REINFORCEMENT LEARNING

Spring 2023

CS6431 Natural Language Processing

Credits

1. B2: *Machine learning: an algorithmic perspective*. 2nd Edition, Marsland, Stephen. CRC press, 2015
2. https://www.samyzaf.com/ML/rl/qmaze.html
3. http://www.gwydir.demon.co.uk/jo/maze/makemaze/index.htm
4. https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

Assignment

Read:

B2: Chapter 11

Problems:

B2:11.1, 11.2, 11.4

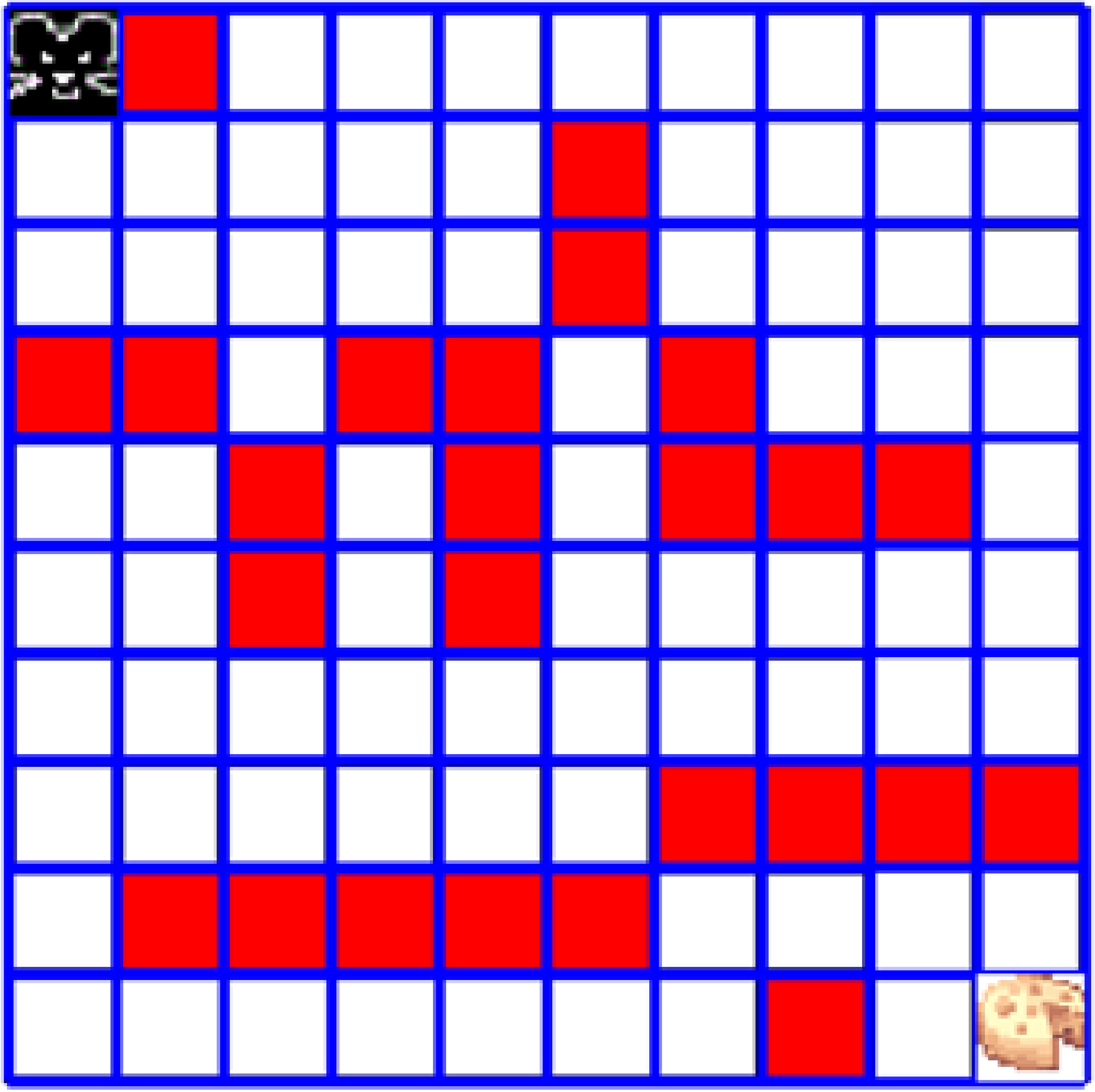
Reinforcement Learning

- Information is provided about whether or not the answer is correct, but not how to improve it.
- Reinforcement learner has to try out different strategies and see which work best.
  - Trying out $\equiv$ searching, is a fundamental part of any reinforcement learner
  - Searches over the state space of possible inputs and outputs in order to try to maximize a ***reward***.

- Reinforcement Learning: Comes from Animal intelligence
  - You repeat an action that gives you more satisfaction (or reward). Such an action gets reinforced with the situation that caused it.

