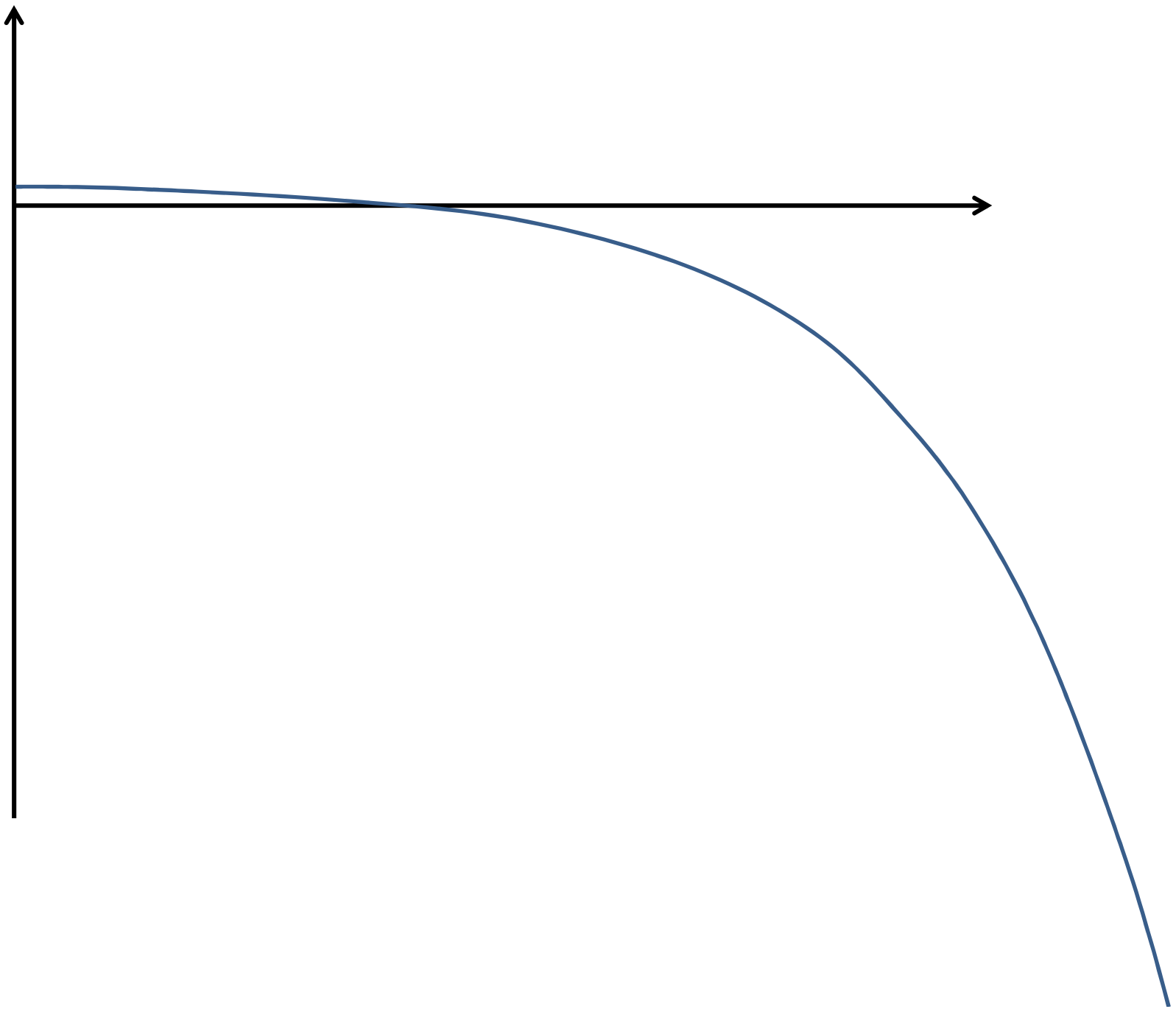
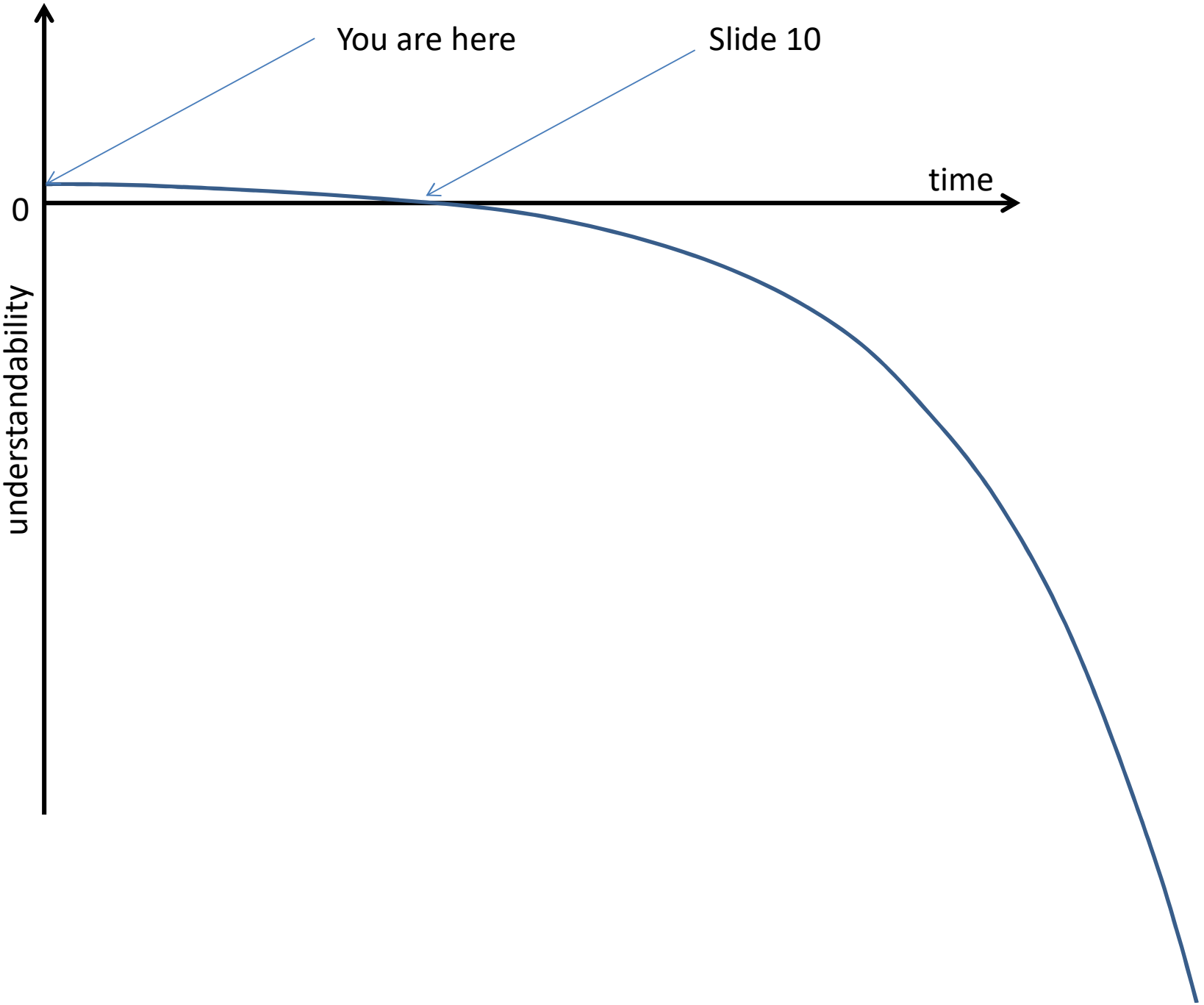
Reinforcement Learning

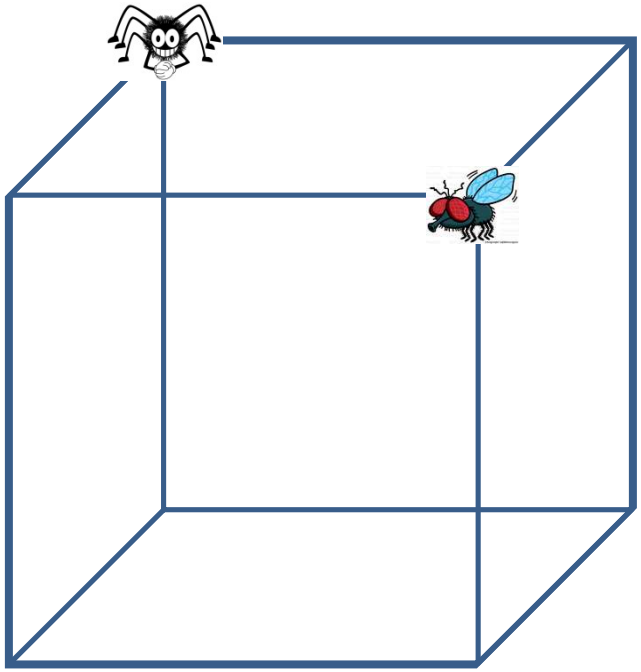
11-785, Fall 2019

Defining MDPs, Planning



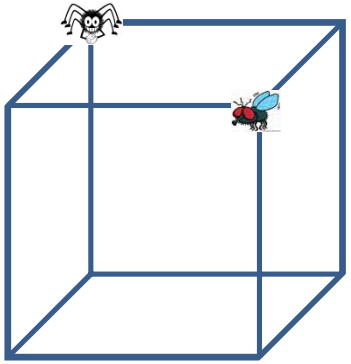

You are here
Slide 10
time
0
understandability

The story of Flider and Spy



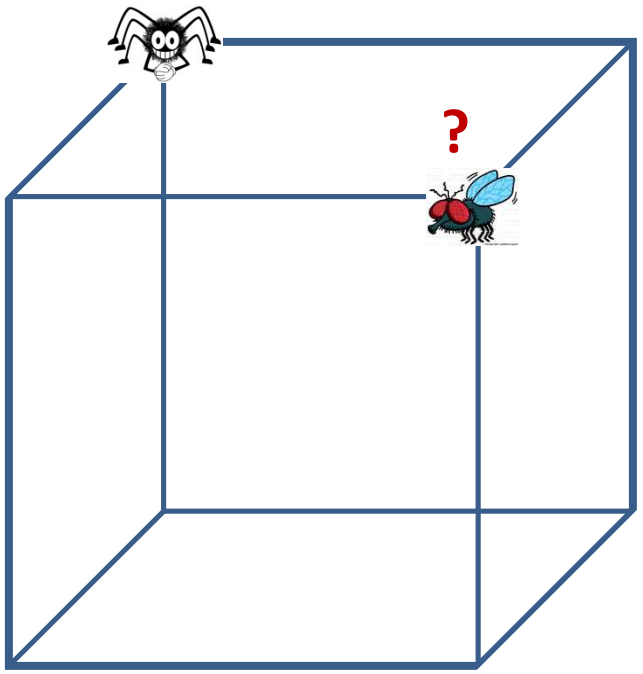
- Flider the spider is at the far corner of the room, and Spy the fly is sleeping happily at the near corner

The story of Flider and Spy



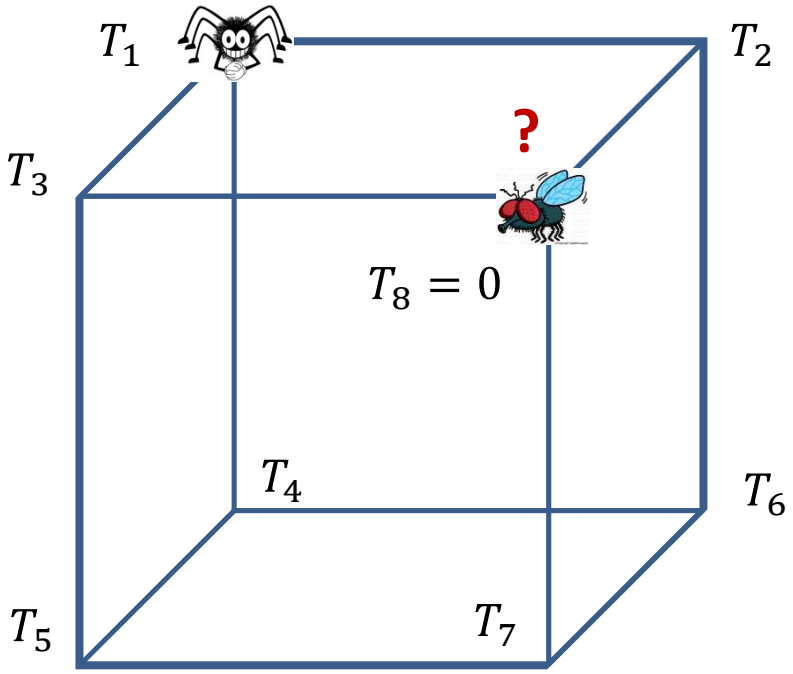
- Flider only walks along edges
- She begins walking along one of the three edges at random
- She takes one minute to cover the distance from one corner to the other along any edge
- When she arrives at the new corner, she randomly chooses one of the three edges and continues walking (she may even turn back)

The story of Flider and Spy



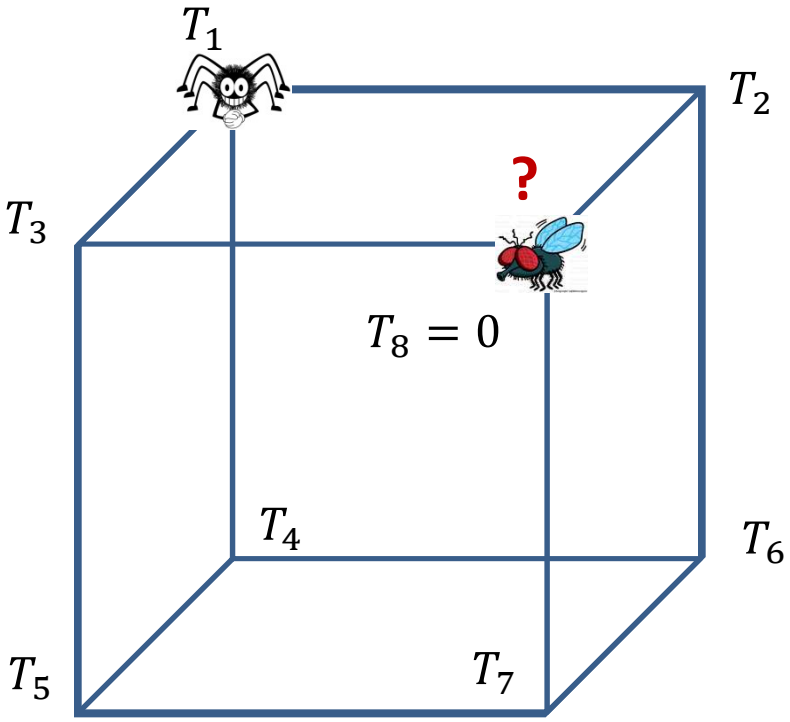
- What is the life expectancy of Spy?

Flider and Spy



- Let T_i be the life expectancy if Flider is at the ith corner

Flider and Spy



$$T_1 = \frac{1}{3}(1 + T_2) + \frac{1}{3}(1 + T_3) + \frac{1}{3}(1 + T_4)$$

$$T_2 = \frac{1}{3}(1 + T_1) + \frac{1}{3}(1 + T_6) + \frac{1}{3}(1 + T_8)$$

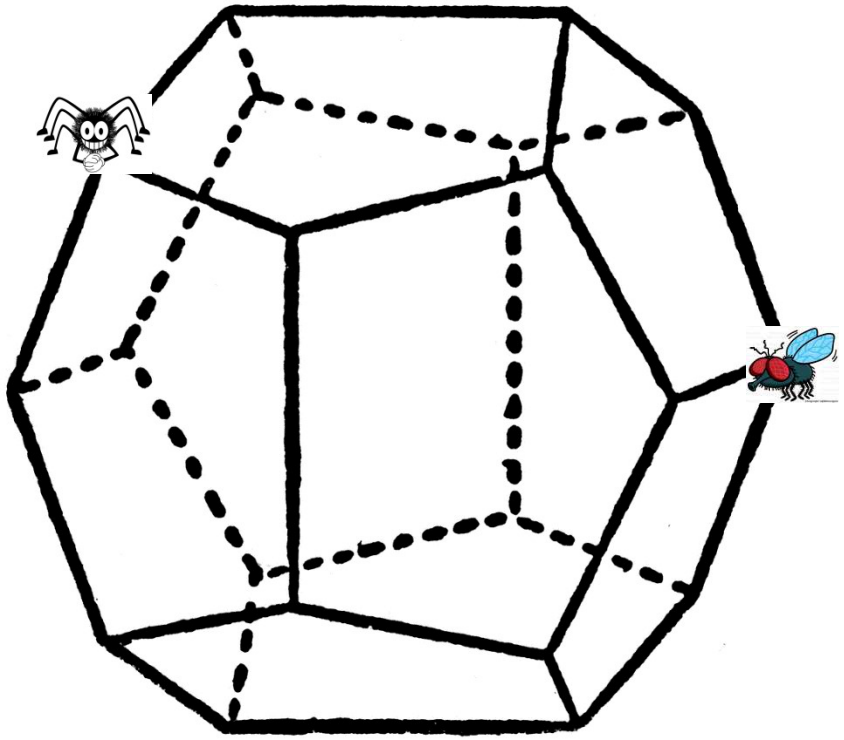
$$\vdots$$

$$T_7 = \frac{1}{3}(1 + T_8) + \frac{1}{3}(1 + T_6) + \frac{1}{3}(1 + T_5)$$

$$T_8 = 0$$

- $1 + T_i$ is the life expectancy if Flider the Spider begins walking towards the ith corner
  - 1 minute to get to the corner plus the time taken to get from that corner to Spy the fly
- 8 Equations, 8 unknowns, trivial to solve

Flider and Spy



- What if the room has many corners?
  - A Fuller's dome?

SWITCHING GEARS ABIT...

Alpha Go

Deep Blue: 1997

Alpha Go: 2016

- "Learning" to play
  - How?

Problems?

- How do we write a program to beat Magnus Carlsson?

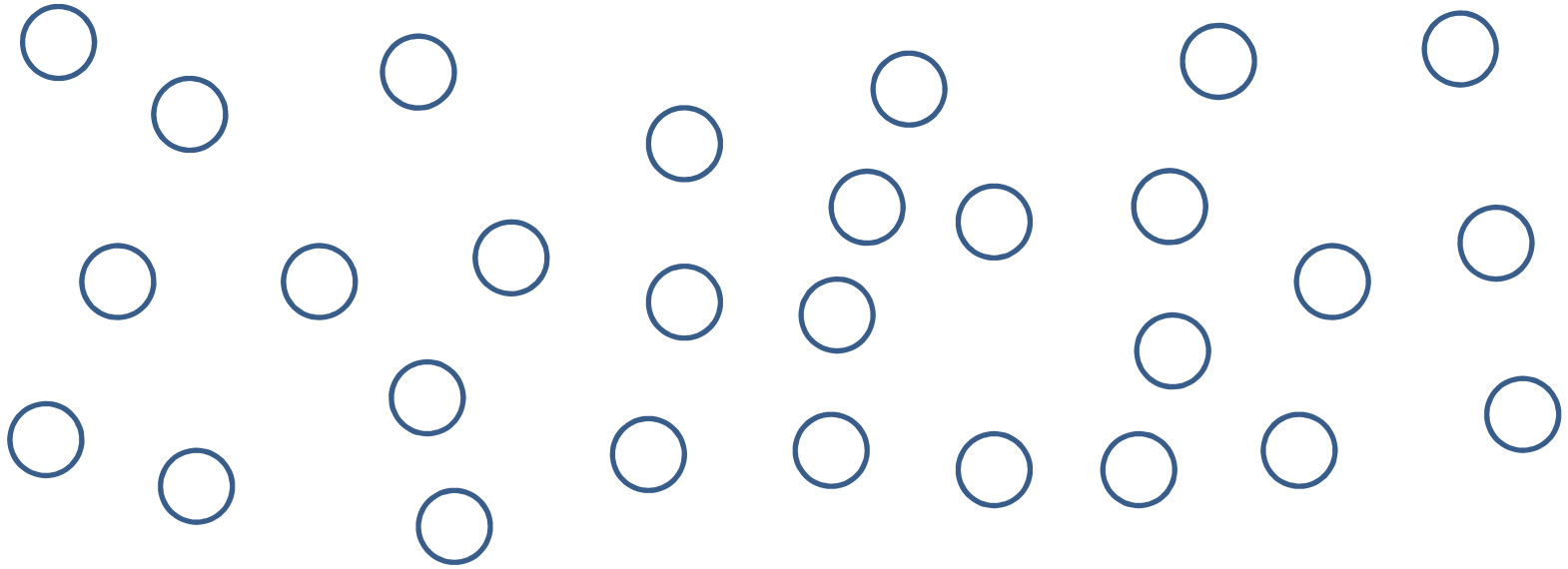
- Can't just make a table of rules
  - Too many positions, too many combinations
- How about some general principles?
  - Who will enumerate them?
  - How many are there in the first place?

- How do humans do it?

Learning from experience

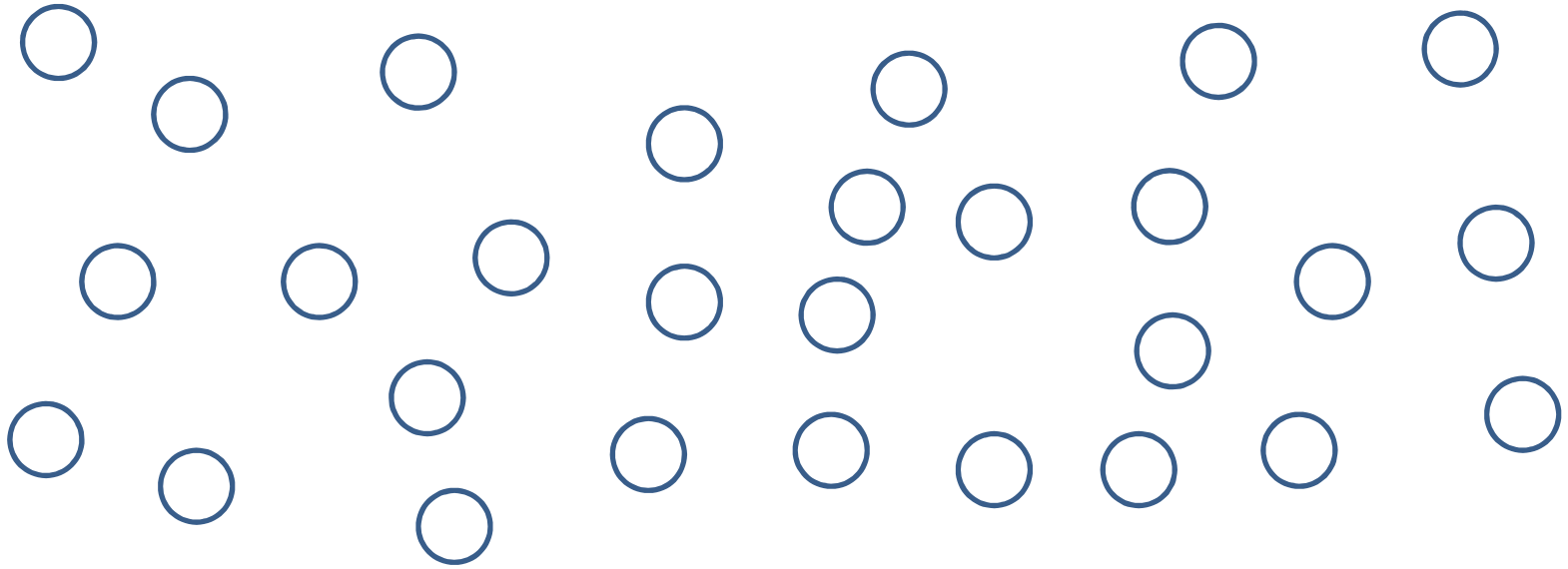
- Learn by playing (or observing)
  - Problem: The tree of possible moves is exponentially large
- Learn to generalize
  - What do we mean by "generalize"?
  - If a particular board position always leads to loss, avoid any moves that move you into that position

Lets draw a diagram..



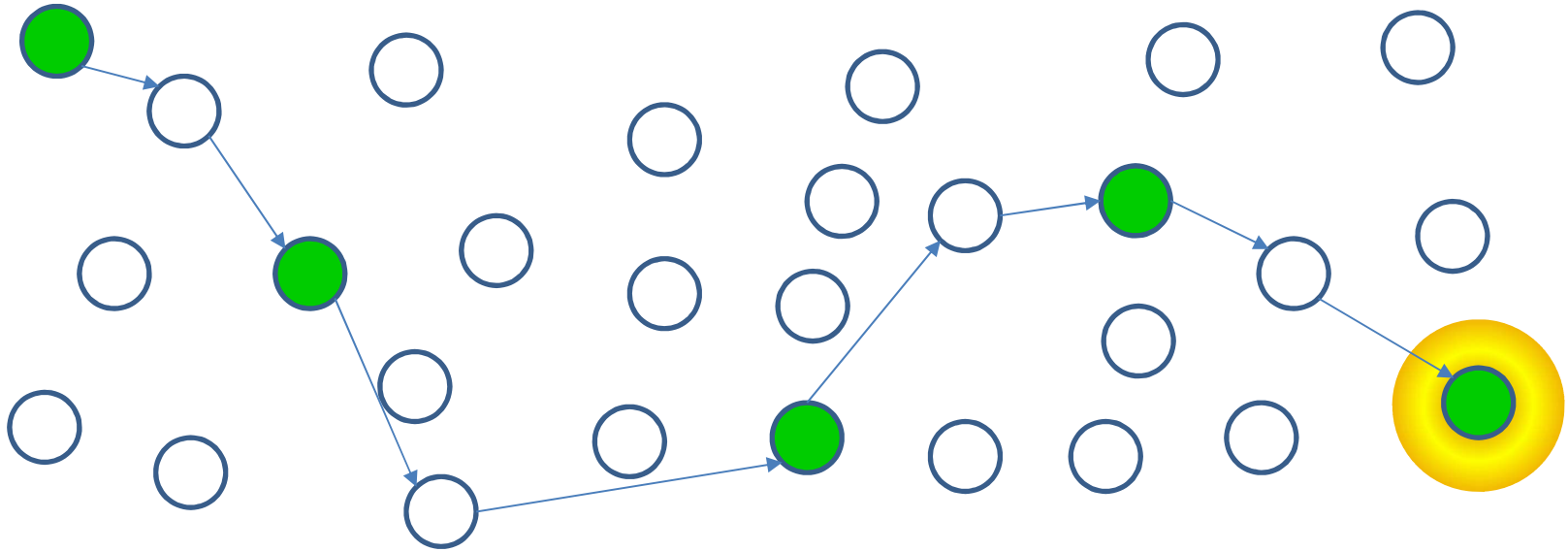
- Circles are game states
  - Exponentially large number of them
  - In the beginning we don't know if they are good or bad
- Each state can move into one of N states depending on the opponent's move
  - Figure does not show arrows

Lets draw a diagram..



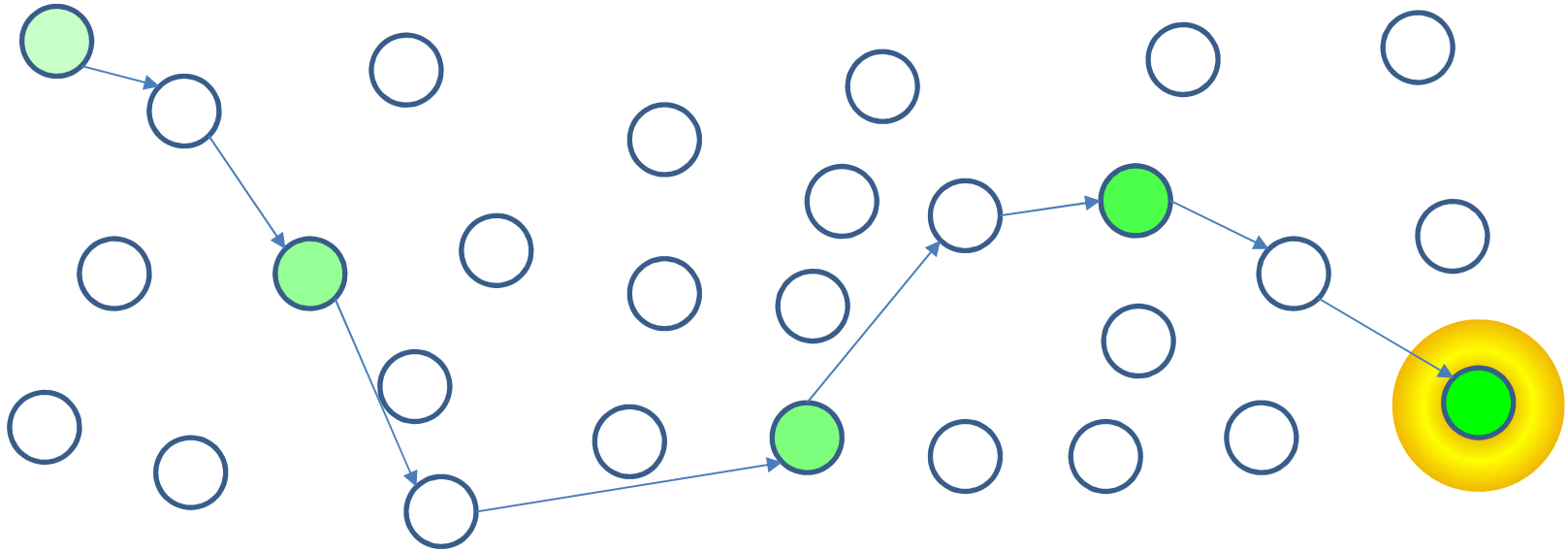
- Play a very large number of games
  - Each time a board position leads to victory, give it a little green color
  - Each time it leads to a loss give it a little red

A game we won



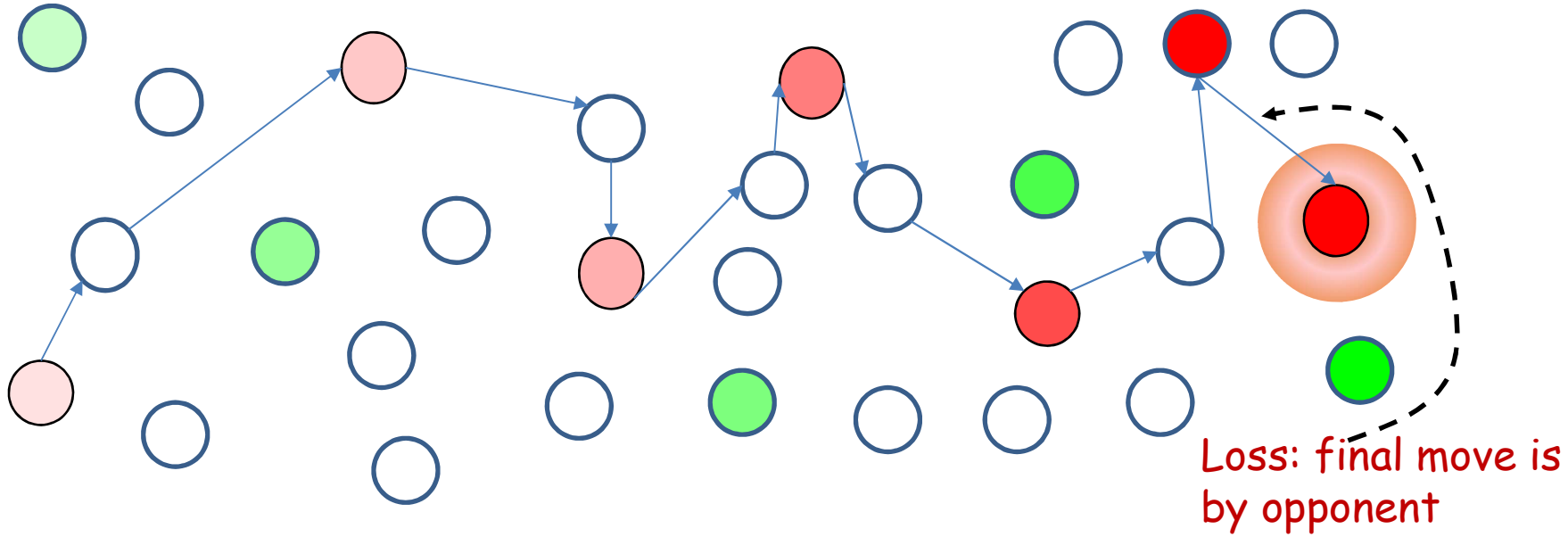
- Sequence of game states we moved into until the winning state
  - Alternates with states arrived at by moves by the opponent
- All of these are “winning” states: color them green

