Markov Decision Processes (MDP)

M4-1

Chun Kai Ling

15 Feb 2018

This lecture

- First half: modelling
  - Examples of MDPs
  - Learn how to model MDPs
  - Familiarize with how the parameters of MDPs affect solutions
- Second half: solving
  - Value iteration
  - Policy iteration
- Extensions (optional)

The story so far…

- Equilibrium concepts in games (Nash, SE)
- Decision Trees
- Graphical Models (GMM, DBN, MRF)

$\}$ Focus is on prediction, not control!

Intelligence is more than just predicting `stuff'

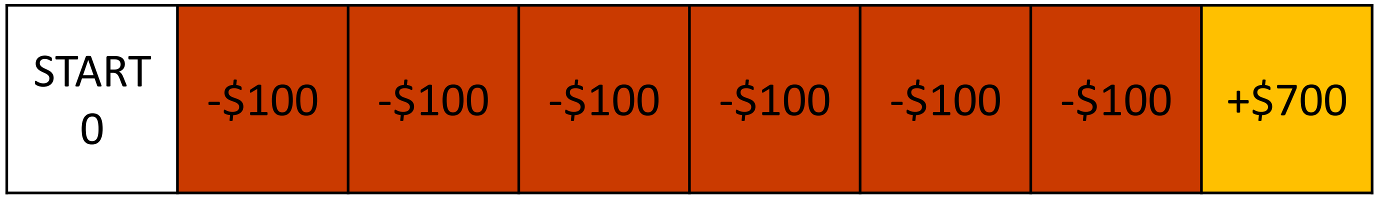
Loosely speaking:

MDPs: Markov Chain + Reward + Control

boring

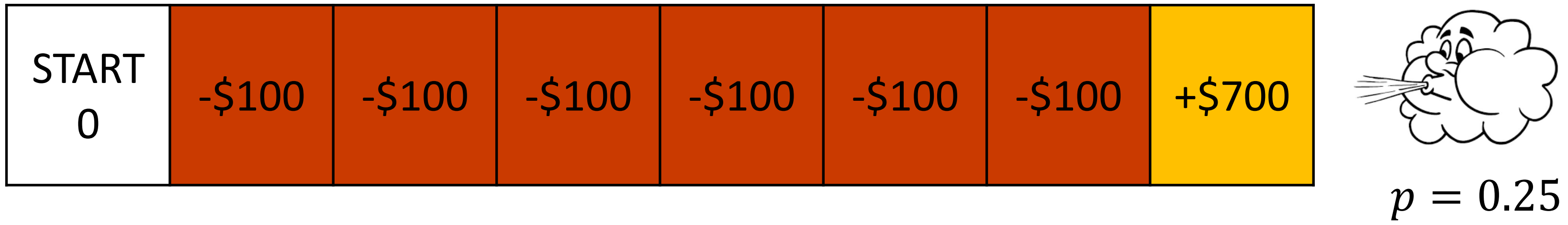
fun

“Extremely exciting” game (EEG)



- Choose to move **left**, **right** or **keep still** at each time step
    - Restricted to staying within bounds
- The game ends once you reach the rightmost tile.
- Suffer penalties every time step when in red tiles.
- Best strategy = ?

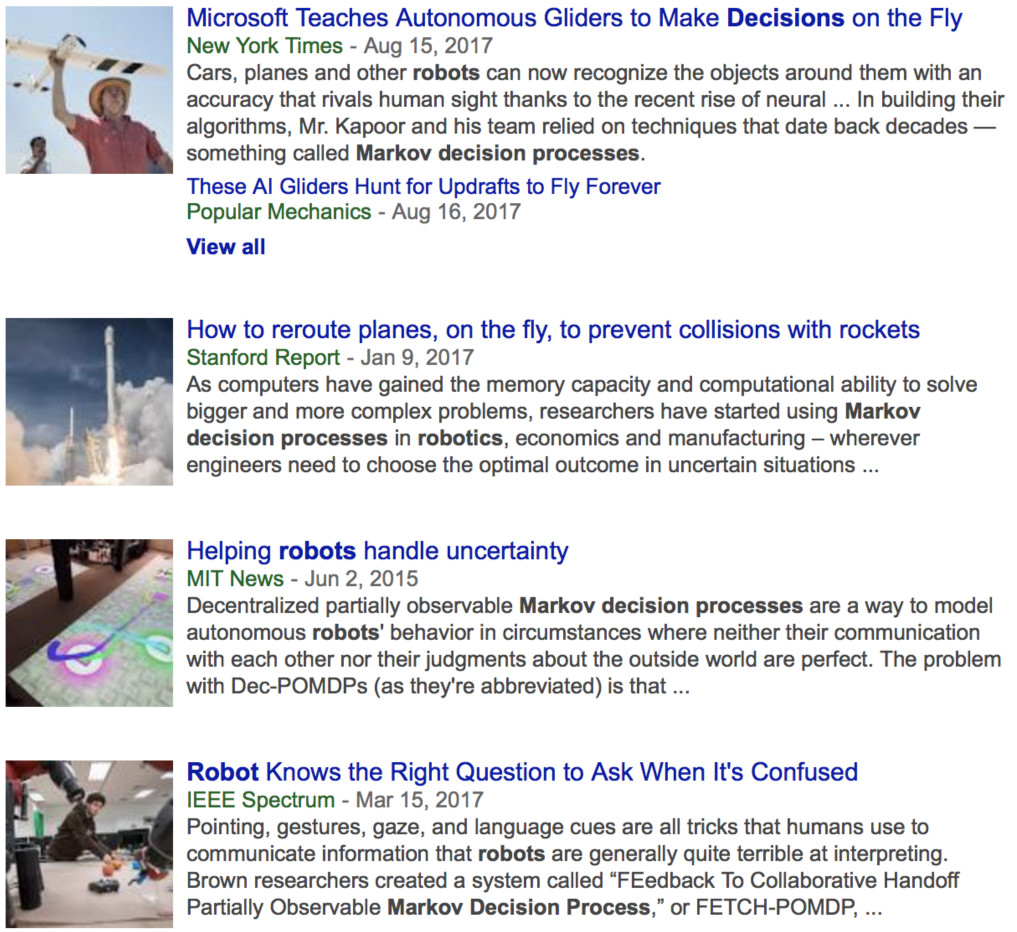
“Extremely exciting” game (Part 2)