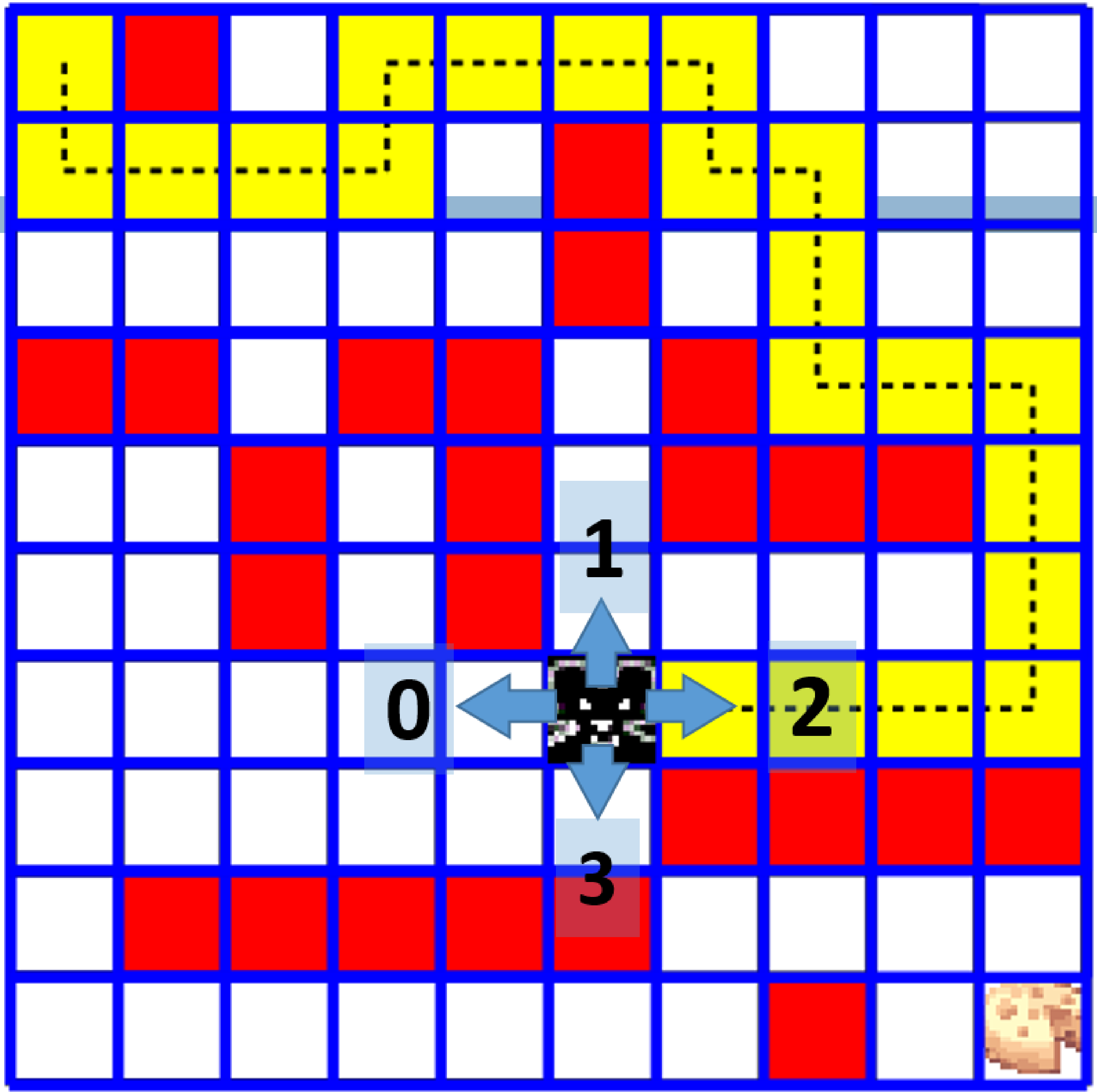
An Example

- Robot mouse has no idea of the room layout – obstructions and ways
- Robot mouse only knows how to recognize block with cheese – the end goal.

