# CS4710: Artificial Intelligence Reasoning Under Uncertainty

Part 2: Hidden Markov Models, Bayesian Model Tracing, and a few other applications of uncertainty

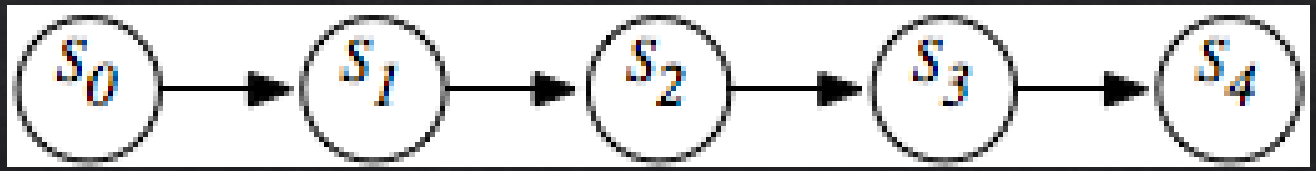
# Topics

- Markov Chains
- Markov Decision Processes
- Partially observable Markov Decision Processes

# Markov Chains

# What is a Markov Chain?

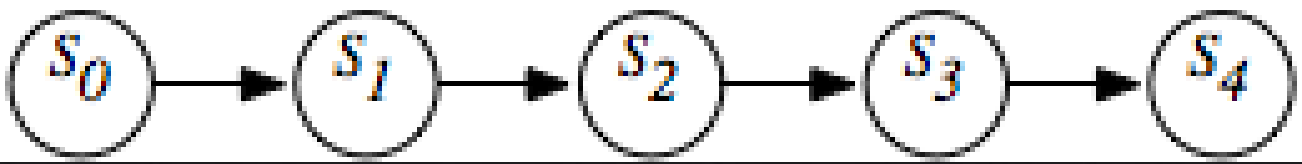
A **Markov chain** is a special sort of belief network used to represent sequences of values, such as the sequence of states in a dynamic system or the sequence of words in a sentence.



# What is a Markov Chain?

To define a **Markov Chain**, we define:

- A set of states ***S***
- For each time step t >= 0
  - $P(S_{t+1} \mid S_t)$ is probability you will move to some new state
- $P(S_0)$ specifies the initial state(s) of the system



# The Markov Assumption

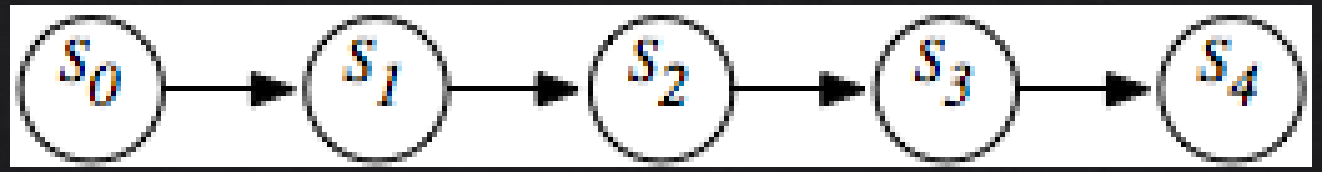
Markov Chains adhere to the Markov Assumption.

Namely:

For all t >= 0:

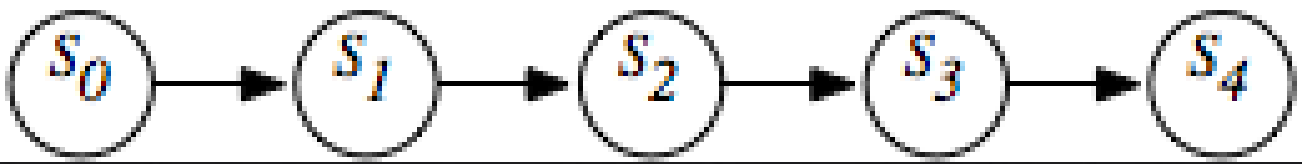
$$P(S_{t+1} \mid S_0, S_1, \ldots, S_t) = P(S_{t+1} \mid S_t)$$

i.e., The next state depends ONLY on the current state and not the path you took to get to the current state.

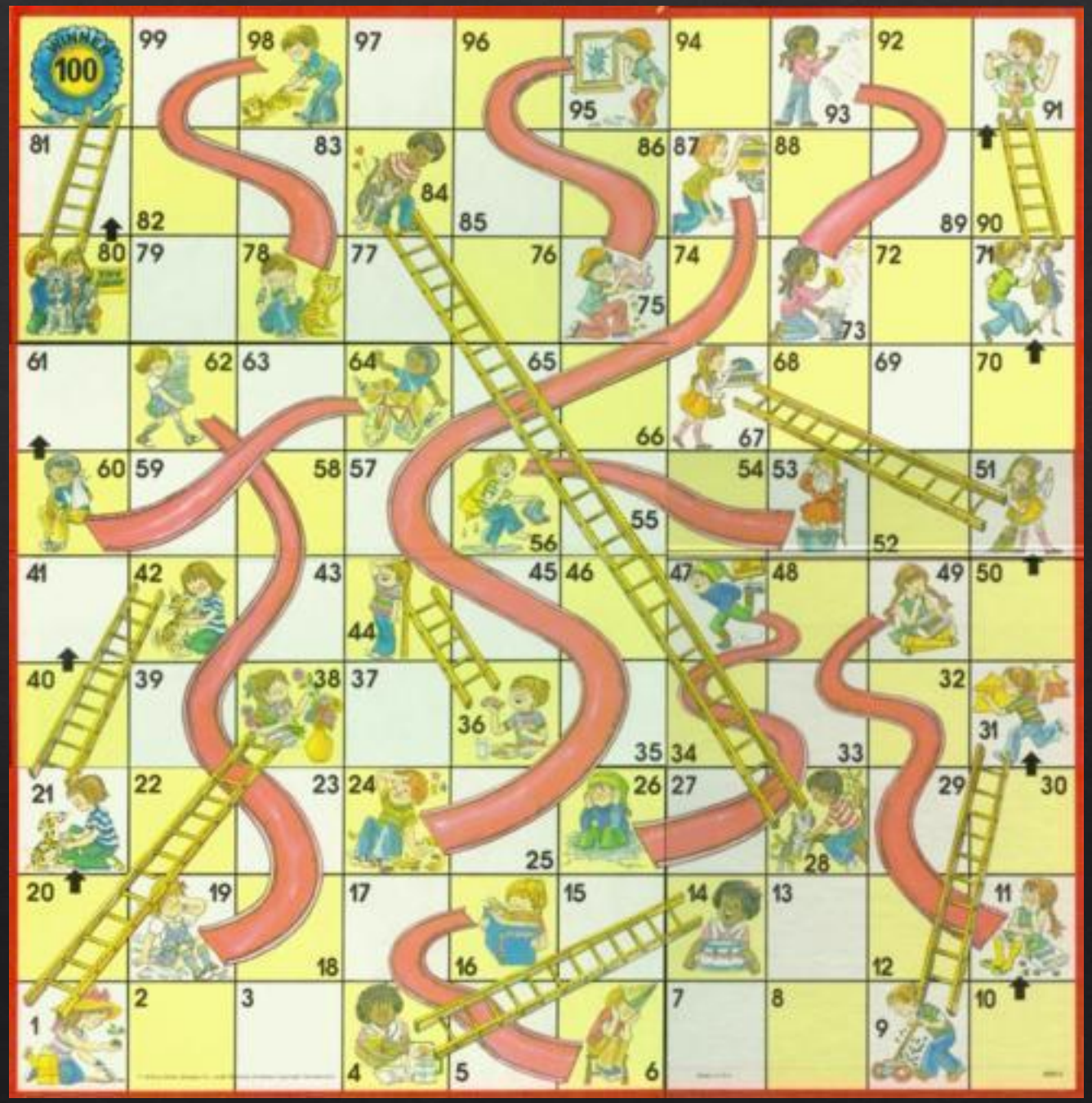


# Advantages

- Simple model that is easy to specify
- Assumption of stationarity (probabilities don't change as a function of t) is often natural. If probabilities do change, you can often incorporate this somehow.
- Network can extend indefinitely.



# Markov Chains Example: Chutes and Ladders ☺


WINNER
100
99 98 97 96 95 94 93 92 91
81 82 83 84 85 86 87 88 89 90
80 79 78 77 76 75 74 73 72 71
61 62 63 64 65 66 67 68 69 70
60 59 58 57 56 55 54 53 52 51
41 42 43 44 45 46 47 48 49 50
40 39 38 37 36 35 34 33 32 31
21 22 23 24 25 26 27 28 29 30
20 19 18 17 16 15 14 13 12 11
1 2 3 4 5 6 7 8 9 10

# How the Game is Played

- Chutes and Ladders is a board game where players spin a pointer to determine how they will advance
- The board consists of 100 numbered squares
- The objective is to land on square 100
- The spin of the pointer determines how many squares the player will advance at his turn with equal probability of advancing from 1 to 6 squares

# How the Game is Played

- However, the board is filled with chutes, which move a player backward if landed on. There are also ladders, which advance a player.
- Chutes have pictures of bad behavior which leads to disasters
- Ladders have pictures of good behavior leading to rewards
- Most of the chutes and ladders produce relatively small changes in position, but several produce large gains or losses

# Markov Chains

- Transition matrix
  - How we represent the chain
- Probability vectors
  - How we represent likelihoods of being in various states over time
- Absorbing vs. non-absorbing Markov chains
- Steady-state matrices

# Transition Matrix

Below, is the layout of the transition matrix for Chutes and Ladders (101x101)

| | | | |
|---|---|---|---|
| p0,0 | p1,0 | ..... | p100,0 |
| p0,1 | p1,1 | ..... | p100,1 |
| . | . | | . |
| . | . | | . |
| p0,100 | p1, 100 | ..... | p100,100 |

# Three properties:

- 1) The probability of moving from state i to j is independent of what happened before moving to state j and how one got to state i (Markov assumption)

- 2) Sum of probabilities for each state must be one

- 3) X(t) = probability distribution vector of the probability of the system being in each of the states at time n

# Probability Vector

- The probability vector is a column vector in which the entries are nonnegative and add up to one

- The entries can represent the probabilities of finding a system in each of the states

- So, the starting X for chutes and ladders is:

```
X = [   1.0        //100 percent chance we start in state 1
        0.0
        0.0
        .
        .     ]
```

# Two types of Markov Chains

1) Absorbing Markov Chains
   - Can not get out of certain states
   - Once a system enters an absorbing state, the system remains in that state from then on

2) Non-absorbing Markov Chains
   - Can always get out of every state

What type is Chutes and Ladders?

# Absorbing Markov Chains (Chutes and Ladders)

- Two conditions must be met for a Markov Chain to be absorbing:
  - Must have at least one state which cannot be left once it has been entered
  - It must be possible, through a series of one or more moves, to reach at least one absorbing state from every non-absorbing state (given enough time, every subject will eventually be trapped in an absorbing state)

# Common Question

- A common question arising in Markov-chain models is, what is the long-term probability that the system will be in each state?
- The vector containing these long-term probabilities is called the steady-state vector of the Markov chain

# Steady-state matrix

- The steady-state probabilities are average probabilities that the system will be in a certain state after a large number of transition periods
- The convergence of the steady-state matrix is independent of the initial distribution
- Long-term probabilities of being on certain squares

# The Chutes and Ladders Transition Matrix

# Techniques of Transition Matrix

- How to do analytic computations
  - 2D array of integers of size 101 by 101
  - Each array entry represents a move from one square to another
    - i.e. Pij represents the players move from position i to position j
    - A square with a ladder beginning in that square is considered a pseudo-state and has a 0 probability of landing on it
    - A square with a chute beginning in that square is considered a pseudo-state and also has a 0 probability of landing on it

# Results of Transition Matrix

- The matrix is way too large to show on this slide
- Some example probabilities computed:
  - If a player is on square 97, the probability of staying on that square is 1/2 and the probability of moving to square 78 is 1/6

# Techniques of Probability Vectors

- How we could do the analytic computations
  - 1D array of integers of size 101
  - Probability vectors Vn represent the probability of being on a certain square on move n
  - Vo = {1,0,…0} and means that the probability of being on square 0 is 1 or 100%
  - Vo * P = V1;
  - V1 * P = V2, etc.

