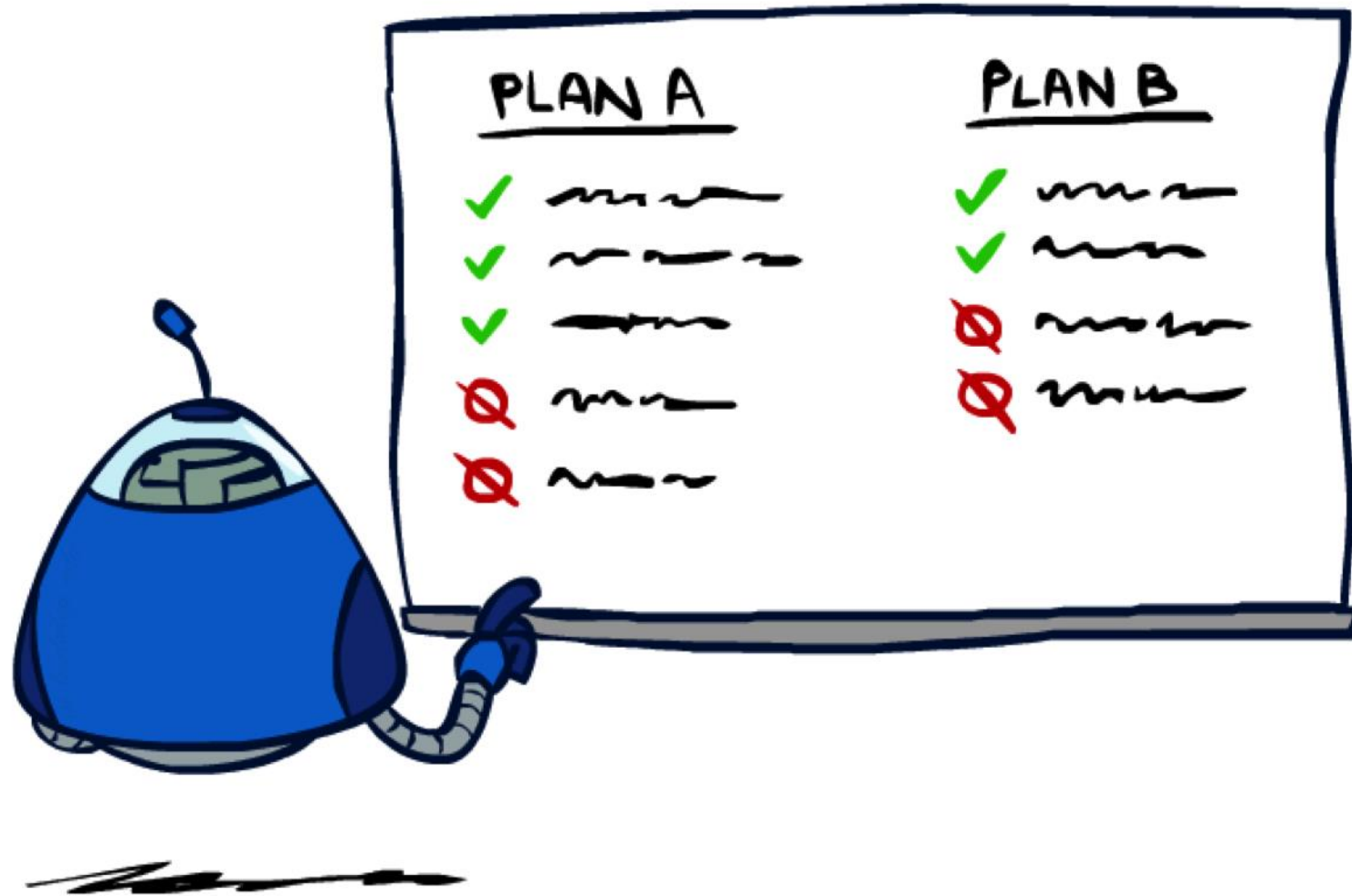
Rational Preferences

❖We want some constraints on preferences before we call them rational, such as:

> Axiom of Transitivity: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

❖For example: an agent with intransitive preferences can be induced to give away all of its money
  - If B > C, then an agent with C would pay (say) 1 cent to get B
  - If A > B, then an agent with B would pay (say) 1 cent to get A
  - If C > A, then an agent with A would pay (say) 1 cent to get C
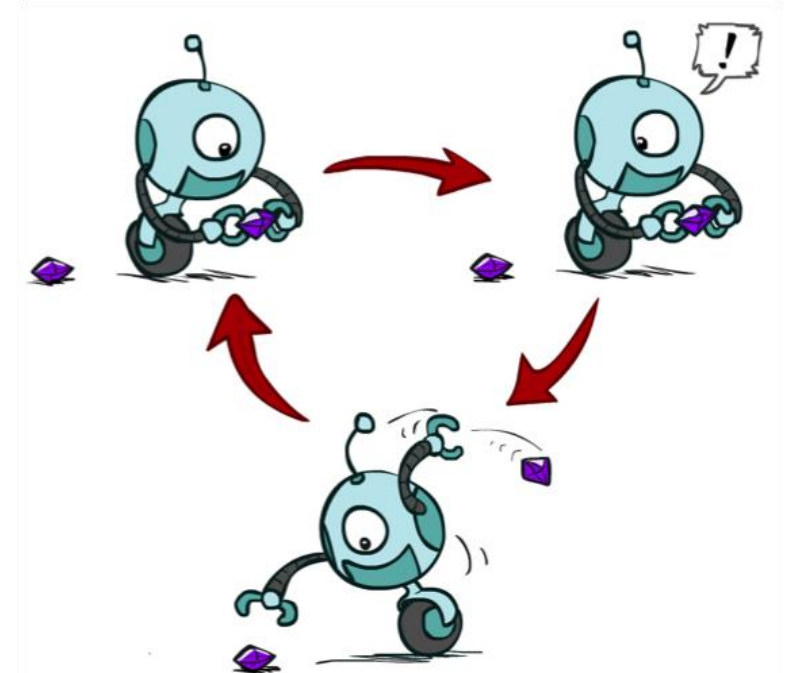
Rational Preferences

The Axioms of Rationality

Orderability
$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$
Transitivity
$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$
Continuity
$$A \succ B \succ C \Rightarrow \exists p \; [p, A; \; 1-p, C] \sim B$$
Substitutability
$$A \sim B \Rightarrow [p, A; \; 1-p, C] \sim [p, B; 1-p, C]$$
Monotonicity
$$A \succ B \Rightarrow$$
$$(p \geq q \Leftrightarrow [p, A; \; 1-p, B] \succeq [q, A; \; 1-q, B])$$

Theorem: Rational preferences imply behavior describable as maximization of expected utility

MEU Principle

❖Theorem [Ramsey, 1931; von Neumann & Morgenstern, 1944]
 ➢Given any preferences satisfying these constraints, there exists a real-valued function U such that:

$$U(A) \geq U(B) \ \Leftrightarrow \ A \succeq B$$

$$U([p_1, S_1; \ \ldots \ ; \ p_n, S_n]) = \sum_i p_i U(S_i)$$

 ➢I.e. values assigned by U preserve preferences of both prizes and lotteries!