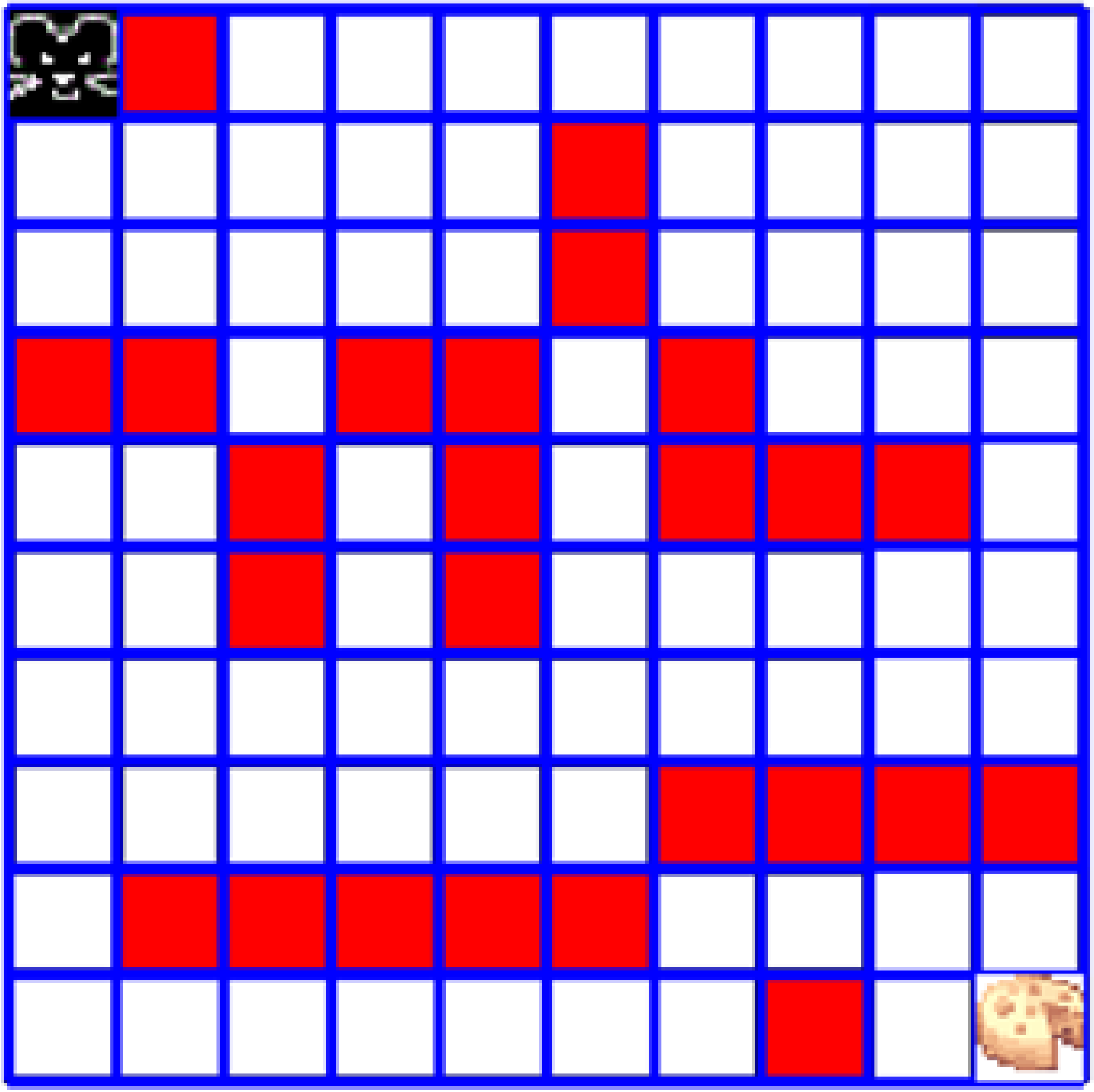


Restricted Actions

- Can move only in one of the four ways

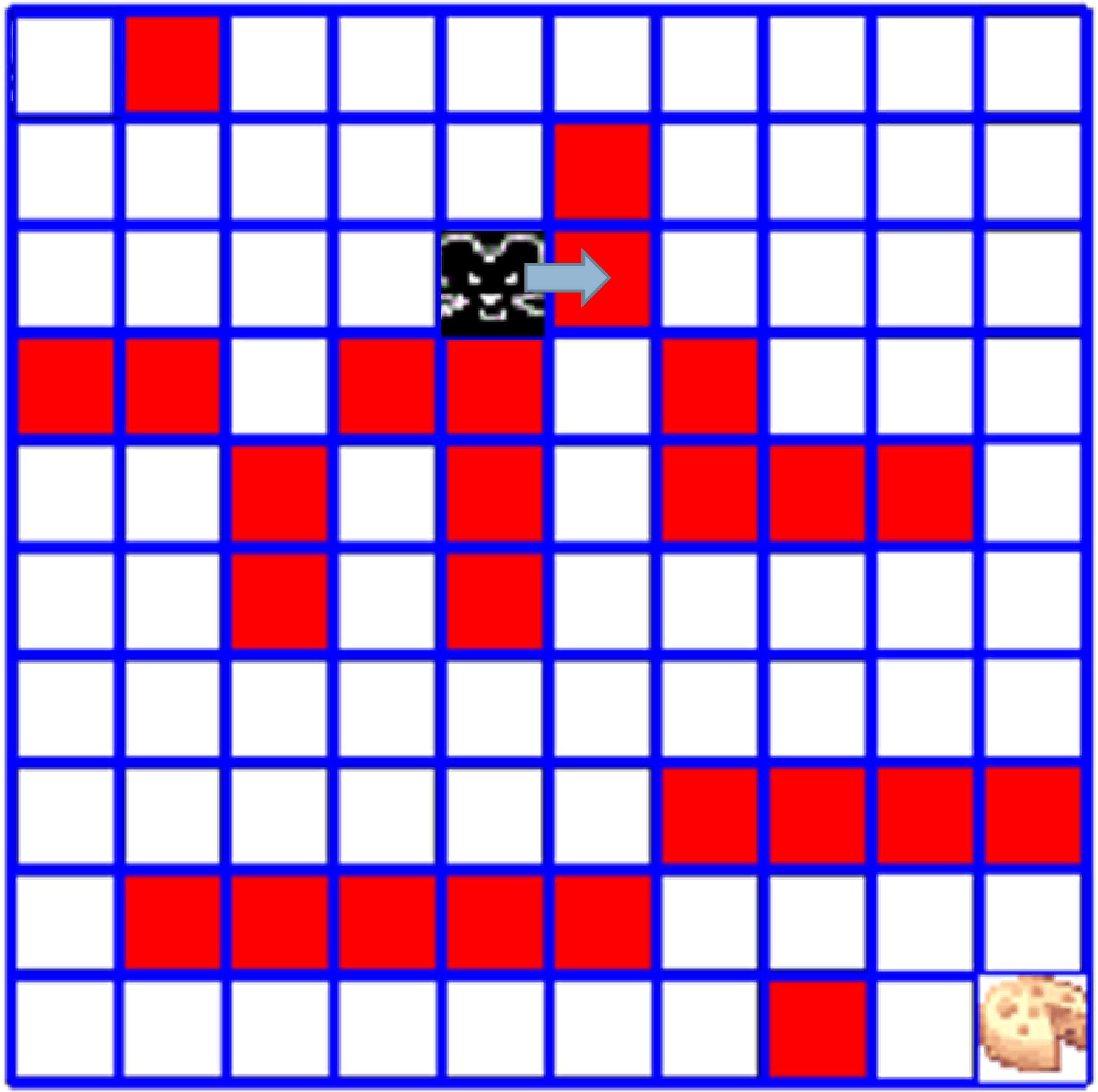


- Approach: explore and learn

