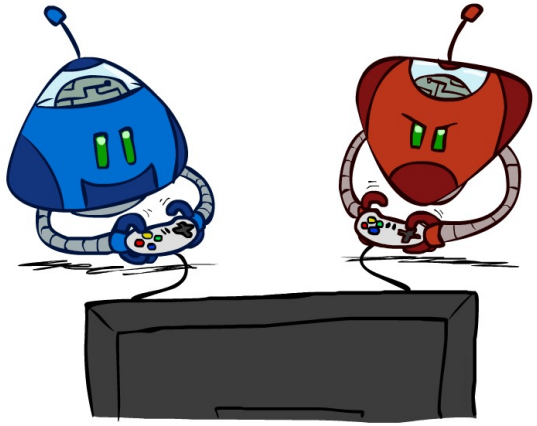
Announcements

- Homework
  - HW1 was due today
  - HW2 (CSP and AS) is out today. Due October 5th
  - HW3 (MDP) will be out next Tuesday. Due October 5th (7th)
- Project
  - Project 1 is due Saturday
  - Project 2 is out today. Due on October 12th
- Review Session on October 4th week
  - Monday 4th 3-5 pm by TA (Solving questions and review questions)
  - Wednesday 6th 1-3 pm and 6-7 pm by me (Bring your questions session)
  - If you need anything else or cannot make it to any, by appointment
  - All will be hosted via zoom. Links are to be announced later
- Practice exams
  - CS188 past exams (Goes back to Spring 2011)

CS 3568: Intelligent Systems

Adversarial Search (Part 3)



Instructor: Tara Salman

Texas Tech University

Computer Science Department

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).]

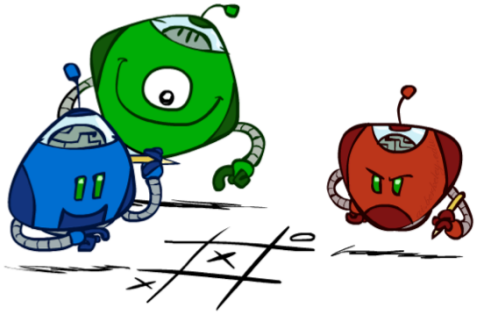
Last Week and This Lecture

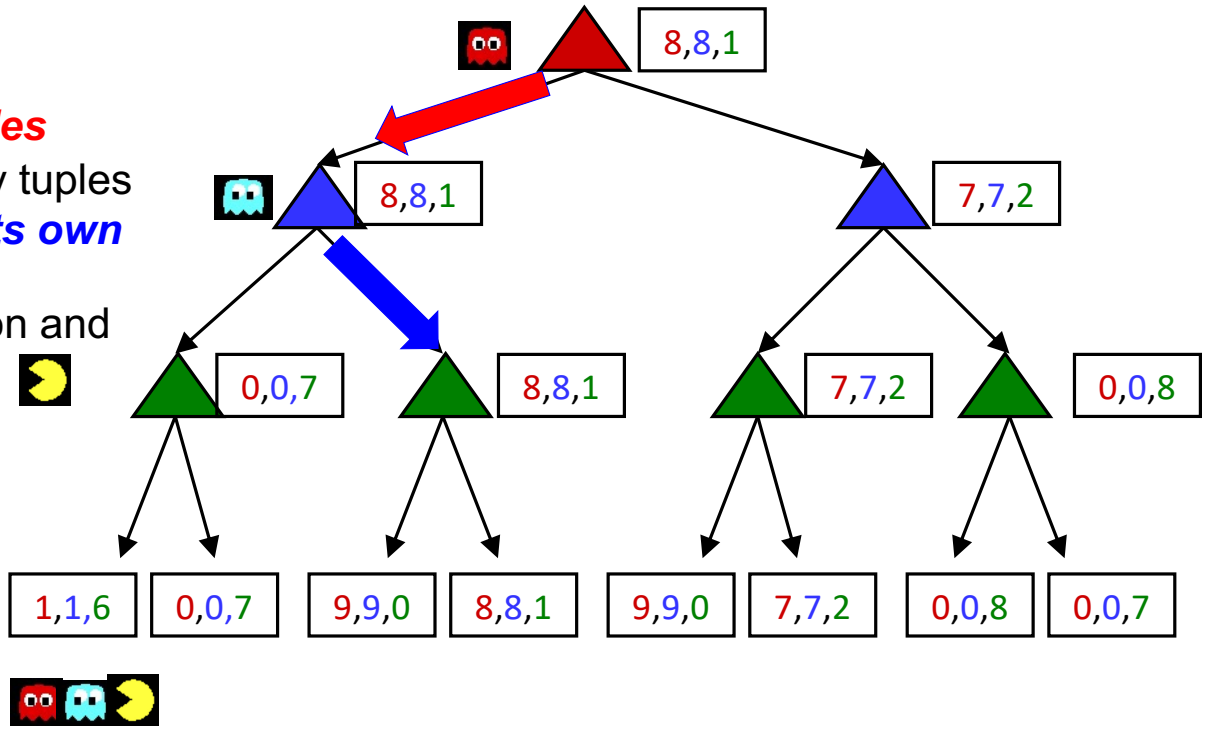
- Last Week (games or adversarial search)
  - Minimiax and Expectmax
  - Ways to enhance Minimax
    - Alpha-beta pruning, deep limited search
  - Evaluation Function
  - The importance of correct modeling of the world

- This lecture
  - Other type of games
  - How to assign utilities
  - New Topic (MDP)

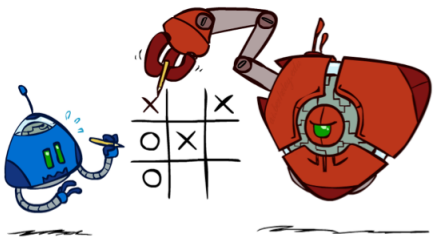
Generalized minimax

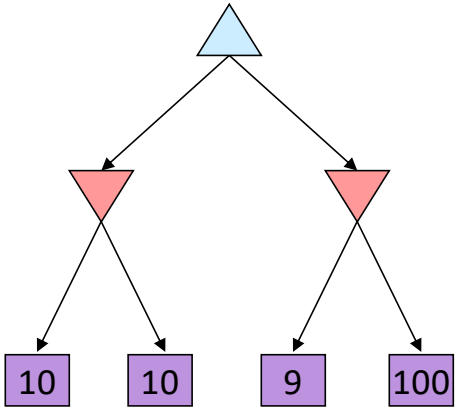
- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have ***utility tuples***
  - Node values are also utility tuples
  - ***Each player maximizes its own component***
  - Can give rise to cooperation and competition dynamically…





