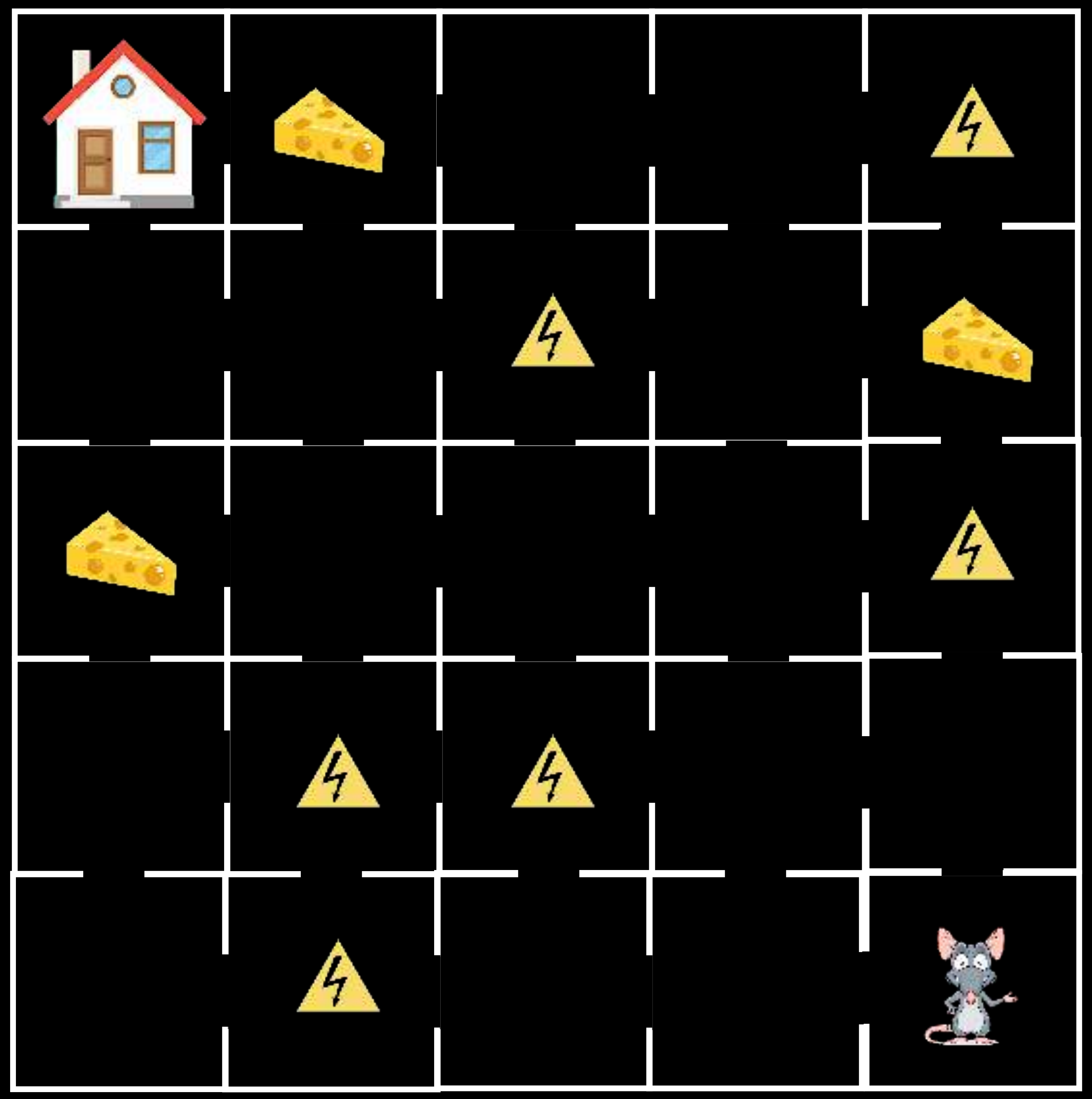
Lecture 32: Introduction to Reinforcement Learning 2

CS109B Data Science 2

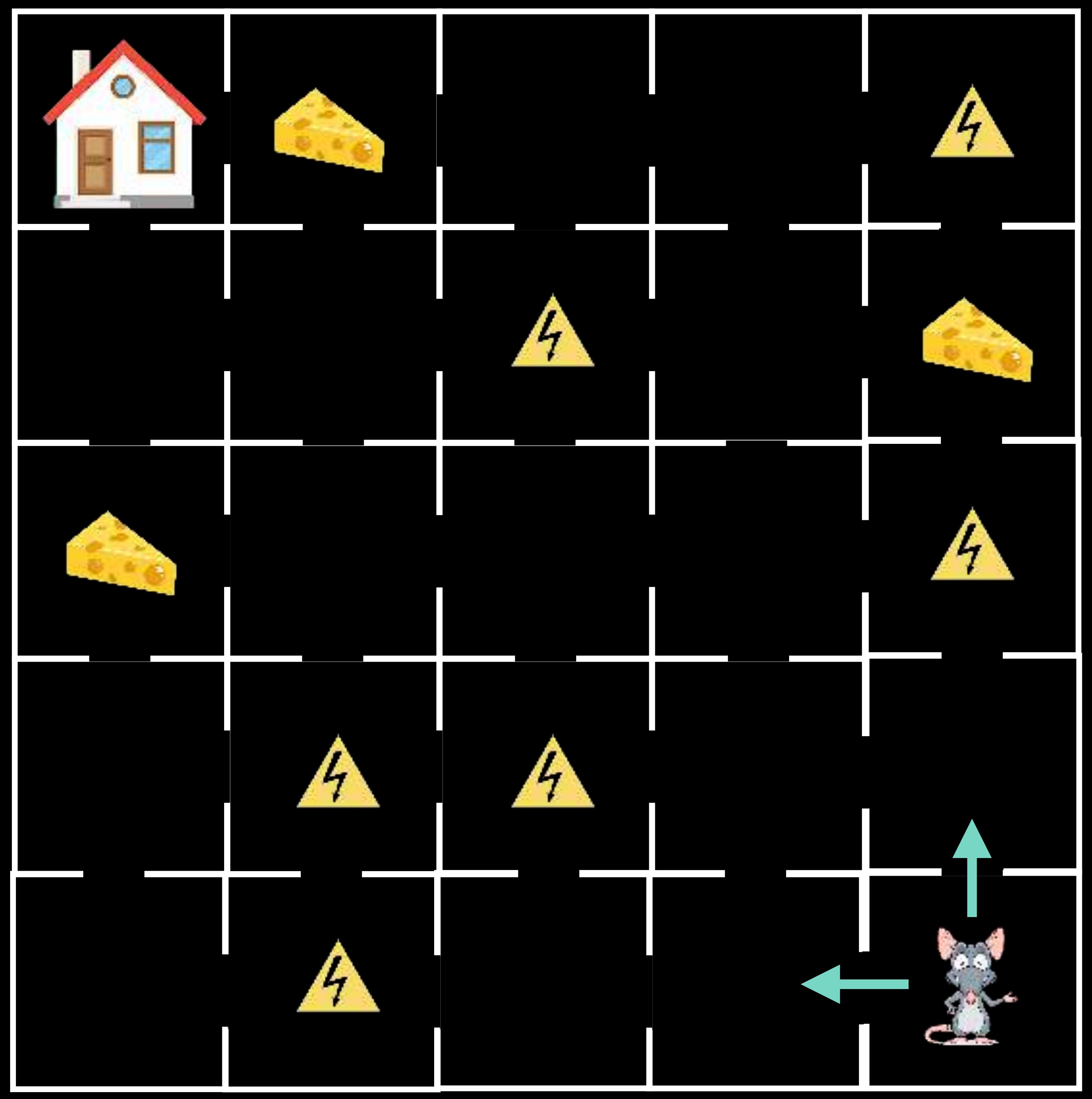
Pavlos Protopapas, Mark Glickman, and Chris Tanner

Consider a scenario where we have a mouse starting from an intial state, S_0

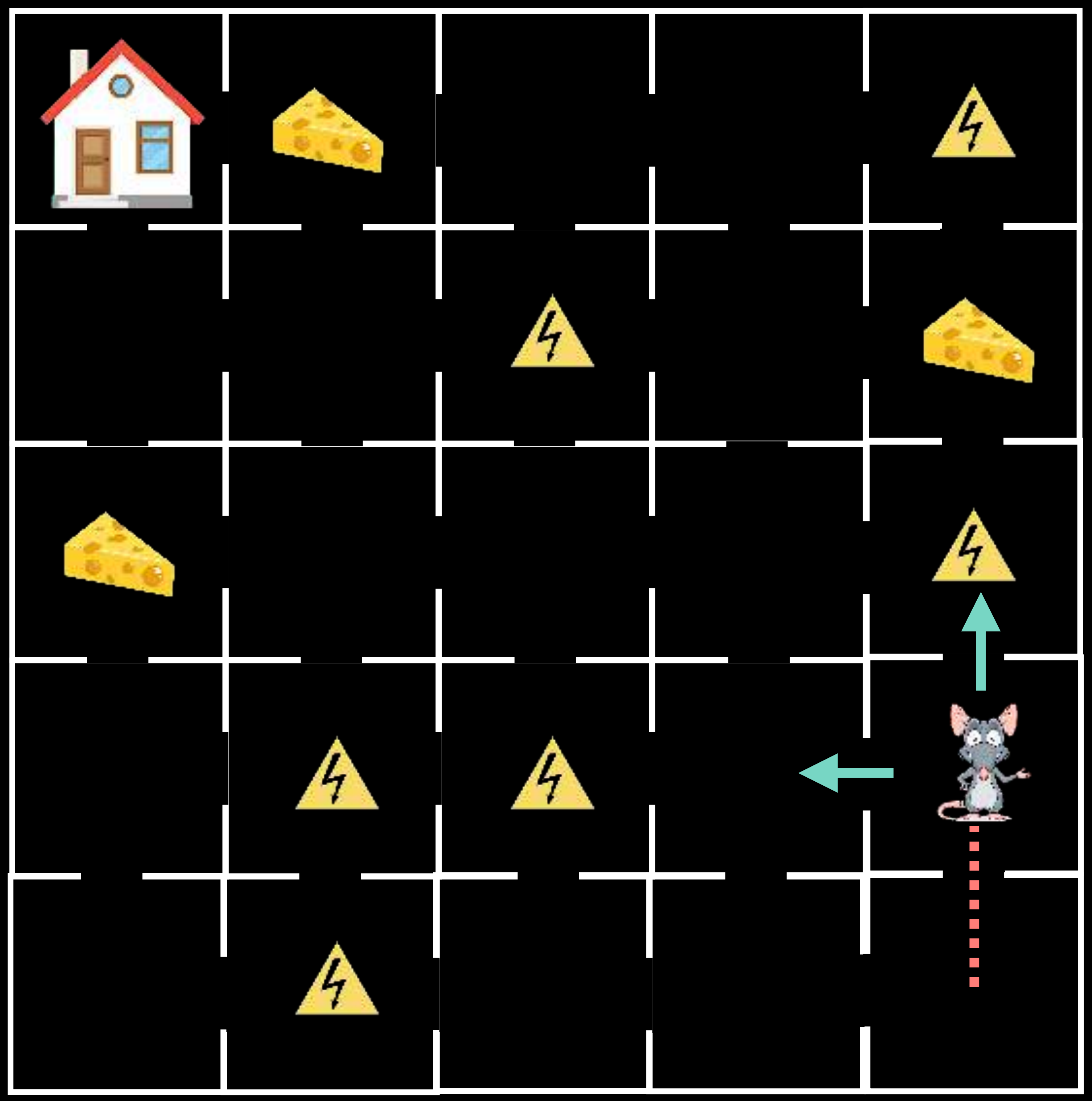


Consider a scenario where we have a mouse starting from an intial state, S_0

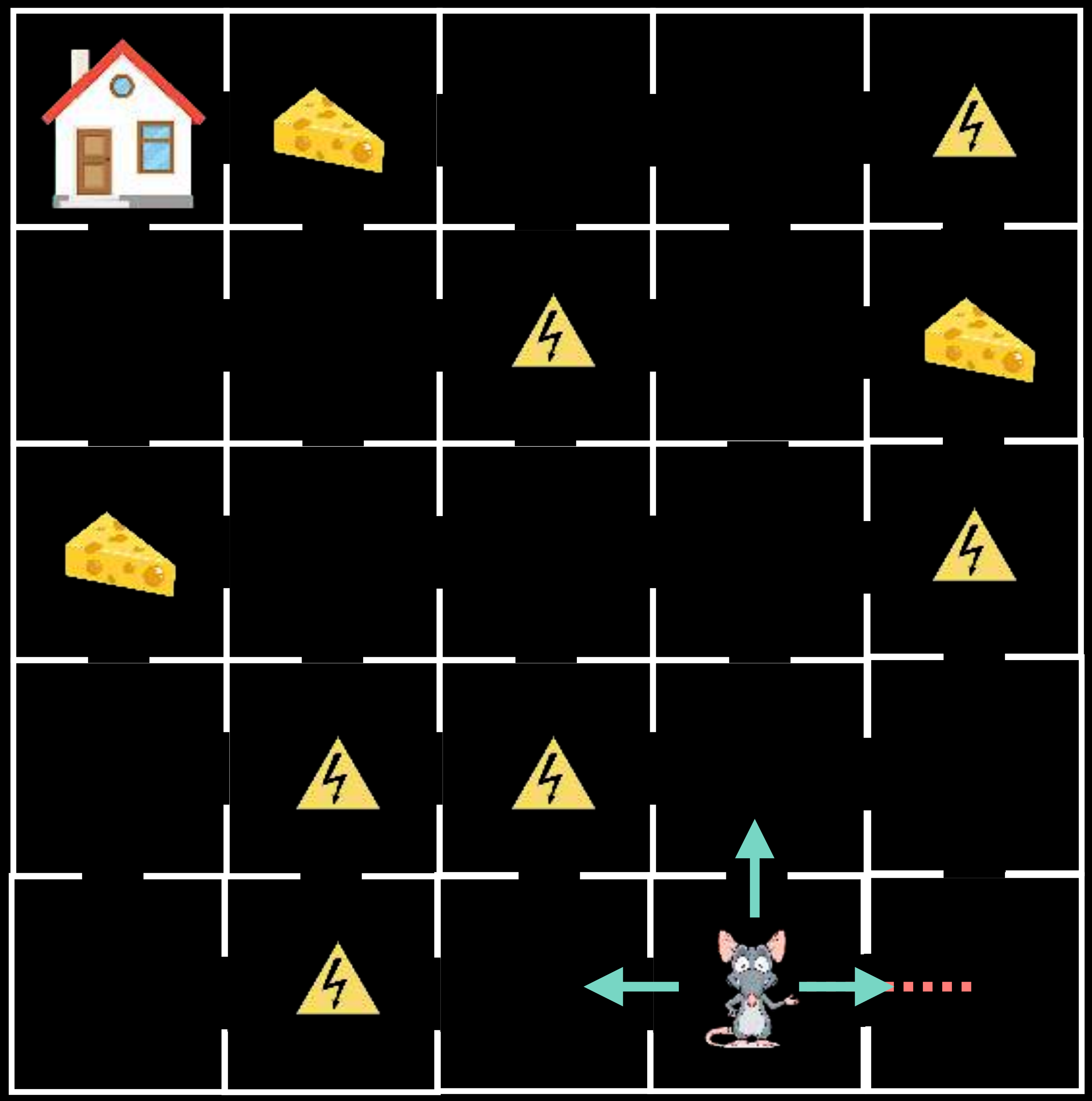
It can take 2 possible actions, go up or go left.