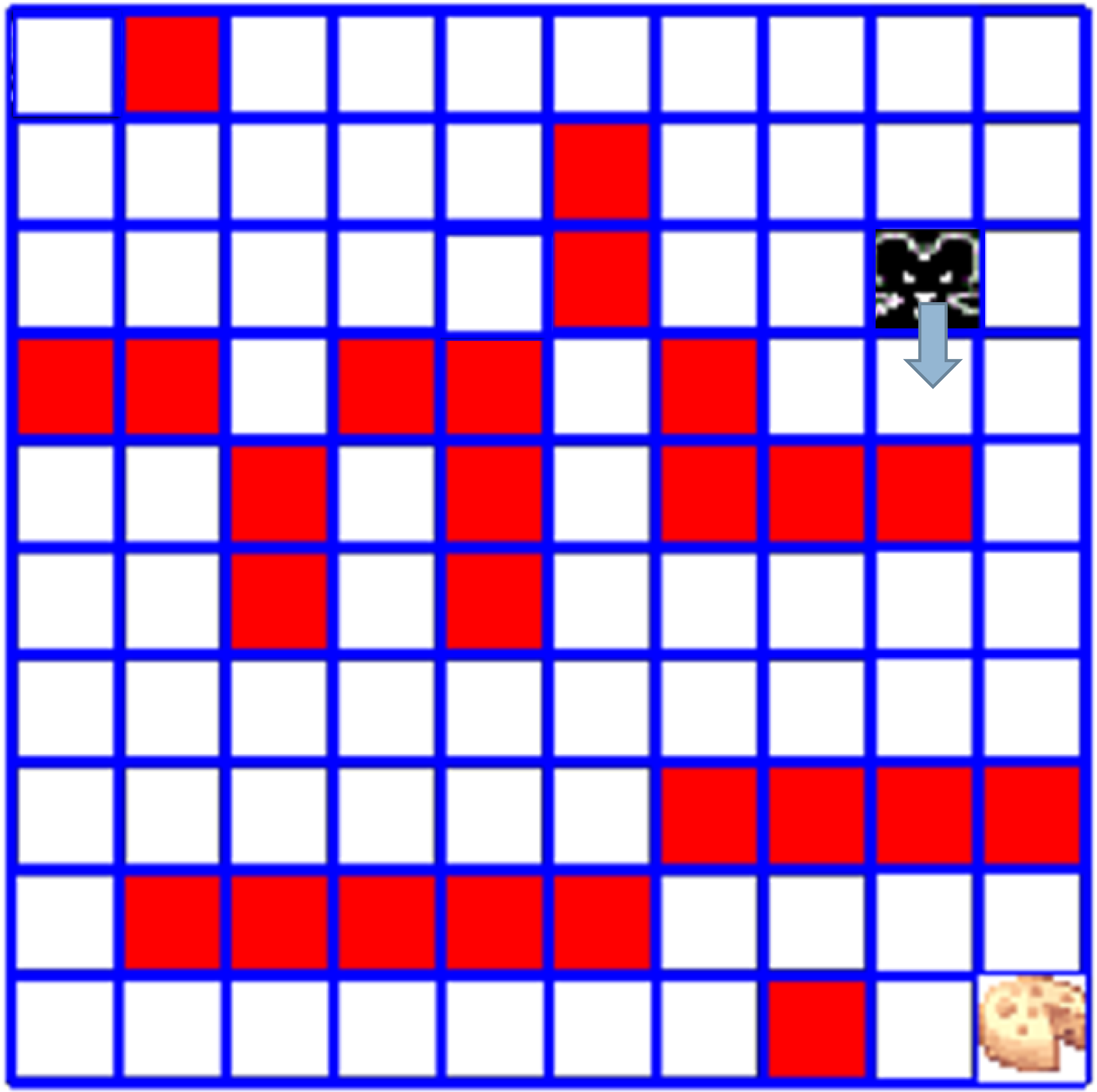


Learning Part

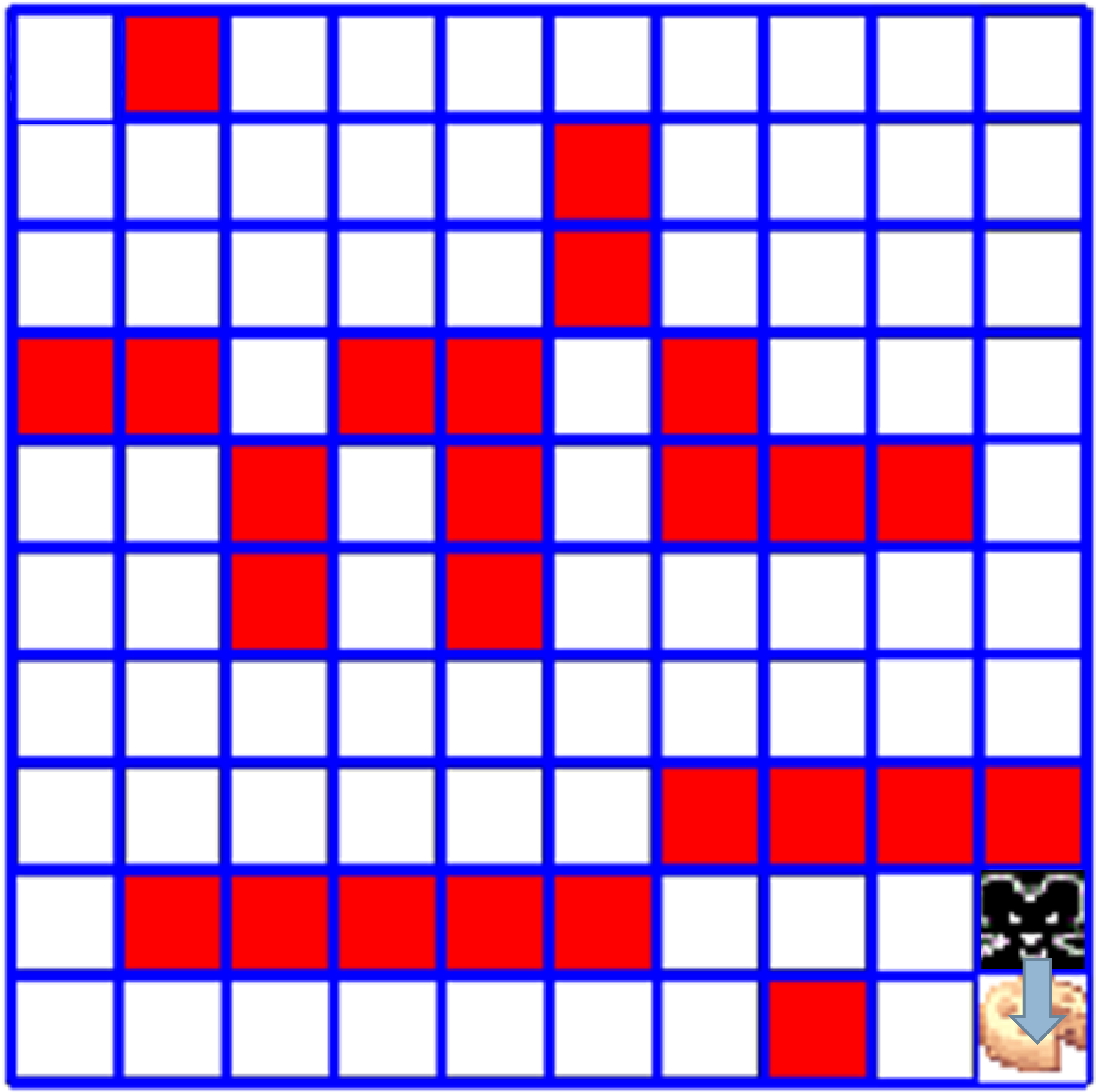
- Cell (3,5) and action 2 (going right) results in punishment
  - Punishment can be denoted by some –ve value.