A game we won



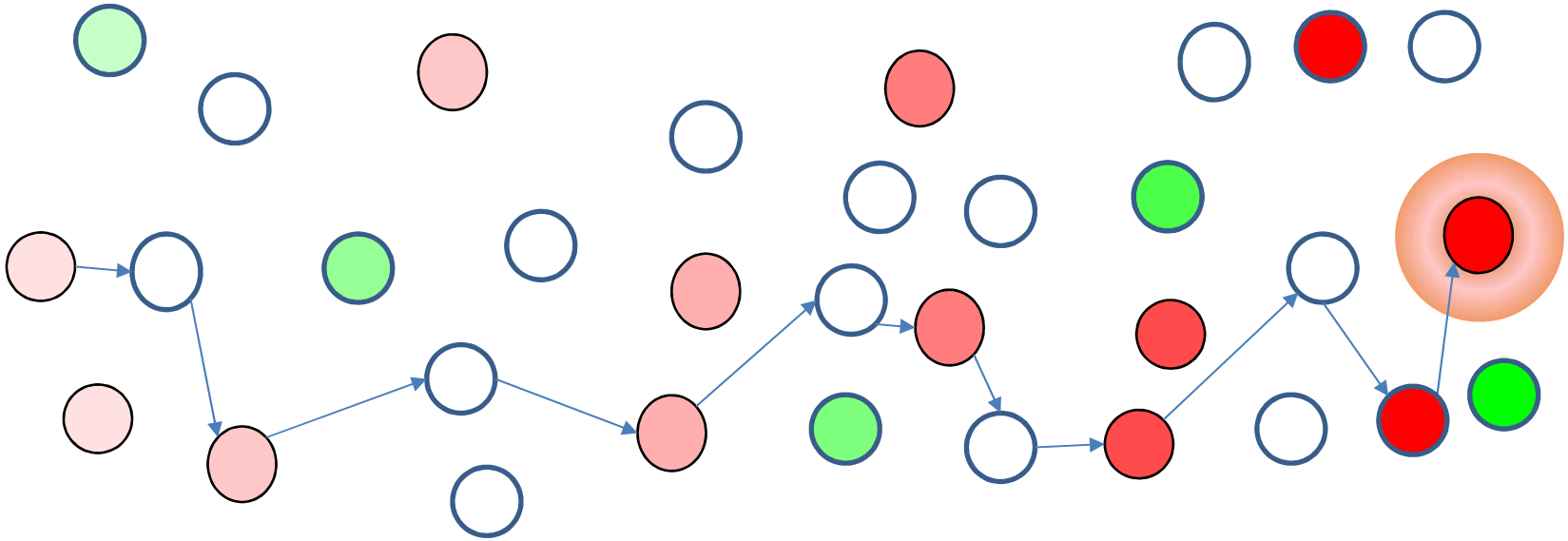
- Sequence of game states we moved into until the winning state
  - Alternates with states arrived at by moves by the opponent
- All of these are “winning” states: color them green
- But things from the distant past are less certain
  - Too many possibilities; cant be certain of their “winningness”
  - Fade the green with distance

A game we lost



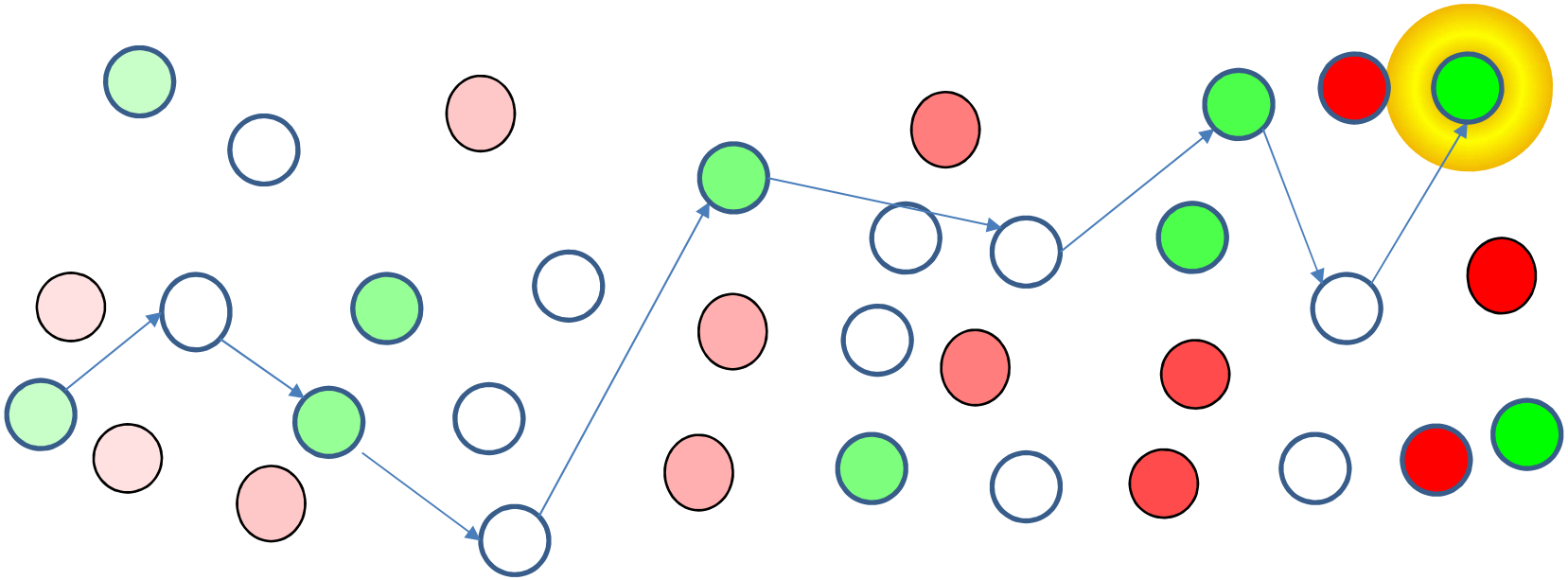
- Sequence of game states we moved into until the losing state
  - Alternates with states arrived at by moves by the opponent
- All of these are “losing” states: color them red
- But things from the distant past are less certain
  - Too many possibilities; cant be certain of their “losingness”
  - Fade the red with distance

Continue playing games



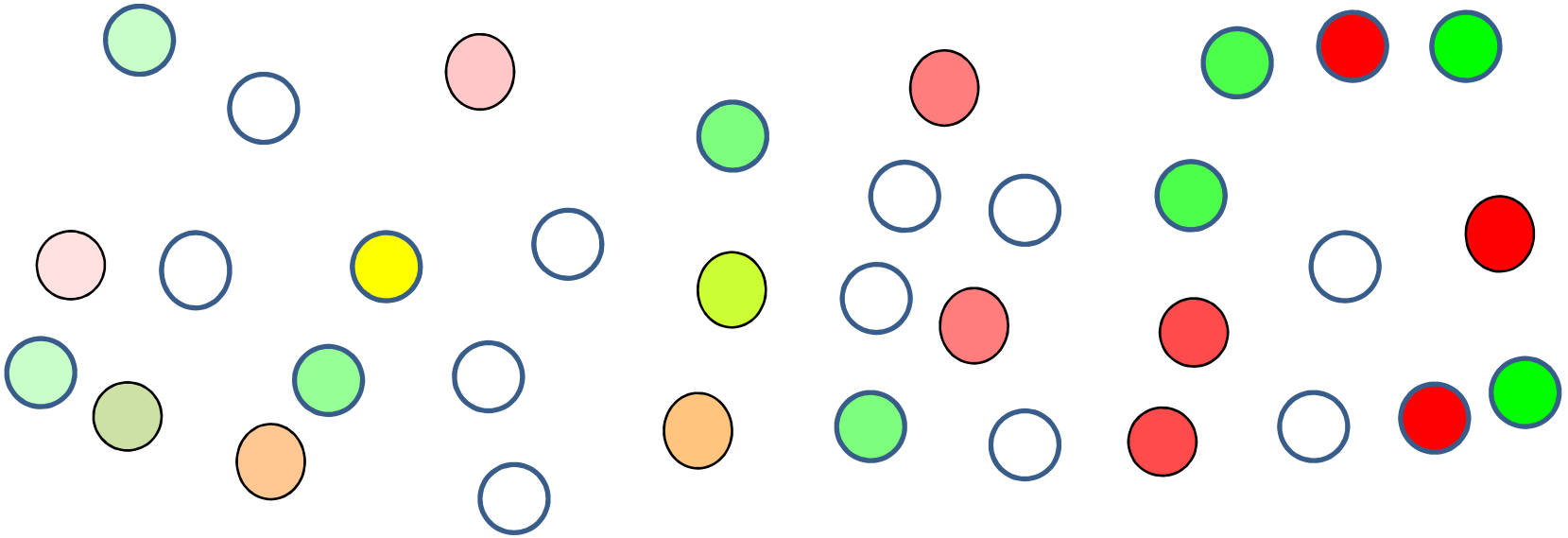
- Play many many games
- Some of which you will lose..

Continue playing games



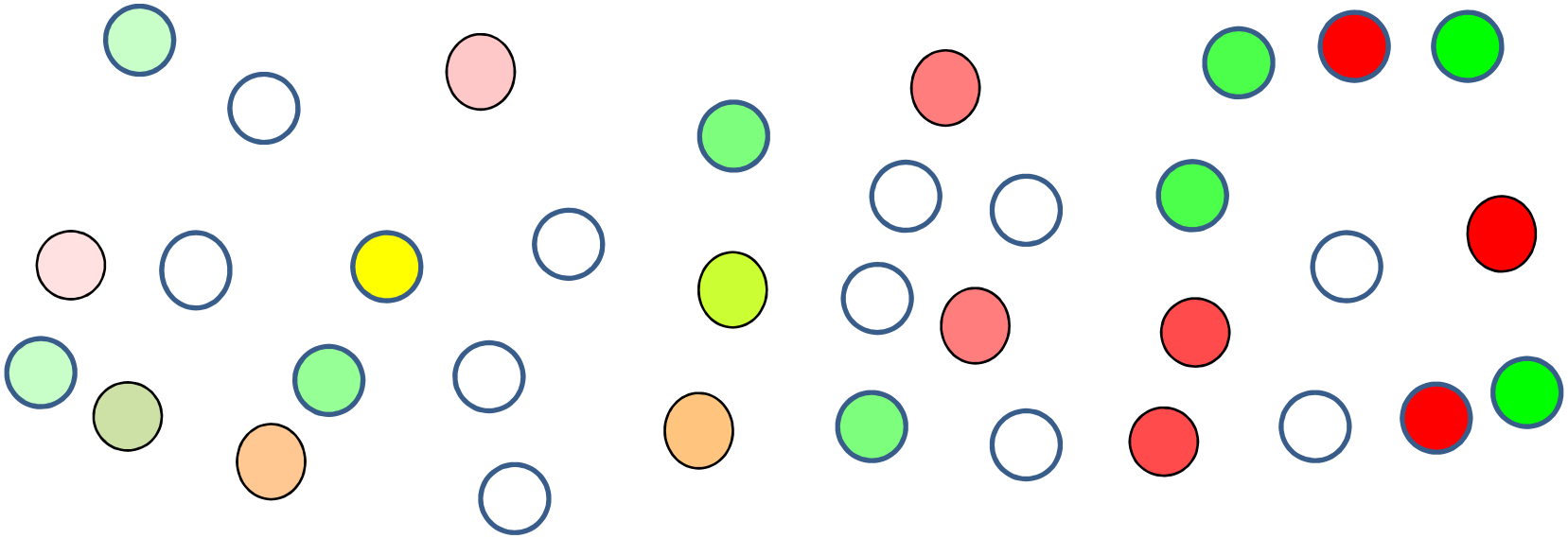
- Play many many games
- Some of which you will lose..
- And some you'll win..

Continue playing games



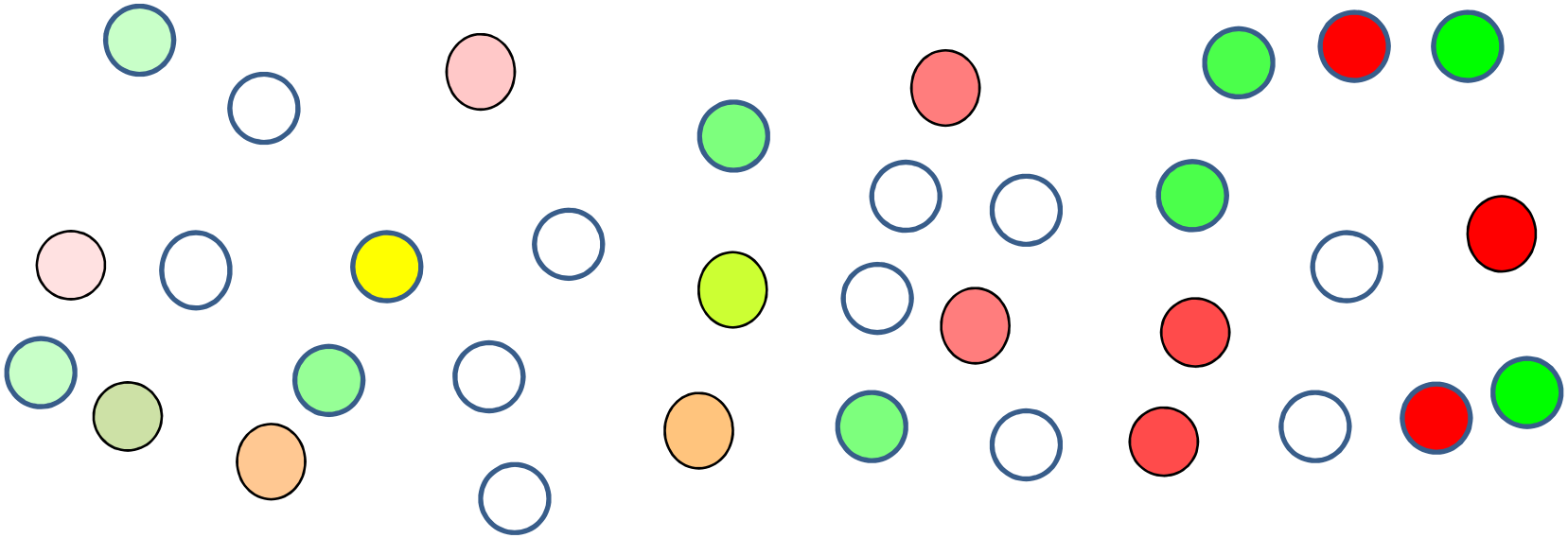
- When multiple games visit a state, simply average the colors derived from all visits
  - Some states will get greener
  - Some will get redder
  - Some, that can lead to both victory and loss will become different shades of yellow..
    - More in the early stages of the game than during the endgame

Collecting more games…



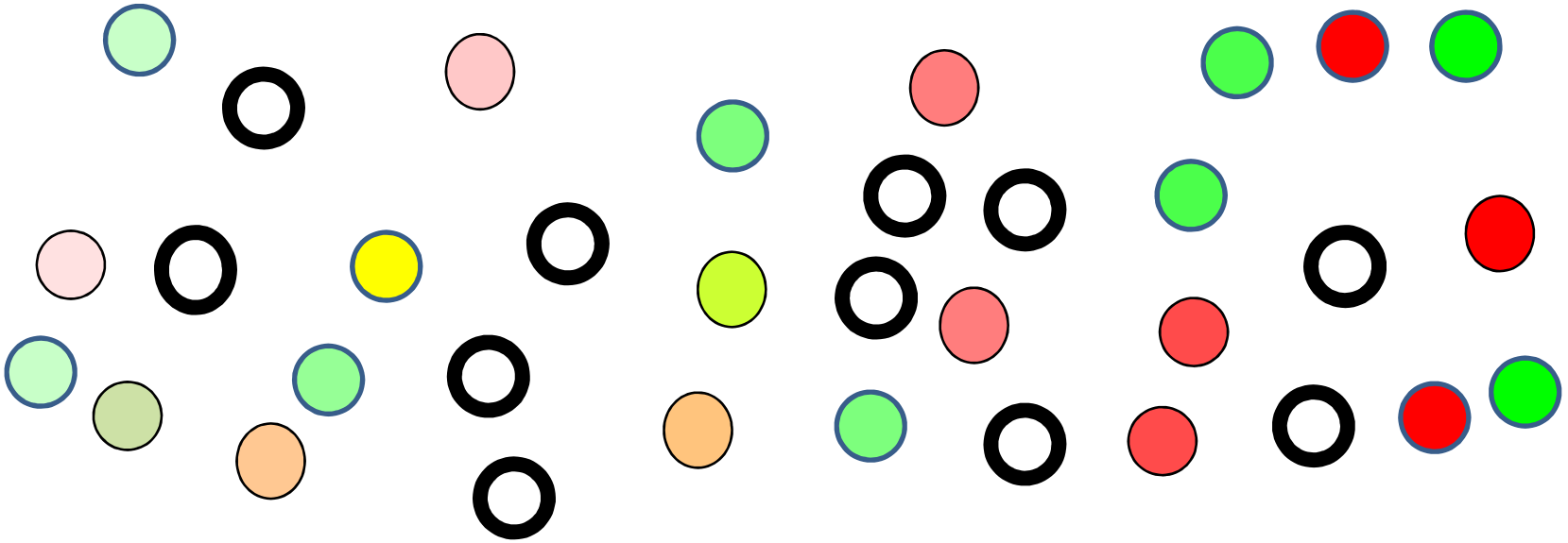
- You can also learn colors from your *opponent's moves*
  - When you win he/she loses and vice versa
- You can learn from *others'* games
  - Collections of games by amateurs and experts, of which you can find millions in the books
- To *really* speed up matters, play with yourself
  - A schizophrenic computer can play thousands of games with itself in the time that it plays with another person

Lets draw a diagram..



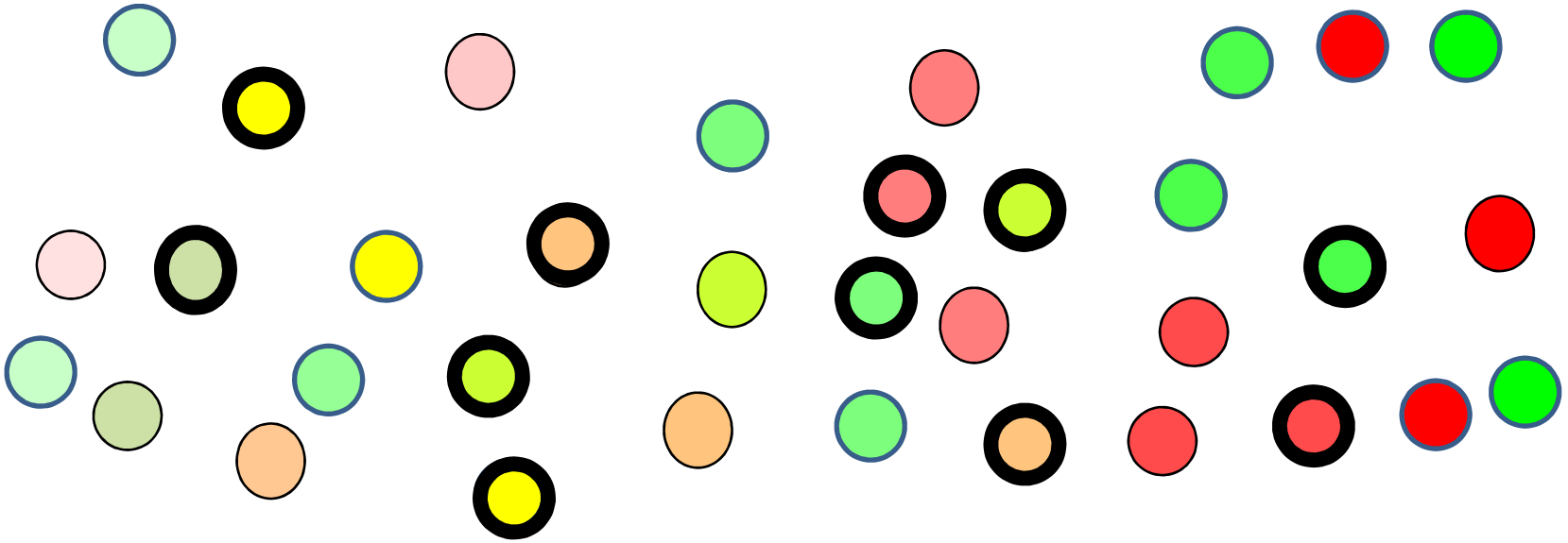
- Eventually, we'll get many board positions with different shades of green (more winning than losing), red (more losing than winning) or various shades of yellow/green/orange (can go either way)

Lets draw a diagram..



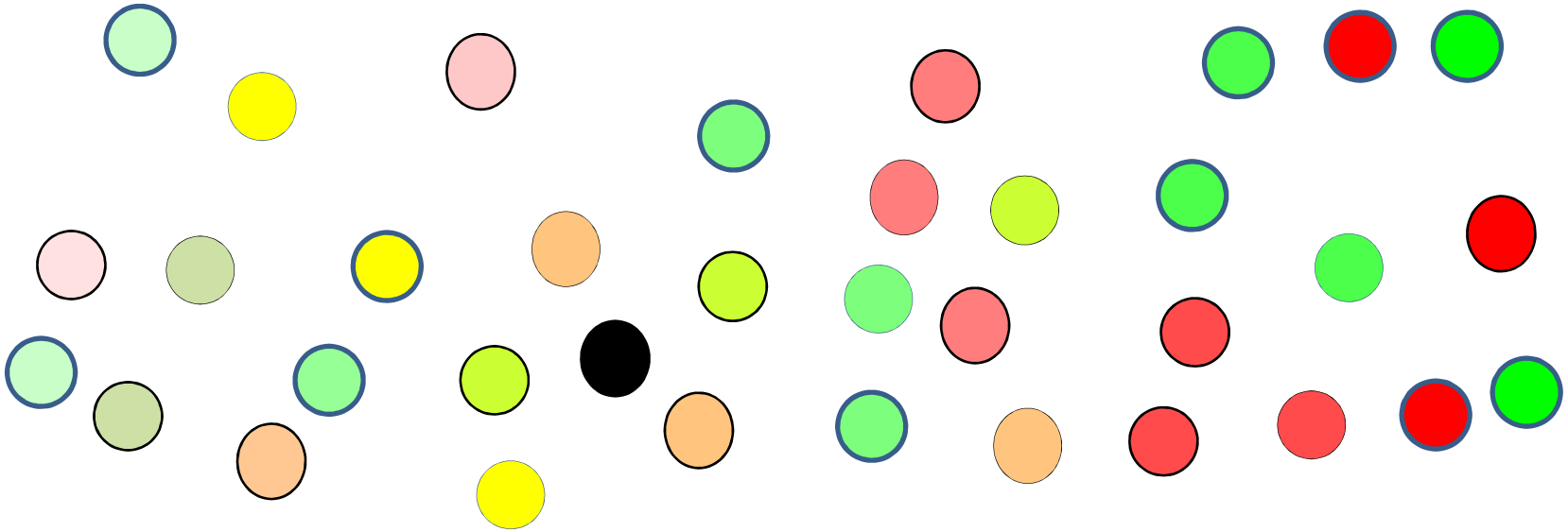
- Eventually, we'll get many board positions with different shades of green (more winning than losing), red (more losing than winning) or various shades of yellow/green/orange (can go either way)
- We will also get many "blank" positions that were never visited in all our practice games
  - In fact the *vast majority* of positions will be unvisited!

Lets draw a diagram..



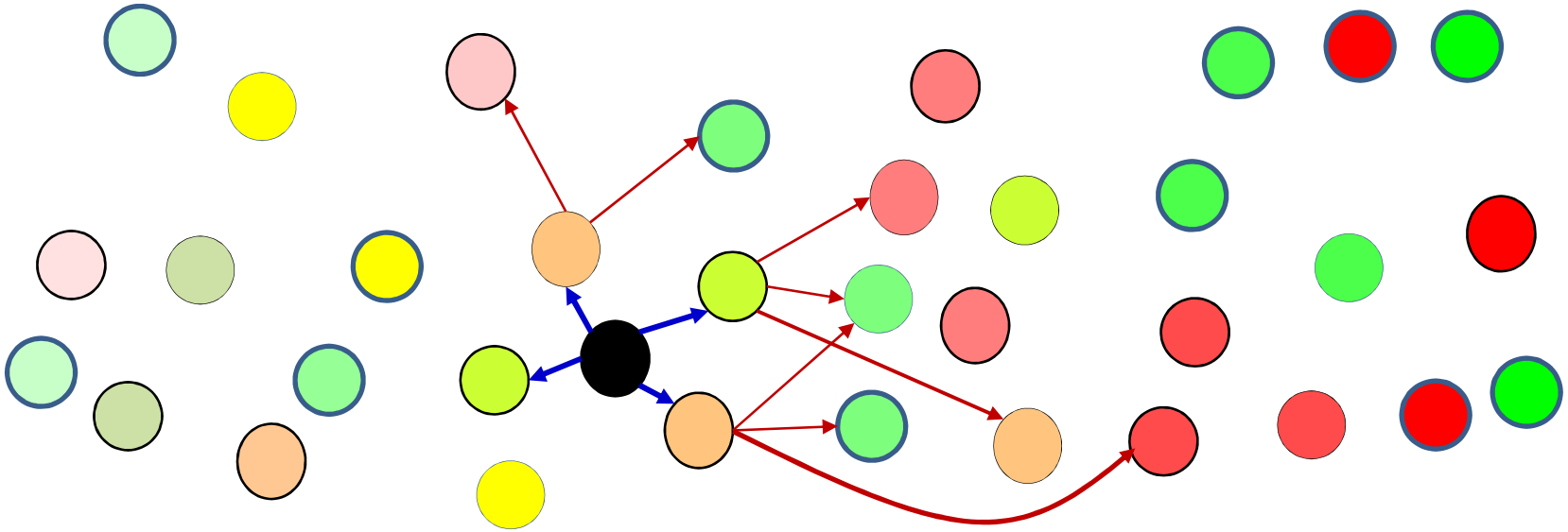
- Generalization: From the coloured nodes, learn some way of colouring the blank nodes too
  - Which will have some colour between red and green
    - Different nodes will have different colours
- **The magic: some *function* that assigns color to different board positions**
  - How do you describe a board position numerically
  - What type of function maps a board position to a color between red and green

GAME TIME



- Black circle: Current position
- Where do we move?

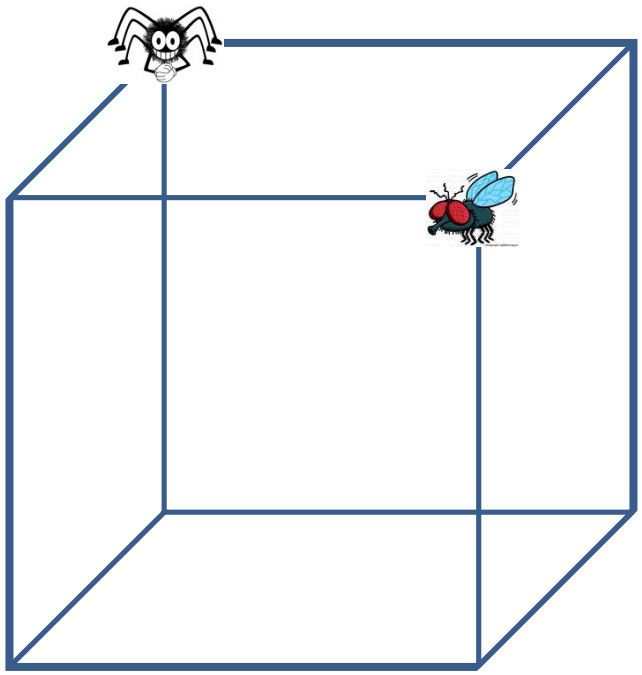
GAME TIME

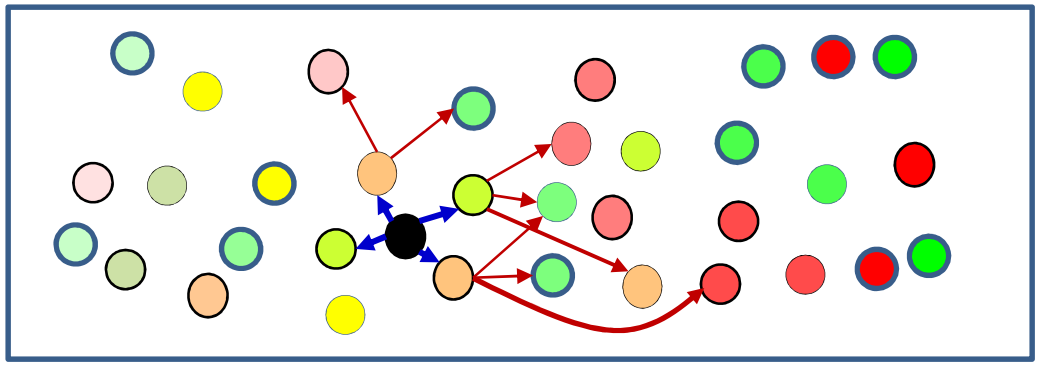


- Evaluate all our possible moves
  - And all of opponents responses
  - Can evaluate the graph to any depth (compute and memory bound)

- Identify the move that gives the opponent the least chance to win
  - The opponent's best path leads to a least red state

A little terminology





- Markov Process: Does not matter how you got here, only matters where you are

An interesting class of problems

- Is a move good?
  - You will not know until the end of the game

An interesting class of problems

- Is an investment plan good?
  - You will not know for a while

An interesting class of problems

- Do I
  - Change lane left?
  - Change lane right?
  - Accelerate?
  - Decelerate?

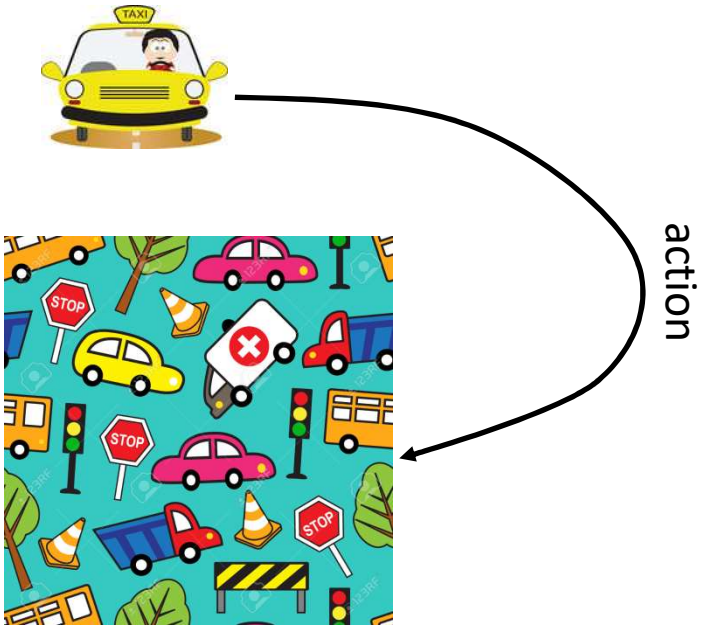
Reward-based problems

- And many others
- Common theme: These are control problems where
  - Your actions beget rewards
    - Win the game
    - Make money
    - Get home sooner
  - But not deterministically
    - A world out there that is not predictable
- From experience of *belated* rewards, you must learn to act rationally

General cartoon of the world

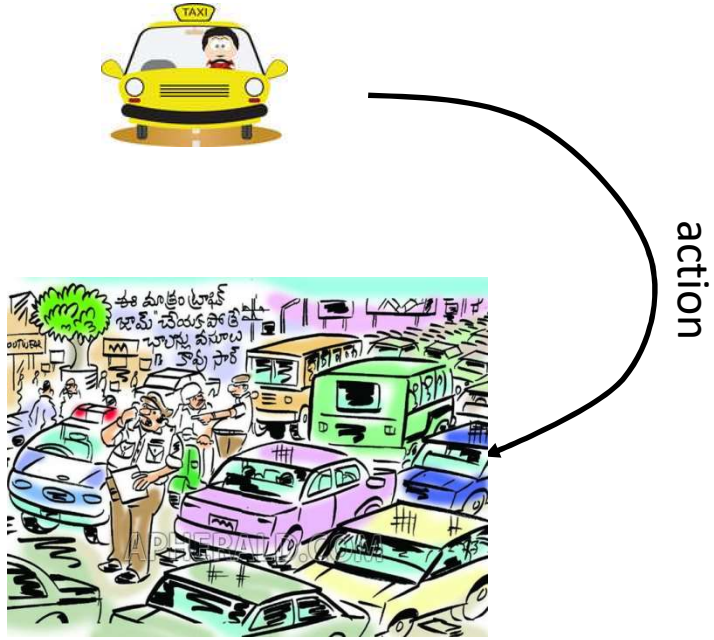
- Agent operates in an environment
  - Agent may be you..
  - Environment is the game, the market, the road..

