A Brief Introduction to Reinforcement Learning and Markov Decision Processes

Eirik Fagerhaug

Norwegian University of Science and Technology

Slides Adapted from/based on:

- Slides from Erlend Coates
- Slides from Hanna Hajishirzi
- Slides from Deepmind (Hado von Hasselt)
- Grokking DRL (Miguel Morales)
- AI: A Modern Approach (Russel and Norvig)

Reading Material

Russel and Norvig, Chapter 16.1 and Chapter 23.1

Speaker notes

A lot of stuff in one sub-chapter

- → Introduction to Reinforcement Learning

- Markov Decision Processes (MDPs)

Speaker notes

We will first go through a short introduction to Reinforcement learning, before discussing Markov decision processes.

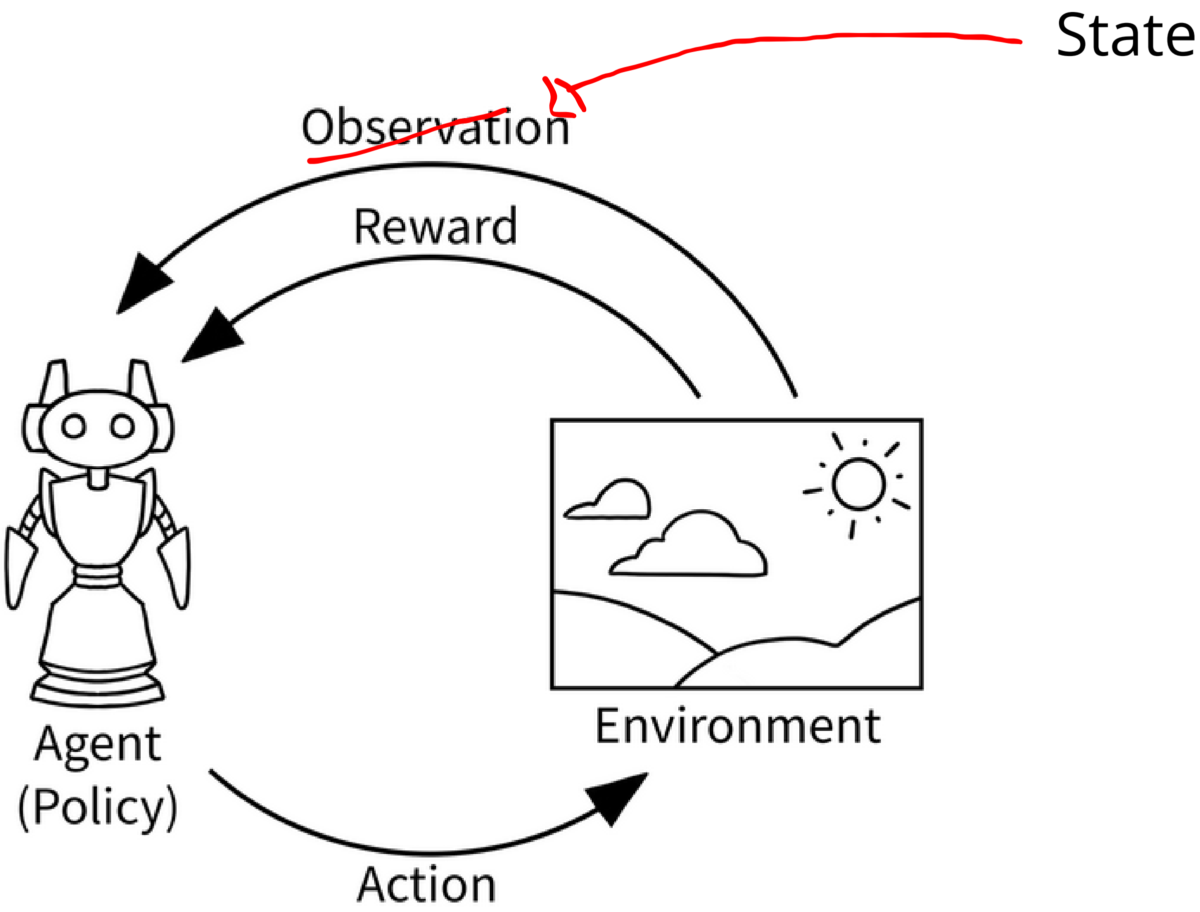
Reinforcement Learning

- People and animals learn by interacting with our environment
- This differs from certain other types of learning
  - It is active rather than passive
  - Interactions are often sequential
- We are goal-directed
- We can learn without examples of optimal behavior
- Instead, we learn by receiving some reward signal

Speaker notes

Reinforcement learning is based on psychology, and on how humans and animals learn.
This differs from a lot of other types of learning, and we can say that we have an active, goal-directed way of learning and can learn without being given examples of optimality.

The interaction loop



Goal: optimize the sum of rewards through repeated interaction

Speaker notes

Reinforcement learning uses what we call the interaction loop.
Here we have some agent acting on some environment and receiving a state and a reward as a result.
Note that there are a lot of different terminologies in reinforcement learning, one example is "observation" and "state", for now, we will only use the term "state" as this is the term used by the book.
The goal of reinforcement learning is to maximize the sum of rewards through repeated interaction.

The reward hypothesis

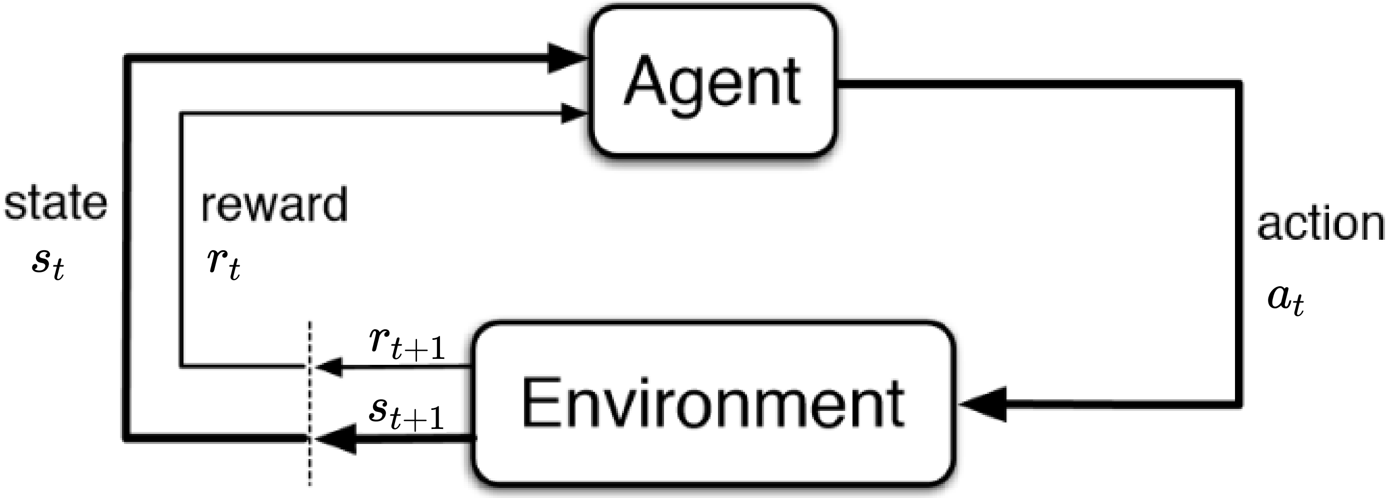
Reinforcement learning is based on the reward hypothesis:

"Any goal can be formalized as the outcome of maximizing a cumulative reward."

What is Reinforcement Learning?

- The science and framework of learning to make decisions from interaction
- This requires us to think about:
  - time
  - (long-term) consequences of actions
  - actively gathering experience
  - predicting the future
  - dealing with uncertainty
- Huge potential scope

Agent-Environment interaction



- At each step *t* the agent:
  - Receives state s_t (and reward r_t)
  - Executes action a_t
- The environment:
  - Receives action a_t
  - Emits state s_{t+1} (and reward r_{t+1})

Speaker notes

We can formalize the interaction loop mentioned earlier

States, Actions and Rewards

The state is a unique and self-contained configuration of the environment.

An action is what the agent does to affect the environment.

The reward is a measure of how well the agent is doing.

States, Actions and Rewards: Examples

Example: Playing Chess

- State: The organization of pieces on the board
- Action: The agents' (your) move
- Reward: +/- for winning or losing, e.g., +1 for winning, +1/2 for a draw, 0 for losing

Example: Managing an investment portfolio

- State: The stock market and your own portfolio
- Action: Sell / buy
- Reward: +/- for increase/decrease in portfolio total worth

Example: Making a robot walk

- State: Orientation (and position) of robot and its limbs
- Action: Moving the joints/limbs
- Reward: +/- for forward-motion / falling over

Speaker notes

Note that we are using the term reward also if it is negative.

RL Core concepts

The reinforcement learning formalism includes:

- Environment (dynamics of the problem)
- Reward signal (specifies the goal)
- An Agent, containing:
  - Policy
  - Utility function (also called value function)
  - A model

Elements of an RL agent

An RL agent may include one or more of the following components:

- Policy: Agents' plan/behavior.
- Utility function: A function to evaluate how good/bad a state is.
- Model: A model predicts what the environment will do next.

RL methods can be classified as either utility-based or policy-based and as either model-based or model-free.

Speaker notes

We will go over all of these terms either today or next week.

Example Video

Speaker notes

There are a lot more examples, including some longer documentaries on the net, that I would recommend people to watch, I can add some links to blackboard.

- Introduction to Reinforcement Learning

→ - Markov Decision Processes (MDPs)

FrozenLake Environment

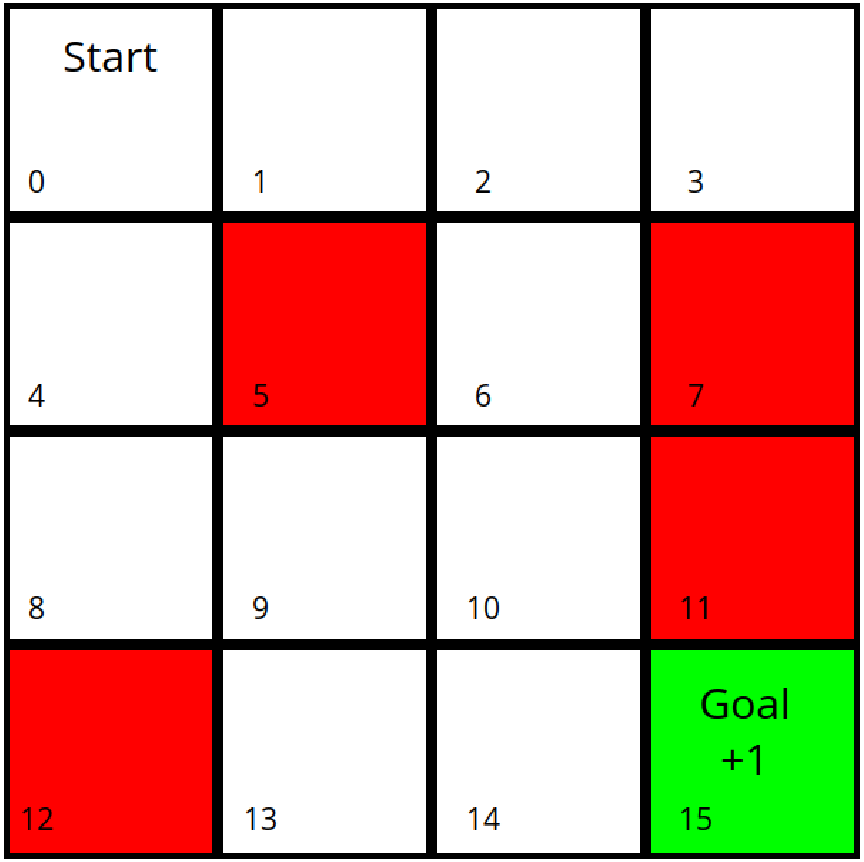
Winter is here. You and your friends were tossing around a frisbee at the park when you made a wild throw that left the frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you'll fall into the freezing water. At this time, there's an international frisbee shortage, so it's absolutely imperative that you navigate across the lake and retrieve the disc. However, the ice is slippery, so you won't always move in the direction you intend.

Speaker notes

Before we start discussing MDPs I want to introduce a "game", or environment from OpenAI's Gym framework.

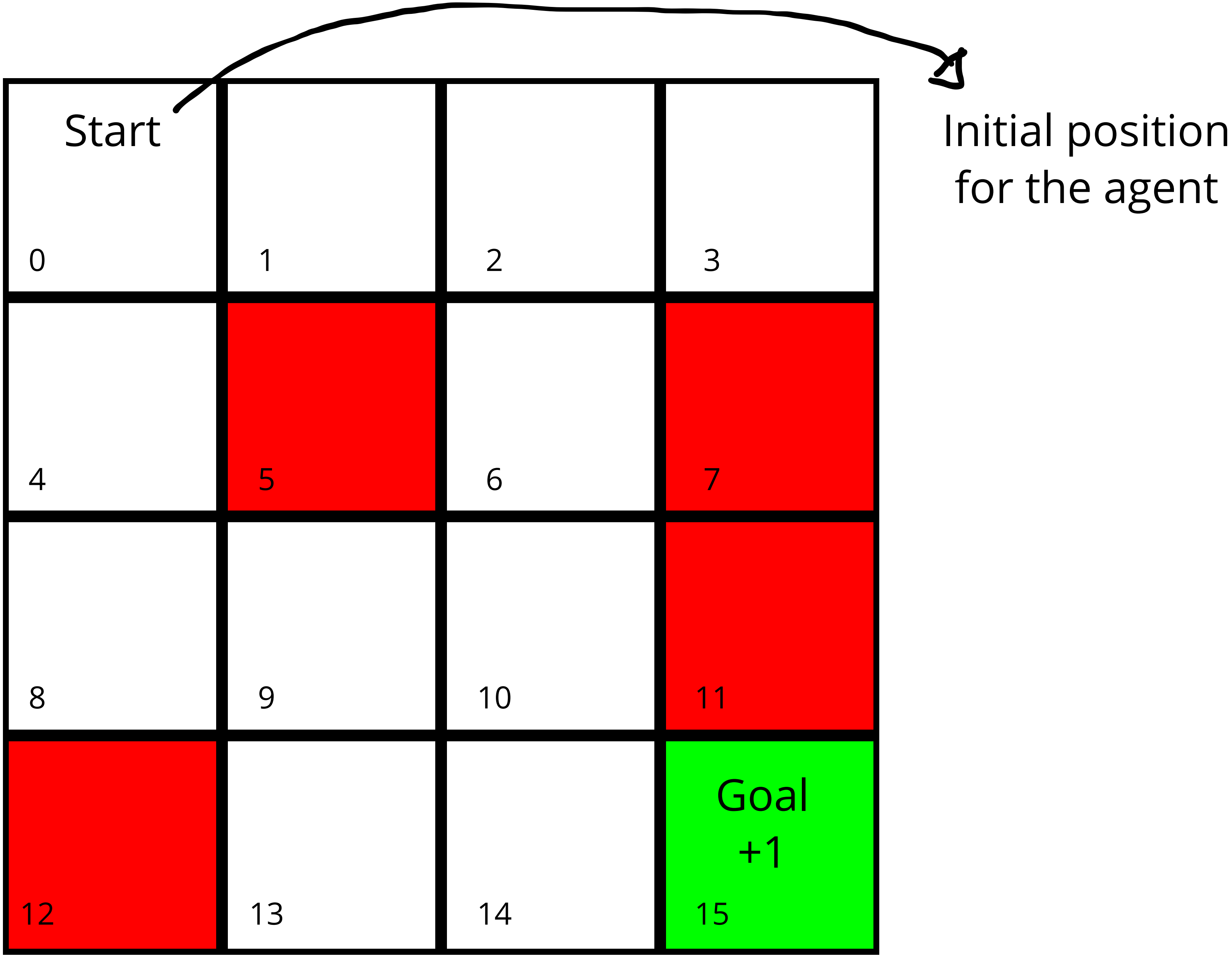
I will be using this as an example in the slides, and it will be used in the assignments, though mostly next week.