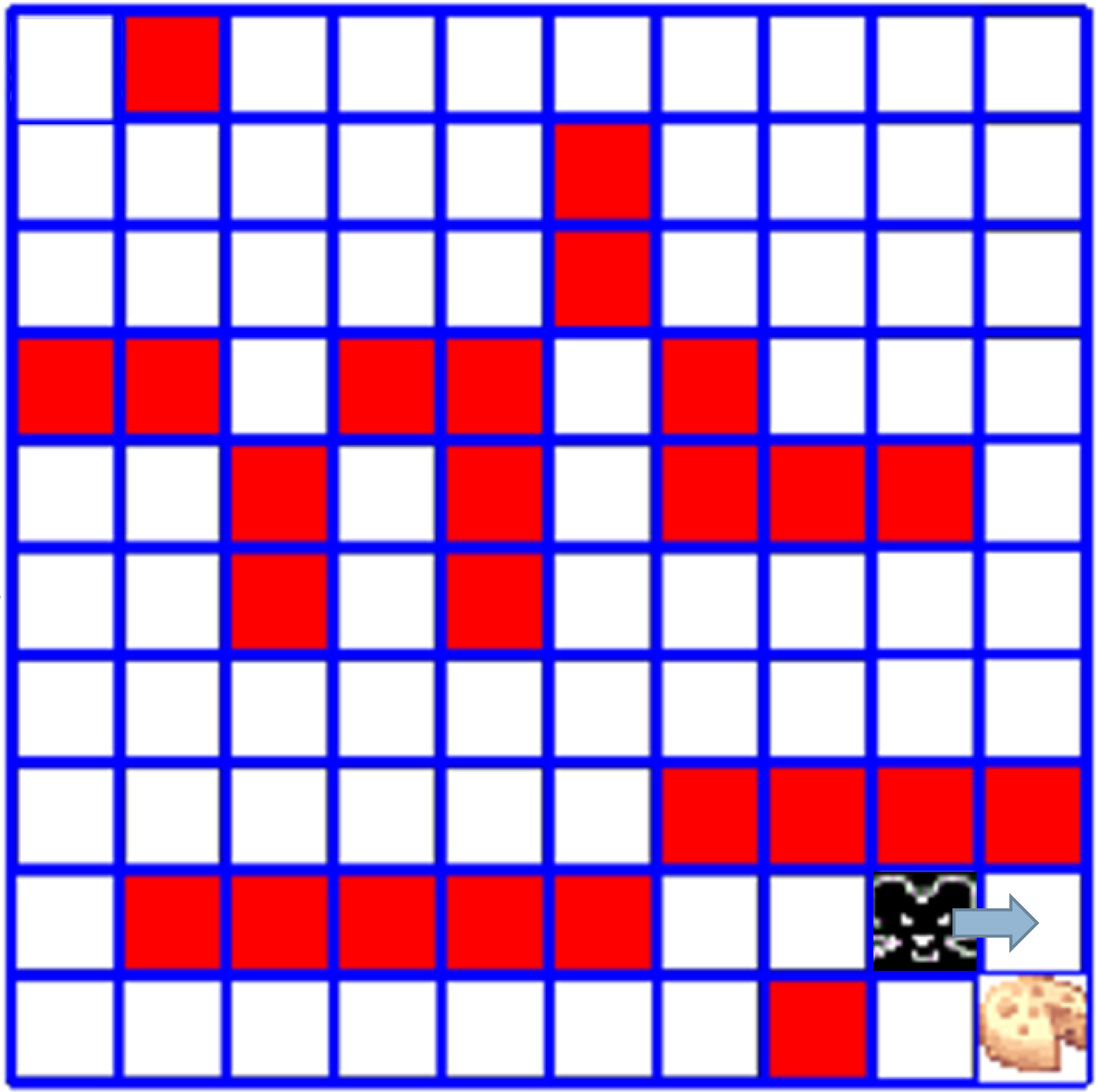
- Cell (3,9) and action 3 (going down) results in just another block
  - That is, neither hitting an obstruction nor reaching the goal
  - Can be represented by a zero value



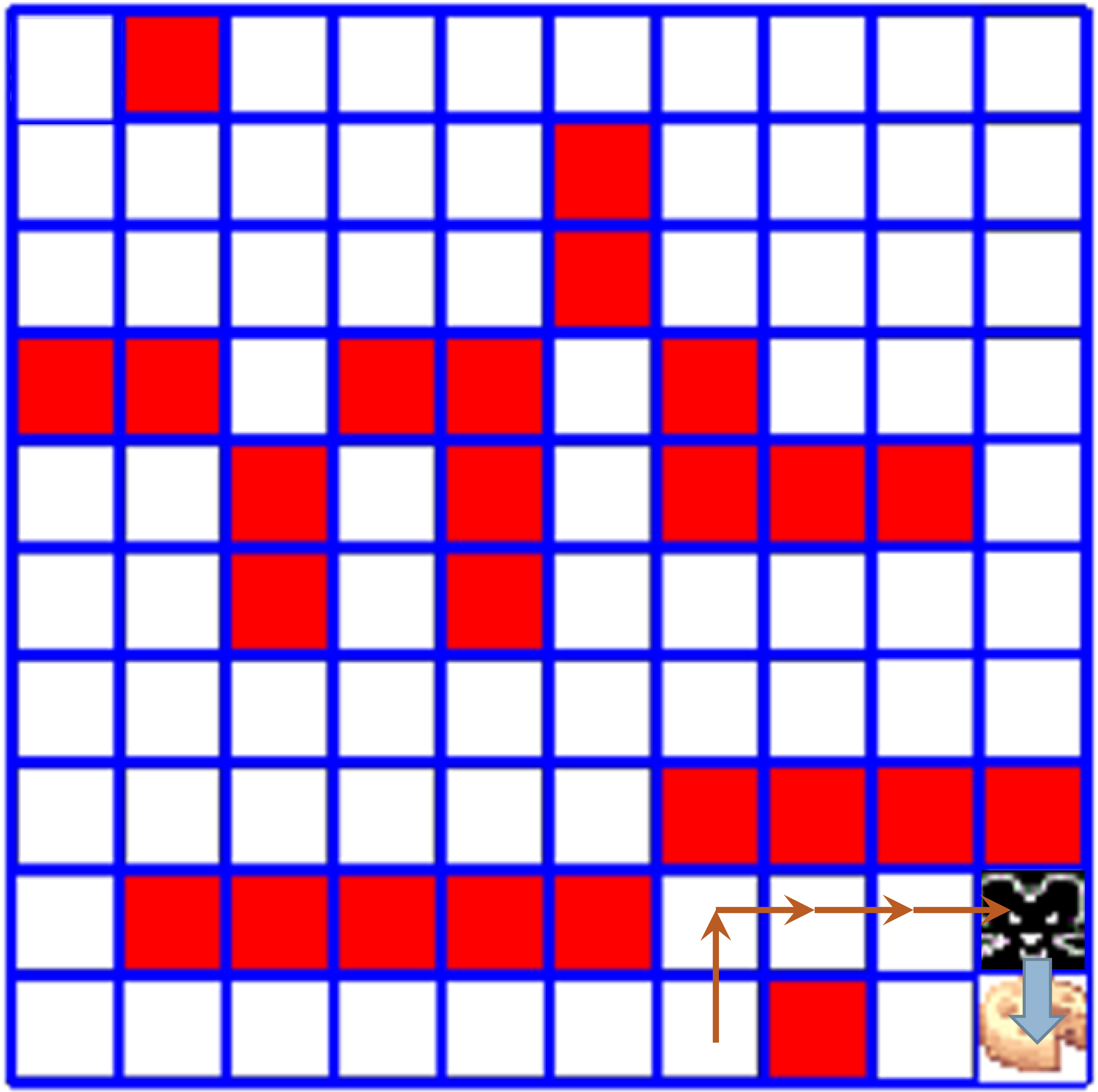
- Cell (9,10) and action 3 (going down) results in reaching the goal
  - Can be represented by a high positive value
- This also makes the cell (9,10) very valuable because it has a way to reach the goal