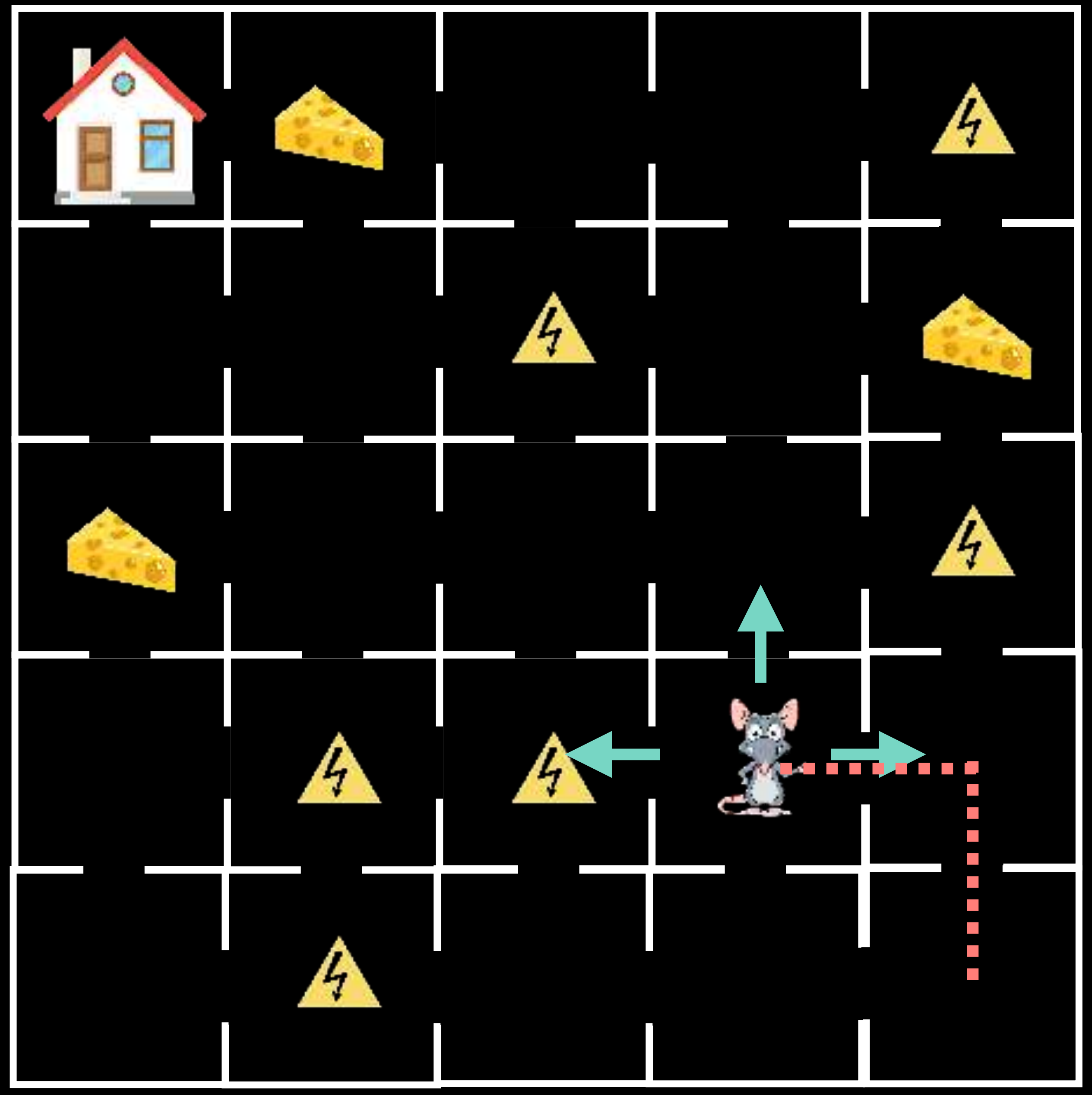


Let us assume the mouse goes up, then it again has to choose between 2 actions, up or left.

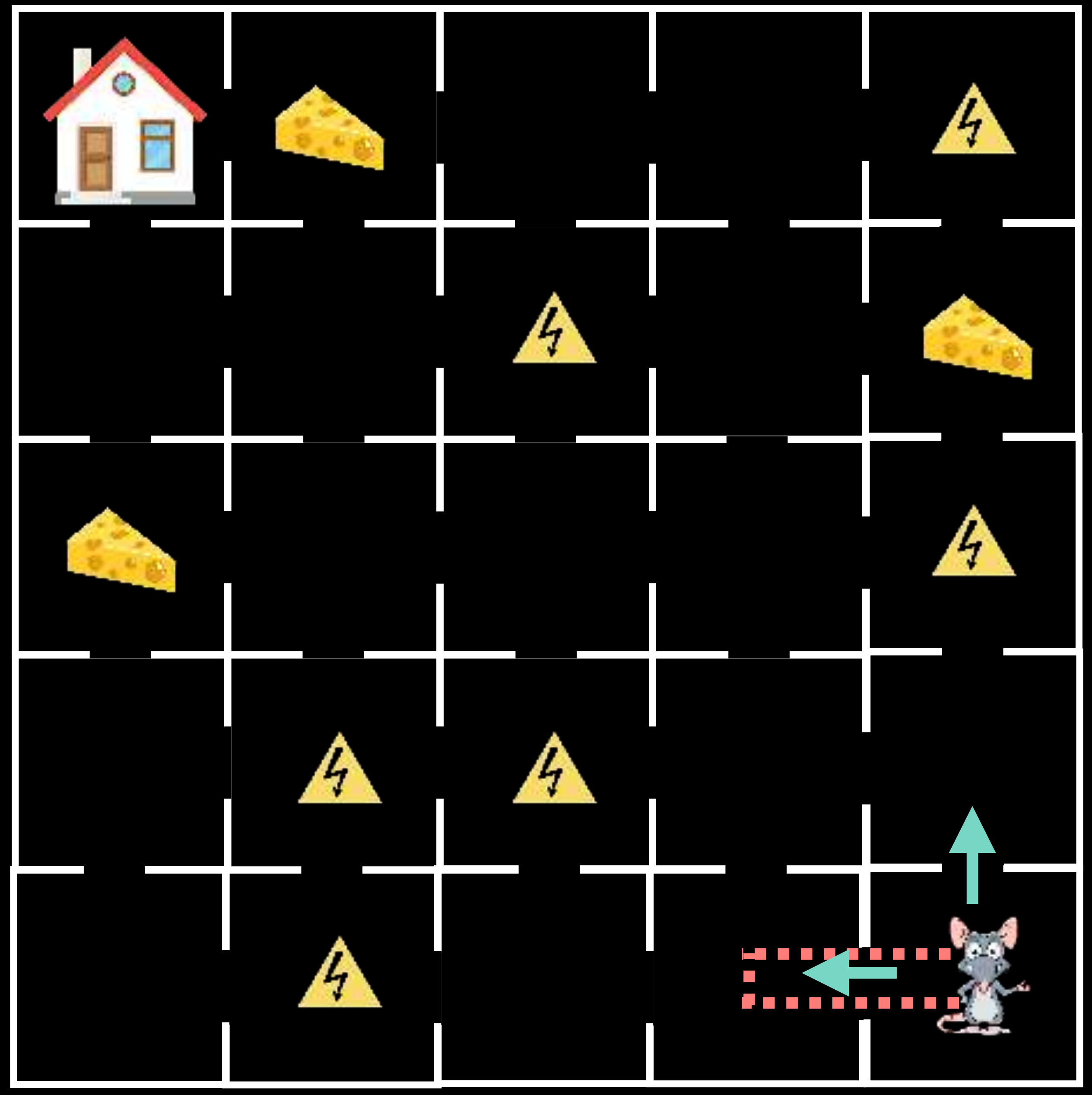


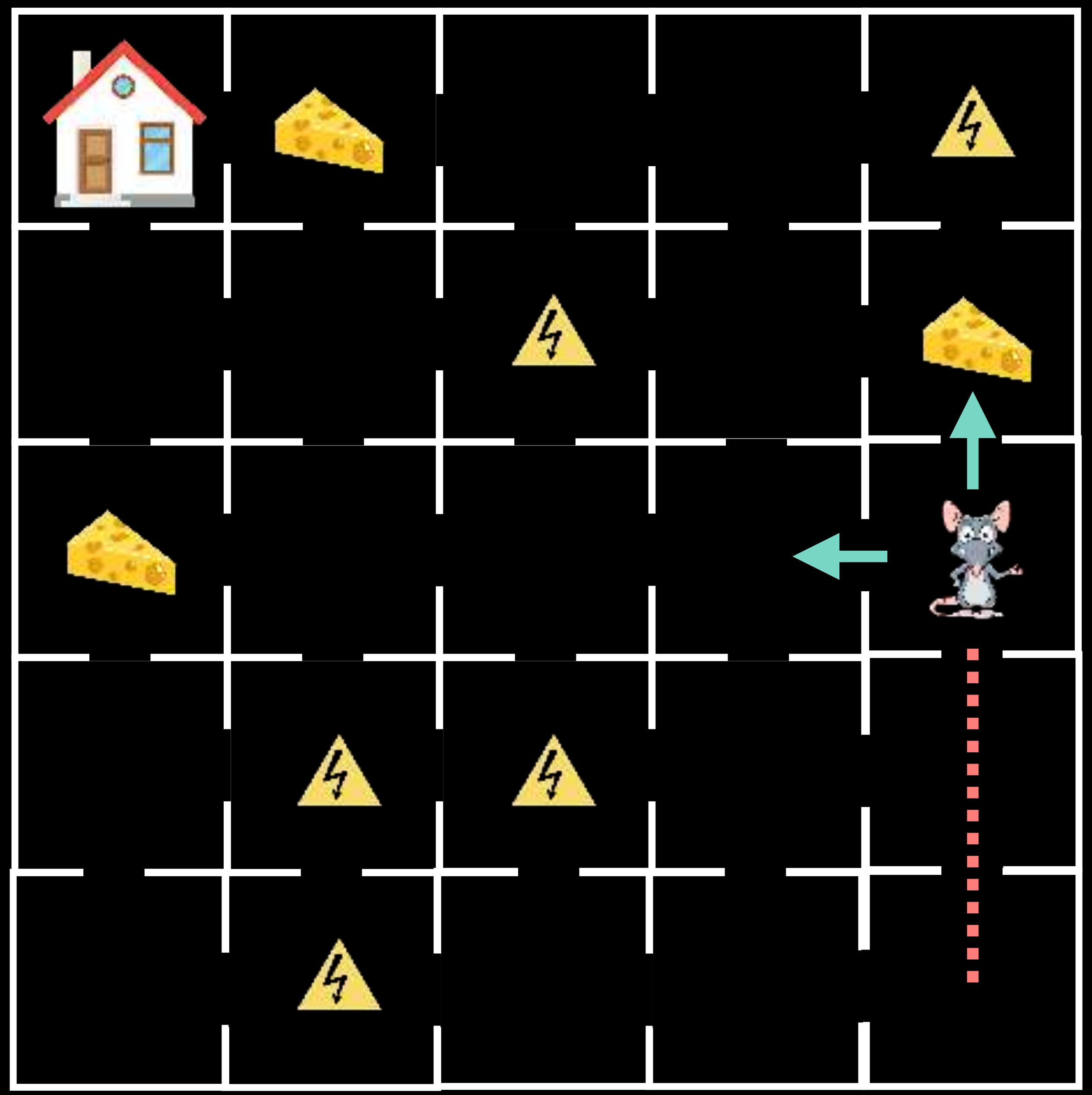
Let us assume the mouse goes left, then it again has to choose between 3 actions, up, left and right