General cartoon of the world



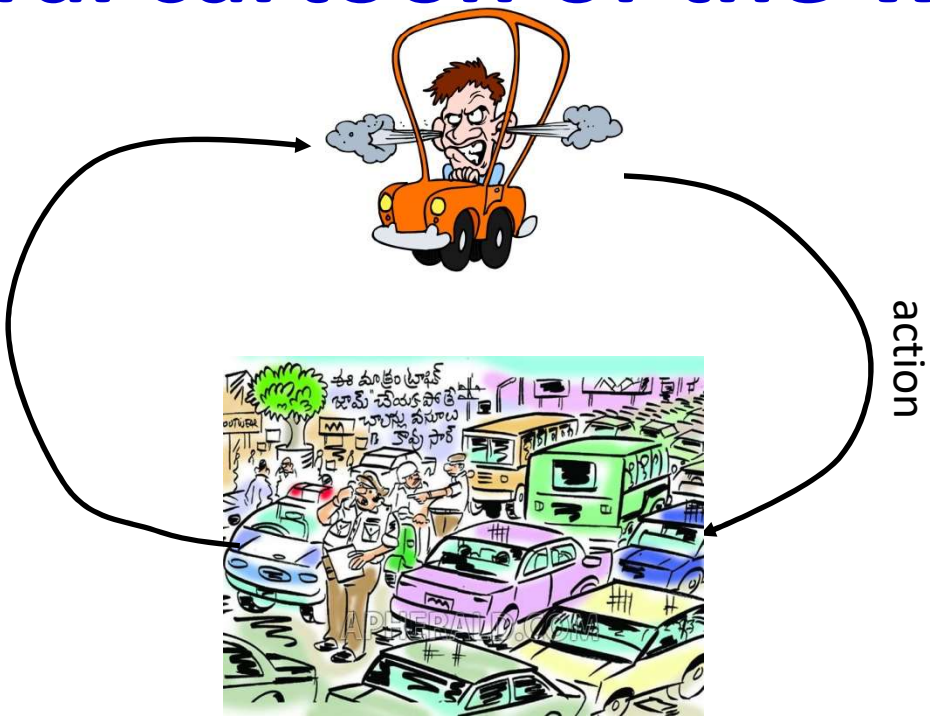
- Agent takes actions which affect the environment

General cartoon of the world



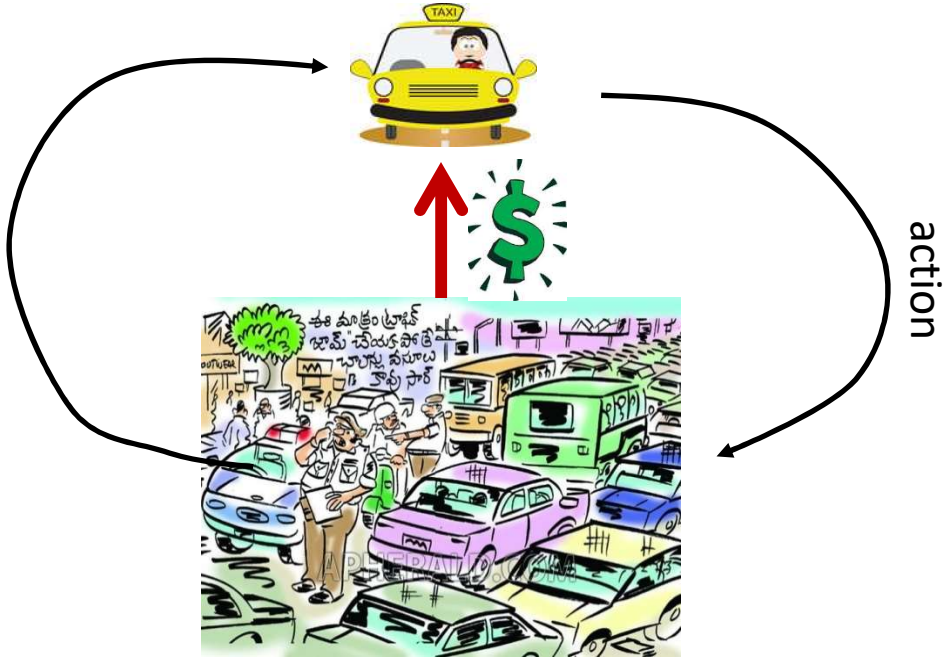
- Agent takes actions which affect the environment
- Which changes in a somewhat unpredictable way

General cartoon of the world



- Agent takes actions which affect the environment
- Which changes in a somewhat unpredictable way
- Which affects the agent's situation

General cartoon of the world



- The agent also receives rewards..
  - Which may be apparent immediately
  - Or not apparent for a very long time

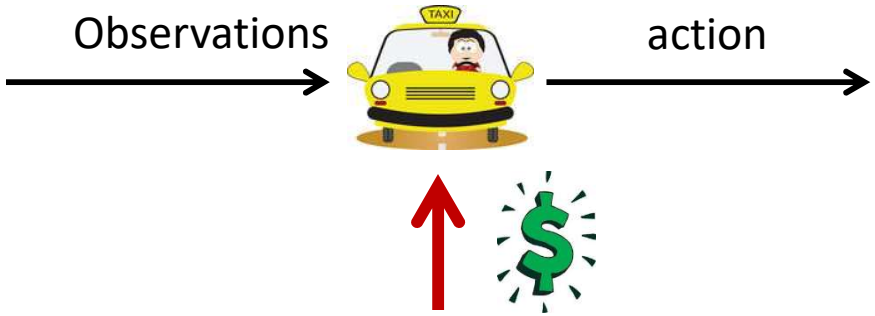
Challenge

- How must the agent behave to maximize its rewards

What the environment "experiences"

- Responds to some action by the agent
  - Changes in response
- Returns some reward (or punishment) to agent

What the Agent sees

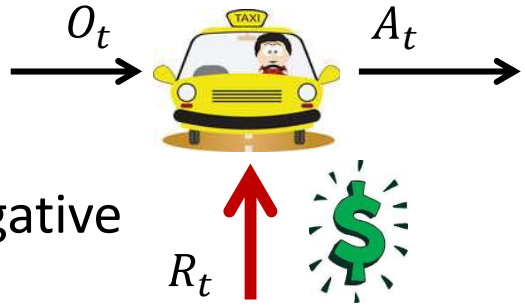


- The agent may *observe* something about its environment
  - Sensor readings, images of a game board, stock indices..
- The agent takes actions
- The agent receives rewards

- This is the agent's world; it must make sense of it
- Again: Agent's objective → to take the actions that maximize rewards

Lets formalize the problem

- These can be cast as problems of *reinforcement* learning
- There is no supervisor, only a *reward* signal
  - Did you get home sooner
  - Did you win the game
  - Did you make money?
- i.e. nobody telling the agent "you did well"



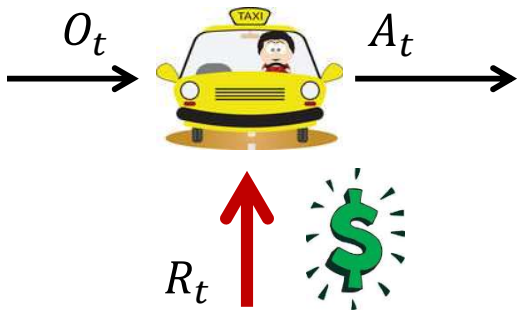
- *Reward is a scalar –* a single number, may be negative
  - Game was won/lost (binary)
  - Time taken to arrive
  - Amount of money made
- Reward may be delayed
  - Wait till the end of the game!
- Agents actions affect its current and future rewards
  - Must optimize actions for maximum reward

To Maximize Reward

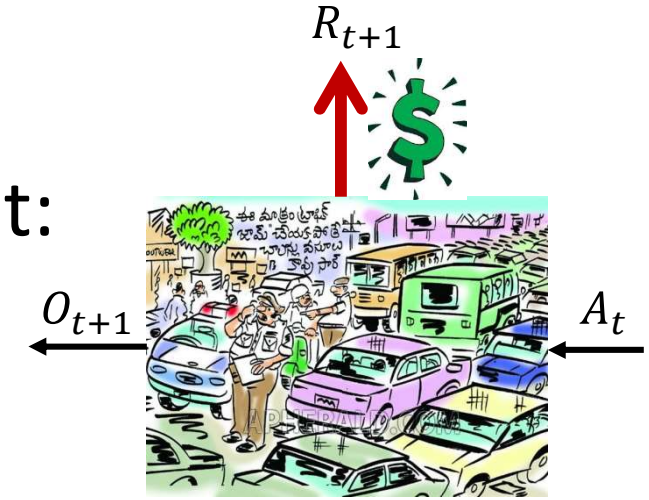
- We can represent the environment as a *process*
  - A mathematical characterization with a *true* value for its parameters representing the *actual* environment
- The agent must *model* this environment process
  - Formulate its own model for the environment, which must ideally match the true values as closely as possible
    - Based only on what it observes
- Agent must formulate winning strategy based on model of environment

Lets formalize the system

- At each time t the agent:
  - Makes an observation O_t of the environment
  - Receives a reward R_t
  - Performs an action A_t

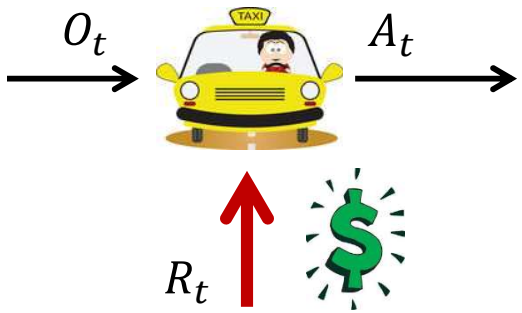


- At each time t the environment:
  - Receives an action A_t
  - Emits a reward R_{t+1}
  - Changes and produces an observation O_{t+1}

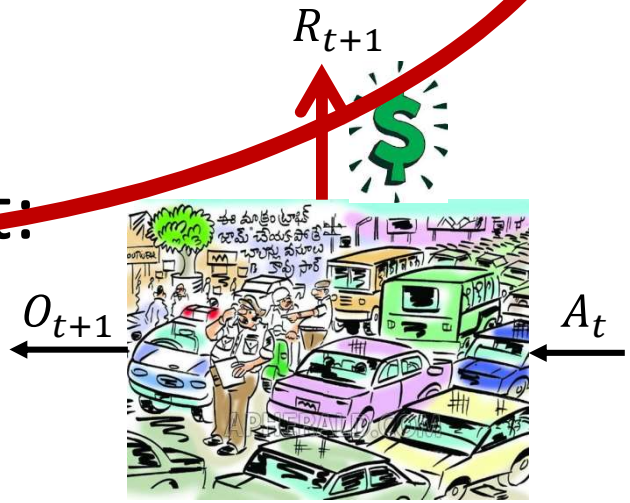


Lets formalize the system

- At each time t the agent:
  - Makes an observation O_t of the environment
  - Receives a reward R_t
  - Performs an action A_t



- At each time t the environment:
  - Receives an action A_t
  - Emits a reward R_{t+1}
  - Changes and produces an observation O_{t+1}



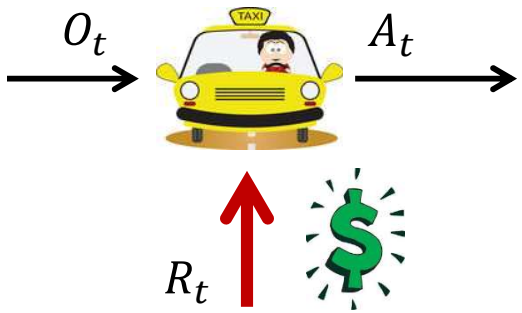
From the perspective of the Agent

- What the agent perceives..
- The following History:
- $H_t = O_0, R_0, A_0, O_1, R_1, A_1, \ldots, O_t, R_t$

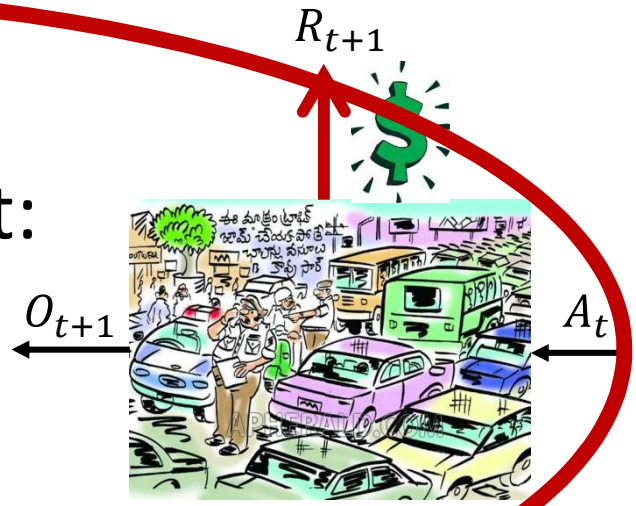
- The total history at any time is the sequence of observations, rewards and actions
- We need to model this sequence such that at any time t, the best $A_t|H_t$ can be chosen
  - The Strategy that maximizes total reward $R_0 + R_1 + \cdots + R_T$

Lets formalize the system

- At each time t the agent:
  - Makes an observation O_t of the environment
  - Receives a reward R_t
  - Performs an action A_t



- At each time t the environment:
  - Receives an action A_t
  - Emits a reward R_{t+1}
  - Changes and produces an observation O_{t+1}



Can define a “state”

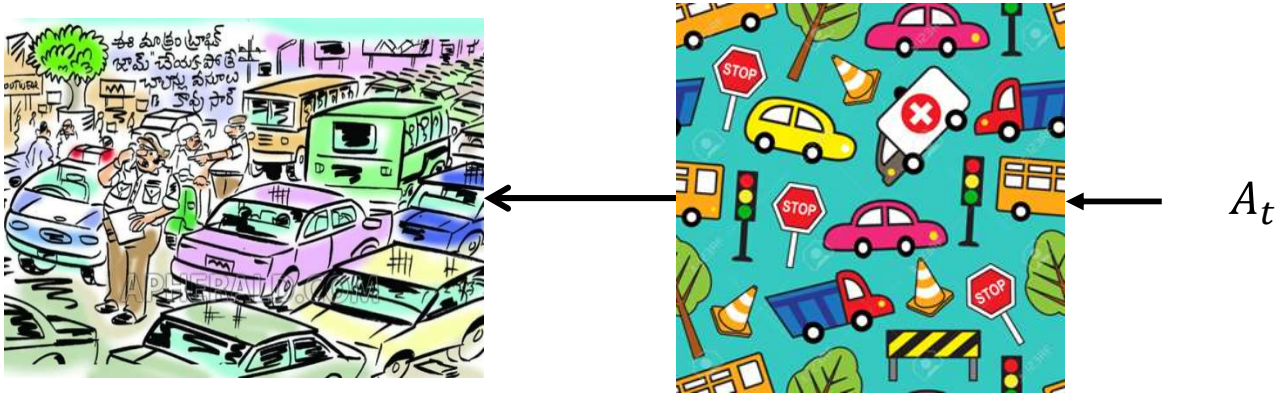
- Fully captures the “status” of the system
  - E.g., in an automobile: [position, velocity, acceleration]
  - In traffic: the position, velocity, acceleration of *every* vehicle on the road
  - In Chess: the state of the board + whose turn it is next

The state of the *environment*

A_t

- The environment's state!
  - This is what will finally decide the rewards
- May be a complex combination of many things
- Generally assumed to be dynamic – keeps changing
- The agent's actions can affect the way in which it responds
  - But agent may not be able to observe all of it

Markov property

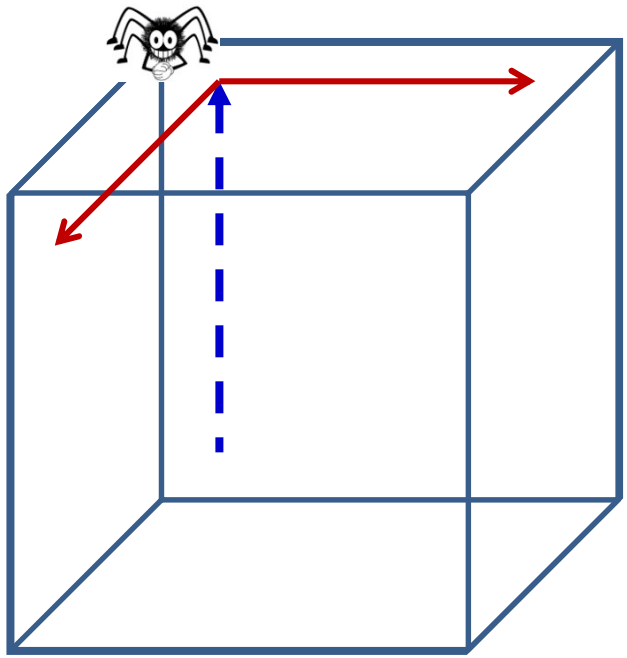


- Markov Property: A well-defined state fully captures all information needed to predict the future
  - No additional information from the past required

$$P(S_{t+1}|S_0, S_1, \dots, S_t) = P(S_{t+1}|S_t)$$

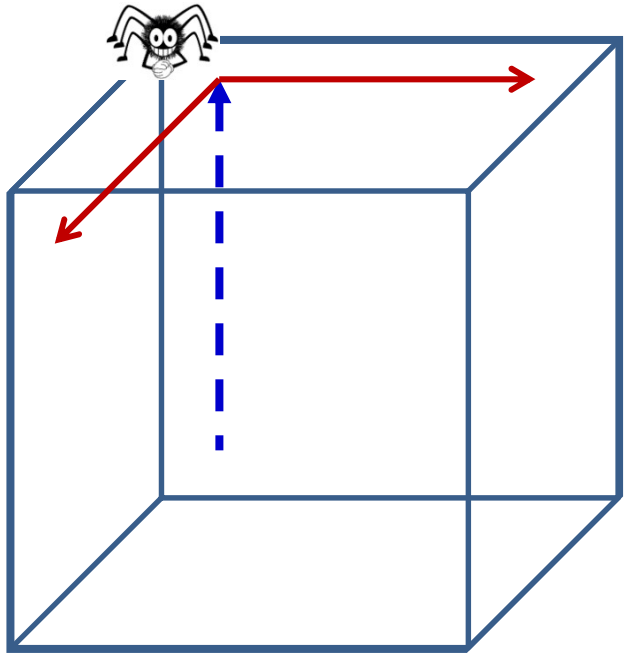
- The environment's future only depends on its present

A brief trip to Nostalgia..



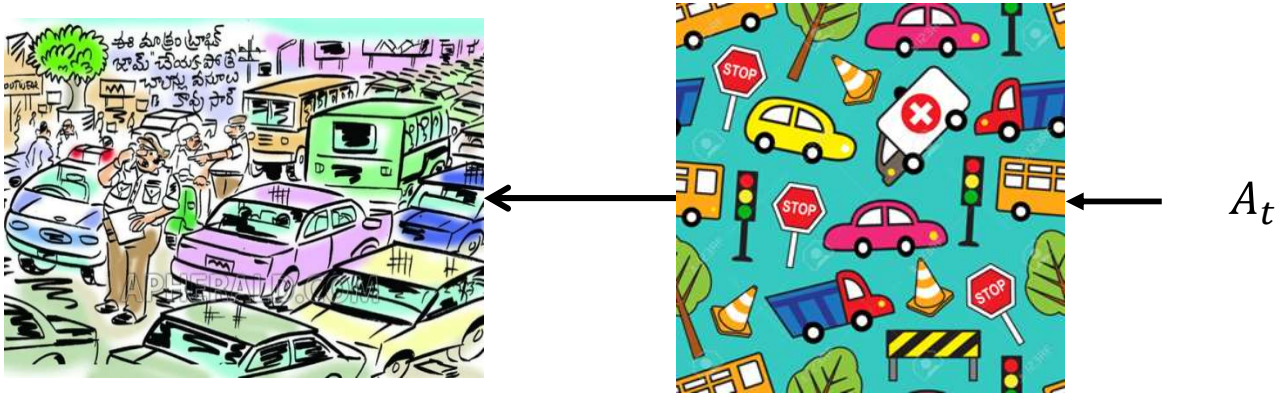
- Glider, Flider's brother, never turns around during his wanderings
  - On arriving at any corner, he chooses one of the two "forward" paths randomly.
    - The future possibilities depend on the edge he arrived from
  - Is he Markovian?

Glider is a Markov dude!



- Any causal system can be viewed as Markov, with appropriately defined state
  - The *Information state* S_t may differ from the *apparent* state s_t
  - Defining $S_t = s_1, s_2, \ldots, s_t$
  - $P(S_{t+1}|S_0, S_1, \ldots, S_t) = P(S_{t+1}|S_t)$

Markov property

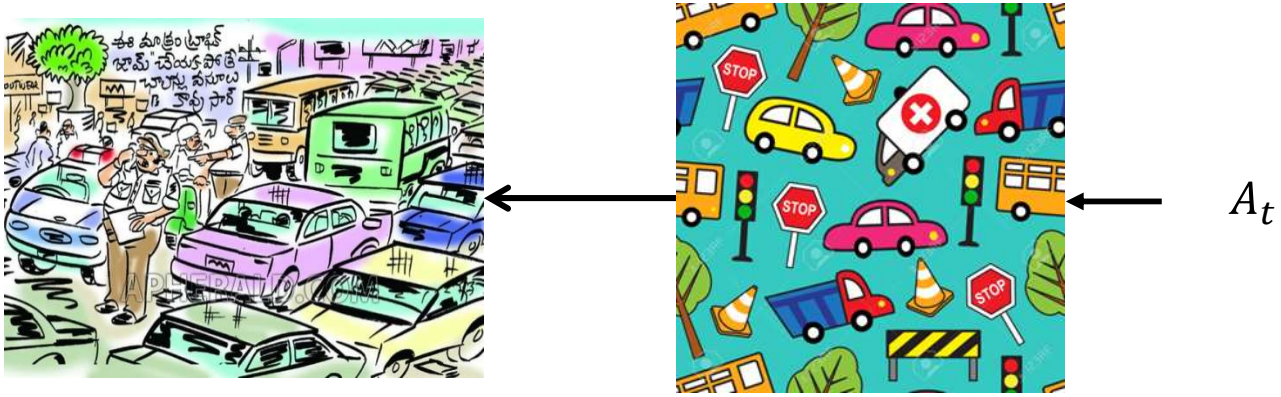


- **Assumption: The *true* environment state is Markov**

$$P(S_{t+1}|S_0, S_1, \dots, S_t) = P(S_{t+1}|S_t)$$

- The environment's future only depends on its present

Markov property



A_t

- **To be able to maximize his reward, the agent must ideally know all about the environment state and its dynamics..**

To Maximize Reward

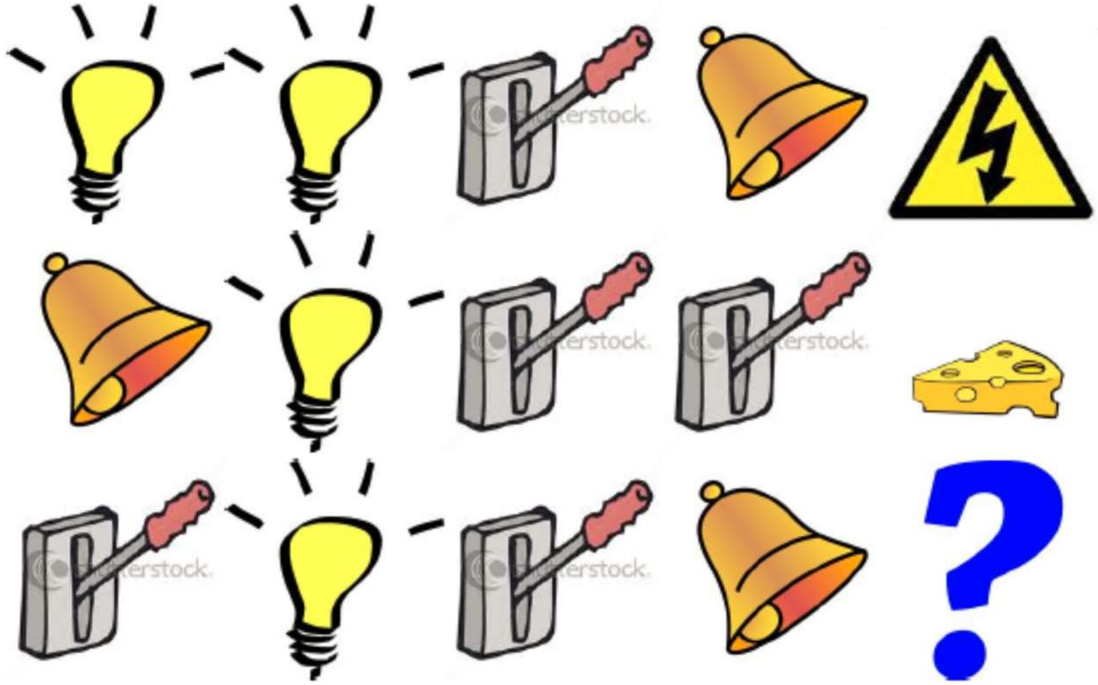
- We can represent the environment as a *process*
  - A mathematical characterization with a *true* value for its parameters representing the *actual* environment
- The agent must *model* this environment process
  - Formulate its own model for the environment, which must ideally match the true values as closely as possible
    - Based only on what it observes
- Agent must formulate winning strategy based on model of environment

The Agent's Side of the Story

- Agent has an internal representation of the environment state
  - May not match the true one at all
- May be defined in any manner
  - Formally the agent state $S_t = f(H_t)$ is some function of the history
  - The closer the agent's model is to the true environment state, the better the agent will be able to strategize

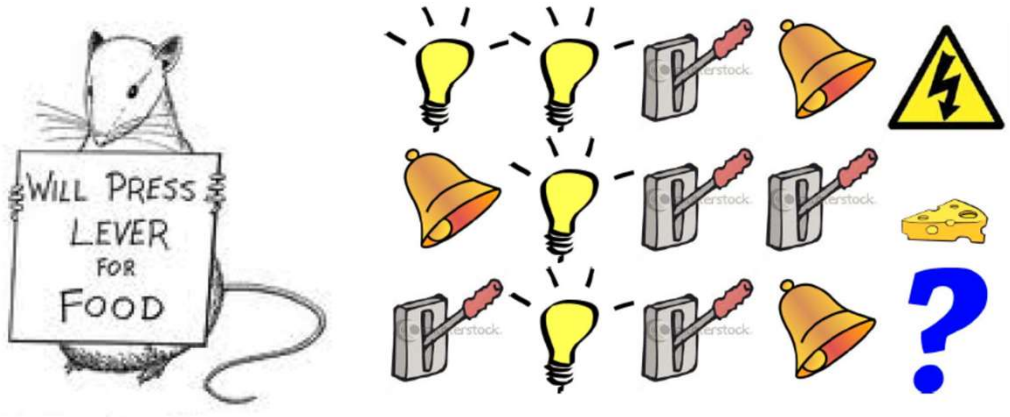
Defining Agent State

Image lifted from David Silver



- What is the outcome?

Defining Agent State



- Different definitions of state result in different predictions
- *True* environment state not really known
  - Would greatly improve prediction if known

Markov property and observability

- **Environment state is Markov**
  - An assumption that is generally valid for a properly defined *true* environment (information) state

$$P(S_{t+1}|S_0, S_1, \dots, S_t) = P(S_{t+1}|S_t)$$

- In theory, if the agent *doesn't* observe the environment's internals, he *cannot* model what he observes of the environment as Markov!
  - Amazing, but trivial result
  - E.g. the observations generated by an HMM are not Markov

- In practice, the agent may assume anything
  - The agent may only have a local model of the true state of the system
    - But can still assume that the states in its model behave in the same Markovian way that the environment's actual states do

Markov property and observability

Chess: environment state fully observable to agent

Poker: environment state only partially and indirectly observable to agent

- **Observability**
  - The agent's observations inform it about the environment state
  - The agent may observe the *entire* environment state
    - Now the agents state is isomorphic to the environment state
    - Note – observing the state is not the same as knowing the state's true dynamics $P(S_{t+1} = s_j | S_t = s_i)$
      - **Markov Decision Process**
  - Or only *part* of it
    - E.g. only seeing some stock prices, or only the traffic immediately in front of you
    - **Partially Observable Markov Decision Process**

Markov property and observability

Chess: environment state fully observable to agent

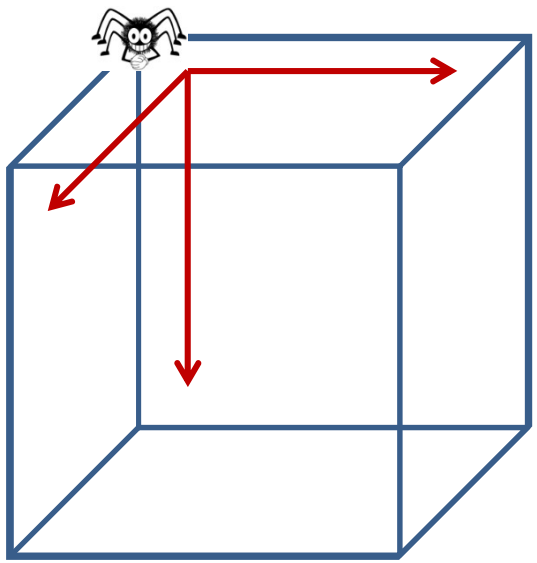
Poker: environment state only partially and indirectly observable to agent

- **Observability**
  - The agent's observations inform it about the environment state
  - The agent may observe the *entire* environment state
    - Now the agents state is isomorphic to the environment state
    - Note – observing the state is not the same as knowing the state's true dynamics $P(S_{t+1} = s_j | S_t = s_i)$
      - **Markov Decision Process**

We focus on this in our lectures

  - Or only *part* of it
    - E.g. only seeing some stock prices, or only the traffic immediately in front of you
    - **Partially Observable Markov Decision Process**

The World as we model It



Where the spider can go next only depends on where she is

- Definition of Markov property:
  - The state of the system has a Markov property if the future only depends on the present

$$P(S_{t+1}|S_0, S_1, \dots, S_t) = P(S_{t+1}|S_t)$$

- States can be *defined* to have this property

A Markov *Process*

- A Markov *process* is a random process where the future is only determined by the present
  - Memoryless

- Is fully defined by the set of states $\mathcal{S}$, and the *state transition probabilities* $P(s_i \,|s_j)$
  - Formally, the tuple $\mathrm{M} = \langle \mathcal{S}, \mathcal{P} \rangle$.
  - $\mathcal{S}$ is the (possibly finite) set of states
  - $\mathcal{P}$ is the complete set of transition probabilities $P(s \,|s')$
  - Note $P(s \,|s')$ stands for $P(S_{t+1} = s | S_t = s')$ at any time t
  - Will use the shorthand $P_{s,s'}$

The transition probability

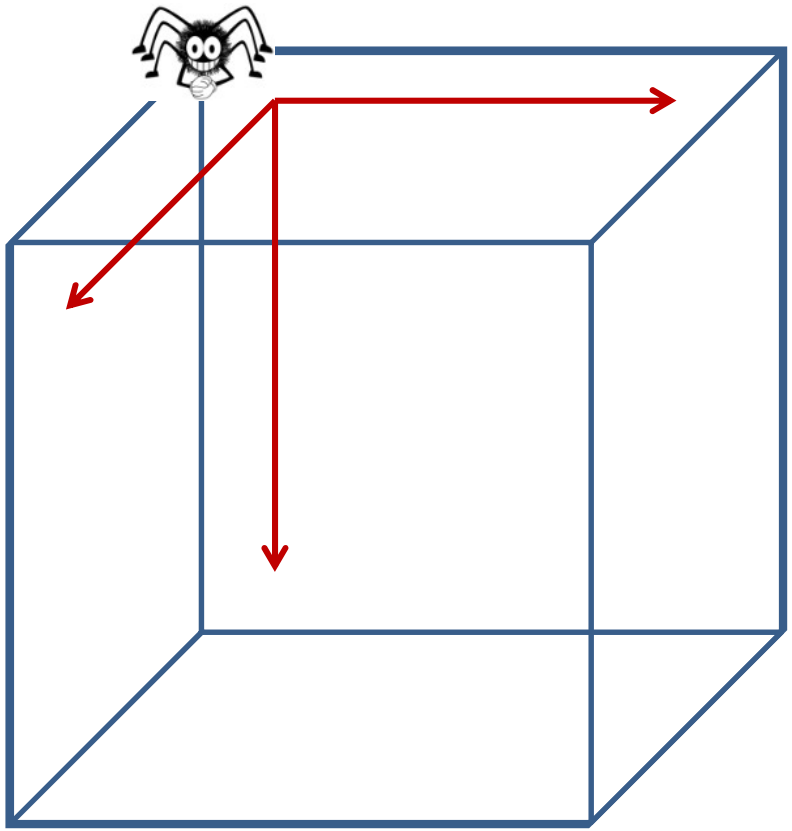
- For processes with a discrete, finite set of states, is generally arranged as *transition probability matrix*

$$\mathcal{P} = \begin{bmatrix} P_{s_1,s_1} & P_{s_2,s_1} & \cdots & P_{s_N,s_1} \\ P_{s_1,s_2} & P_{s_2,s_2} & \cdots & P_{s_N,s_2} \\ \vdots & \vdots & \ddots & \vdots \\ P_{s_1,s_N} & P_{s_2,s_N} & \cdots & P_{s_N,s_N} \end{bmatrix}$$

- More generally (for continuous-state processes, e.g. the state of an automobile), it is modelled as a parametric distribution

$$P_{s,s'} = f(s; \theta_{s'})$$

State Transition Probabilities

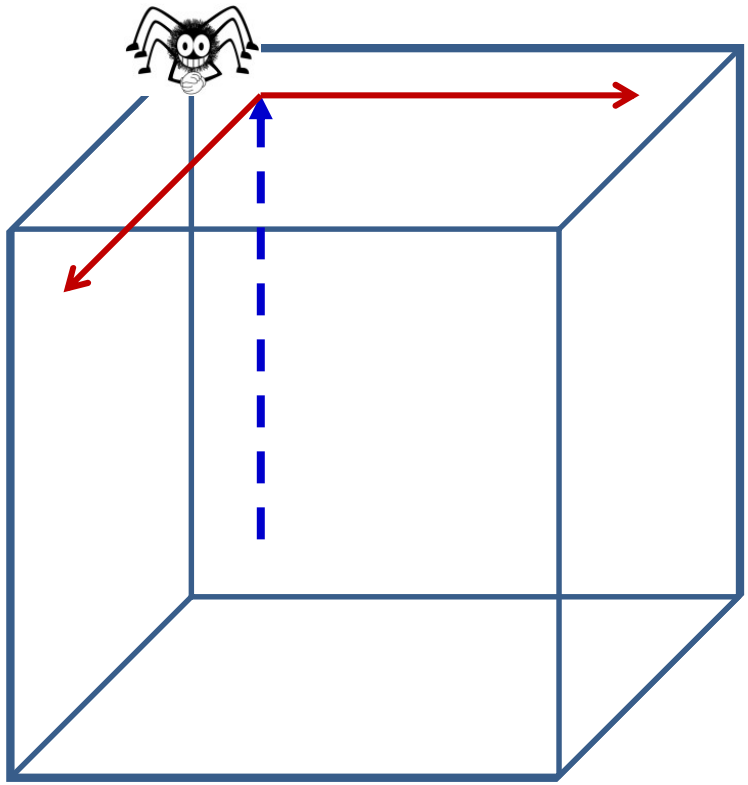


Where the spider can go next only depends on where she is

From any corner, she is equally likely to wander off in any direction

- What is the transition probability matrix?