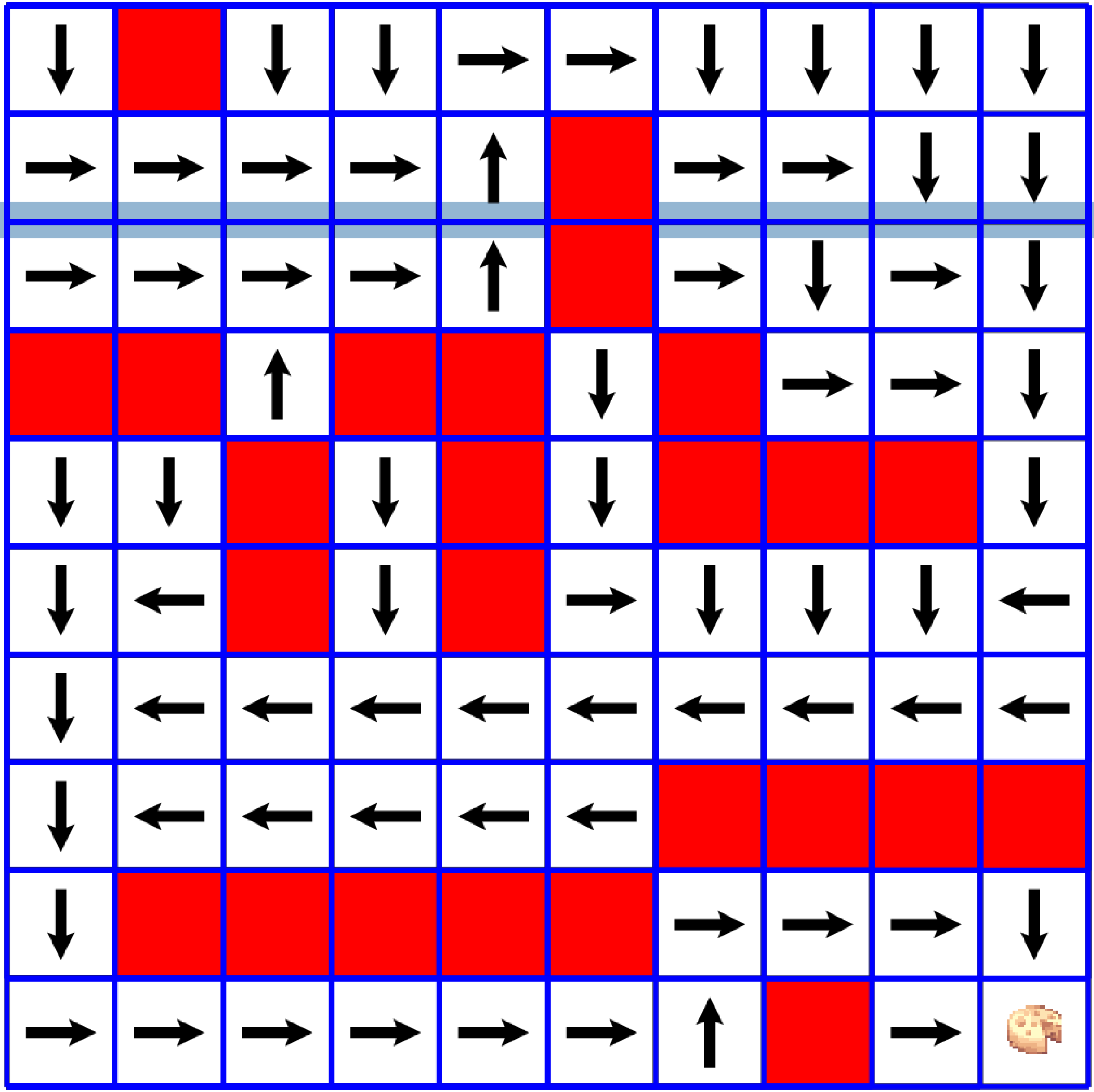
- Cell (9,9) and action 2 (going right) results in reaching the valuable cell (9,10)
  - Can be given a high positive value
- Hence, cell (9,9) also becomes valuable.



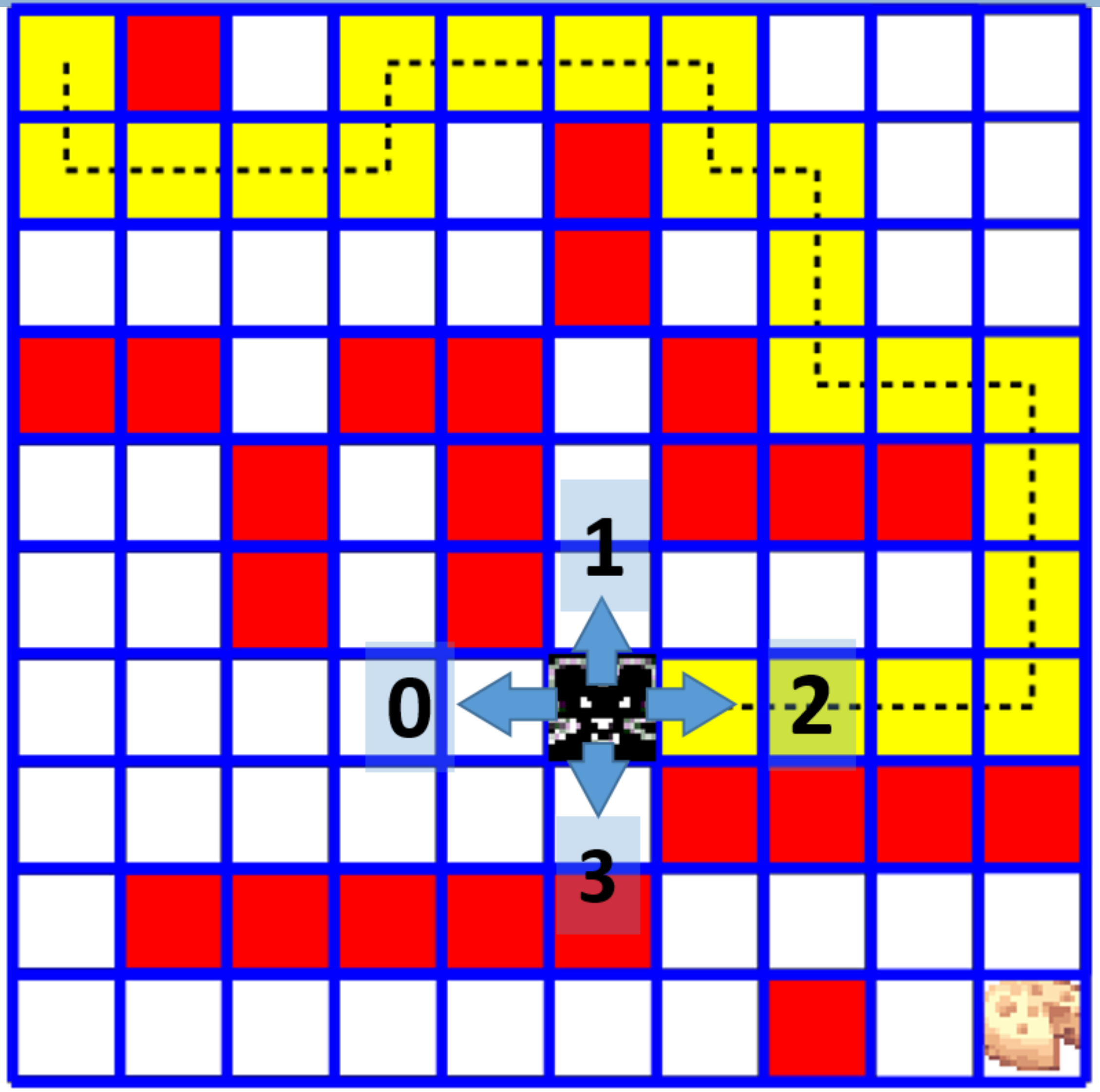
- How much “history” we want to maintain is also a choice.