❖Maximum expected utility (MEU) principle:
 ➢Choose the action that maximizes expected utility
 ➢Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
 ➢E.g., a lookup table for perfect tic-tac-toe, a reflex vacuum cleaner
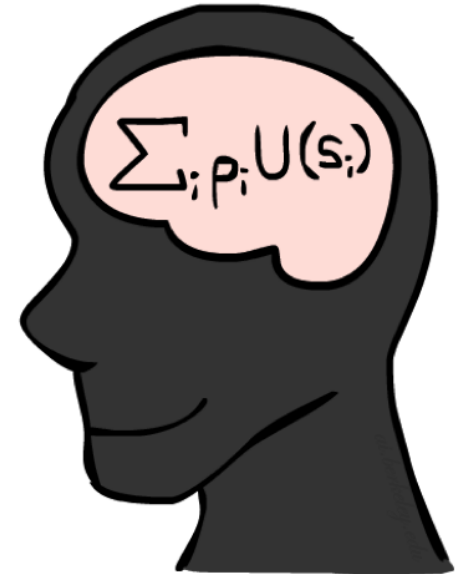
Human Utilities
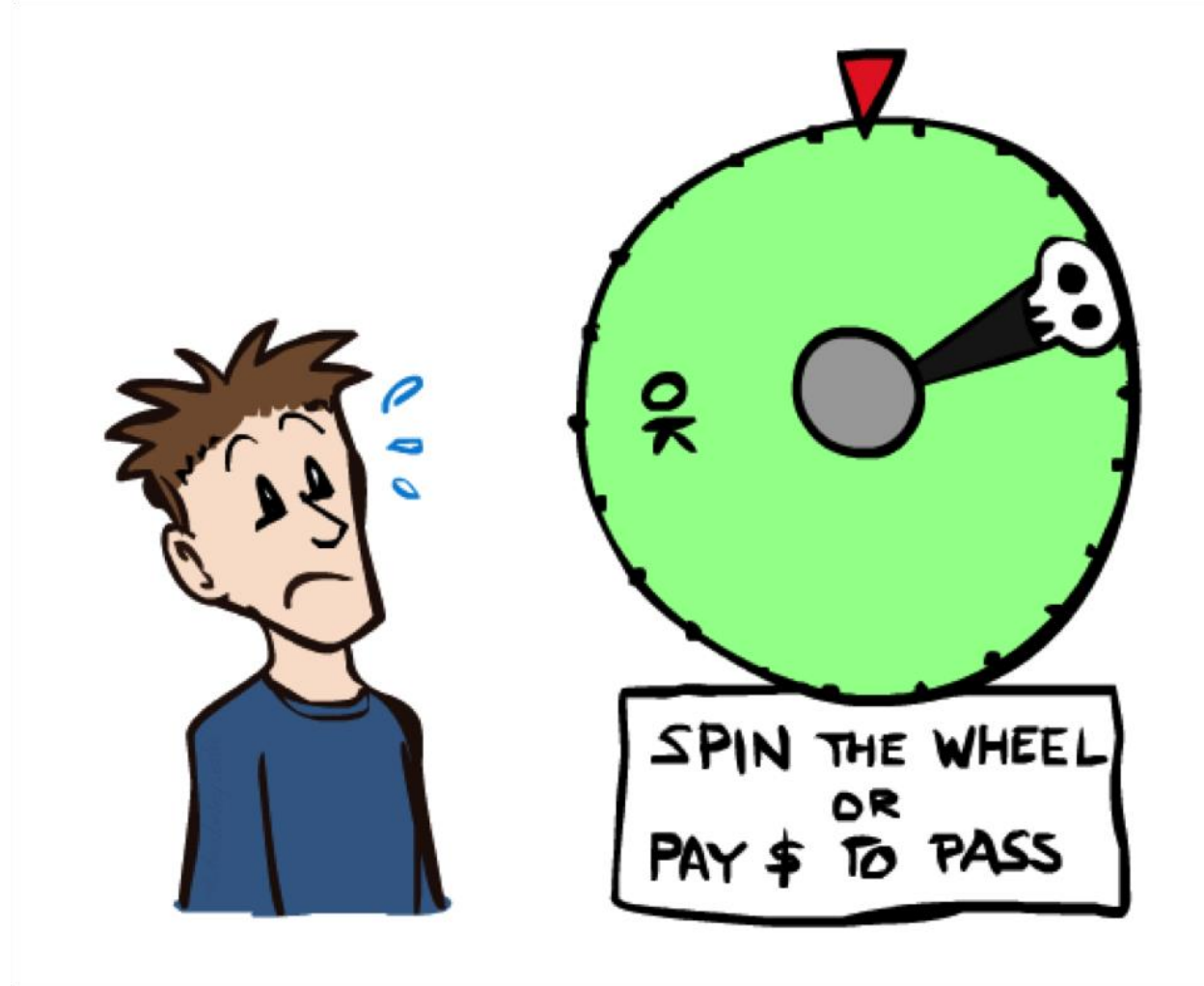
Utility Scales

❖Normalized utilities: $u_+ = 1.0$, $u_- = 0.0$

❖Micromorts: one-millionth chance of death, useful for paying to reduce product risks, etc.