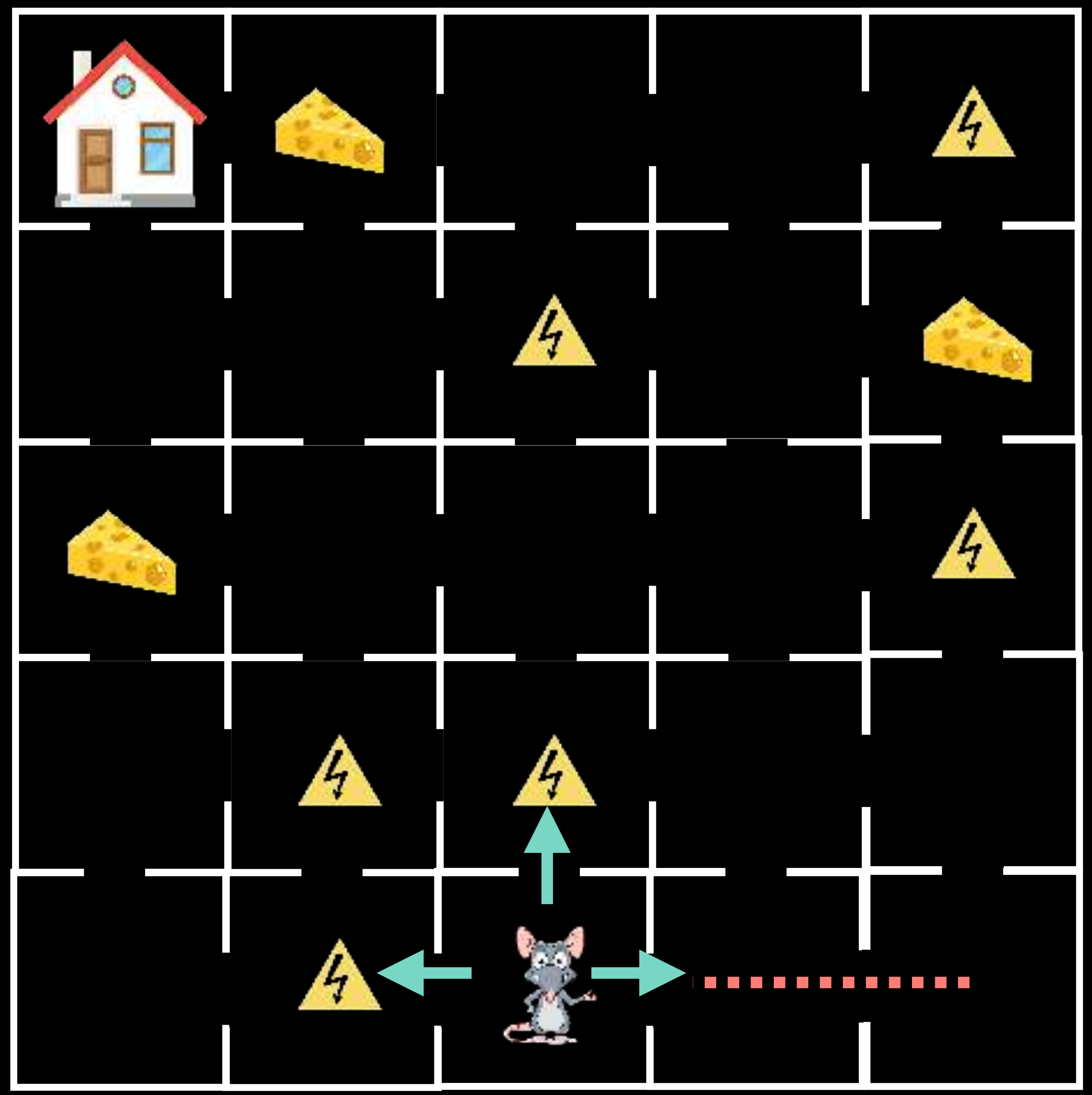


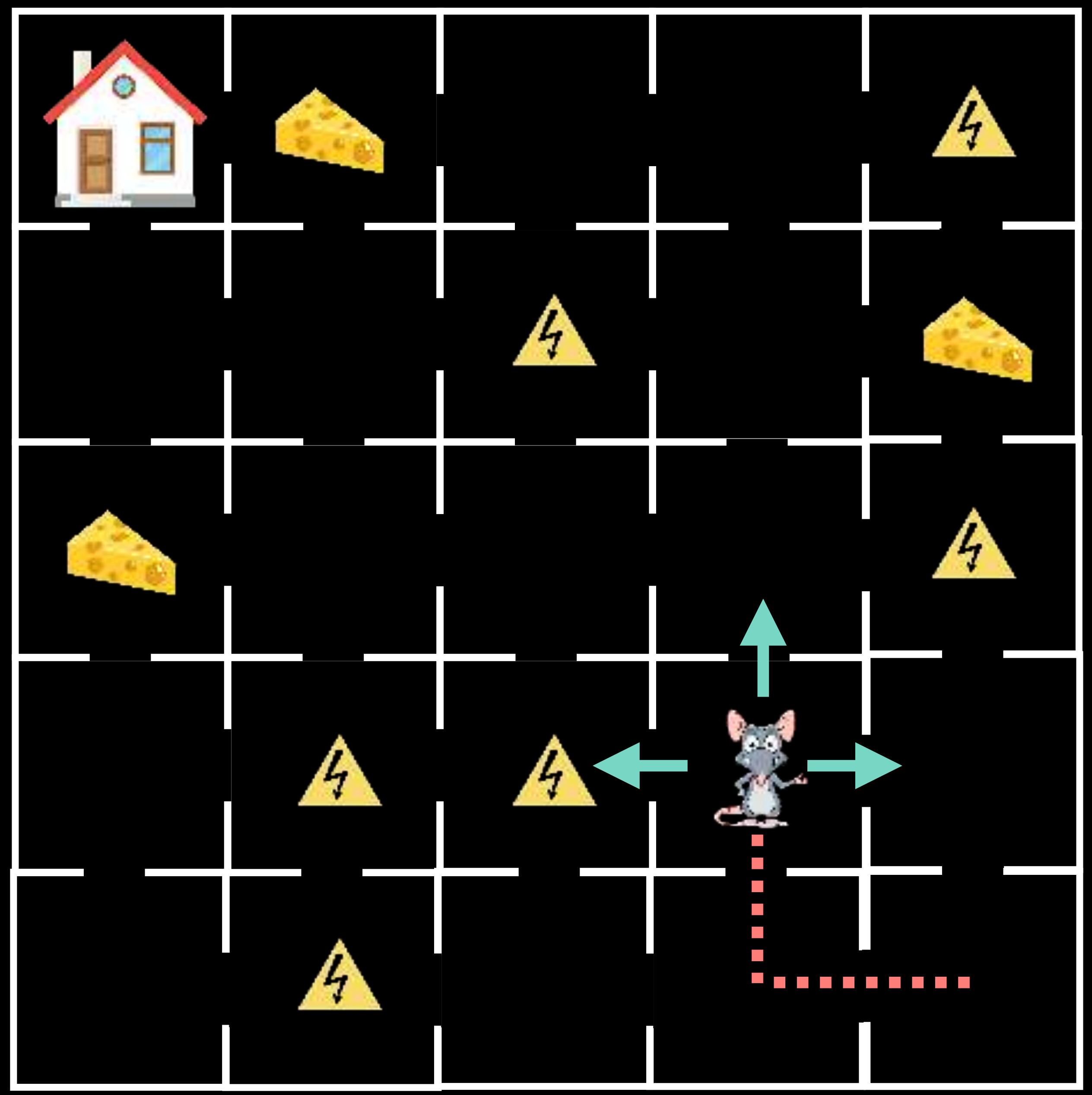


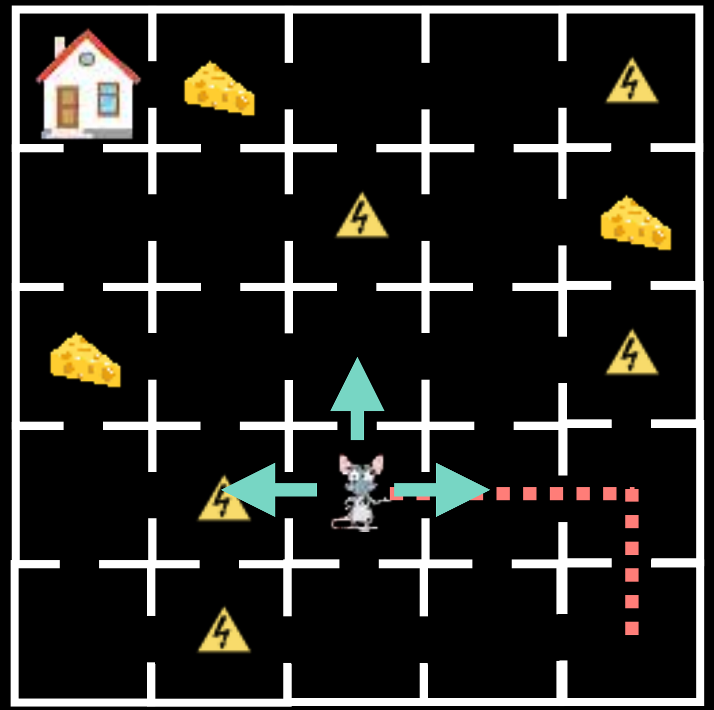
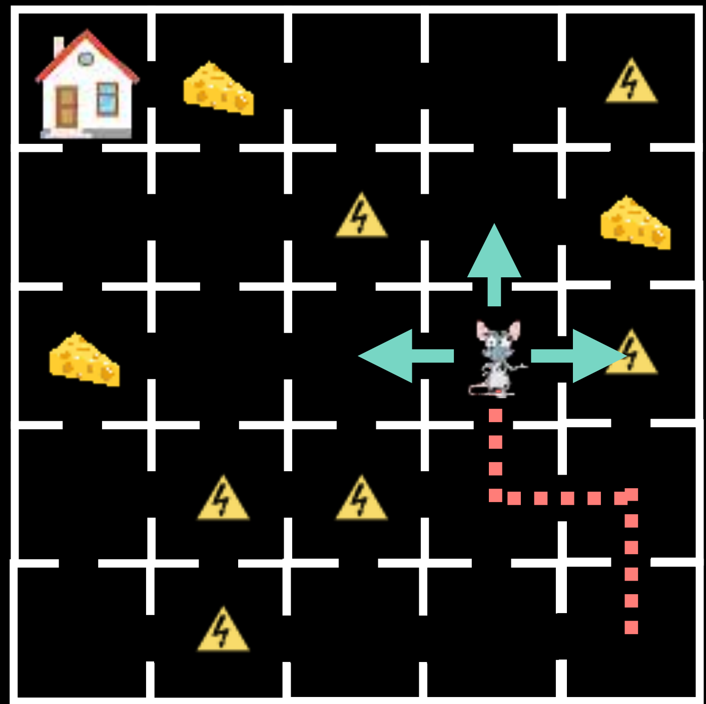
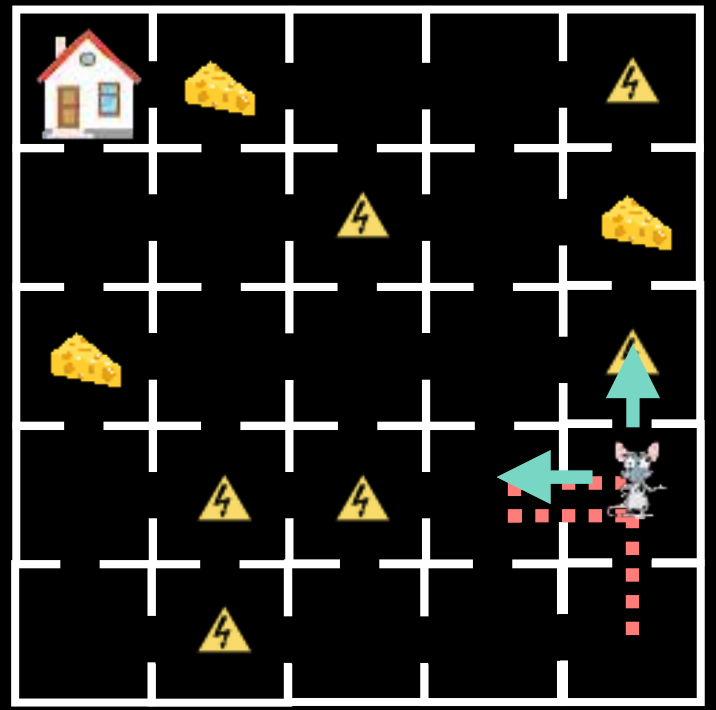
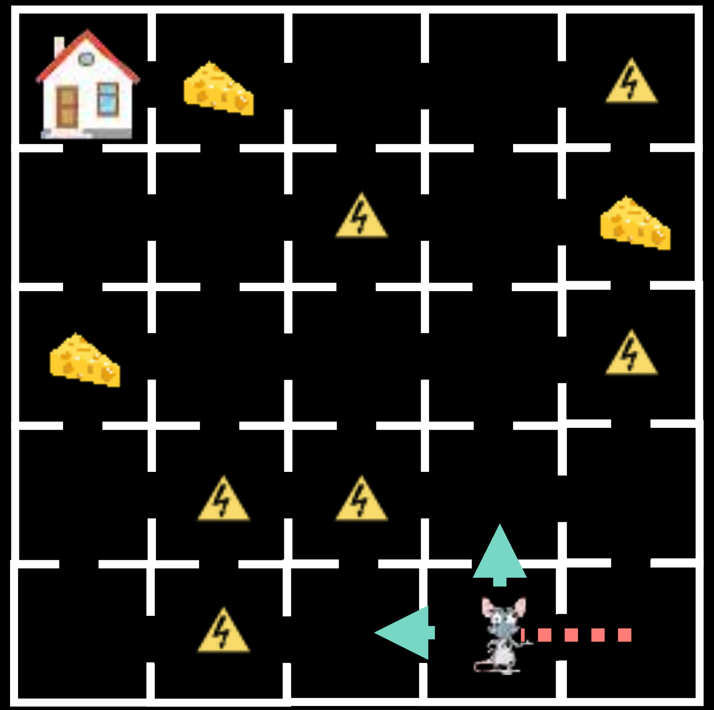
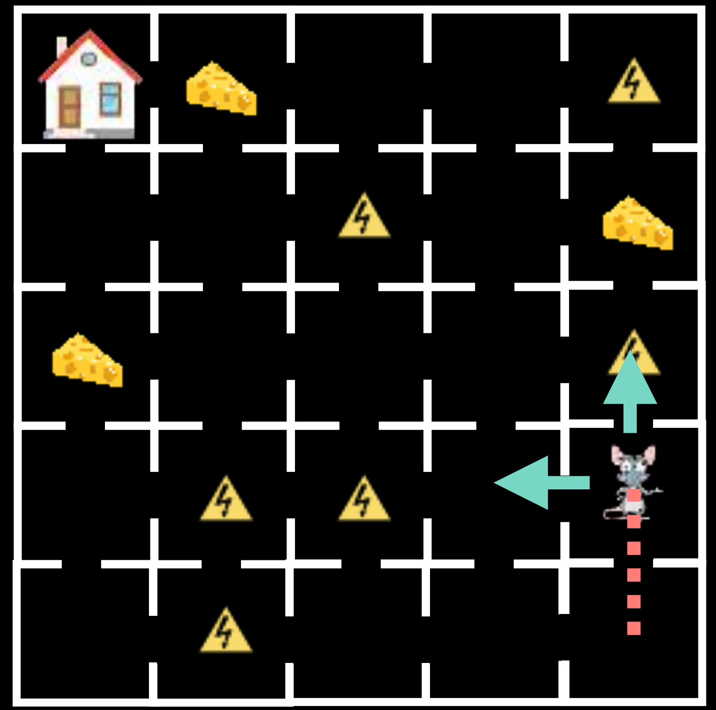
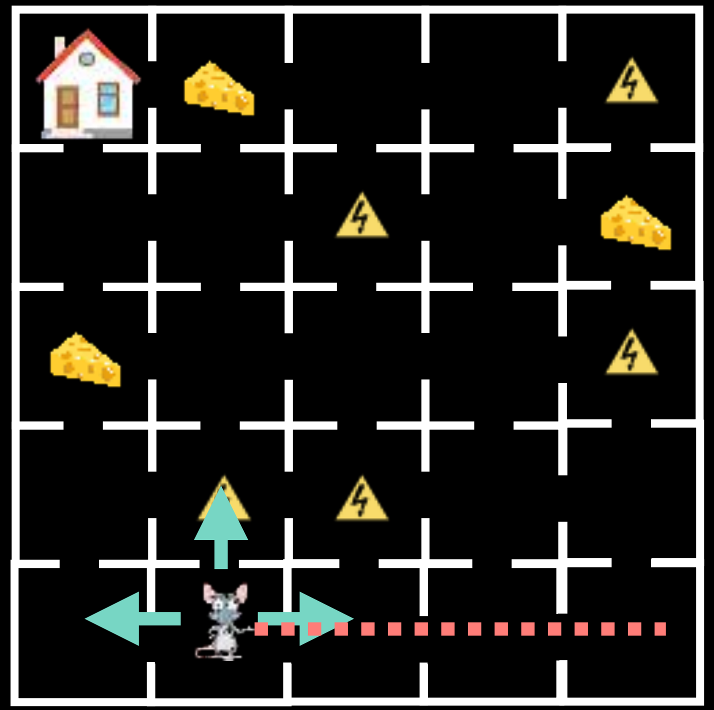
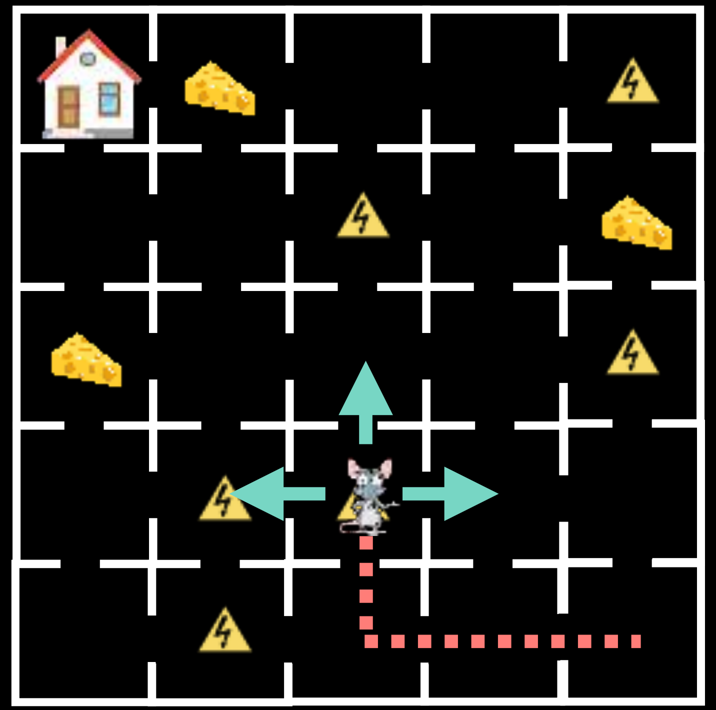
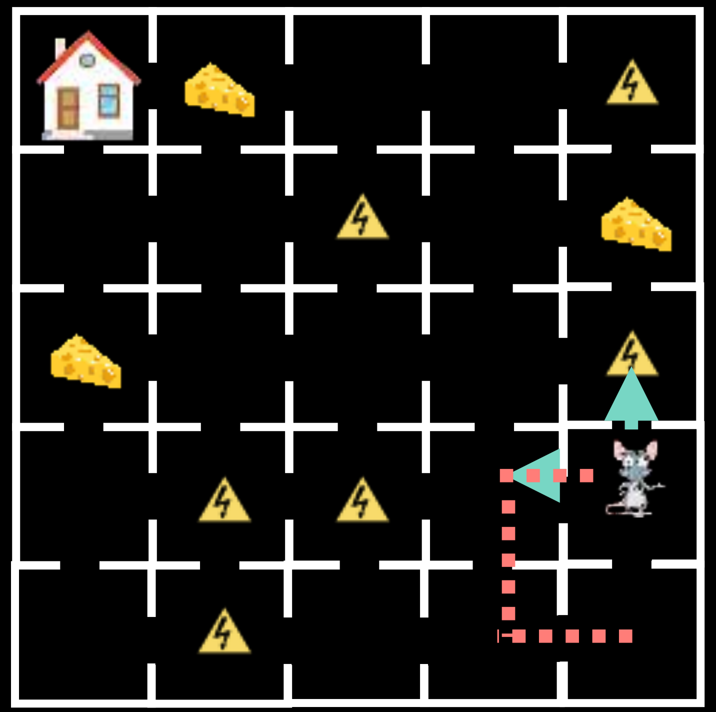
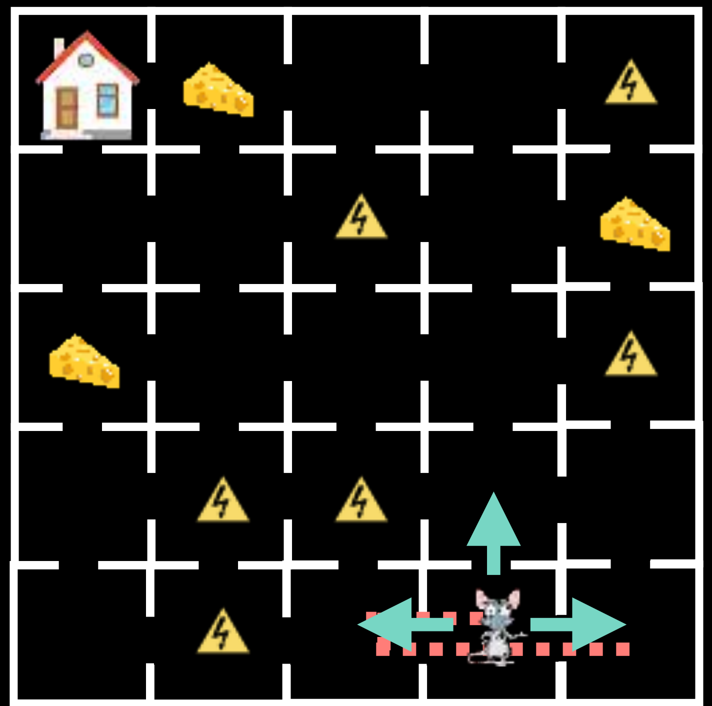
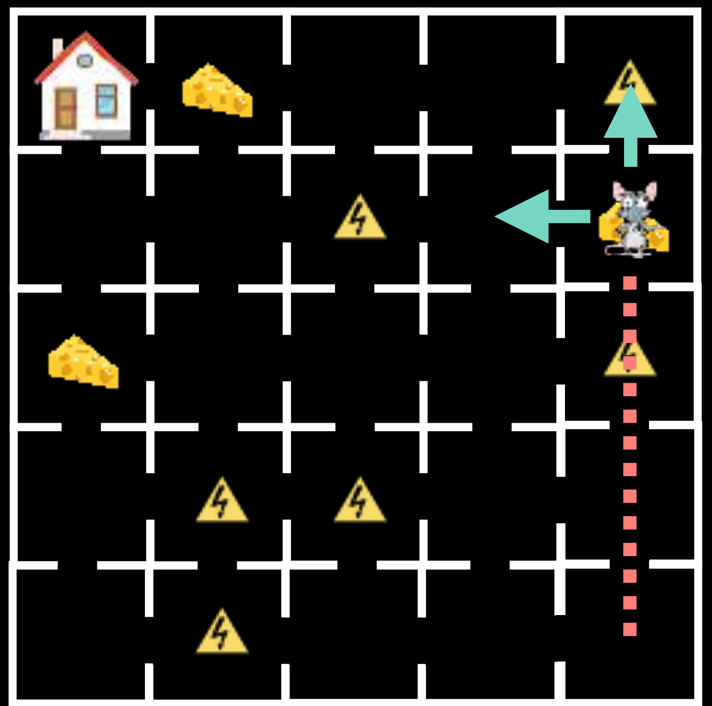
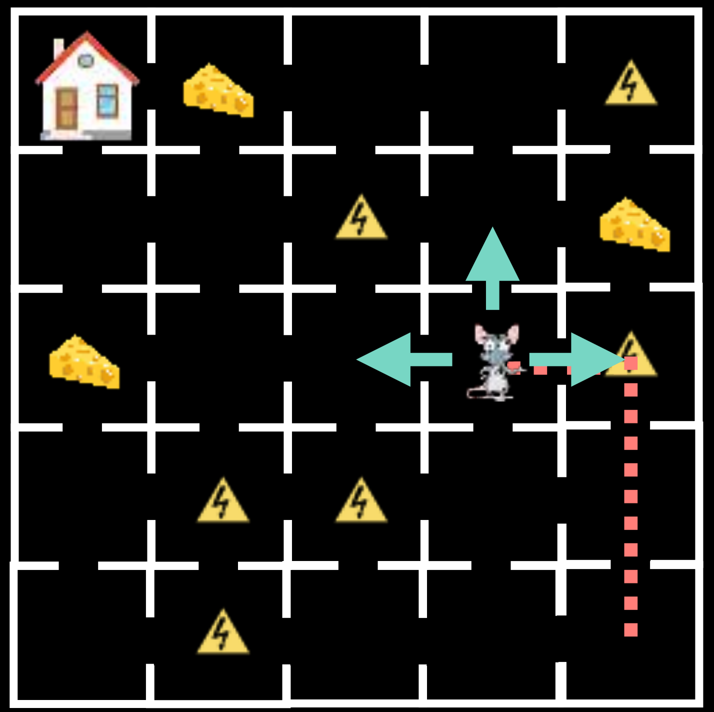
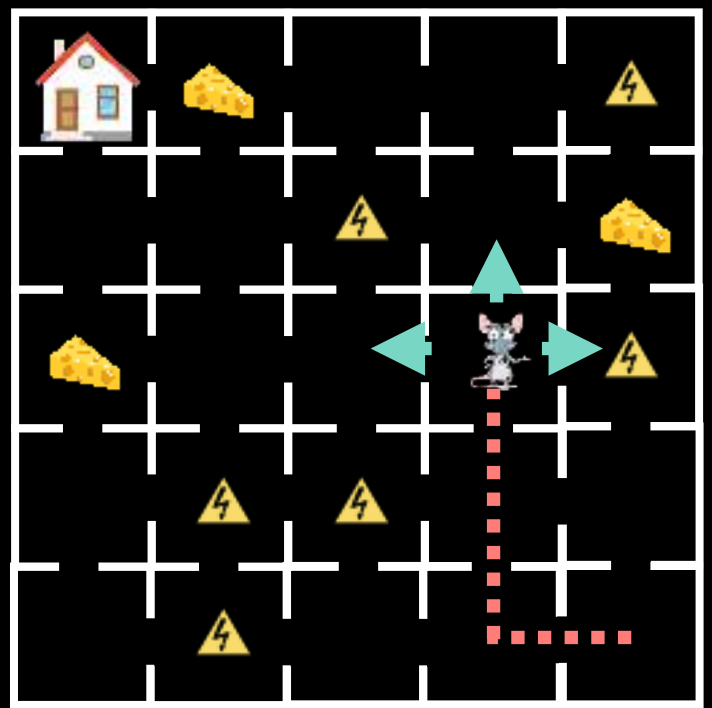
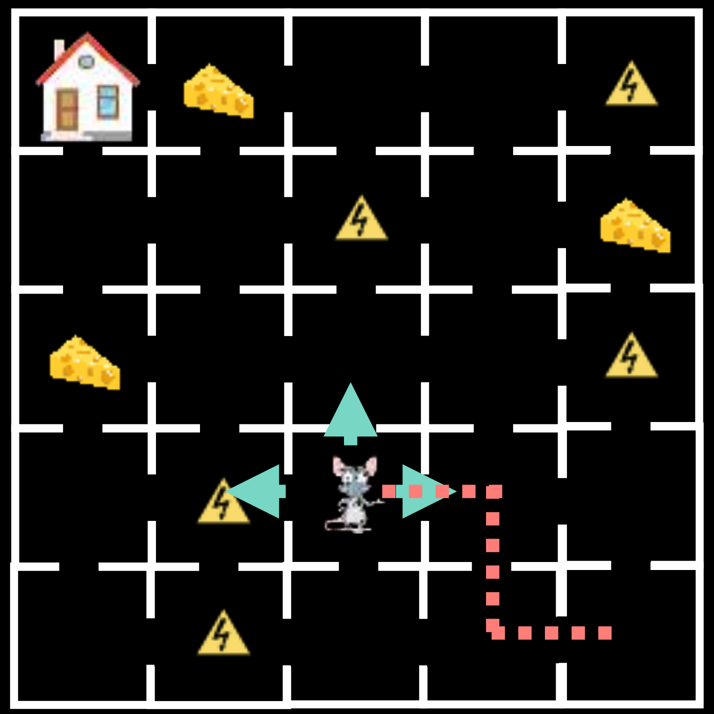
For each of the options, we further let the mouse explore all possible actions.