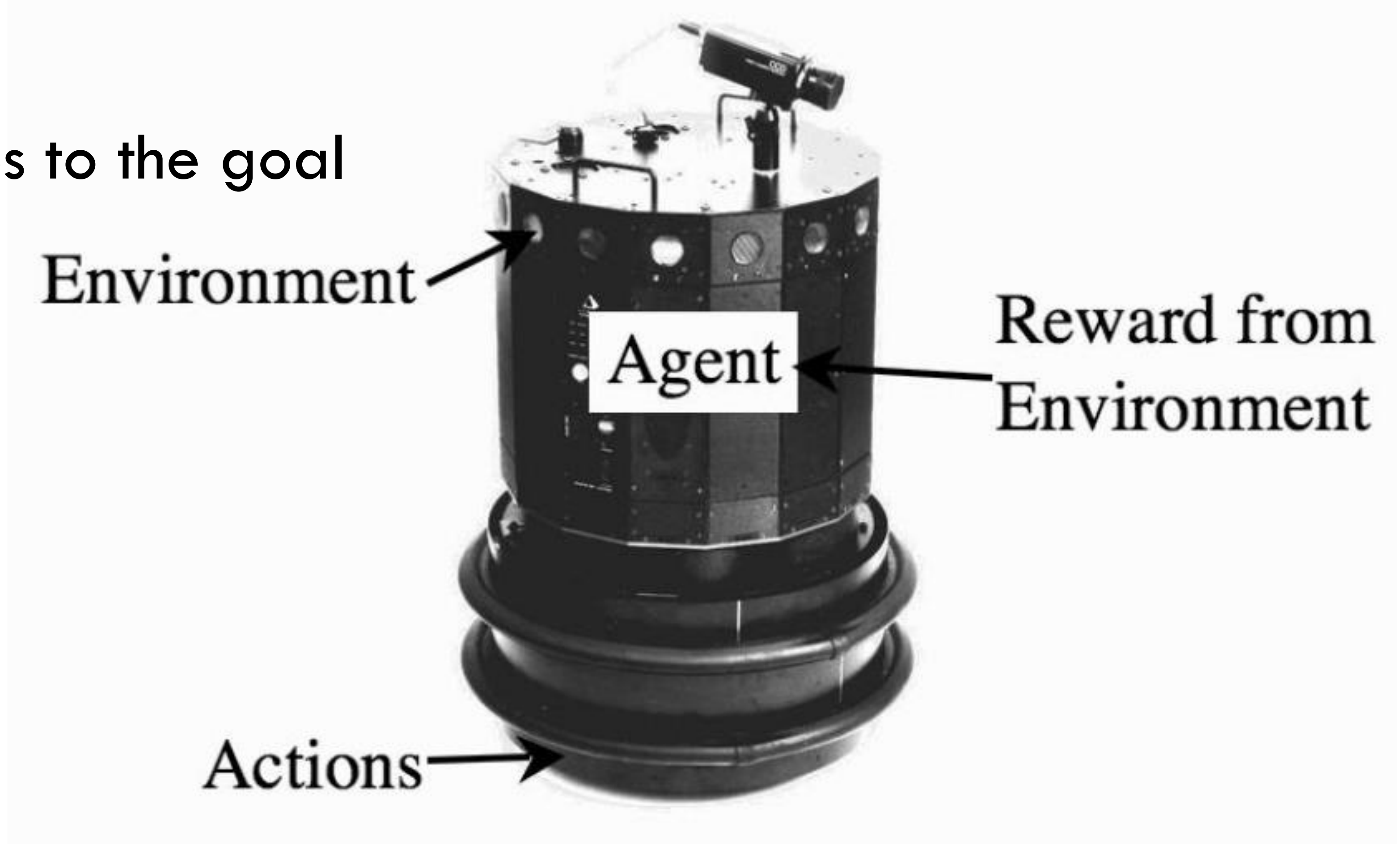
- After learning sufficiently, the best action for each state will look something like this