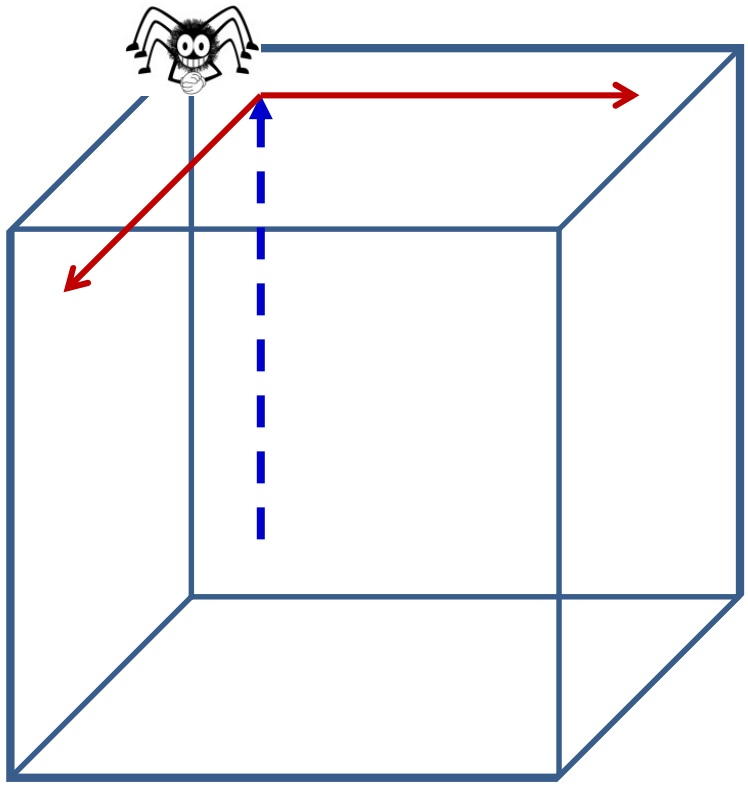
The World as we model It



This spider does not like to turn back

Is this a Markov process?

The World as we model It



This spider does not like to turn back

Is this a Markov process?

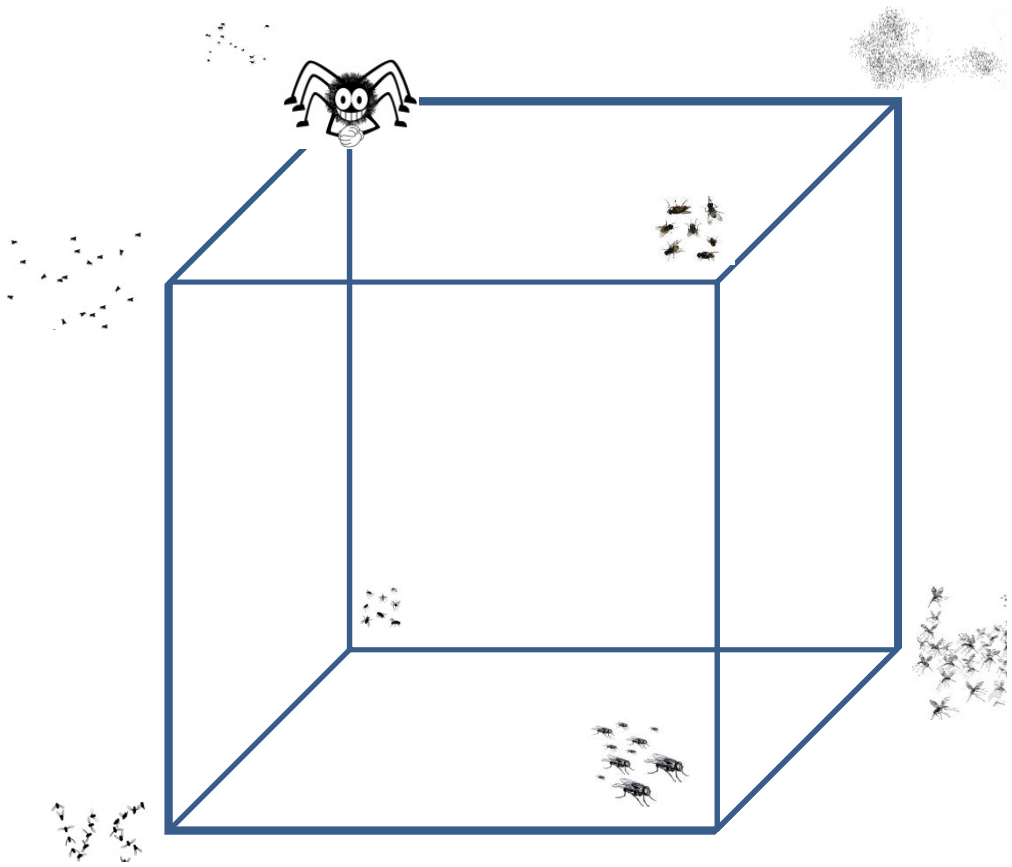
How many states?
What is the transition matrix?

A Markov Reward Process

- A Markov *Reward* Process (MRP) is a Markov Process where states give you rewards

- At each state s, upon arriving at that state, you obtain a reward r, drawn from a distribution $P(r|s)$

Markov Reward Process

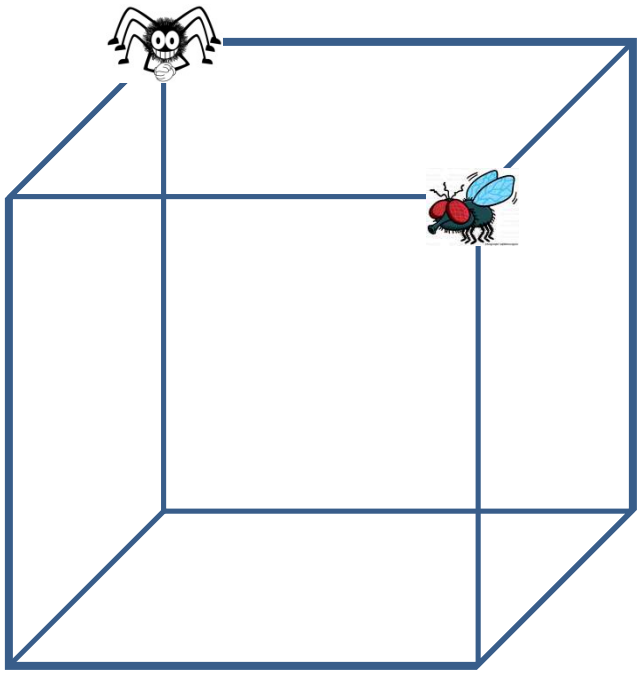


Reward: Upon arriving at any corner, the spider may catch a fly from the swarm hovering there

Rewards are corner specific and probabilistic: Different corners have different sized swarms with flies of different sizes. The spider only has a *probability* of catching a fly, but may not always catch one.

- Flider and the Markov reward process!

Is This an MRP?



- Is this a Markov Reward Process?

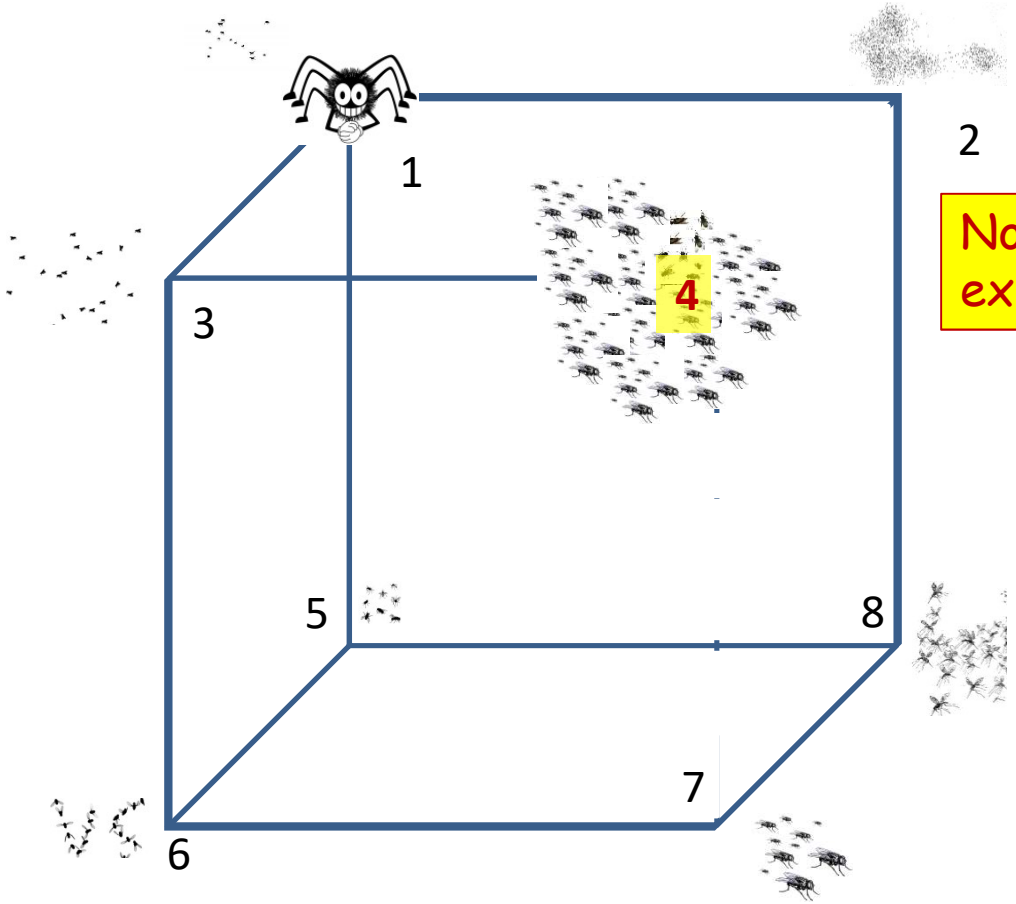
Markov Reward Process

- Formally, a Markov Reward Process is the tuple $\mathrm{M} = \langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is the (possibly finite) set of states
  - $\mathcal{P}$ is the complete set of transition probabilities $P_{s,s'}$,
  - $\mathcal{R}$ is a *reward* function, consisting of the distributions $P(r|s)$
    - Or alternately, the expected value $R_s = E[r|s]$
  - $\gamma \in [0,1]$ is a *discount* factor

Markov Reward Process

- Formally, a Markov Reward Process is the tuple $\mathrm{M} = \langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is the (possibly finite) set of states
  - $\mathcal{P}$ is the complete set of transition probabilities $P_{s,s'}$
  - $\mathcal{R}$ is a *reward* function, consisting of the distributions $P(r|s)$
    - Or alternately, the **expected** value $R_s = E[r|s]$
  - $\gamma \in [0,1]$ is a *discount* factor *What on earth is this?*

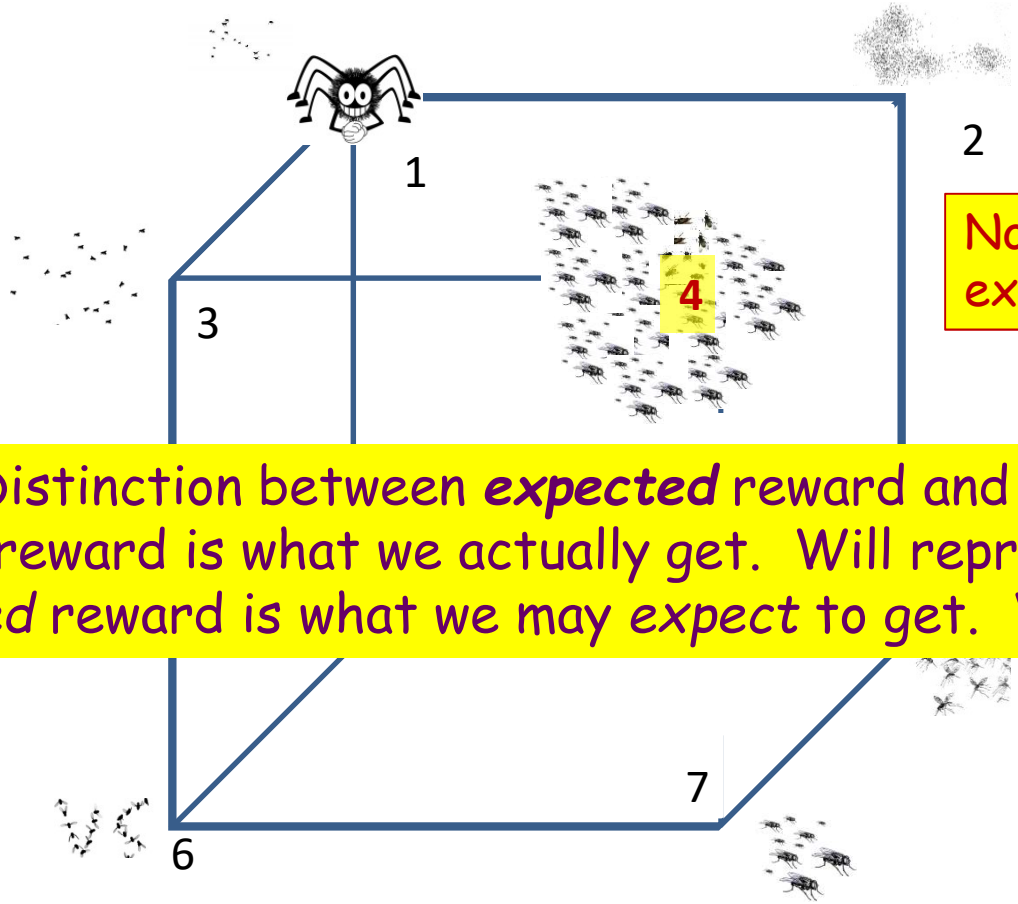
Rewards and *Expected* rewards



No route to corner 4 except from corner 3

- One step *expected* reward: R_1
  - Will this be greater if the spider heads to corner 2 or to corner 3?

Rewards and *Expected Rewards*

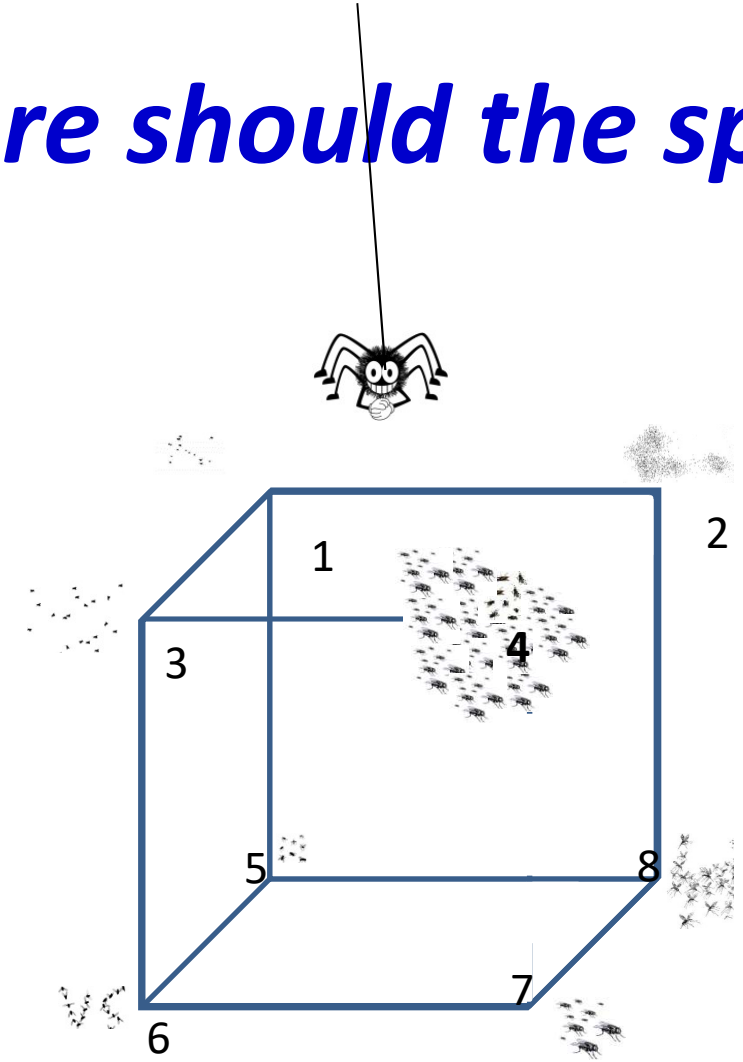


No route to corner 4 except from corner 3

Note: Distinction between ***expected*** reward and ***sample*** reward
Sample reward is what we actually get. Will represent by r
Expected reward is what we may *expect* to get. Will represent by R

- One step *expected* reward: R_1
  - Will this greater if the spider heads to corner 2 or to corner 3?

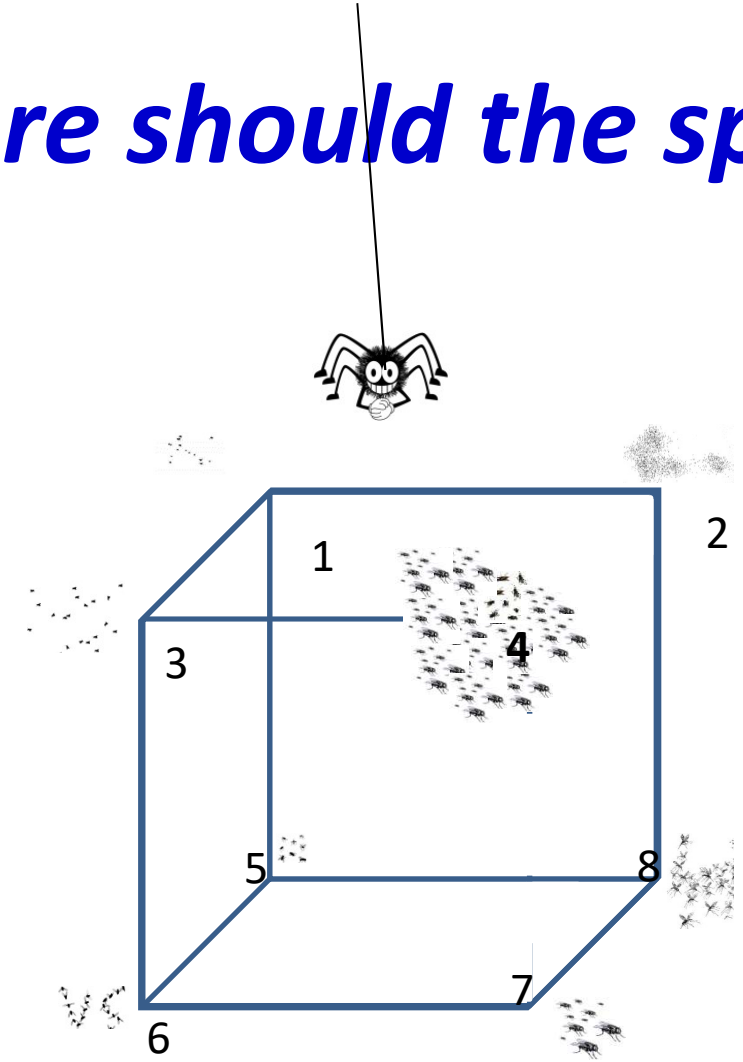
Where should the spider be?



- Flider has the option of landing on corner 1, 2 or 3 before she begins wandering the room
  - Which is the better corner to land on?

Where should the spider be?

Need to know the *long-term consequences* of landing in the two corners

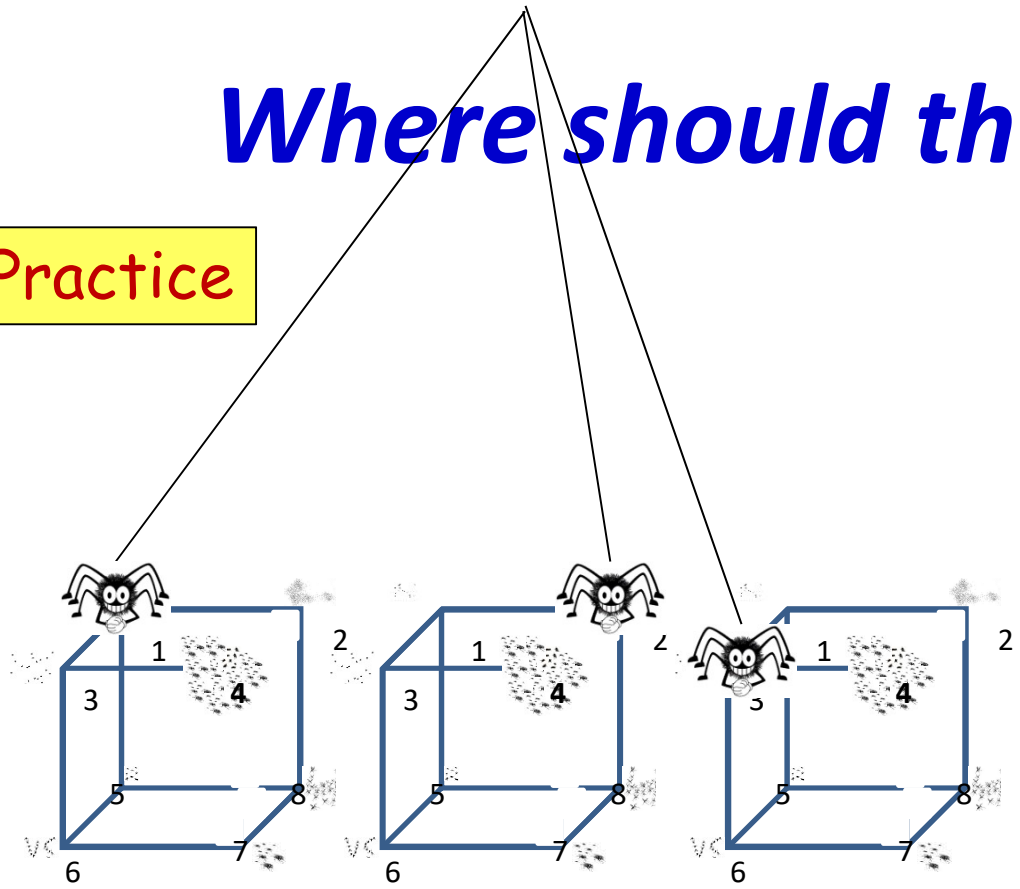


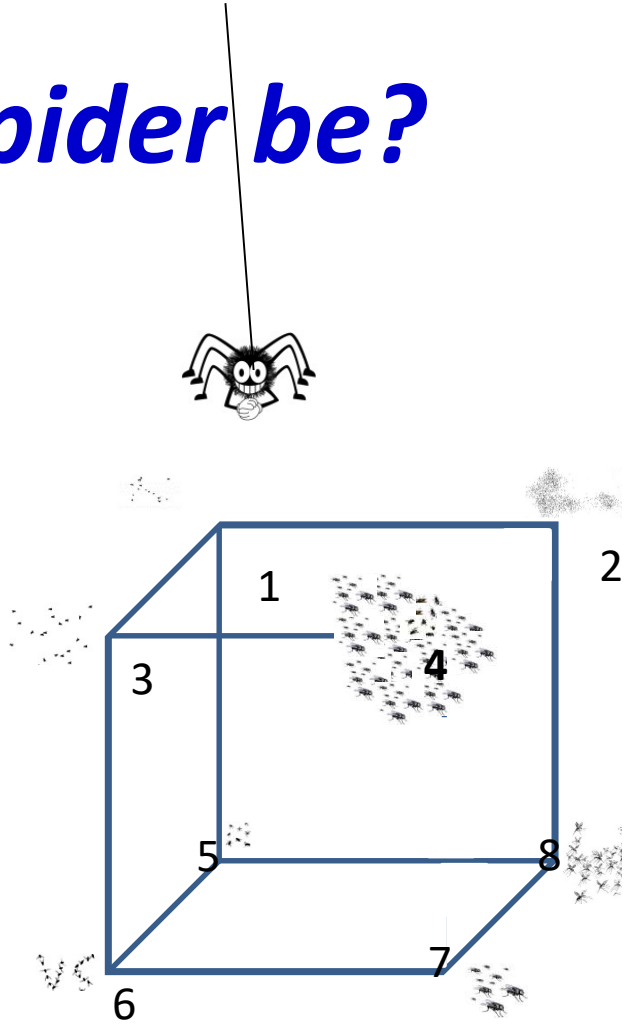
Where can she expect to get more food *in the long term?*

- Flider has the option of landing on corner 1, 2 or 3 before she begins wandering the room
  - Which is the better corner to land on?

Where should the spider be?

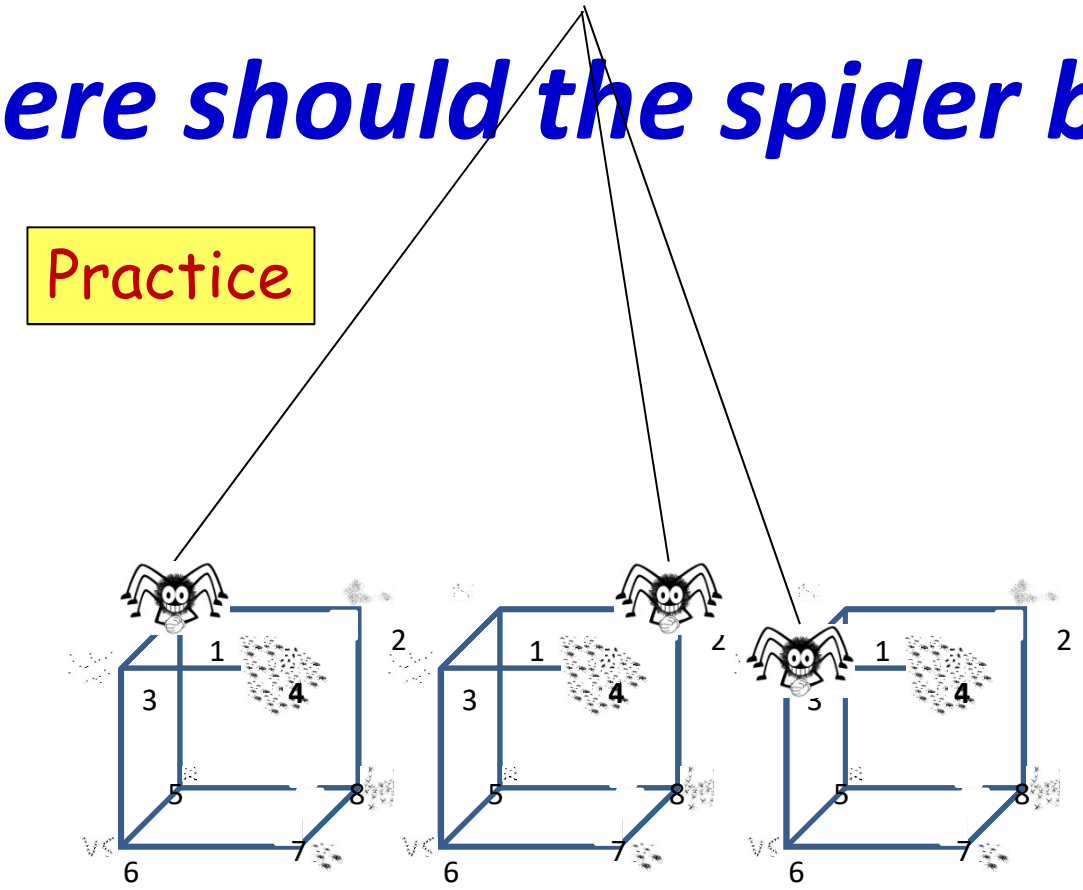
Practice





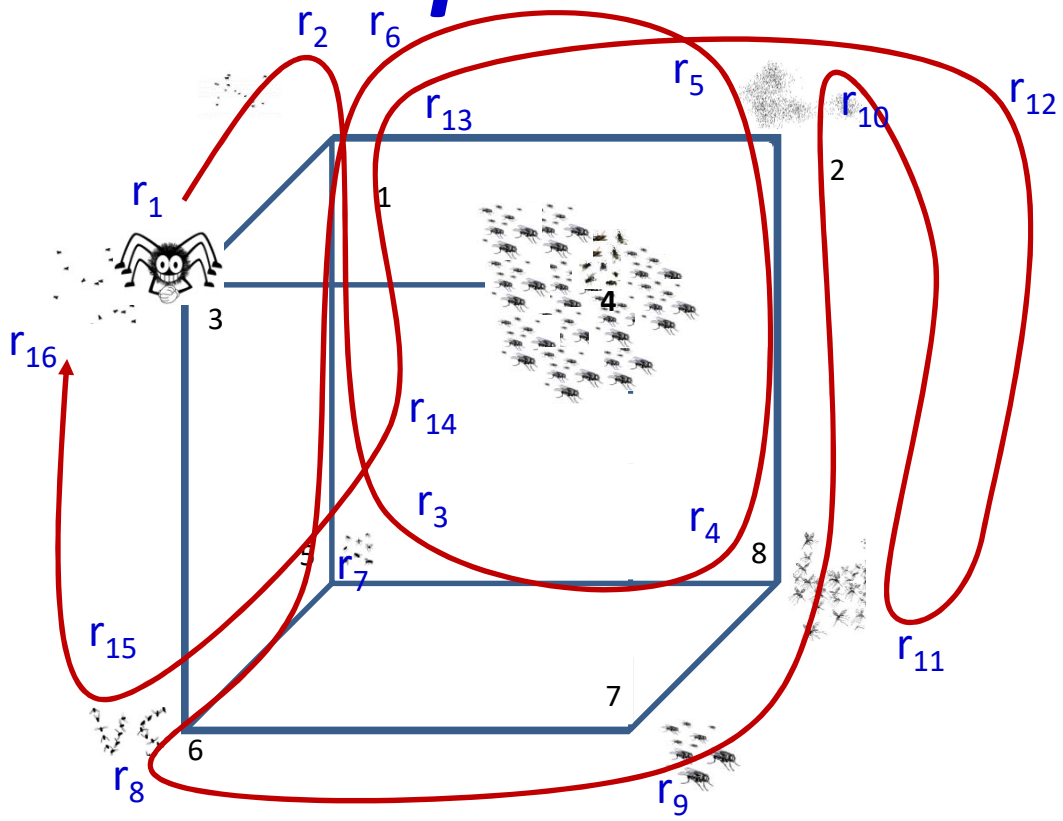
- Assume she is allowed to “practice” once from each corner
  - To plan her future strategy

Where should the spider be?



- Must use her "practice" turn to assign a "value" to each of the corners
  - Guess how much food she would get in the long term from that corner

Flider practices



- Starting from 3, she gets r_1, r_2, r_3….
- Is $r_1 + r_2 + r_3$ … a realistic representation of what she'd get if she did it again?