- Choose to move **left**, **right** or **keep still** at each time step
  - Restricted to staying within bounds
- The game ends once you reach the rightmost tile.
- Suffer penalties every time step when in red tiles.
- A strong wind is blowing leftward, causing you to move back with probability $p = 0.25$ regardless of your action.
- Best strategy = ?
- Not obvious if penalties / probabilities are non-uniform.

Foundation for many cool applications

- Robotics
- `Game' playing
- Traffic Control
- Server management
- Modelling human behavior
- Useful starting point for many problems!
  - Go, Atari games
- Closely related areas
  - Reinforcement Learning
  - Imitation learning, Inverse RL



Microsoft Teaches Autonomous Gliders to Make **Decisions** on the Fly
New York Times - Aug 15, 2017
Cars, planes and other **robots** can now recognize the objects around them with an accuracy that rivals human sight thanks to the recent rise of neural ... In building their algorithms, Mr. Kapoor and his team relied on techniques that date back decades — something called **Markov decision processes**.
These AI Gliders Hunt for Updrafts to Fly Forever
Popular Mechanics - Aug 16, 2017
View all

How to reroute planes, on the fly, to prevent collisions with rockets
Stanford Report - Jan 9, 2017
As computers have gained the memory capacity and computational ability to solve bigger and more complex problems, researchers have started using **Markov decision processes** in **robotics**, economics and manufacturing – wherever engineers need to choose the optimal outcome in uncertain situations ...

Helping **robots** handle uncertainty
MIT News - Jun 2, 2015
Decentralized partially observable **Markov decision processes** are a way to model autonomous **robots'** behavior in circumstances where neither their communication with each other nor their judgments about the outside world are perfect. The problem with Dec-POMDPs (as they're abbreviated) is that ...

Robot Knows the Right Question to Ask When It's Confused
IEEE Spectrum - Mar 15, 2017
Pointing, gestures, gaze, and language cues are all tricks that humans use to communicate information that **robots** are generally quite terrible at interpreting. Brown researchers created a system called “FEedback To Collaborative Handoff Partially Observable **Markov Decision Process**,” or FETCH-POMDP, ...

Google news for ‘Markov Decision Process robotics’

If you have no idea how to proceed, formulating something as an MDP and doing Monte-Carlo Tree search probably works reasonably well…

- A wise man

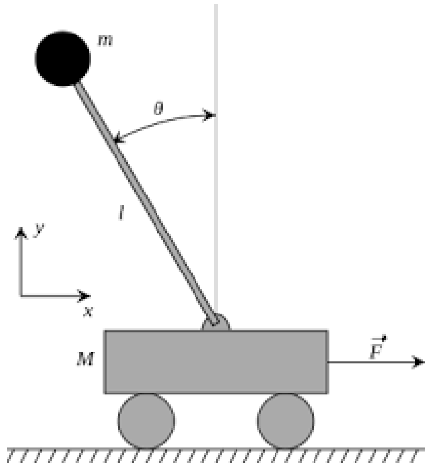
Videos

MDP = (S, A, T, R, γ)

- **S:** set of states, $s_t \in S$ (where can I be?)
- **A:** set of actions, $a_t \in A$ (what can I do?)
- **T**ransition function: $P(s_{t+1}|s_t, a_t)$ (what happens next?)
- **R**eward function $r_t = R(s_t, a_t)$ (what do I gain?)
- $\gamma \in [0, 1]$ (discount factor)

Markov Chains!