❖QALYs: quality-adjusted life years, useful for medical decisions involving substantial risk

❖Note: behavior is invariant under positive linear transformation

$$U'(x) = k_1 U(x) + k_2 \quad \text{where } k_1 > 0$$

❖With deterministic prizes only (no lottery choices), only ordinal utility can be determined, i.e., total order on prizes

Human Utilities

❖ Utilities map states to real numbers. Which numbers?

❖ Standard approach to assessment (elicitation) of human utilities:

➢ Compare a prize A to a standard lottery L_p between
- "best possible prize" u_+ with probability p
- "worst possible catastrophe" u_- with probability 1-p

➢ Adjust lottery probability p until indifference: A ~ L_p

➢ Resulting p is a utility in [0,1]

Pay $30	~

0.999999   0.000001

No change   *Instant death*
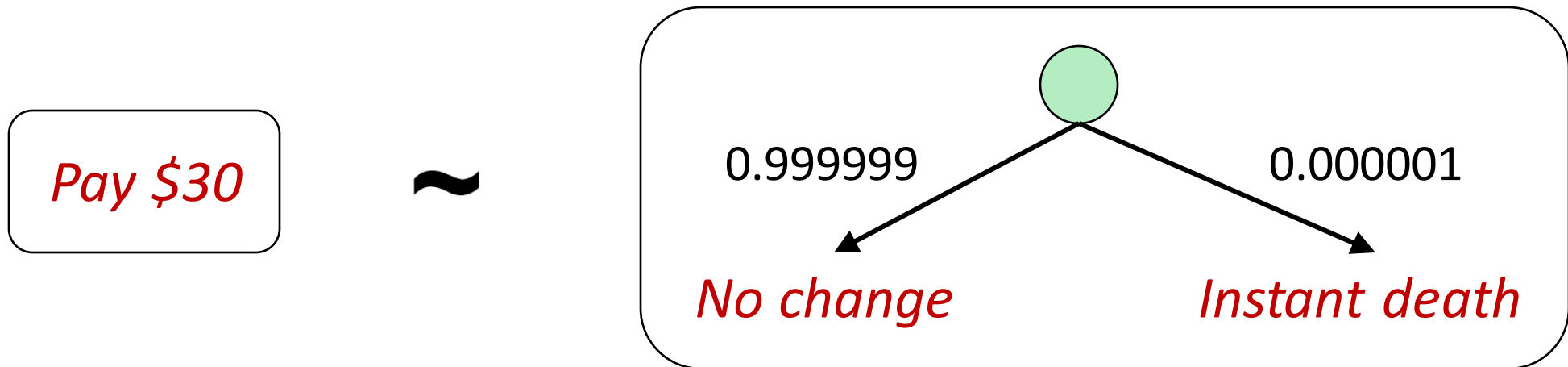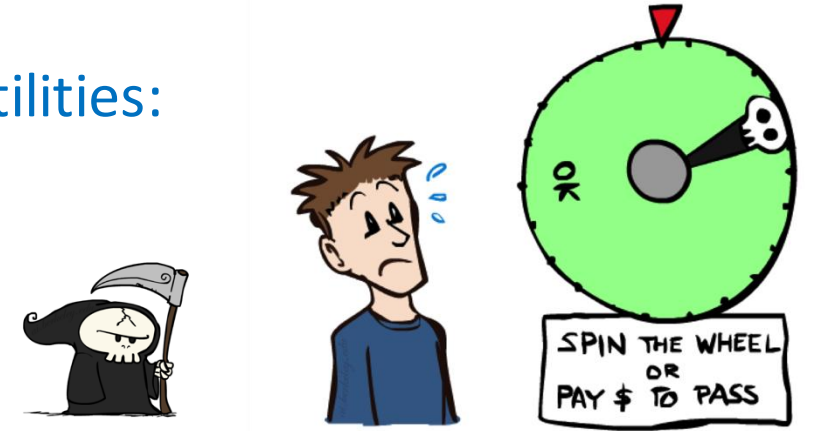
Money

