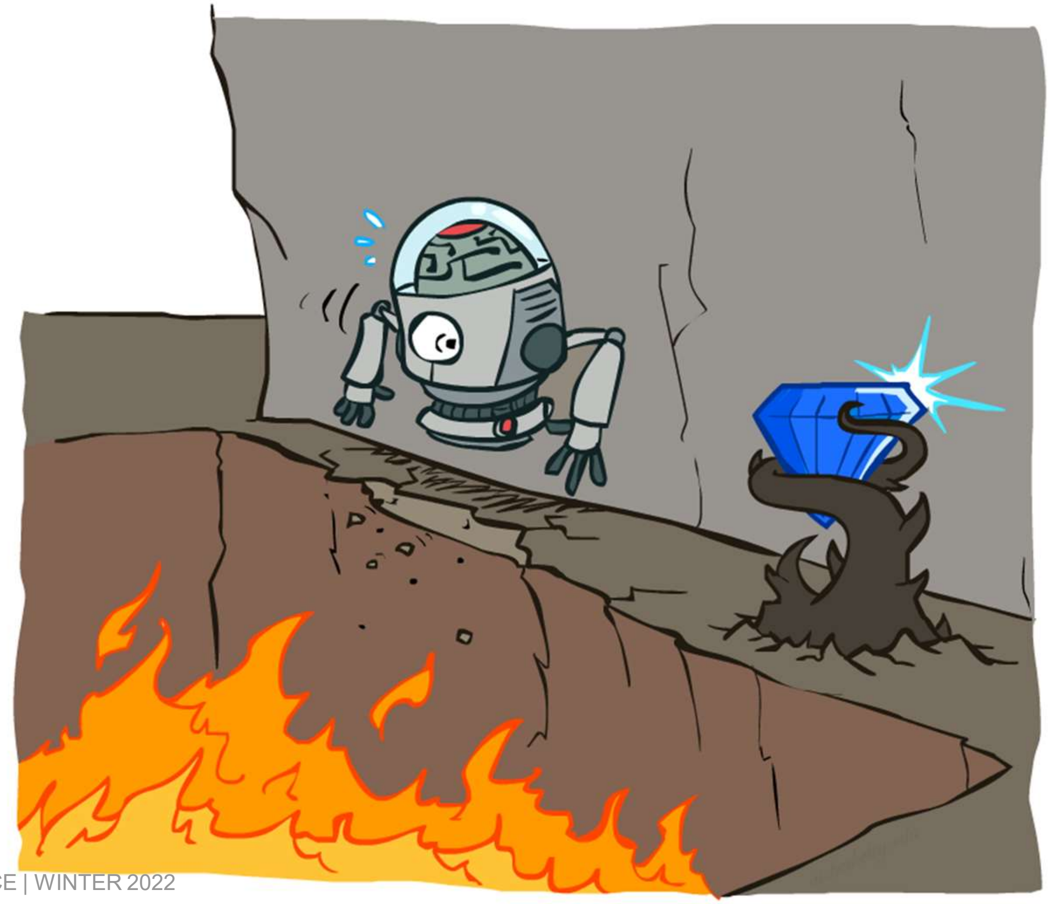
Tampere University

Making Complex Decisions

CHAPTER 17 IN THE TEXTBOOK

Tampere University

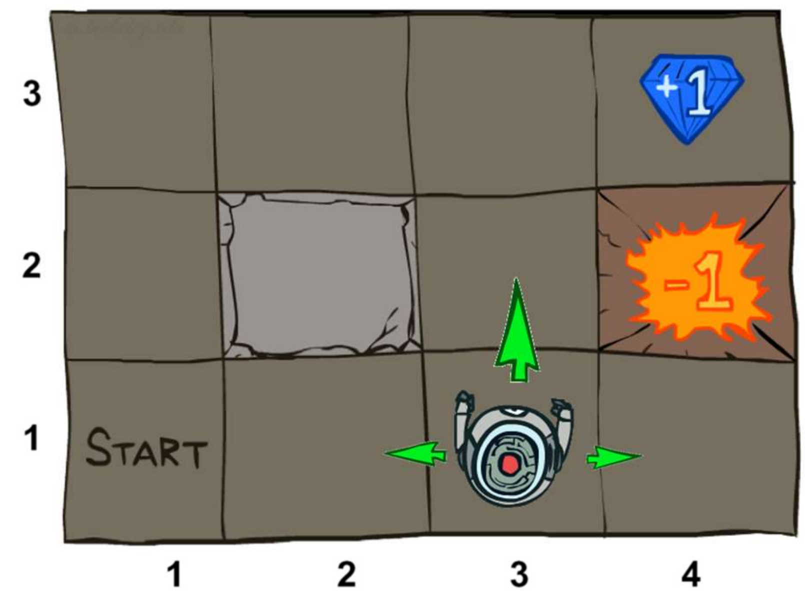
Non-Deterministic Search



Tampere University

Example: Grid World

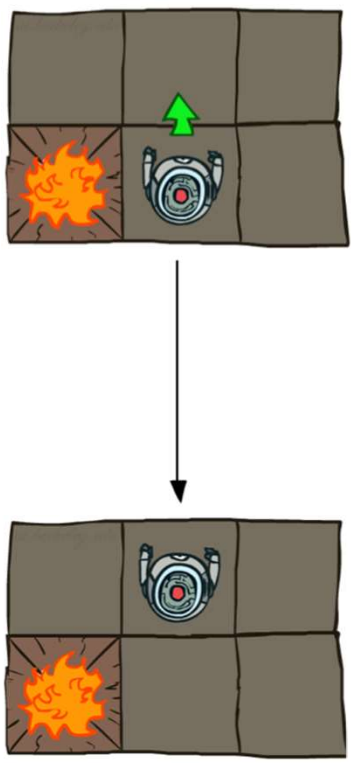
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is 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



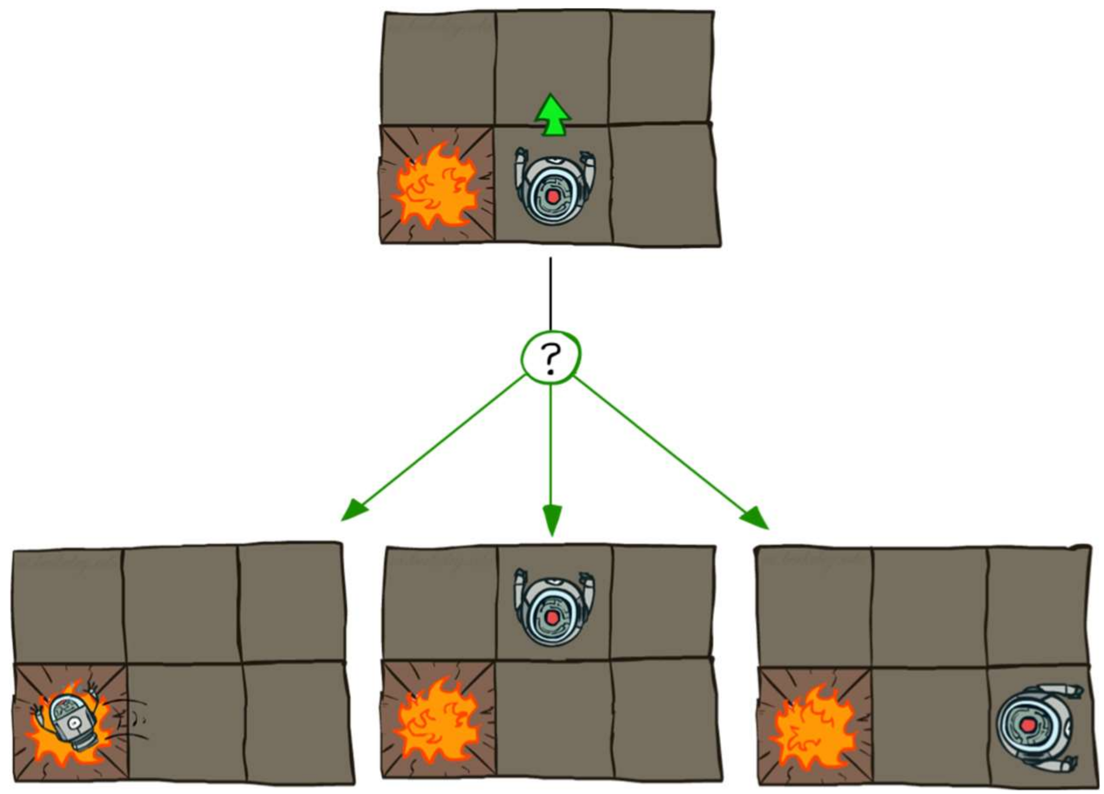
Tampere University

Grid World Actions

Deterministic Grid World



Stochastic Grid World



- Let the living reward $R(s)$ be -0.04 in all states except in the terminal states
- In a deterministic environment, a solution would be easy: the agent will always reach $+1$ with moves

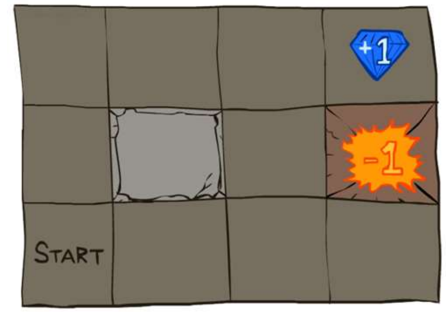
$$[N, N, E, E, E]$$

$$[E, E, N, N, E]$$

- Reward sum for both

$$5 \times -0.04 + 1 = 0.8$$

- But actions are unreliable, a sequence of moves will not always lead to the desired outcome



- Now the sequence $[N\ N, E, E, E]$ leads to the goal state with probability

$$0.8^5 = 0.32768A$$

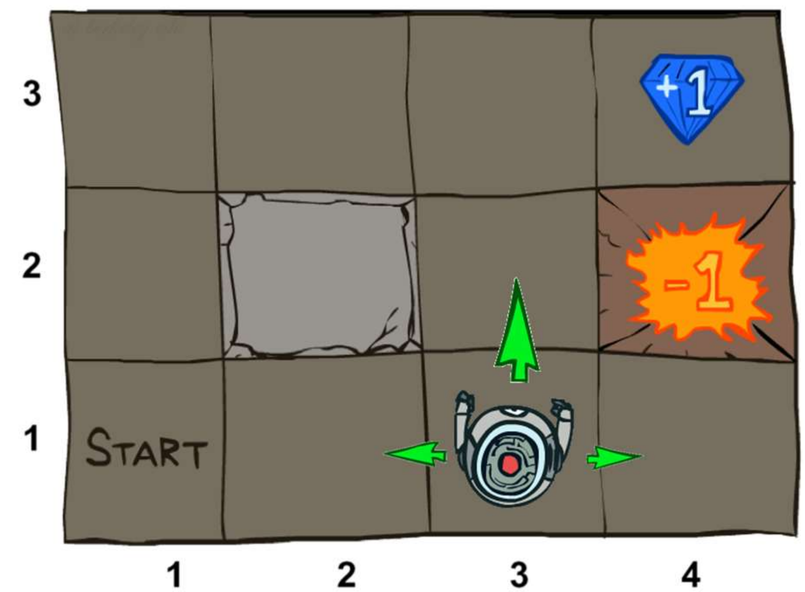
- In addition, the agent has a small chance of reaching the goal by accident going the other way around the obstacle
  - 2 x North fails – land East
  - 2 x East fails – land North
  - 1 x East succeeds

  with a probability $0.1^4 \times 0.8$, for a grand total of 0.32776

Tampere University

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

Video of Demo Gridworld Manual Intro

What is Markov about MDPs?