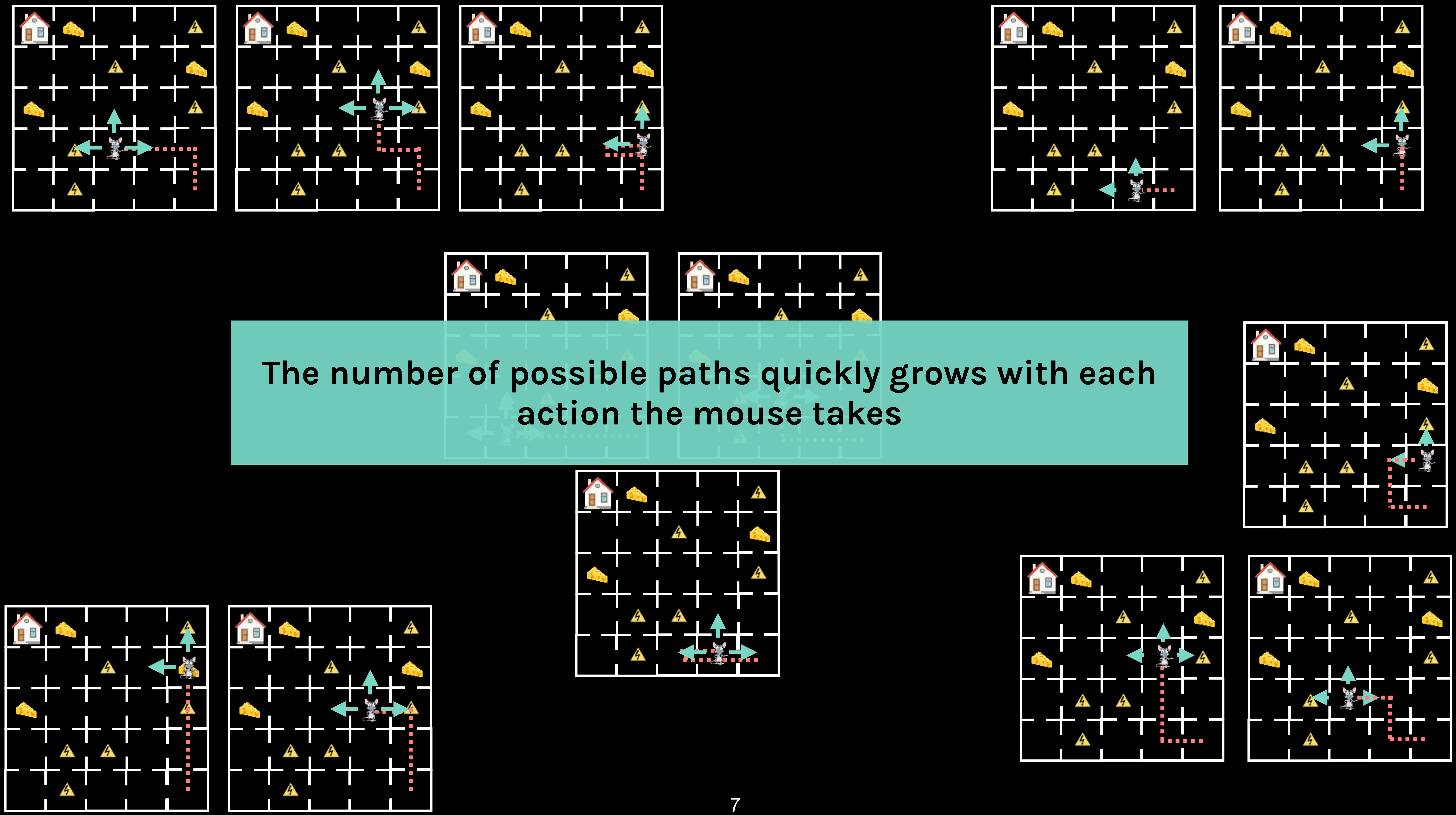










The number of possible paths quickly grows with each action the mouse takes

So how do we find the possible reward for each path taken and the best path to reach the goal?

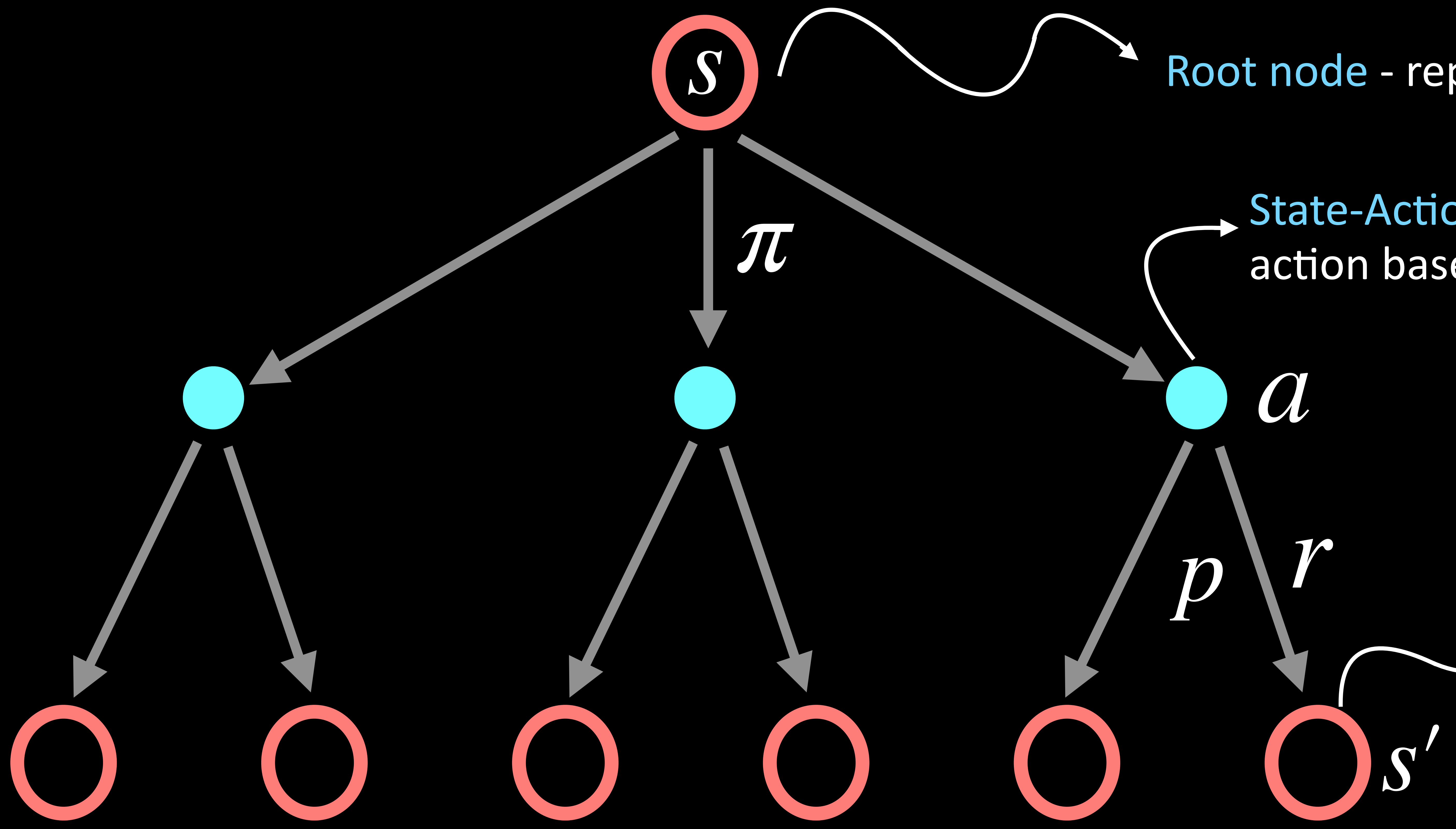
Bellman Equation

Gives the relationship between value of a state and value of its successor states.