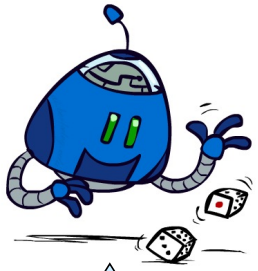
Chance outcomes in trees

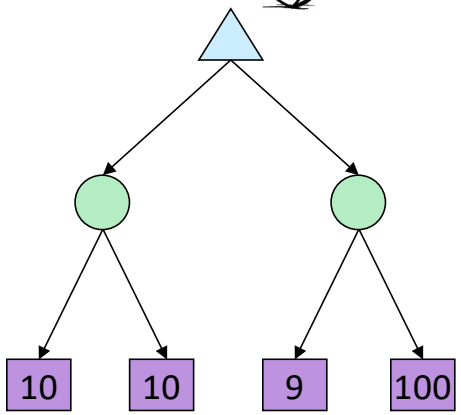




Tictactoe, chess

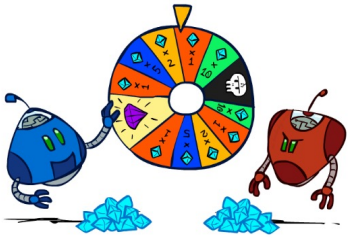
Minimax

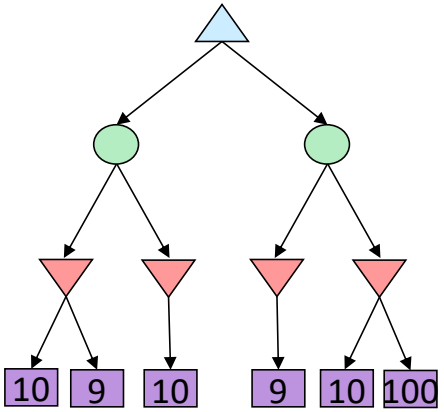




Tetris, investing

Expectimax





Backgammon, Monopoly

Expectiminimax

Expectiminimax

function decision(s) returns an action

return the action a in Actions(s) with the highest value(Result(s,a))

function value(s) returns a value

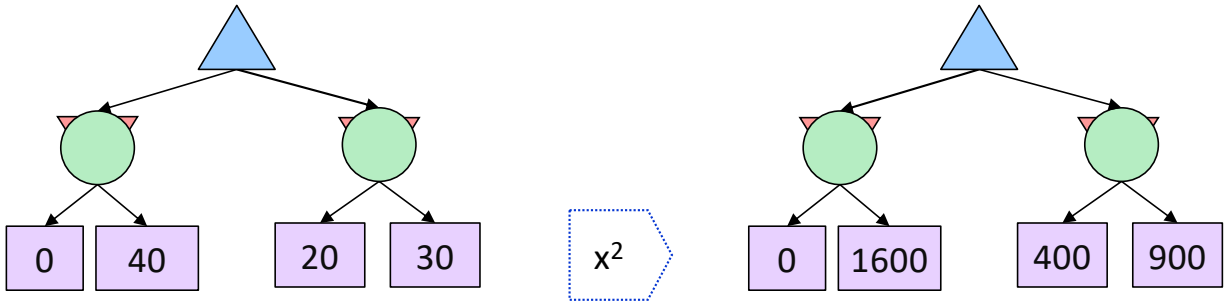
if Terminal-Test(s) then return Utility(s)

if Player(s) = MAX then return $\max_{a \text{ in Actions(s)}}$ value(Result(s,a))

if Player(s) = MIN then return $\min_{a \text{ in Actions(s)}}$ value(Result(s,a))

if Player(s) = CHANCE then return $\text{sum}_{a \text{ in Actions(s)}}$ Pr(a) * value(Result(s,a))

What Utilities to Use?



- For worst-case minimax reasoning, evaluati esn't matter
  - We just want better states to have higher eva ions (get the ordering right)
  - Minimax decisions are ***invariant with respect to monotonic transformations on values***

x>y => f(x)>f(y)

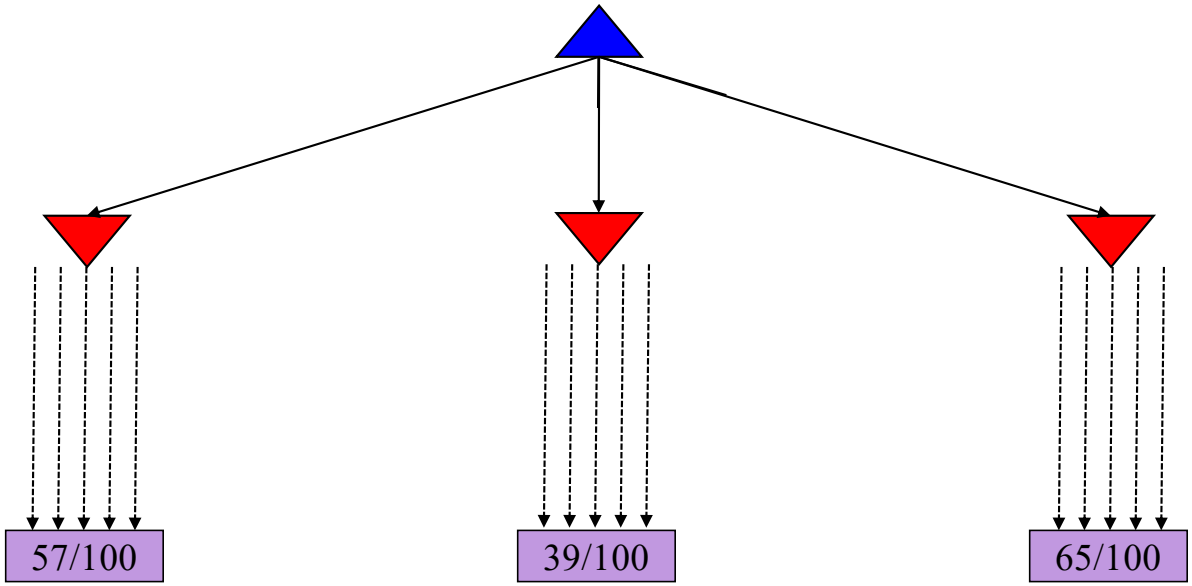
- For average-case expectimax reasoning, we need *magnitudes* to be meaningful

Monte Carlo Tree Search

- Methods based on alpha-beta search assume a fixed horizon
  - Pretty hopeless for Go, with $b > 300$
- MCTS combines two important ideas:
  - ***Evaluation by rollouts*** – play multiple games to termination from a state s (using a simple, fast rollout policy) and count wins and losses
  - ***Selective search*** – explore parts of the tree that will help improve the decision at the root, regardless of depth