❖ Money <u>does not</u> behave as a utility function, but we can talk about the utility of having money (or being in debt)

❖ Given a lottery L = [p, $X; (1-p), $Y]

➤ The expected monetary value EMV(L) is p*X + (1-p)*Y

➤ U(L) = p*U($X) + (1-p)*U($Y)

➤ Typically, U(L) < U(EMV(L))

➤ In this sense, people are risk-averse

➤ When deep in debt, people are risk-prone
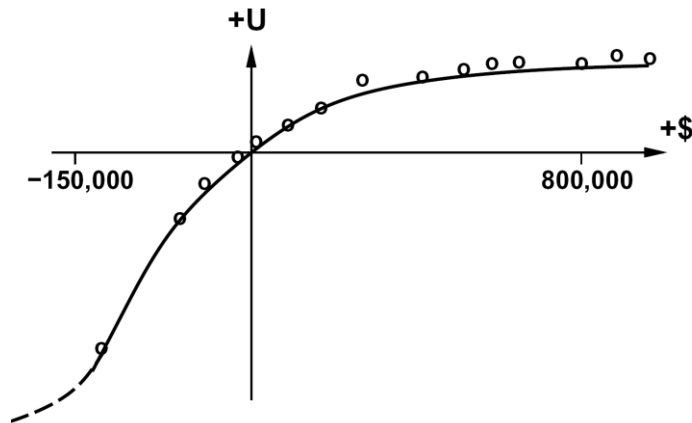
Example: Insurance