# Results of Transition Matrix

- After 1000 moves, the probability vector reached a limit of {0,0,…1}
- This means that after 1000 moves, the game is expected to be won!!!

# Hidden Markov Model (HMM)

# Hidden Markov Models

- A **<u>hidden markov model</u>** is like a markov chain except that the underlying state the system is in is "hidden"
- By "hidden" we mean that we don't know what state the system is in at any one time
- In Chutes and Ladders, we knew what state we were in (what space we are on) but the movements between states was random (rolling the dice)
- I want you to know these exist, but we aren't going to study them in this class

# Markov Decision Processes (MDP)

# Motivation

Agents:

- Observe the world and react to what they see
- Can make decisions about what actions to take given a situation

Need something that models:

- States the agent or the world (or both) can be in
- Actions available
- How these actions affect the environment

Markov Chains do some of these things, but we need an extension to the model

# Markov Decision Processes

A **Markov Decision Process (MDP)** is a model that contains:

- A set of possible world states ***S***
- A set of possible actions ***A***
- A real valued reward function ***R(s,a)***
- A description ***T*** of each action's effects in each state

We assume the **Markov Property**: The effects of an action taken in a state depend only on that state and not on the prior history.

# Representing Actions

Depending on problem you are trying to solve, you may want to model:

Deterministic Actions:

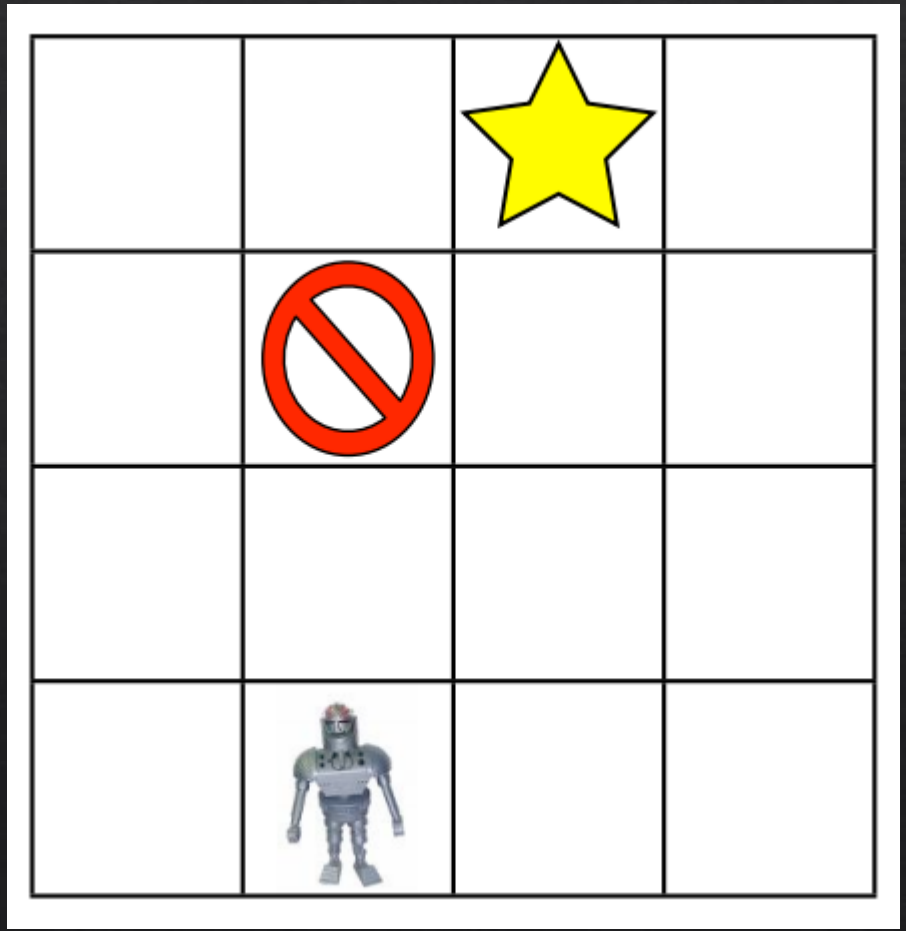
T: (S x A) → S For each state and action, move to a new state (100%)

Stochastic Actions:

T: (S x A) → Prob(S) For each state and action, specify a prob distribution over next states. Represented as P(s' | s, a)

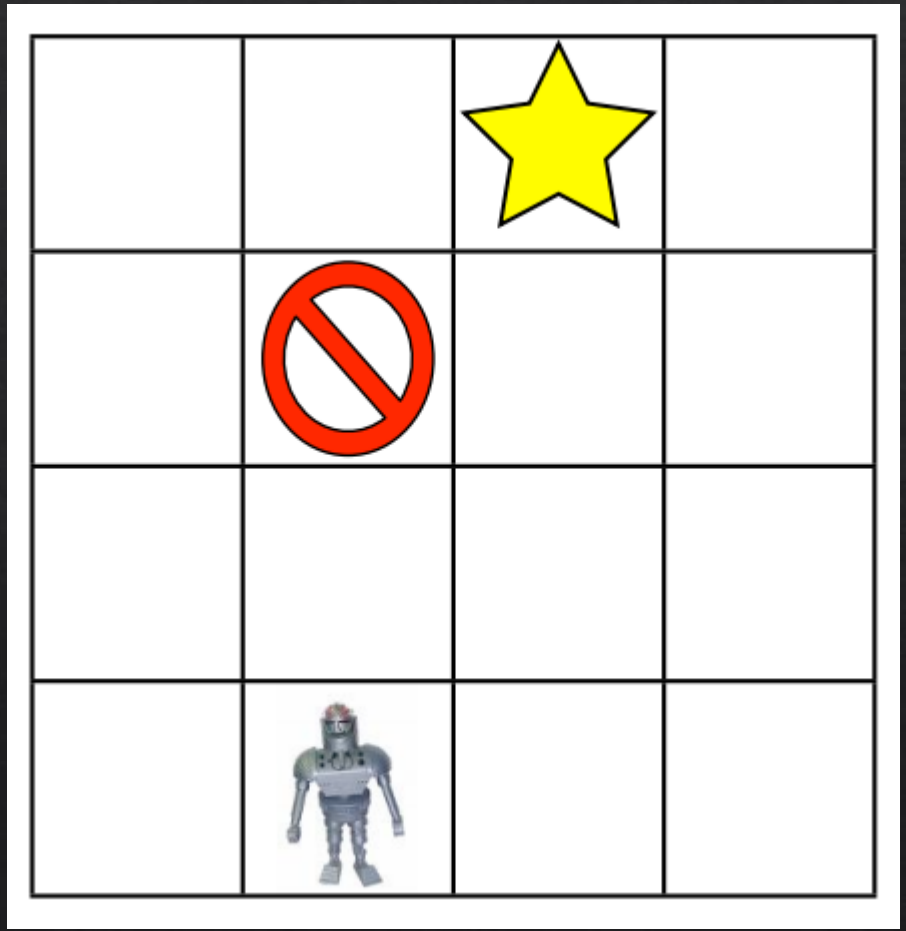
# Example MDP: Robot

- Robot needs to reach some goal and avoid special "traps"
  - E.g., stairs and such
- Actions are to move up, down, left, right
- Actions may fail or have surprising effects
  - There might be a wall in some location
  - We might not be certain where these walls are located



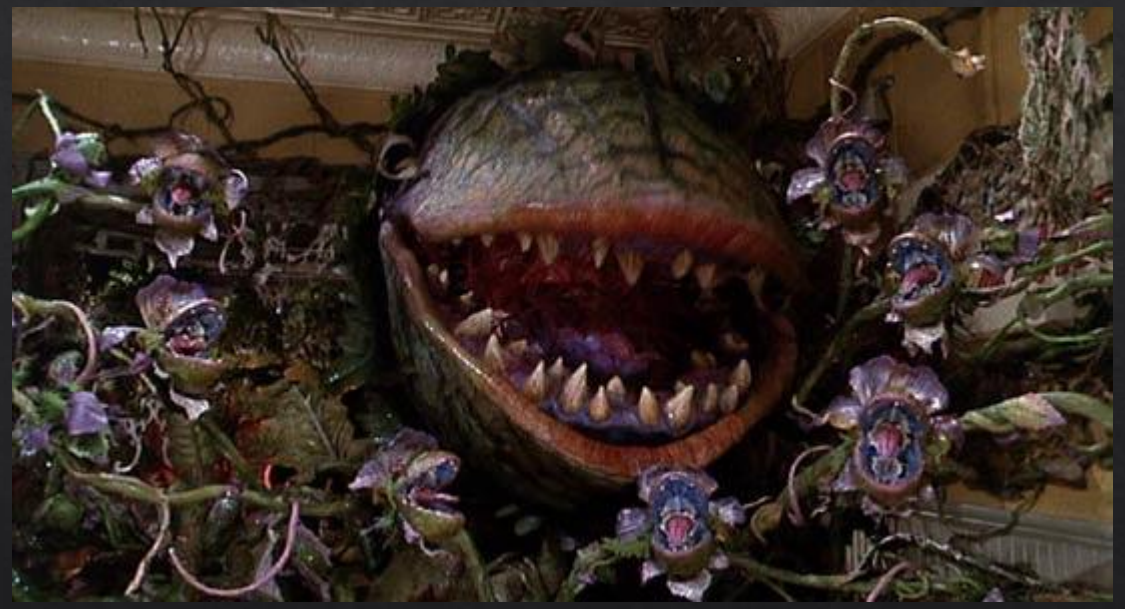
# Example MDP: Robot

- States: specify robot's location
- Actions: Move in one direction
- Reward: 1 if at the goal, -1 if you hit a trap, 0 otherwise
- Transitions: specified using probabilistic rules
  - E.g., the odds a space is a valid location (not a wall)



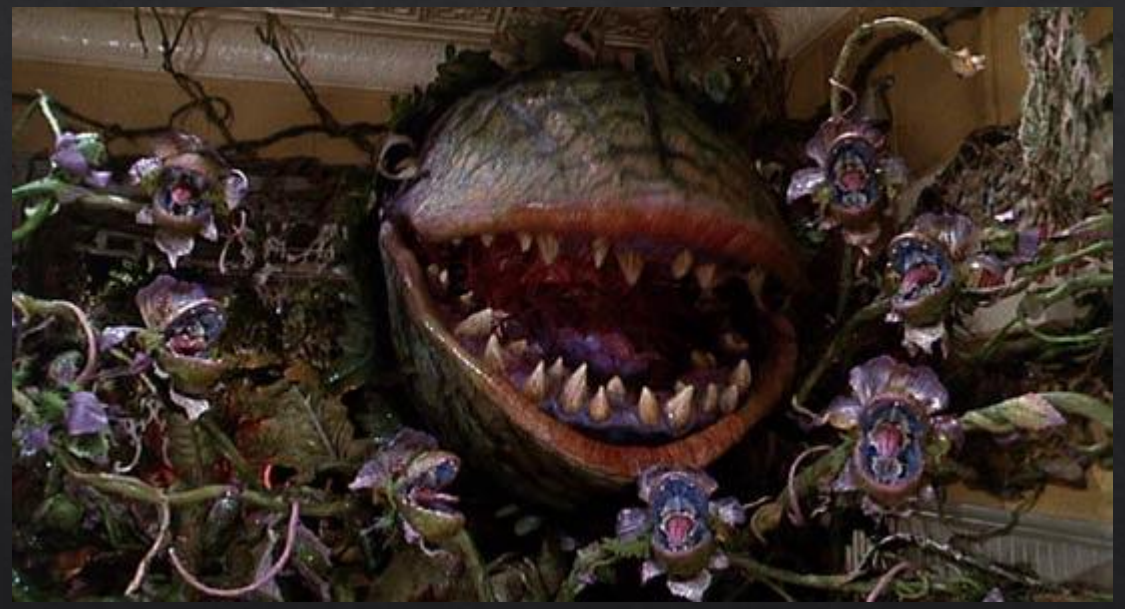
# Another Example MDP: Plant Eating Robot