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

- For MDPs, “Markov” means action outcomes depend only on the current state

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

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

Andrey Markov
(1856-1922)

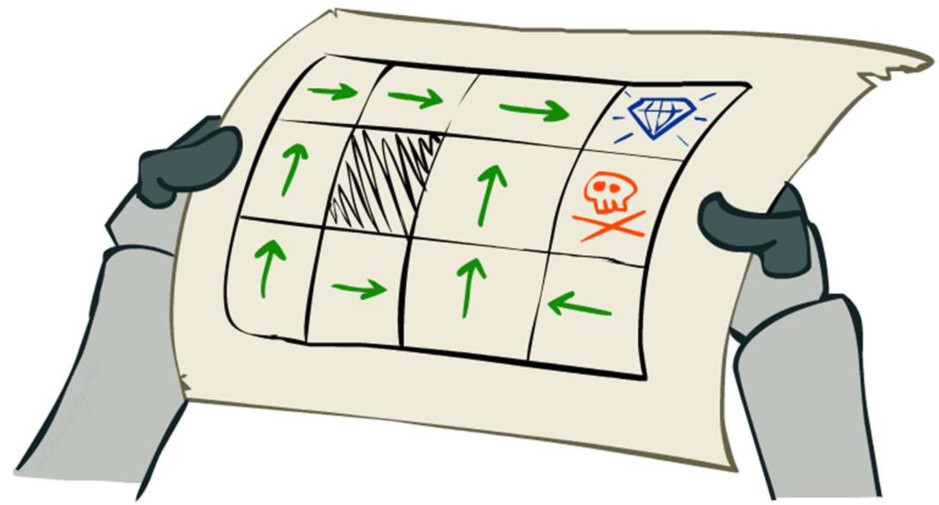
- This is just like search, where the successor function could only depend on the current state (not the history)

Policies

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

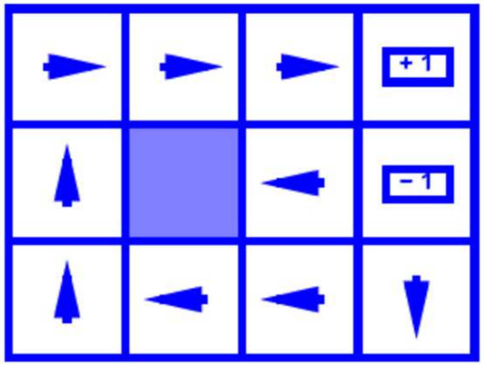
- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

- Expectimax doesn't compute entire policies
  - It computes the action for a single state only