The goal of the game is that you, or the agent, have to navigate a frozen lake filled with holes to retrieve a frisbee.


Start
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Goal
+1
15

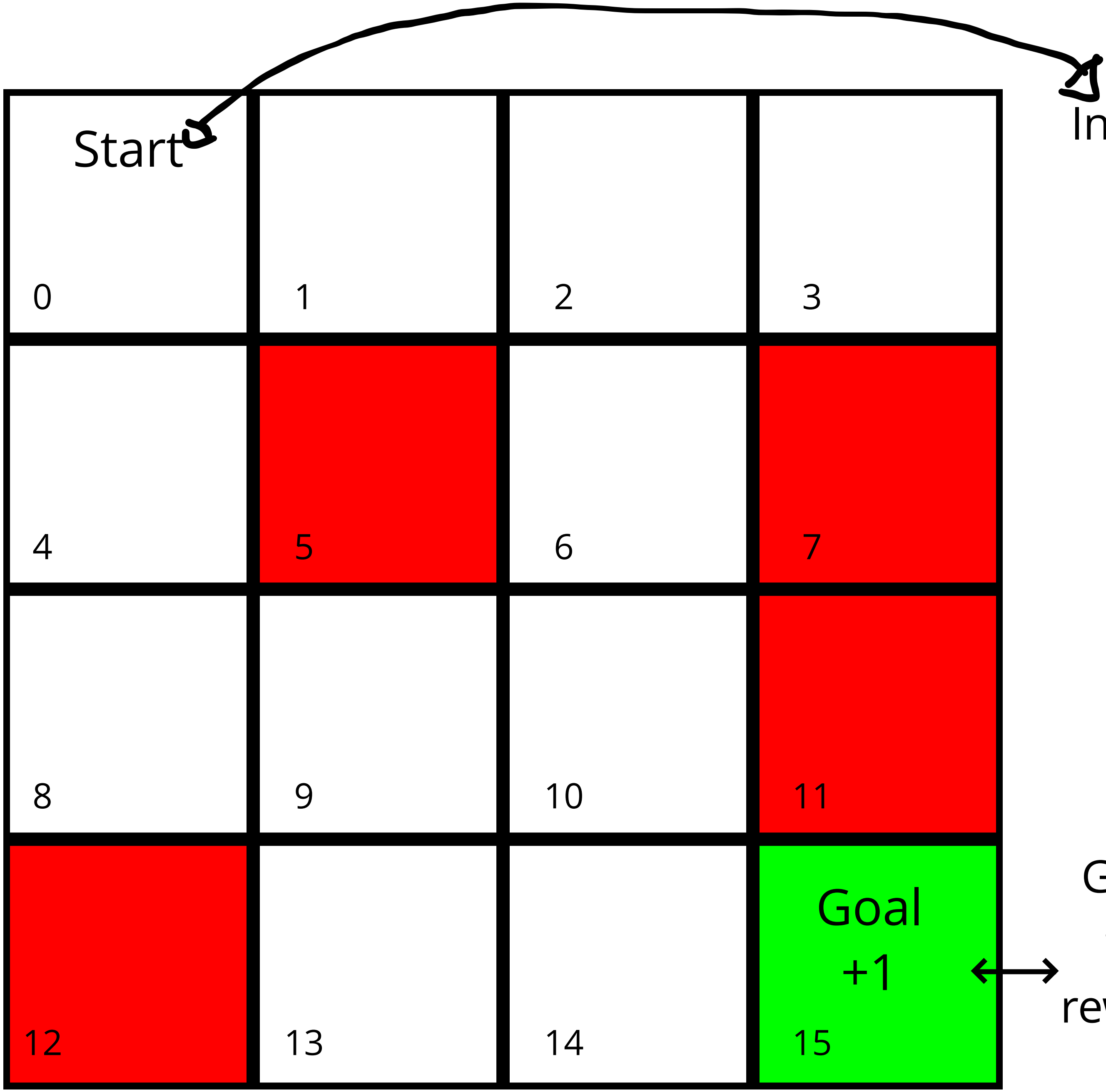
Speaker notes

On the left, we have an image of how OpenAI gym renders the game, while on the right is how I will represent it in the slides.
It is a simple grid-based environment, the default one being of size 4x4.


Start
Initial position for the agent
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Goal
+1
15

Speaker notes

The environment has one initial starting position, in this case, tile 0.



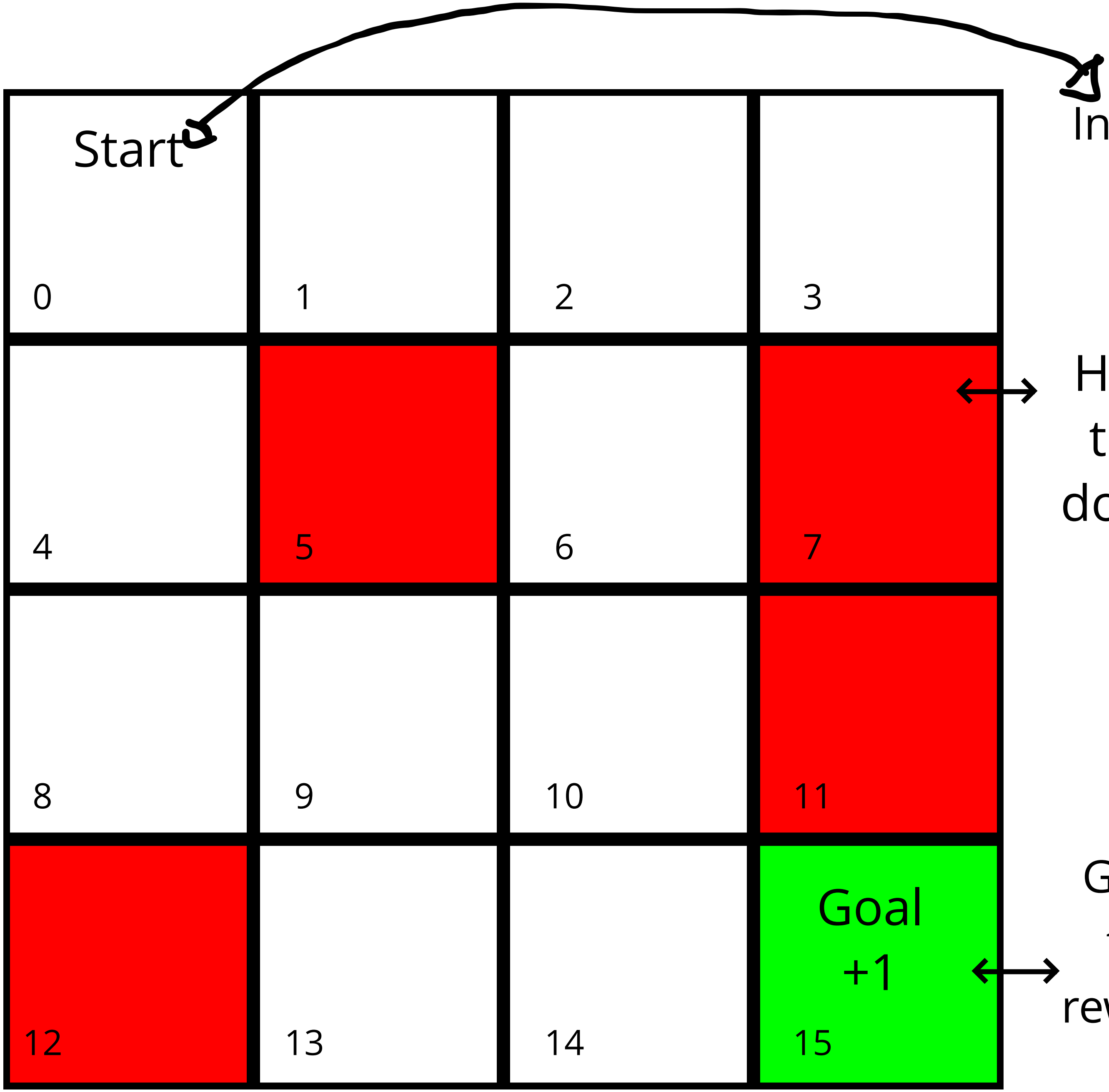
Start 0	1	2	3
4	🟥 5	6	🟥 7
8	9	10	🟥 11
🟥 12	13	14	🟩 Goal +1 15

Initial position for the agent

Goal state; ends the game and rewards the agent with 1 point

Speaker notes

and it has one end-state, which rewards the agent with one point and ends the game upon entering