$s_t = \{i | \text{location } i \text{ occupied}\}$
$a_t \in \{\text{left, right, rotate}\}$
$T = \text{some tetris physics,}$
$\text{"distribution of next tile"}$
$R(s_t) = \#\text{complete rows}$



$s_t = (x_t, x_t', \theta_t, \theta_t')$
$a_t \in \{\text{left, right}\}$
$T = \text{some physics}$
$R(s_t) = \text{height of pole}$

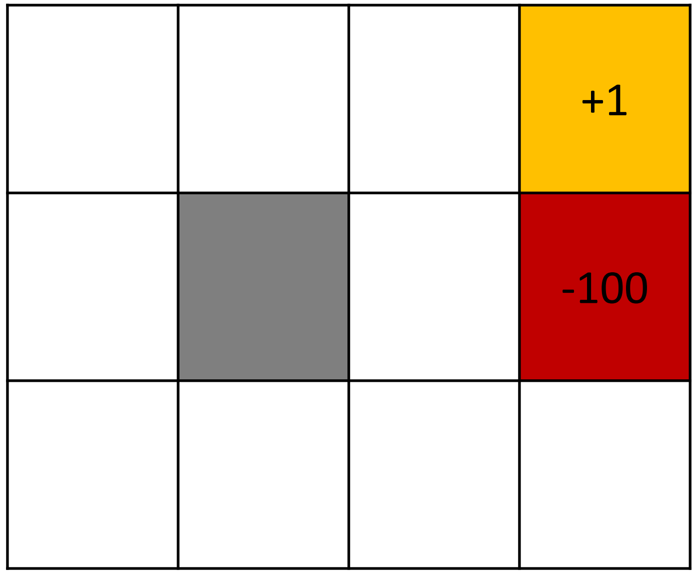
Q1: What are S,A,T,R for the previous game?
Q2: How large is $|S|$?

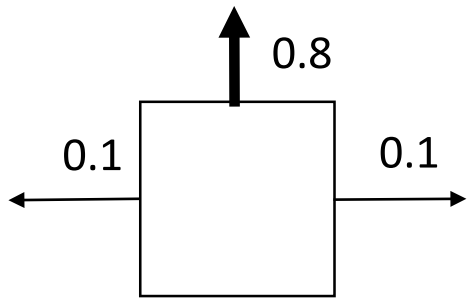
Behaving (near)-optimally in an MDP

- Assume $t = 0,1,2 \ldots$
- Total expected *discounted* reward: $\sum_t \gamma^t r_t, 0 \leq \gamma < 1$
- Payoffs now worth more than in the future
  - Model compound interest, opportunity costs etc.
  - Technical convenience (useful later when solving)
- Define a policy $\pi(s) = a$
  - Maps state to action, defines a *plan*
- Goal: find π to maximize expected discounted reward
  - $\pi^* = \mathrm{argmax}_\pi \mathbb{E}_\pi[\sum_t \gamma^t R(s_t, \pi(s_t))]$
- Myopic strategy does not work: a_t affects future states

Grid World

- The canonical example for MPDs
- Move in desired direction with probability 0.8
- Move in perpendicular direction with probability 0.1
- No movement when hitting walls





What is the best policy?

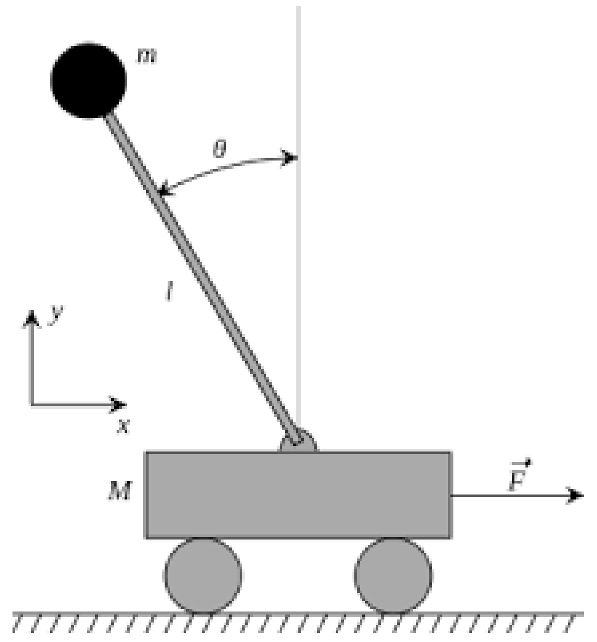
Behaving (near)-optimally in an MDP

- Q1: Should π depend on t? Or just s_t? Do we have to care about histories of states and actions?
  - Independence from t is reasonable due to Markov Property
- Q2: Could we ever benefit from non-deterministic π ?
  - No: could always do better by switching to the more rewarding deterministic policy.
  - Grey Area: when there are *aliased* states (not in this lecture)