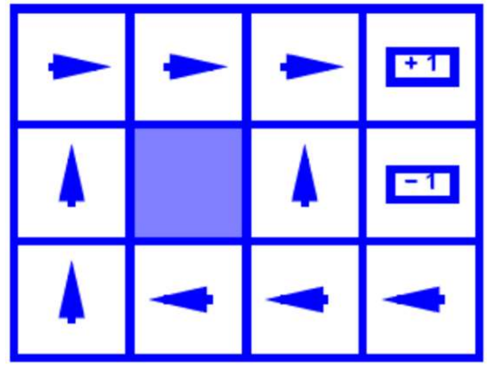


Optimal policy when $R(s, a, s') = -0.03$ for all non-terminals s

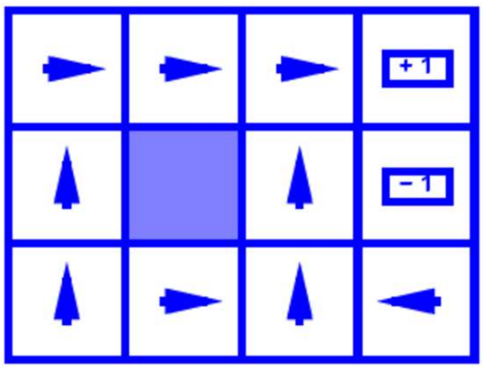
Optimal Policies



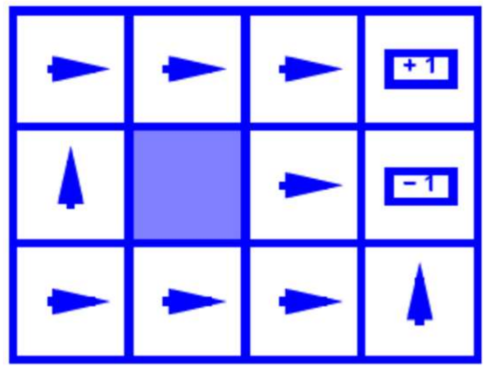
$R(s) = -0.01$



$R(s) = -0.03$



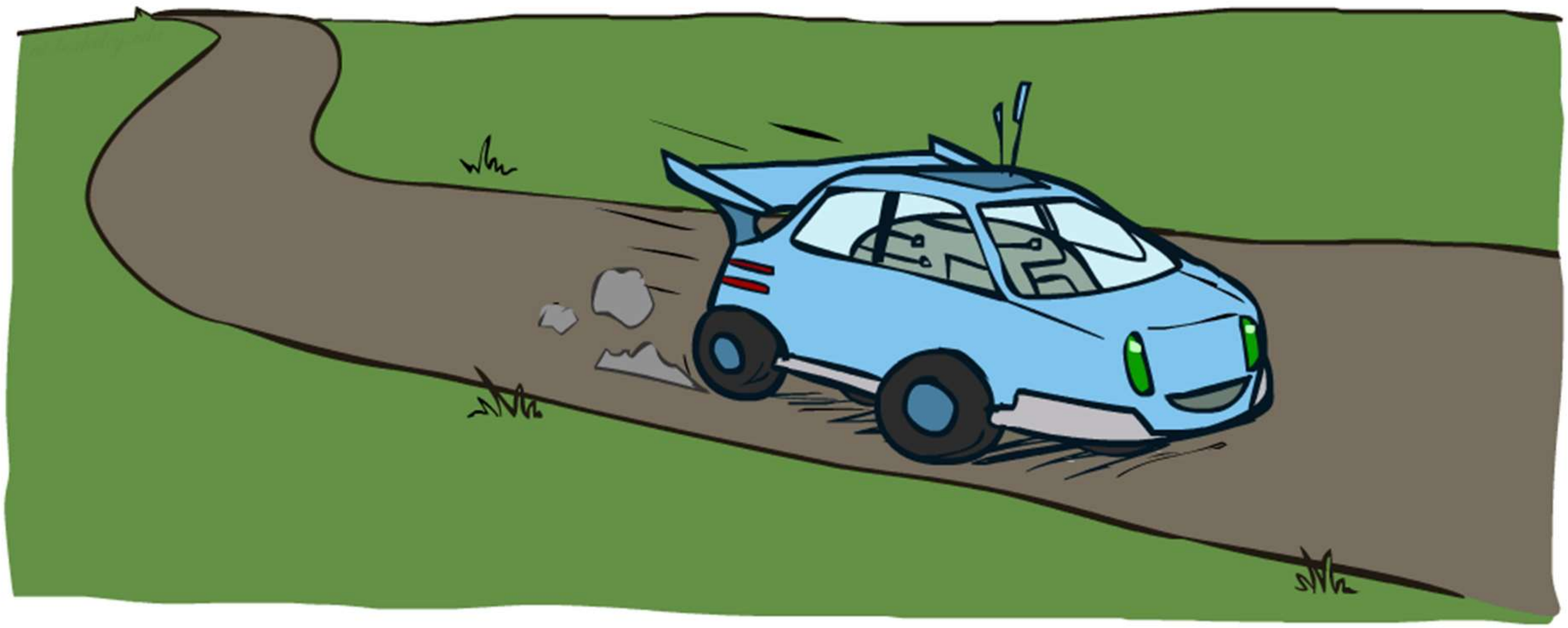
$R(s) = -0.4$



$R(s) = -2.0$

Tampere University

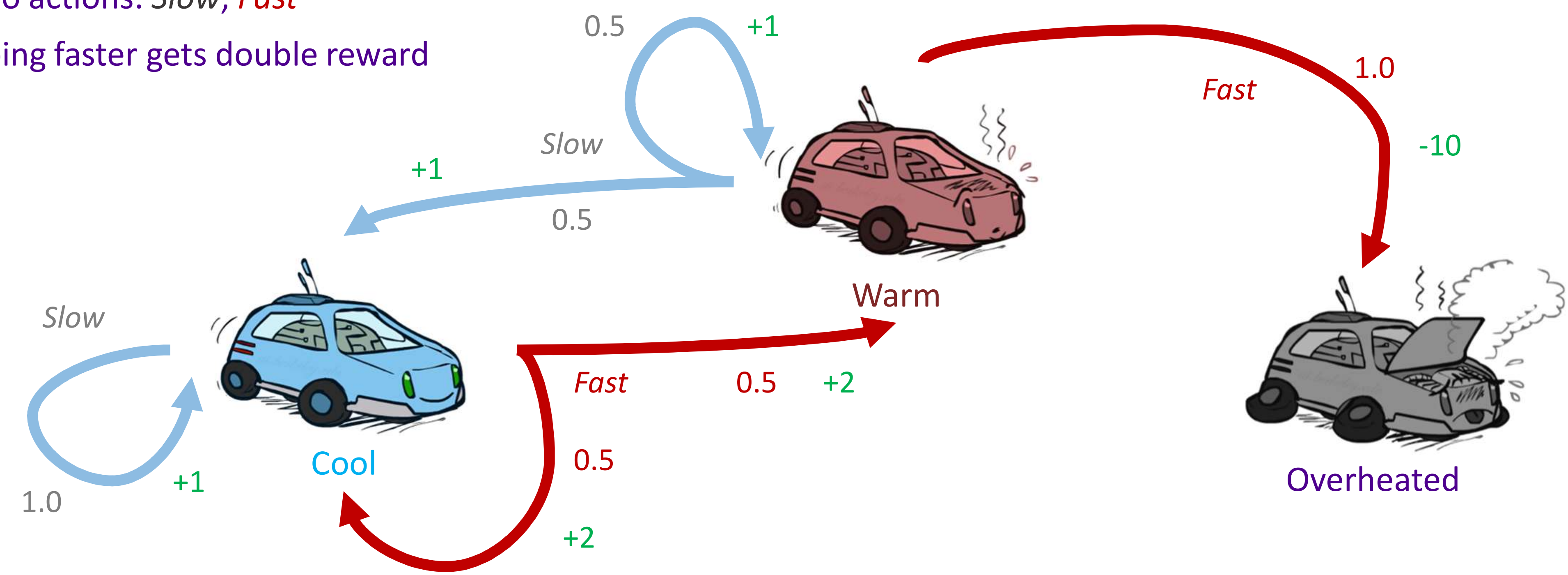
Example: Racing



Tampere University

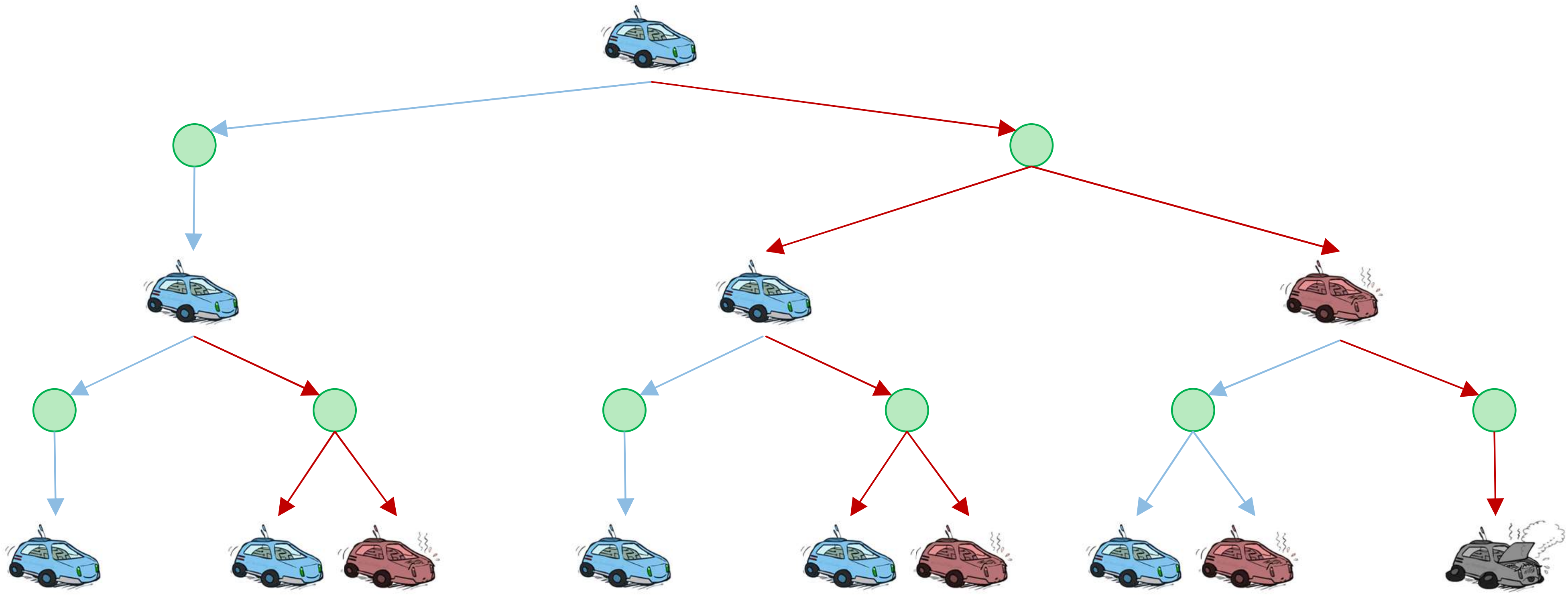
Example: Racing

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



Tampere University

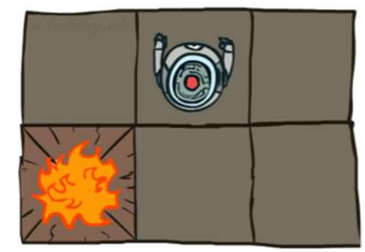
Racing Search Tree

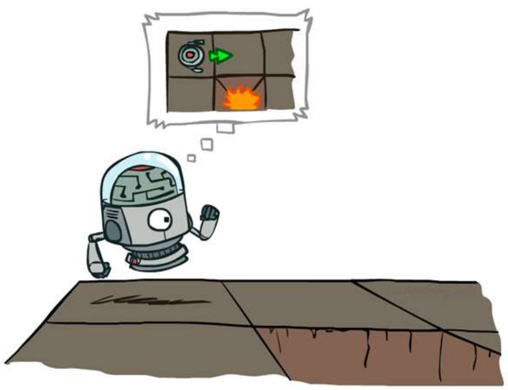


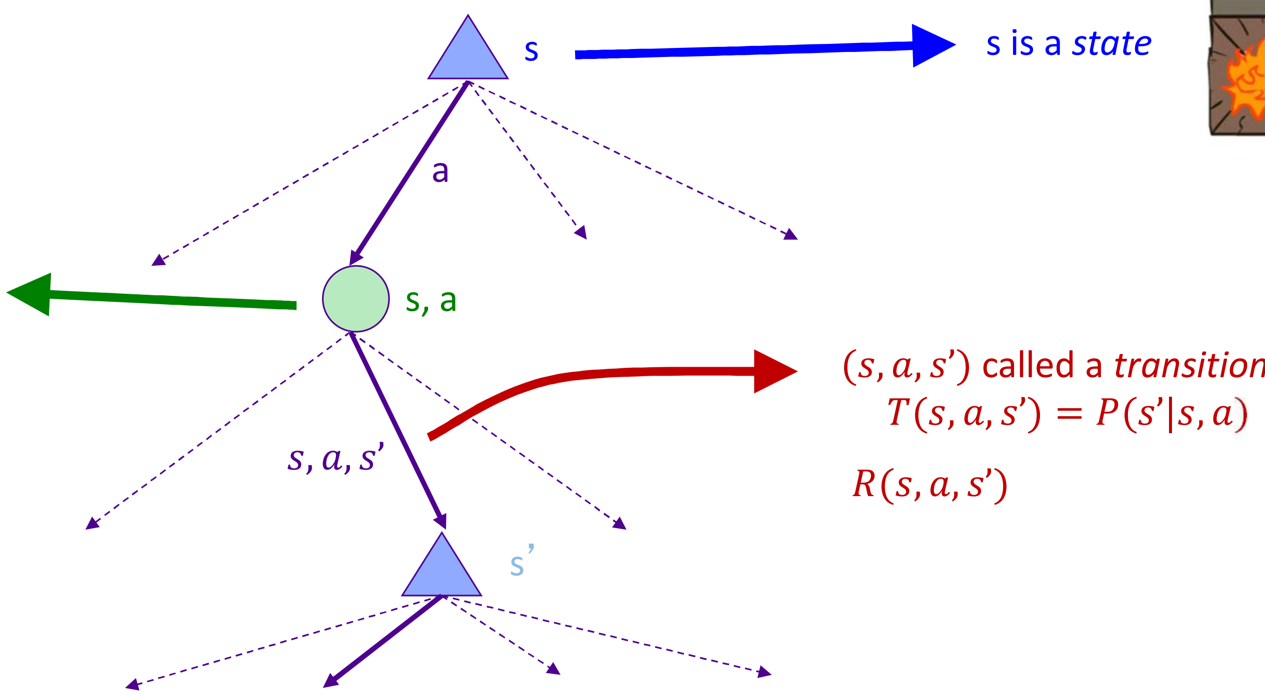
Tampere University

MDP Search Trees

- Each MDP state projects an expectimax-like search tree

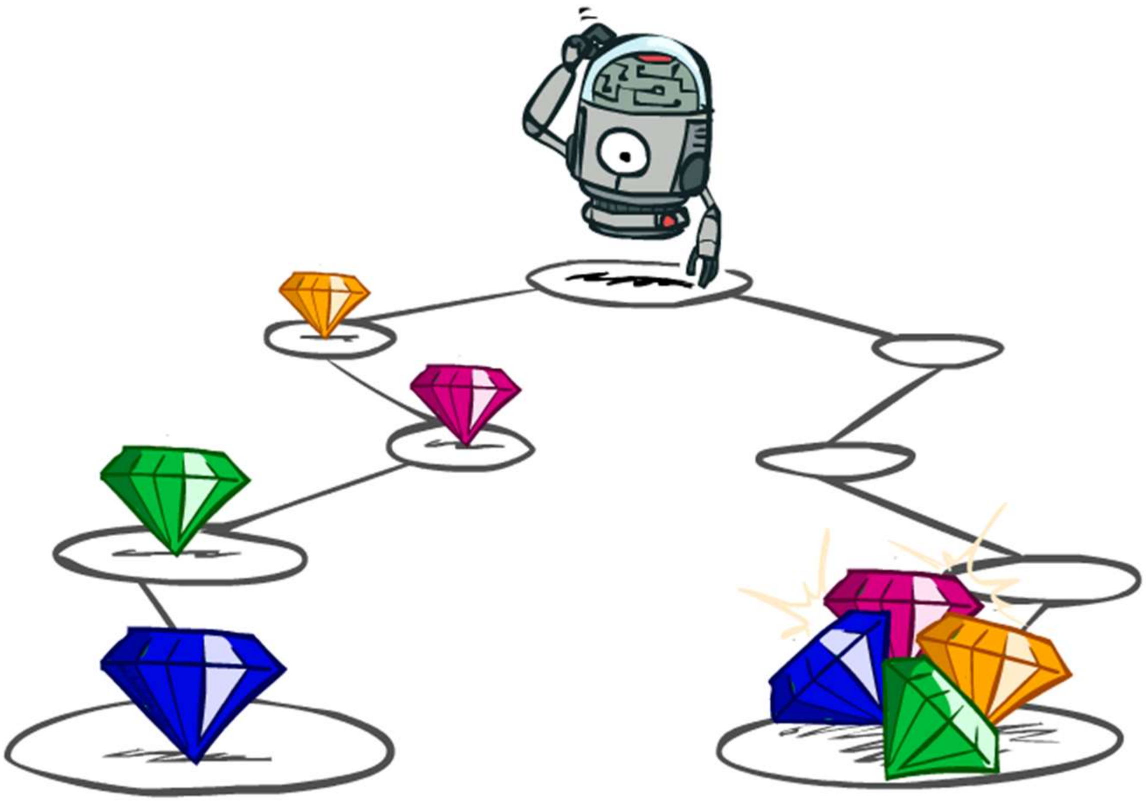






Tampere University

Utilities of Sequences



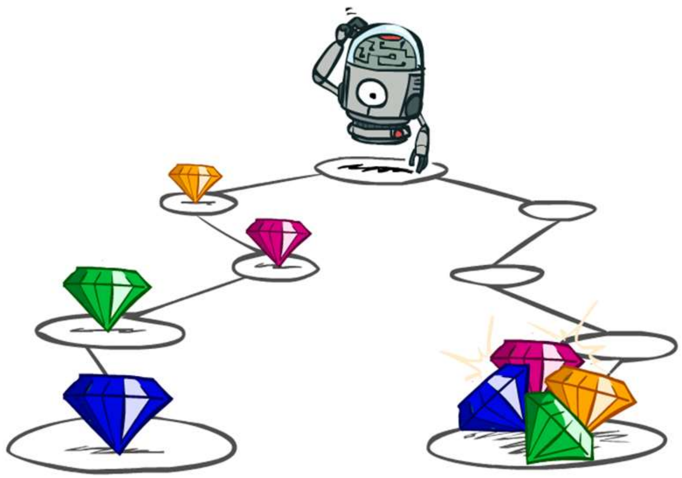
Tampere University

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]



Tampere University

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

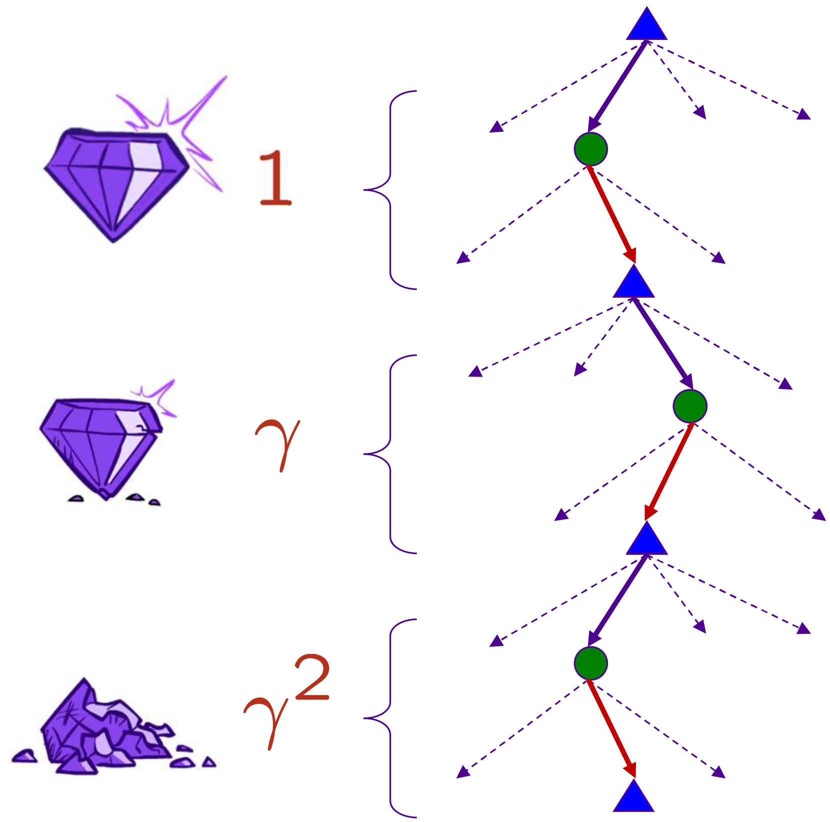
Worth Next Step

γ^2

Worth In Two Steps

Discounting

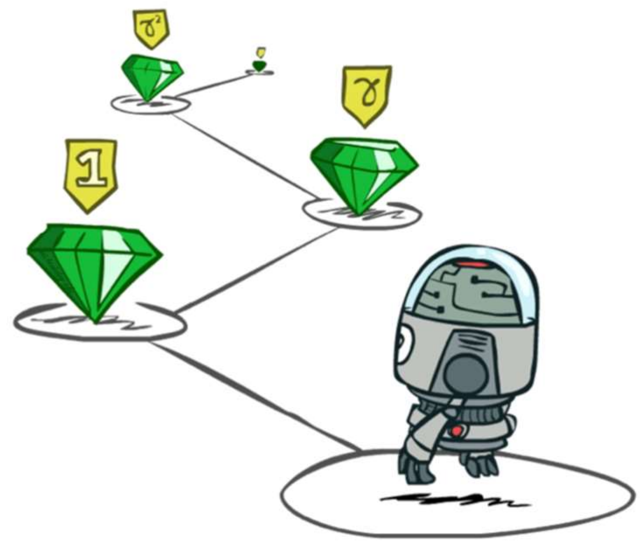
- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge
- Example: discount of 0.5
  - $U([1,2,3]) = 1 * 1 + 0.5 * 2 + 0.25 * 3$
  - $U([1,2,3]) 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



Stationary Preferences

- **Theorem**: if we assume stationary preferences:

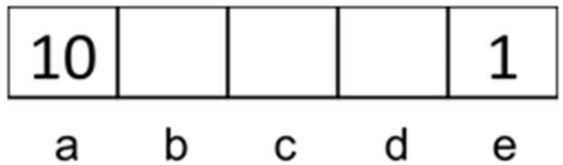
$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$



- 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: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

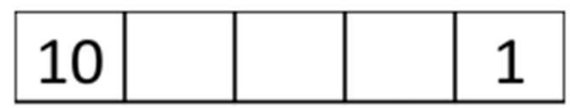
Quiz: Discounting

- Given:

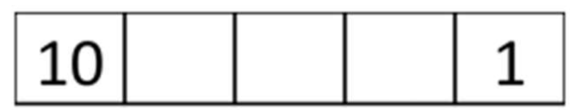


  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic

- Quiz 1: For γ = 1, what is the optimal policy?



- Quiz 2: For γ = 0.1, what is the optimal policy?