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{\{s', r\}} p(\{s', r\} \mid s, a)[r + \gamma v_\pi(s')]$$

This equation averages all possibilities, weighting each by the probability of occurring.

It states that the value of the current state is the reward plus the discounted value of the next state.



Root node - represents the current state

State-Action pairs - the agent picks any action based on the policy

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

Next state - based on the action the environment responds with the next state and a reward

Backup diagram for $v_\pi(s)$

Recursive form of Bellman Equation

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}[G_t \,|\, S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s] \\
&= \sum_a \pi(a\,|\,s) \sum_{\{s',r\}} p(\{s', r\}\,|\,s, a)[r + \gamma \mathbb{E}[G_{t+1} \,|\, S_{t+1} = s']] \\
&= \sum_a \pi(a\,|\,s) \sum_{\{s',r\}} p(\{s', r\}\,|\,s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots \\
&= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \ldots) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Optimal policy and optimal value function

A policy π is said to be better than policy π' if its expected return is greater than or equal to that of π' for all states.

$$\pi \geq \pi' \text{ if and only if } v_\pi(s) \geq v_{\pi'}(s), \;\; \forall \; s \in S$$

All optimal policies π_* have the same state-value function called optimal state-value function $v_*(s)$ and the same optimal action-value function $q_*(s)$.

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

$$q_*(s, a) = \max_\pi \; q_\pi(s, a)$$

For any given MDP, there exists an optimal policy that is better than or equal to all other policies.

Finding an Optimal Policy

An optimal policy is got by maximising over $q_*(s, a)$,

$$\pi_*(a \mid s) = \begin{cases} 1 & \text{if } a = \underset{a \in A}{\operatorname{argmax}}\, q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

For each state, we are selecting the action that gives the highest q-value.

Bellman Optimality Equation

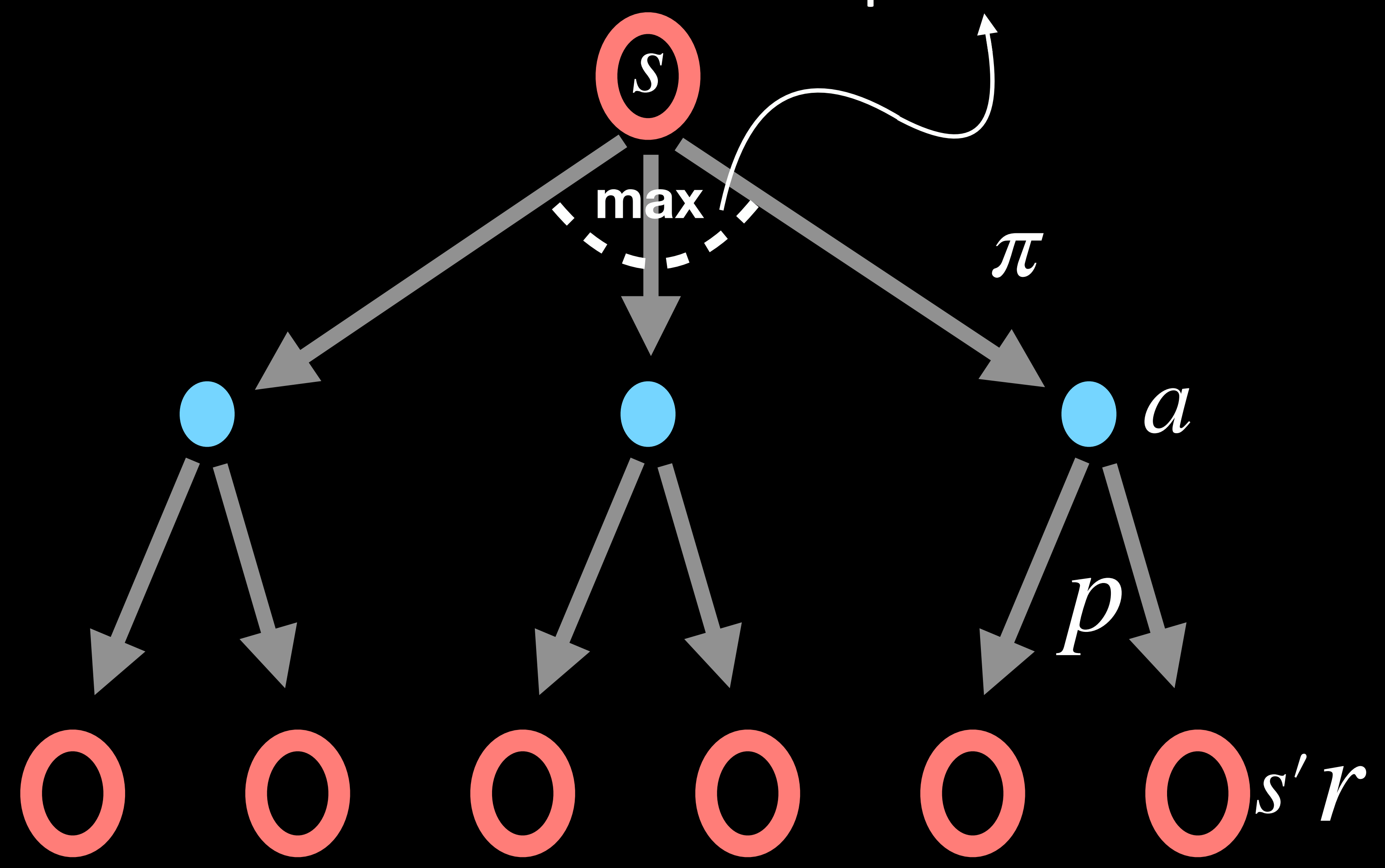
Value of a state under an optimal policy is equal to the expected return for the best action from that state.

$$v_*(s) = \max_{\pi} \sum_{\{s',r\}} p(\{s',r\} \,|\, s,a)[r + \gamma v_*(s')]$$

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

$$q_*(s) = \sum_{\{s',r\}} p(\{s',r\} \,|\, s,a) \; [r + \gamma \max_{a'} \; q_*(s',a')]$$

pick an action based on the max of all q values



The Bellman Optimality Equation considers the maximum instead of average value given some policy.

Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear.

- It can be solved using iterative methods -

    - Policy Iteration
    - Value Iteration
    - Q-Learning

Dynamic Programming

Optimal solutions can be decomposed into sub-problems.

The Bellman equation gives the recursive decomposition

Subproblems may occur many times and hence the solution can be cached and reused.

The value function stores and reuses the solution

Prediction

Given the MDP and policy π compute the value function v_π

Control

Given the MDP, find the optimal value function v_* and the optimal policy π_*

Policy Evaluation

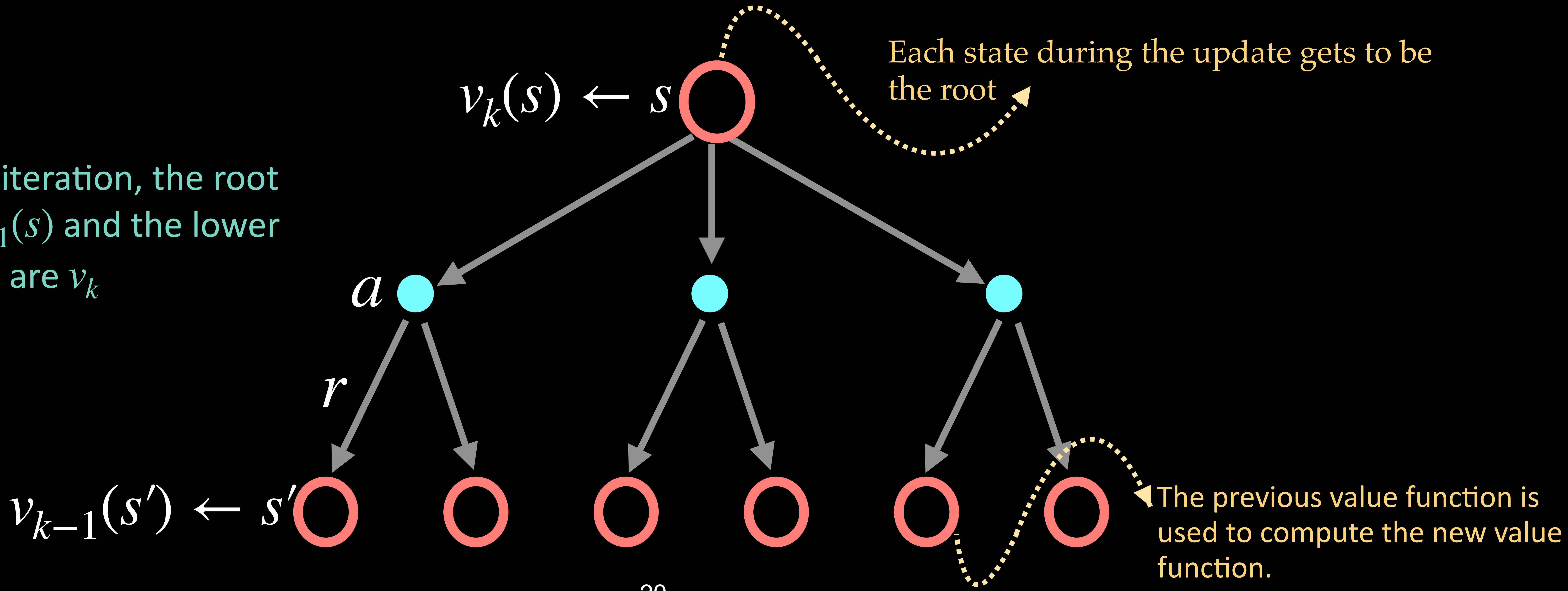
Given a policy π, find out how good it is - by computing the value function v_π

This is the Prediction problem based on the Bellman Expectation equation

$$v_k(s) = \sum_a \pi(a \mid s) \sum_{\{s',r\}} p(\{s',r\} \mid s,a)[r + \gamma v_{k-1}(s')]$$

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{\{s',r\}} p(\{s',r\} \mid s,a)[r + \gamma v_k(s')]$$

In the next iteration, the root node is $v_{k+1}(s)$ and the lower level nodes are v_k



Iterative Policy Evaluation

PSUEDO CODE

INPUT - π, the policy to be evaluated

Initialise $v(s)$, $\forall\ s \in S$ arbitrarily, except $v(terminal) = 0$

Loop:

$\triangle \leftarrow 0$

Loop for each $s \in S$:

$v \leftarrow v(s)$

$$v(s) \leftarrow \sum_{a} \pi(a \,|\, s) \sum_{\{s',r\}} p(\{s', r\} \,|\, s, a)[r + \gamma v(s')]$$

$\triangle \leftarrow \max(\triangle, |\, v - v(s) \,|)$

until $\triangle < \theta$

a small threshold $\theta > 0$ determining accuracy of estimation

Iterative Policy Evaluation

PSUEDO CODE

INPUT - π, the policy to be evaluated

Initialise $v(s)$, $\forall\ s \in S$ arbitrarily, except $v(terminal) = 0$

Loop:

$\triangle \leftarrow 0$

Loop for each $s \in S$:

$v \leftarrow v(s)$

$v(s) \leftarrow \sum_{a} \pi(a \mid s) \sum_{\{s',r\}} p(\{s', r\} \mid s, a)[r + \gamma v(s')]$

$\triangle \leftarrow \max(\triangle, |v - v(s)|)$

until $\triangle < \theta$

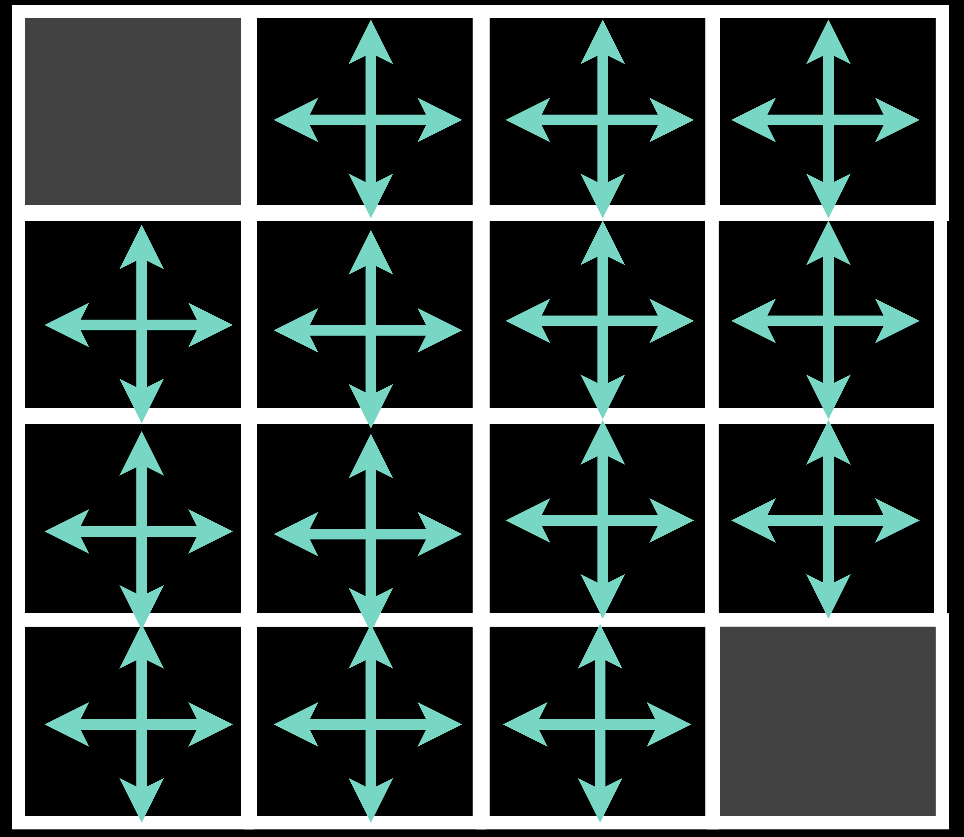
a small threshold $\theta > 0$ determining accuracy of estimation

The cost of going from one state to another is -1 i.e negative reward for each step taken

All the grey states are terminal states where the reward is always zero

k=0

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



Random Policy

k=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

The value of the state is -1 + the weighted sum of the previously estimated values of the states you expect to end up in.

k=2

0.0	-1.75	-2.0	-2.0
-1.75	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.75
-2.0	-2.0	-1.75	0.0

Probability of going in each of the 4 directions here is equal.

k=3

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

k=10

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

NOTE - These values are not equal. The decimal value will prove that the left cell value is lower than that of the right cell.

k=∞

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy Improvement

For a state s, is it better to follow policy π or choose another action $a \neq \pi(s)$?

To determine which one is better take an action a and compute the value

$$q_\pi(s, a) = \sum_{\{s',r\}} p(\{s', r\} \mid s, a)[r + \gamma v_\pi(s')]$$

Take an action $a \neq \pi(s)$

Value of following policy π after taking an action

If $q_\pi(s, a) \geq v_\pi(s)$, then we consider it overall better to take the action a every time state s is encountered.