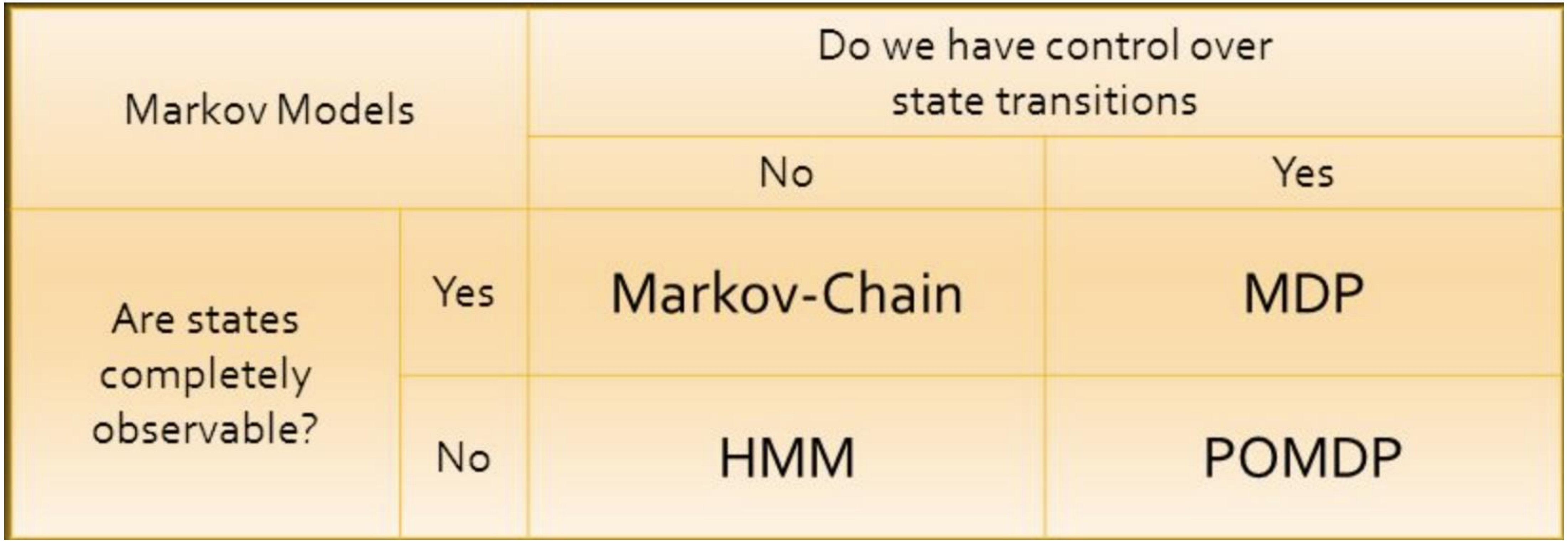
M4-1 Quiz 1 (Markov Property)

Which of the following state representations are suitable for pole-cart balancing ?

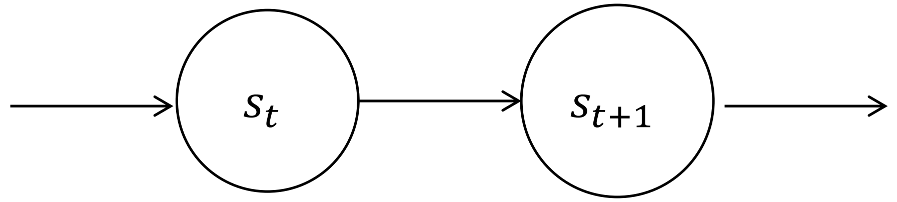
A) $s_t = (x_t, x'_t, \theta_t, \theta'_t)$
B) $s_t = (x_t, \theta_t)$
C) $s_t = (x_t, x_{t-1}, \theta_t, \theta_{t-1})$
D) $s_t = (x'_t, \theta_t, \theta'_t)$



Relationship to other models (optional)