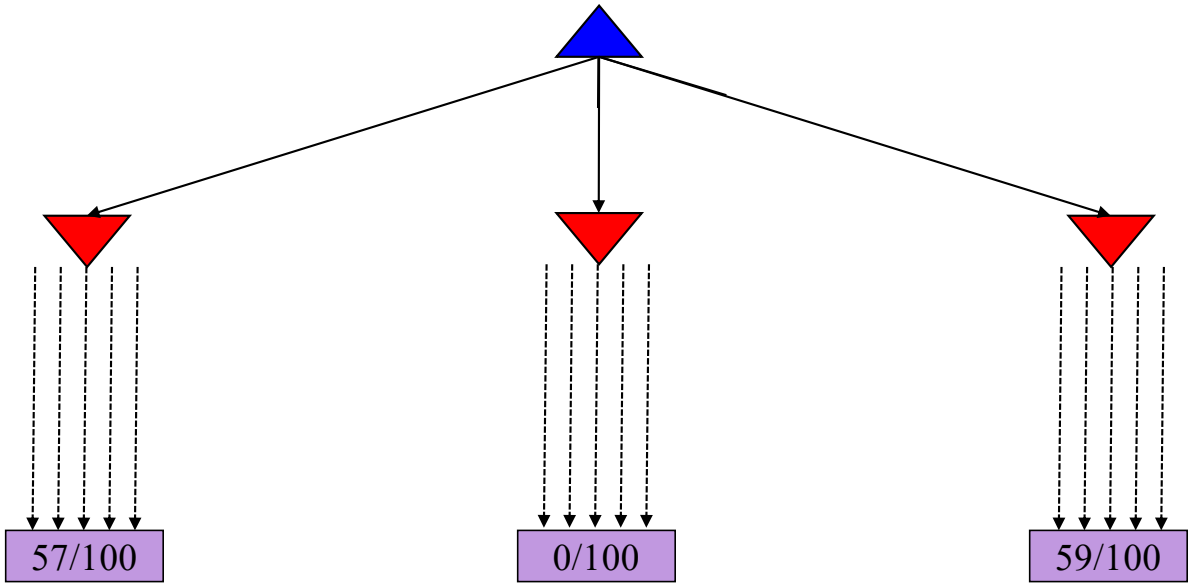
MCTS Version 0

- Do *N* rollouts from each child of the root, record fraction of wins
- Pick the move that gives the best outcome by this metric



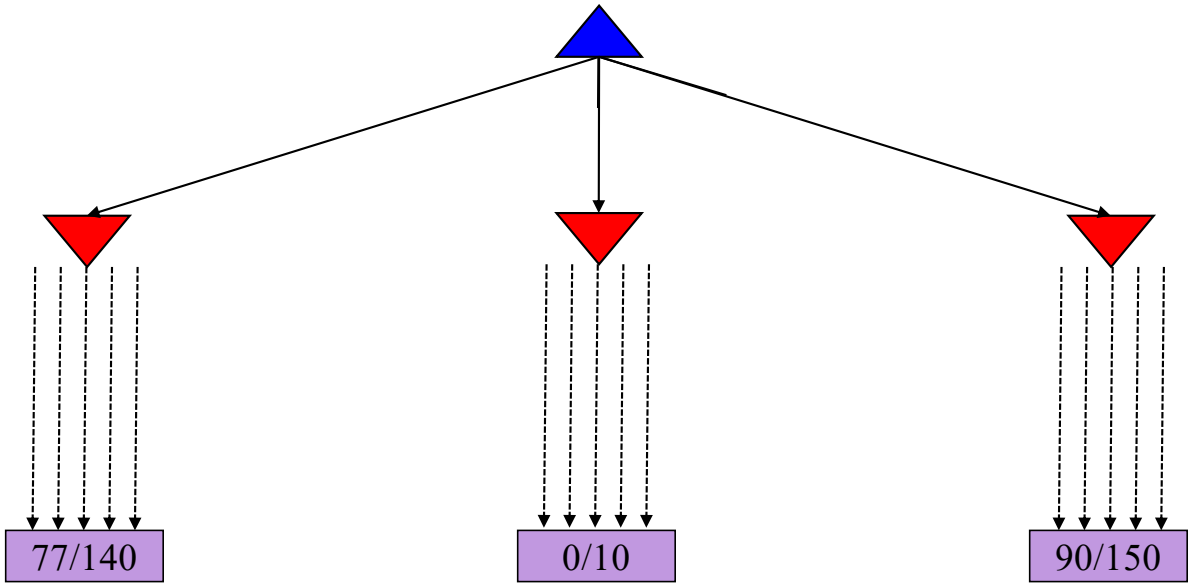
MCTS Version 0

- Do *N* rollouts from each child of the root, record fraction of wins
- Pick the move that gives the best outcome by this metric



MCTS Version 0.9

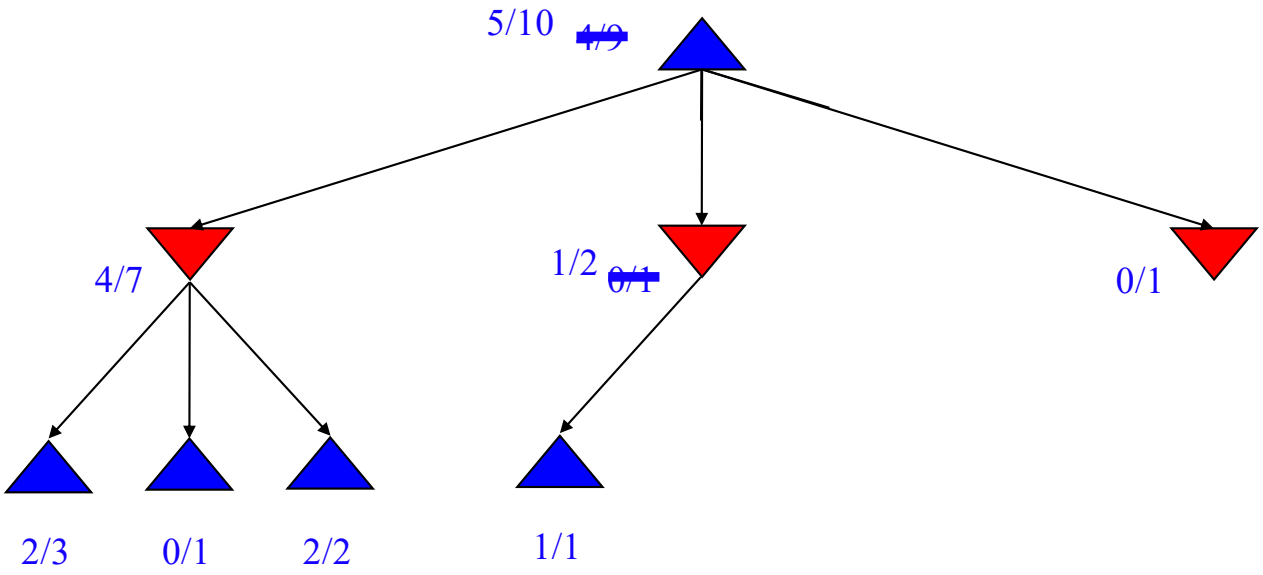
- Allocate rollouts to more promising nodes