This is a special case, in general, we want

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \quad \forall\ s \in S$$

Value of state s for the given policy π

Value of state s by taking an action $a = \pi'(s)$

$$v_\pi(s) \leq q_\pi(s, \pi'(s))$$

Expected value by taking an action based on policy π' in state S_t and then following policy π

$$= \mathbb{E}[r_{t+1} + \gamma v_\pi(S_{t+1} \,|\, S_t = s, A_t = \pi'(s))]$$

$$= \mathbb{E}_{\pi'}\,[r_{t+1} + \gamma v_\pi(S_{t+1} \,|\, S_t = s]$$

Expected value by taking an action based on policy π' in state S_{t+1} and then following policy π

$$\leq \mathbb{E}_{\pi'}\,[r_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}))\ |\ S_t = s]$$

Expected value by taking an action based on policy π' in state S_{t+1} and then following policy π from state S_{t+2} onwards

$$= \mathbb{E}_{\pi'}\,[r_{t+1} + \gamma \mathbb{E}_{\pi'}[r_{t+2} + \gamma v_\pi(S_{t+2})\ |\ S_{t+1}, A_{t+1} = \pi'(S_{t_1})]\ |\ S_t = s]$$

$$= \mathbb{E}_{\pi'}\,[r_{t+1} + \gamma r_{t+2} + \gamma^2 v_\pi(S_{t+2})\ |\ S_t = s]$$

Expected value by taking an action based on policy π' in state S_{t+2} and then following policy π from state S_{t+3} onwards

$$\leq \mathbb{E}_{\pi'}\,[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 v_\pi(S_{t+3})\ |\ S_t = s]$$

$$\vdots$$

Until termination

$$\leq \mathbb{E}_{\pi'}\,[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots\ |\ S_t = s]$$

$$\boxed{v_\pi(s) \leq v_{\pi'}(s)}$$

Expected value starting from state s and following policy π'

The algorithm shown gives a better policy wrt one state, s.

The same is iterated to all states and to all possible actions, selecting at each state the action that appears best according to $q_\pi(s, a)$

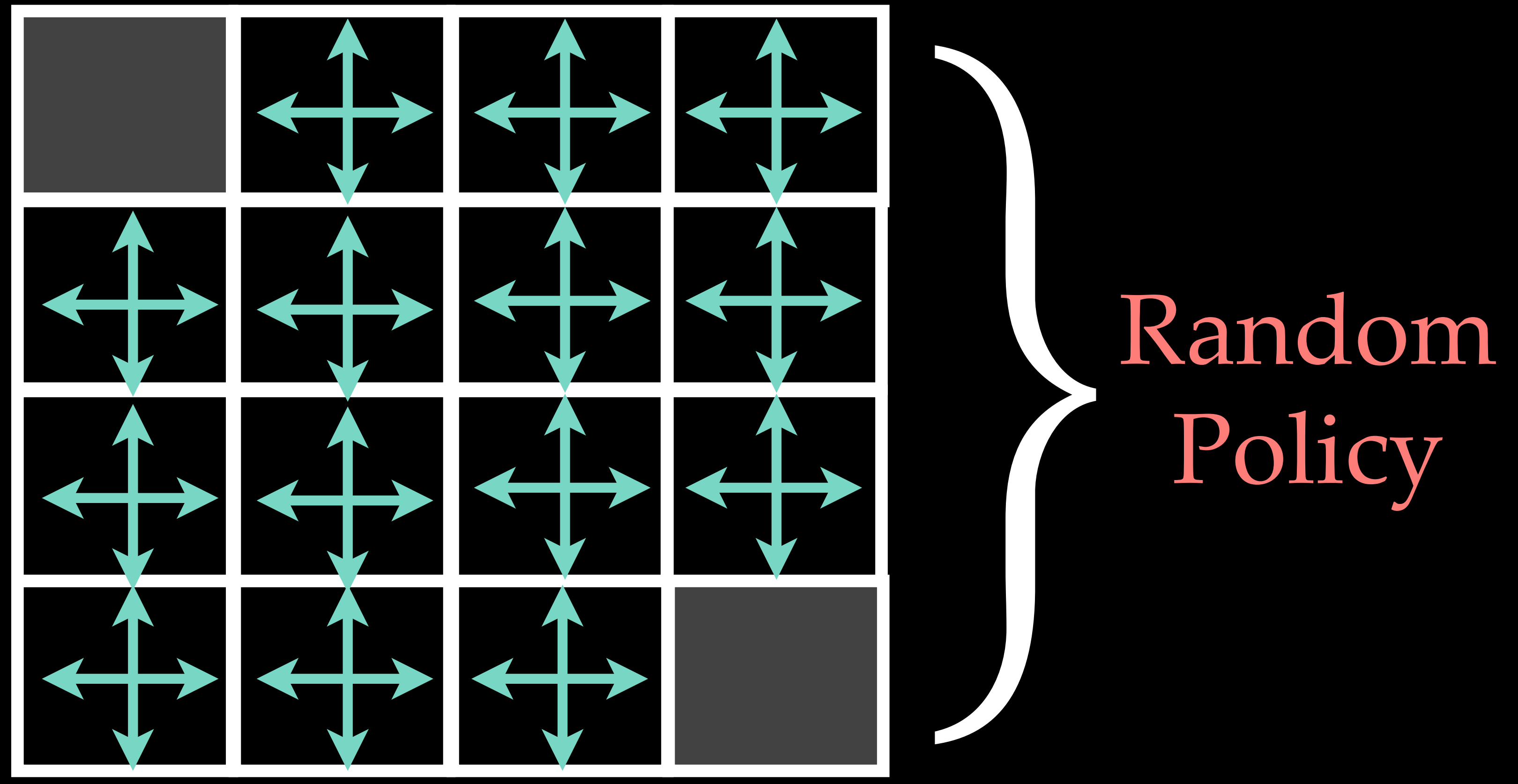
$$
\begin{aligned}
\pi'(s) &= \underset{a}{\operatorname{argmax}}\ q_\pi(s, a) \\
&= \underset{a}{\operatorname{argmax}}\ \mathbb{E}[r_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s, A_t = a] \\
&= \underset{a}{\operatorname{argmax}} \sum_{\{s', r\}} p(\{s', r\} \,|\, s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

he greedy policy takes the action that looks best in one step lookahead according to policy π

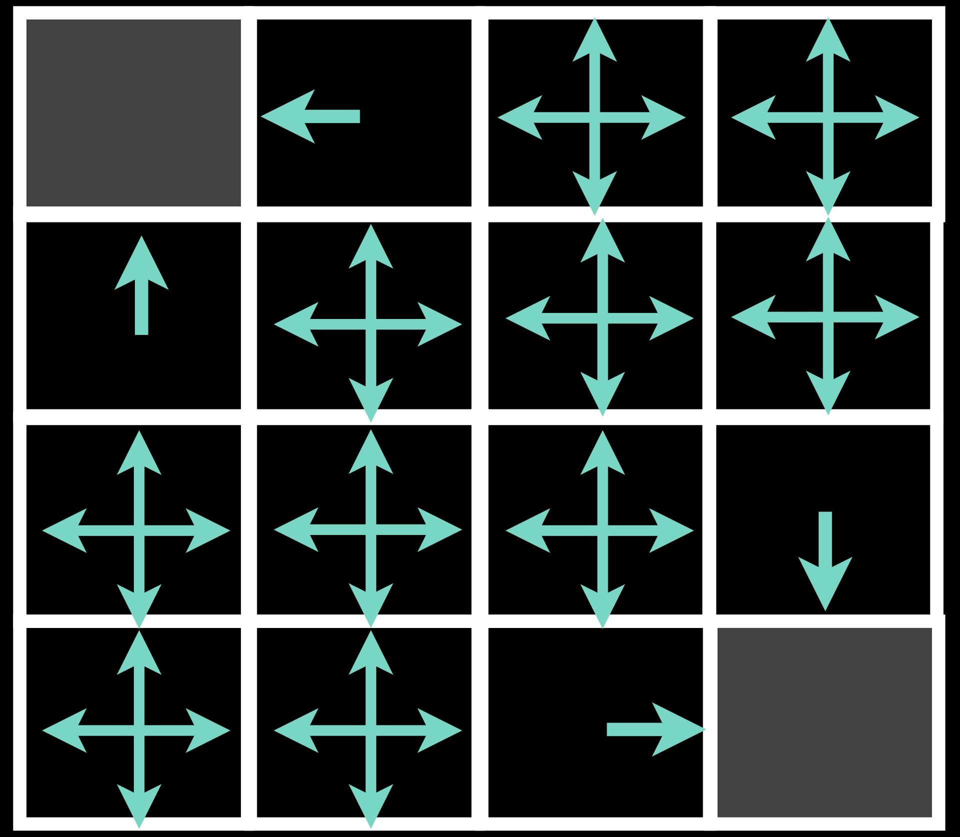
The cost of going from one state to another is **-1** i.e negative reward for each step taken

k=0

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



The policy is updated greedily wrt the value function.

Policy Iteration

Find best policy in an MDP

Given a policy π

Iterate
1. Evaluate the policy π

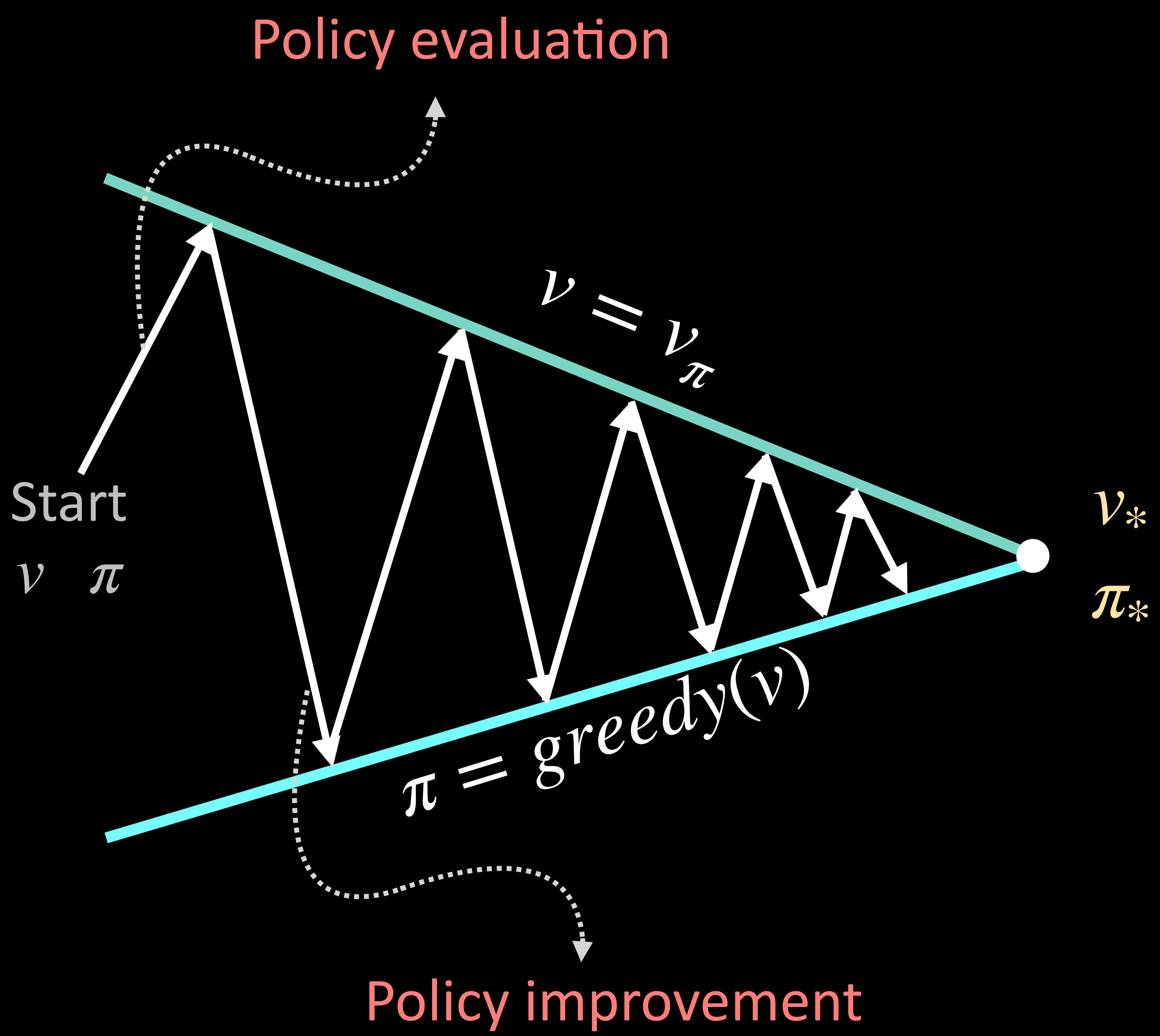
$$v_\pi(s) = \mathbb{E}_\pi\left[r_{t+1} + \gamma r_{t+1} + \ldots \mid S_t = s\right]$$

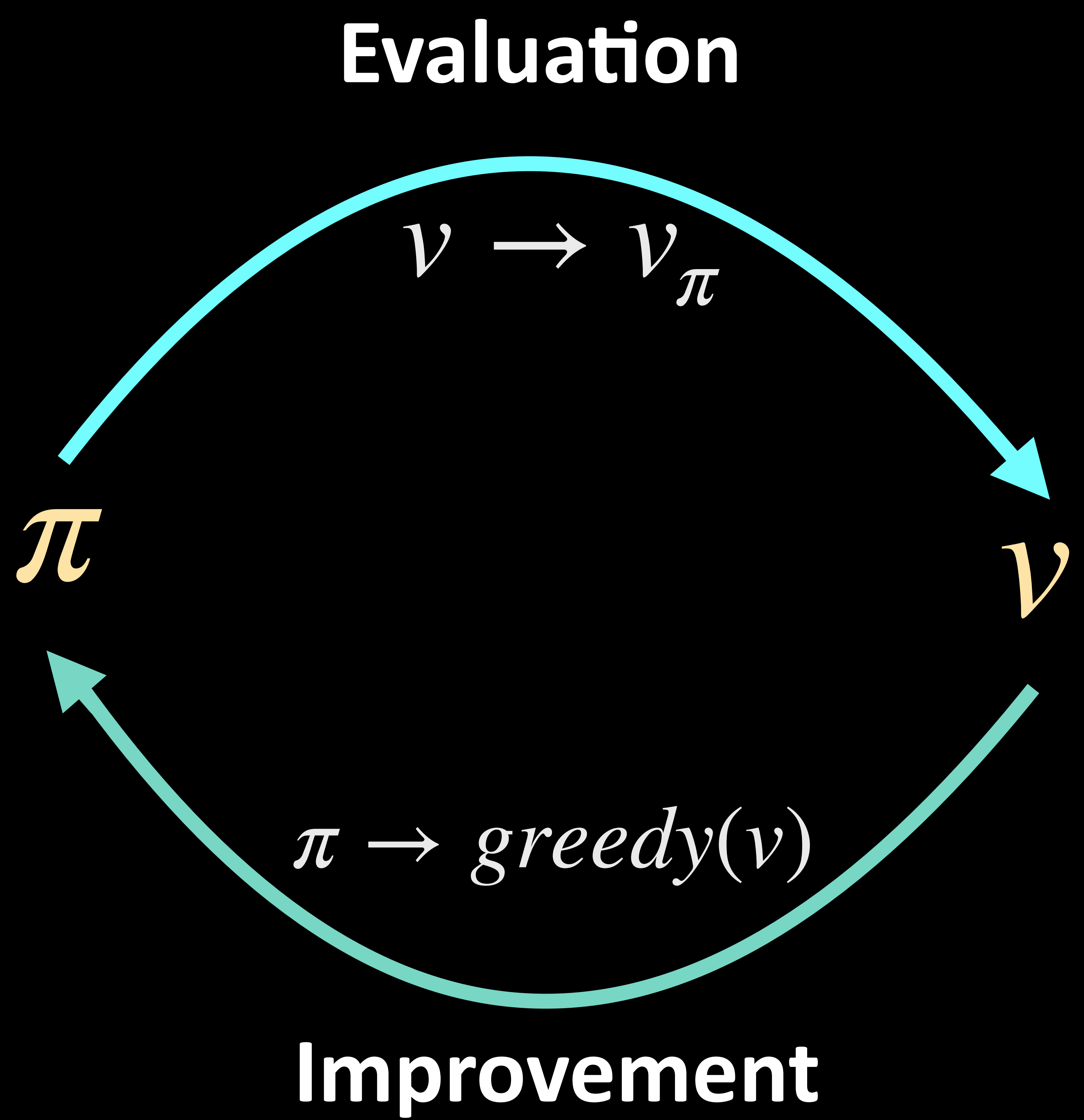
2. Improve the policy by acting greedily wrt v_π

$$\pi' = greedy(v_\pi)$$

POLICY ITERATION

The process of policy iteration converges to the optimal policy π_*


Policy evaluation
$v = v_\pi$
Start
v π
v_*
π_*
$\pi = greedy(v)$
Policy improvement