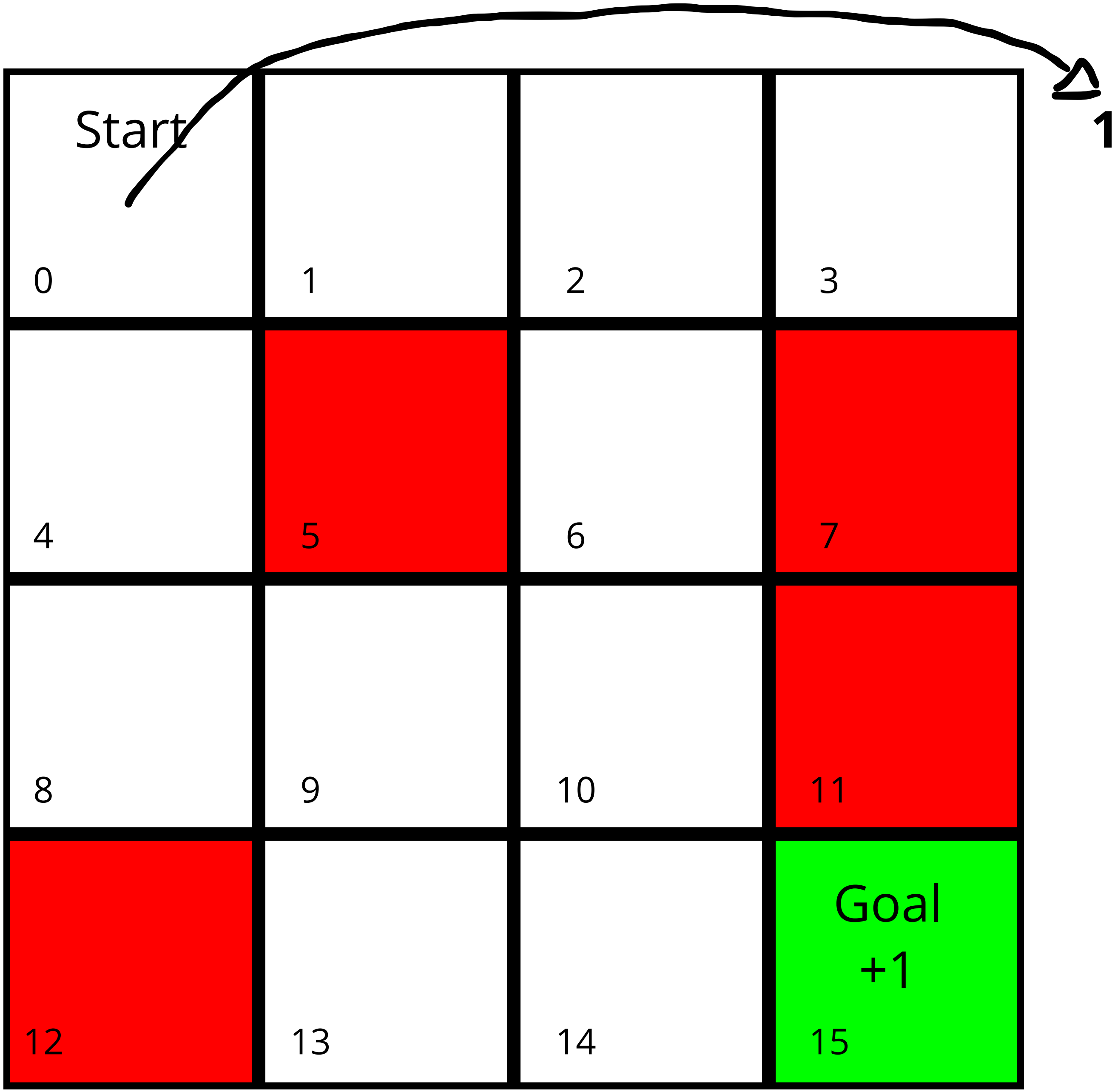
Initial position for the agent

Hole; also ends the game, but does not reward the agent

Goal state; ends the game and rewards the agent with 1 point

Speaker notes

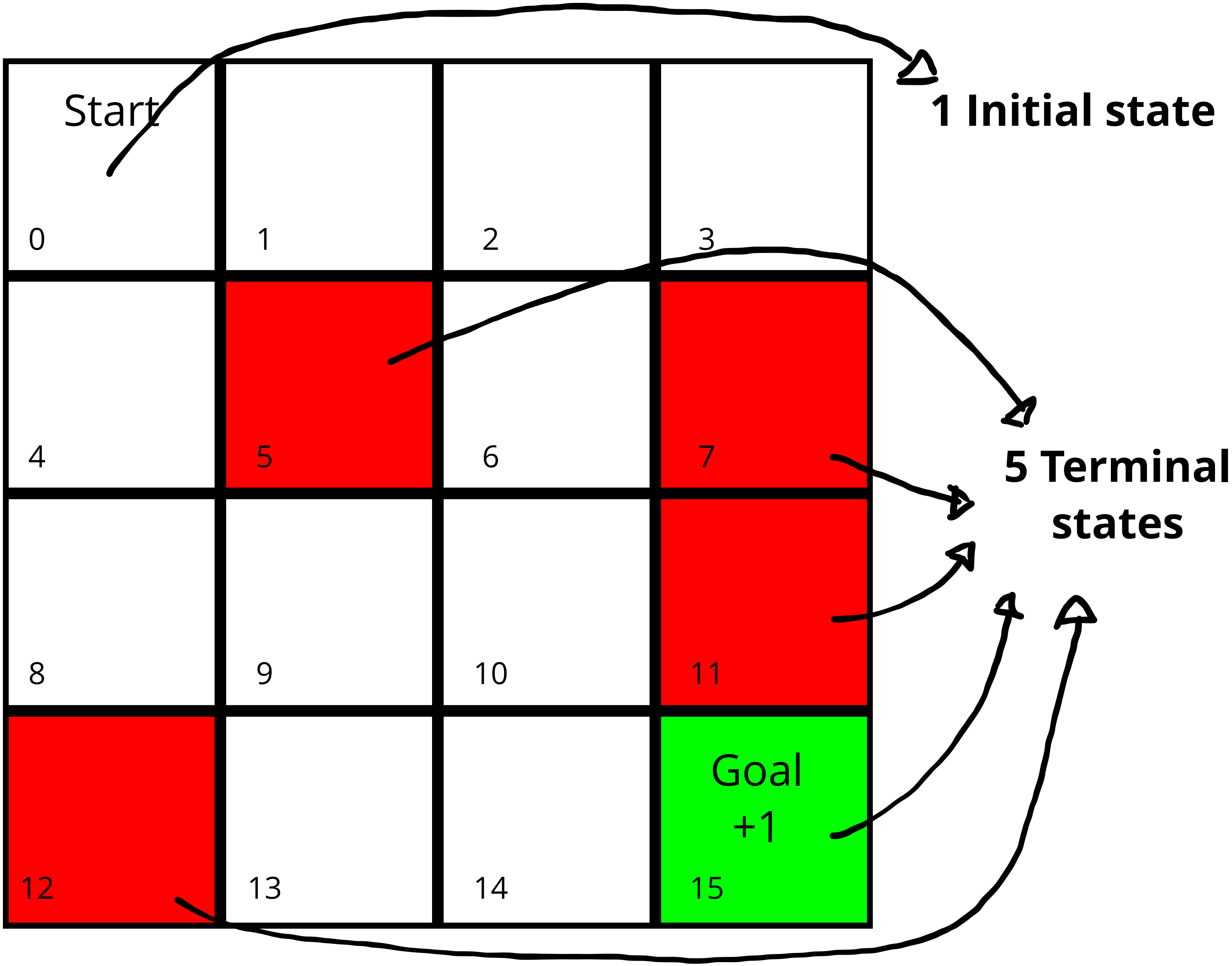
There are also a number of holes, in this case, 4, that also terminate the game upon entering, but without giving any rewards.



1 Initial state

Speaker notes

This gives us one initial state



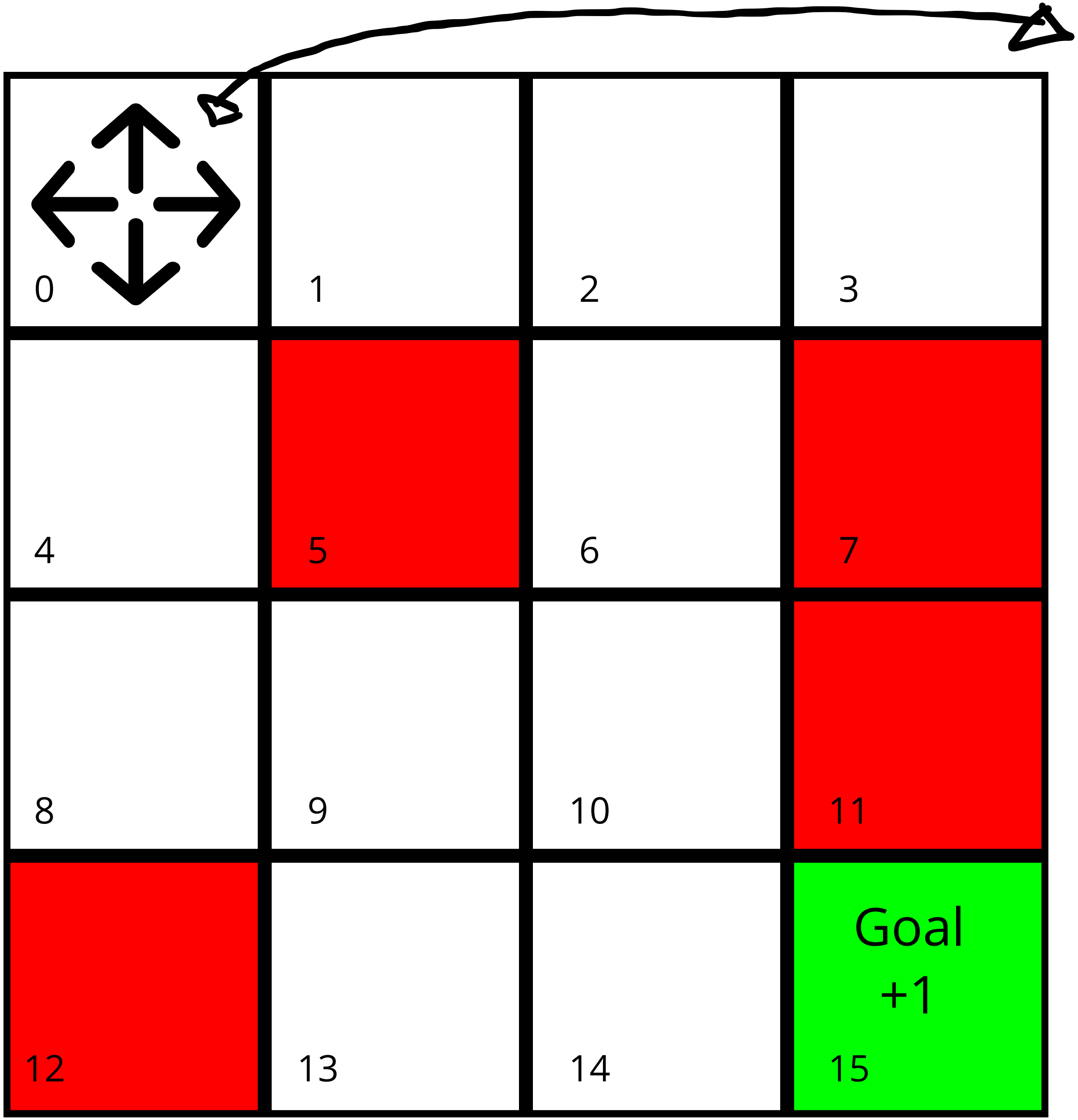
Start 0	1	2	3
4	🟥 5	6	🟥 7
8	9	10	🟥 11
🟥 12	13	14	🟩 Goal +1 15

1 Initial state

5 Terminal states

Speaker notes

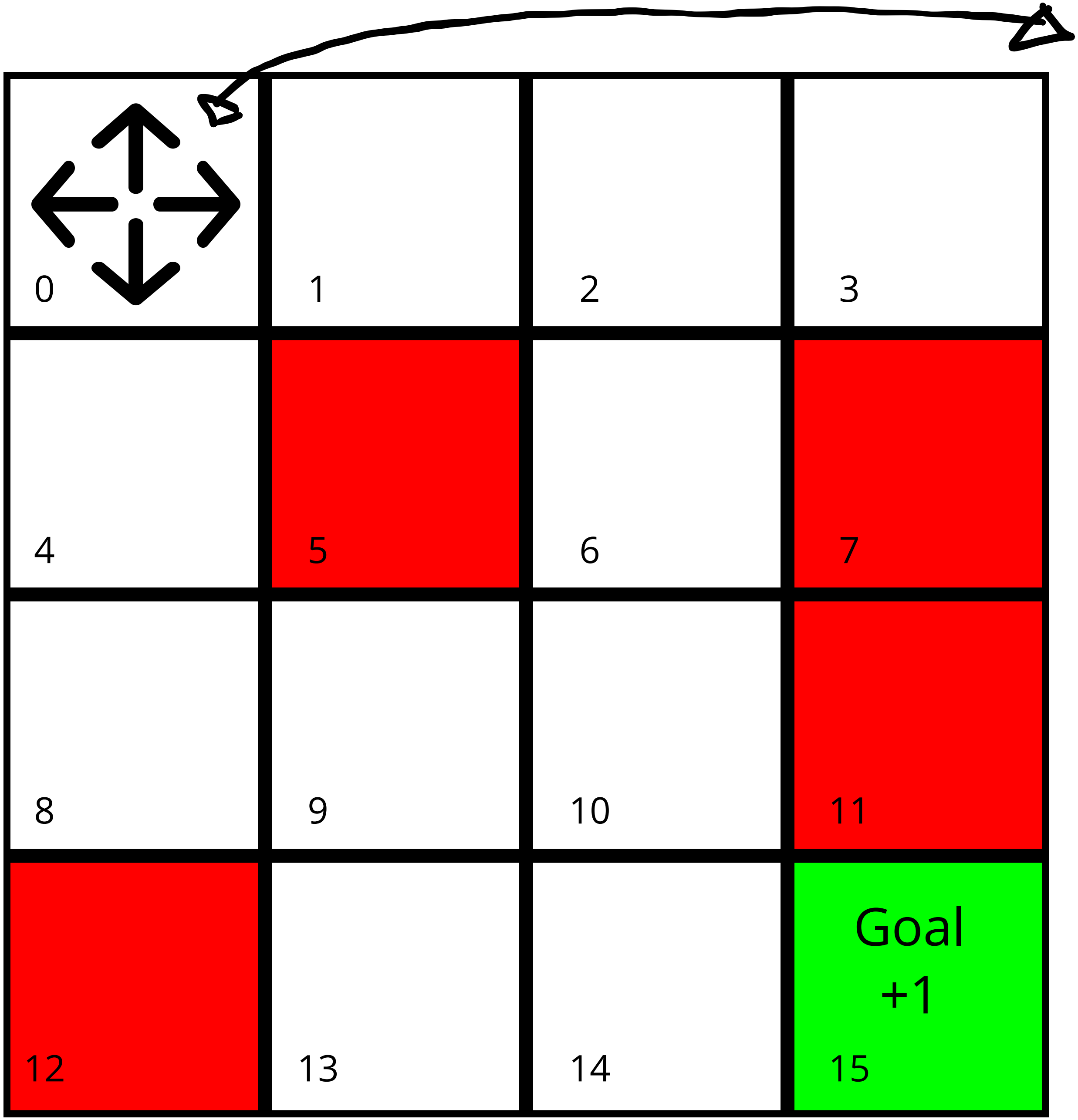
and a total of 5 terminal states



Action space; 4
Possible actions
(Left, Down, Right
or Up)

Speaker notes

The action space contains 4 different actions, 0, 1, 2, and 3, that map to left, down, right, and up.

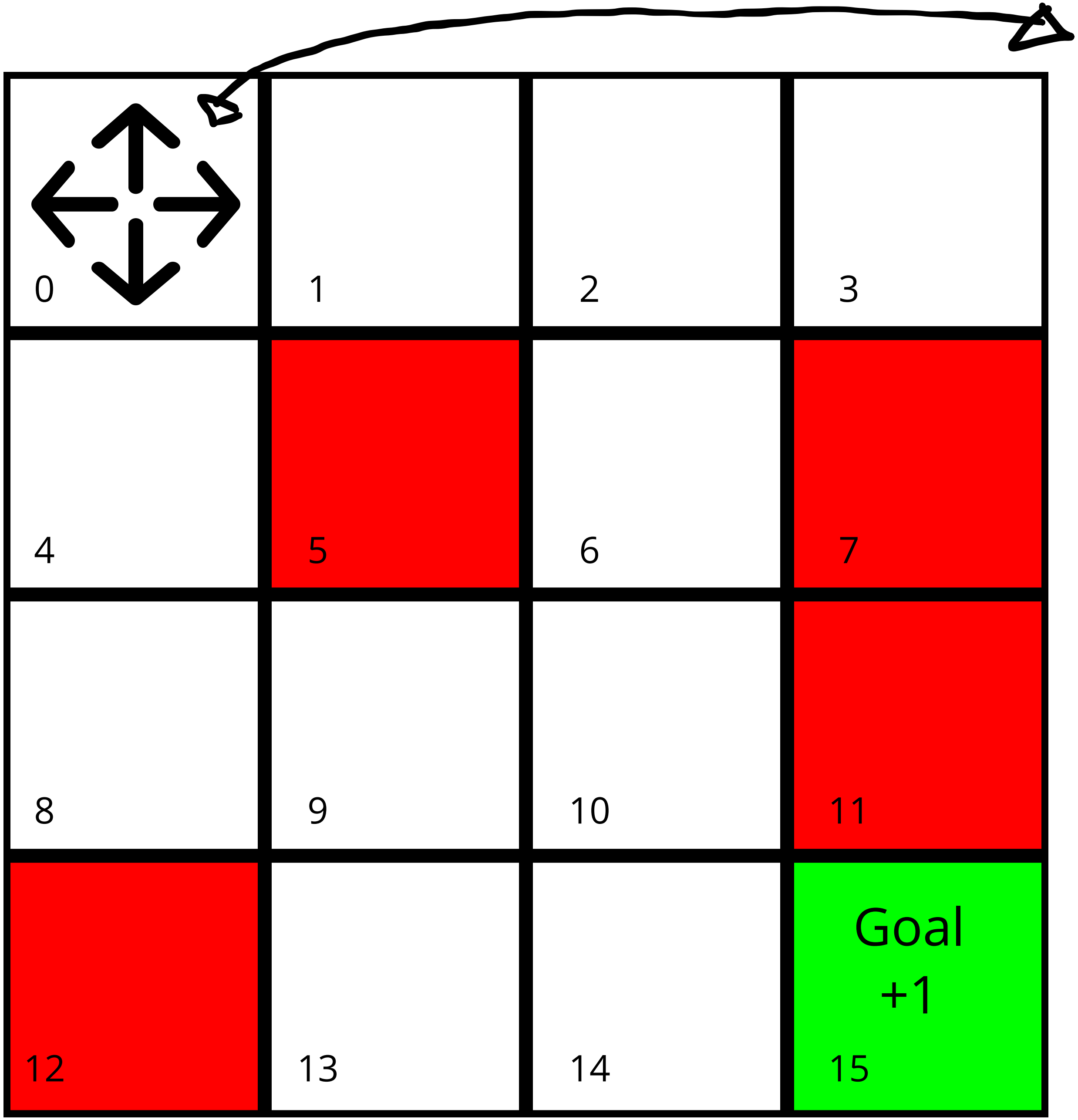


Action space; 4 Possible actions (Left, Down, Right or Up)

Stochastic environment; 66% chance of slipping and ending up ∓90° of intended action.