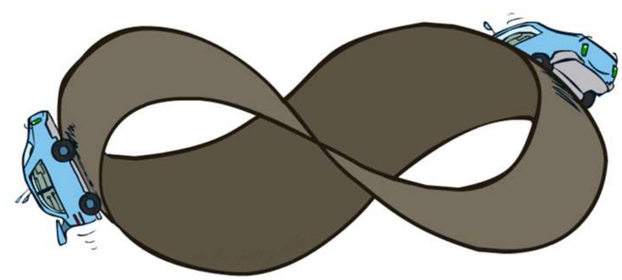
- Quiz 3: For which γ are West and East equally good when in state d?

Tampere University

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 $0 < \gamma < 1$ (Sum of an infinite geometric series)

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

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

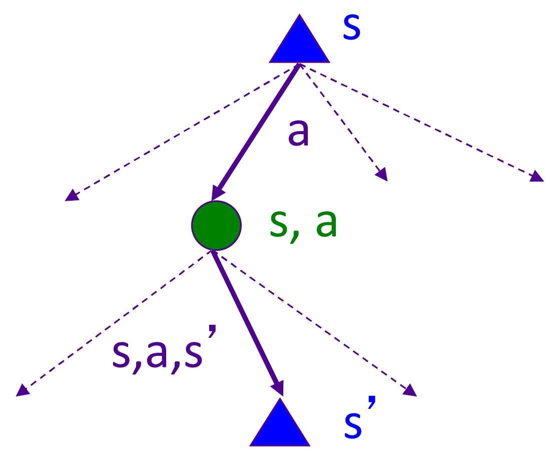
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

Tampere University

Recap: Defining MDPs

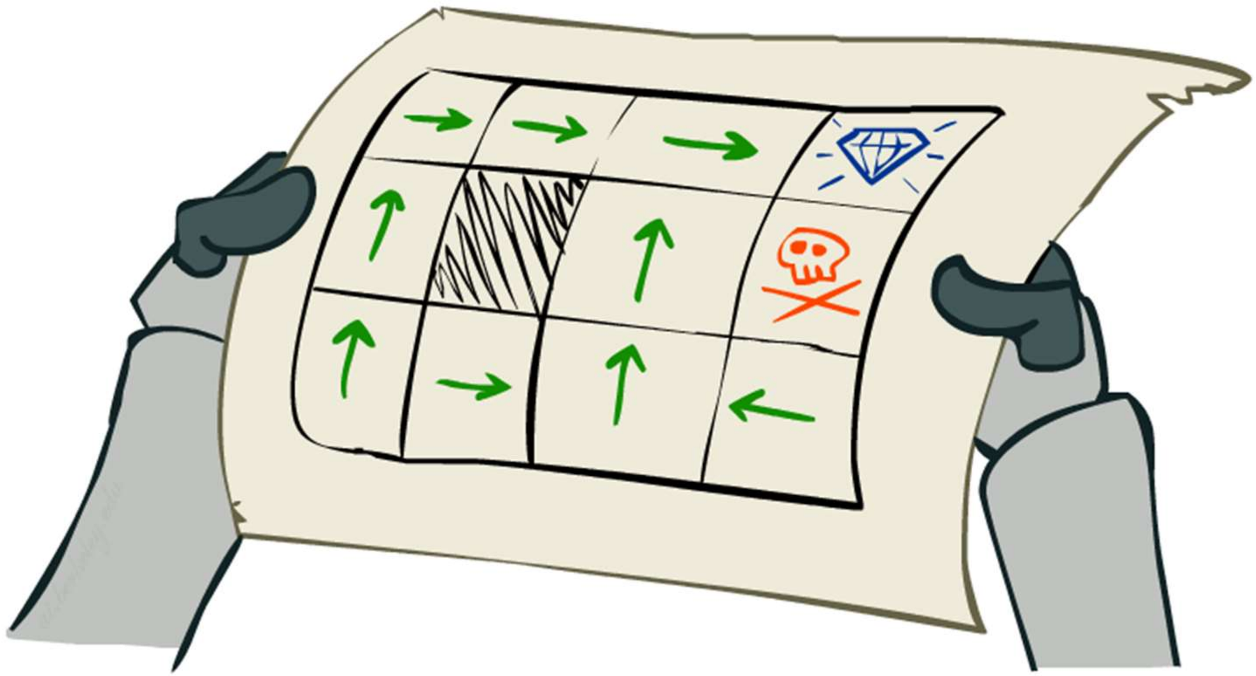
- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions $P(s'|s,a)$ (or $T(s,a,s')$)
  - Rewards $R(s,a,s')$ (and discount γ)

- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards



Tampere University

Solving MDPs



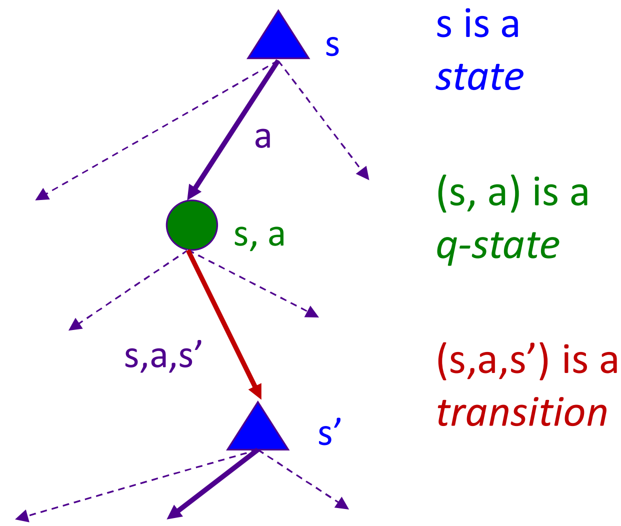
Tampere University

Optimal Quantities

- The value (utility) of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s, a):
  $Q^*(s, a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:
  $\pi^*(s)$ = optimal action from state s



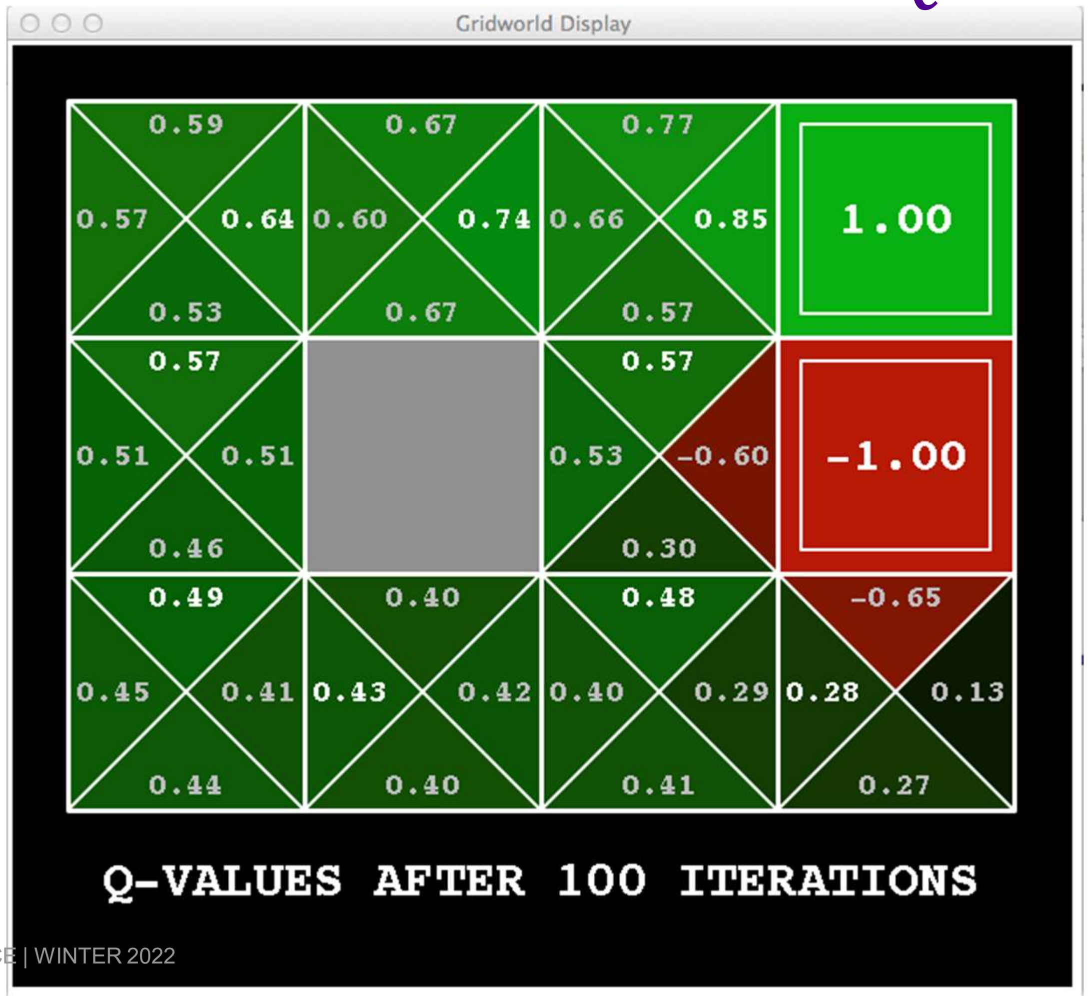
[Demo – gridworld values (L8D4)]

Snapshot of Demo – Gridworld V Values

Noise = 0.2
Discount = 0.9
Living reward = 0

Tampere University

Snapshot of Demo – Gridworld Q Values



Noise = 0.2
Discount = 0.9
Living reward = 0

Tampere University

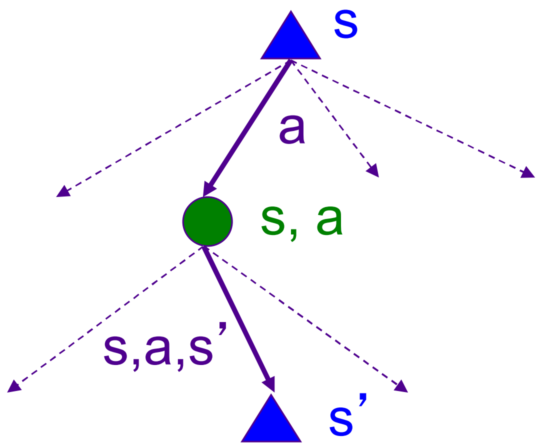
Values of States

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computes!
- Recursive definition of value:

$$V^*(s) = \max_a Q^*(s,a)$$

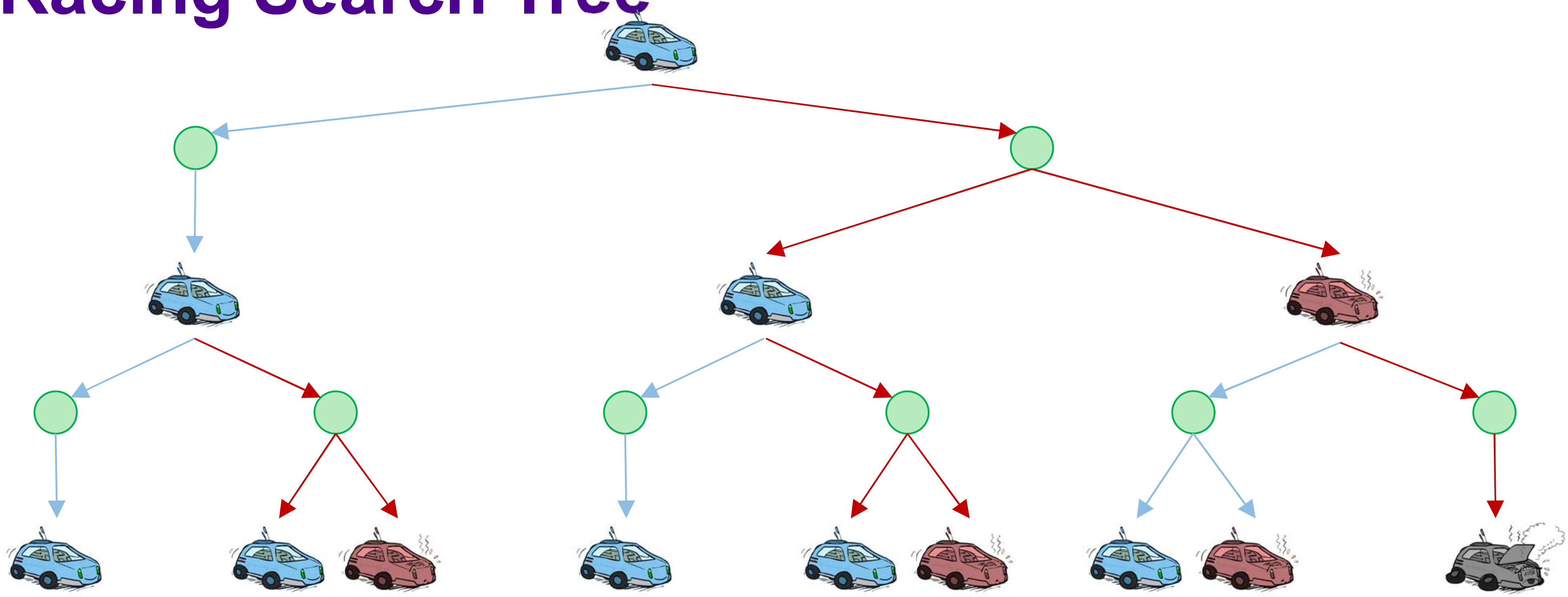
$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$