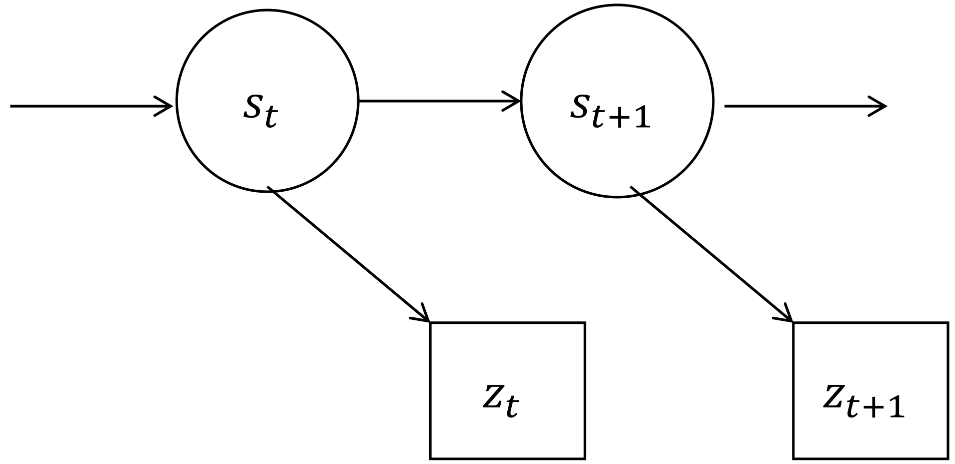


Markov Models		Do we have control over state transitions	
		No	Yes
Are states completely observable?	Yes	Markov-Chain	MDP
	No	HMM	POMDP



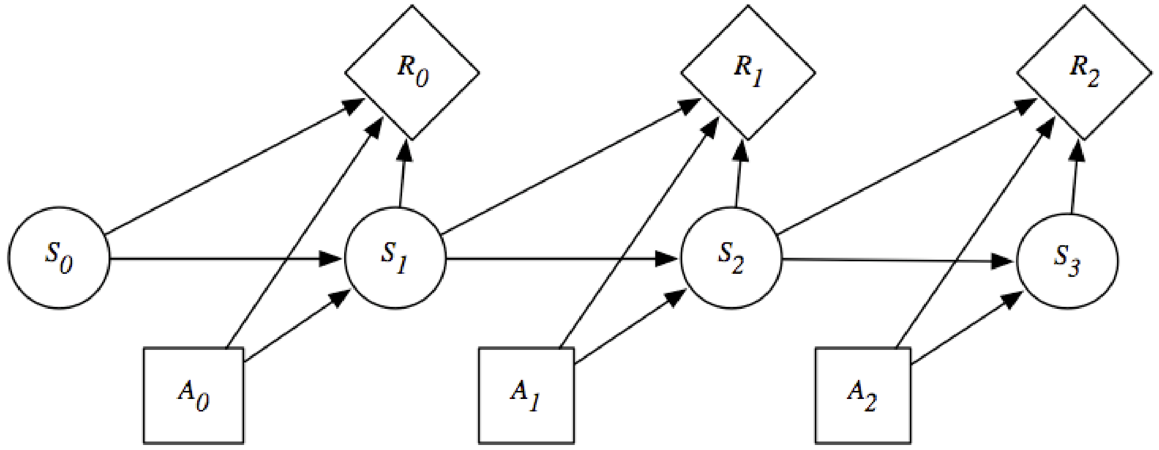
Markov Chains

- Find stationary/limiting distribution
- Find time to first return
- Find distribution of states at time t given state at time t-n



HMM

- Tractable inference, Viterbi etc.
- Filtering, e.g. Kalman filters, EKF, IKF …
- Sample based filtering, particle, histogram filters…



Markov Decision Processes

Classic theory on MDPs borrow from Markov Chain theory

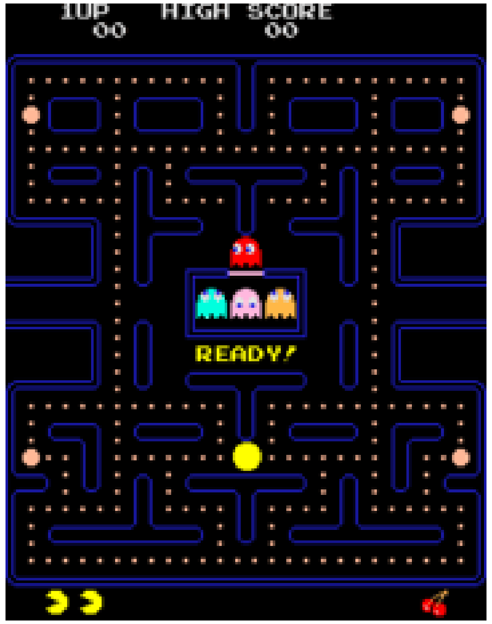
Are these well modelled by MDPs?