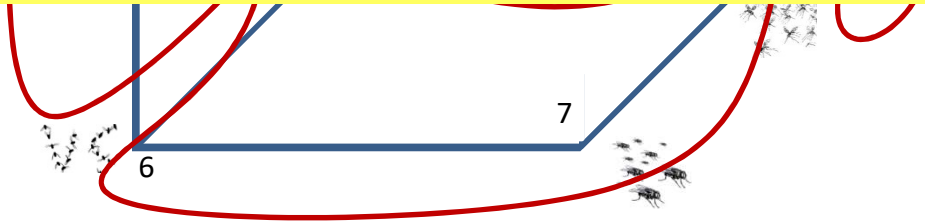
Flider practices

r_1 is somewhat realistic – it is obtained from corner 3

r_2: she had a choice of 3 corners for her next stop and chose one randomly during practice. Unlikely she'll go to the same corner in the next run (less representative)

r_3: she had 9 possible corners to choose from in 2 steps. r_3 is even less representative of future runs

And so on…



- Starting from 3, she gets r_1, r_2, r_3….
- Is $r_1 + r_2 + r_3$ … a realistic representation of what she'd get if she did it again?

Flider practices

r_1 is somewhat realistic – it is obtained from corner 3

r_2: she had a choice of 3 corners for her next stop and chose one randomly during practice. Unlikely she'll go to the same corner in the next run (less representative)

r_3: she had 9 possible corners to choose from in 2 steps. r_3 is even less representative of future runs

And so on…

A better guess for how good it is to land at "3":

$$r_1 + a_1 r_2 + a_2 r_3 + a_3 r_4 + \cdots$$

Where $0 \leq a_i \leq 1$
(you "trust" the readings from farther in the future less)

- Is $r_1 + r_2 + r_3$ … a realistic representation of what she'd get if she did it again?

Flider practices

r_1 is somewhat realistic – it is obtained from corner 3

r_2: she had a choice of 3 corners for her next stop and chose one randomly during practice. Unlikely she'll go to the same corner in the next run (less representative)

r_3: she had 9 possible corners to choose from in 2 steps. r_3 is even less representative of future runs

And so on…

A better guess for how good it is to land at "3":

$$r_1 + a_1 r_2 + a_2 r_3 + a_3 r_4 + \cdots$$

Where $0 \leq a_i \leq 1$
(you "trust" the readings from farther in the future less)

- A "mathematically good" choice: $a_i = \gamma^i$ where $0 \leq \gamma \leq 1$ hat she'd get if she did it again?

The discounted return

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- The *return* is the total *future* reward all the way to the end
- But each future step is slightly less "believable" and is hence discounted
  - We trust our own observations of the future less and less
    - The future is a fuzzy place

- The discount factor γ is our belief in the predictability of the future
  - $\gamma = 0$: The future is totally unpredictable, only trust what you see immediately ahead of you (myopic)
  - $\gamma = 1$: The future is clear; consider all of it (far sighted)
- **Part of the Markov Reward Process model**

The discounted return

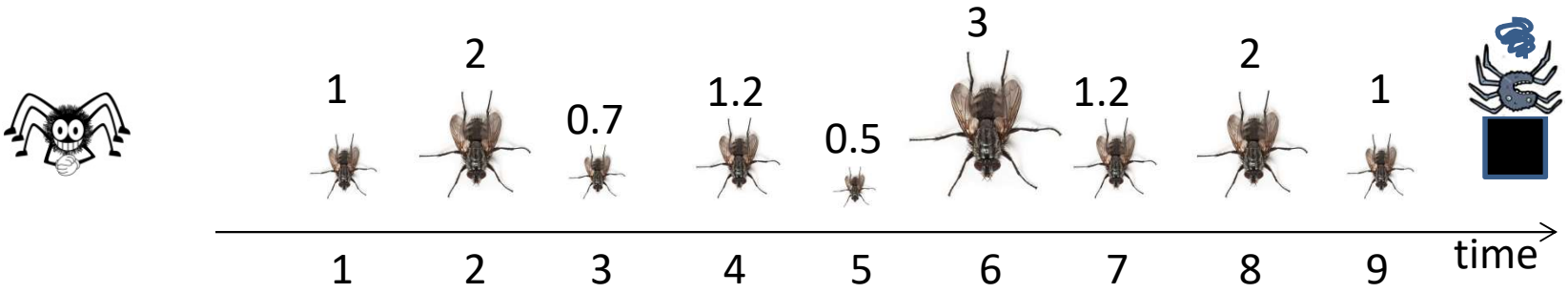
Caveat: Weird notation. r_{t+1} is actually associated with the state at time t

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- The *return* is the total *future* reward all the way to the end
- But each future step is slightly less "believable" and is hence discounted
  - We trust our own observations of the future less and less
    - The future is a fuzzy place

- The discount factor γ is our belief in the predictability of the future
  - $\gamma = 0$: The future is totally unpredictable, only trust what you see immediately ahead of you (myopic)
  - $\gamma = 1$: The future is clear; consider all of it (far sighted)
- **Part of the Markov Reward Process model**

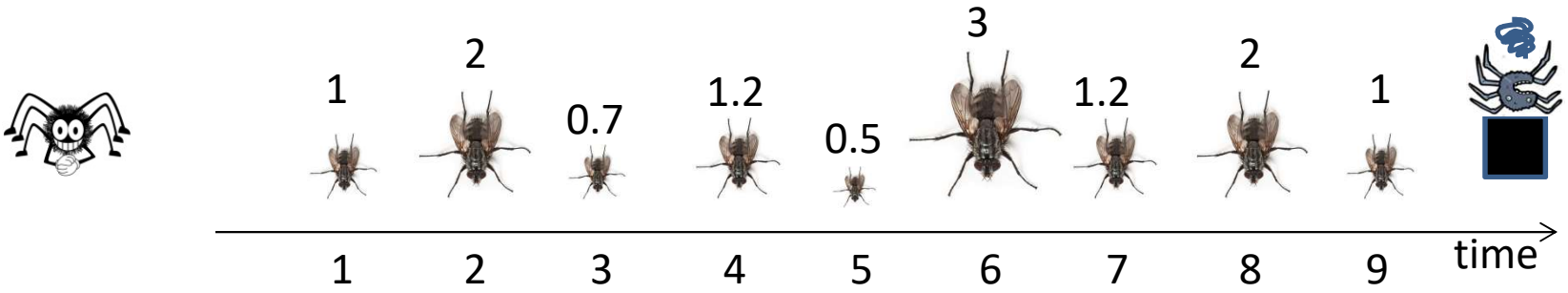
Rewards



- Our spider goes wandering..
  $r_1 = 1, r_2 = 2, r_3 = 0.7, r_4 = 1.2, r_5 = 0.5, \ldots$

- Are these *sample rewards* or *expected rewards?*

Rewards

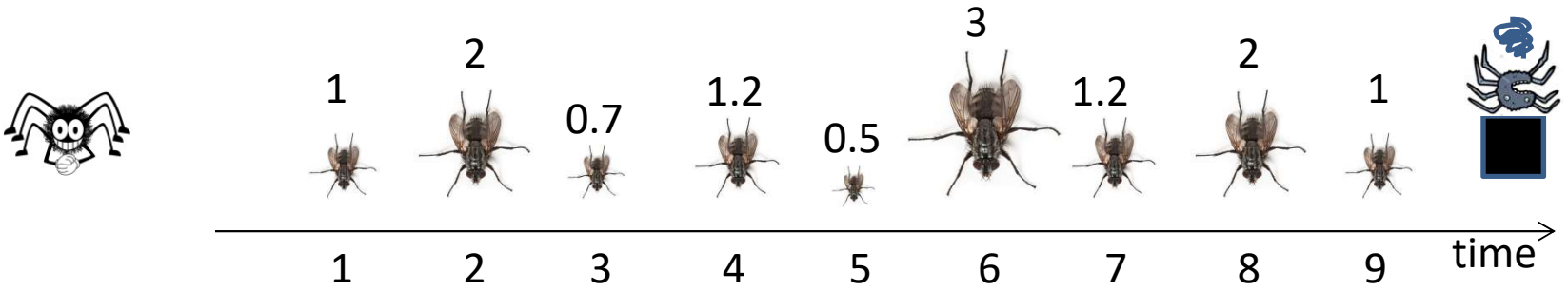


- Our spider goes wandering..

$$r_1 = 1, r_2 = 2, r_3 = 0.7, r_4 = 1.2, r_5 = 0.5, \ldots$$

- Are these *sample rewards* or *expected rewards?*
- What are the *expected* rewards at t=1,2,…

Rewards

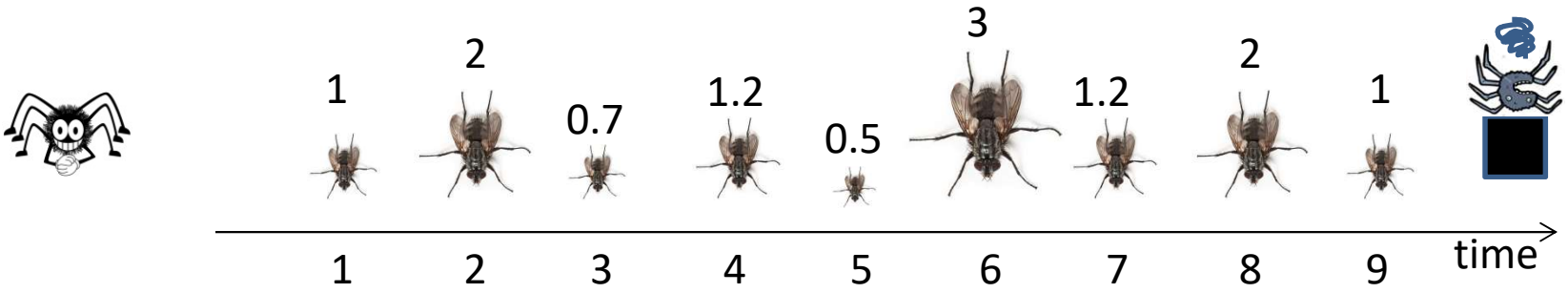


- Our spider goes wandering..

$$r_1 = 1, r_2 = 2, r_3 = 0.7, r_4 = 1.2, r_5 = 0.5, \ldots$$

- Are these *sample rewards* or *expected rewards?*
- What are the *expected* rewards at t=1,2,…
- Under what condition would both be the same?

Returns

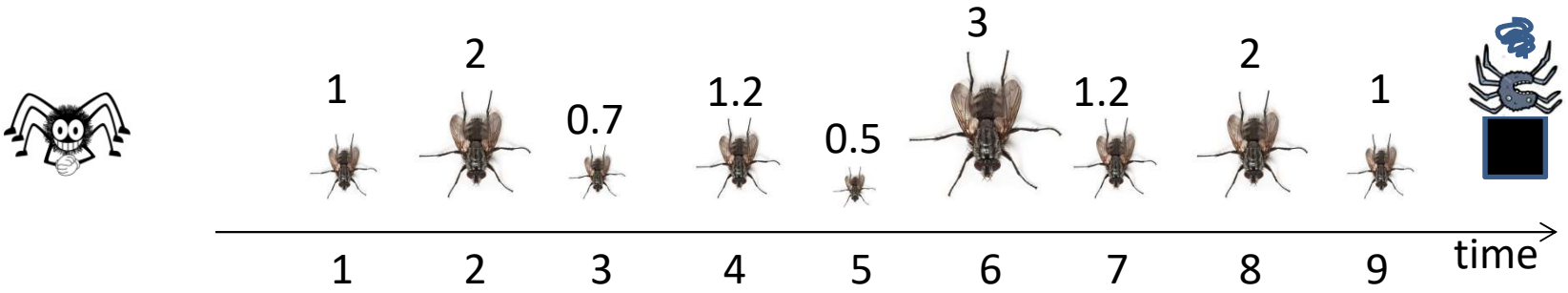


- Our spider goes wandering..

$$r_1 = 1, r_2 = 2, r_3 = 0.7, r_4 = 1.2, r_5 = 0.5, \ldots$$

- We decide the discounting factor $\gamma = 1$
  - Really trusting the future
- What is the return G_t at $t = 1$?

Returns

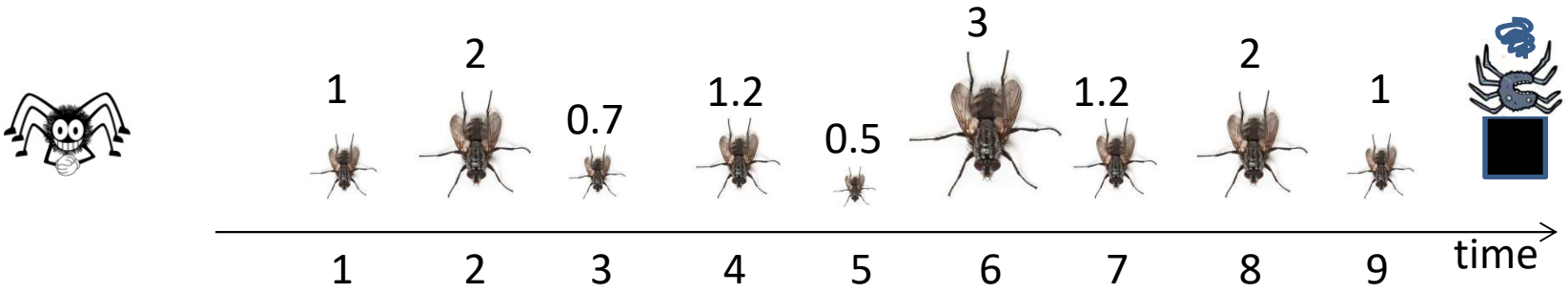


- Our spider goes wandering..

$$r_1 = 1, r_2 = 2, r_3 = 0.7, r_4 = 1.2, r_5 = 0.5, \ldots$$

- We decide the discounting factor $\gamma = 1$
  - Really trusting the future
- What is the return G_t at $t = 1$?
- What is the return G_t at $t = 7$?

Returns

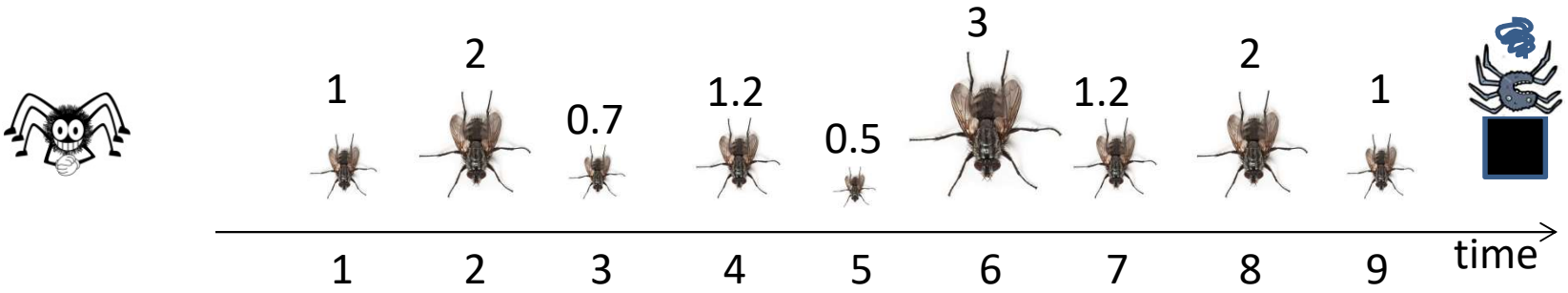


- Our spider goes wandering..

$$r_1 = 1, r_2 = 2, r_3 = 0.7, r_4 = 1.2, r_5 = 0.5, \ldots$$

- We decide the discounting factor $\gamma = 1$
  - Really trusting the future
- What is the return G_t at $t = 1$?
- What is the return G_t at $t = 7$?
- Are these *sample* returns or *expected* returns?

Returns



- Our spider goes wandering..

  $r_1 = 1, r_2 = 2, r_3 = 0.7, r_4 = 1.2, r_5 = 0.5, \ldots$

- What is the return at $t = 2$ with $\gamma = 0.5$

Returns

- Discounted sample returns G_t by themselves carry a fuzzy meaning
  - Why should we discount something we already observed?
- However, they make sense as *samples* of the possible future when you are at any state
  - If you are at any state, what is the *expected* return $E[G_t]$

Introducing the "Value" function

- The "Value" of a state is the expected total discounted return, when the process begins in that state

$$V_s = E[G_0 | S_0 = s]$$

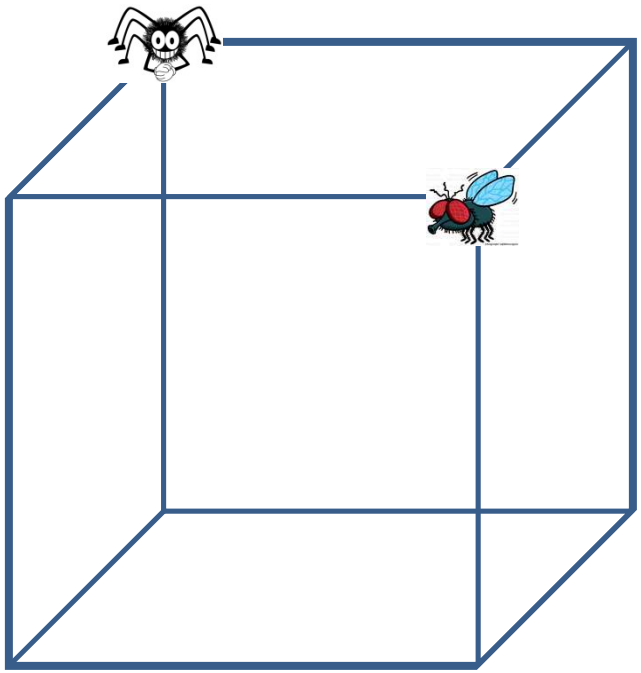
- Or, since the process is Markov and the future only depends on the present and not the past

$$V_s = E[G_t | S_t = s]$$

- Or more generally

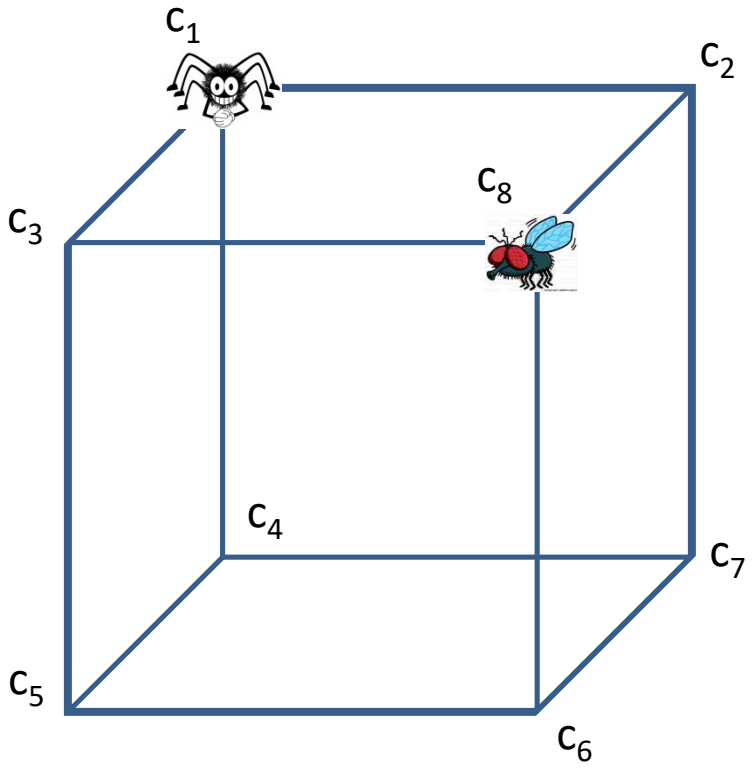
$$V_s = E[G | S = s]$$

The spider again



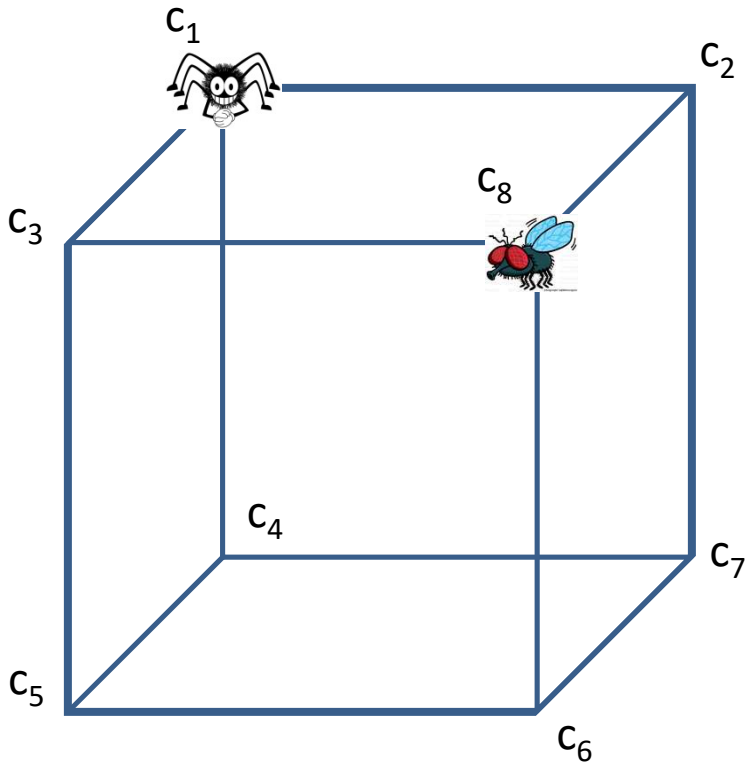
- The spider gains a reward of value 1 if she consumes the fly
- The spider has infinite patience
- What is the value of starting at each corner?

The spider again



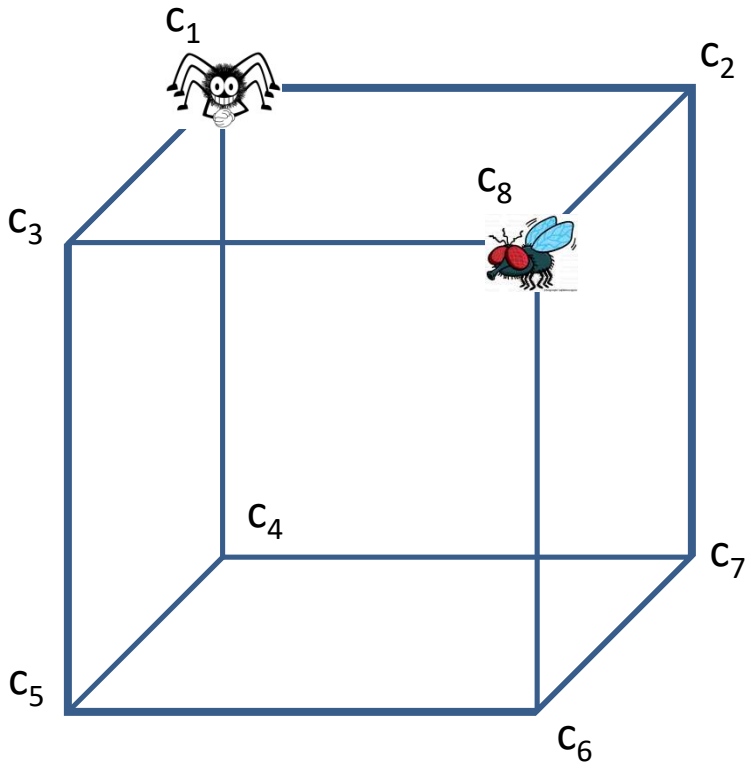
- Regardless of which corner the spider starts at, she will eventually, randomly, nab the fly
- The expected return from any corner is 1!

The spider



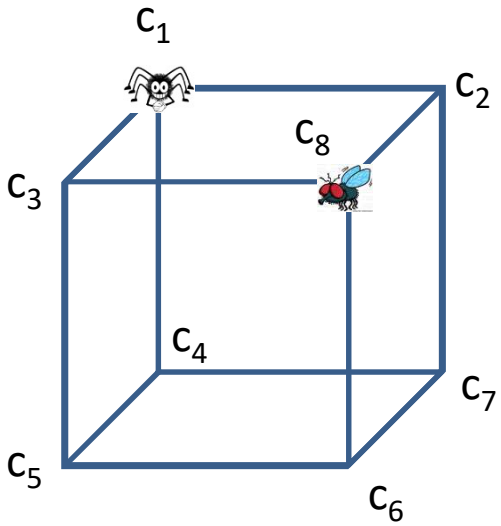
- The *value of being at any corner is 1 for all corners*
  - She can expect to get a fly from anywhere

The *hungry* spider



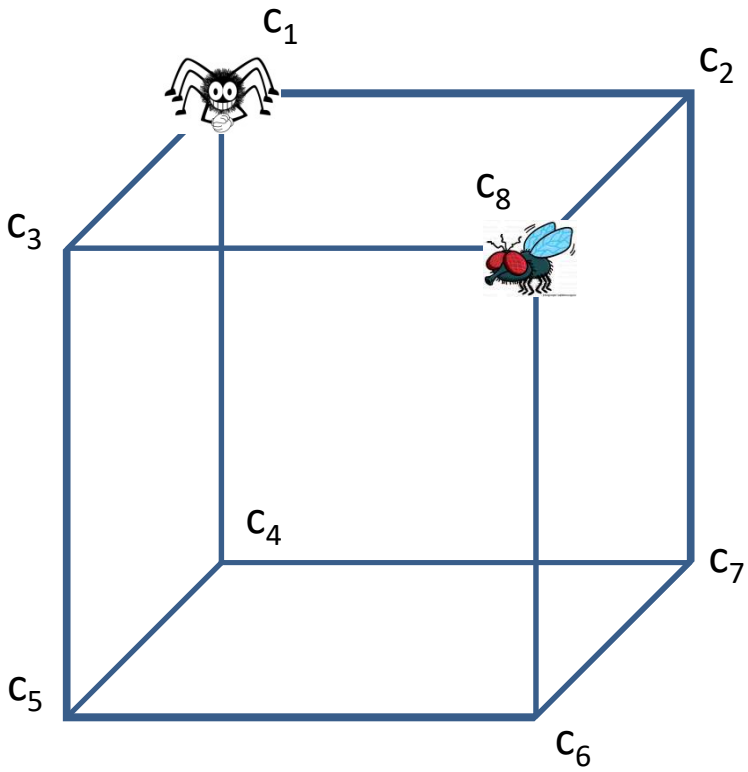
- The spider is hungry
- She gets a negative reward of -1 for every minute spent finding food
- What is the expected return if she starts at c_1

The *hungry* spider



- Posing the problem: There is a total reward/penalty associated with each corner
  - -1 if the corner has no fly
    - Will definitely spend at least one more minute hunting
  - 1 at the corner that has the fly (satisfied!)
- Thus $r_{c_x} = -1$ for $c_1 \ldots c_7$
- $r_{c_8} = 1$
- Note: We could also assign costs/rewards to edges in addition to nodes, if we want more detail, but wont do so for our lectures

The *hungry* spider



$$V_{c_1} = -1 + \frac{1}{3}V_{c_2} + \frac{1}{3}V_{c_3} + \frac{1}{3}V_{c_4}$$

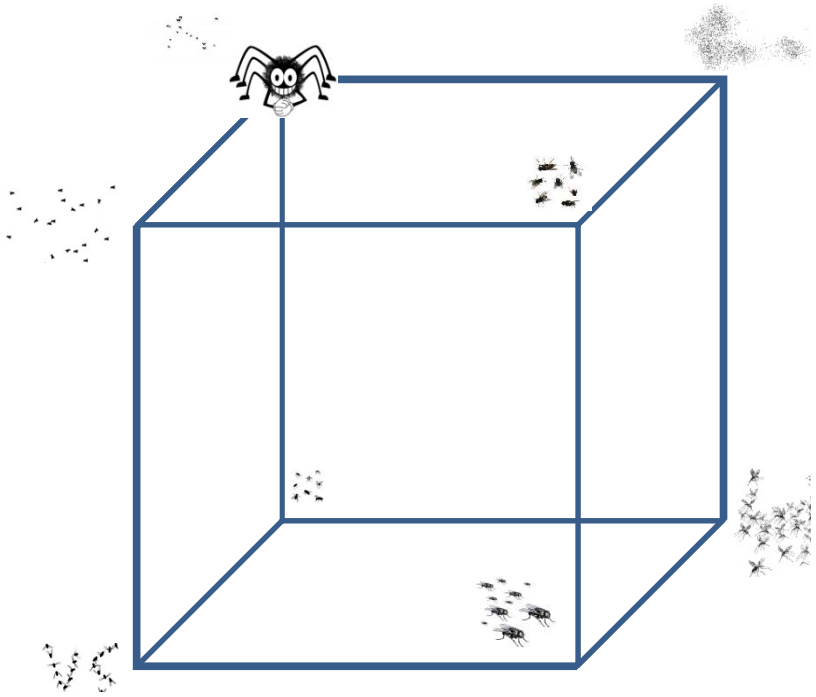
$$V_{c_2} = -1 + \frac{1}{3}V_{c_1} + \frac{1}{3}V_{c_7} + \frac{1}{3}V_{c_8}$$

$$\vdots$$

$$V_{c_8} = 1$$

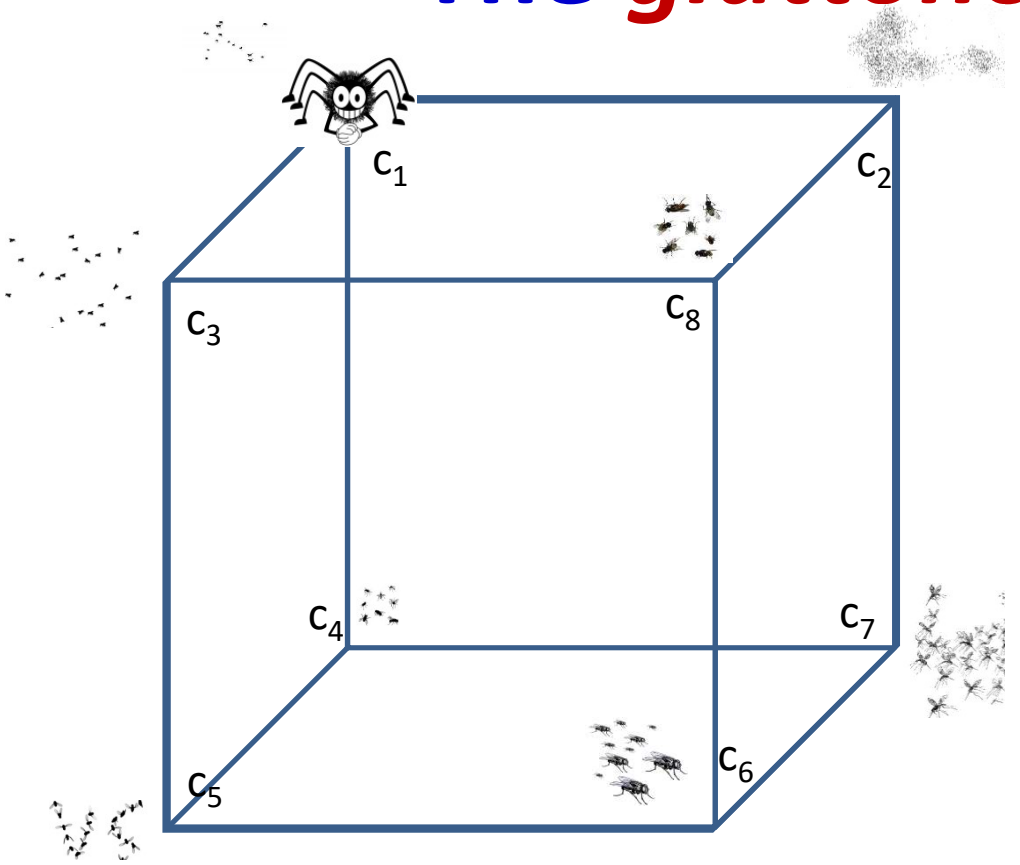
- A familiar solution
- Assuming $\gamma = 1$
  - A natural fit in this problem

The *gluttonous* spider



- There are flies at every corner
- The expected reward *after* arriving at any corner is R_{c_x}
  - The average size of the fly there minus the travel penalty
- Immediate rewards matter
  - Ideally $\gamma < 1$
  - Give more importance to immediate rewards than future ones

The *gluttonous* spider



$$V_{c_1} = R_{c_1} + \gamma\left(\frac{1}{3}V_{c_2} + \frac{1}{3}V_{c_3} + \frac{1}{3}V_{c_4}\right)$$

$$V_{c_2} = R_{c_2} + \gamma\left(\frac{1}{3}V_{c_1} + \frac{1}{3}V_{c_7} + \frac{1}{3}V_{c_8}\right)$$

$$\vdots$$

$$V_{c_8} = R_{c_8} + \gamma\left(\frac{1}{3}V_{c_2} + \frac{1}{3}V_{c_3} + \frac{1}{3}V_{c_6}\right)$$

- A familiar solution
- What happens if $\gamma = 1$?