Speaker notes

The environment is also stochastic; the agent has a chance of slipping on the ice when moving, the chance is 33% chance of slipping +90 degrees of the intended movement, and a 33% chance of moving -90 degrees of the intended movement.



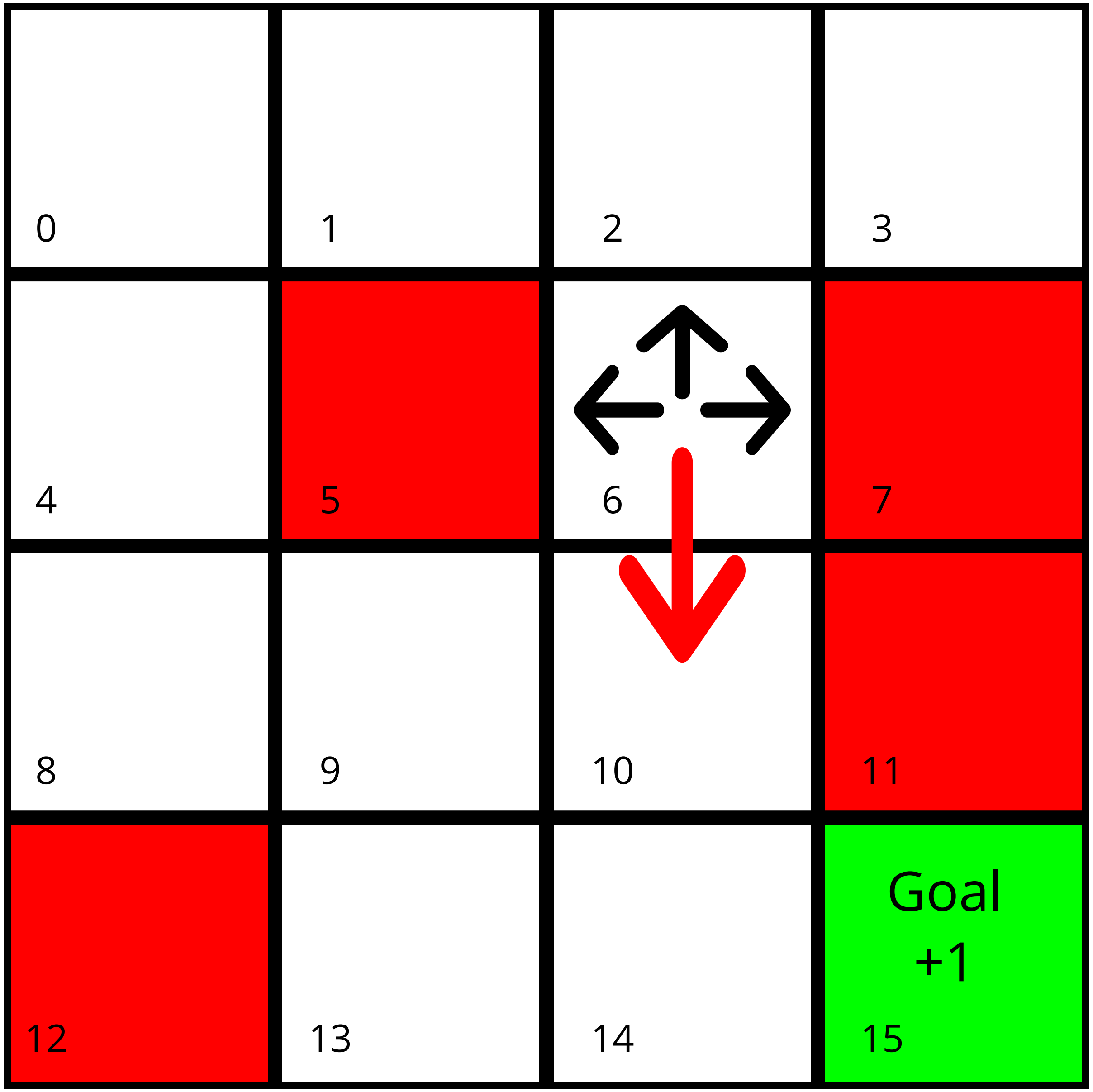
Action space; 4 Possible actions (Left, Down, Right or Up)

Stochastic environment; 66% chance of slipping and ending up ∓90° of intended action.

Walls; Walking into a wall will put you back to your original position

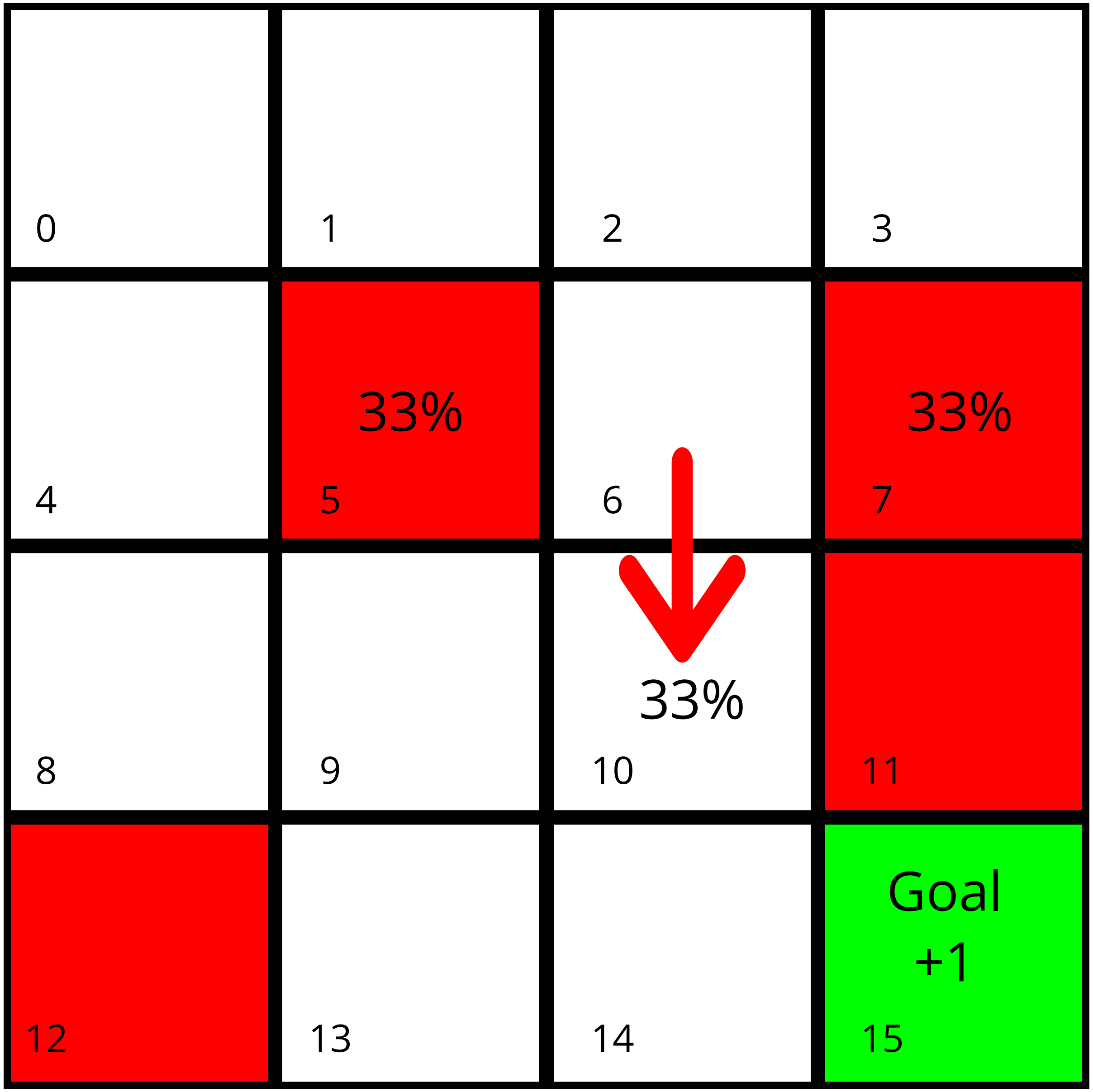
Speaker notes

There are also walls, so if you intentionally move into a wall, or slip and move into a wall, you will not move.


0
1
2
3
4
5
6
7
8
9
10
11
Goal
+1
12
13
14
15

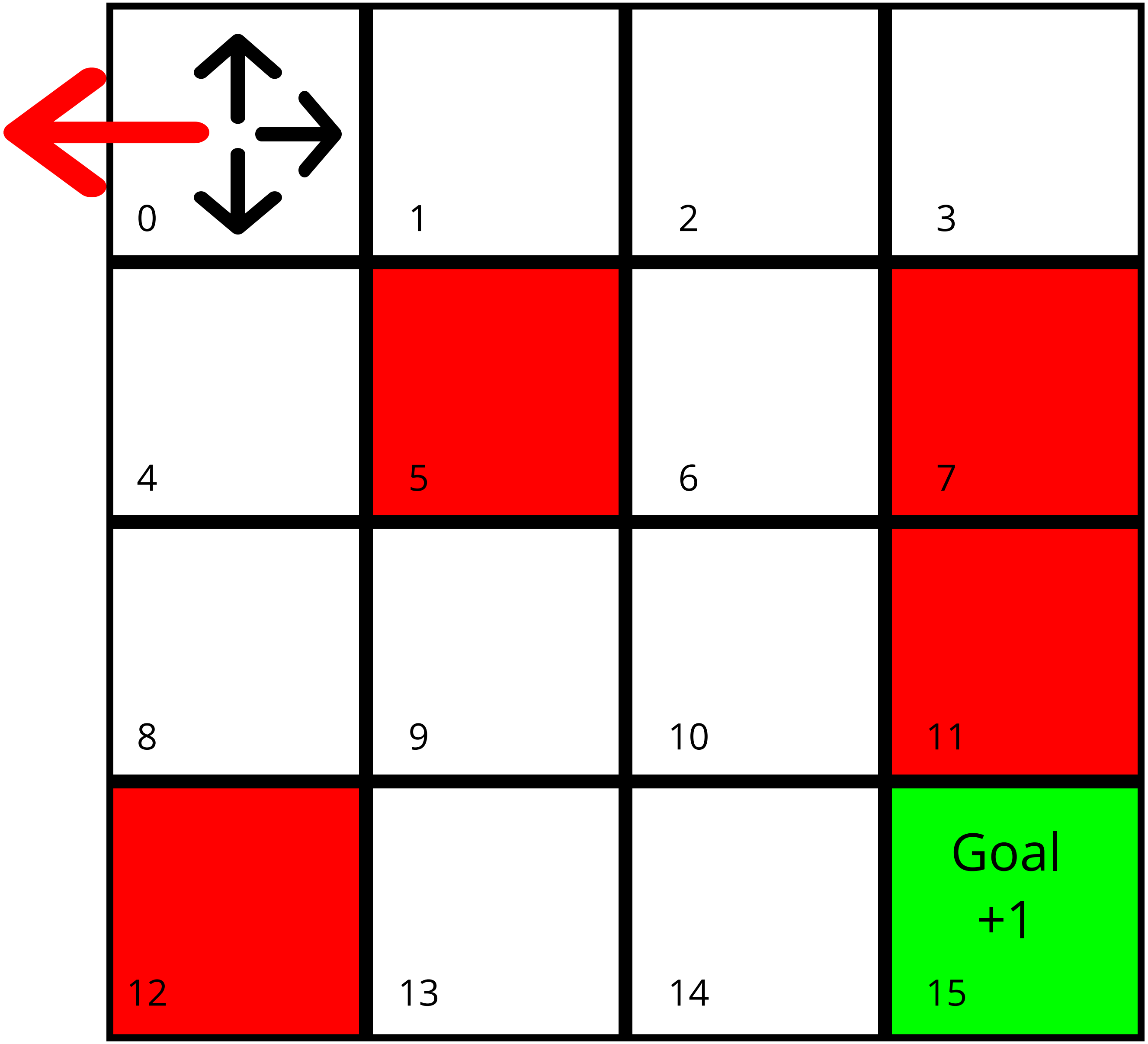
Speaker notes

So for example, if we are standing in tile 6 and move down:


0
1
2
3
33%
33%
4
5
6
7
33%
8
9
10
11
Goal
+1
12
13
14
15

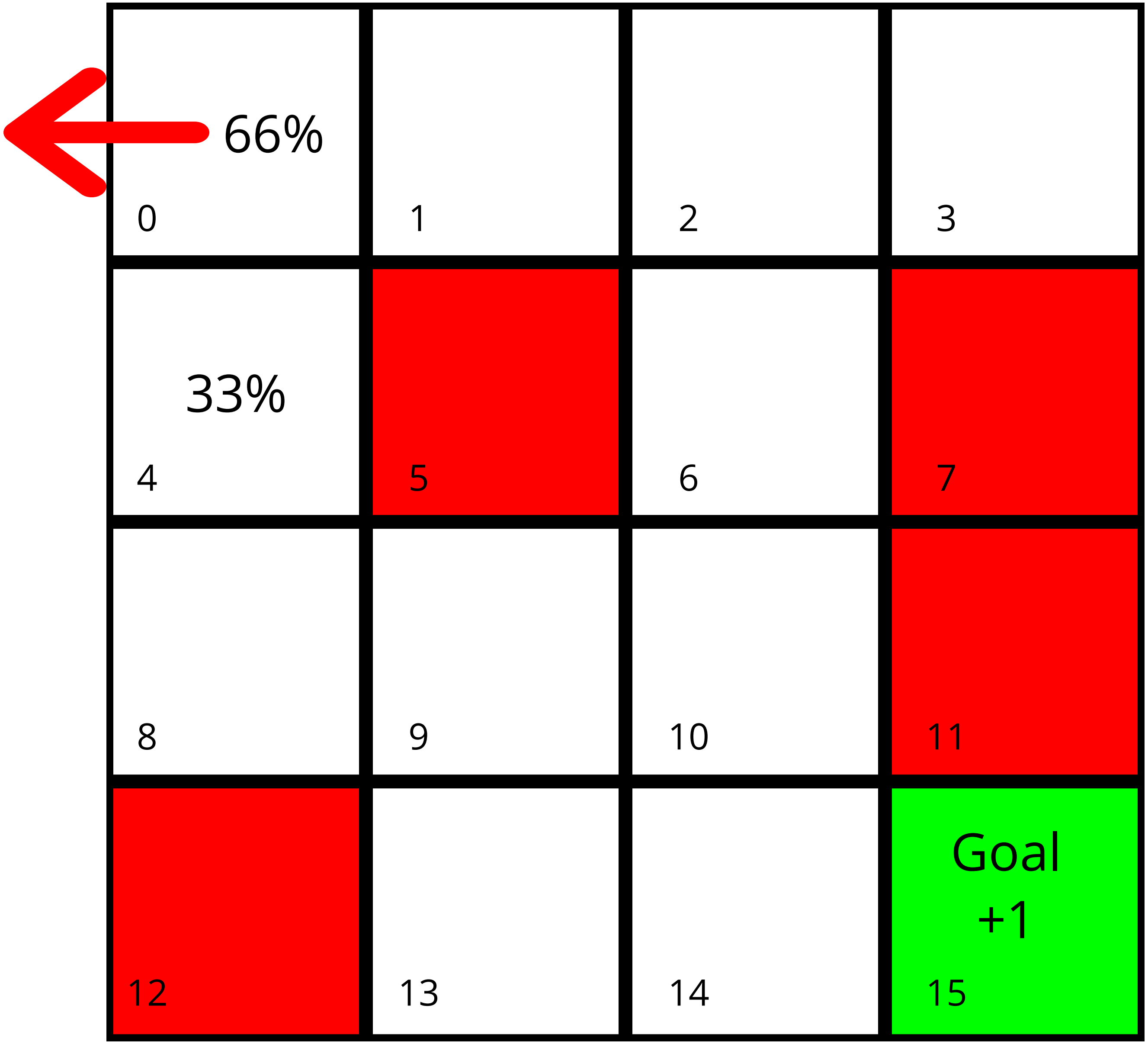
Speaker notes

We have a 33 percent chance of moving to the intended tile 10, 33% chance of slipping and moving to tile 5, and another 33% chance of slipping while moving to tile 7, in other words, a 66% chance of ending the game without receiving any rewards.


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Goal
+1
15

Speaker notes

Another example is if you are standing in state 0 and moving left


66%
0
1
2
3
33%
4
5
6
7
8
9
10
11
12
13
14
Goal
+1
15

Speaker notes

Here you will have a 66 percent chance of not moving at all, and a 33% chance of moving down to tile 4.

Properties of FrozenLake:

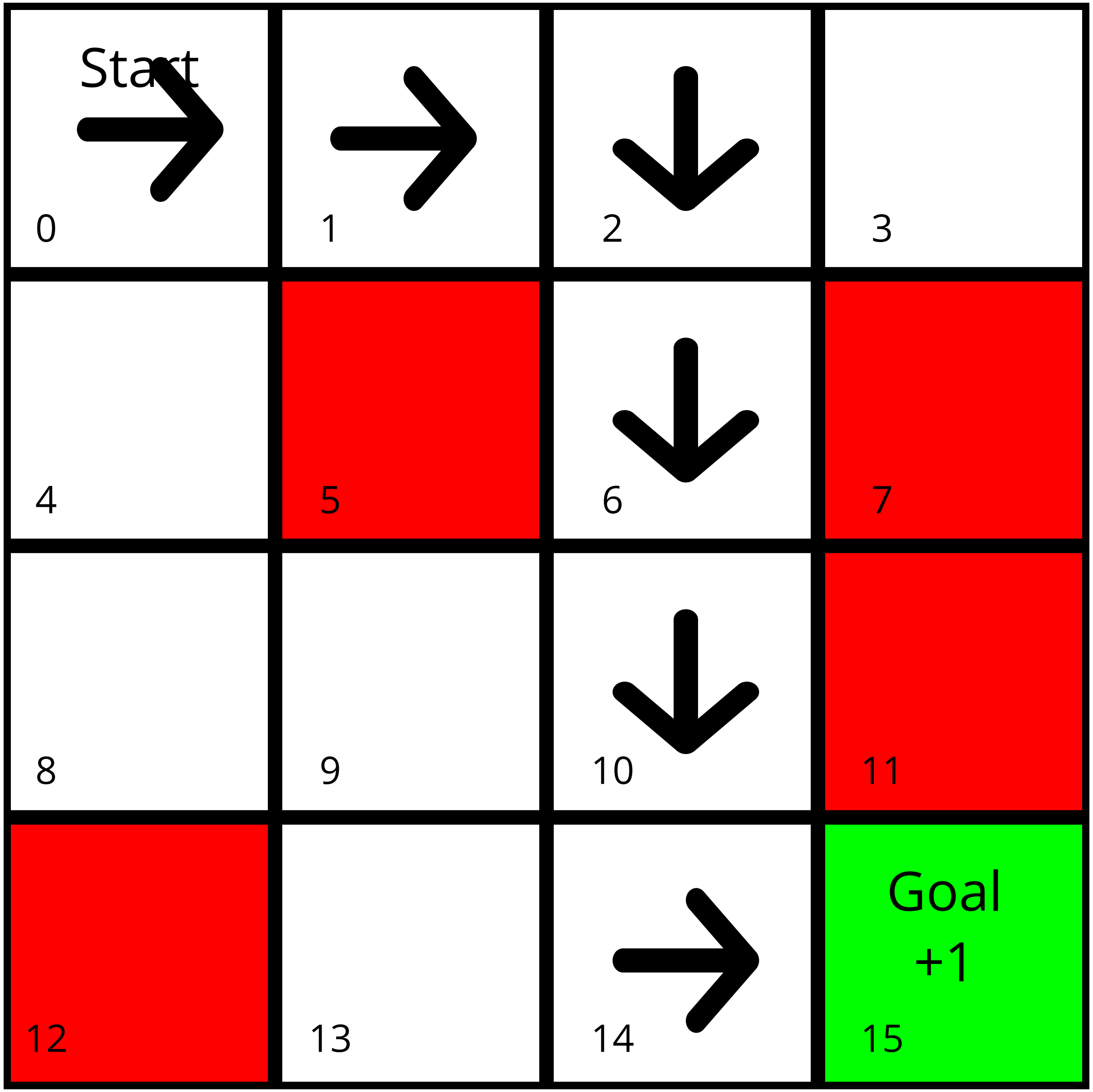
- Sequential decision problem
- Stochastic Environment
- Fully observable
- with a Markovian Transition model
- with additive rewards

These properties allow it to be modelled using the mathematical model known as Markov Decision Process (MDP)

Speaker notes

This environment has a list of properties that makes it perfect to be modeled using the mathematical framework known as a Markov decision process, or MDP for short;

Note; for now we will in general only consider environments with these properties, including being fully observable.


Start
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Goal
+1
15

Speaker notes

Since the environment is stochastic, a simple search algorithm that for example gives a plan such as this, would not work very well,
as you would very often end up in a state without a connected action.

A Markov Decision Process (MDP) is defined by:

- A set of states $s \in S$

↕

A *state* is a unique and self-contained configuration of the environment

A Markov Decision Process (MDP) is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$

an action is what the agent does to affect the environment.

A Markov Decision Process (MDP) is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition model $P(s' | s, a)$

Probability that action a taken from state s leads to s'

We expect the sum of the probabilities across all possible next states to sum to 1

$$\sum_{s' \in S} p(s' | s, a) = 1$$

Speaker notes

Since we use stochastic environments, we use the red function here for the transition model, i.e. the probability that action a taken from state s leads to state s mark.

A Markov Decision Process (MDP) is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition model $P(s' | s, a)$

The transition function should be *Markovian*.

Action outcomes only depend on the current state.

$$p(s_{t+1} | s_t, a_t) = p(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots)$$

Speaker notes

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

For MDPs, this means that the outcome of an action only depends on the current state, and not the preceding states and/or actions.

A Markov Decision Process (MDP) is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition model $P(s' | s, a)$
- A reward function $R(s, a, s')$

the reward is a measure of how well the agent is doing.

A Markov Decision Process (MDP) is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition model $P(s' | s, a)$
- A reward function $R(s, a, s')$

- A start state S_0

A Markov Decision Process (MDP) is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition model $P(s'|s,a)$
- A reward function $R(s,a,s')$
- A start state S_0

Sometimes

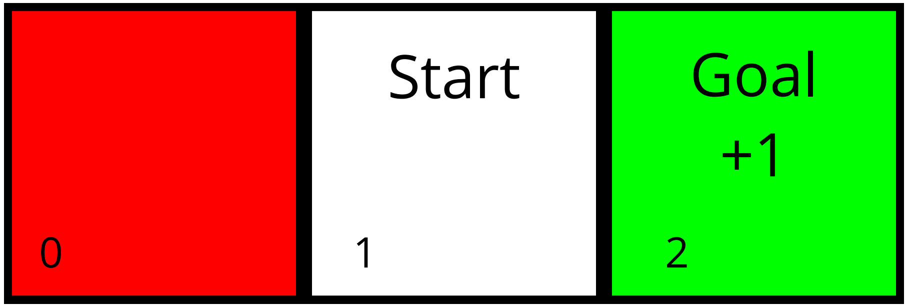
- A discount factor γ
- The horizon H

MDP's can be considered non-deterministic search problems

Speaker notes

Sometimes a discount factor gamma and a horizon H, the book does not mention them as a part of the MDP properties.
We will get back to them.

Graphical representations of MDPs



Start

Goal

+1

0

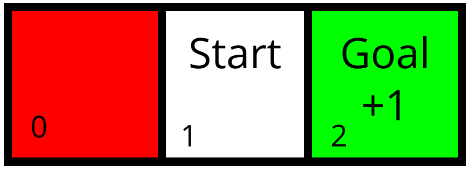
1

2

Speaker notes

We can also make simple graphical representations of very simple MDPs

Here we make a very small Frozen Lake environment with 3 tiles.



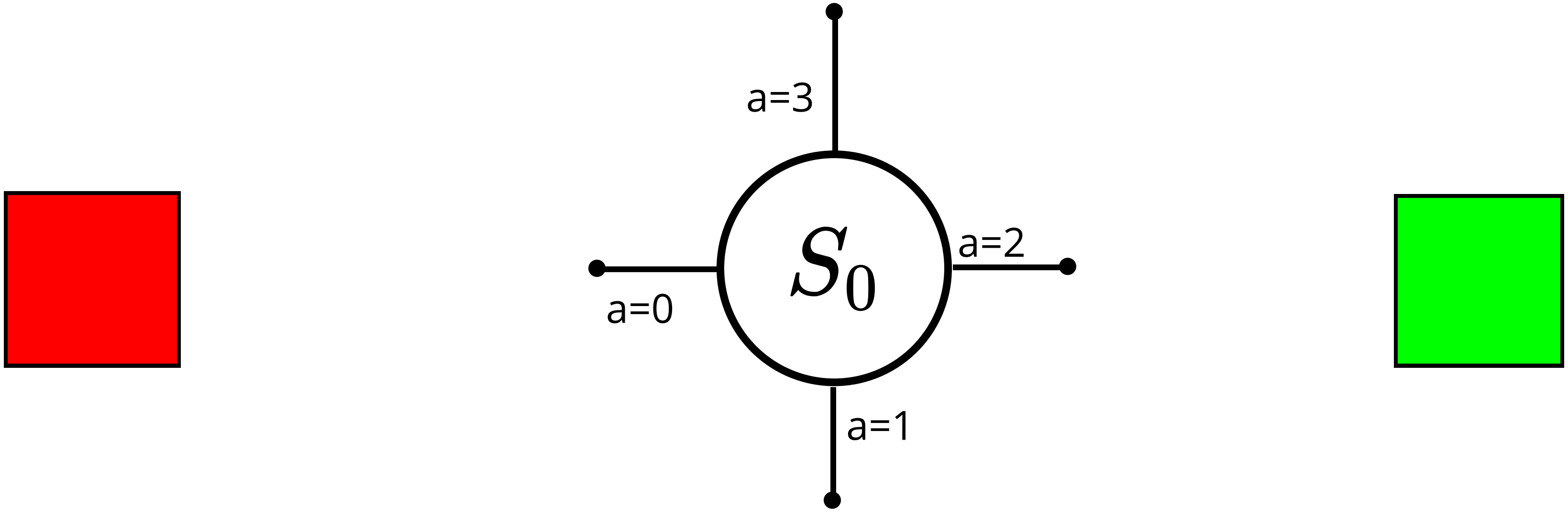
Actions:

Left = 0

Down = 1

Right = 2

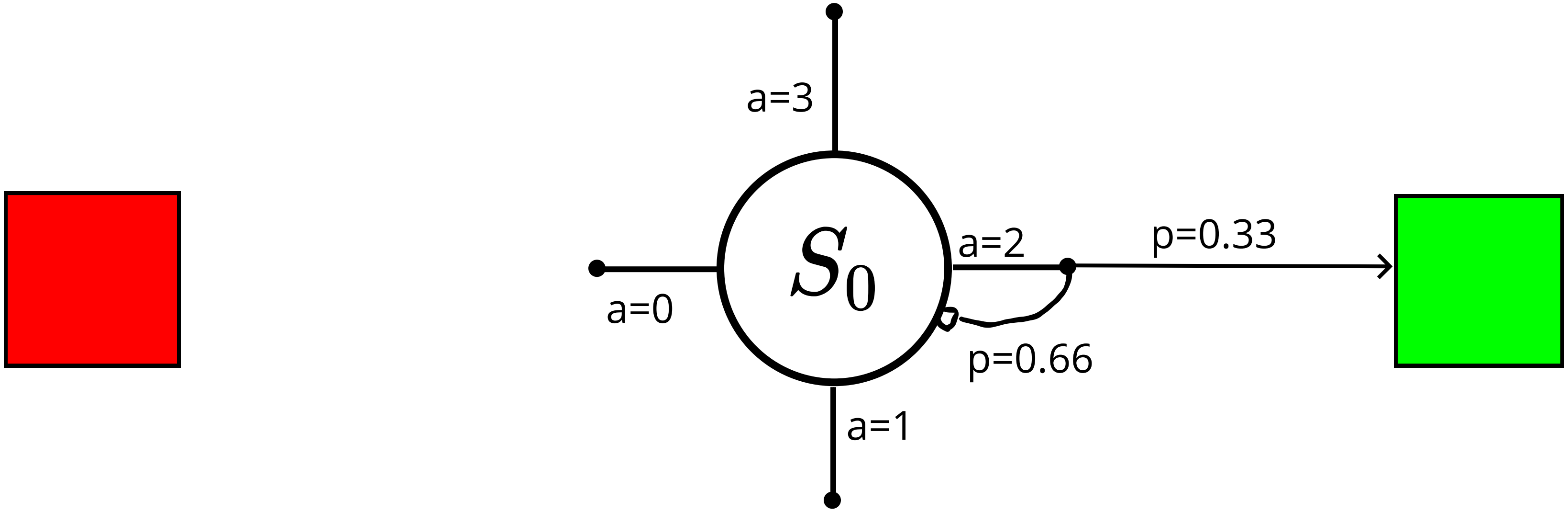
Up = 3



Speaker notes

We can start by drawing up the states and the possible actions from the states, I use circles as normal states and squares as terminal states.
I also mark the initial state with S zero.

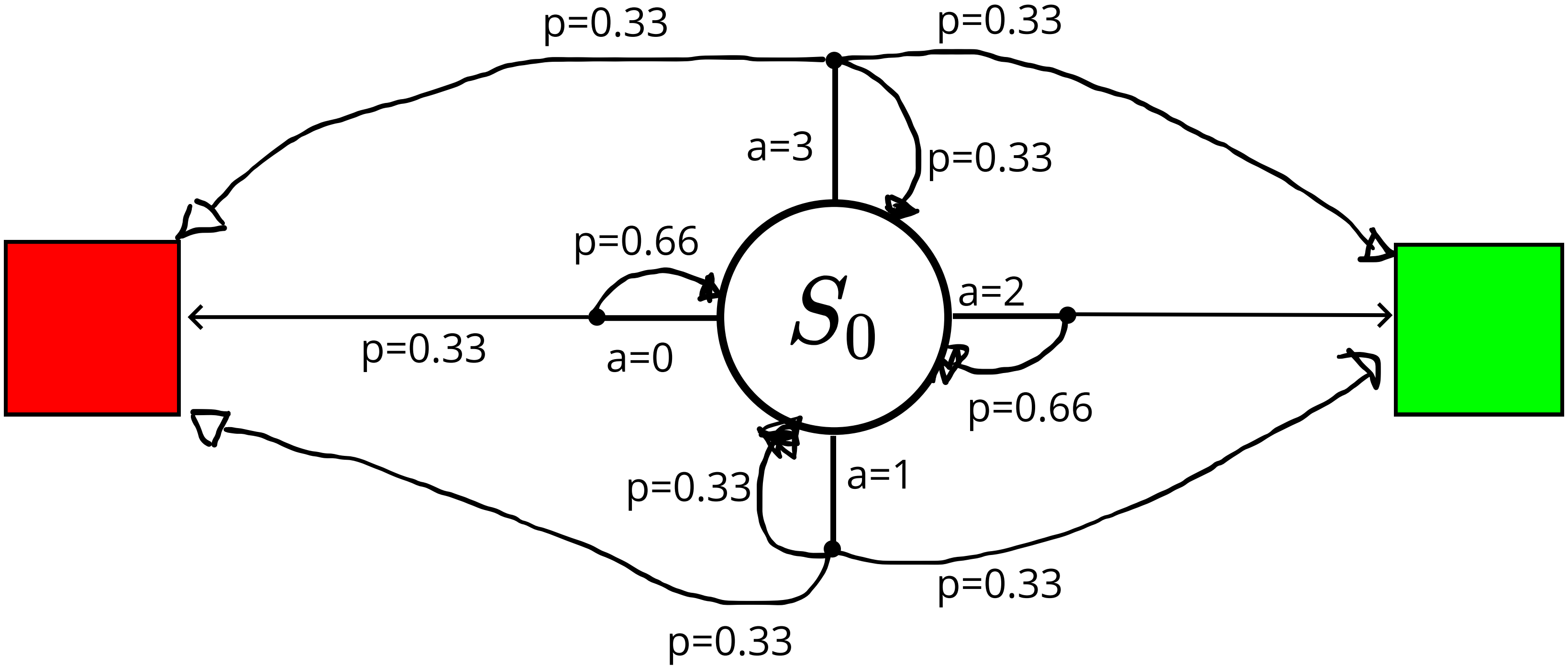
🟥 0	Start 1	🟩 Goal +1 2



Speaker notes

Here I have added the transition model from S zero given action 2 (or right), where it has a 66% chance of returning to S zero, and a 33% chance of reaching the terminal state on the right.

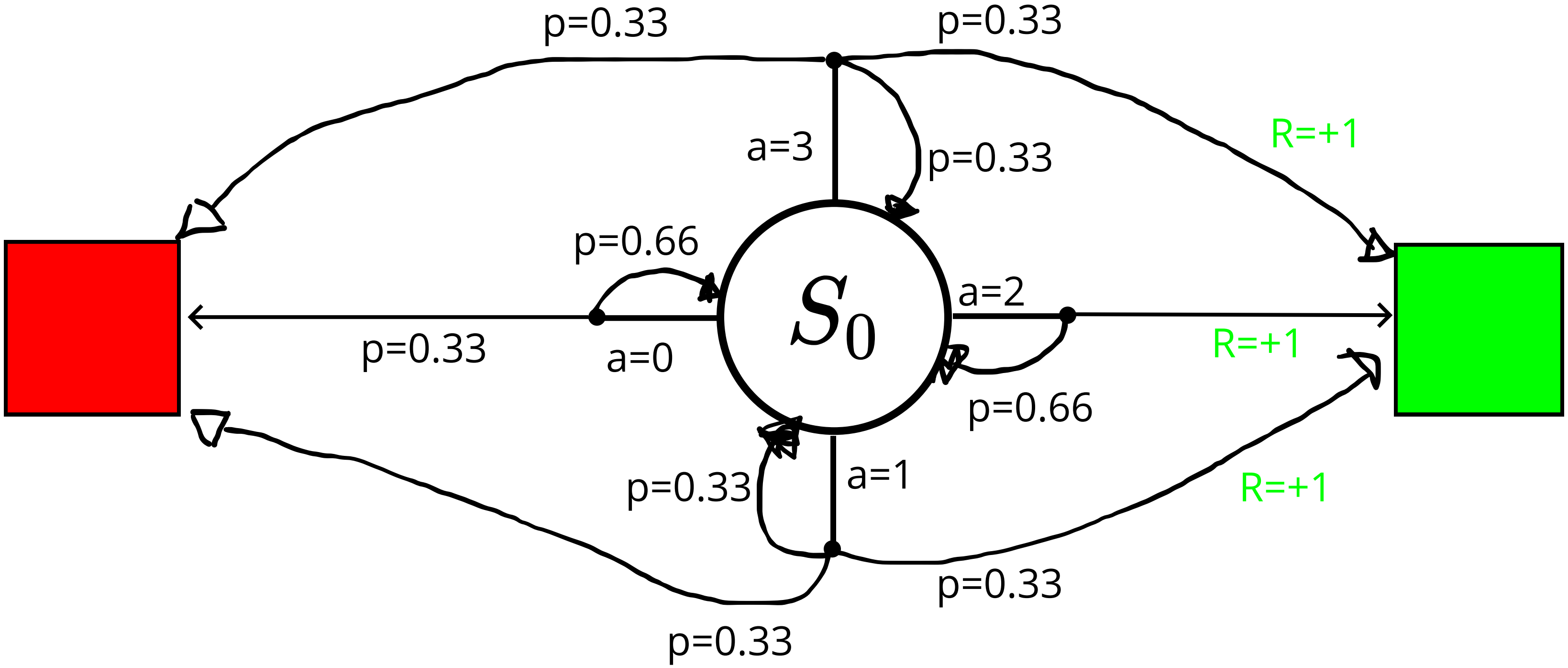
🟥 0	Start 1	🟩 Goal +1 2



Speaker notes

By repeating for all actions we end up with this

🟥 0	Start 1	🟩 Goal +1 2



Speaker notes

And if we add rewards as well, we have added all the properties of the MDP.

Solving the MDP: The RL agent

An RL agent may include one or more of the following components:

- Policy: Agents' plan/behavior.
- Utility function: A function to evaluate how good/bad a state is.
- Model: A model predicts what the environment will do next.

RL methods can be classified as either utility-based or policy-based and as either model-based or model-free.

Speaker notes

To consider how we can solve the MDP, let's first go back to the components of the RL agent, we can consider only model-free ones for now, but let us take a deeper look at the policy and soon the utility function.

Policy

Recall: "A policy defines an agents' behavior"

It is a map from a state to an action, and is denoted π

For MDPs we want a plan for **all** possible states.

Formally:

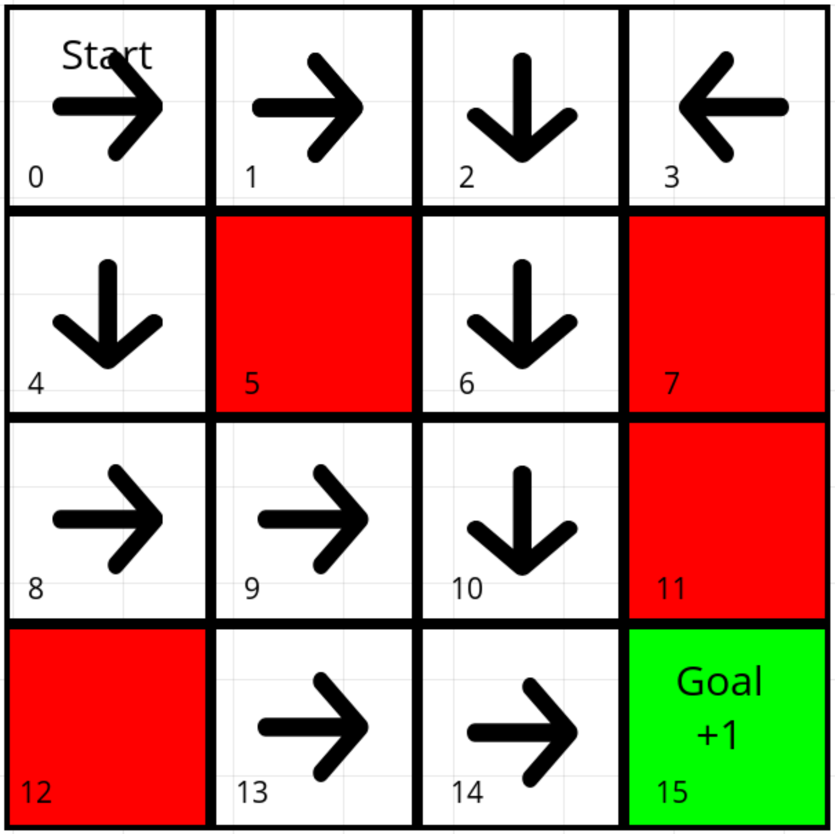
A policy: $\pi : S \rightarrow A$

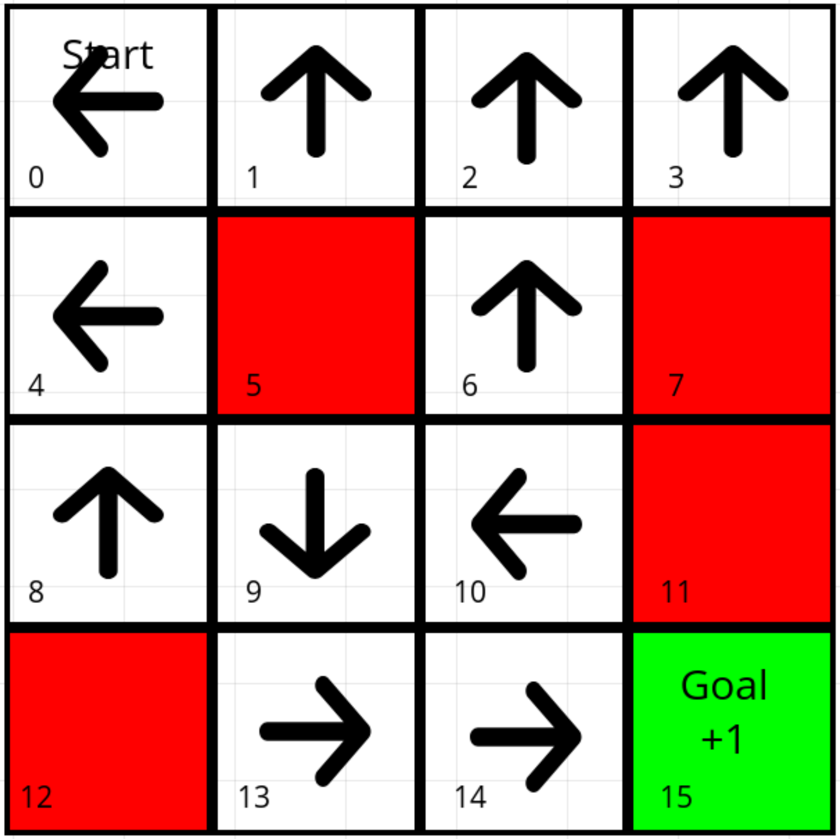
gives an action for each state

An optimal policy: $\pi^* : S \rightarrow A$

gives an action for each state that maximizes the expected utility.