Poker

Monopoly

2048

Mario



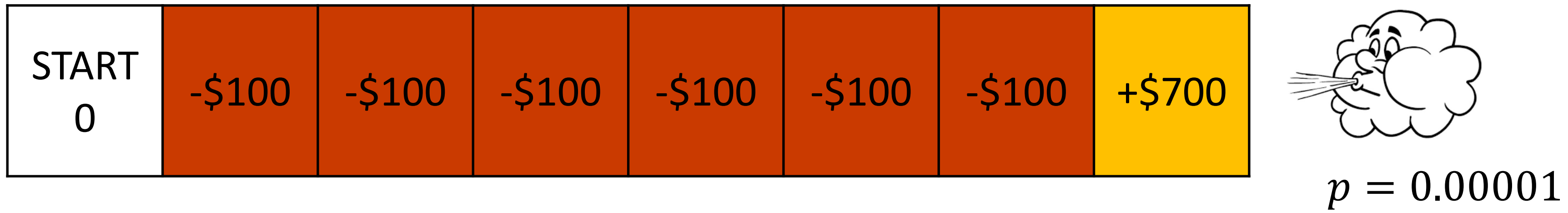
Pacman

Minesweeper

Snake and ladders

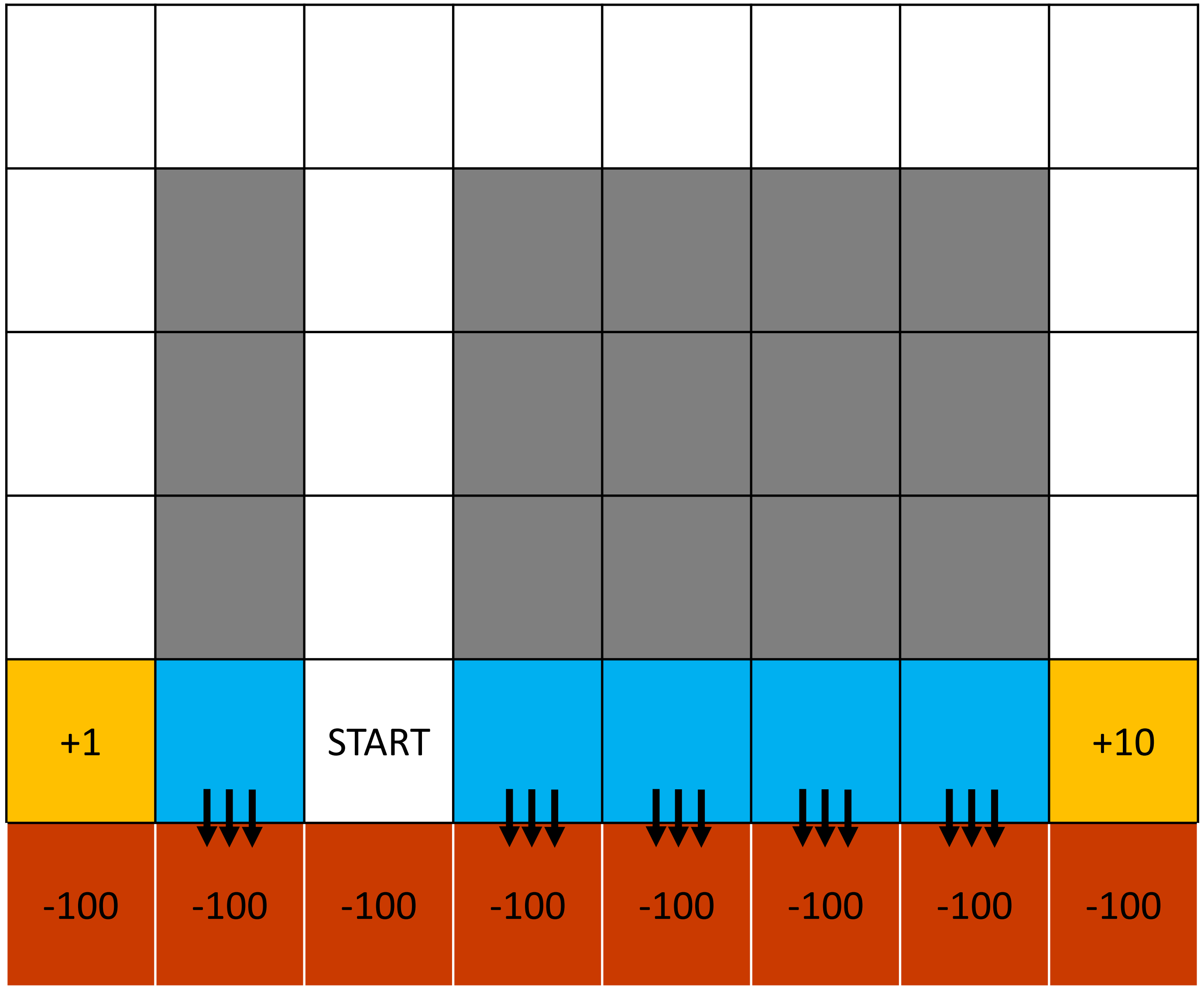
Importance of discount factors & time

- Infinite horizon discounted reward is not the only formulation
  - Average reward (not in this lecture)
  - Finite horizons



- What happens if $\gamma = 0.25$? What about $\gamma = 0.99$?
  - Future discounted reward is too low to compensate for short-term pain
- What if the game only carries on for $t_{\max} = 8$ steps?
  - Impossible to reach goal after being pushed back once.
  - Time-dependent policy $\pi_t(s)$, or augmented state (s, t)
  - Dynamic programming

M4-1 Quiz 2



Legend:

Grey: walls, Red: cliff (terminal state), Orange: gold (terminal state), Blue: slippery slope

Actions:

- up, down, left, right
- Taking any action on blue tiles causes you to fall down with probability p.