- Robot surrounded by $m$ plants
- Needs to eat plants to stay alive
- Robot knows current energy level $e$
- For each plant, robot knows:
  - $P(\Delta e \mid eati)$
- If energy hits zero, robot dies



# Another Example MDP: Plant Eating Robot

- States: Energy of robot
- Actions: Choose to eat a plant
- Reward
  - Total Energy



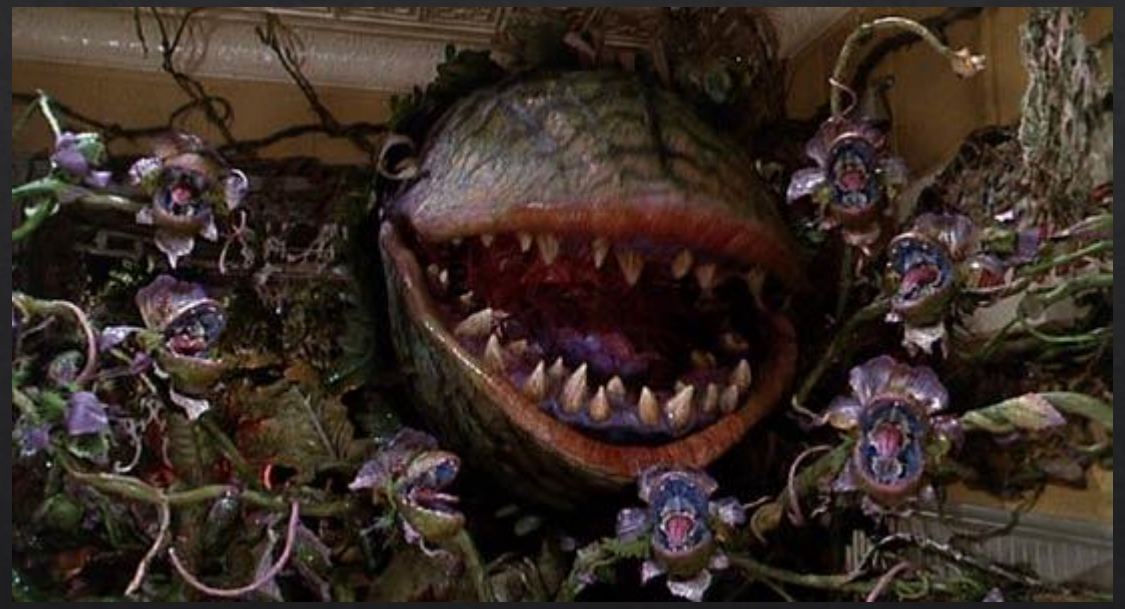
# Another Example MDP: Plant Eating Robot

- Transition model: probability of each state given previous state and action

- We have: $P(\Delta e \mid eati)$

- We want: $P(e^{t+1} \mid et, eati)$

$$= \sum_{\Delta e} P(\Delta e \mid eati) * \delta$$



$\delta$ is new energy

# Solutions to MDPs

A "Solution" to an MDP is when the agent is taking the set of actions, in order, that optimize its behavior.

This "Solution" is called a **policy** because it governs how the agent will act in all situations.

# Solutions to MDPs

A **policy π** is a mapping from S to A:

I.e, for each state, there is one action the agent always takes when in that state.

So, to follow a policy, the agent simply:

1. Determine current state **s**
2. Execute action **π(s)**
3. Go to step 1

# Solutions to MDPs

So, to follow a policy, the agent simply:

1. Determine current state **s**
2. Execute action **π(s)**
3. Go to step 1

This assumes **Full Observability**, meaning the current state and new state are always fully known to the system (though we may not know, when performing action $p_i(s)$ what state we will end up in, we do know that state once we are in it).

# Evaluating an Optimal Policy
# Value Iteration

# Reminder: Markov Decision Processes

A **Markov Decision Process (MDP)** is a model that contains:

- A set of possible world states ***S***
- A set of possible actions ***A***
- A real valued reward function ***R(s,a)***
- A description ***T*** of each action's effects in each state

***Let's calculate the optimal policy pi for this MDP!***

# Evaluating a Policy

How do we determine how good a policy $\underline{\pi}$ is?

For deterministic models, we could just sum the rewards (from ***r(s,a)***) as the agent takes actions, but results might be infinite.

For stochastic actions, instead we use **<u>expected total reward</u>**, but this again typically yields infinite value.

So, how do we compare policies of infinite value?

# MDPs: Objective Functions

Define an **Objective Function**

Maps infinite sequences of rewards to single real numbers

Options:

1. Set a finite horizon and total reward
2. Discounting to prefer earlier rewards
3. Average reward rate in the limit

#2 is most common

# MDPs: Discounting Objective Functions

A reward ***n*** steps away is discounted by $\gamma^n$

$\gamma$ is the discount rate ($0 < \gamma < 1$)

- Models mortality: agent may die at any moment
- models preference for short solutions
- smoothed out version of limited horizon look ahead

We use a **<u>cumulative discounted reward</u>** as objective:

$V(s) = r1 + \gamma * r2 + \gamma^2 * r3 + \ldots = ??$

# MDPs: Discounting Objective Functions

$V_\pi(s) = r1 + \gamma * r2 + \gamma^2 * r3 + \ldots =$

$= \sum_{i=1}^{\infty} \gamma^{i-1} * ri$

$\leq \frac{1}{1-\gamma} * M$ //M is maximum reward

AND

$\geq \frac{1}{1-\gamma} * m$ //m is minimum reward

# MDPs: More Importantly!

$$V_k(s) = r_k + \gamma * r_{k+1} + \gamma^2 * r_{k+2} + \ldots =$$

$$= r_k + \gamma(rk_{+1} + \gamma(rk_{+2} + \ldots))$$

$$= r_k + \gamma V_{k+1}(s')$$

So…a policies value (at step k) is the reward you get at that step plus gamma times the value of following the policy afterwards.

A recursive definition! CS Majors Rejoice!!!

# MDPs: Value Functions

A value function $V_\pi : S \rightarrow R$ represents the expected objective value obtained following policy $\pi$ from each state in S

Value functions partially order the policies

A policy is optimal if there is not other policy that has a higher value function starting at ANY state

# Value of a Policy

So…let's finally flesh out our definition of a policy's value

$$Q^{\pi}(s,a) = \Sigma_{s'} P(s'/s,a)\,(R(s,a,s') + \gamma V^{\pi}(s')).$$

$$V^{\pi}(s) = Q^{\pi}(s,\pi(s)).$$

# Value of a Policy

…and an optimal policy analogously (the * means that decision is optimal)

$$Q^*(s,a) = \Sigma_{s'} P(s'/s,a)\,(R(s,a,s') + \gamma V^*(s')).$$

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

$$\pi^*(s) = argmax_a\, Q^*(s,a).$$

# Value Iteration

**<u>Value Iteration</u>** is an algorithm that takes an MDP and calculates an **<u>optimal policy</u>**.

It starts at the end, in the sense that it defines **k > 0** be *the optimal policy over the last 'k' steps of an agent's processing*.

*Notice there isn't really an end, this is just so we can iterate and converge to some optimal policy.

**<u>Base Case:</u>** Let $V_0$ be an arbitrary value function (notice this is stored for each state)

# Value Iteration

**<u>Recursive Case:</u>** For increasing values of ***k***, compute:

$$Q_{k+1}(s,a) = \Sigma_{s'} P(s'/s,a)\,(R(s,a,s') + \gamma V_k(s')) \quad \text{for } k \geq 0$$

$$V_k(s) = \max_a Q_k(s,a) \quad \text{for } k>0.$$

# Partially Observable Markov Decision Processes (POMDP)

*This section courtesy of: cs.brown.edu/research/ai/pomdp/tutorial

# POMDP

- Markov decision processes are fully observable
  - i.e., we always know what state we are in at any time
- Some of the transitions were probabilistic, but after we took an action we always could observe what state we are in
- **Partially Observable MDPs** are the same except the agent is not sure what state it is in at any given time

# POMDP: Definition

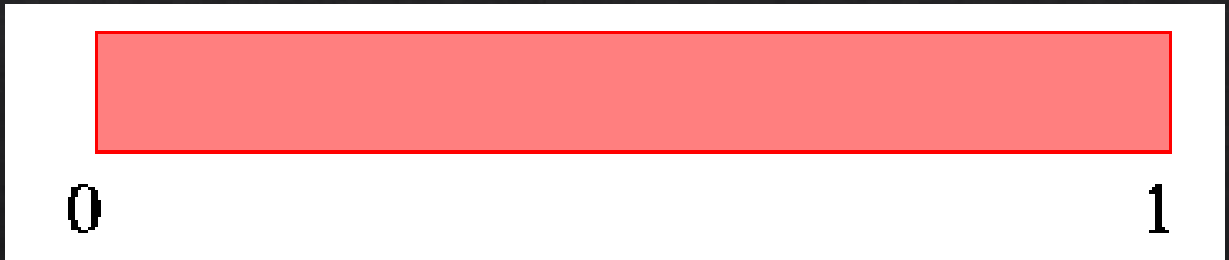
A **Partially Observable Markov Decision Process** is defined by the following:

| | |
|---|---|
| S | Set of states of the world |
| A | Set of actions the agent can take |
| O | Set of possible observations agent can make |
| $P(S_0)$ | Probability distribution of the start state |
| P(S' \| S, A) | Prob. of moving to state *S'* if in *S* and take action *A* |
| R(S, A, S') | Reward if start in *S*, take action *A*, move to *S'* |
| P(O \| S) | Probability of observing O given agent in state S |

# POMDP: Definition

- Because we do not know what state we are in, we must "guess" or have some "belief" in what state we are in.

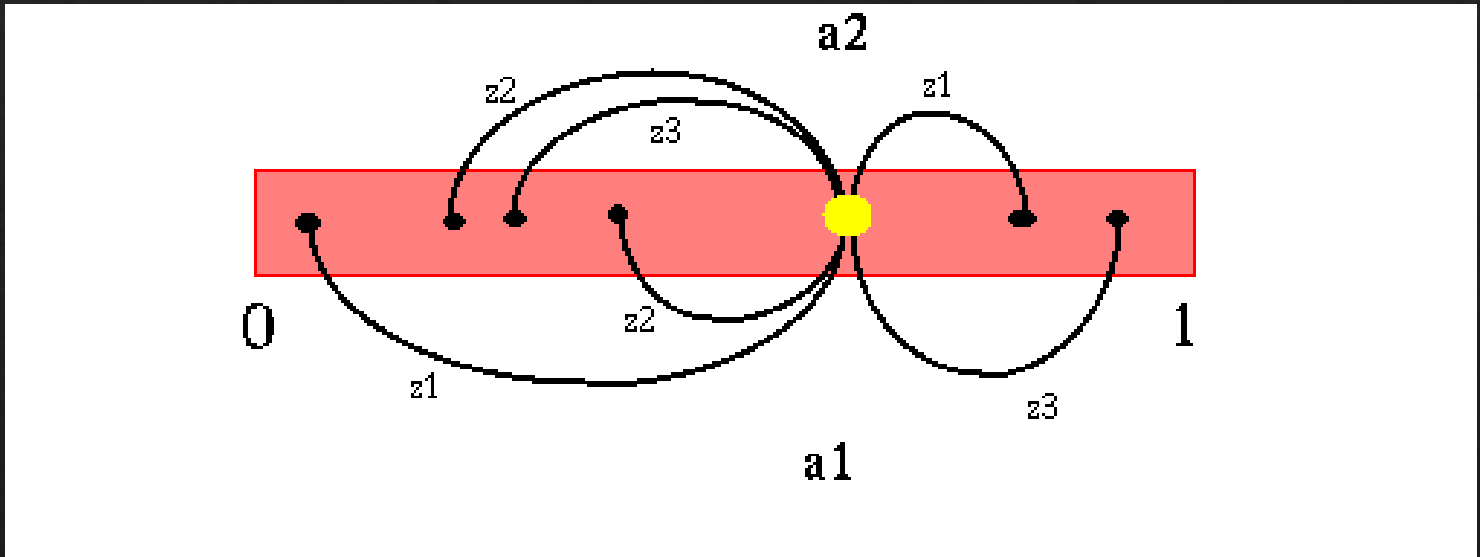
- Below is the belief space for a two state POMDP. Our belief in our state can fall anywhere in between. We want our belief to be near the edges!