MCTS Version 2.0: UCT

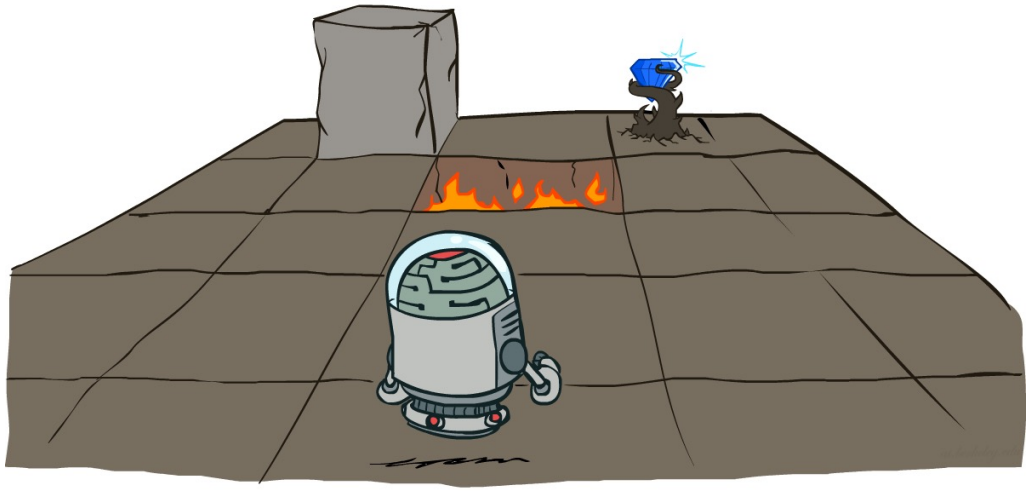
- Repeat until out of time:
  - Given the current search tree, recursively apply UCB to choose a path down to a leaf (not fully expanded) node *n*
  - Add a new child *c* to *n* and run a rollout from *c*
  - Update the win counts from *c* back up to the root
- Choose the action leading to the child with highest *N*

UCT Example



CS 3568: Intelligent Systems

Markov Decision Processes



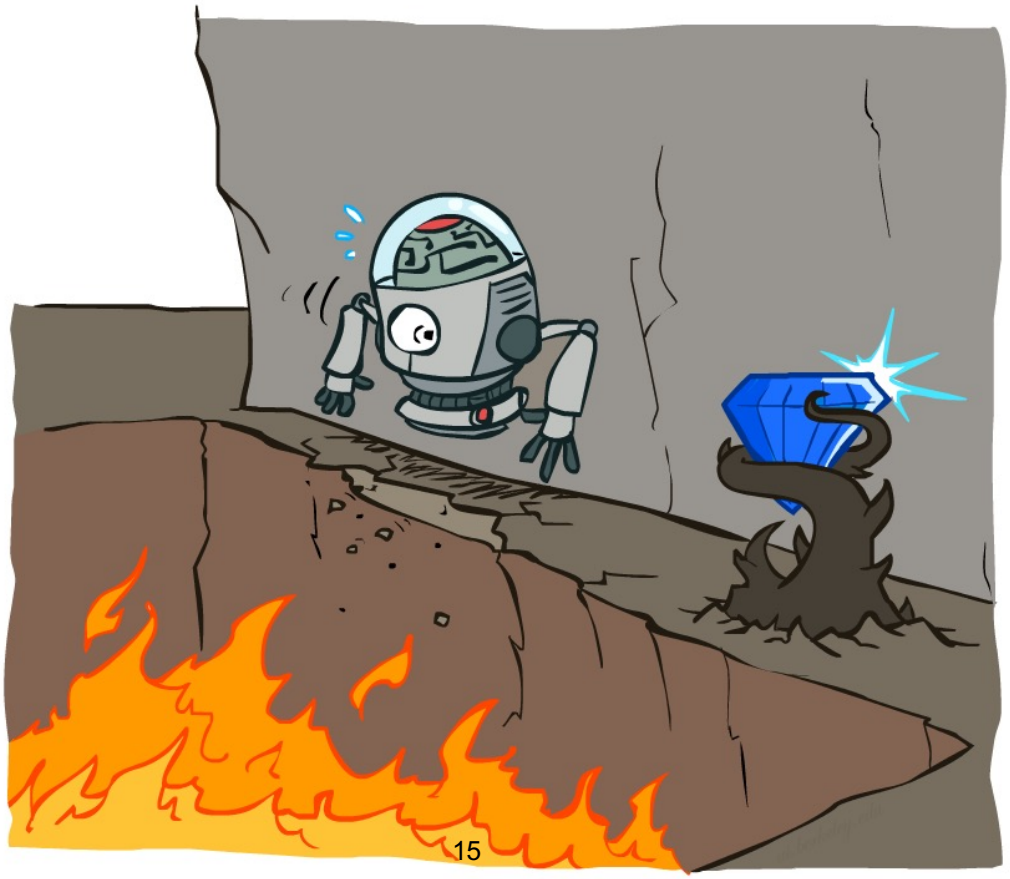
Instructor: Tara Salman

Texas Tech University

Computer Science Department

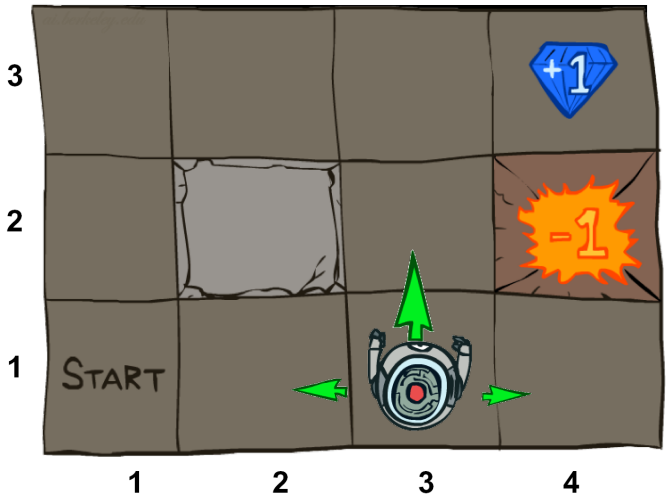
[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).]