Q: Which of the settings for gamma and p result in an optimal agent's first action to be "Left"?

A: $\gamma = 0.1,\ p = 0$

B: $\gamma = 0.99999,\ p = 0$

C: $\gamma = 0.1,\ p = 0.2$

D: $\gamma = 0.99999,\ p = 0.2$

Solving MDPs (not exhaustive!)

- Exact
  - Value Iteration
  - Policy Iteration
  - Linear Programming
- Approximate
  - Sampling based
  - Function approximation

Value Functions (finite horizon H)

- The *value function* at time t, for policy π is
- $V_t^\pi(s) =$ expected total reward assuming
  - We adopt π
  - We begin in s
  - We have t timesteps *remaining*
- Bellman Equations:

$$V_t^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s' | s, \pi(s)) V_{t-1}^\pi(s')$$

$$V_0^\pi = 0$$

- *Value of state = immediate reward + future reward*

Optimal Value Functions (finite horizon)

- The value function for the optimal policy is $V_t^{\pi^*}(s)$
  - Abbreviated as $V_t^*(s)$

$$V_t^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_{t-1}^*(s')\right]$$

$$V_0^* = 0$$

- *Pick the action which maximizes current + future reward (assuming continued optimal behavior)*
- Similar in spirit to dynamic programming.

Value Iteration (Infinite Horizon)

- $V^*(s) = \max_{a \in A} [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')]$
- Self referencing *fixed point* equations!
- Fixed point iteration
    - Pretend we have a really long horizon
    - Perform dynamic programming!
- Initialize $V_0^*(s) \leftarrow 0$
- Iterate $V_{i+1}^*(s) \leftarrow \max_{a \in A} [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_i^*(s')]$
- aka ‘Value update’, ‘Bellman backups/updates’

- Q1: How does V^* help us get the optimal policy?
  - Optimal policy is retrieved using one step look-ahead
  - $\pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \, [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')]$
- Q2: Is the value function always finite?
  - Yes, if $\gamma < 1$
  - Total payoff cannot exceed $\frac{R_{\max}}{1-\gamma}$.
- Q3: Does value iteration converge to V^* ?
  - Yes, use the fact that Bellman backups are a contraction.
- Q4: Is the fixed point V^* unique?
  - Yes.

Value Iteration: Example

START 0	-\$100	-\$100	-\$100	-\$100	-\$100	-\$100	+\$700

$p = 0.1, \gamma = 0.9$

V_0	0	0	0	0	0	0	0	0
V_1	0	-100	-100	-100	-100	-100	-100	+700
V_2	0	-100	-190	-190	-190	-190	458	+700

Demo by TA + Video

Policy Iteration

- Value iteration: V^* estimates gradually improved by Bellman backups
  - Optimal policy is induced by V^*
  - Q: Is it necessary to get an accurate estimate of V^* to induce π^*
- Policy iteration: π gradually ‘improved’.
  - V^π used to evaluate policy
- New idea: Iterate between 2 steps
  - Policy Evaluation (check how good current policy is)
  - Policy Improvement (get a ‘better’ policy)

Policy iteration (con’t)

- Policy Evaluation
  - Method 1: Use value iteration
  - $V_{i+1}^{\pi}(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s,a) V_i^{\pi}(s')$, $V_0^{\pi}(s) \leftarrow 0$
  - Method 2: Solve system of linear equations (no max operator for fixed π)
  - Terminate if Bellman equations holds
- Policy Improvement
  - Since Bellman equations did *not* hold, some condition was violated
  - $\pi(s) \neq \underset{a \in A}{\operatorname{argmax}} \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi}(s') \right]$
  - Improve π by setting this to be true
- Theorem: Policy iteration converges to the optimal policy in a finite number of steps

Extensions (Optional)

- Scaling up: Function Approximation & Fitted Value Iteration
- Online vs. Offline planning
  - Which do I really need, π^* or $\pi^*(s_0)$?
- What happens if my model of the environment is inaccurate?
- Options (macro-actions)

Value Iteration

Value iteration example

Problem definition: numbers are rewards/penalties **per timestep**

START 0	-\$100	-\$100	-\$100	-\$100	-\$100	-\$100	+\$700

$p = 0.1, \gamma = 0.9$

Step 0: Initialization, $V_0 = 0$

Current value function, written as an array.

0	0	0	0	0	0	0	0

Recall
The objective is to find a good estimate of V^*, by repeatedly refining our estimate.