❖ Consider the lottery [0.5, $1000; 0.5, $0]
 ➢ What is its expected monetary value? ($500)
  ➢ What is its certainty equivalent?
    ▪ Monetary value acceptable in lieu of lottery
    ▪ $400 for most people
  ➢ Difference of $100 is the insurance premium
    ▪ There's an insurance industry because people will pay to reduce their risk
    ▪ If everyone were risk-neutral, no insurance needed!
  ➢ It's win-win: you'd rather have the $400 and the insurance company would rather have the lottery (their utility curve is flat and they have many lotteries)
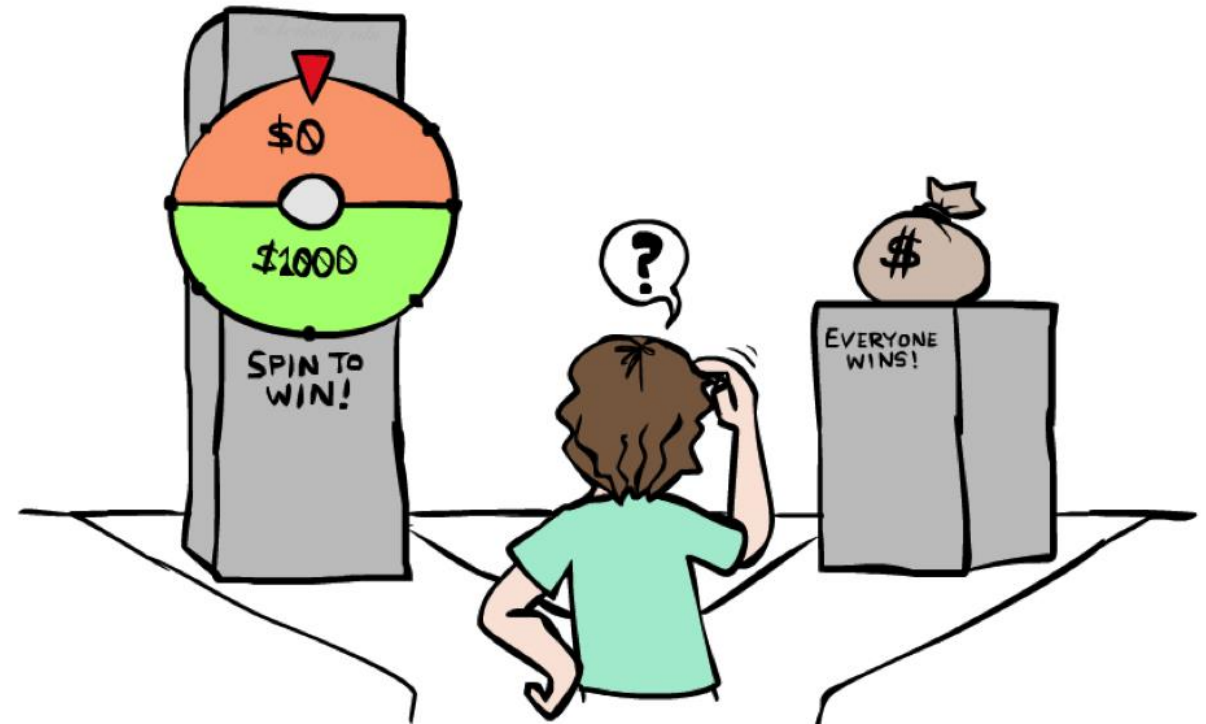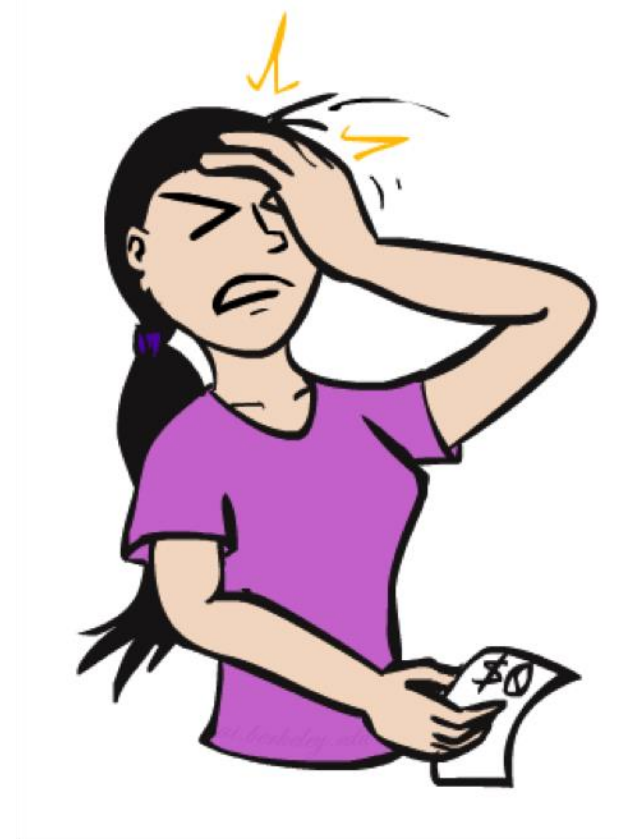
Example: Human Rationality?

❖Famous example of Allais (1953)

➢A: [0.8, $4k;   0.2, $0]
➢B: [1.0, $3k;   0.0, $0]

➢C: [0.2, $4k;   0.8, $0]
➢D: [0.25, $3k;   0.75, $0]

❖Most people prefer B > A, C > D

❖But if U($0) = 0, then
➢B > A $\Rightarrow$ U($3k) > 0.8 U($4k)
➢C > D $\Rightarrow$ 0.8 U($4k) > U($3k)

Consent

Data for this course is taken from several books specifically AIMA by Russel Norvig, slides and already similar course taught in other universities specifically MIT, UC- Berkley (ai.berkeley.edu) and RIT and is the sole property of the respective owner. The copyright infringement is not intended, and this material is solely used for the academic purpose or as a teaching material