# POMDP: Belief Space

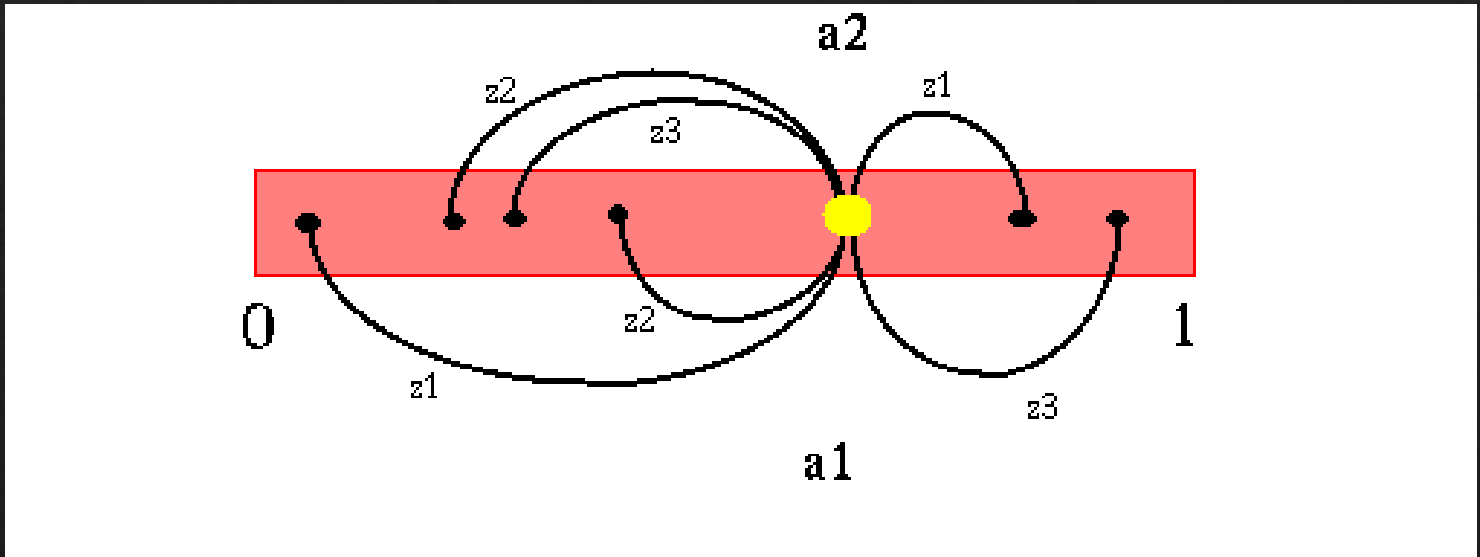
- As we take actions, our belief in our state changes. One goal of the POMDP is to update the belief in the current state at every step. This is not hard, as we can use the probabilities based on observations to guess the state.

- Visually, this might look like below: Jumping around the belief state when taking actions



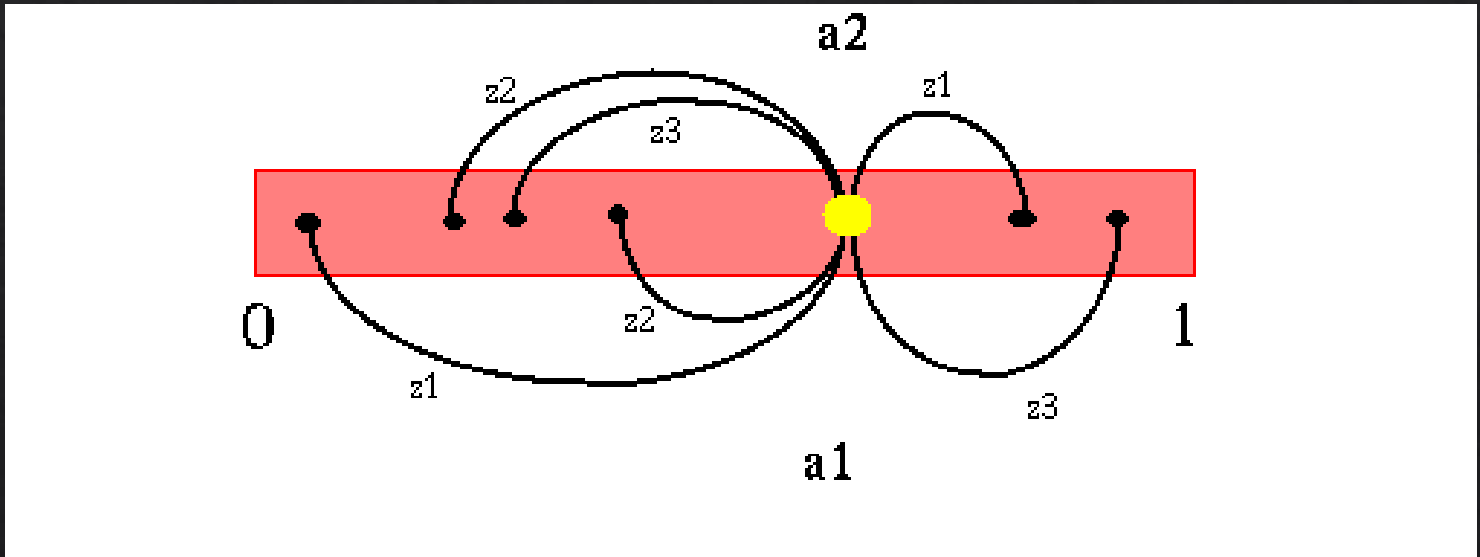
# POMDP: Belief Space

- This **Belief Space** is markovian, the next belief I will have in my state depends only on my current belief…and this is starting to look like an MDP!



# POMDP: Belief Space

- So: let's solve POMDPs by:
  - Converting into an equivalent MDP (belief states = states of and MDP)
  - Use value iteration as before to find the optimal policy
  - Problem: Belief space is still continuous, so can't represent with an array like before!



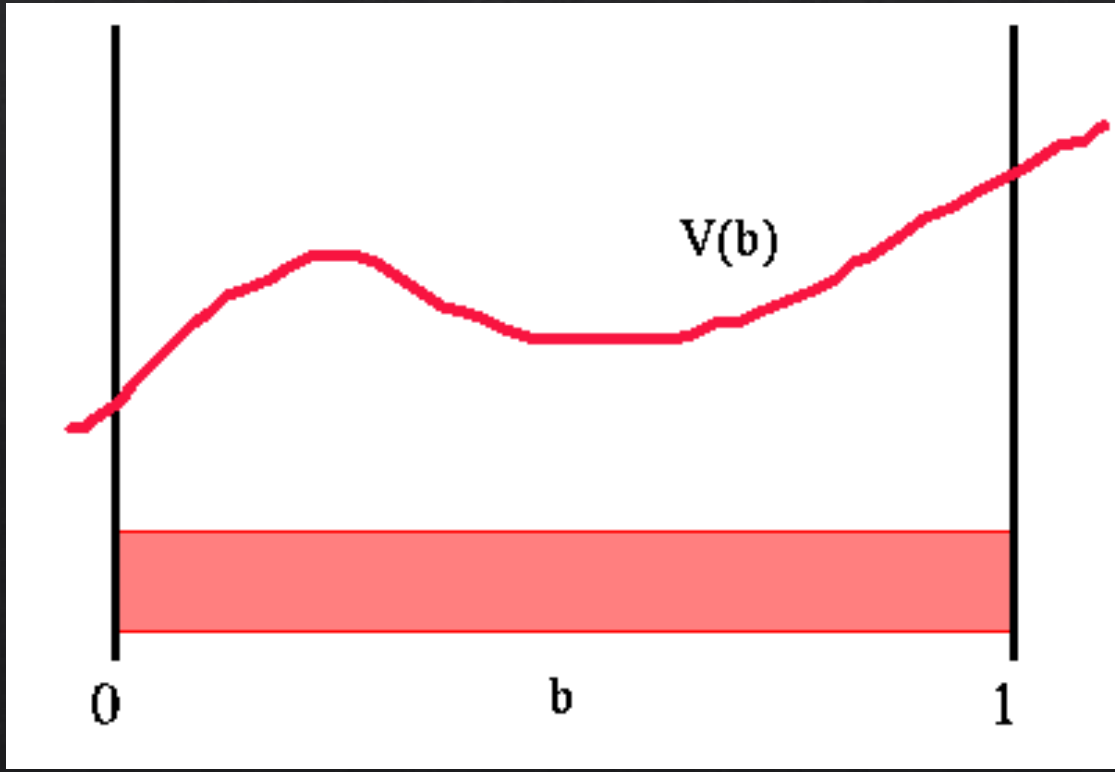
# POMDP: Problem!

- This algorithm has some issues. The belief space is continuous, not discreet!

- With MDPs, we had a value function that was just a number

- Now, we have value functions continuous over the given belief space, see right →

How can we represent V(b)??



# POMDP: Insight!

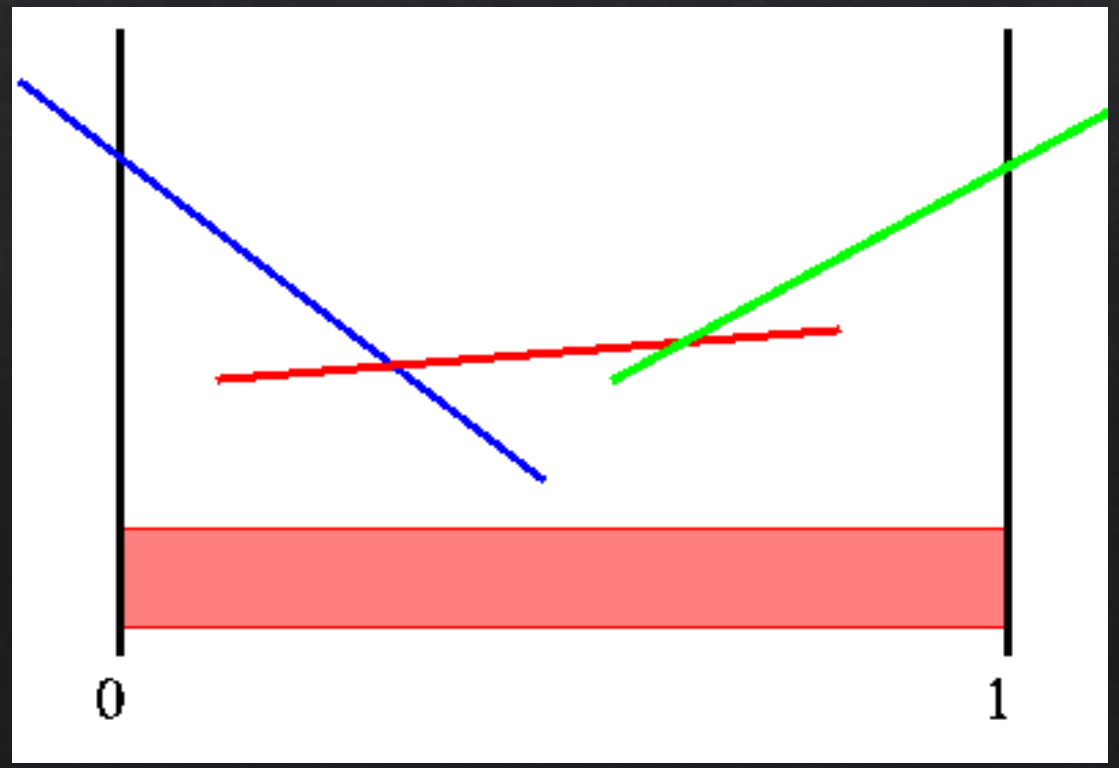
- It turns out that for finite horizon problems, the value function of a POMDP can be represented as a piecewise linear and convex function.