The Bellman Expectation Equation

- The value function of a state is the *expected discounted return*, when the process begins at that state

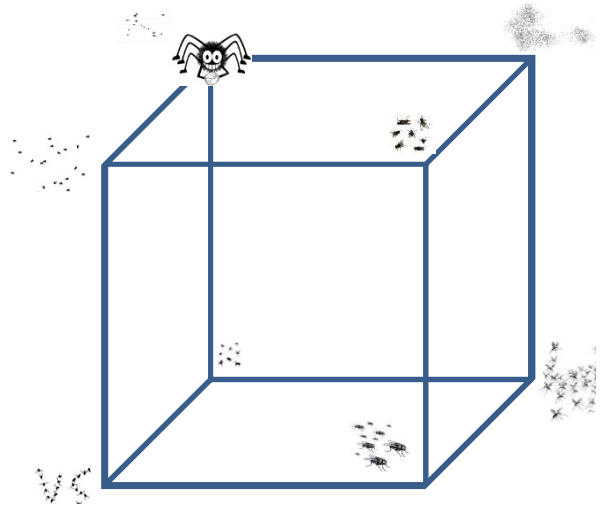
$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

$$V_s = E[G|S = s]$$

- **The Bellman Expectation Equation:**

$$V_s = R_s + \gamma \sum_{s'} P_{s',s} V_{s'}$$

Why discounted return?



- In processes with infinite horizon, which can go on for ever, the total undiscounted return will be infinite for every path $\sum_{k=0}^{\infty} r_{t+k+1}$ will be infinite for every path
  - For finite horizon processes, a discount factor $\gamma = 1$ is good. It lets us talk in terms of actual total return
  - For infinite horizon processes, discounting $\gamma < 1$ is required for meaningful mathematical analysis : $\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

The Bellman Expectation Equation

$$V_s = R_s + \gamma \sum_{s'} P_{s',s} V_{s'}$$

$$\begin{bmatrix} V_{s_1} \\ V_{s_2} \\ \vdots \\ V_{s_N} \end{bmatrix} = \begin{bmatrix} R_{s_1} \\ R_{s_2} \\ \vdots \\ R_{s_N} \end{bmatrix} + \gamma \begin{bmatrix} P_{s_1,s_1} & P_{s_2,s_1} & \cdots & P_{s_N,s_1} \\ P_{s_1,s_2} & P_{s_2,s_2} & \cdots & P_{s_N,s_2} \\ \vdots & \vdots & \ddots & \vdots \\ P_{s_1,s_N} & P_{s_2,s_N} & \cdots & P_{s_N,s_N} \end{bmatrix} \begin{bmatrix} V_{s_1} \\ V_{s_2} \\ \vdots \\ V_{s_N} \end{bmatrix}$$

$$\mathcal{V} = \mathcal{R} + \gamma \mathcal{P} \mathcal{V}$$

- Bellman expectation equation in matrix form

The Bellman Expectation Equation

$$\mathcal{V} = \mathcal{R} + \gamma \mathcal{P} \mathcal{V}$$

- Given the MRP $\mathrm{M} = \langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - I.e. the expected rewards at every state, and the transition probability matrix,
  - the value functions for all states can be easily computed through matrix inversion

$$\mathcal{V} = (I - \gamma \mathcal{P})^{-1} \mathcal{R}$$

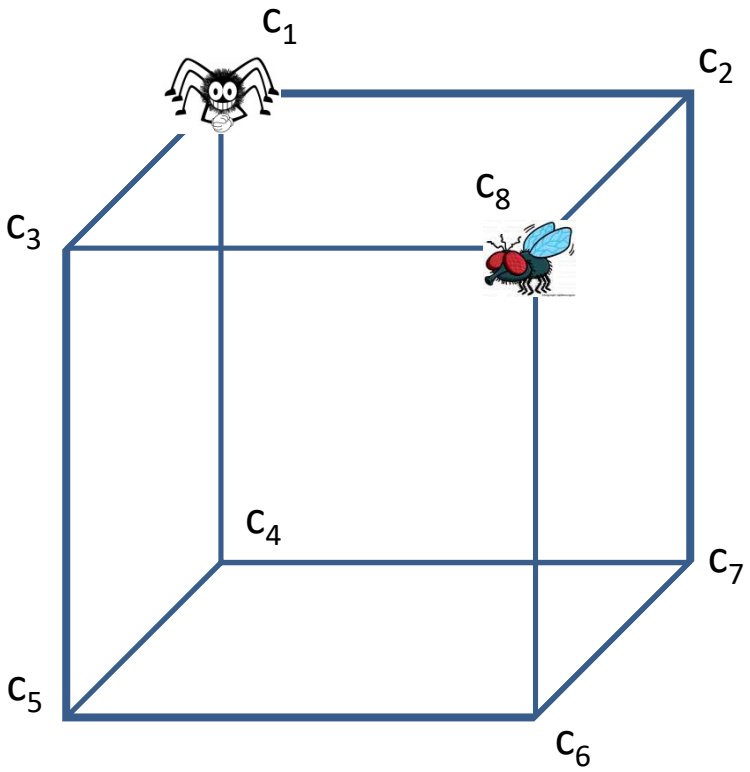
- Finding the values of states is a key problem in planning and reinforcement learning

- Unfortunately, for very large state spaces, the above matrix inversion is not tractable
  - Also not invertible for small state spaces if $\gamma = 1$
    - Inversion cannot be used to find $\mathcal{V}$ even when it is finite (e.g. our fly problem), if $\gamma = 1$
- Much of what we will deal with is how to tackle this problem

Moving on..

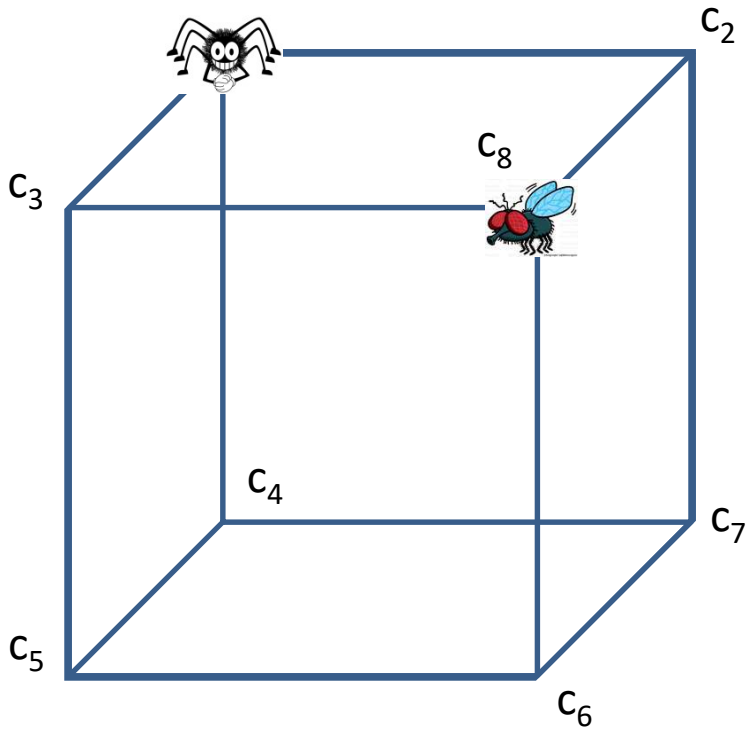
- Up next … Markov *Decision* Processes

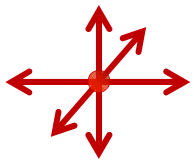
MDP



- We have assumed so far that the agent behaves randomly
  - The agent has no *agency*
  - Lets make the agent more intelligent..

A more realistic problem

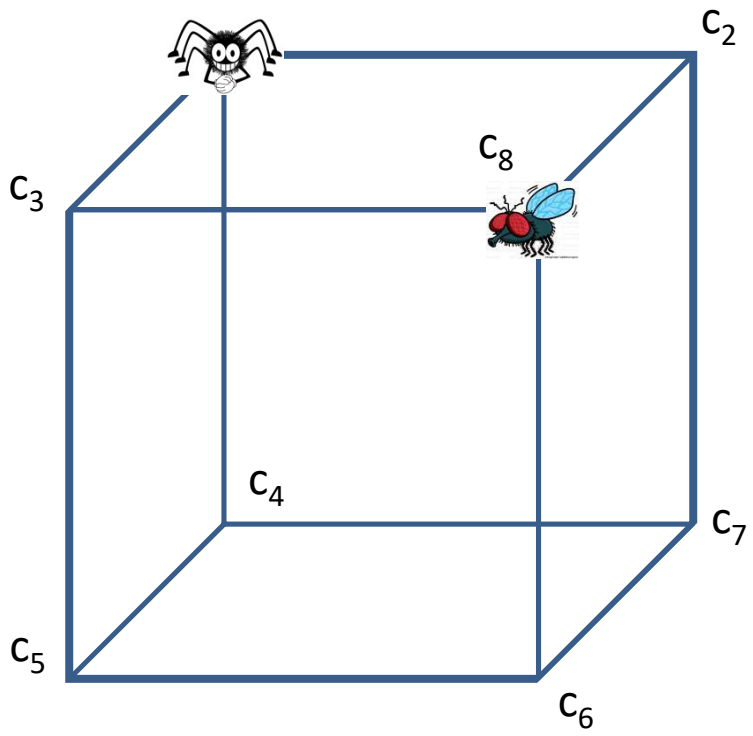




Full set of possible actions

- The spider actively chooses which way to move
  - The agent *takes action*
  - Ideally, it would move in the general direction of the fly
- However, each time the spider moves, the fly jumps up and settles at another corner
  - The agent's action changes the environment!

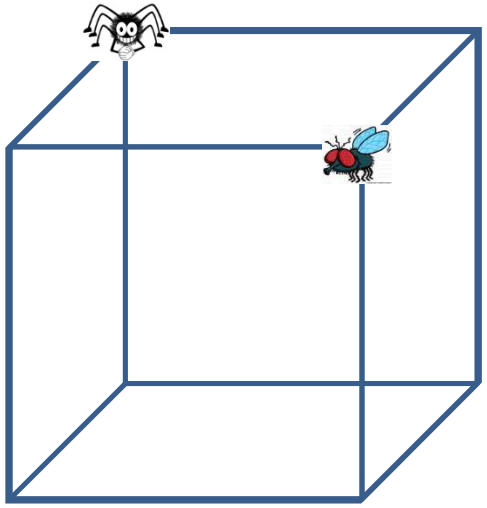
A more realistic problem

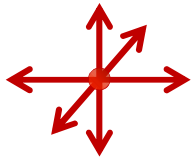


How do we model this system?

- The spider actively chooses which way to move
  - The agent *takes action*
  - Ideally, it would move in the general direction of the fly
- However, each time the spider moves, the fly jumps up and settles at another corner
  - The agent's action changes the environment!

Redefining the problem

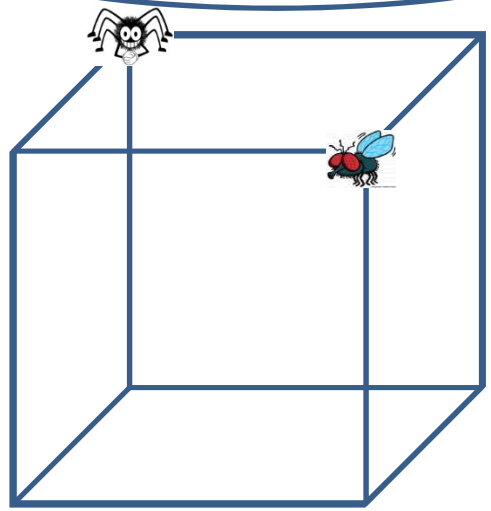


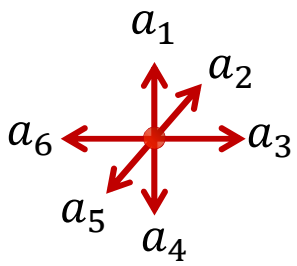


Full set of possible actions

- Each time the spider moves in any direction, the fly randomly jumps
- The fly arrives at a new state but ..
  - The state it arrives in depends on where the fly jumped
  - Which depends on which direction the Spider moved
- The spider's action *modifies the state transition probabilities!!*

What is $P^{a}_{s,s'}$

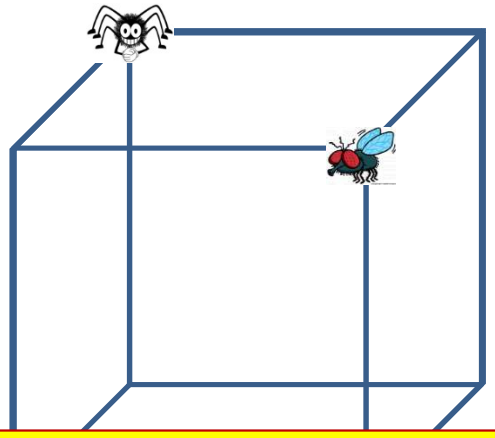


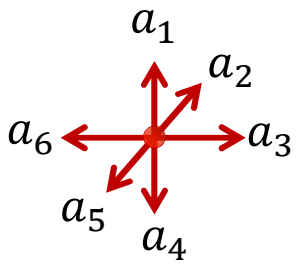


Full set of possible actions

- Each time the spider moves in any direction, the fly randomly jumps
- The fly arrives at a new state but ..
  - The state it arrives in depends on where the fly jumped
  - Which depends on which direction the Spider moved
- The spider's action *modifies the state transition probabilities!!*

What is $P^{a}_{s,s'}$



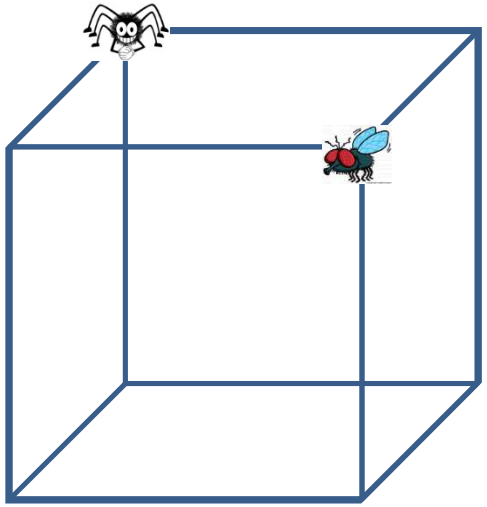


Full set of possible actions

Trick question.
Must modify our notion of states and actions,
and define the behavior of the fly, to characterize.

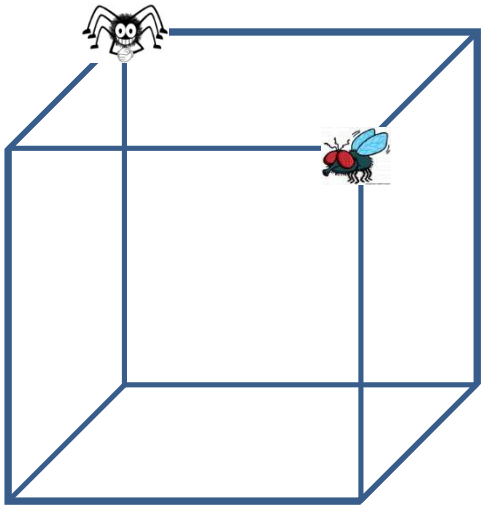
- Each time the spider moves in any direction, the fly randomly jumps
- The fly arrives at a new state but ..
  - The state it arrives in depends on where the fly jumped
  - Which depends on which direction the Spider moved
- The spider's action *modifies the state transition probabilities!!*

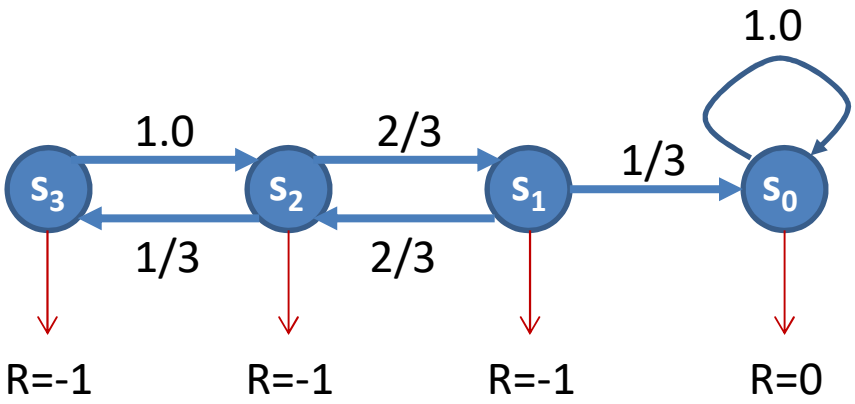
Trick Question: Redefining the States



- There are, in fact, only four states, not eight
  - Manhattan distance between fly and spider = 0 (s_0)
  - Distance between fly and spider = 1 (s_1)
  - Distance between fly and spider = 2 (s_2)
  - Distance between fly and spider = 3 (s_3)
- Can, in fact, redefine the MRP entirely in terms of these 4 states
- There are two actions a+ and a-
- Need an idea of the behavior of the fly

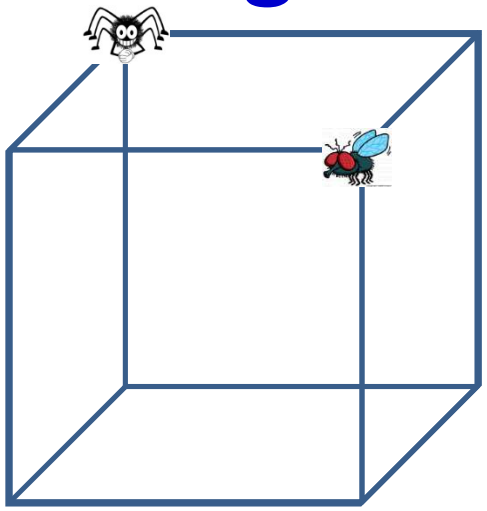
The Fly Markov Reward Process





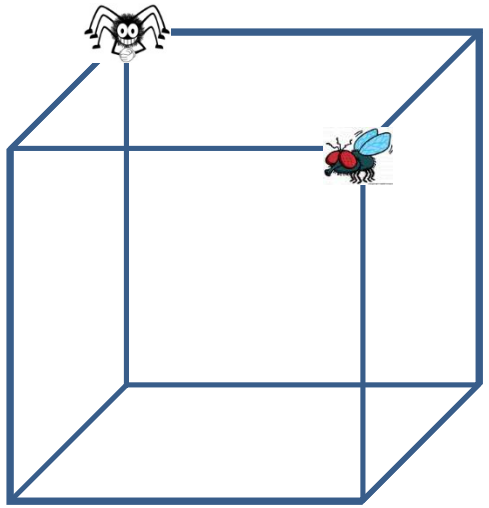
- There are, in fact, only four states, not eight
  - Manhattan distance between fly and spider = 0 (s_0)
  - Distance between fly and spider = 1 (s_1)
  - Distance between fly and spider = 2 (s_2)
  - Distance between fly and spider = 3 (s_3)
- Can, in fact, redefine the MRP entirely in terms of these 4 states

The Markov *Decision* Process: Defining Actions



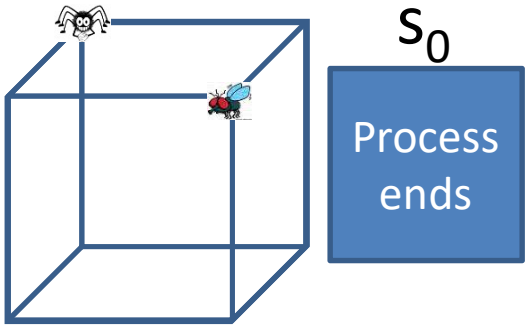
- Two types of actions:
  - a_+ : Increases distance to fly by 1
  - a_- : Decreases distance to fly by 1

The Fly Markov Decision Process



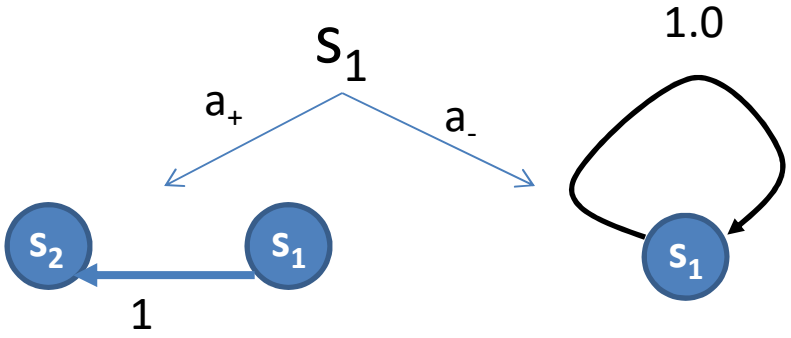
- The behavior of the fly:
  - If the spider is moving *away from it,* it does nothing
  - If the spider is moving *towards* it, it randomly hops to a different adjacent corner
    - 2/3 of the time, it increases the distance to the fly by 1
    - 1/3 of the time, it *decreases* the distance to the fly by 1

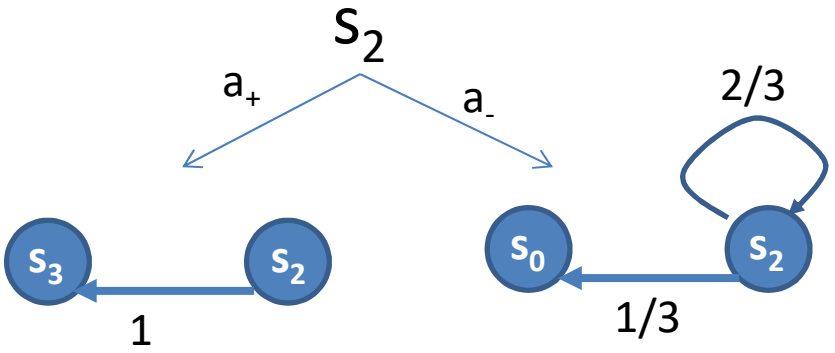
The Fly Markov Decision Process

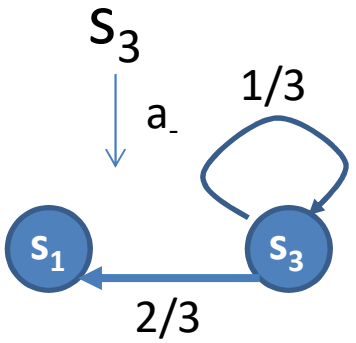


S_0

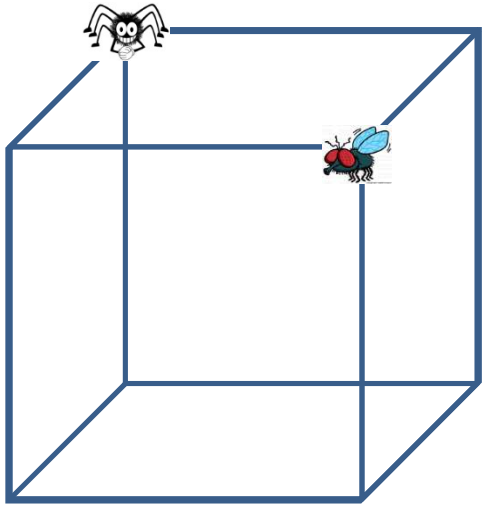
Process ends

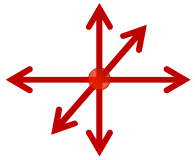






Redefining the problem





Full set of possible actions

- Each time the spider moves in any direction, the fly randomly jumps

Note: This is a simile for many problems in life, e.g. driving, stock market, advertising, etc.
The agents actions modifies how the environment behaves

  - Which depends on which direction the spider moved
- The spider's action *modifies the state transition probabilities!!*

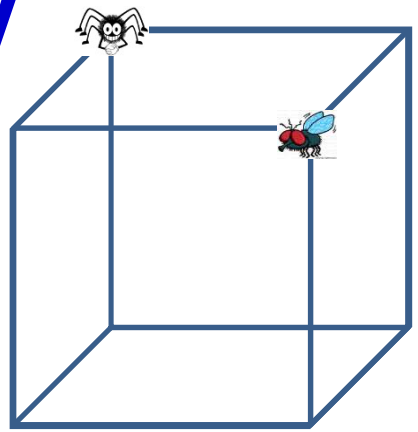
The Markov Decision Process

- A ***Markov Decision Process*** is a Markov Reward Process, where the agent has the ability to decide its actions!
  - We will represent individual actions as a
  - We will represent the action at time t as A_t
- The agent's actions affect the environment's behavior
  - The transitions made by the environment are functions of the action
  - The rewards returned are functions of the action

The Markov Decision Process

- Formally, a Markov Decision Process is the tuple $\mathrm{M} = \langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is a (possibly finite) set of states : $\mathcal{S} = \{s\}$
  - $\mathcal{A}$ is a (possibly finite) set of *actions* : $\mathcal{A} = \{a\}$
  - $\mathcal{P}$ is the set of *action conditioned* transition probabilities $P^{a}_{s,s'} = P(S_{t+1} = s | S_t = s', A_t = a)$
  - $\mathcal{R}$ is an *action conditioned reward* function
$$R^{a}_{s} = E[r | S = s, A = a]$$
  - $\gamma \in [0,1]$ is a *discount* factor

Introducing: Policy



- The *policy* is the probability distribution over actions that the agent may take at any state

$$\pi(a|s) = P(A_t = a|S_t = s)$$

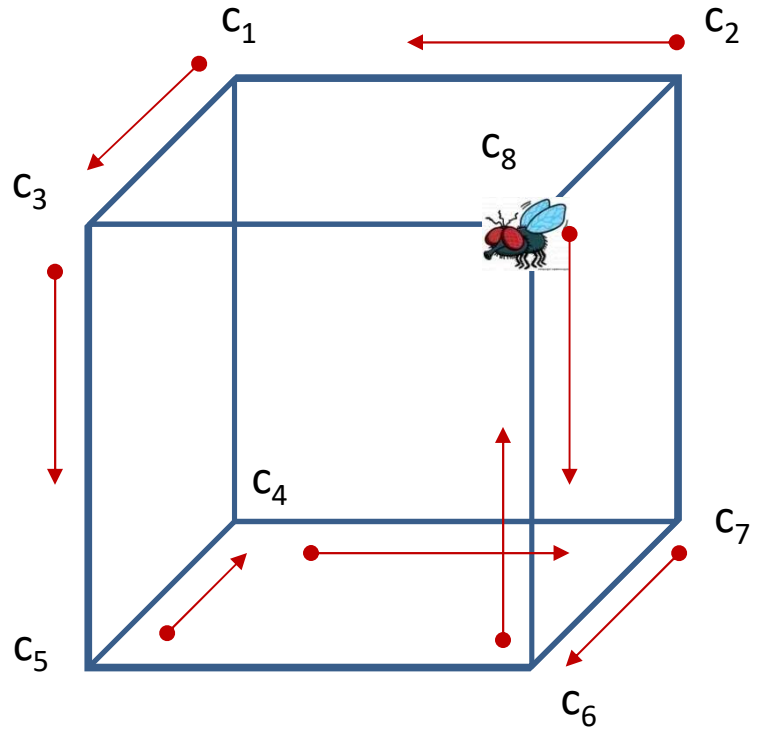
  - What are the preferred actions of the spider at any state

- The policy may be deterministic, i.e.

$$\pi(a|s) = 1 \; for \; a = \; a_s; \quad 0 \; for \; a \neq a_s$$

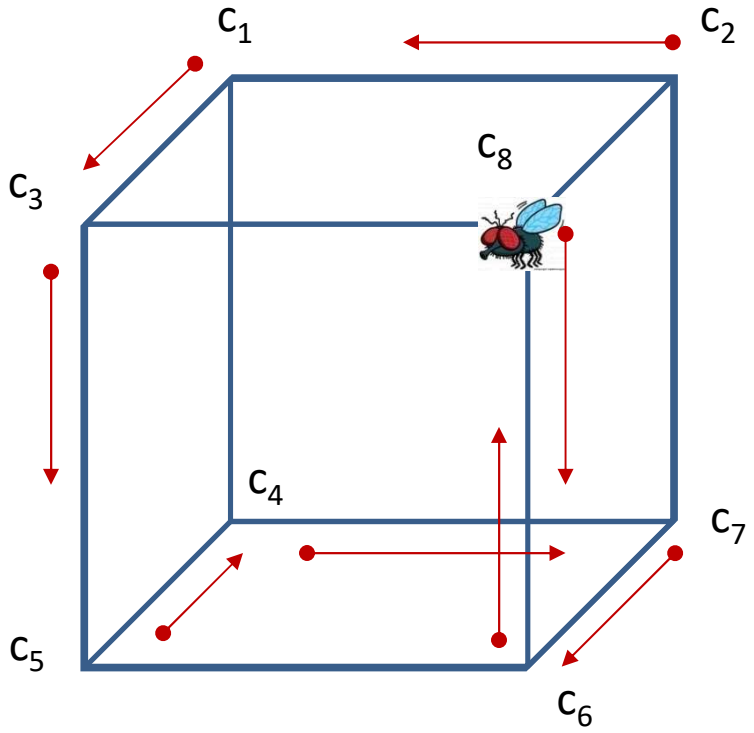
  - where a_s is the preferred action in state s

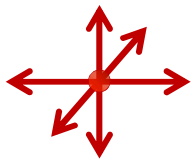
An example of a policy



- Assuming the fly does not move
  - This example is not a particularly good policy for the spider

An example of a policy

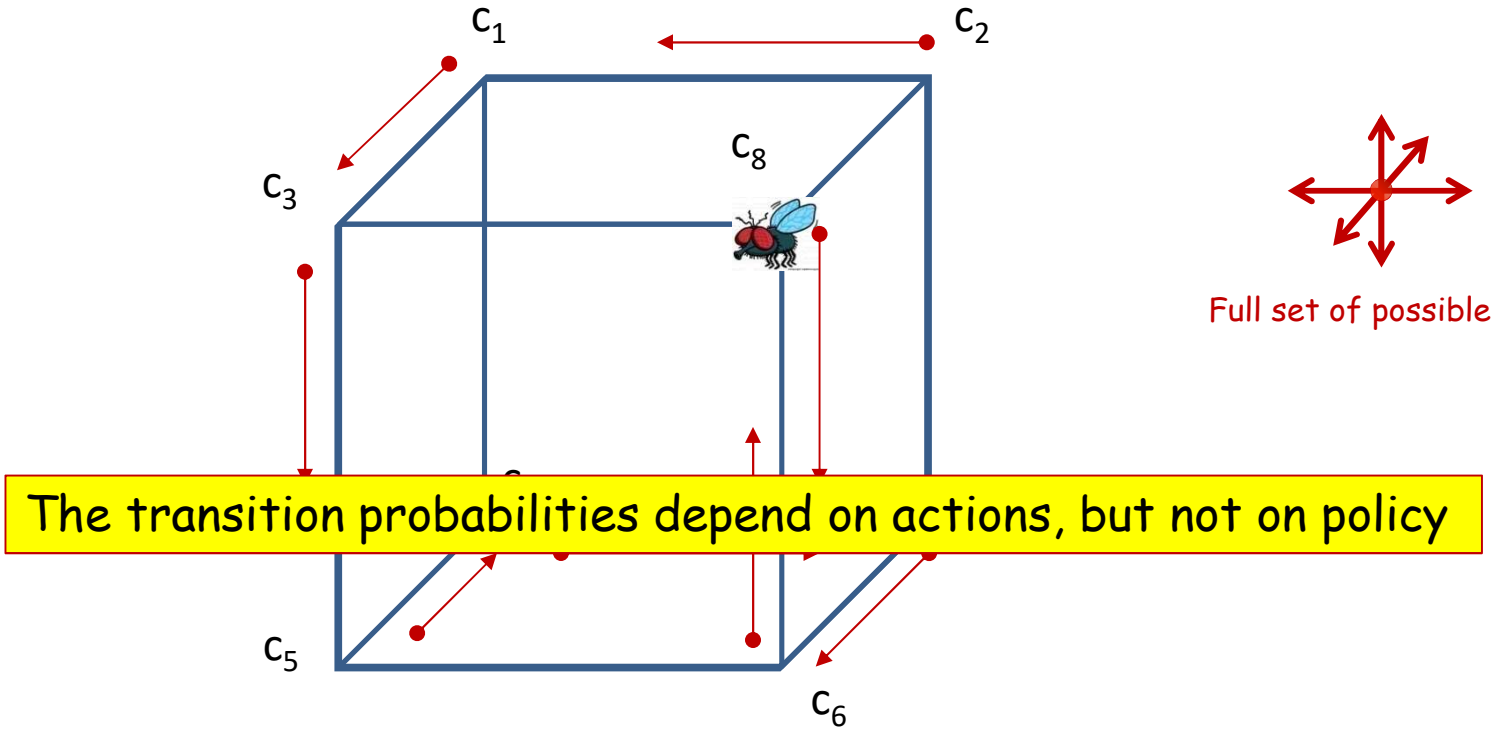




Full set of possible actions

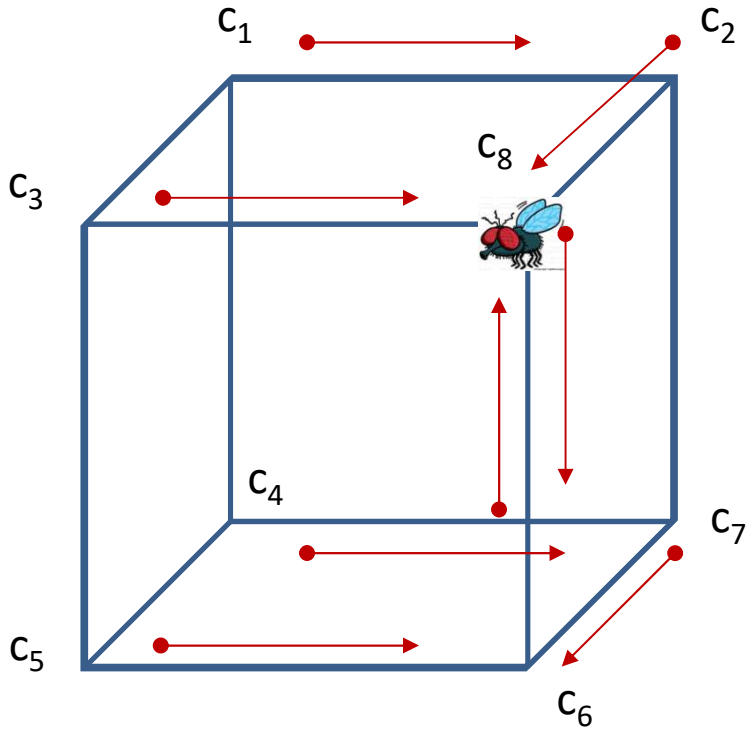
- What are the (action dependent) transition probabilities of the states here?