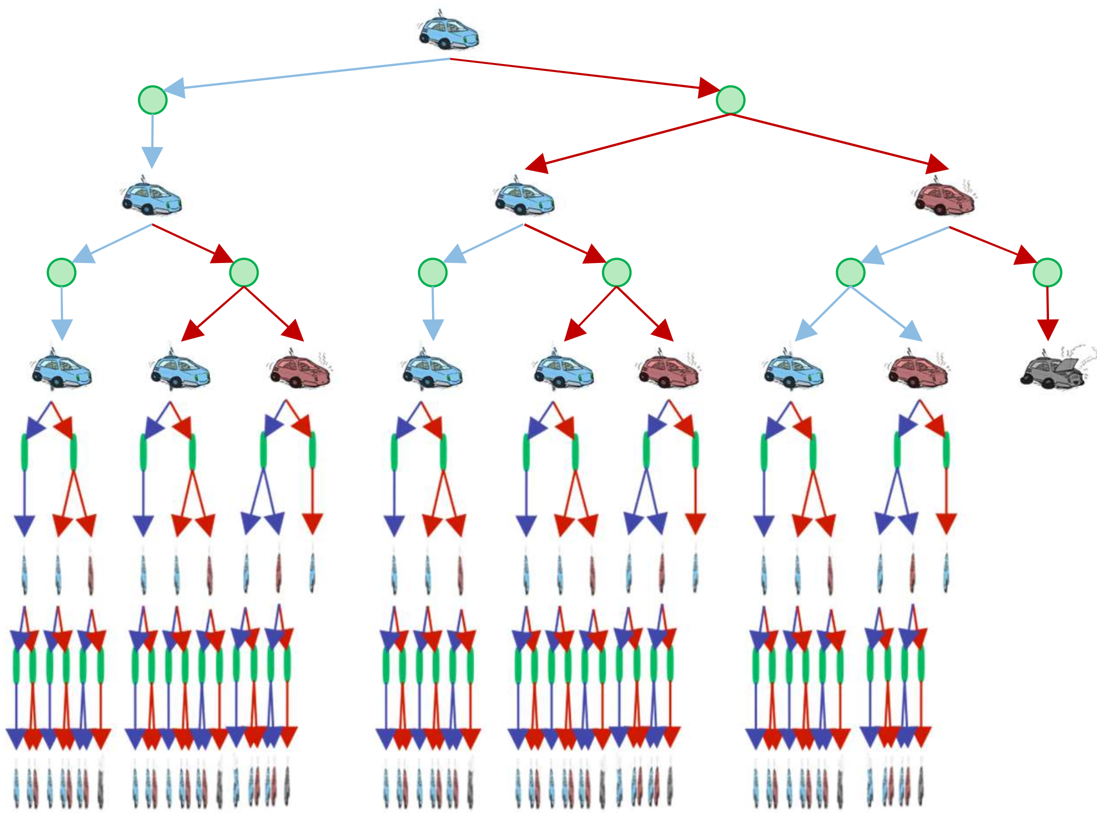
Tampere University

Racing Search Tree



Tampere University

Racing Search Tree

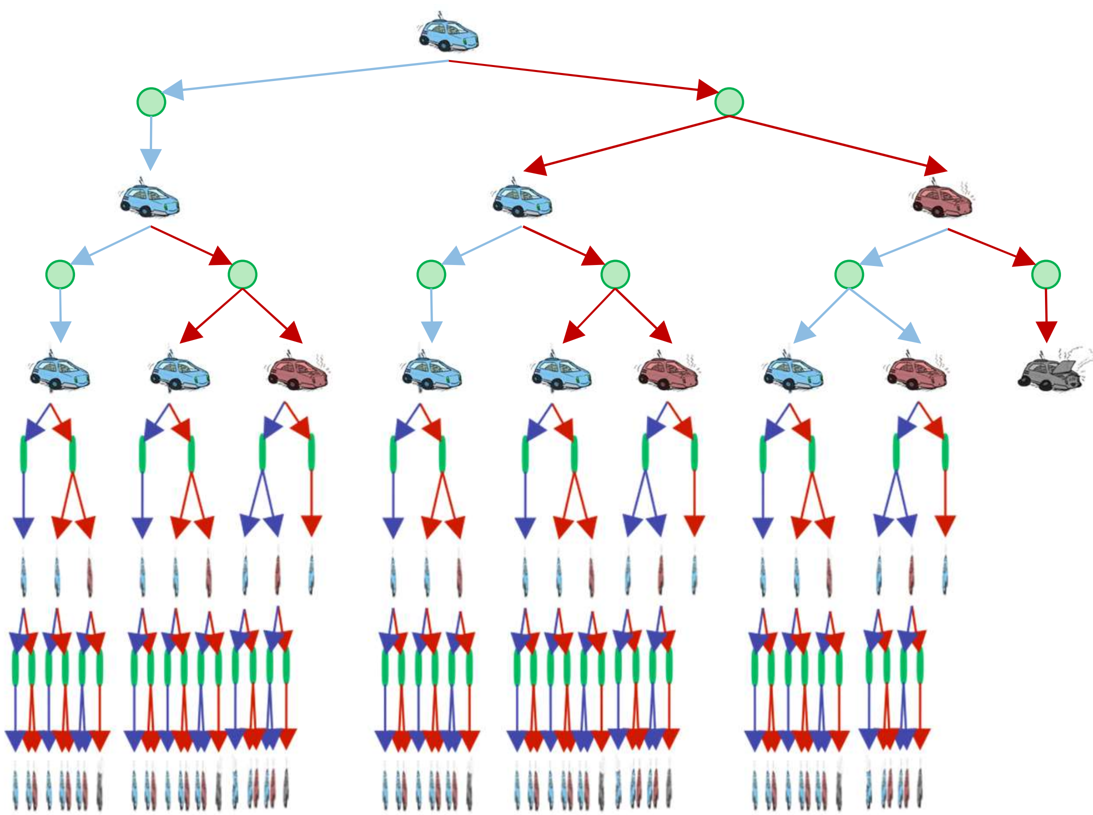


Racing Search Tree

- We're doing way too much work with expectimax!

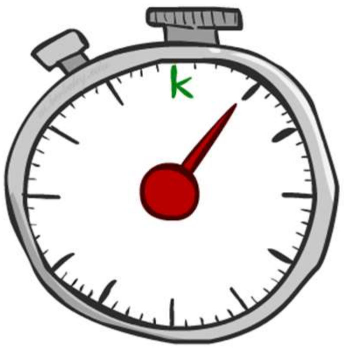
- Problem: States are repeated
  - Idea: Only compute needed quantities once

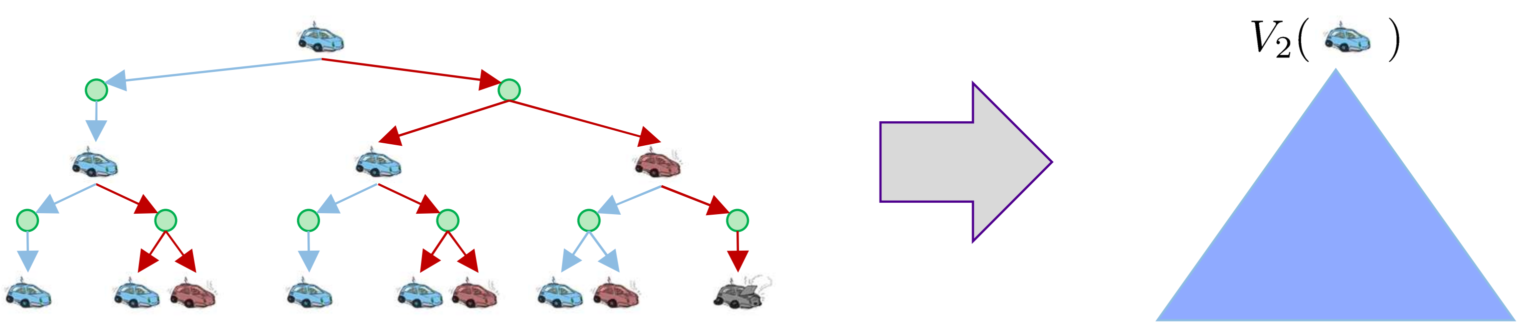
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don't matter if $\gamma < 1$



Time-Limited Values

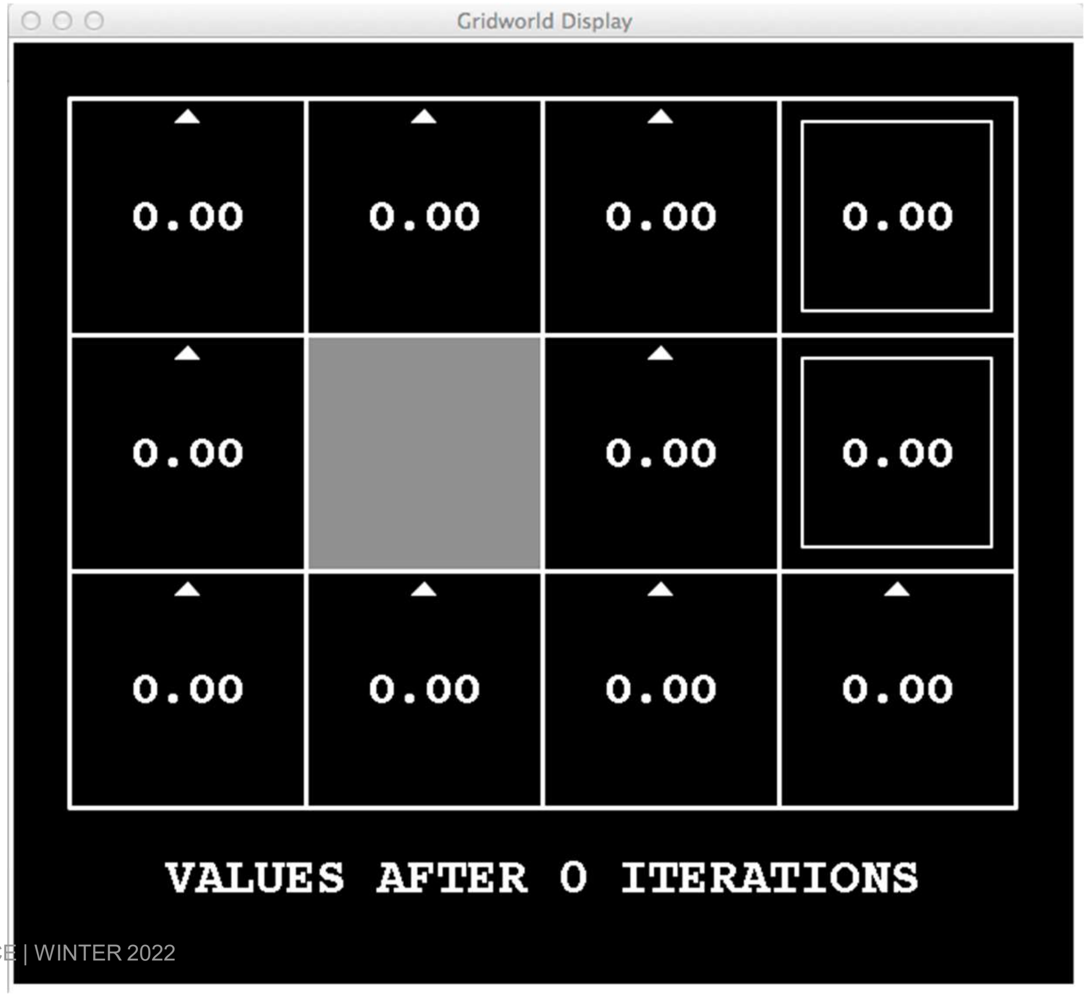
- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s