Non-Deterministic Search



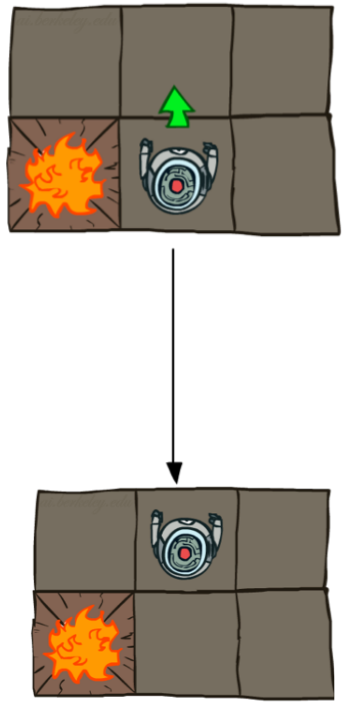
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards

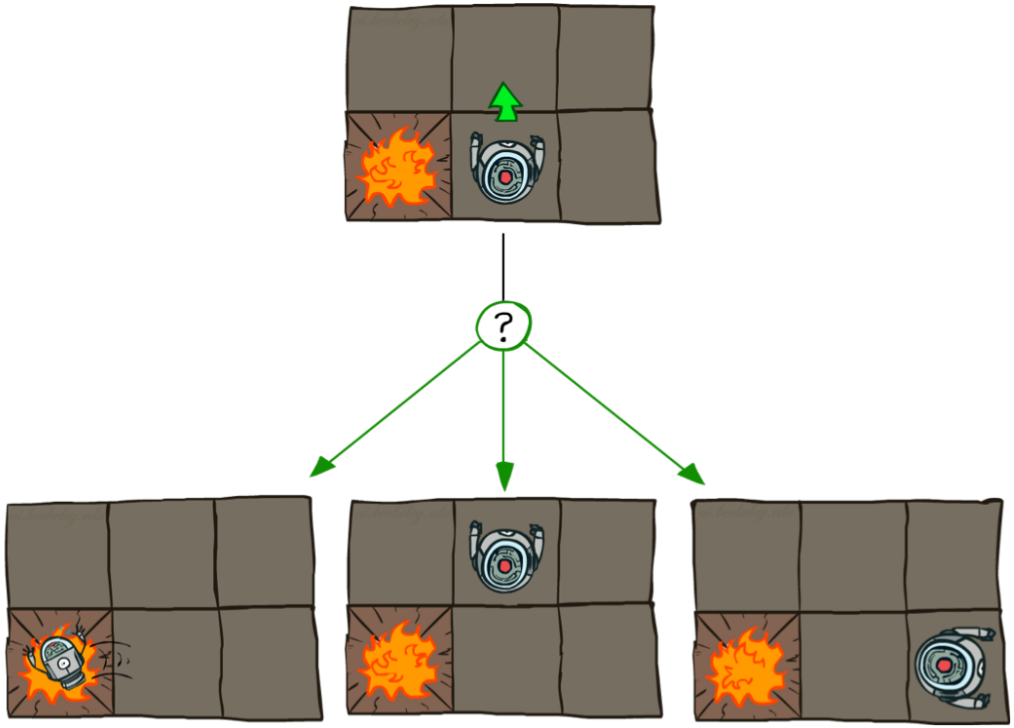


Grid World Actions

Deterministic Grid World

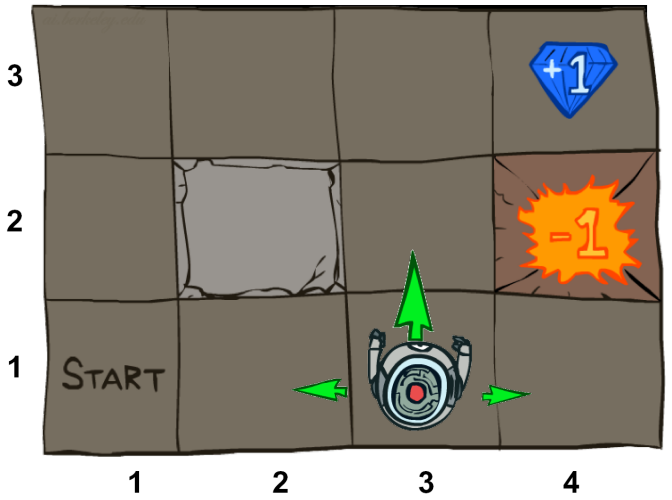


Stochastic Grid World



Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s'| s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state
  - Maybe a terminal state



- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We'll have a new tool soon

What is Markov about MDPs?

- "Markov" generally means that given the present state, the future and the past are independent

- For Markov decision processes, "Markov" means action outcomes depend only on the current state

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- This is just like search, where the successor function could only depend on the current state (not the history)

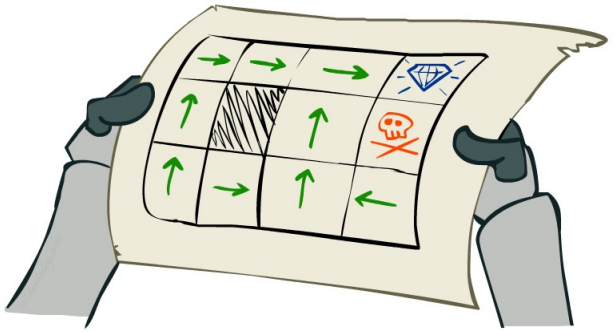
Andrey Markov
(1856-1922)

Policies

- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

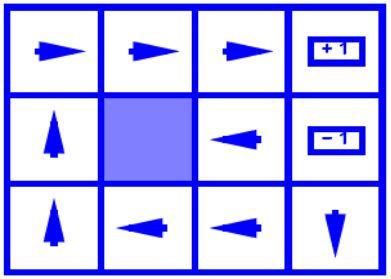
- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

- Expectimax didn't compute entire policies
  - It computed the action for a single state only