- Or, as a set of line segments like shown

- Why? Let's hold off on that for a moment.

for now, just believe me ☺

Thus, for value iteration, only need to store these finite lines in memory



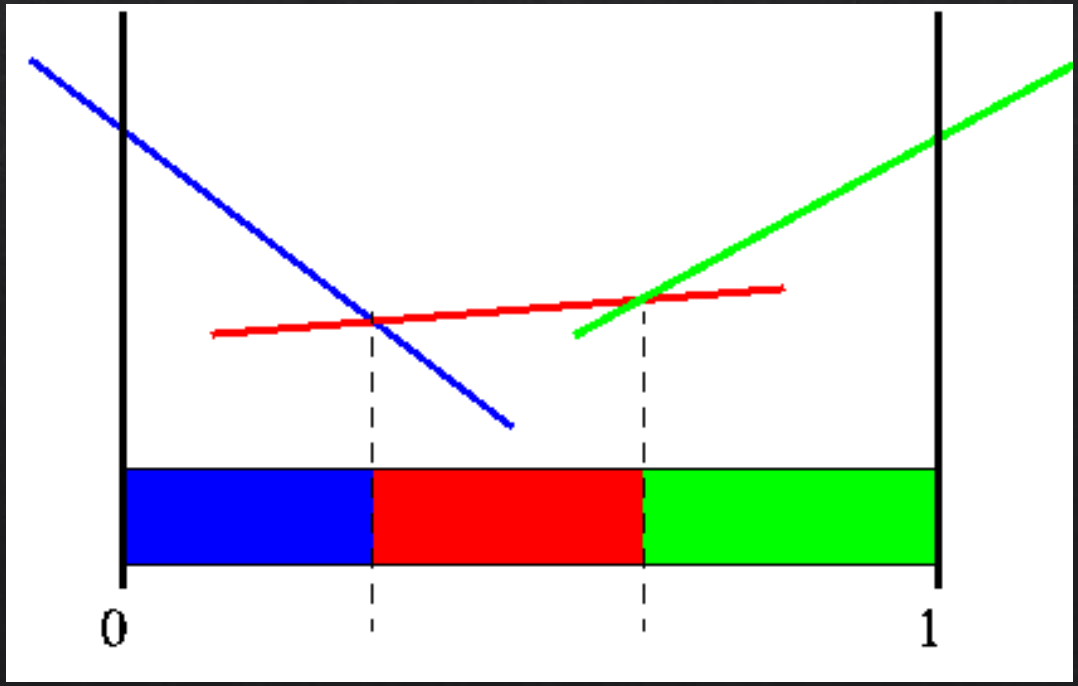
# POMDP: Insight!

- So, to solve for a policy, we want to use the same algorithm (value iteration)

- Problem 1: Value function is continuous!
  - Solved, because function is piecewise linear convex

- Problem 2: To find a states new value (while iterating), we need to loop over all next states, but this is infinite!

Sigh…just need to fix problem 2 now

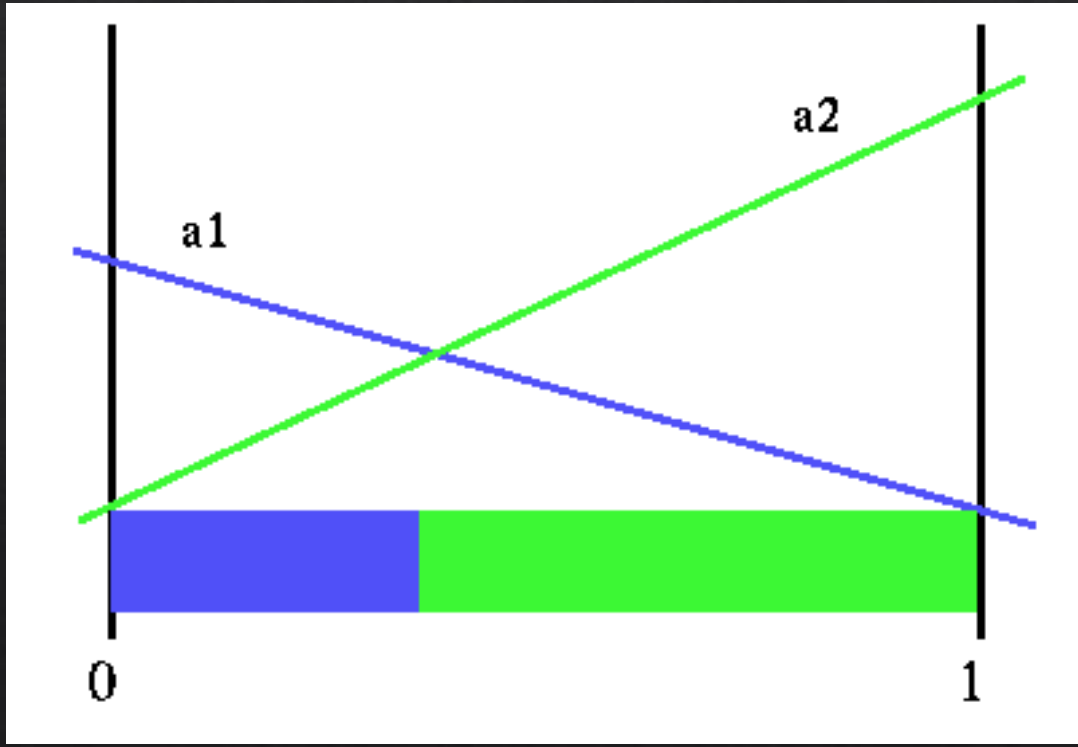


# POMDP: Example!

- Let's use a simple example to try to understand this. Assume we only take 1 action:

- a1 and a2 are two actions we can take
- Where they intersect 0 and 1 are the rewards from the POMDP
- The lines are interpolated between the two

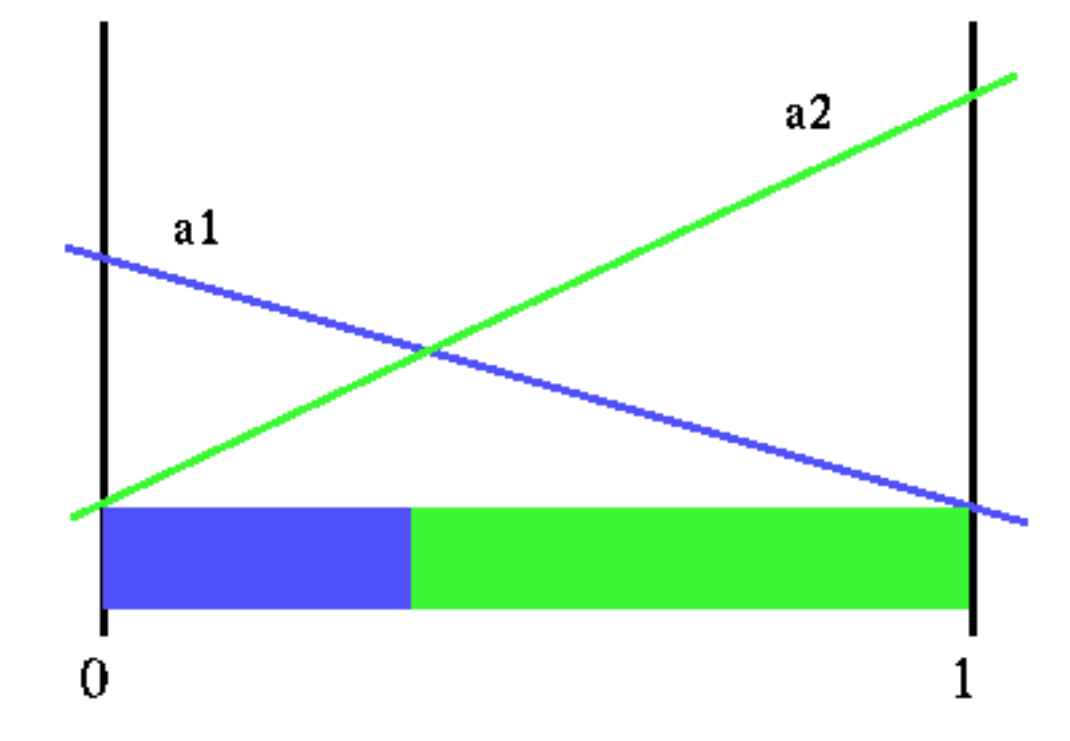
- Where they cross, it shouldn't really matter which action you take (in theory)



# POMDP: Example!

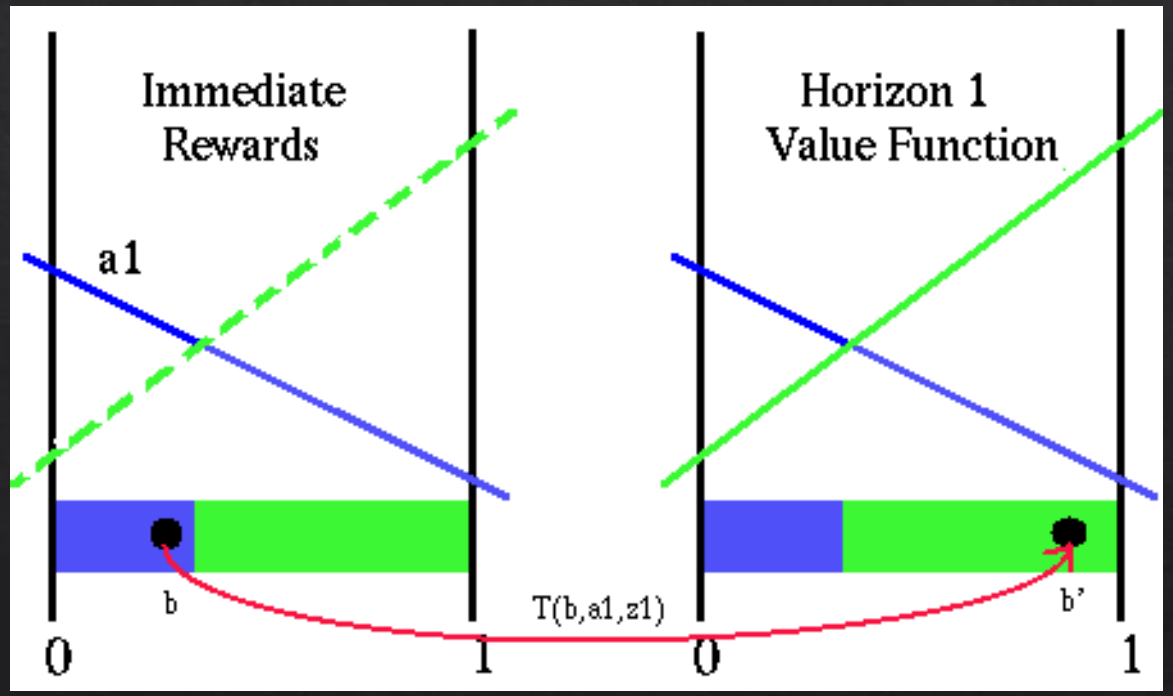
- This works if we are only taking one action, but what if we are taking two? Or three? Or ‘n’?