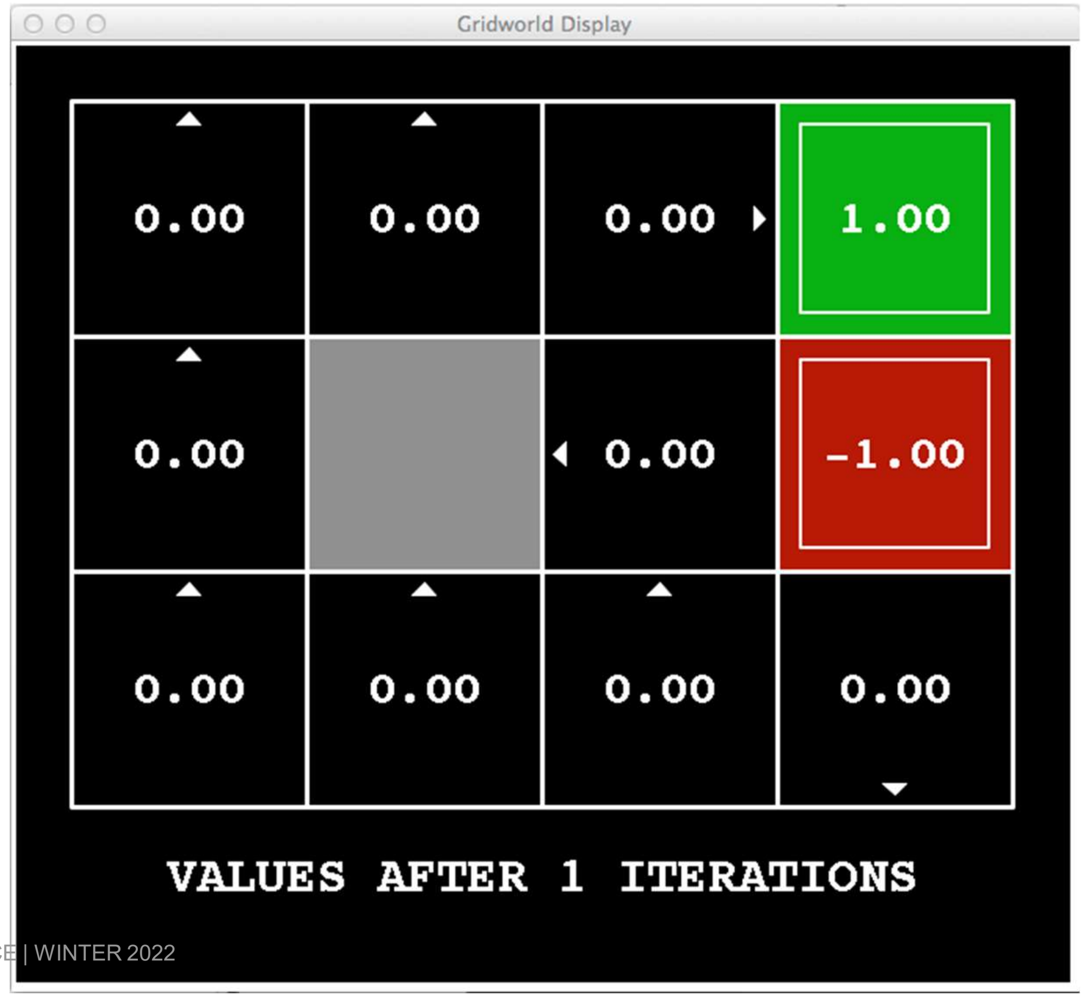
[Demo – time-limited values (L8D6)]

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

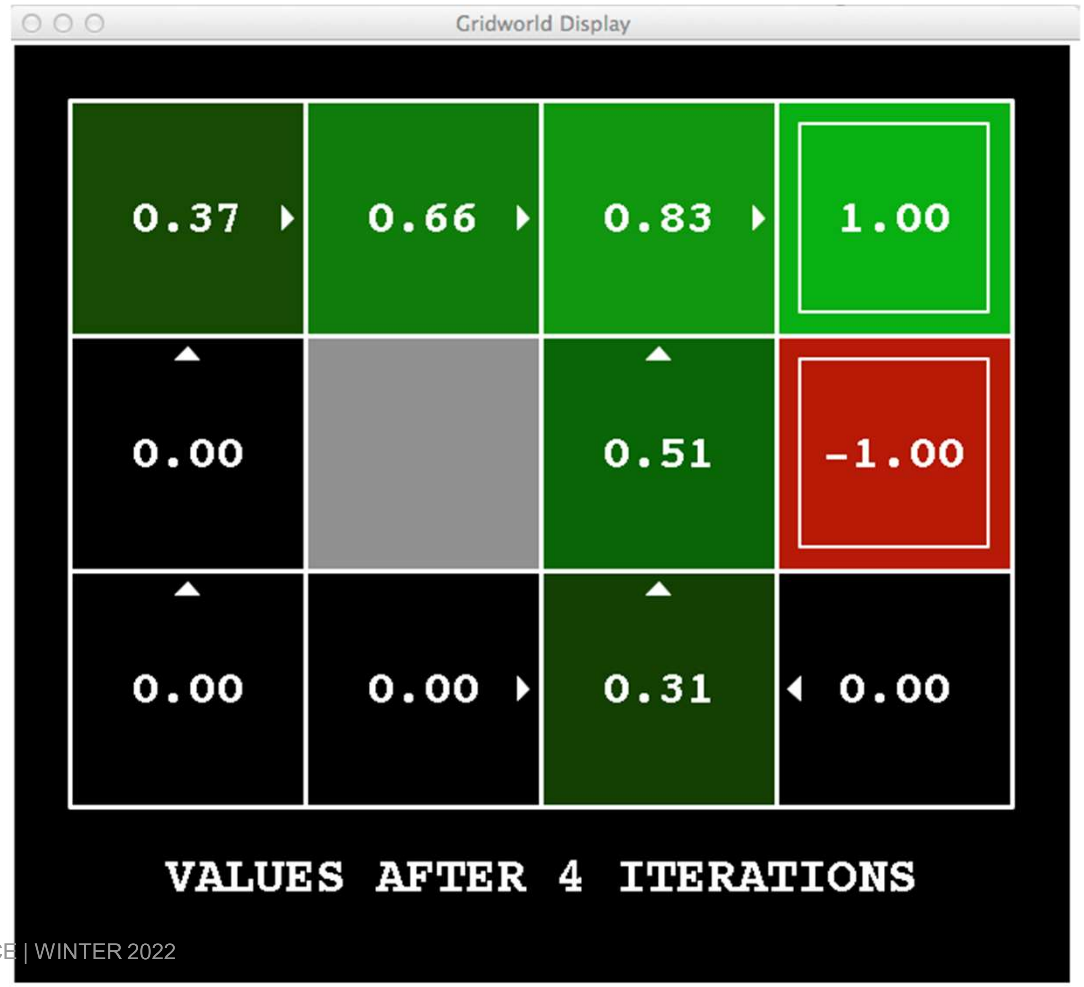
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

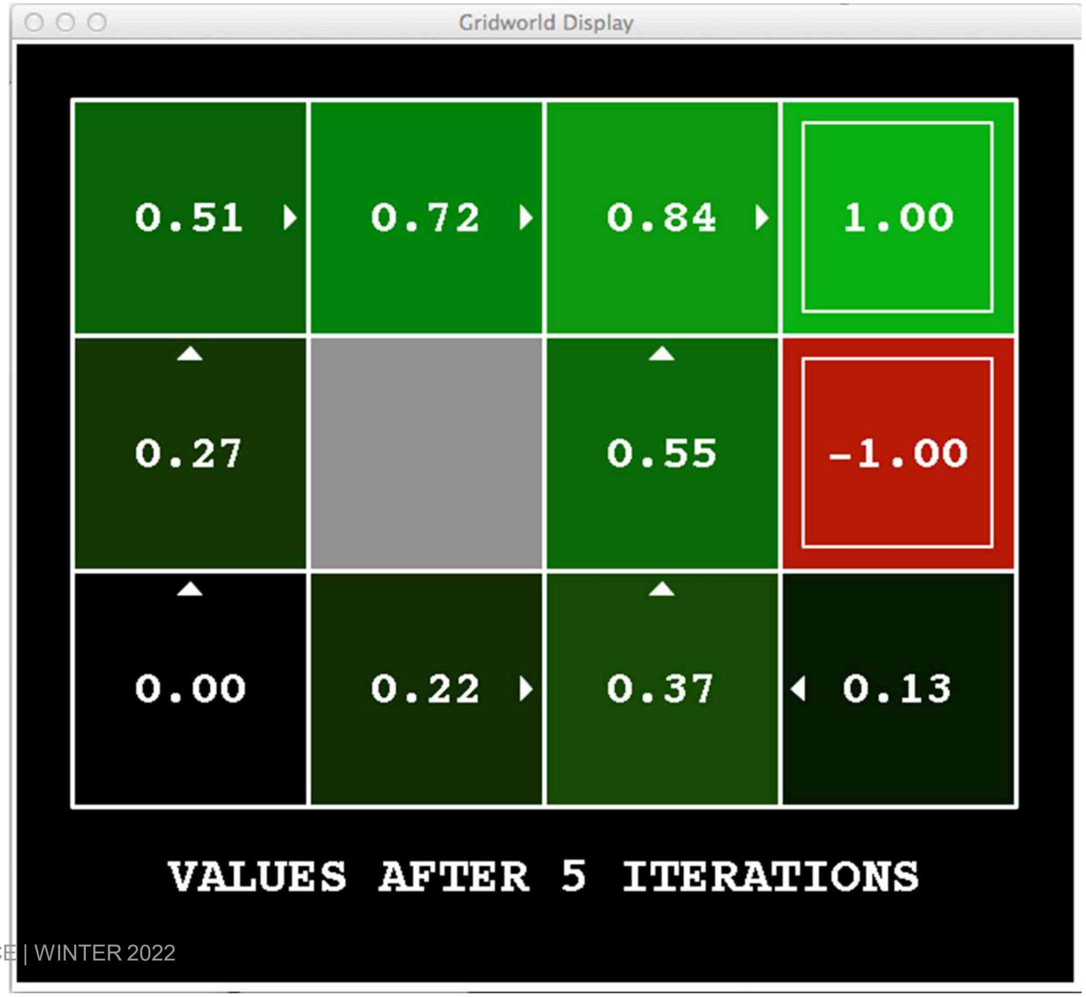
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