An example of a policy



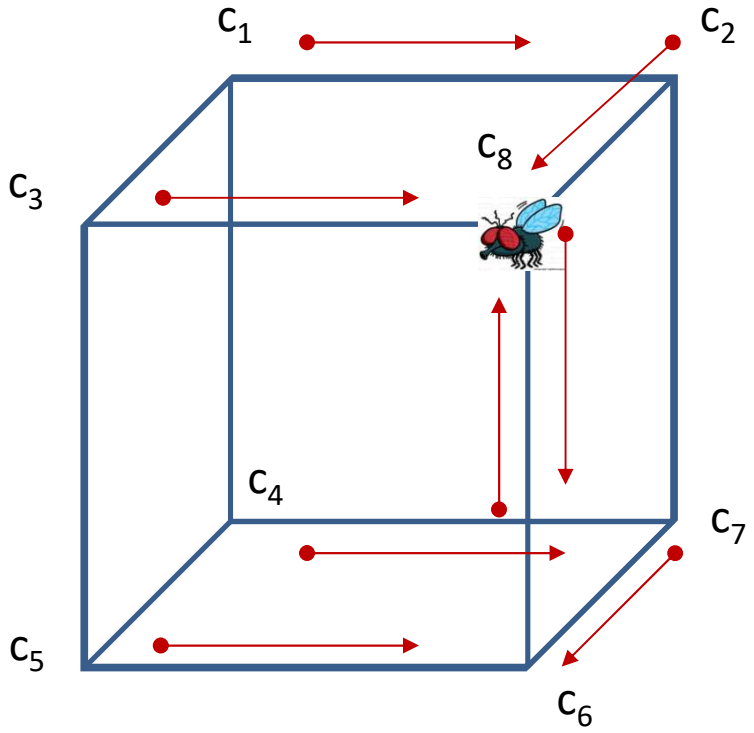
- What are the (action dependent) transition probabilities of the states here?

An example of a policy



- Assuming the fly does not move
  - This is a different *optimal* policy

An example of a policy

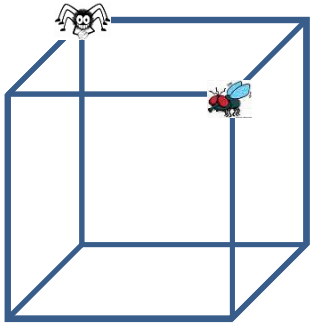


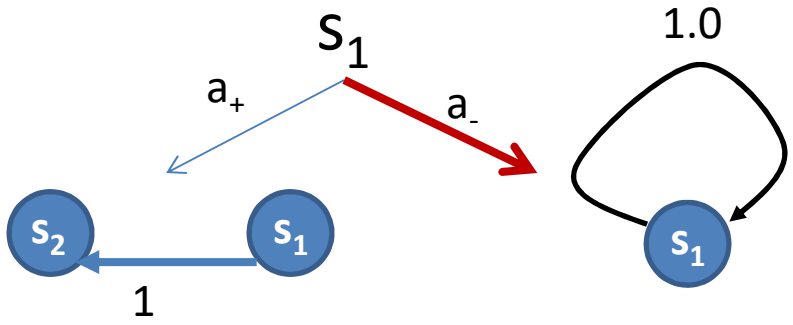
- Assuming the fly does not move
  - This is a different *optimal* policy
  - What are the transition probabilities here?

The value function of an MDP

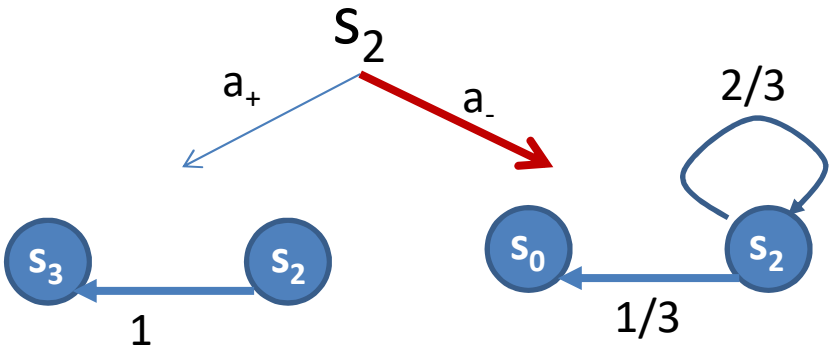
- **The *expected return* from any state depends on the policy you follow**

The Fly MDP: Policy 1

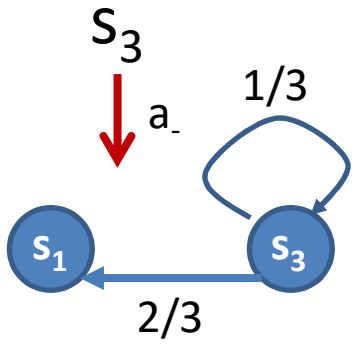




$$V_{s_1} = R_{s_1} + \gamma V_{s_1}$$

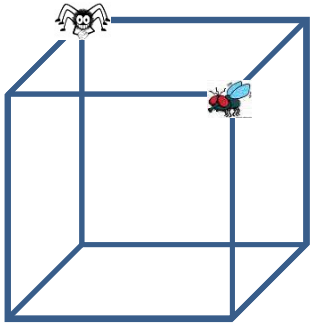


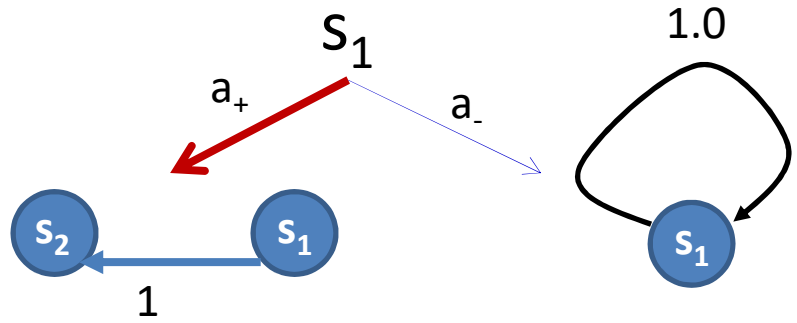
$$V_{s_2} = R_{s_2} + \gamma\left(\frac{1}{3}V_{s_0} + \frac{2}{3}V_{s_2}\right)$$



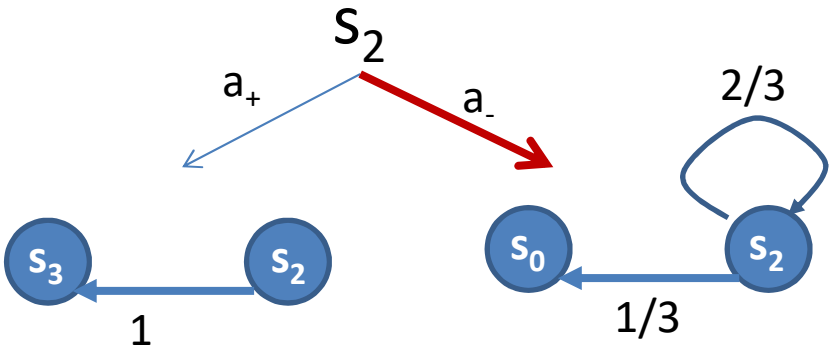
$$V_{s_3} = R_{s_3} + \gamma\left(\frac{2}{3}V_{s_1} + \frac{1}{3}V_{s_3}\right)$$

The Fly MDP: Policy 2 (optimal)

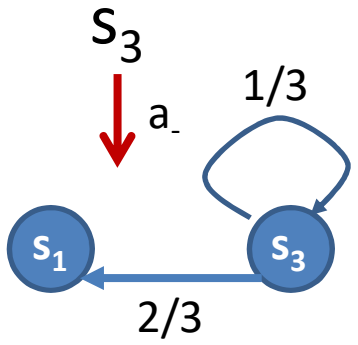




$$V_{s_1} = R_{s_1} + \gamma V_{s_2}$$

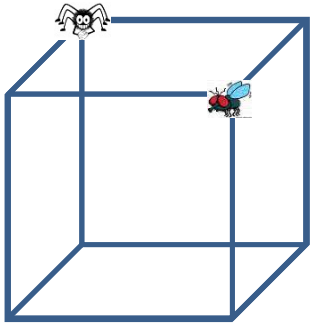


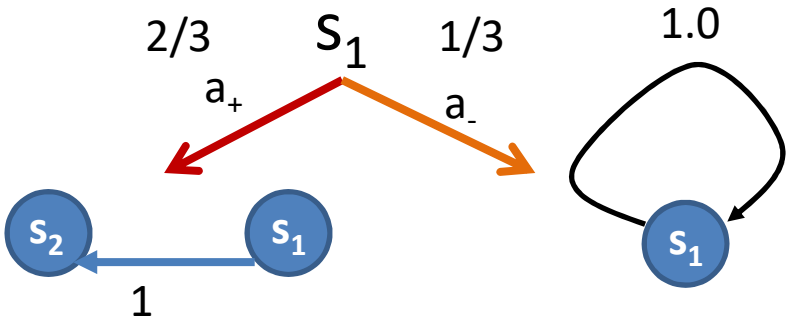
$$V_{s_2} = R_{s_2} + \gamma\left(\frac{1}{3}V_{s_0} + \frac{2}{3}V_{s_2}\right)$$

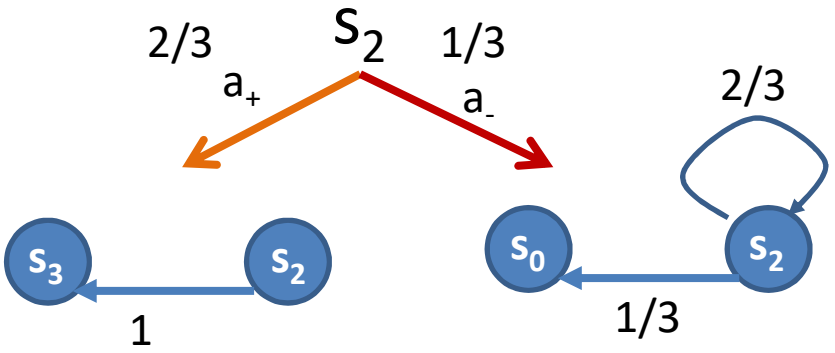


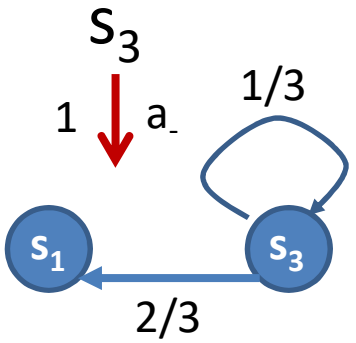
$$V_{s_3} = R_{s_3} + \gamma\left(\frac{2}{3}V_{s_1} + \frac{1}{3}V_{s_3}\right)$$

The Fly MDP: Stochastic Policy

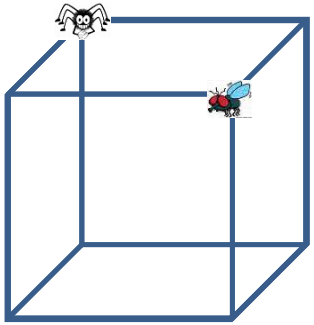


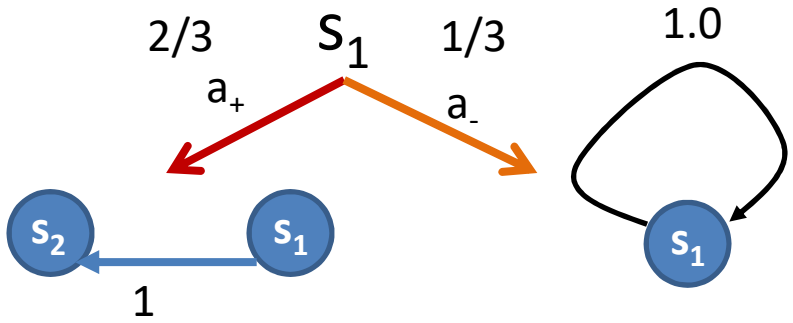




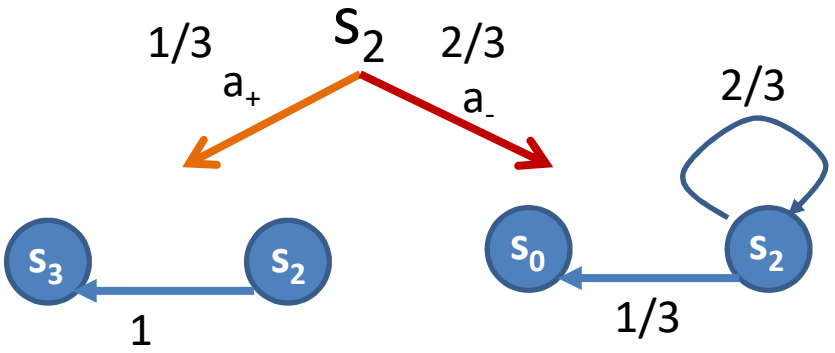


The Fly MDP: Stochastic Policy

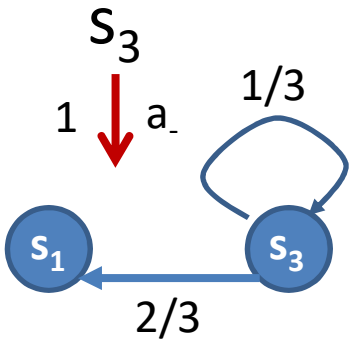




$$V_{s_1} = \frac{2}{3}\left(R_{s_1} + \gamma V_{s_2}\right) + \frac{1}{3}\left(R_{s_1} + \gamma V_{s_1}\right)$$



$$V_{s_2} = \frac{1}{3}\left(R_{s_2} + \gamma V_{s_3}\right) + \frac{2}{3}\left(R_{s_2} + \gamma\left(\frac{1}{3}V_{s_0} + \frac{2}{3}V_{s_2}\right)\right)$$



$$V_{s_3} = R_{s_3} + \gamma\left(\frac{2}{3}V_{s_1} + \frac{1}{3}V_{s_3}\right)$$

The *state value* function of an MDP

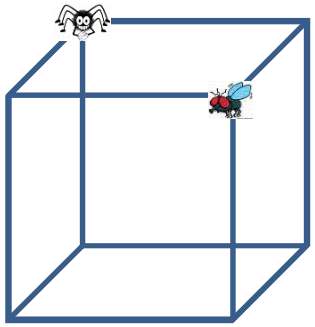
- The *expected return* from any state depends on the policy you follow
- We will index the value of any state by the policy to indicate this

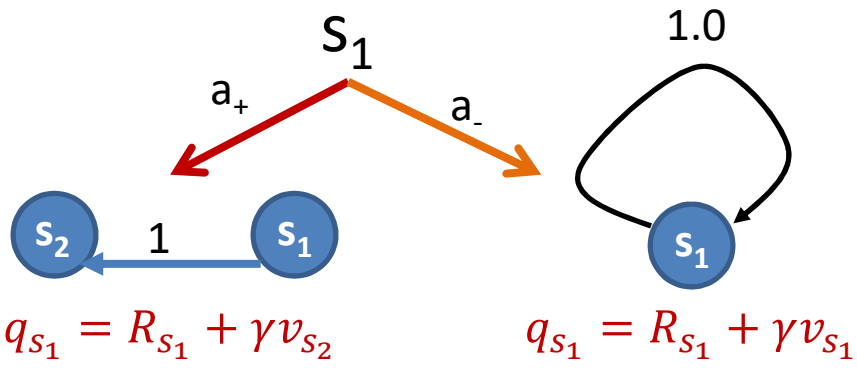
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

Bellman Expectation Equation for State Value Functions of an MDP

Note: Although reward was not dependent on action for the fly example, more generally it will be

The action value function of an MDP





- There are different value equations associated with different actions
- So we can actually associate value to ***state action pairs***
- **Note:** The LHS in the equation is the action-specific value at the source state, but the RHS is the overall value of the target states

The *action value* function of an MDP

- The *expected return* from any state under a given policy, when you follow a specific action

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s')$$

Bellman Expectation Equation for Action Value Functions of an MDP

All together now

- The Bellman expectation equation for state value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

- For action value function

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s')$$

- Giving you (obviously)

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

- And

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a \sum_{a' \in \mathcal{A}} \pi(a|s') q_\pi(s',a')$$

The Bellman Expectation Equations

- The Bellman expectation equation for state value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a v_\pi(s') \right)$$

- The Bellman expectation equation for action value function

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a \sum_{a\prime \in \mathcal{A}} \pi(a|s') q_\pi(s',a')$$

"Computing" the MDP

- Finding the state and/or action value functions for the MDP:
  - Given complete MDP (all transition probabilities $P^a_{s,s'}$, expected rewards R^a_s, and discount γ)
  - and a policy π
  - find all value terms $v_\pi(s)$ and/or $q_\pi(s,a)$

- The Bellman expectation equations are simultaneous equations that can be solved for the value functions
  - Although this will be computationally intractable for very large state spaces

Computing the MDP

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- Given the expected rewards at every state, the transition probability matrix, the discount factor and the policy:

$$\mathcal{V}_\pi = (\mathrm{I} - \gamma \mathcal{P}_\pi)^{-1} \mathcal{R}_\pi$$

- Matrix inversion $O(N^3)$; intractable for large state spaces

Optimal Policies

- Different policies can result in different value functions
- What is the *optimal* policy?

- The optimal policy is the policy that will maximize the expected total discounted reward at every state:
  $E[G_t|S_t = s]$

$$= E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|S_t = s\right]$$

Optimal Policies

- Different policies can result in different value functions
- What is the *optimal* policy?

- The optimal policy is the policy that will maximize the expected total discounted reward at every state: $E[G_t|S_t = s]$

$$= E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|S_t = s\right]$$

  - Recall: why do we consider the *discounted* return, rather than the actual return $\sum_{k=0}^{\infty} r_{t+k+1}$?

Policy Ordering Definition

- A policy π is "better" than a policy π' if the value function under π is greater than or equal to the value function under π' at all states

$$\pi \geq \pi' \Rightarrow v_{\pi}(s) \geq v_{\pi'}(s) \; \forall s$$

- Under the better policy, you will expect better overall outcome no matter what the current state

The optimal policy theorem

- **Theorem**: For any MDP there exists an optimal policy π_* that is better than or equal to every other policy:

$$\pi_* \geq \pi \quad \forall \pi$$

- **Corollary**: If there are *multiple* optimal policies π_{opt1}, π_{opt2}, … all of them achieve the same value function

$$v_{\pi_{opti}}(s) = v_*(s) \quad \forall s$$

- All optimal policies achieve the same action value function

$$q_{\pi_{opti}}(s, a) = q_*(s, a) \quad \forall s, a$$

How to find the optimal policy

- For the optimal policy:

$$\pi_*(a|s) = \begin{cases} 1 \;\; for \quad \underset{a'}{\operatorname{argmax}}\, q_*(s,a') \\ 0 \quad\quad otherwise \end{cases}$$

- Easy to prove
  - For any other policy π, $\quad q_\pi(s,a) \leq q_*(s,a)$

- Knowing the optimal action value function $q_*(s,a) \;\; \forall s,a$ is sufficient to find the optimal policy

The optimal value function

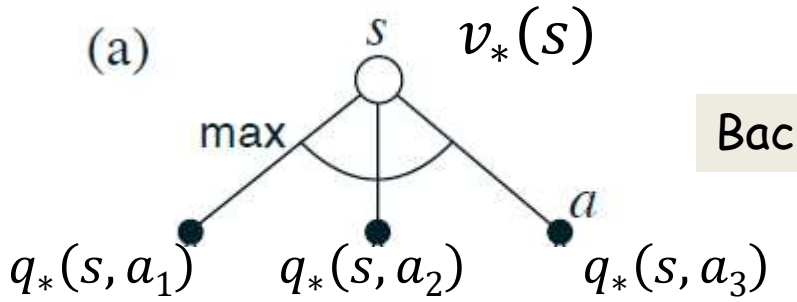
$$\pi_*(a|s) = \begin{cases} 1 \;\; for \quad \underset{a'}{\operatorname{argmax}}\, q_*(s, a') \\ 0 \quad\;\; otherwise \end{cases}$$

- Which gives us

$$v_*(s) = \max_a q_*(s, a)$$

Pictorially

Figures from Sutton



Backup Diagram

$$v_*(s) = \max_a q_*(s, a)$$

- Blank circles are states, filled dots are state-action pairs

The optimal value function

$$\pi_*(a|s) = \begin{cases} 1 \;\; for \quad \underset{a'}{\operatorname{argmax}}\, q_*(s,a') \\ 0 \quad\quad otherwise \end{cases}$$

- Which gives us

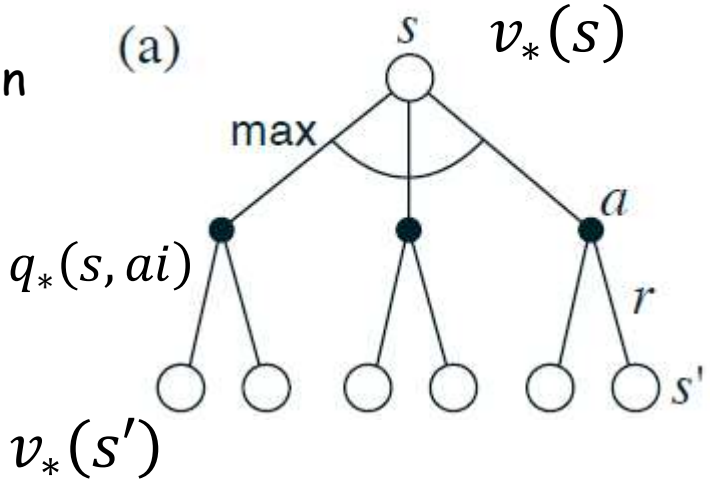
$$v_*(s) = \max_a q_*(s,a)$$

- But, for the optimal policy we also have

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

Backup Diagram

Figures from Sutton

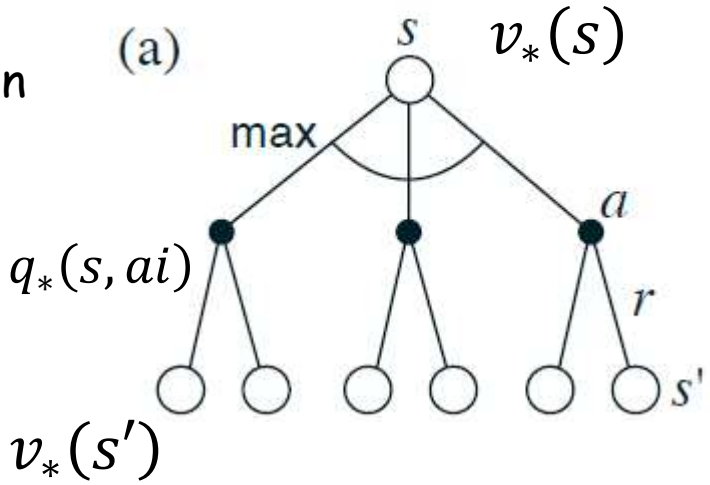


$$v_*(s) = \max_a q_*(s, a)$$

$$q_*(s, a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

Backup Diagram

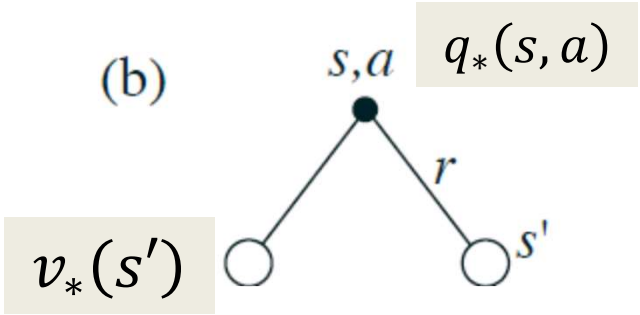
Figures from Sutton



$$v_*(s) = \max_a R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

Backup Diagram

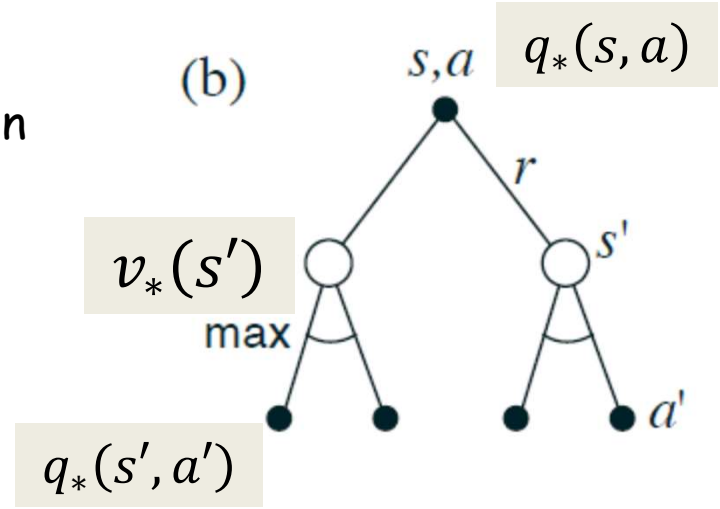
Figures from Sutton



$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

Backup Diagram

Figures from Sutton

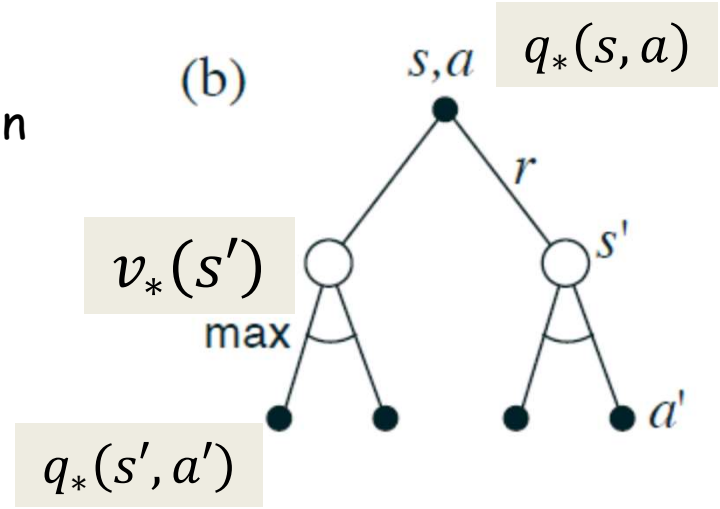


$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

$$v_*(s') = \max_{a'} q_*(s',a')$$

Backup Diagram

Figures from Sutton



$$q_*(s,a) = R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a \max_{a\prime} q_*(s',a')$$

Bellman *Optimality* Equations

- Optimal value function equation

$$v_*(s) = \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

- Optimal action value equation

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a \max_{a'} q_*(s',a')$$

Optimality Relationships

- Given the MDP: $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma \rangle$
- Given the optimal action value functions, the optimal value function can be found

$$v_*(s) = \max_a q_*(s,a)$$

- Given the optimal value function, the optimal action value function can be found

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s')$$

- Given the optimal action value function, the optimal policy can be found

$$\pi_*(a|s) = \begin{cases} 1 \;\; for \;\; \underset{a'}{\operatorname{argmax}}\, q_*(s,a') \\ 0 \;\;\;\; otherwise \end{cases}$$

"Solving" the MDP

- **Solving the MDP equates to finding the optimal policy** $\pi_*(a|s)$

- Which is equivalent to finding the optimal value function $v_*(s)$
- Or finding the optimal action value function $q_*(s, a)$

- Various solutions will estimate one or the other
  - Value based solutions solve for $v_*(s)$ and $q_*(s, a)$ and derive the optimal policy from them
  - Policy based solutions directly estimate $\pi_*(a|s)$

Solving the Bellman Optimality Equation

- No closed form solutions

- Solutions are iterative
- Given the MDP (Planning):
  - Value iterations
  - Policy iterations

- Not given the MDP (Reinforcement Learning):
  - Q-learning
  - SARSA..

QUESTIONS before we dive?

Planning with an MDP

- Problem:
  - **Given:** an MDP $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma \rangle$
  - **Find:** Optimal policy π_*

- Can either
  - **Value-based Solution:** Find optimal value (or action value) function, and derive policy from it OR
  - **Policy-based Solution:** Find optimal policy directly

Value-based Planning

- "Value"-based solution

- **Breakdown:**
  - **Prediction:** Given *any* policy π find value function $v_\pi(s)$
  - **Control:** Find the optimal policy

Value-based Planning

- "Value"-based solution

- **Breakdown:**
  - **Prediction:** Given *any* policy π find value function $v_\pi(s)$
  - **Control:** Find the optimal policy

Preliminaries

- How do we represent the value function?
- Table:
  - Value function
    - $s \rightarrow v_\pi(s)$
    - For a process with N discrete states, must store/compute N unique values
  - Action value functions
    - $s, a \rightarrow q_\pi(s, a)$
    - For a process with N discrete states and M discrete actions, must store/compute NM unique values
- Later we will see how to represent these when the number of states/actions is too large or continuous

The Bellman Expectation Equation for the value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

- In vector form

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix} = \begin{bmatrix} R_{s_1} \\ R_{s_2} \\ \vdots \\ R_{s_N} \end{bmatrix} + \gamma \begin{bmatrix} P_{s_1,s_1} & P_{s_2,s_1} & \cdots & P_{s_N,s_1} \\ P_{s_1,s_2} & P_{s_2,s_2} & \cdots & P_{s_N,s_2} \\ \vdots & \vdots & \ddots & \vdots \\ P_{s_1,s_N} & P_{s_2,s_N} & \cdots & P_{s_N,s_N} \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix}$$

- Where
  - $R_s = \sum_{a \in \mathcal{A}} \pi(a|s) R_s^a$
  - $P_{s',s} = \sum_{a \in \mathcal{A}} \pi(a|s) P_{s',s}^a$

The Bellman Expectation Equation for the value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

- In vector form

$$\begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix} = \begin{bmatrix} R_{s_1} \\ R_{s_2} \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} P_{s_1,s_1} & P_{s_2,s_1} & \cdots & P_{s_N,s_1} \\ P_{s_1,s_2} & P_{s_2,s_2} & \cdots & P_{s_N,s_2} \\ \vdots & \vdots & \ddots & \vdots \\ & & & P_{s_N,s_N} \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ v_\pi(s_2) \\ \vdots \\ v_\pi(s_N) \end{bmatrix}$$

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- Where
  - $R_s = \sum_{a \in \mathcal{A}} \pi(a|s) R_s^a$
  - $P_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s'} P_{s,s'}^a$

Solving the MDP

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- Given the expected rewards at every state, the transition probability matrix, the discount factor and the policy:

$$\mathcal{V}_\pi = (\mathrm{I} - \gamma \mathcal{P}_\pi)^{-1} \mathcal{R}_\pi$$

- Easy for processes with a small number of states
- Matrix inversion $O(N^3)$; intractable for large state spaces

What about the action value function?

- The Bellman expectation equation for action value function

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a \sum_{a' \in \mathcal{A}} \pi(a|s') q_\pi(s',a')$$

$$Q_\pi = \mathcal{R}_{\pi,Q} + \gamma \mathcal{P}_{\pi,Q} Q_\pi$$

$NM \times 1$ $NM \times 1$ $NM \times NM$ $NM \times 1$

Even worse!!

So how do we solve these

- The equations are too large, how do we solve them?

- First, a little lesson – from middle school…

What they never taught you in school

- Consider the following equation:

$$ax = b$$

- Where $0 < a < 2$
- Trivial solution: $x = a^{-1}b = \frac{b}{a}$

- But my CPU does not permit division..
  - How do I solve this?

What they never taught you in school

- Must solve the following without division

$$ax = b$$

  - where $0 < a < 2$
- Rewrite as follows

$$x = (1 - a)x + b$$

- The following iteration solves the problem:

$$x^{(k+1)} = (1 - a)x^{(k)} + b$$

- Can start with any $x^{(0)}$
- Proof??