- Let’s start with two, we will show this in three stages:
  - 1. Compute value of belief state of given action and observation
  - 2. Compute value of belief state given only an action
  - 3. Compute actual value of belief state



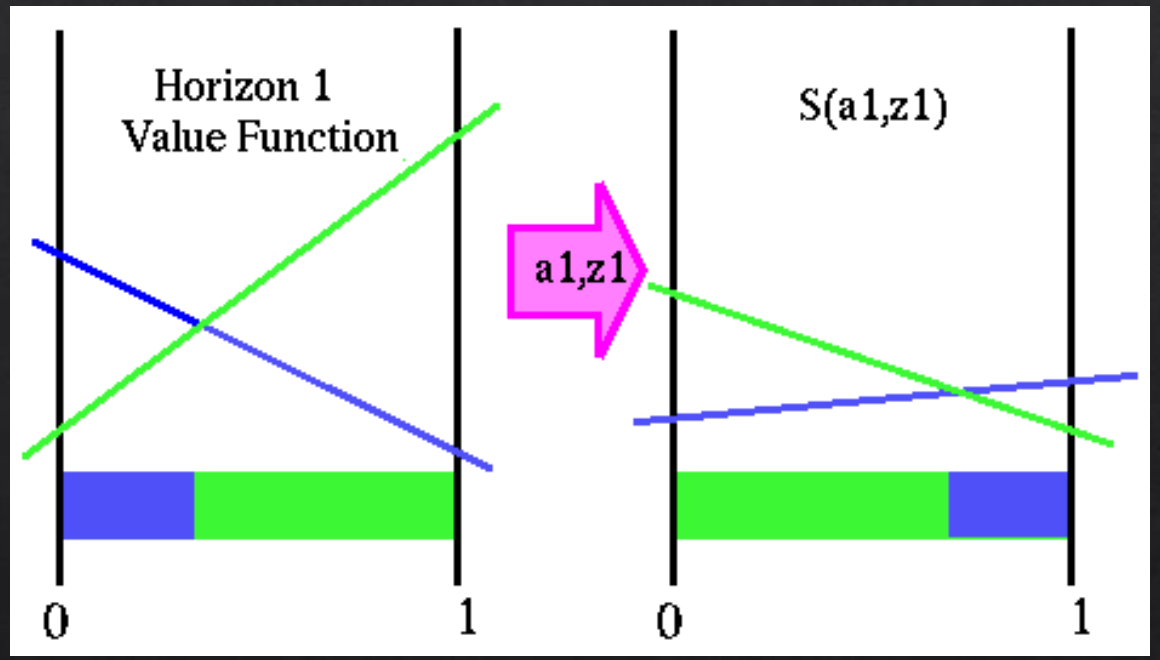
# POMDP

- 1. Compute value of belief state of given action and observation
- If the next observation (z1) is fixed, what is the best set of two actions (value)?
- Easy, just do a1 then a2 (see pic at right)
- T(b,a1,z1) returns the new belief state if in belief state b currently, take action a1, and then observe z1



# POMDP

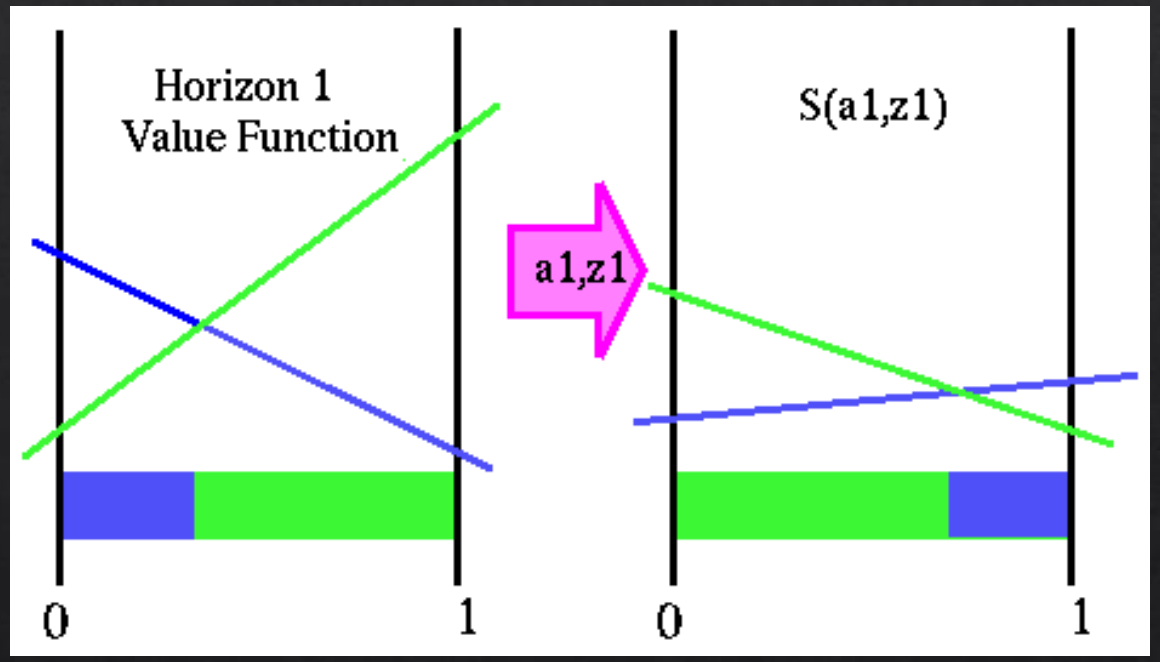
- 1. Compute value of belief state of given action and observation
- To simplify this, we want to compute S(a1,z1), which is the value of all beliefs, given you've already taken a1 and are now observing z1
- So, S(a1,z1) tells you the value of the horizon 1 function but as a function of your current belief b (no need to transform b into b')
- So, for some belief b we get S(a1,z1) = (r(a1,b) + T(b,a1,z1)*r(ai,b')) P(x) //probability this whole situation happens



# POMDP

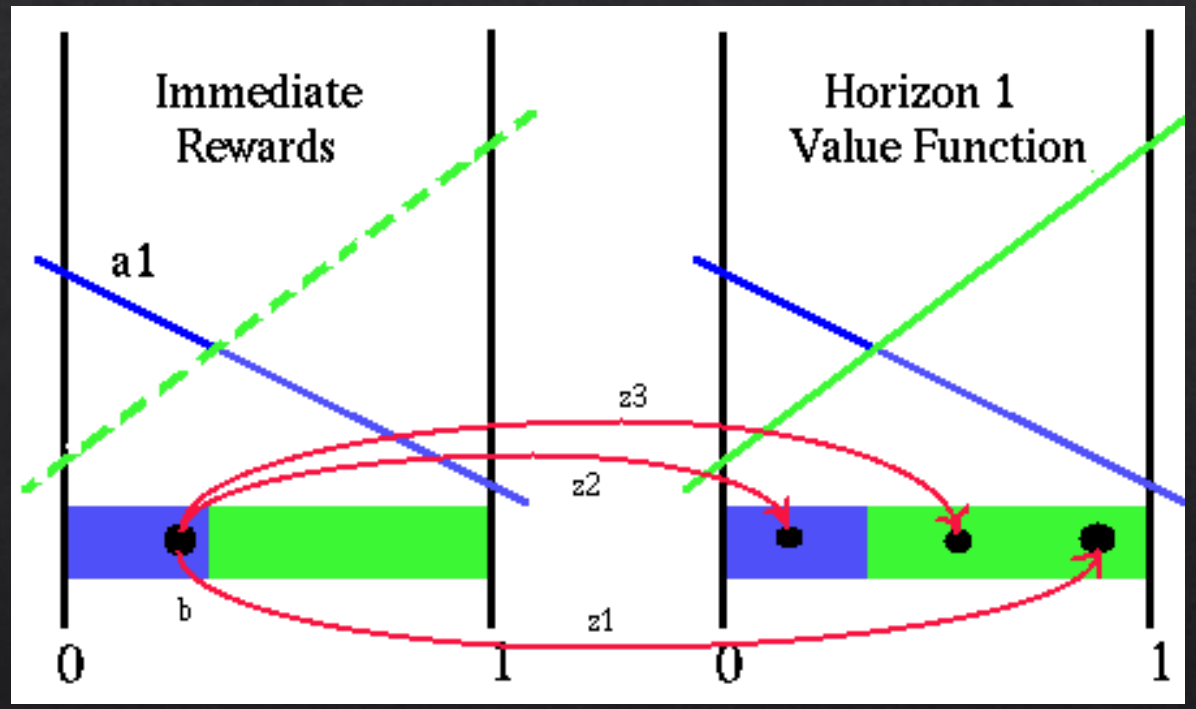
- 1. Compute value of belief state of given action and observation

- Not clear how to do this transformation! But for now, just try to understand the intuition!



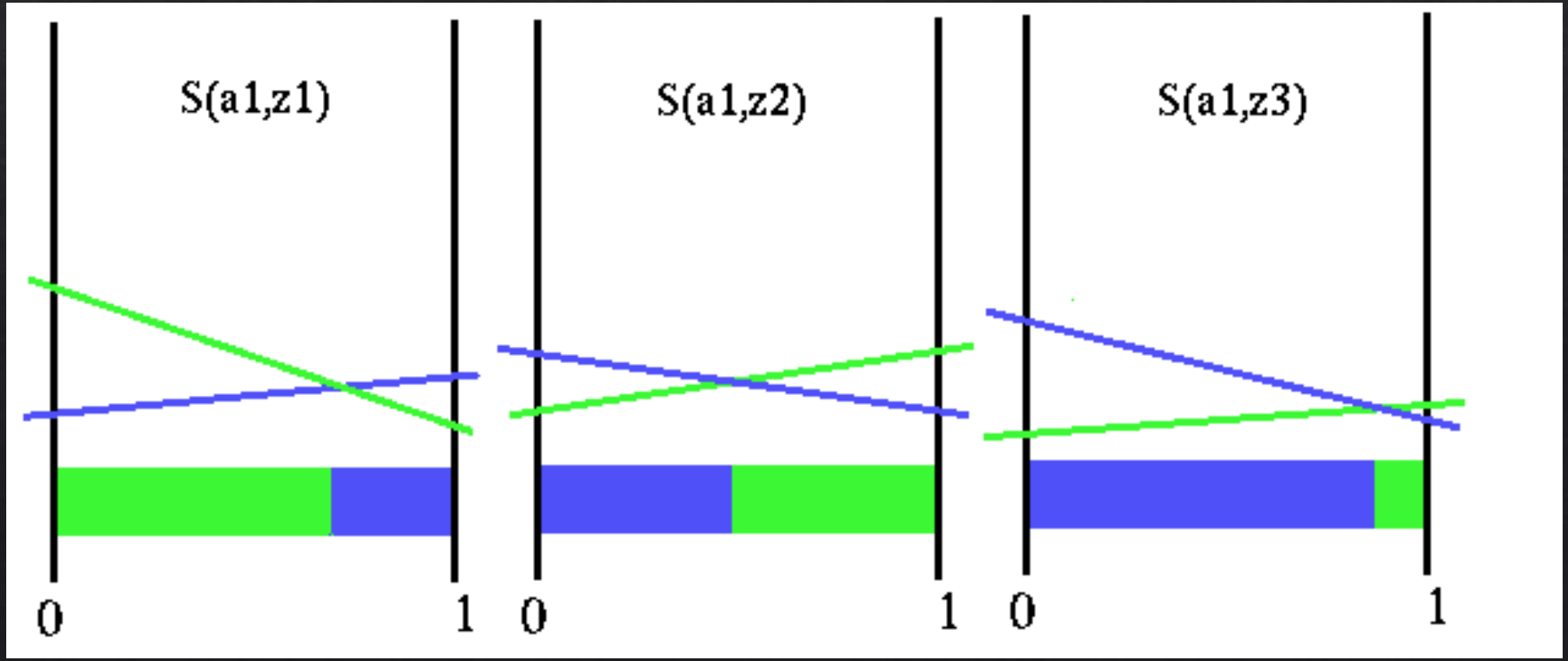
# POMDP

- 2. Compute value of belief state given only an action
- When we get rid of the constant observation z1, we now have many new observations that could occur. Need to account for all possible next observations
- So, do the same thing, but weight each by probability of the next observation