Evaluation
$v \rightarrow v_\pi$
π
v
$\pi \rightarrow greedy(v)$
Improvement

PSUEDO CODE

INITIALIZE - $v(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$, except $v(terminal) = 0$

POLICY EVALUATION

Loop:

$\triangle \leftarrow 0$

Loop for each $s \in S$:

$v \leftarrow v(s)$

$$v(s) \leftarrow \sum_{\{s',r\}} p(\{s', r\} \,|\, s, \pi(s))[r + \gamma v(s')]$$

$\triangle \leftarrow \max(\triangle, |v - v(s)|)$

until $\triangle < \theta$

a small threshold $\theta > 0$ determining accuracy of estimation

ITERATE TO CONVERGENCE

POLICY IMPROVEMENT

policy-stable $\leftarrow$ *true*

For each $s \in S$:

old-action $\leftarrow \pi(s)$

$$\pi(s) \leftarrow \operatorname*{argmax}_a \sum_{\{s',r\}} p(\{s', r\} \,|\, s, a)[r + \gamma v(s')]$$

If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

If *policy-stable,* then stop and return $v \approx v_*$ and $\pi \approx \pi_*$; else policy evaluation

If improvements stop,

$$q(s, \pi'(s)) = \max_{a \in A} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

The Bellman optimality equation has been satisfied ,

$$v_\pi(s) = \max_{a \in A} q_\pi(s, a)$$

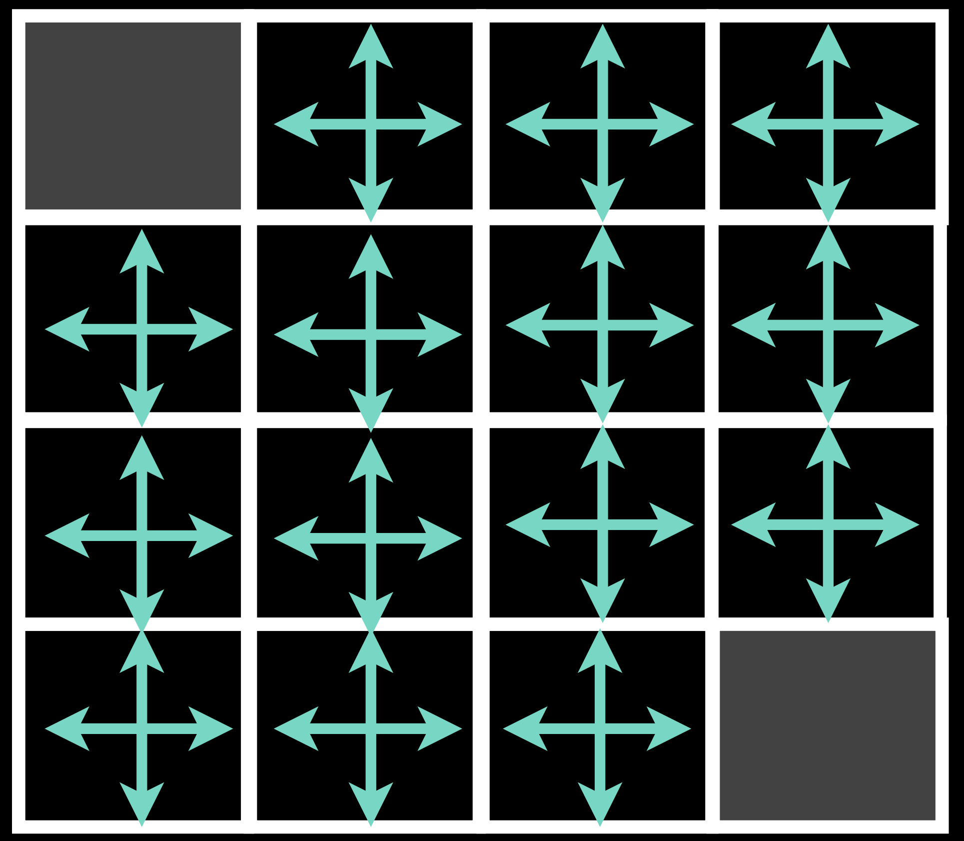
$v_\pi(s) = v_*(s)$ for all $s \in S \quad \Longrightarrow \quad \pi$ is the optimal policy

The cost of going from one state to another is -1 i.e negative reward for each step taken

All the grey states are terminal states where the reward is always zero

k=0

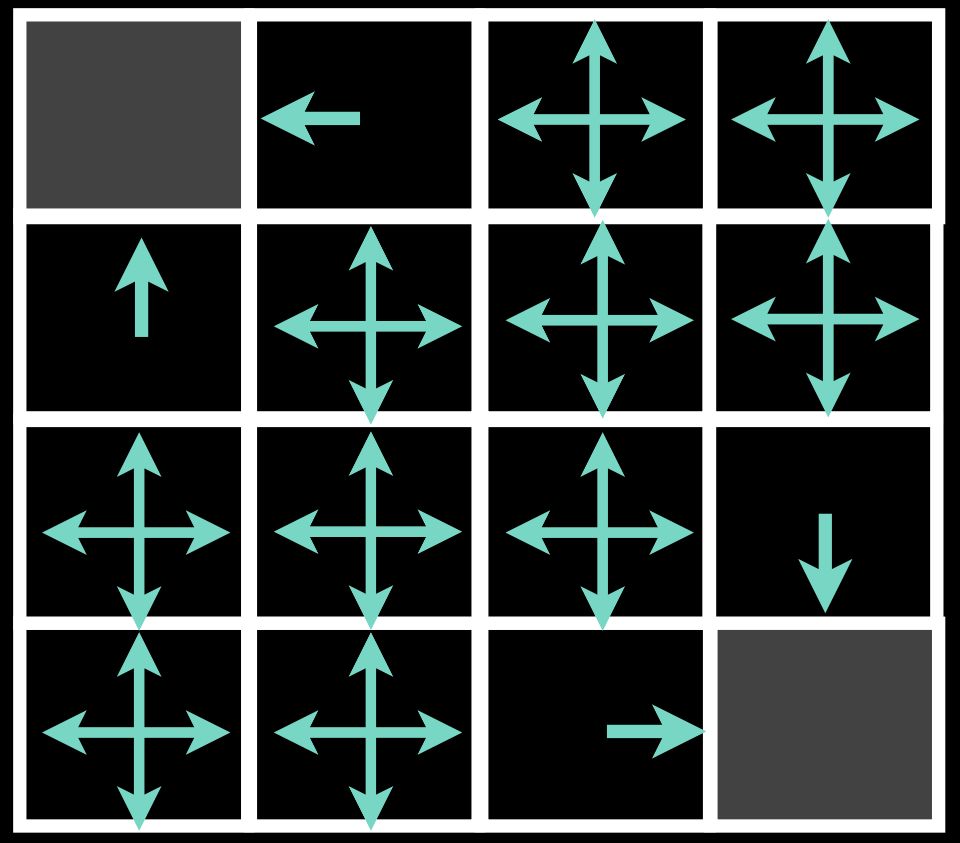
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



Random Policy

k=1

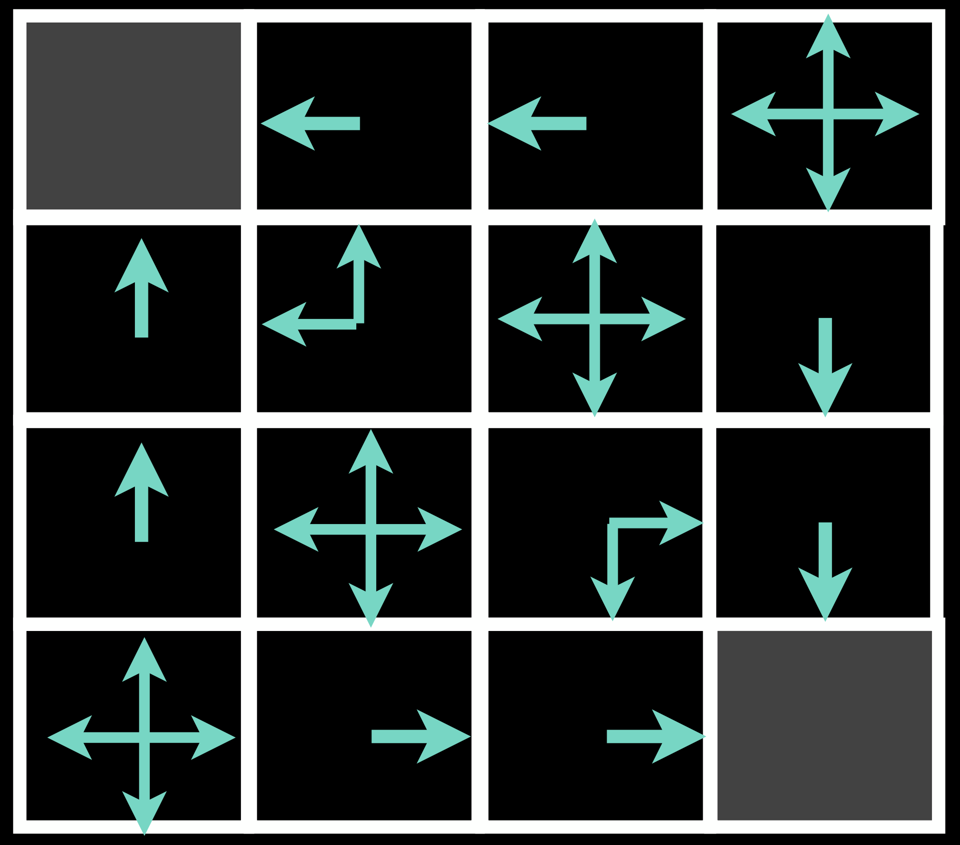
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



The value of the state is -1 + the weighted sum of the previously estimated values of the states you expect to end up in.

k=2

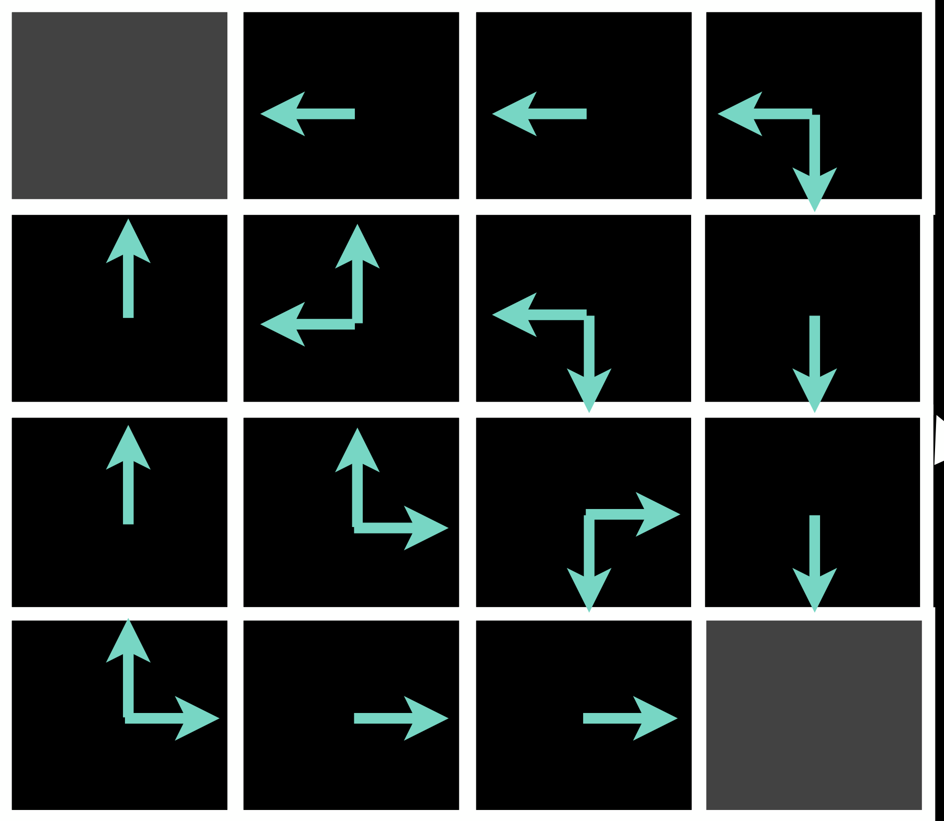
0.0	-1.75	-2.0	-2.0
-1.75	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.75
-2.0	-2.0	-1.75	0.0



Probability of going in each of the 4 directions here is equal.

k=3

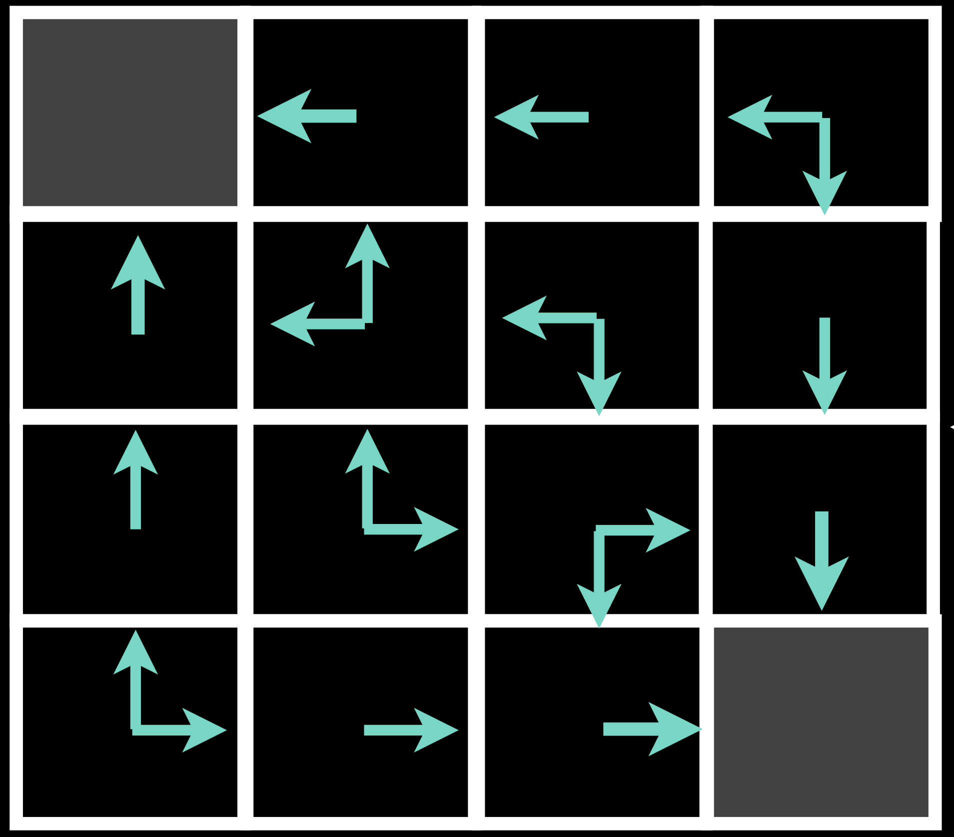
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



The value function while evaluating a given policy helps get an optimal policy

k=10

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

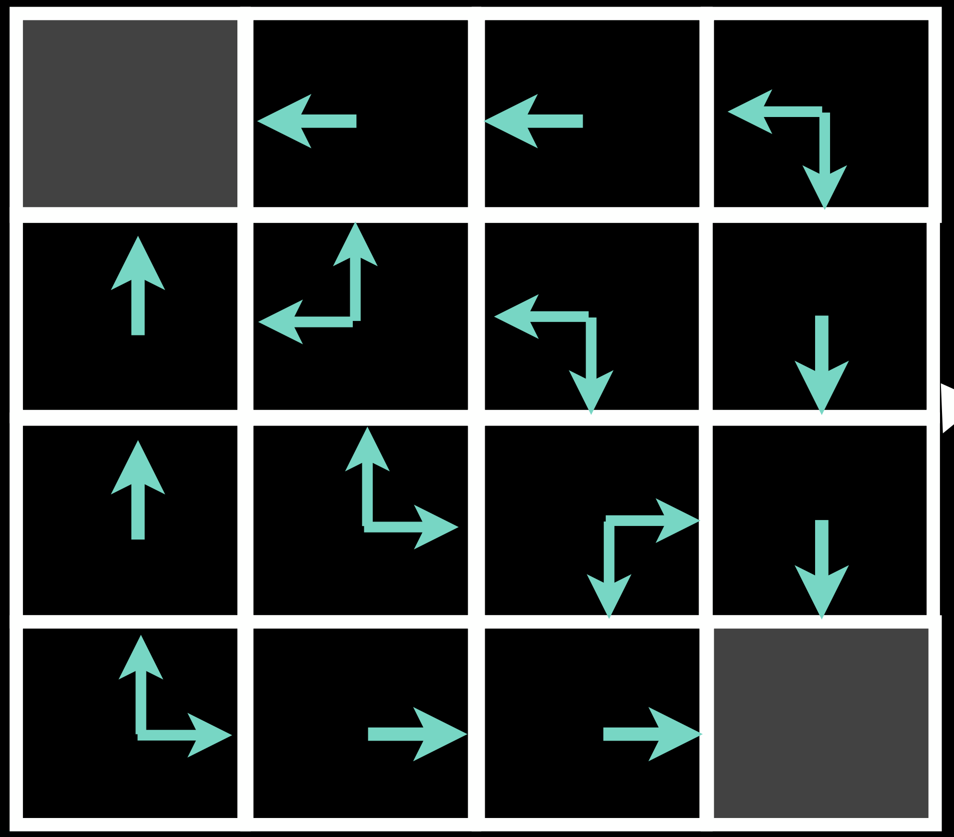


Optimal Policy

NOTE - These values are not equal. The decimal value will prove that the left cell value is lower than that of the right cell.

k=∞

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Modified Policy Iteration

Instead of looping till convergence, stop after *k* iterations of iterative policy evaluation.