Problem definition: numbers are rewards/penalties **per timestep**

START 0	-$100	-$100	-$100	-$100	-$100	-$100	+$700

$p = 0.1, \gamma = 0.9$

Iteration 1: Perform Bellman Backups

For each state, compute expected cumulative rewards for each action

Lets consider the second last square.

Problem definition: numbers are rewards/penalties **per timestep**

START 0	-$100	-$100	-$100	-$100	-$100	-$100	+$700

$p = 0.1, \gamma = 0.9$

Currently updating value of this state

Iteration 1:

Current value function from previous iteration

0	0	0	0	0	0	0	0

Possible action 1: RIGHT

Immediate reward = -100 (state we are in)

Future reward = 0.9 * 0 (if we move end up right)

+ 0.1 * 0 (if we move end up left)

=> Expected total reward = -100 + 0 = -100

Why 0? Because we use the values from the previous iteration for all estimates of future rewards.

The probabilities 0.9 and 0.1 are obtained from the transition function $P(s'|s,a)$.

Problem definition: numbers are rewards/penalties **per timestep**

START 0	-$100	-$100	-$100	-$100	-$100	-$100	+$700

$p = 0.1, \gamma = 0.9$

Iteration 1:

Currently updating value of this state

Current value function from previous iteration

0	0	0	0	0	0	0	0

Updated value function

TBD	TBD	TBD	TBD	TBD	TBD	-100	TBD

Applying this to all other actions gives:

Possible action 1: RIGHT
Expected total reward = -100 + 0 = -100

Possible action 1: LEFT
Expected total reward = -100 + 0 = -100

Possible action 1: NONE
Expected total reward = -100

=>Value of best action (tied) = -100

Problem definition: numbers are rewards/penalties **per timestep**

START 0	-\$100	-\$100	-\$100	-\$100	-\$100	-\$100	+\$700

$p = 0.1, \gamma = 0.9$

Iteration 1:

Applying this to all states gives:

Current value function from previous iteration

0	0	0	0	0	0	0	0

Updated value function

0	-100	-100	-100	-100	-100	-100	+700

End of iteration 1

Use for iteration 2

Problem definition: numbers are rewards/penalties **per timestep**

START 0	-$100	-$100	-$100	-$100	-$100	-$100	+$700

$p = 0.1, \gamma = 0.9$

Currently updating value of this state

Iteration 2:

Current value function from previous iteration

0	-100	-100	-100	-100	-100	-100	+700

Possible action 1: RIGHT

Immediate reward = -100 (state we are in)
Future reward = 0.9 * 700 (if we end up right)
+ 0.1 * -100 (if we end up left)
=> Expected total reward = -100 + γ620 = 458
We use the values from the previous iteration for all estimates of future rewards. Do not forget discounting!

The probabilities 0.9 and 0.1 are obtained from the transition function $P(s'|s, a)$.

Problem definition: numbers are rewards/penalties **per timestep**

START 0	-$100	-$100	-$100	-$100	-$100	-$100	+$700

$p = 0.1, \gamma = 0.9$

Currently updating value of this state

Iteration 2

Current value function from previous iteration

0	-100	-100	-100	-100	-100	-100	+700

Updated value function

TBD	TBD	TBD	TBD	TBD	TBD	458	TBD

Applying this to all other actions gives:

Possible action 1: RIGHT
Expected total reward = 458

Possible action 1: LEFT
Expected total reward = -100 + $\gamma(-100)$ = -190

Possible action 1: NONE
Expected total reward = -100 + $\gamma(-100)$ = -190

=>Value of best action (right) = 458

Problem definition: numbers are rewards/penalties **per timestep**

START 0	-\$100	-\$100	-\$100	-\$100	-\$100	-\$100	+\$700

$p = 0.1, \gamma = 0.9$

Iteration 2:

Applying this to all states gives:

Current value function from previous iteration

0	-100	-100	-100	-100	-100	-100	+700

Updated value function

0	-190	-190	-190	-190	-190	+458	+700

End of iteration 2

Use for iteration 3

Repeated iterations converge to the optimal value function

Value function converges to V^*

0	-100	-93.7	18.88	157.2	315.4	495.4	+700

Now, perform **one step lookahead** to extract π^*

Problem definition: numbers are rewards/penalties **per timestep**

START 0	-\$100	-\$100	-\$100	-\$100	-\$100	-\$100	+\$700

$p = 0.1, \gamma = 0.9$

Suppose we want the best action at this state

V^* we computed after many iterations

0	-100	-93.7	18.88	157.2	315.4	495.4	+700

Possible action 1: RIGHT

Immediate reward = -100 (state we are in)

Future reward = 0.9 * 157.2 (if we end up right)
+ 0.1 * -93.7 (if we end up left)

=> Expected total reward = -100 + γ620 = 18.88

We use the values from V^* for future rewards.

Do not forget discounting!

Perform for all actions to obtain the best

End of Algorithm

Final policy after extracting best action at each state

---	←	→	→	→	→	→	---