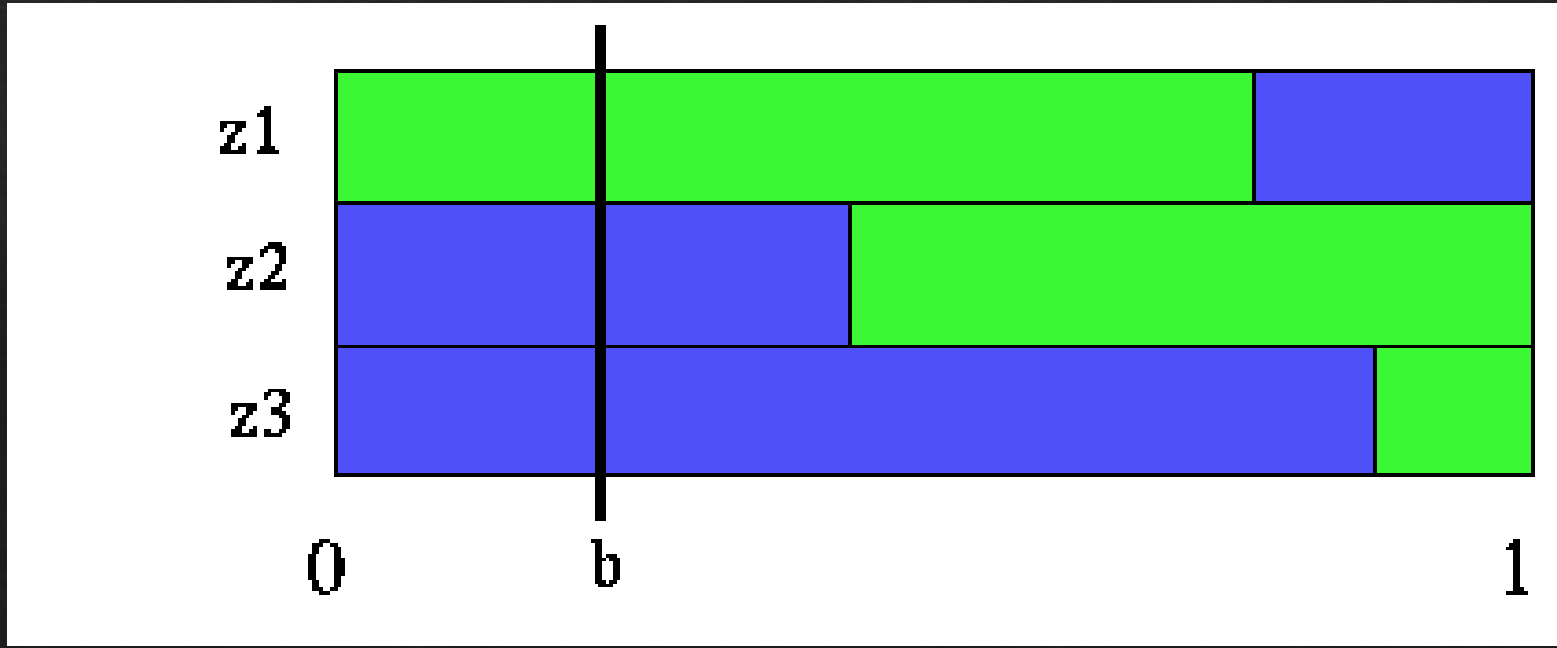
# POMDP

- 2. Compute value of belief state given only an action
- So, we want to calculate at S(a1,z) function over all possible observations z



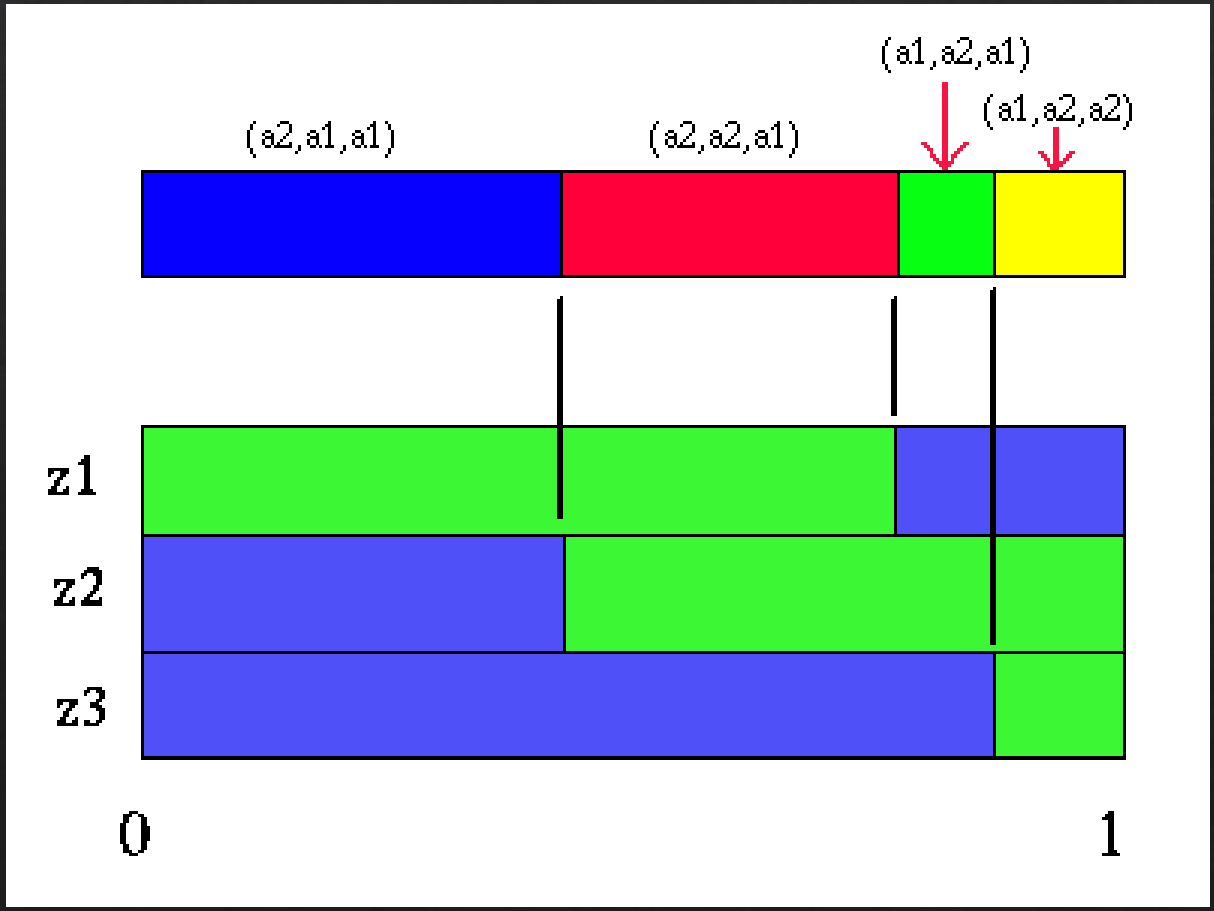
# POMDP

- 2. Compute value of belief state given only an action
- In practice, we would take whichever action is best after making next observation (z1,z2,or z3)
- However, the expected value of this state, given b, is just the sum of the three + horizon1(a1,b)
- Why no weights? The probabilistic weights are actually built into each S() function



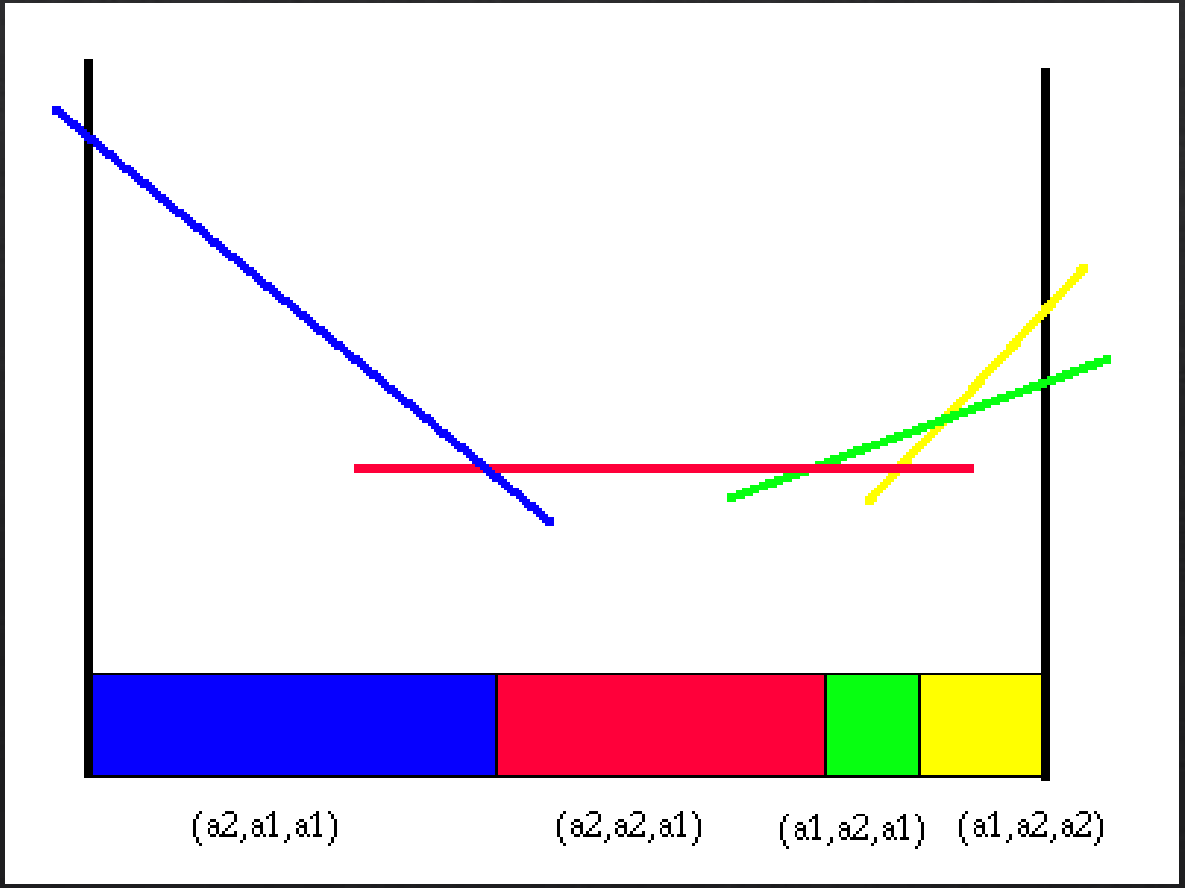
# POMDP

- 2. Compute value of belief state given only an action
- Only four distinct "future strategies"



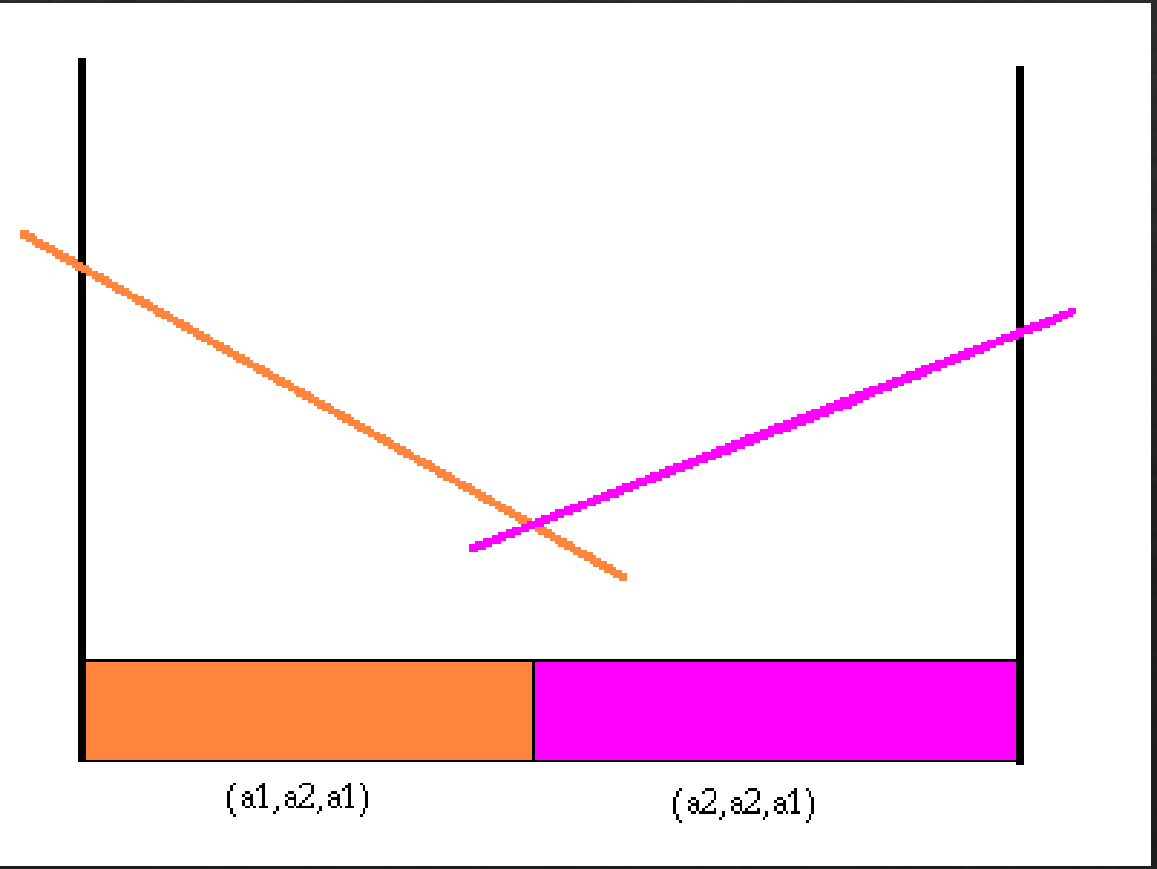
# POMDP

- 2. Compute value of belief state given only an action
- Only four distinct "future strategies"
- So, we know how to solve our problem
- Given belief state b
  - This function to the right tells us the value we should get after taking a1 and one other action.
  - So it is the general value of being in that belief state given that we plan to take two actions (a1 and one other)