Act greedy according to this value to get the new policy and continue the process. This is guaranteed to converge to the optimal policy.

When *k=1* → Value Iteration

Value Iteration

THEOREM OF OPTIMALITY

A policy $\pi(a \mid s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if

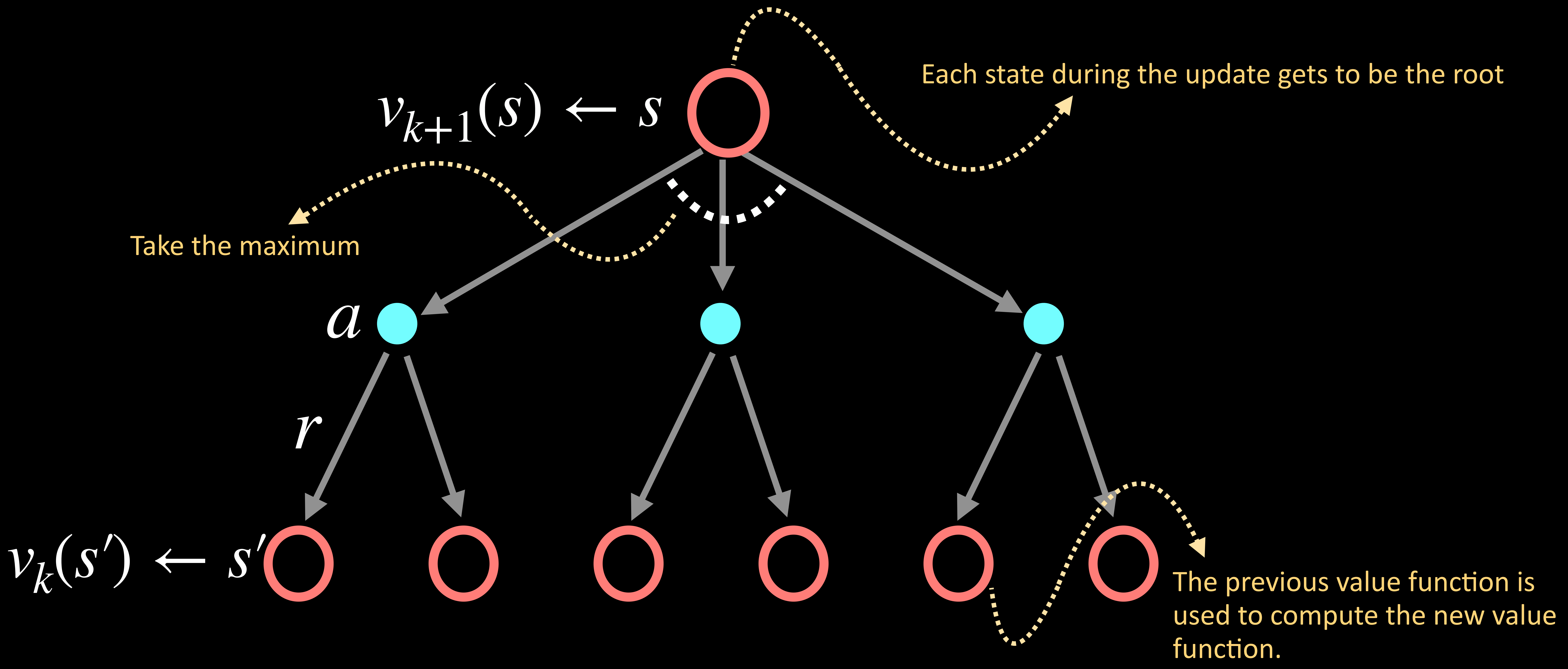
- For any state s' reachable from s
- π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$

Value iteration can be written as a simple operation that combines policy improvement and truncated policy evaluation.

Find the optimal policy π by iterative application of Bellman optimality equation.

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[r_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{\{s',r\}} p(\{s', r\} \mid s, a)[r + \gamma v_k(s')] \end{aligned}$$

The idea is to work backwards through an MDP. Start at the leaf (assume you know the optimal value here) and work your way backwards.



In the next iteration, v_{k+1} is used at the lower nodes to compute the root node i.e. the new value function.

Initialise $v(s), \forall\ s \in S$ arbitrarily, except $v(terminal) = 0$

PSUEDO CODE

POLICY EVALUATION

Loop:

$\triangle \leftarrow 0$

Loop for each $s \in S$:

$v \leftarrow v(s)$

$$v(s) \leftarrow \max_a \sum_{\{s',r\}} p(\{s', r\} \mid s, a)[r + \gamma v(s')]$$

$\triangle \leftarrow \max(\triangle , | v - v(s) |)$

until $\triangle < \theta$

a small threshold $\theta > 0$ determining accuracy of estimation

POLICY IMPROVEMENT

OUTPUT a deterministic policy $\pi \approx \pi_*$, such that,

$$\pi(s) = \operatorname{argmax}_a \sum_{\{s',r\}} p(\{s', r\} \mid s, a)[r + \gamma v(s')]$$

Policy Iteration vs Value Iteration

Includes: **policy evaluation** + **policy improvement**, and the two are repeated iteratively until policy converges.

It is based on the Bellman Expectation equation

The computation alternates between value and policy.

Every v from the loop corresponds to a valid policy π.

Includes: **finding optimal value function** + one **policy extraction**. There is no repetition of the two because once the value function is optimal, then the policy out of it should also be optimal

It is based on the Bellman Optimality equation

Each step gives a new value function. There is no explicit policy computed each step.

Each intermediate v may not correspond to any valid policy π.

Summary

We would like to compute the reward of a state s given a policy π i.e. find v_π

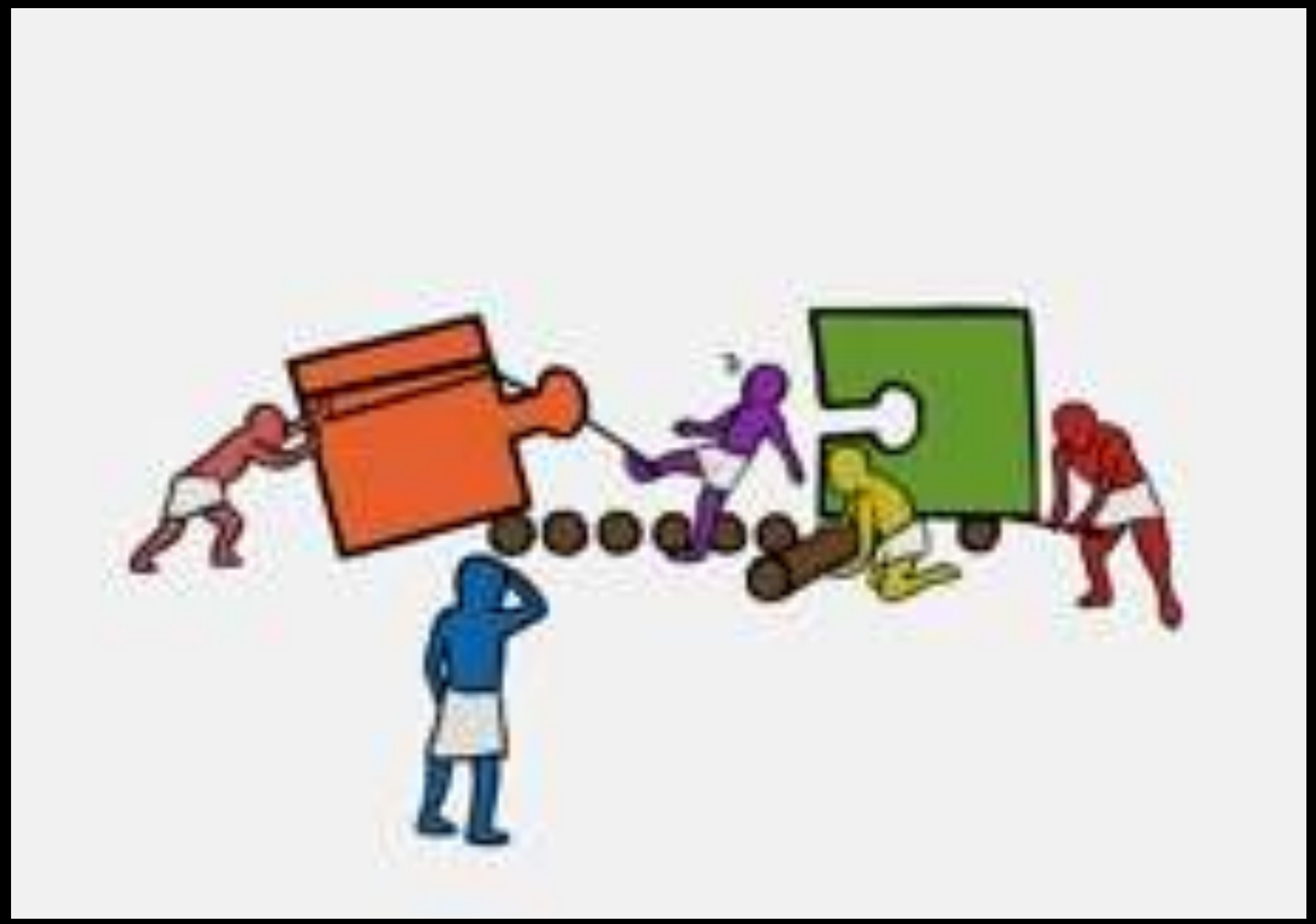
Compute v_* i.e. the best possible reward

PROBLEM	BELLMAN EQUATION	ALGORITHM
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

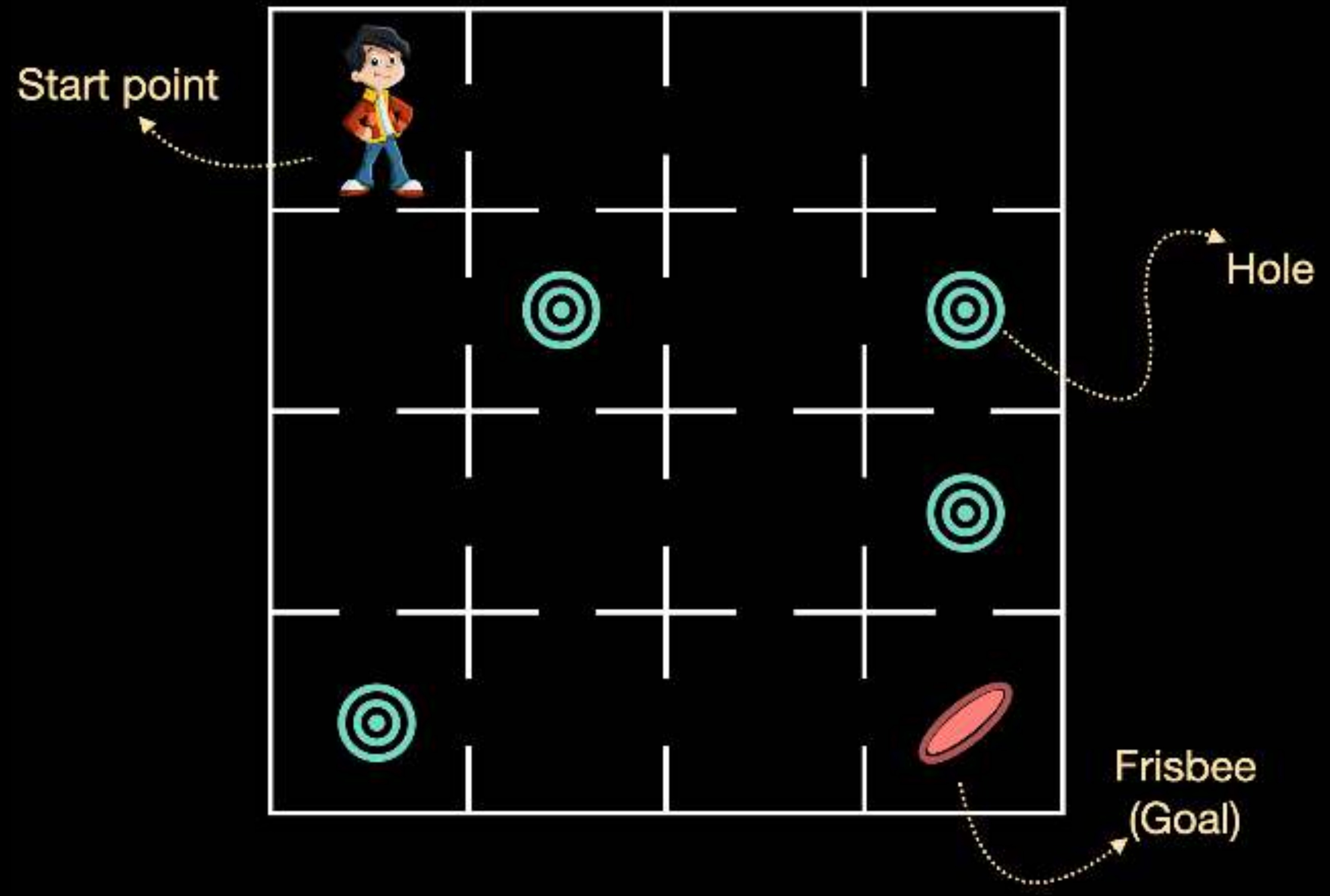
The MDP is given in all the cases here.

For *m* actions and *n* states, algorithms based on state-value function have a complexity of $O(mn^2)$

Exercise: Finding the optimal policy



The aim of this exercise is to find the optimal policy that given the maximum reward given an environment. For this, we will be using a pre-defined environment by OpenAI Gym. We will be using an environment called FrozenLake-v0.



Here, we will learn how to find an optimal policy given a policy and then the optimal value function associated to the optimal policy.