Example policies





Speaker notes

For our simple grid-like environment, it is easy to create a policy by hand, simply by making a map from all possible states to some action.

Any immediate favors among the two examples?

Utility Value

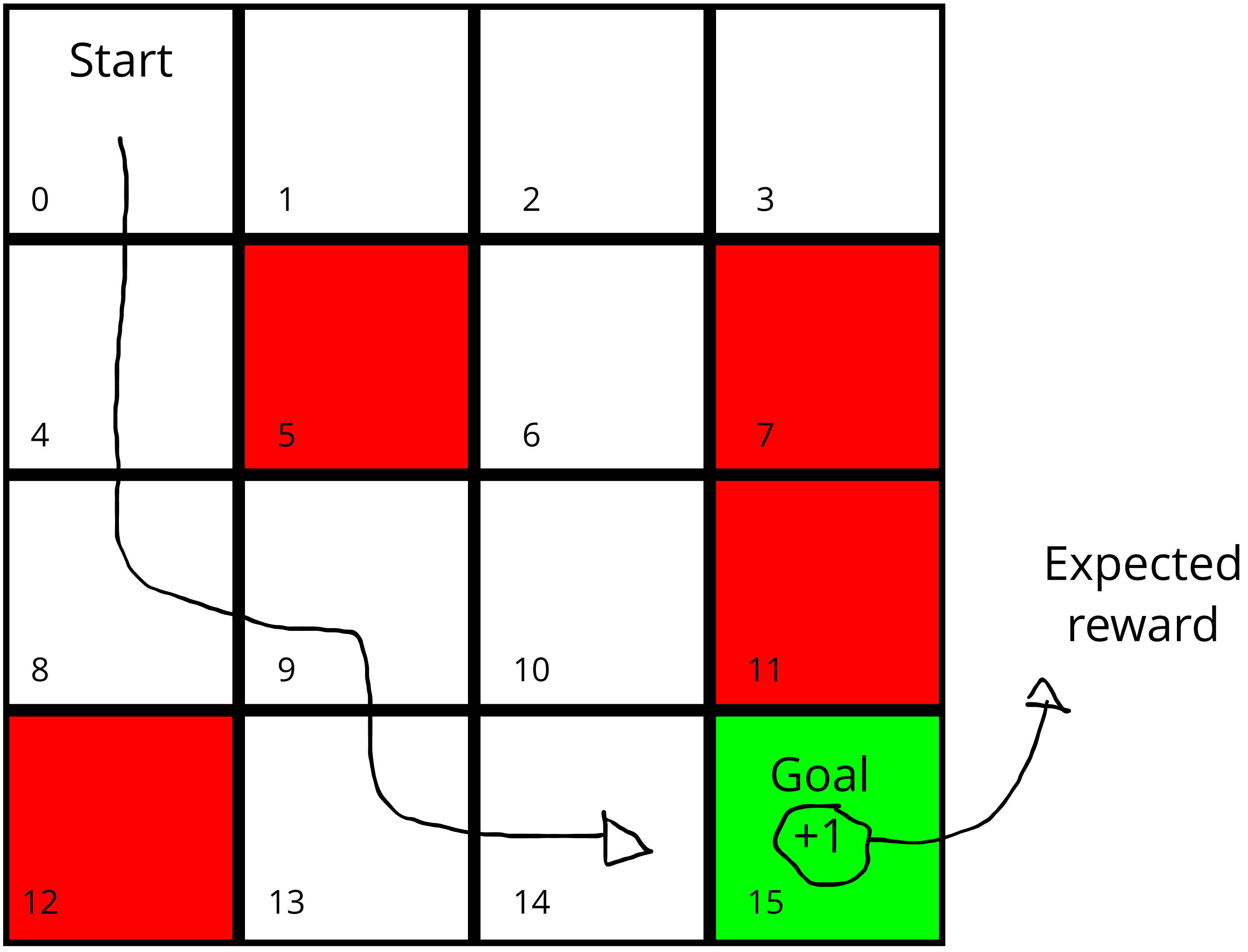
In Reinforcement Learning, the utility is an expectation of the long-term reward.

The utility can be used to:

- Evaluate the desirability of states
- Select between different actions

Speaker notes

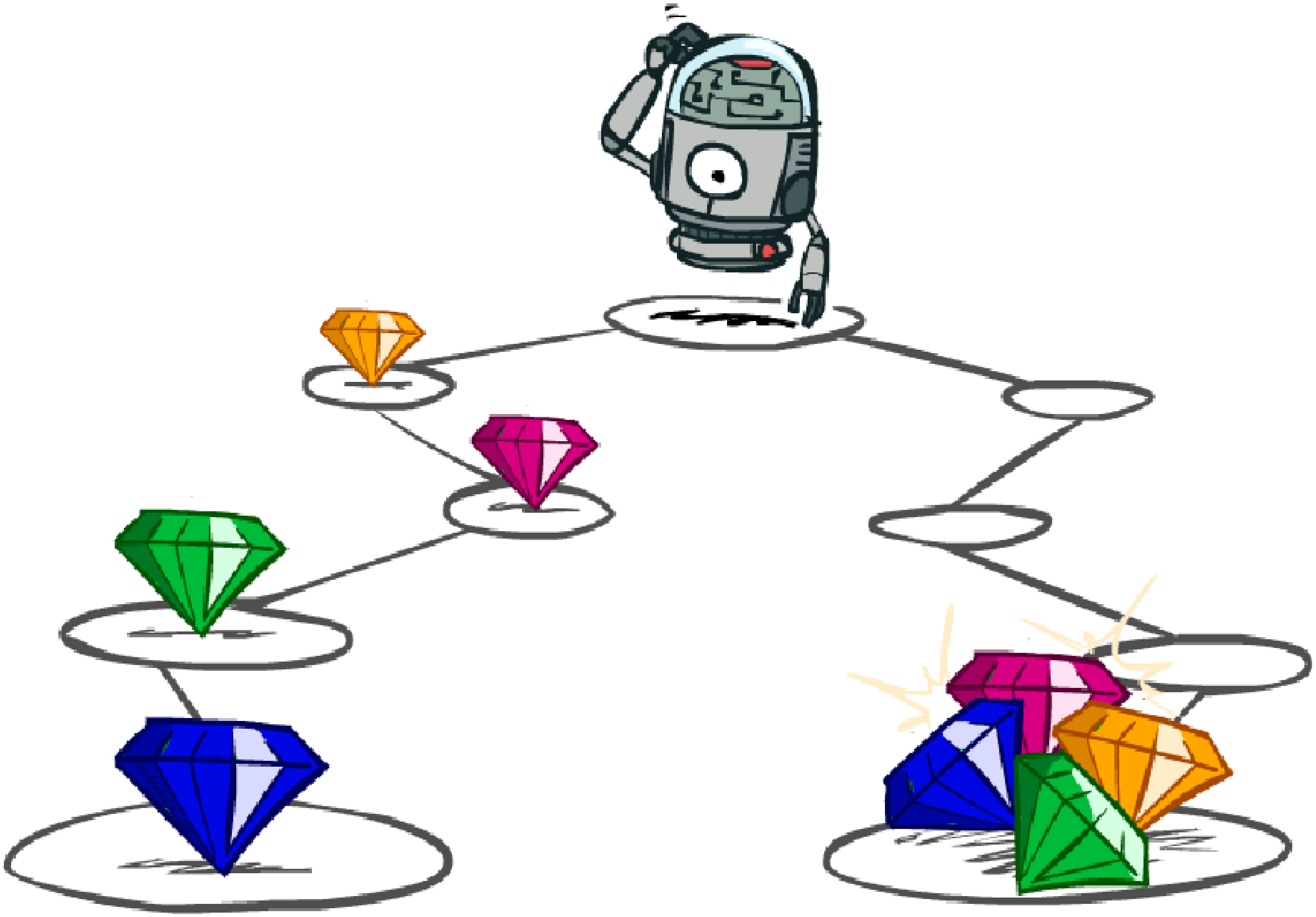
Now that we know more of what a policy is, let us go over the utility and the utility function.


Start
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Goal
+1
15
Expected
reward

Speaker notes

As an example, in our Frozen Lake environment, if we remove the stochasticity, and have a plan to follow the arrow to the goal, we expect to sometime in the future receive a reward of 1.
So the utility, or, expected reward for following that policy, would be 1 for all the states covered by the policy.
If we add back the randomness, however, this expectation will reduce drastically.

Utilities of Sequences



Speaker notes

We also have environments that give multiple rewards over a sequence of states and actions. This, however, could also make it more difficult for us to prioritize.

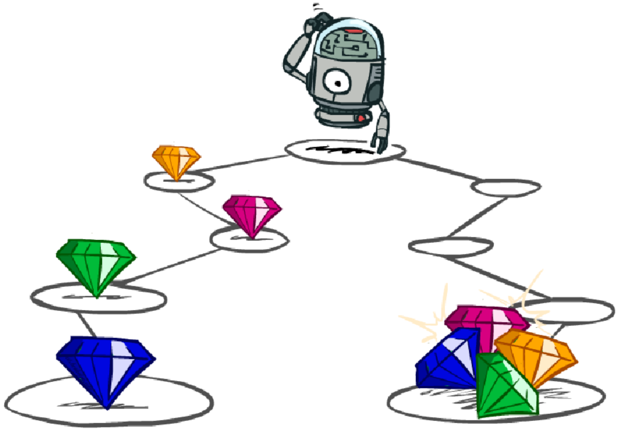
Utlities of Sequences

What preferences should an agent have
over reward sequences?

- More or less? [1,2,2] vs [2,3,4]

  What if more increases risk?

- Now or later? [0,0,1] vs [1,0,0]



Utlities of Sequences

Speaker notes

Here is a small clip of a RL agent in a boat-racing game.

Normally one would assume that the goal of the game is to race the other boats and try to finish first,
but the agent instead decides to drive in the opposite direction and go in circles, as this is what gave it the most rewards.

Utlities of Sequences

What preferences should an agent have
over reward sequences?

- More or less? [1,2,2] vs [2,3,4]

  What if more increases risk?

- Now or later? [0,0,1] vs [1,0,0]

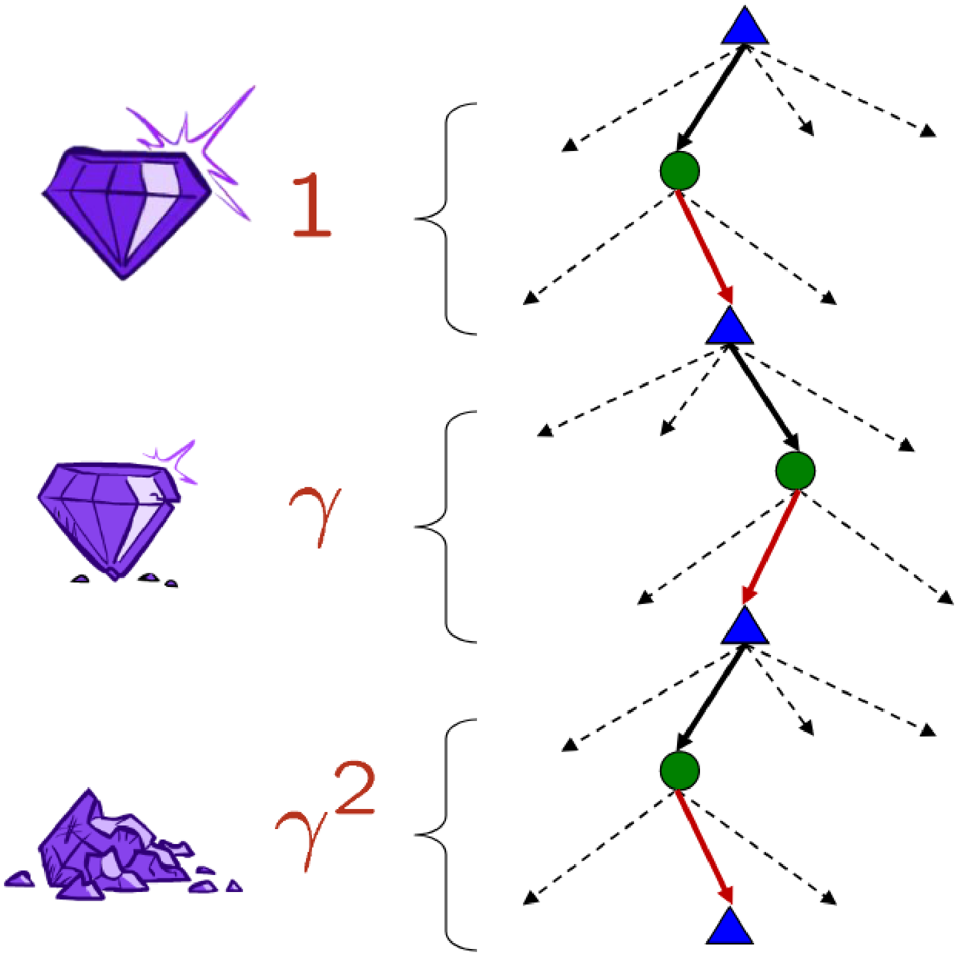
It is generally reasonable to maximize the sum of rewards.

It is also reasonable to prefer rewards now to rewards later.

One solution is to let the values decay exponentially
for every timestep by discounting the reward.

Discounting

- We introduce a discount factor $\gamma \in [0, 1]$
- The discount factor trades off the importance of immediate vs. long-term reward
- For each timestep, we multiply the discount once



Example, $\gamma = 0.5$:

$$U_h([1,2,3]) = 1 * \gamma^0 + 2 * \gamma^1 + 3 * \gamma^2$$

$$U_h([1,2,3]) = 1 * 1 + 2 * 0.5 + 3 * 0.25$$

$$U_h([1,2,3]) < U_h([3,2,1])$$

Finite/Infinite Horizon

Problem: What if the game lasts forever? Do we get infinite rewards?