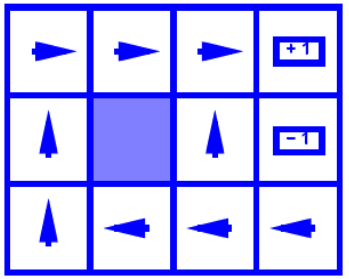


Optimal policy when $R(s, a, s') = -0.03$ for all non-terminals s

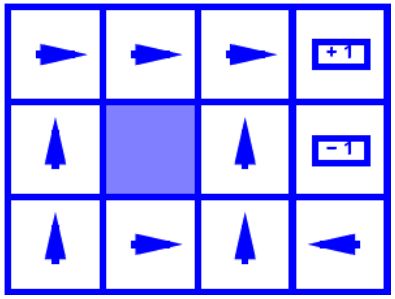
Optimal Policies



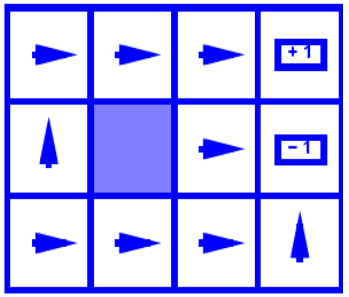
R(s) = -0.01



R(s) = -0.03

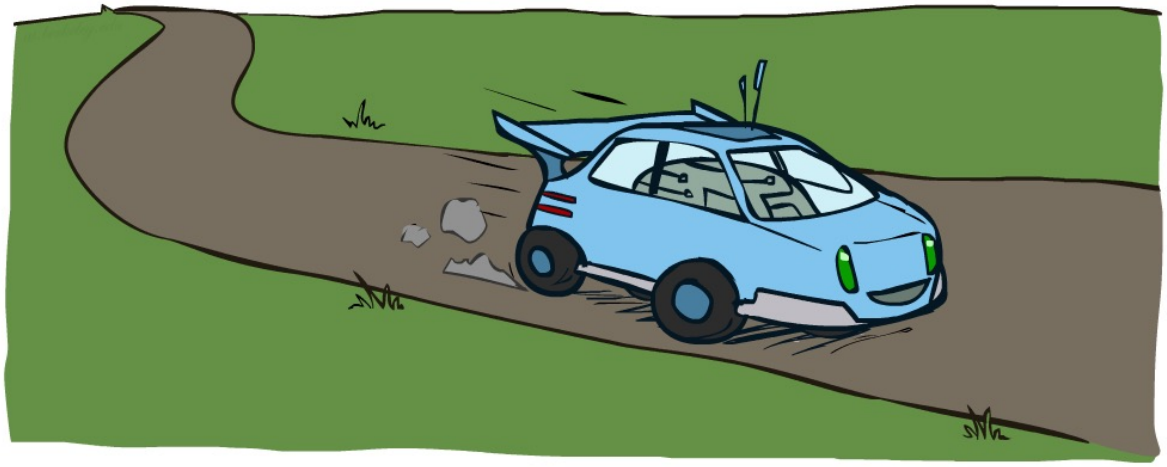


R(s) = -0.4



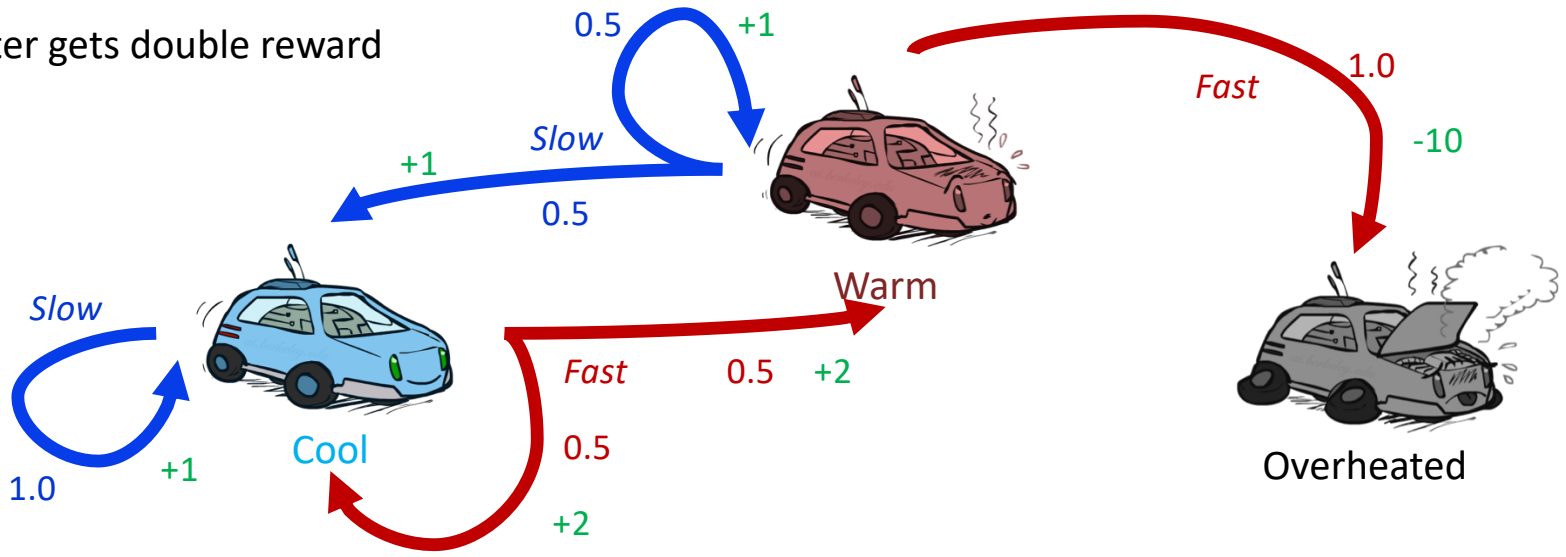
R(s) = -2.0

Example: Racing



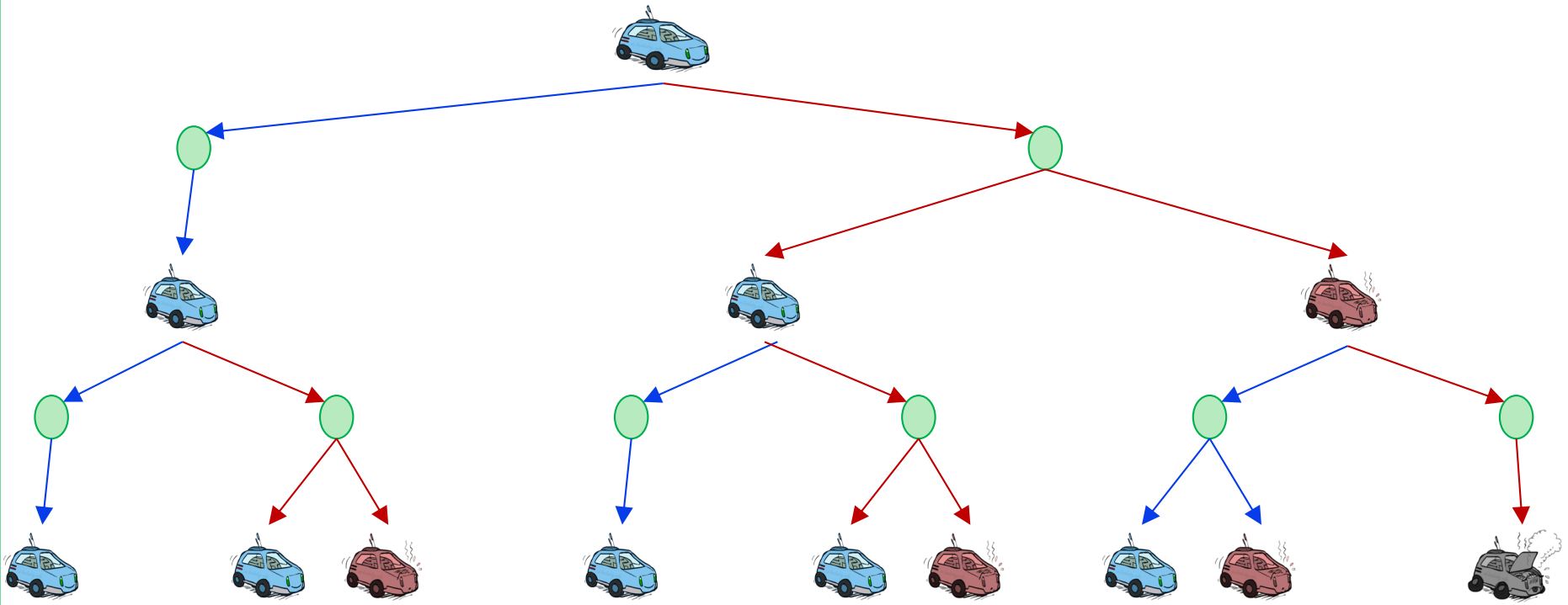
Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