Noise = 0.2
Discount = 0.9
Living reward = 0

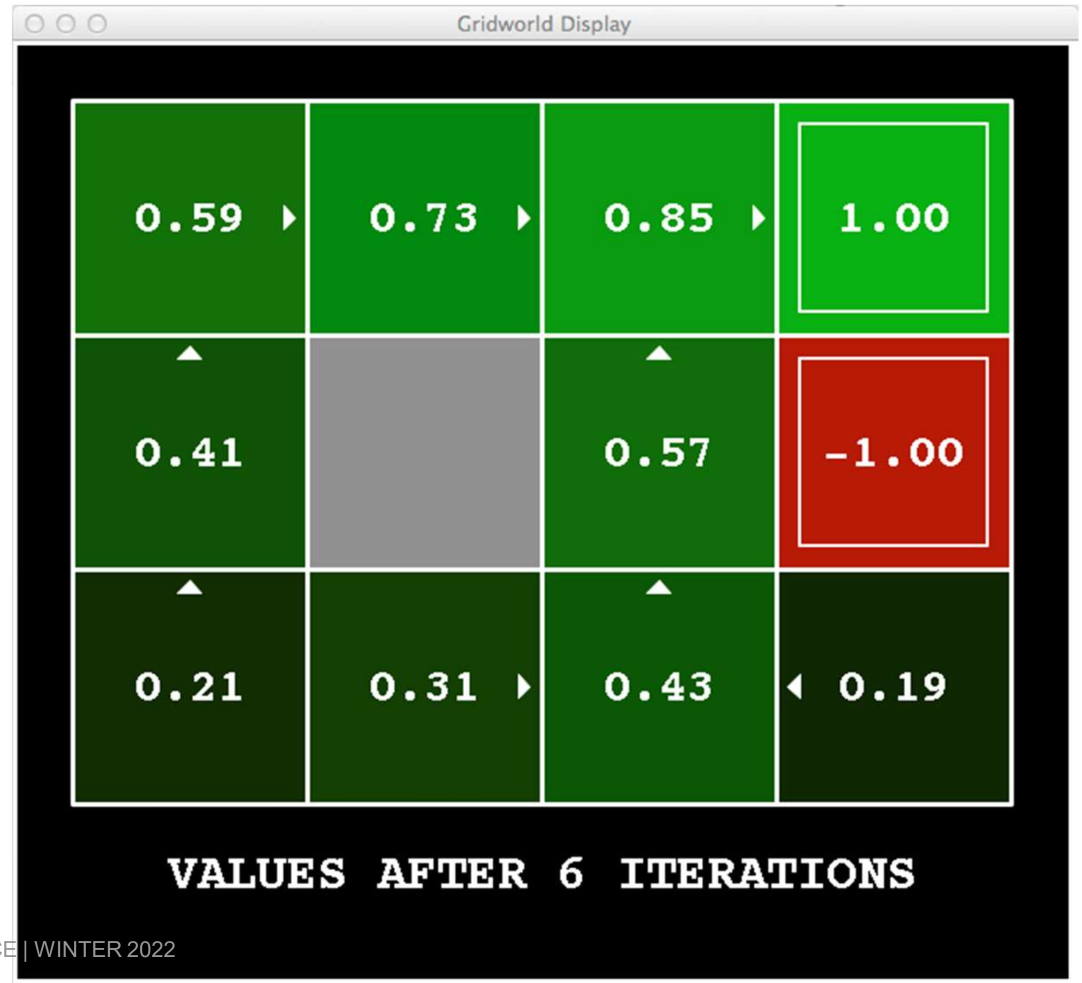
k=5



Noise = 0.2
Discount = 0.9
Living reward = 0

Tampere University

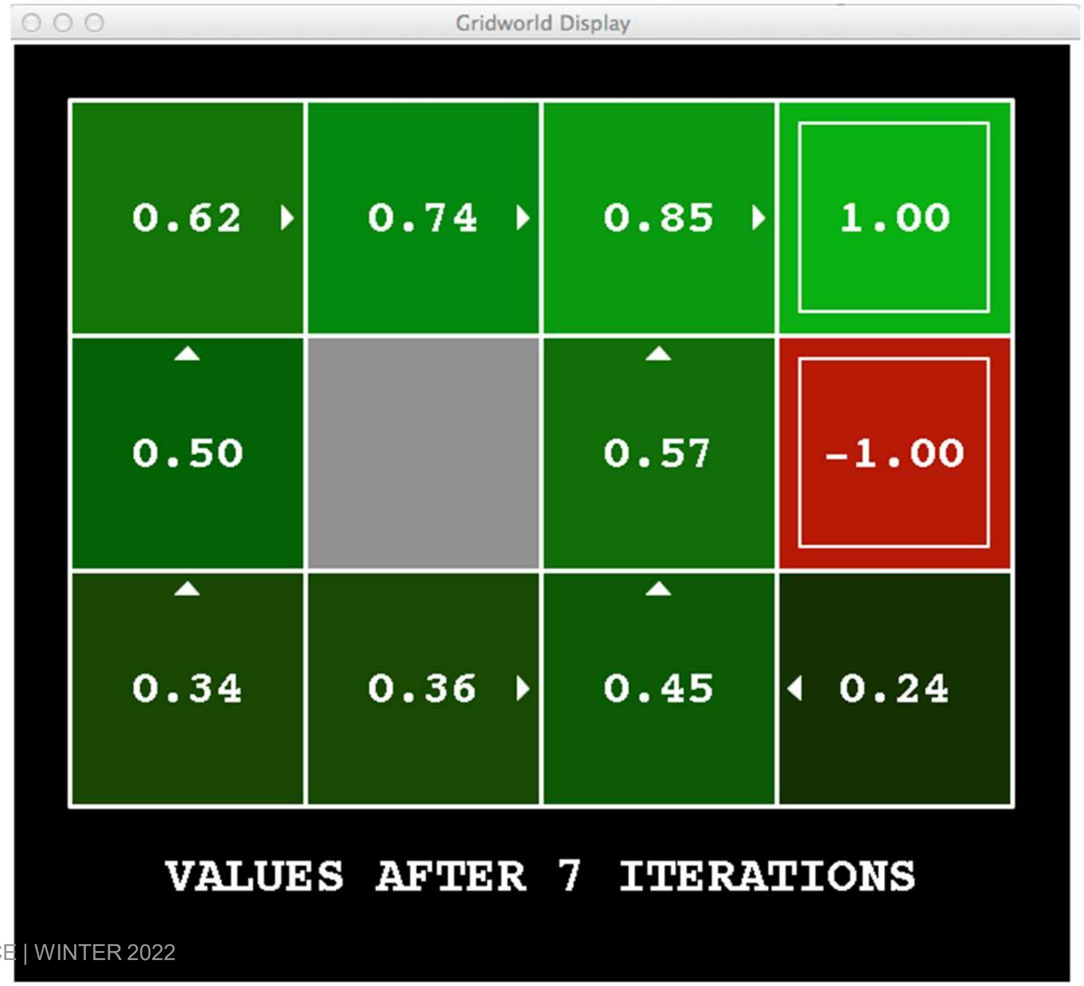
k=6



Noise = 0.2
Discount = 0.9
Living reward = 0

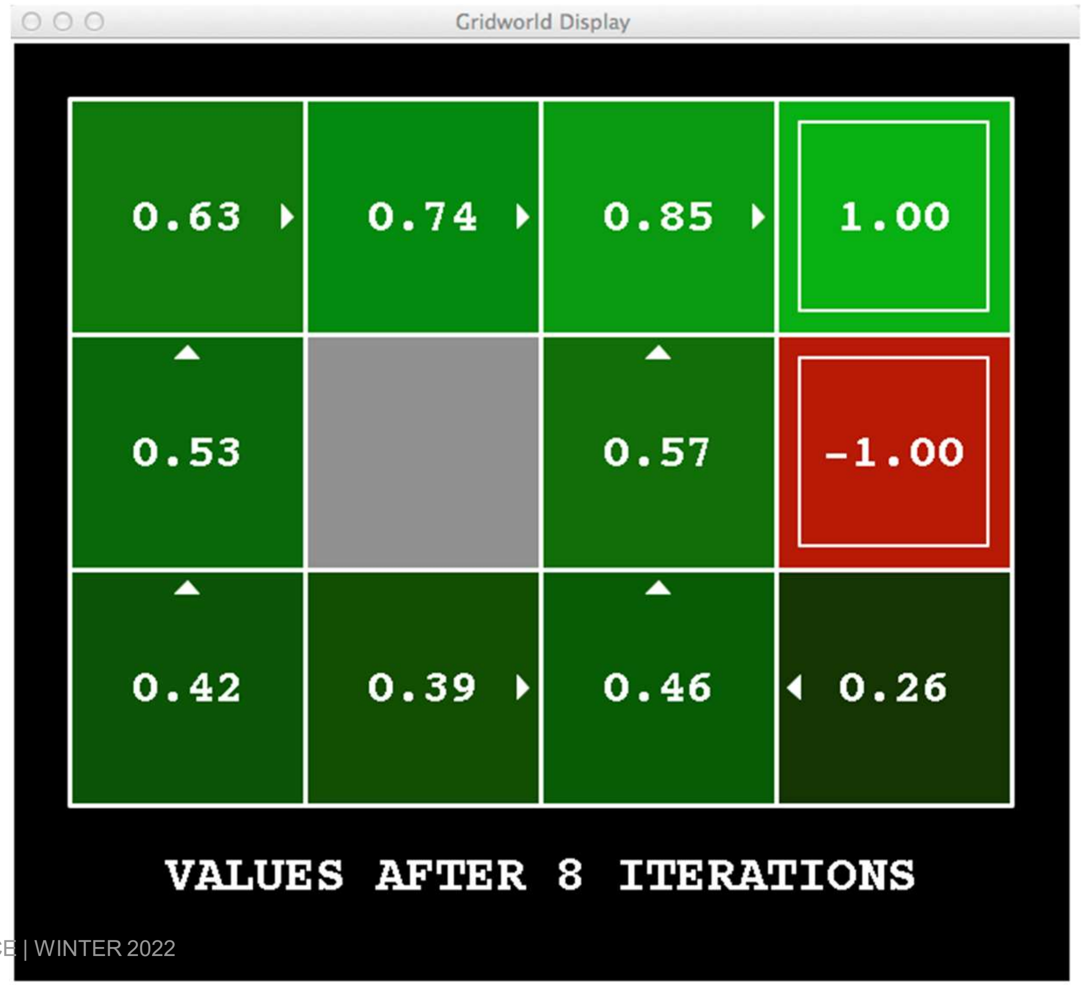
Tampere University

k=7



Noise = 0.2
Discount = 0.9
Living reward = 0

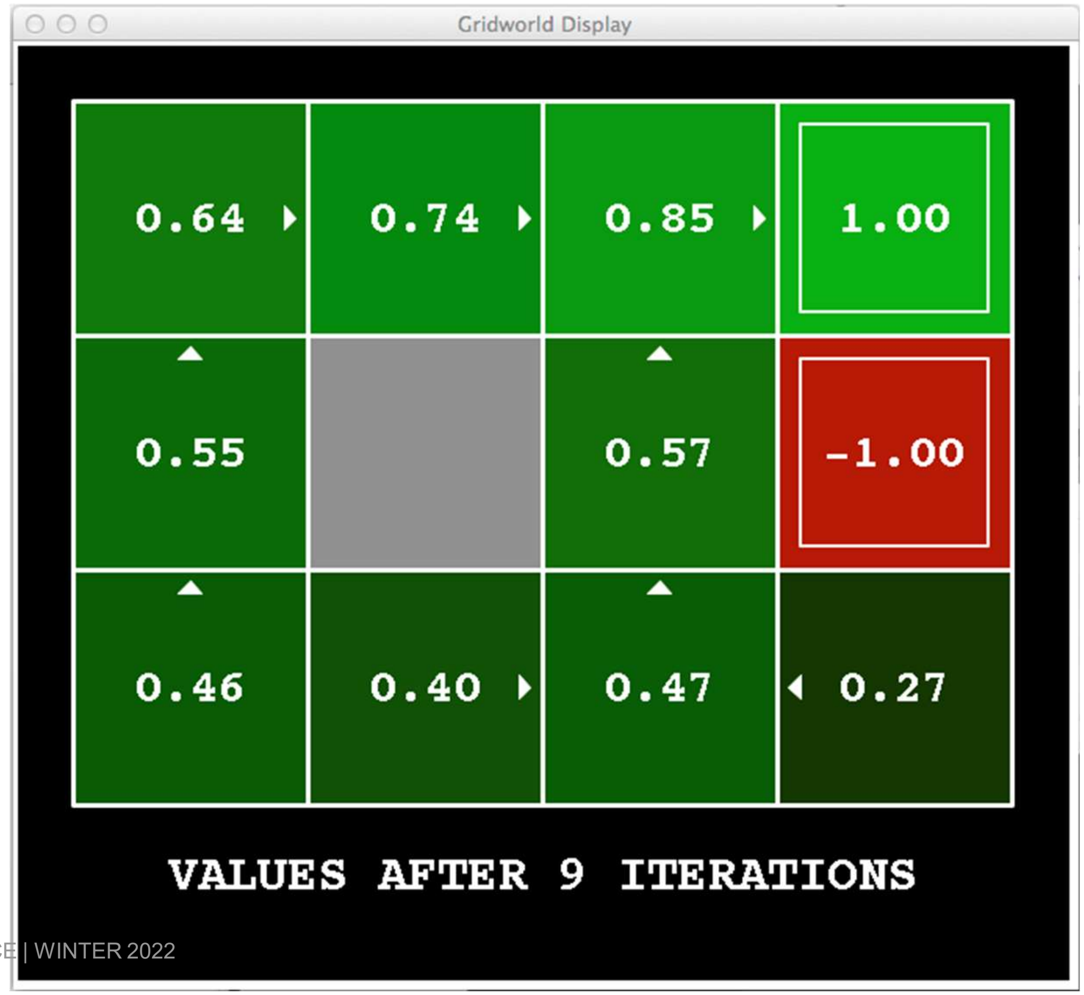
k=8



Noise = 0.2
Discount = 0.9