A Solution: Finite Horizon:

- Terminate episodes after a fixed T steps (or set all rewards to 0)
- Policy π depends on how much time is left

A policy that depends on the time is called nonstationary.

A policy that does not depend on the time is called stationary.

Speaker notes

If the boat-game above had no set end-time, it would probably end up going forever, with the agent trying to get infinite rewards.

Discussion

Could you solve the problem with the boat agent above by changing the horizon or the discount value?

Assume:

- Getting first place would, on average, reward more points than looping for the entire game.
- Getting second place would, on average, reward fewer points than looping for the entire game.
- The current horizon is set to when the second-to-last player reaches the goal + 100 steps.

Speaker notes

Note: This is mean to make the students think about the discount and reward, the problem is not easy to solve without modifying the reward.
It might be worth testing a reduced horizon, set to e.g. some "average" race time.

Additive Discounted Rewards

Imagine that you are following some policy π, thus producing a trajectory of states, actions, and rewards $(s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, ...)$

We can calculate the utility of a trajectory using additive discounted rewards:

$$U_h([s_0, a_0, s_1, a_1, s_2, ...]) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + ...$$

$$= \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$$

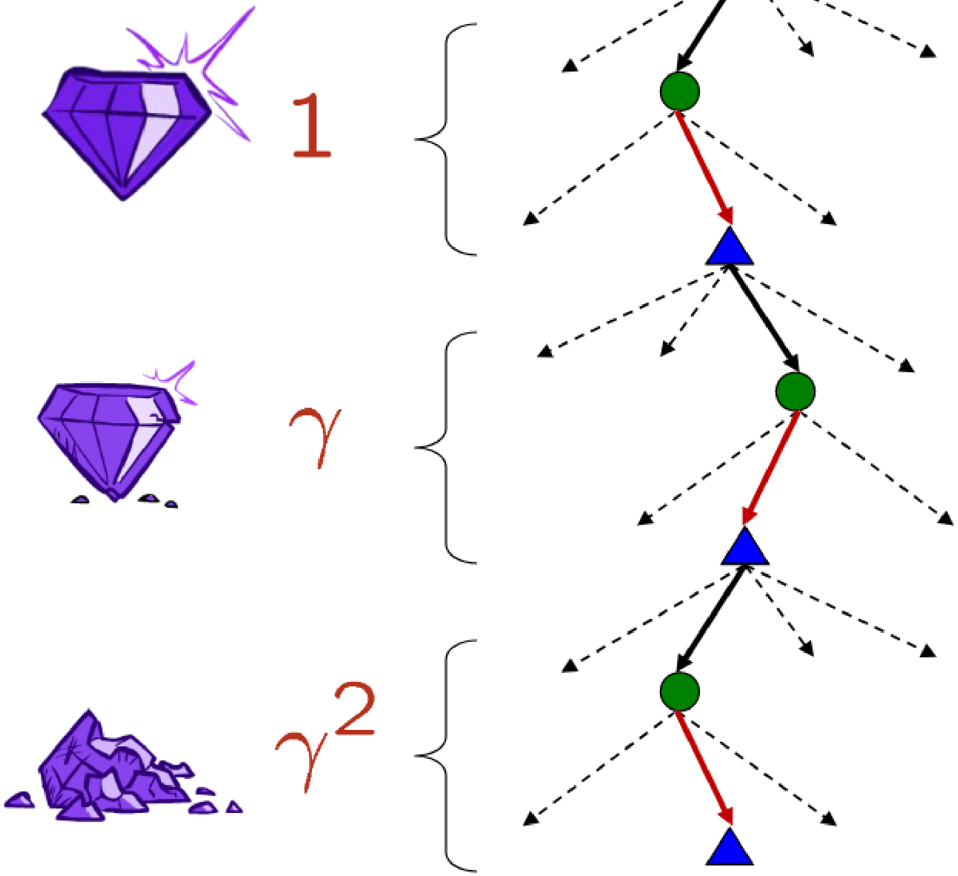
Additive Discounted Rewards

Example, $\gamma = 0.5$:

$$U_h([1,2,3]) = 1 * \gamma^0 + 2 * \gamma^1 + 3 * \gamma^2$$

$$U_h([1,2,3]) = 1 * 1 + 2 * 0.5 + 3 * 0.25$$

$$U_h([1,2,3]) < U_h([3,2,1])$$



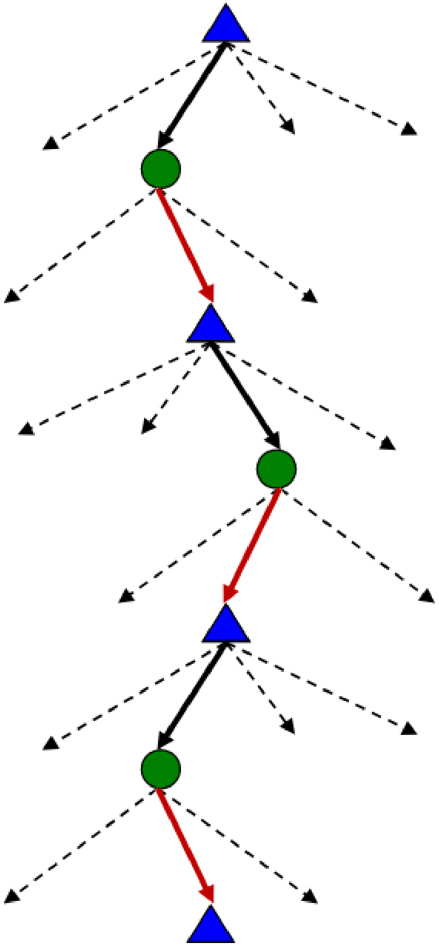
Utility Function

The expected reward from following a policy π starting in state s is given by the function

$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})]$$

Here the expectation E is with respect to the probability distribution over the state sequences determined by s and π

S_t is the state the agent reaches by time t by following the policy π



Speaker notes

In the previous slide, we calculated the utility of a historical sequence of states, actions, and rewards.
As we work in stochastic environments, there is no guarantee that the sequence would be identical if we repeated the same actions from the same starting state, so we need a better function to calculate utility.

Utility Function

The utility of a state is the expected reward for the next step plus the discounted utility of the subsequent state, assuming that the agent chooses the optimal action.

Utility Function:

$$U^{\pi^*}(s) = U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$

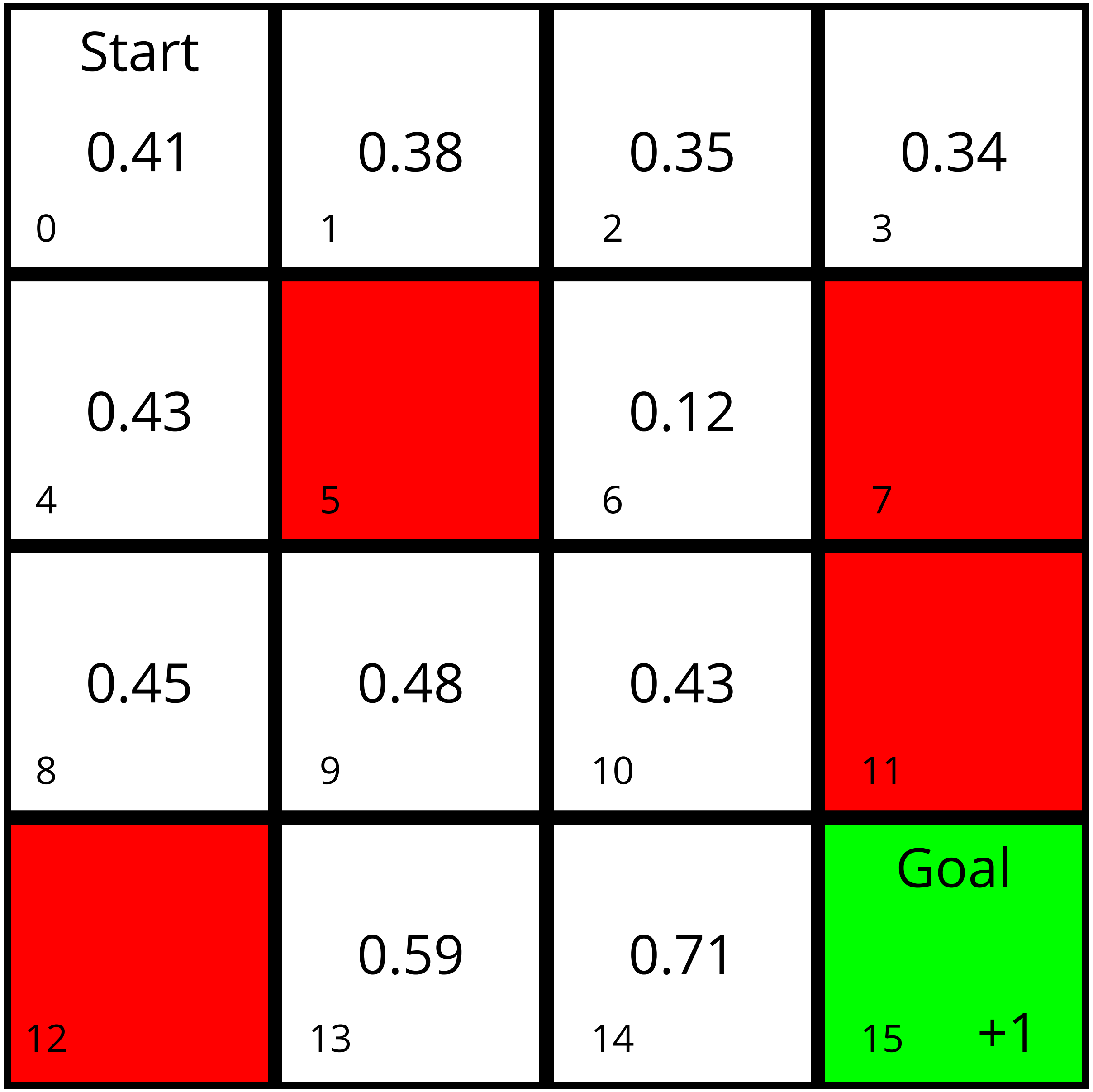
This is also called a Bellman Equation, after Richard E. Bellman

Optimal policy:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$

Speaker notes

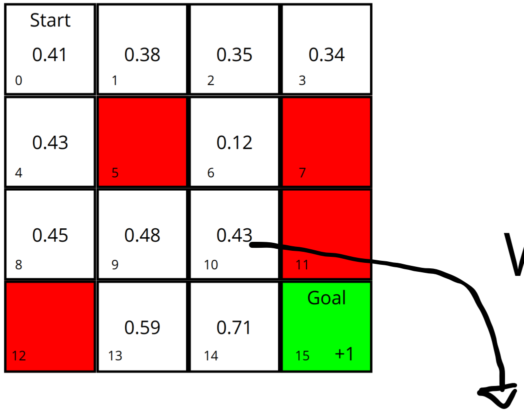
These are part of the exercises for today and quite important to know and understand.



Start 0.41 0	0.38 1	0.35 2	0.34 3
0.43 4	🟥 5	0.12 6	🟥 7
0.45 8	0.48 9	0.43 10	🟥 11
🟥 12	0.59 13	0.71 14	🟩 Goal 15 +1

Speaker notes

As an example, here is the Frozen Lake, with some pre-calculated utilities.



Start 0.41 0	0.38 1	0.35 2	0.34 3
0.43 4	🟥 5	0.12 6	🟥 7
0.45 8	0.48 9	0.43 10	🟥 11
🟥 12	0.59 13	0.71 14	🟩 Goal 15 +1

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$

With $\gamma = 0.9$

$$\begin{aligned} U(10) = max\{&[0.33(0+\gamma U(6)) + 0.33(0+\gamma U(9))0.33(0+\gamma U(14))], \\ &[0.33(0+\gamma U(9)) + 0.33(0+\gamma U(14))0.33(0+\gamma U(11))], \\ &[0.33(0+\gamma U(14)) + 0.33(0+\gamma U(11))0.33(0+\gamma U(6))], \\ &[0.33(0+\gamma U(11)) + 0.33(0+\gamma U(6))0.33(0+\gamma U(9))]\} \end{aligned}$$

$$U(10) = max\{[0.33(0.9 * 0.12) + 0.33(0.9 * 0.48)0.33(0.9 * 0.71)], ...\}$$

$$U(10) = max\{[0.39], [0.35], [0.25], [0.18]\}$$

$$U(10) = 0.39$$

$\pi^*(s) = 0$, which maps to "*left*"

Q-Function

(Action-utility function)

Q-Function w/r to the utility function:

$$U(s) = \max_a Q(s,a)$$

Optimal policy w/r the Q-function:

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

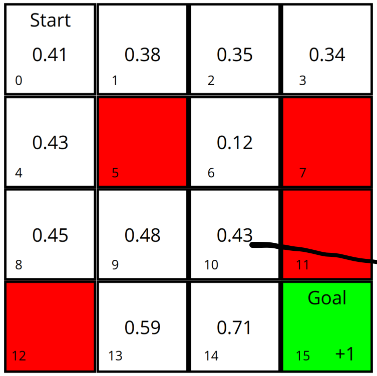
The Q-function as a Bellman Equation:

$$Q(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

Speaker notes

We will also define the Q-Function, or the action-utility function, which is used a lot in reinforcement learning

It is practically the utility function, but with a specific action instead of the assumption of taking the optimal action.



Start 0.41 0	0.38 1	0.35 2	0.34 3
0.43 4	🟥 5	0.12 6	🟥 7
0.45 8	0.48 9	0.43 10	🟥 11
🟥 12	0.59 13	0.71 14	Goal 15 +1

With $\gamma = 0.9$

$$Q(10,0) = 0.33(0 + 0.90.12) + 0.33(0 + 0.9 * 48) + 0.33(0 + 0.9 * 71)$$

$$= 0.39$$

Expected utility after taking action 0 (left) in state 10.

Utility Functions in Reinforcement Learning

- Agents often approximate utility functions.
- With an accurate utility function, we can behave optimally.
- With suitable approximations, we can act well, even in intractably large domains.
- We will discuss and learn some algorithms later in the course.

Speaker notes

Next week we will start taking a look at some algorithms and methods for approximation.

- Introduction to Reinforcement Learning

- Markov Decision Processes (MDPs)

→