# POMDP

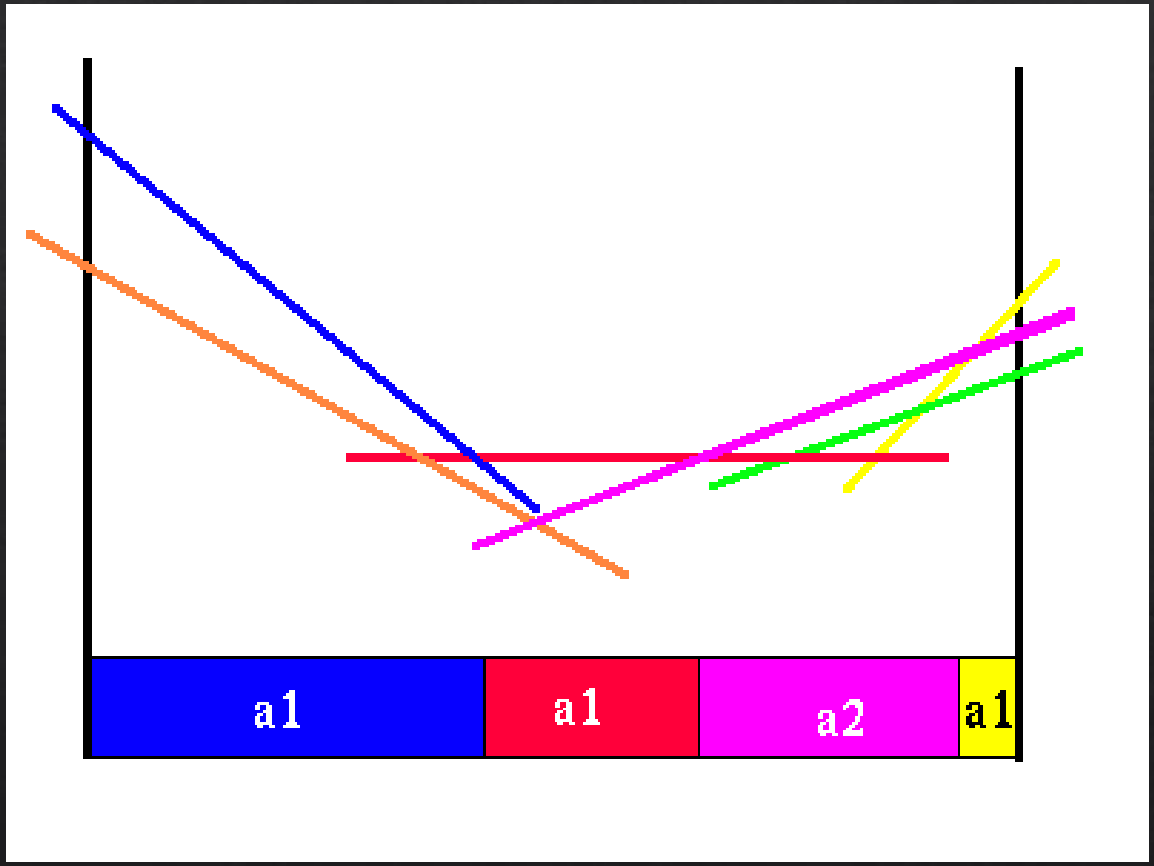
- 2. Compute value of belief state given only an action
- We only did this for a single first action a1
- We need to repeat this process for other actions
- So a2 in this example
- Let's suppose it looks like this →



# POMDP

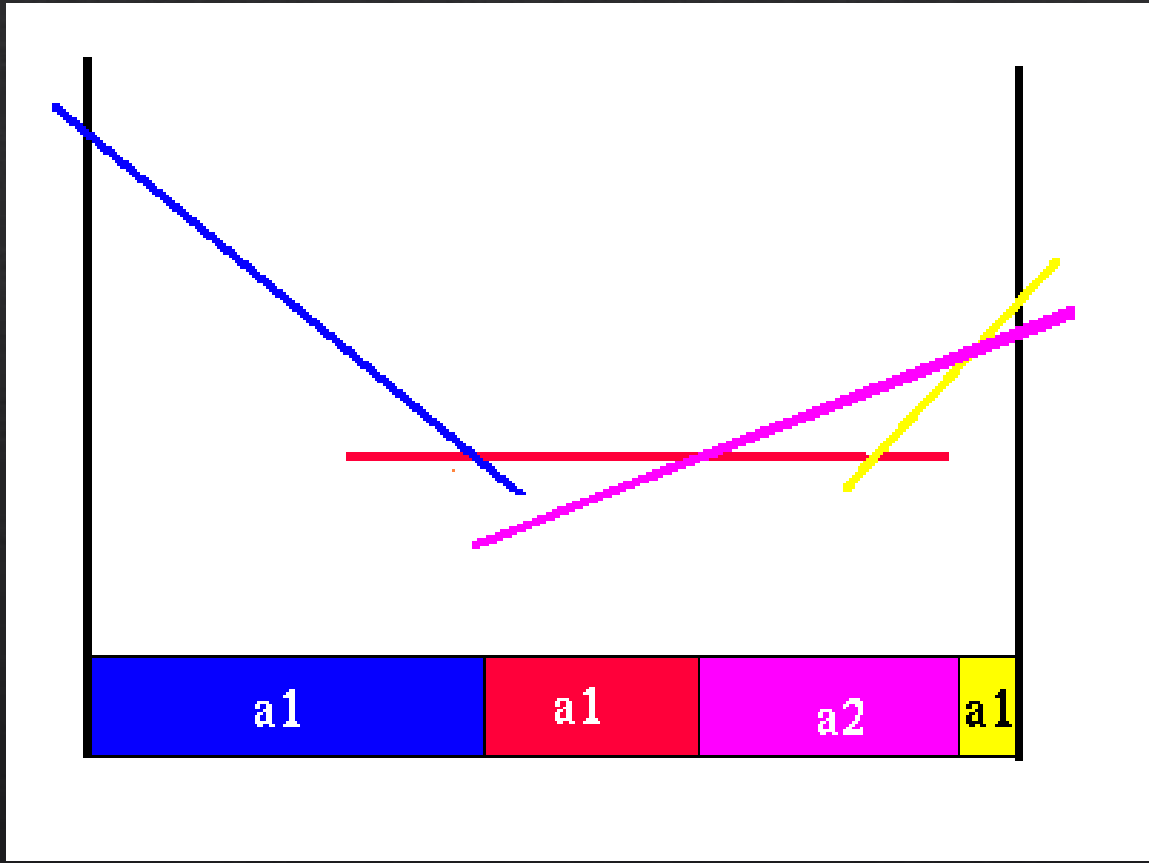
- 3. Compute actual value of belief state

- We simply overlay the belief segments for the two actions on top of one another →



# POMDP

- 3. Compute actual value of belief state
- We simply overlay the belief segments for the two actions on top of one another →
- …and remove lines that are occluded
- **Remind you of anything? Cough Cough cs4102 written 3 lines problem!!!*



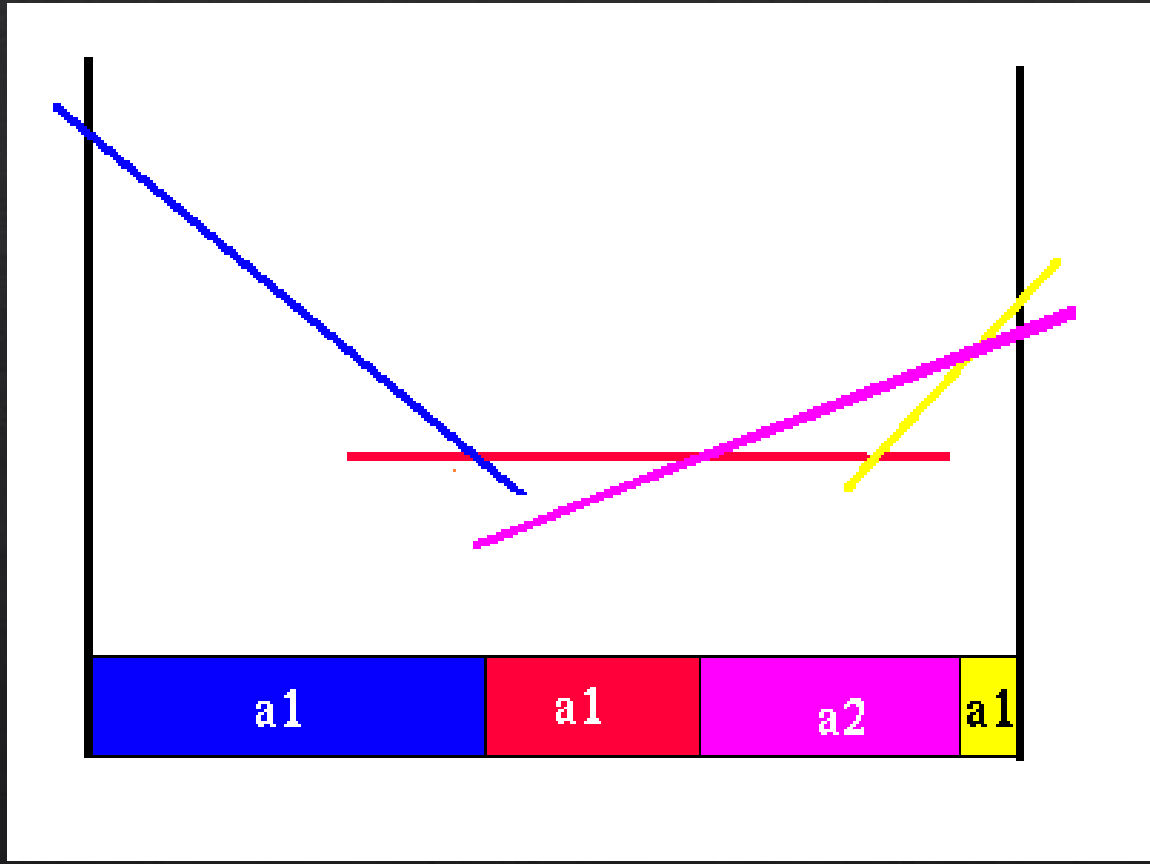
# POMDP: Summary!

What do we do now?

We can repeat the WHOLE process again. This will yield the horizon 3 value function.

Notice that horizon n+1 value function may actually have less line segments that horizon n function.

Keep iterating as many times as possible to look as far ahead as possible.



# Conclusions

# Summary

| Markov Models | | Do we have control over the state transitons? | |
|---|---|---|---|
| | | NO | YES |
| Are the states completely observable? | YES | **Markov Chain** | **MDP**<br>Markov Decision Process |
| | NO | **HMM**<br>Hidden Markov Model | **POMDP**<br>Partially Observable Markov Decision Process |