Living reward = 0

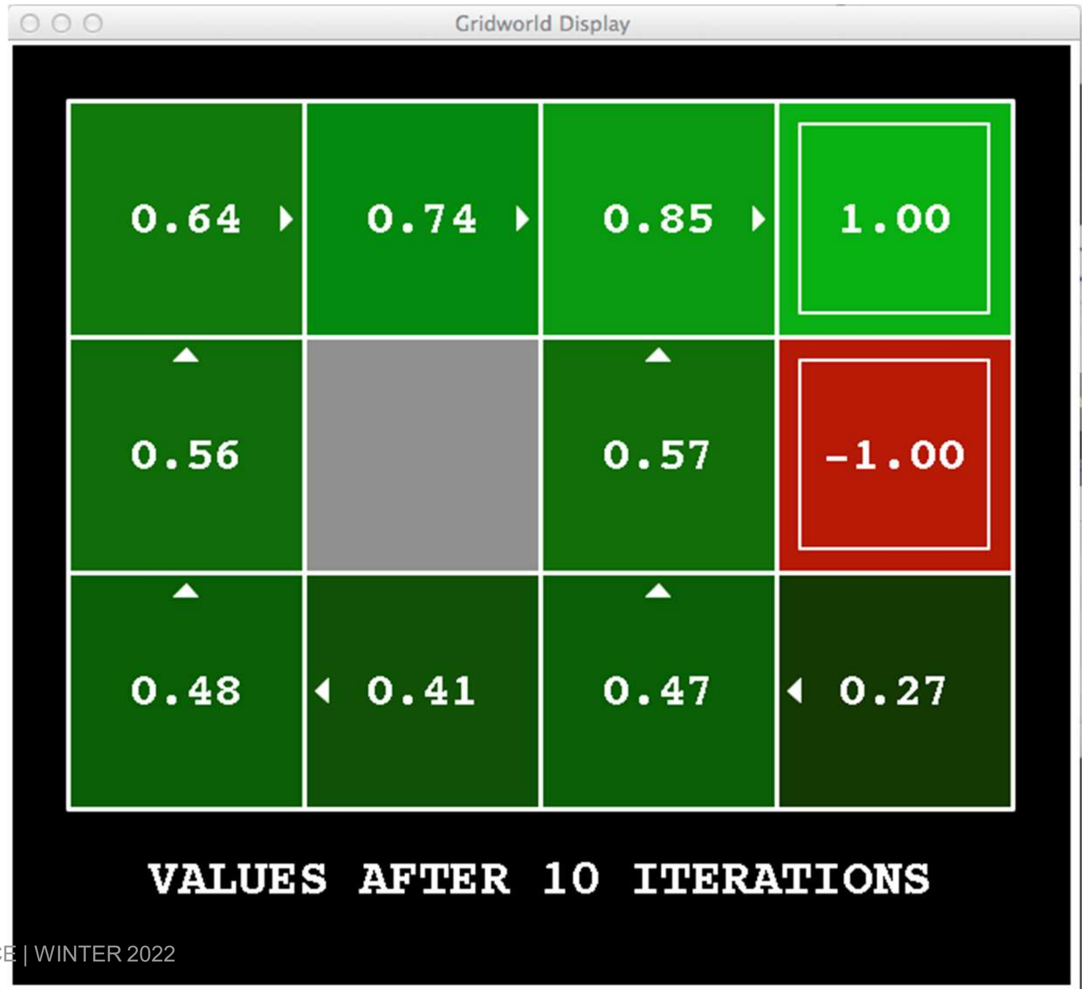
Tampere University

k=9



Noise = 0.2
Discount = 0.9
Living reward = 0

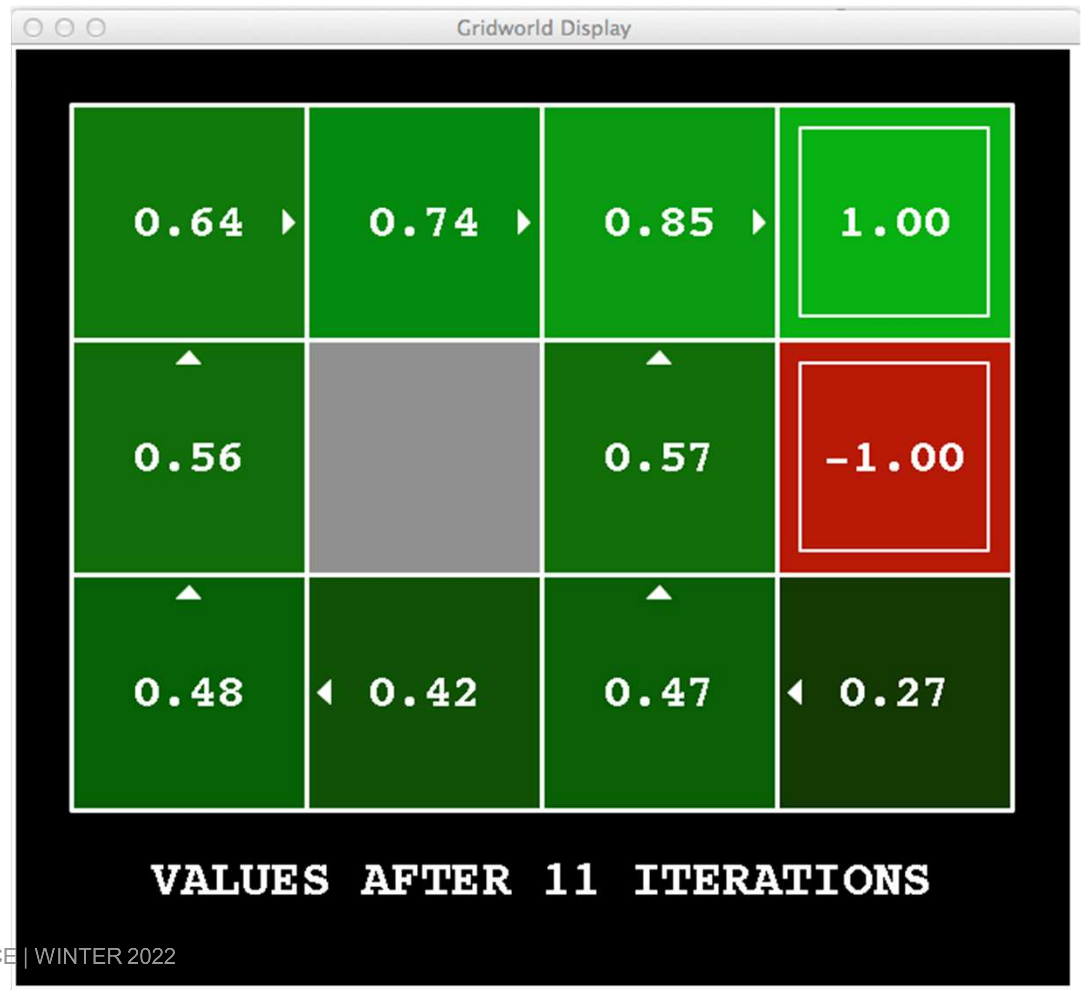
k=10



Noise = 0.2
Discount = 0.9
Living reward = 0

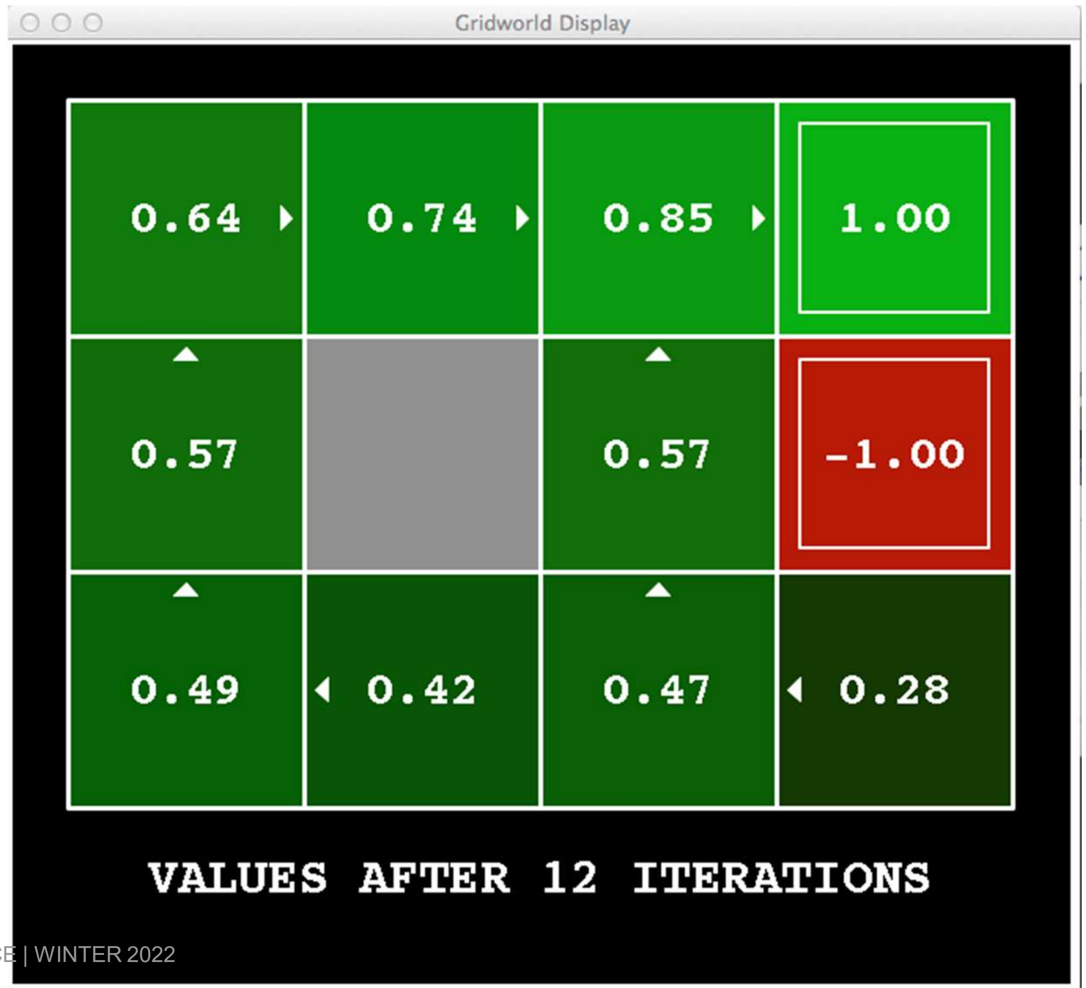
Tampere University

k=11



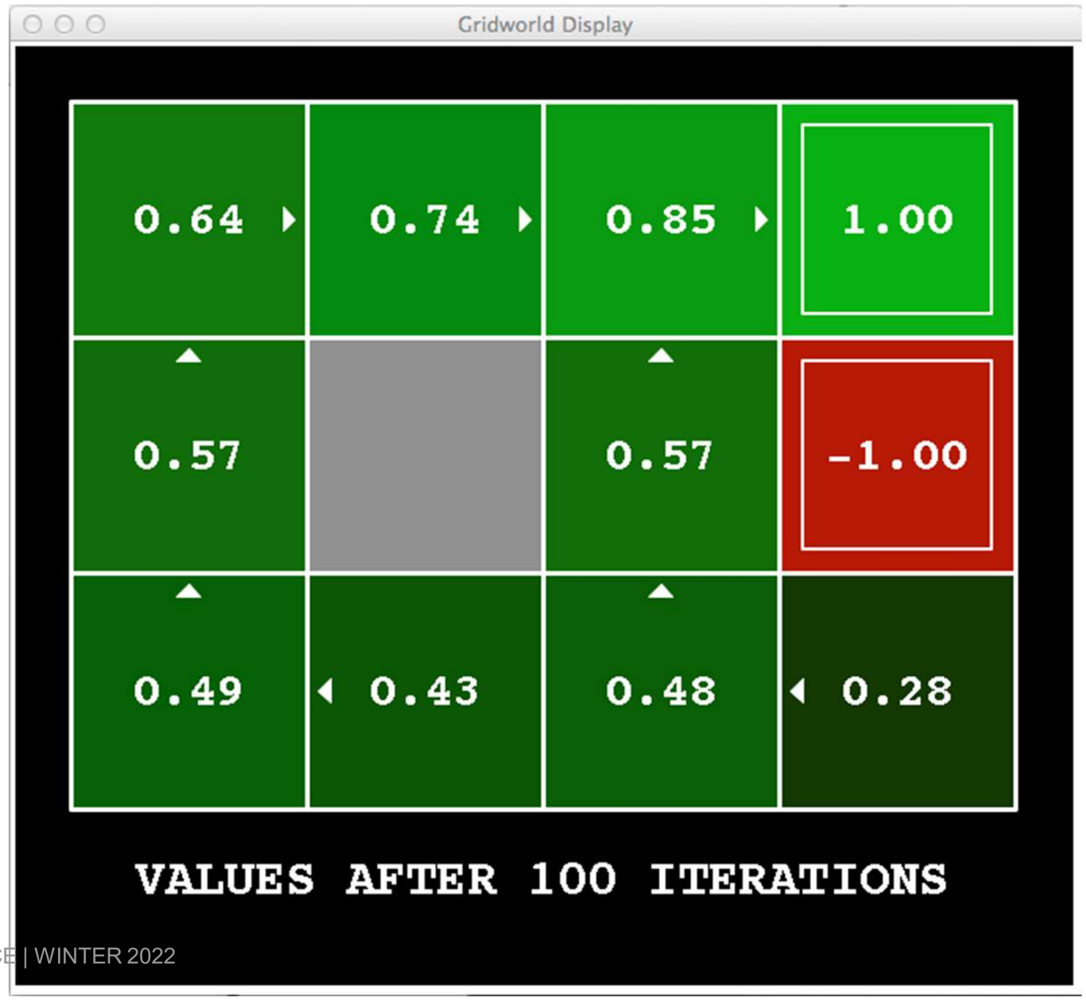
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Tampere University

Computing Time-Limited Values