What they never taught you in school

- Must solve the following without division

$$ax = b$$

  - where $0 < a < 2$
- Rewrite as follows

$$x = (1 - a)x + b$$

- The following iteration solves the problem:

$$x^{(k+1)} = (1 - a)x^{(k)} + b$$

- Can start with any $x^{(0)}$
- Proof?? Hint: $0 < a < 2 \;\Rightarrow\; |1 - a| < 1$

What they never taught you in school

- Consider any vector equation

$$\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{b}$$

  - Where all Eigen values $|\lambda(\mathbf{A})| \leq 1$
    - And some extra criteria…
      - The square submatrix of $(\mathbf{I} - \mathbf{A})$ corresponding to non-zero entries of $\mathbf{b}$ is full rank
      - The square submatrix of $(\mathbf{I} - \mathbf{A})$ corresponding to zero entries of $\mathbf{b}$ is an identity matrix

- The following iteration solves the problem:

$$\mathbf{x}^{(k+1)} = \mathbf{A}\mathbf{x}^{(k)} + \mathbf{b}$$

Eigen values of a probability matrix

- For any Markov transition probability matrix $\mathcal{P}$, all Eigenvalues have magnitude less than or equal to 1

$$|\lambda(\mathcal{P})| \leq 1$$

Solving for the value function

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- This can be solved by following iteration starting from any initial vector

$$\mathcal{V}_\pi^{(k+1)} = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi^{(k)}$$

Solving for the value function

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- This can be solved by following iteration starting from any initial vector

$$\mathcal{V}_\pi^{(k+1)} = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi^{(k)}$$

- But how did that help if we need infinite iterations to converge?

Solving for the value function

$$\mathcal{V}_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi$$

- This can be solved by following iteration starting from any initial vector

$$\mathcal{V}_\pi^{(k+1)} = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi \mathcal{V}_\pi^{(k)}$$

- But how did that help if we need infinite iterations to converge?
  - Solution: Stop when the changes becomes small

$$\left| \mathcal{V}_\pi^{(k+1)} - \mathcal{V}_\pi^{(k+1)} \right| < \varepsilon$$

Actual Implementation

- Initialize $v_\pi^{(0)}(s)$ for all states
- Update

$$v_\pi^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi^{(k)}(s') \right)$$

- Update may be in *batch* mode
  - Keep sweep through all states to compute $v_\pi^{(k+1)}(s)$
  - Update $k = k + 1$
- Or incremental
  - Sweep through all the states performing

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi(s') \right)$$

Actual Implementation

- Initialize $v_\pi^{(0)}(s)$ for all states
- Update

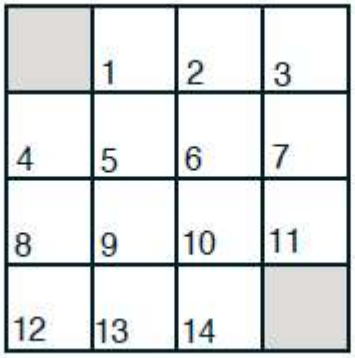
$$v_\pi^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_\pi^{(k)}(s') \right)$$

- This is an instance of *dynamic programming:*
- **dynamic programming** (also known as **dynamic optimization**) is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions. The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in storage space. (Each of the subproblem solutions is indexed in some way, typically based on the values of its input parameters, so as to facilitate its lookup.) (from wikipedia)

An Example

Example from Sutton

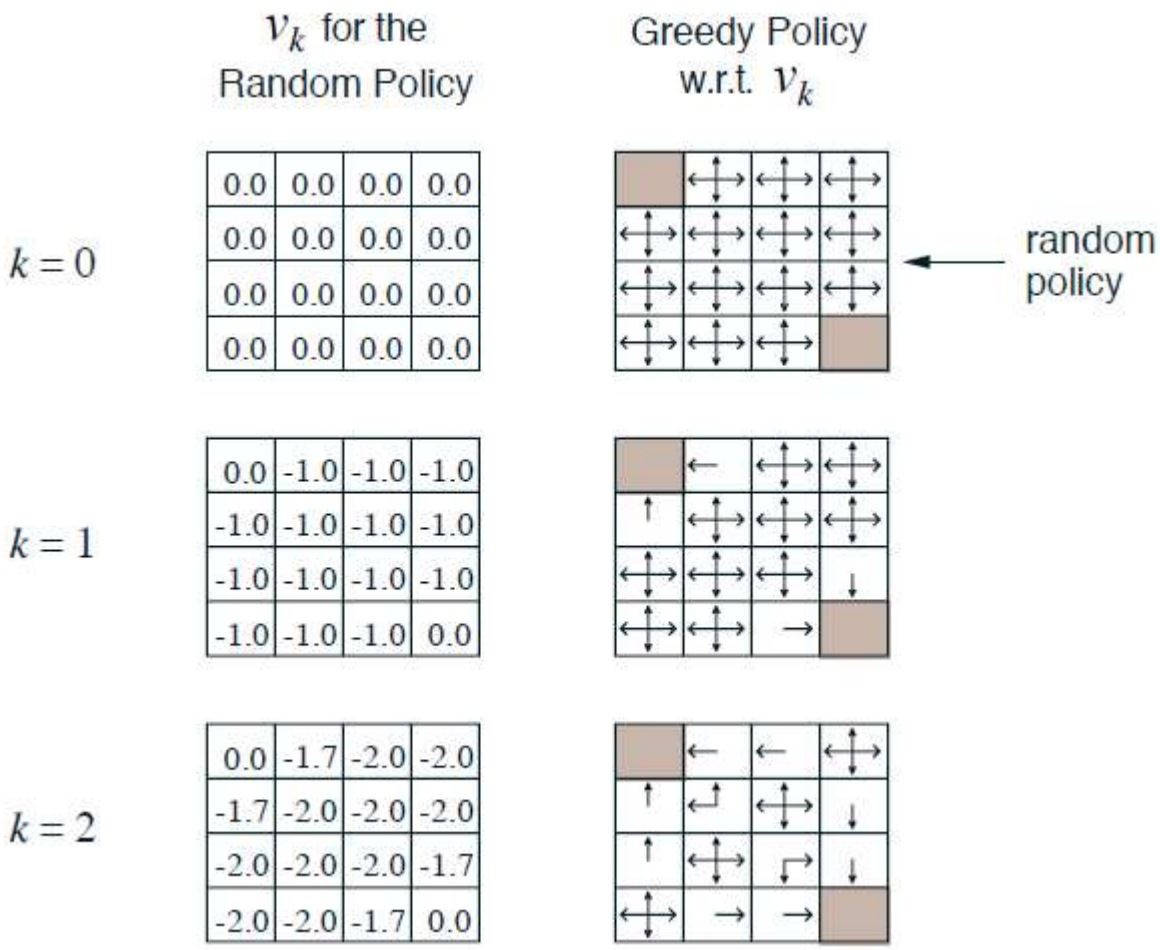
actions



$R = -1$
on all transitions

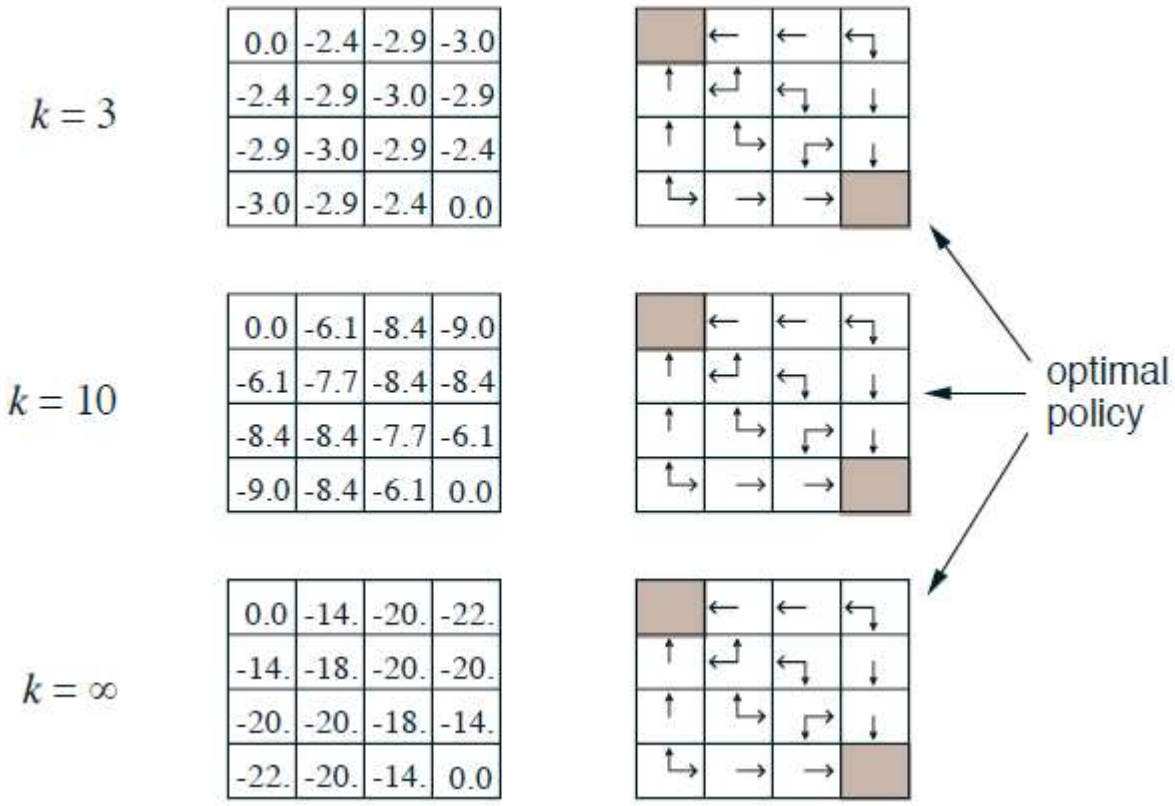
- All squares, except shaded square have reward -1, shaded square has reward 0
- **Policy:** Random – can step in any of the four directions with equal probability
  - If you run into a wall, you just return to the square
- Find the value of being in each square

The Gridworld Example



- Actual iterations use random policy
- Right column shows greedy policy according to current value function

The Gridworld Example



- Iterations use random policy
- Greedy policy converges to optimal long before value function of random policy converges!

Value-based Planning

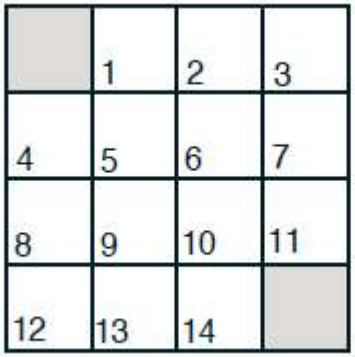
- "Value"-based solution

- **Breakdown:**
  - **Prediction:** Given *any* policy π find value function $v_\pi(s)$
  - **Control:** Find the optimal policy

Revisit the gridworld

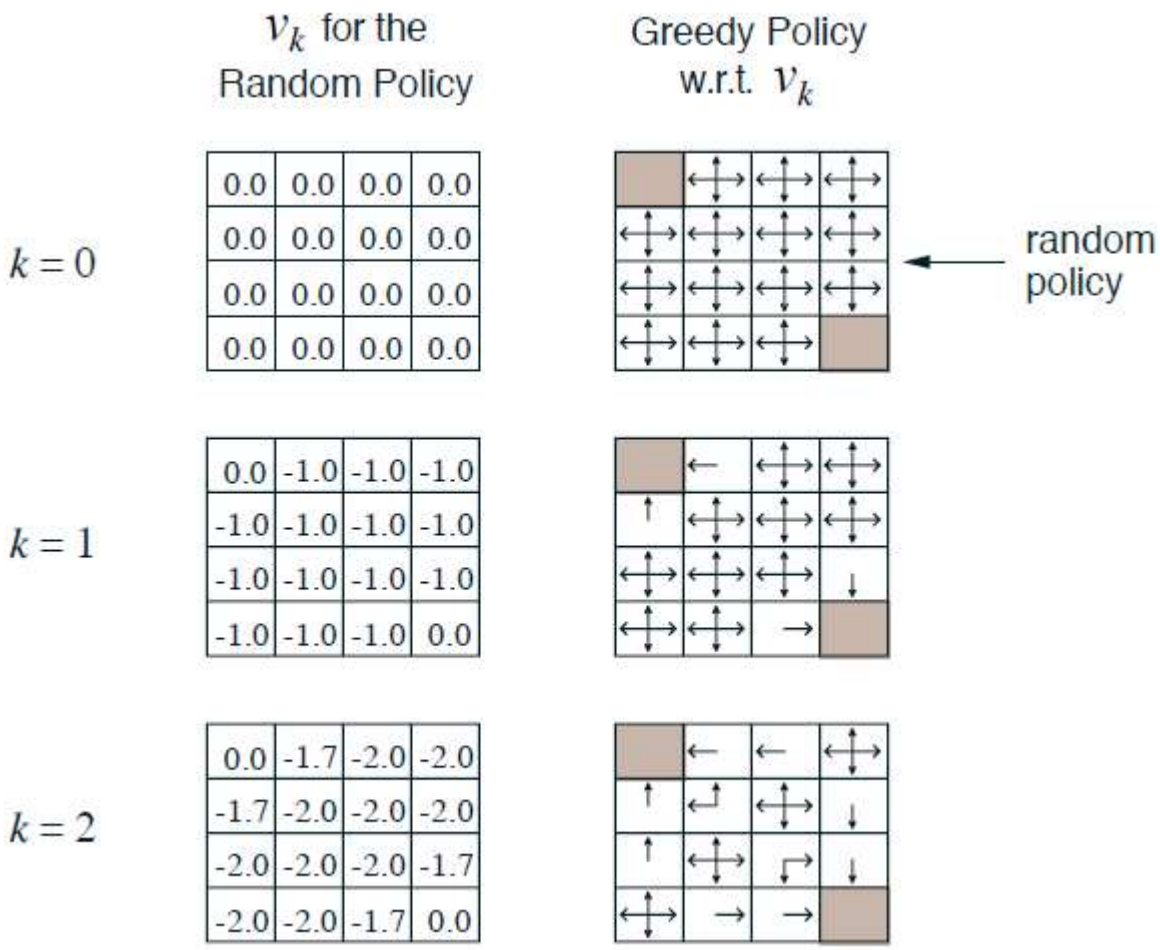
Example from Sutton

actions



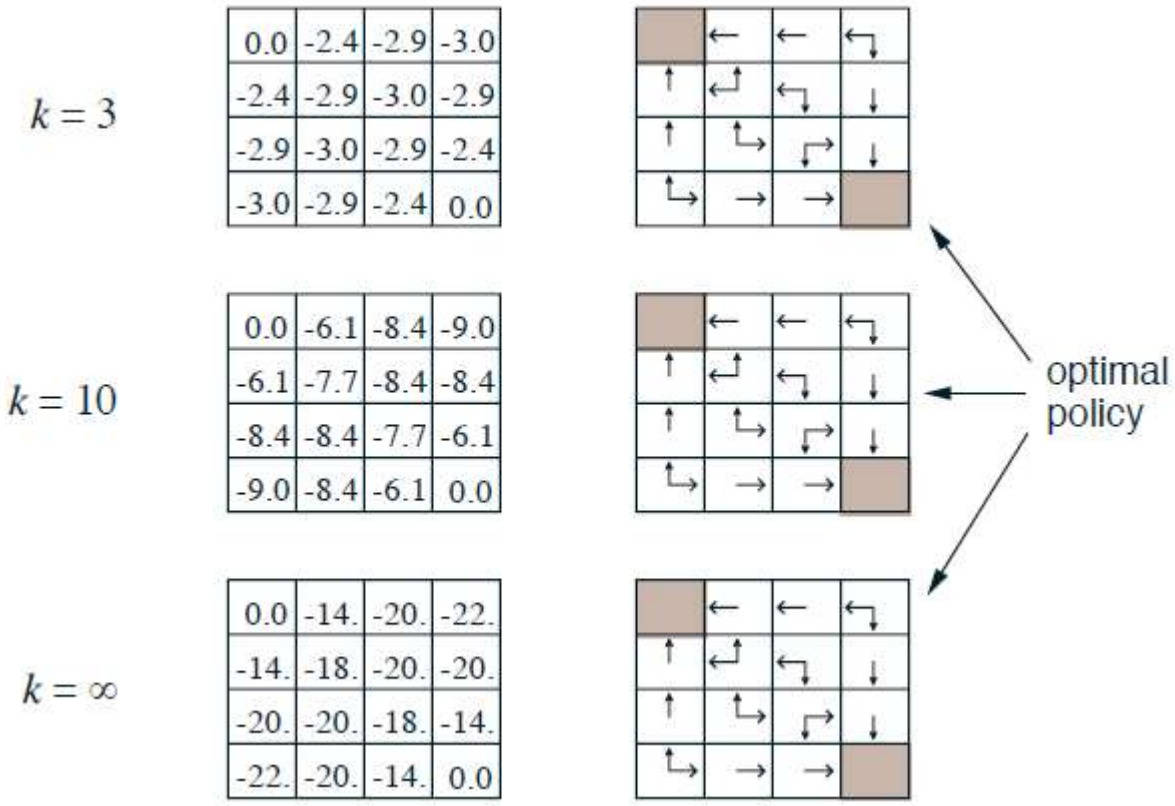
$R = -1$
on all transitions

Revisit the gridworld



- Actual iterations use random policy
- Right column shows greedy policy according to current value function

Revisit the gridworld



- Iterations use random policy
- Greedy policy converges to optimal long before value function of random policy converges!

Finding an optimal policy

- Start with any policy, e.g. random policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Compute action value function $\forall s, a$:

$$q_{\pi^{(k)}}(s,a) = R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a v_{\pi^{(k)}}(s')$$

  - Find the greedy policy

$$\pi^{(k+1)}(a|s) = \begin{cases} 1 \;\; for\; a = \underset{a\prime}{\text{argmax}} \;\; q_{\pi^{(k)}}(s,a') \\ 0 \qquad\qquad otherwise \end{cases}$$

Finding an optimal policy: Compact

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(a|s) = \begin{cases} 1 \;\; for\; a = \underset{a'}{\operatorname{argmax}} \;\; R_s^{a'} + \gamma \sum_{s'} P_{s,s'}^{a'} v_{\pi^{(k)}}(s') \\ 0 \qquad\qquad otherwise \end{cases}$$

Finding an optimal policy: Shorthand

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(s) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

THIS IS KNOWN AS **POLICY ITERATION**
In each iteration, we find a policy, and then find its value

Policy Iteration

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

- This will provably converge to the optimal policy π_*
- In the Gridworld example this converged in one iteration
- More generally, it will take several iterations
  - Convergence when policy no longer changes

Generalized Policy Iteration

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use *any algorithm* to find the value function $v_{\pi^{(k)}}(s)$
  - Use *any algorithm* to find an update policy

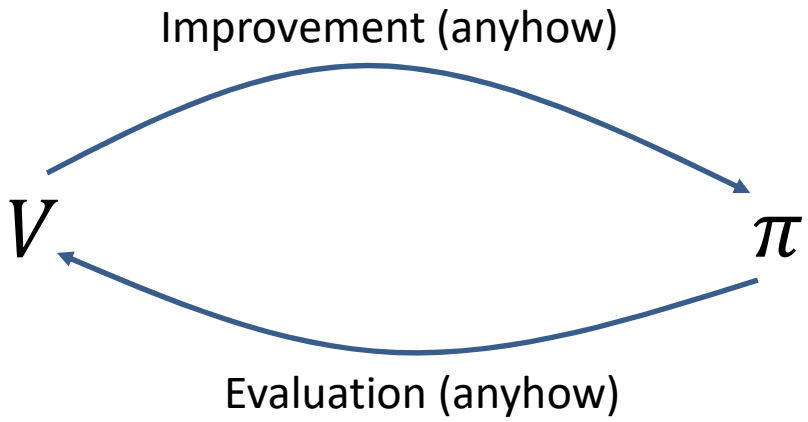
$$\pi^{(k+1)}(\mathrm{s}) = algorithm\left(v_{\pi^{(k)}}(s)\right)$$

Such that $\pi^{(k+1)}(\mathrm{s}) \geq \pi^{(k)}(\mathrm{s})$

- Guaranteed to converge to the optimal policy

Generalized Policy Iteration

- Start with any policy $\pi^{(0)}$



- Guaranteed to converge to the optimal policy

Optimality theorem

- *All* states will hit their optimal value together

- **Theorem:**

  A policy $\pi(a|s)$ has optimal value

  $$v_\pi(s) = v_*(s)$$

  in any state s if and only if for *every* state s' reachable from s,

  $$v_\pi(s') = v_*(s')$$

Policy Iteration

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

- This will provably converge to the optimal policy π_*
- In the Gridworld example this converged in one iteration
- More generally, it will take several iterations
  - Convergence when policy no longer changes

Policy Iteration

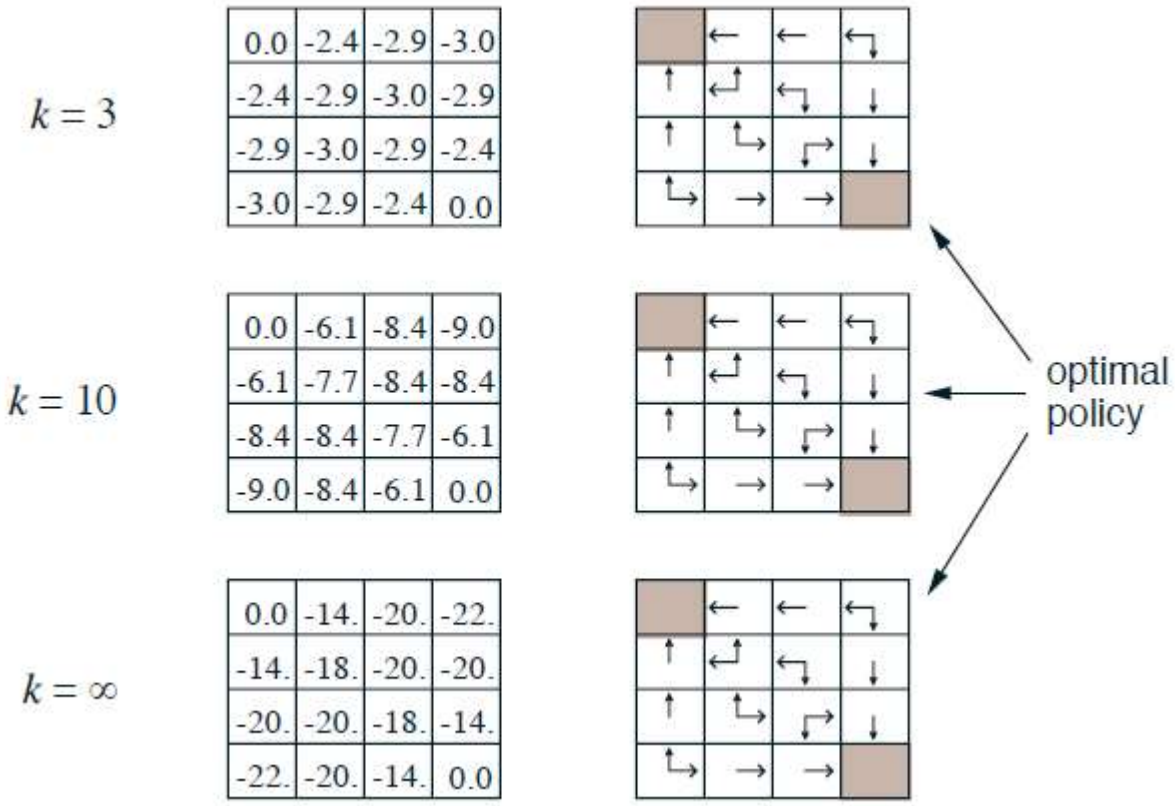
- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

In the gridworld example we didn't even need to run *this* to convergence

The optimal policy was found long before the actual value function converged even in the first upper iteration

- This will provably converge to the optimal policy π_*
- In the Gridworld example this converged in one iteration
- More generally, it will take several iterations
  - Convergence when policy no longer changes

Revisit the gridworld



- Iterations use random policy
- Greedy policy converges to optimal long before value function of random policy converges!

Policy Iteration

- Start with any policy $\pi^{(0)}$
- Iterate ($k = 0$ … convergence):
  - Use prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(s) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

In the gridworld example we didn't even need to run *this* to convergence

The optimal policy was found long before the actual value function converged even in the first upper iteration

Do we even need the prediction DP to converge?

- More generally, it will take several iterations
  - Convergence when policy no longer changes

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use L iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

- This will provably converge to the optimal policy π_*

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$
  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = greedy\left(v_{\pi^{(k)}}(s)\right)$$

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

- Iterate ($k = 0$ … convergence):
  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$

$$v_{\pi^{(k)}}(s) = \sum_{a \in \mathcal{A}} \pi^{(k)}(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s') \right)$$

  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = \underset{a}{\operatorname{argmax}} \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s')$$

Optimal policy estimation

- Start with any policy $\pi^{(0)}$

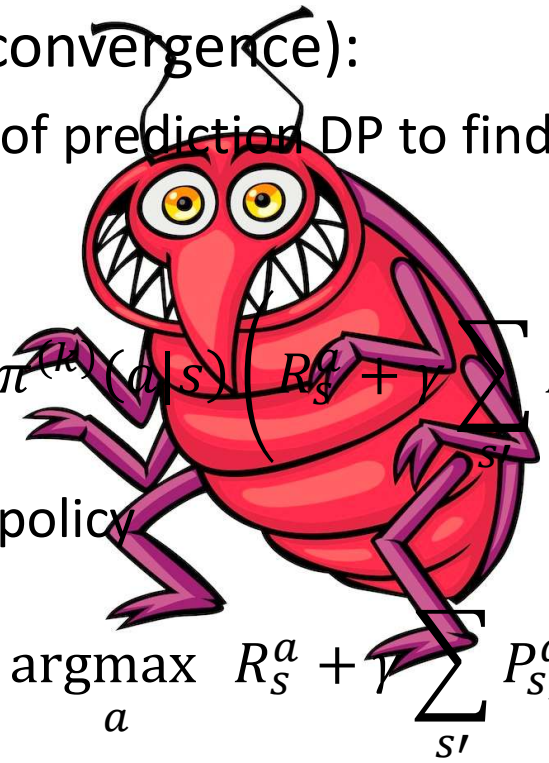
- Iterate ($k = 0$ … convergence):
  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$

BUG

$$v_{\pi^{(k)}}(s) = \sum_{a \in \mathcal{A}} \pi^{(k)}(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s') \right)$$

  - Find the greedy policy

$$\pi^{(k+1)}(\mathrm{s}) = \operatorname*{argmax}_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k)}}(s')$$



Reordering and writing carefully

- Start with any initial value function $v_{\pi^{(0)}}(s)$
- Iterate ($k = 1$ … convergence):
  - Find the greedy policy

$$\pi^{(k)}(a|s) = \begin{cases} 1 \;\; for\; a = \underset{a'}{\text{argmax}} \;\; R_s^{a'} + \gamma \sum_{s'} P_{s,s'}^{a'} v_{\pi^{(k-1)}}(s') \\ 0 \qquad otherwise \end{cases}$$

  - Use 1 iterations of prediction DP to find the value function $v_{\pi^{(k)}}(s)$

$$v_{\pi^{(k)}}(s) = \sum_{a \in \mathcal{A}} \pi^{(k)}(a|s) \left(R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_{\pi^{(k-1)}}(s') \right)$$

Merging

- Start with any initial value function $v_{\pi(0)}(s)$
- Iterate ($k = 1$ … convergence):
  - Update the value function

$$v_{\pi(k)}(s) = \max_a R_s^a + \gamma \sum_{s\prime} P_{s,s\prime}^a v_{\pi(k-1)}(s')$$

- Note: no explicit policy estimation
  - Directly learns value
  - The subscript π is a misnomer

Value Iteration

- Start with any initial value function $v_*^{(0)}(s)$

- Iterate ($k = 1$ … convergence):
  - Update the value function

$$v_*^{(k)}(s) = \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*^{(k-1)}(s')$$

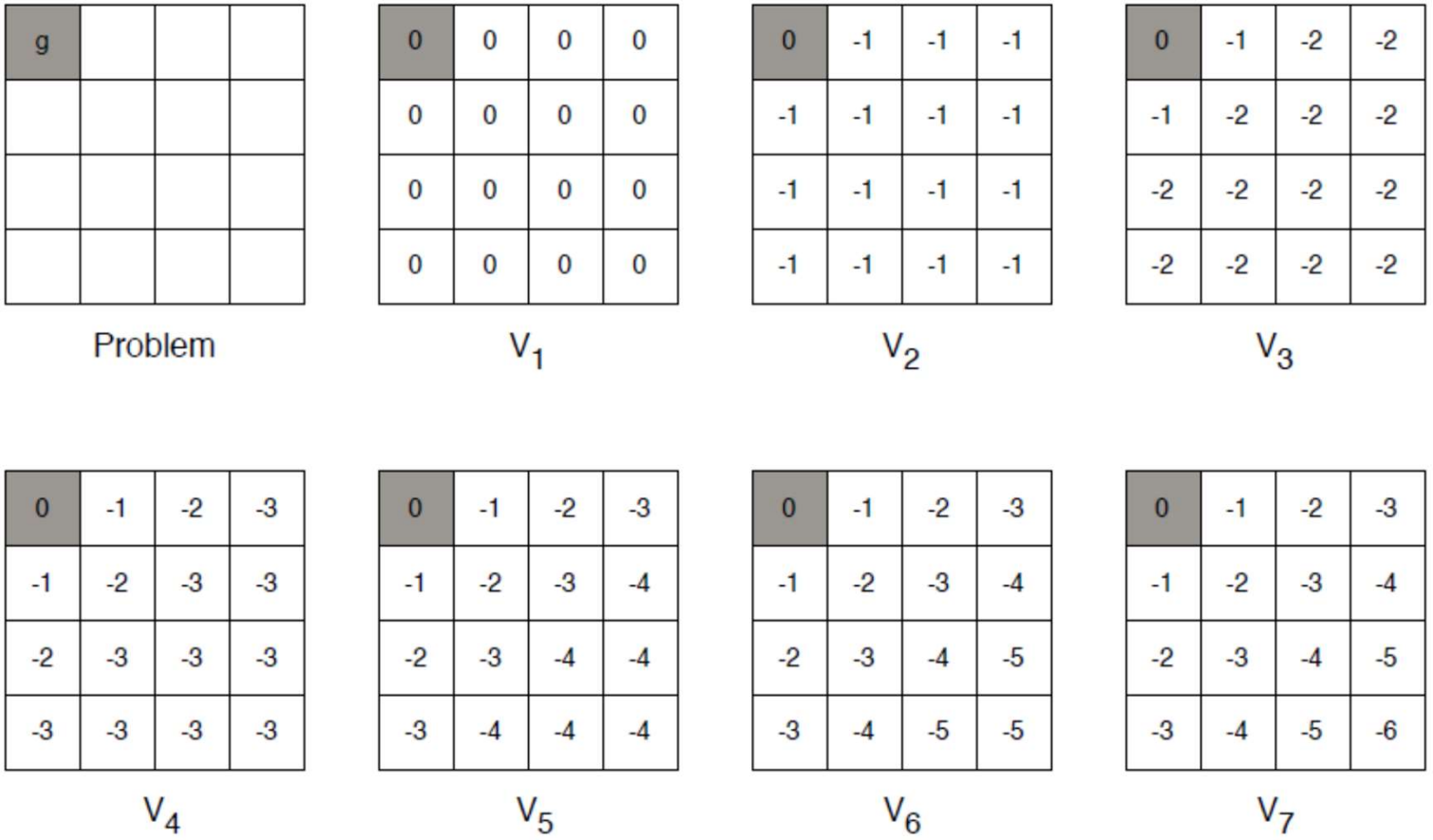
- Note: no explicit policy estimation
- Directly learning *optimal* value function
- Guaranteed to give you optimal value function at convergence
  - But intermediate value function estimates may not represent any policy

Value iteration

$$v_*^{(k)}(s) = \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*^{(k-1)}(s')$$

- Each state simply inherits the cost of its best neighbour state
  - Cost of neighbor is the value of the neighbour plus cost of getting there

Value Iteration Example



- Target: Find the shortest path
- Every step costs -1

Practical Issues

- Updates can be batch mode
  - Explicitly compute $v_*^{(k+1)}(s)$ from $v_*^{(k)}(s)$for all states
  - Set k = k+1

- Or asynchronous
  - Compute $v_*(s)$ in place while we sweep over states
  - $v_*(s) \leftarrow \max_a \; R_s^a + \gamma \sum_{s'} P_{s,s'}^a v_*(s)$

Recap

- Learned about *prediction*
  - Estimating value function given MDP and policy
- Learned *Policy* iteration
  - Iterate prediction and policy estimation
- Learned about *Value* iteration
  - Directly estimate optimal value function

Alternate strategy

- Worked with *Value function*
  - For N states, estimates N terms
- Could alternately work with *action-value function*
  - For M actions, must estimate MN terms
    - Much more expensive
    - But more useful in some scenarios

Next Up

- We've worked so far with planning
  - Someone gave us the MDP
- Next: Reinforcement Learning
  - MDP unknown..