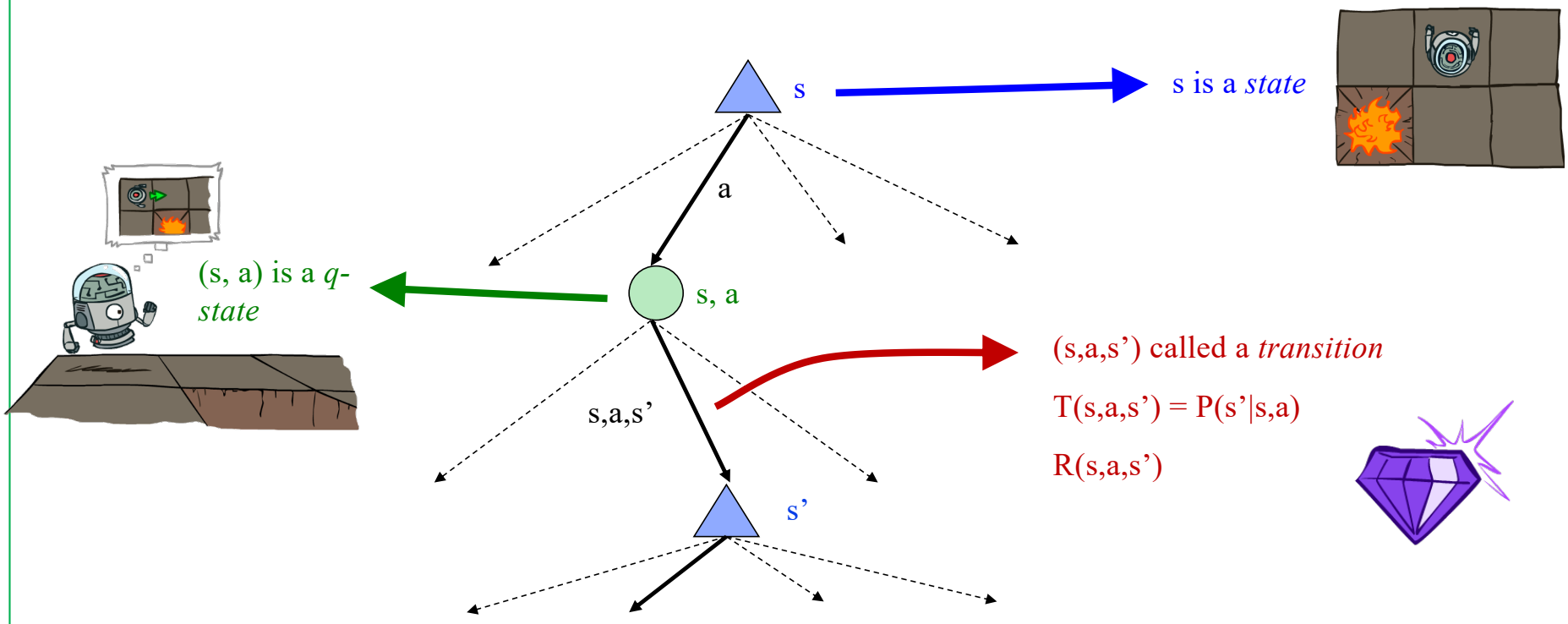
State transition diagram

Racing Search Tree

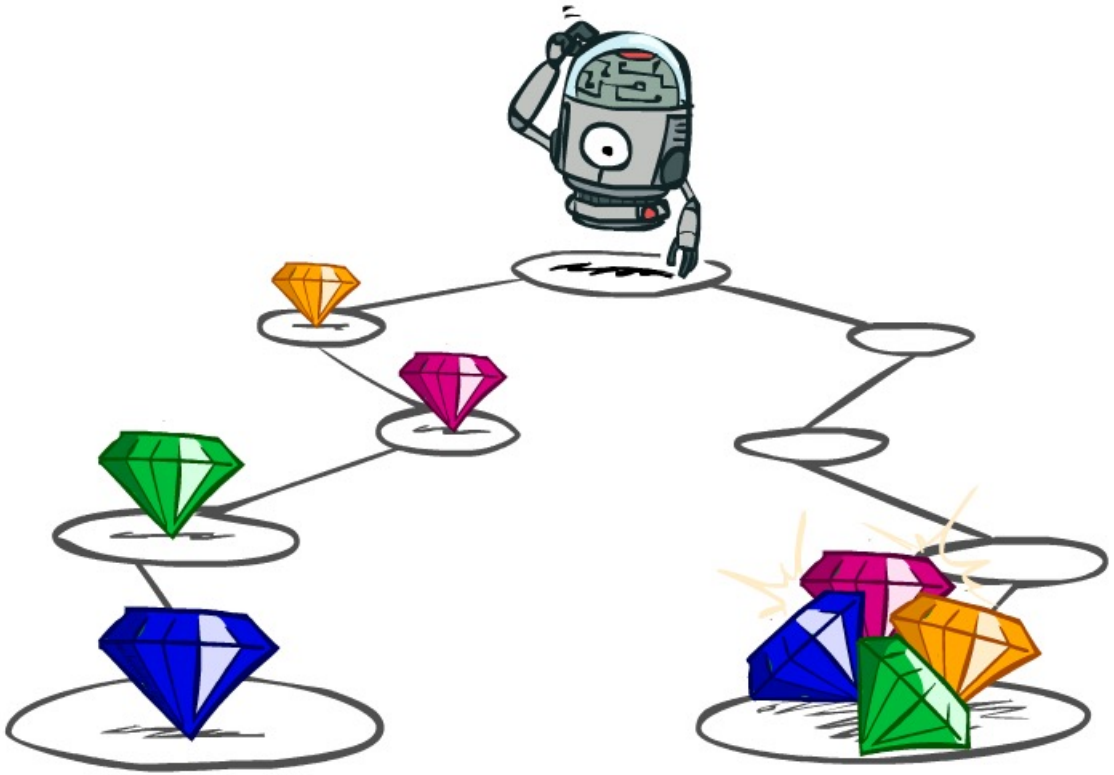


MDP Search Trees

- Each MDP state projects an expectimax-like search tree

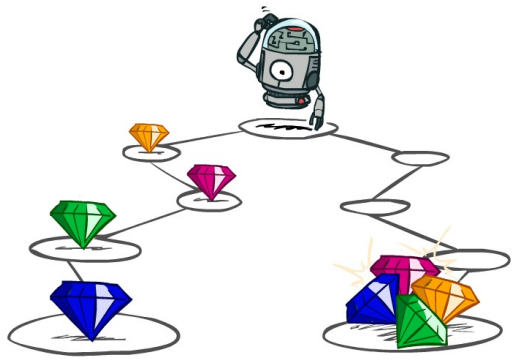


Utilities of Sequences



Utilities of Sequences

- What preferences should an agent have over reward sequences?
- More or less? [1, 2, 2] or [2, 3, 4]
- Now or later? [0, 0, 1] or [1, 0, 0]



Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

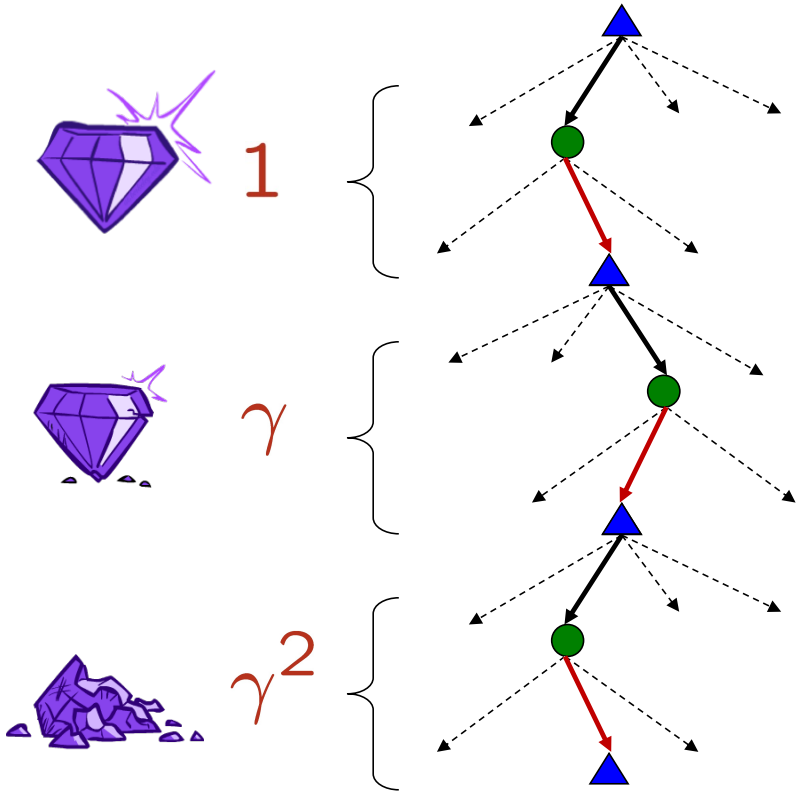
Worth Next Step

γ^2

Worth In Two Steps

Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



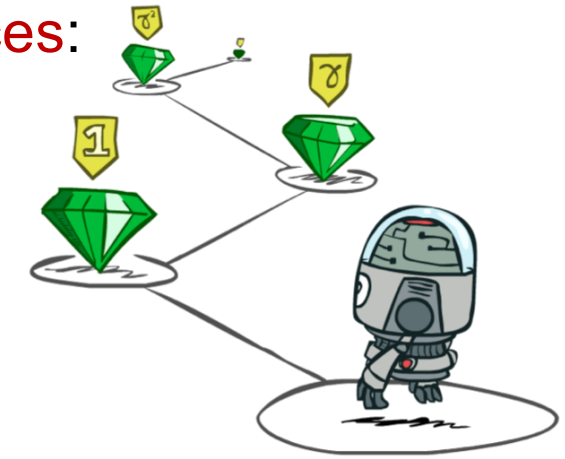
Stationary Preferences

- Theorem: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$

$$\Updownarrow$$

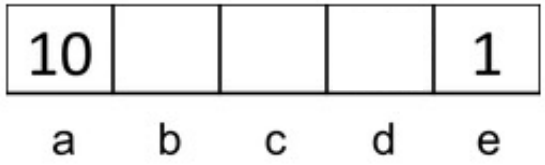
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$



- Then: there are only two ways to define utilities
  - Additive utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$
  - Discounted utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

Quiz: Discounting

- Given:



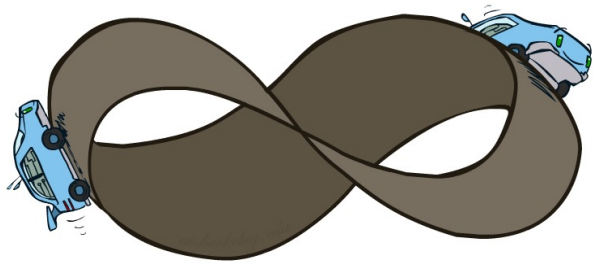
  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic

- Quiz 1: For $\gamma = 1$, what is the optimal policy?

- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?

Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)



  - Discounting: use $0 < \gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - Smaller γ means smaller "horizon" – shorter term focus

  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)