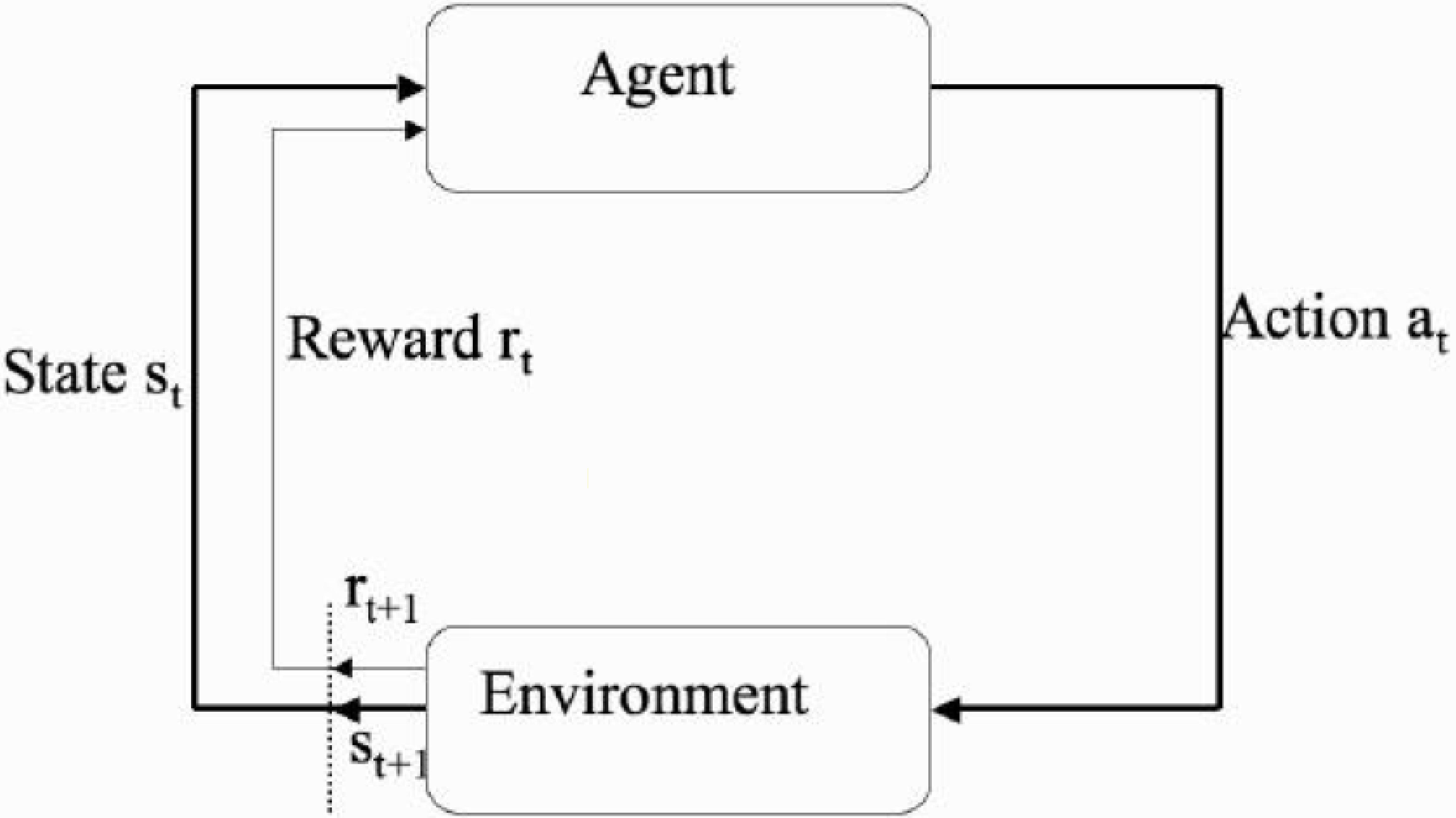


- Three things
  - Current input or the ***state***
  - Possible things that can be done or the ***actions***
  - Aim is to maximize ***reward***
- ***Agent***: that is doing or learning
- ***Environment***: where the agent acts



- State: sensor readings
  - May not tell everything to the robot
  - There may be noise/inaccuracies
- Actions: possible ways in which the robot can drive its motors
  - Actions on the environment
- Reward: how well it navigates to the goal with minimal crashing

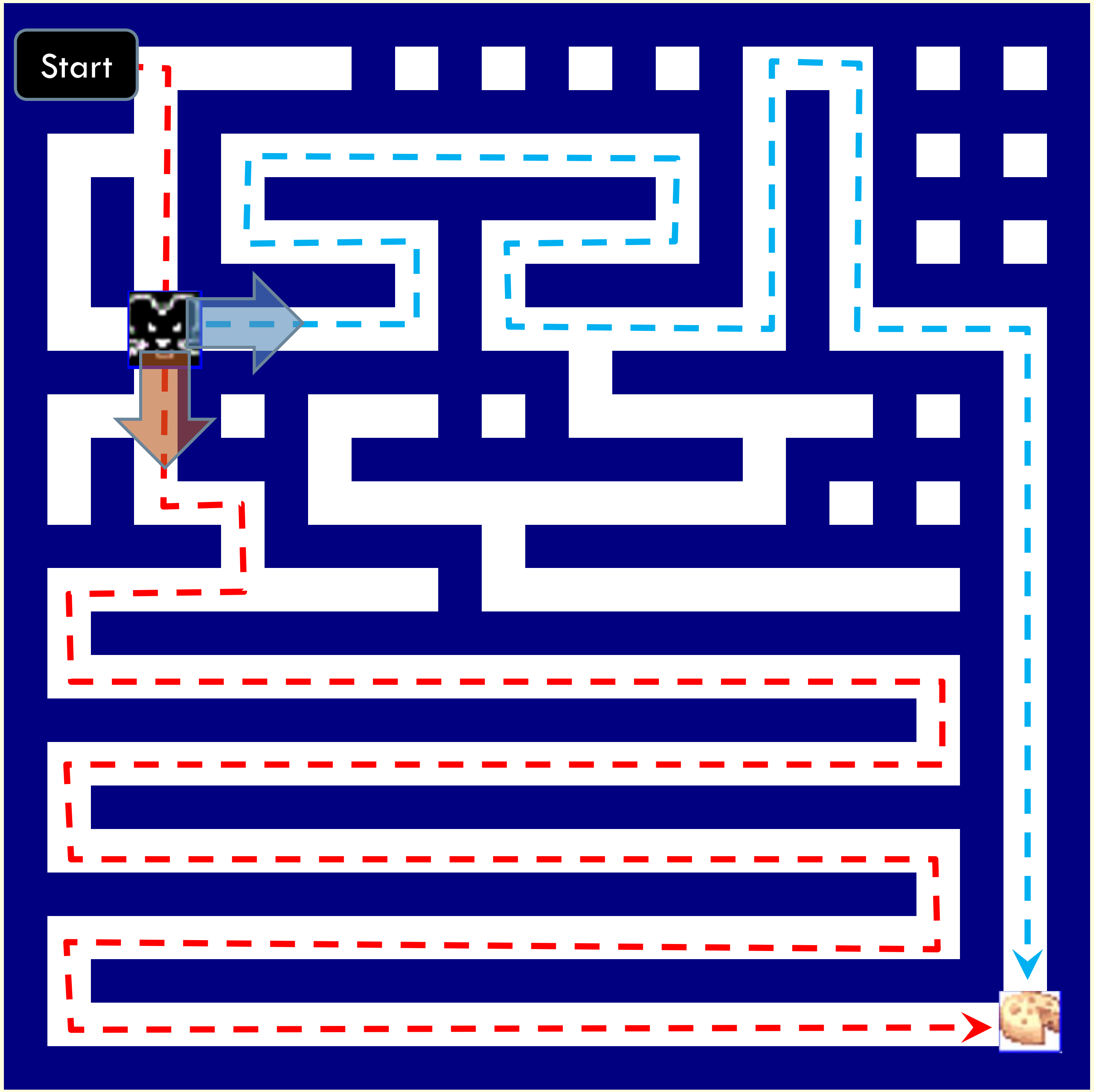



Agent
State s_t
Reward r_t
Action a_t
r_{t+1}
Environment
s_{t+1}

- Reward can be delayed or it can be instant
  - E.g. a robot travelling a maze vs. a robot driving a car
- ***Policy:*** Choice of action that should be taken
  - Should be a combination of exploitation and exploration

Policy: exploitation vs. exploration

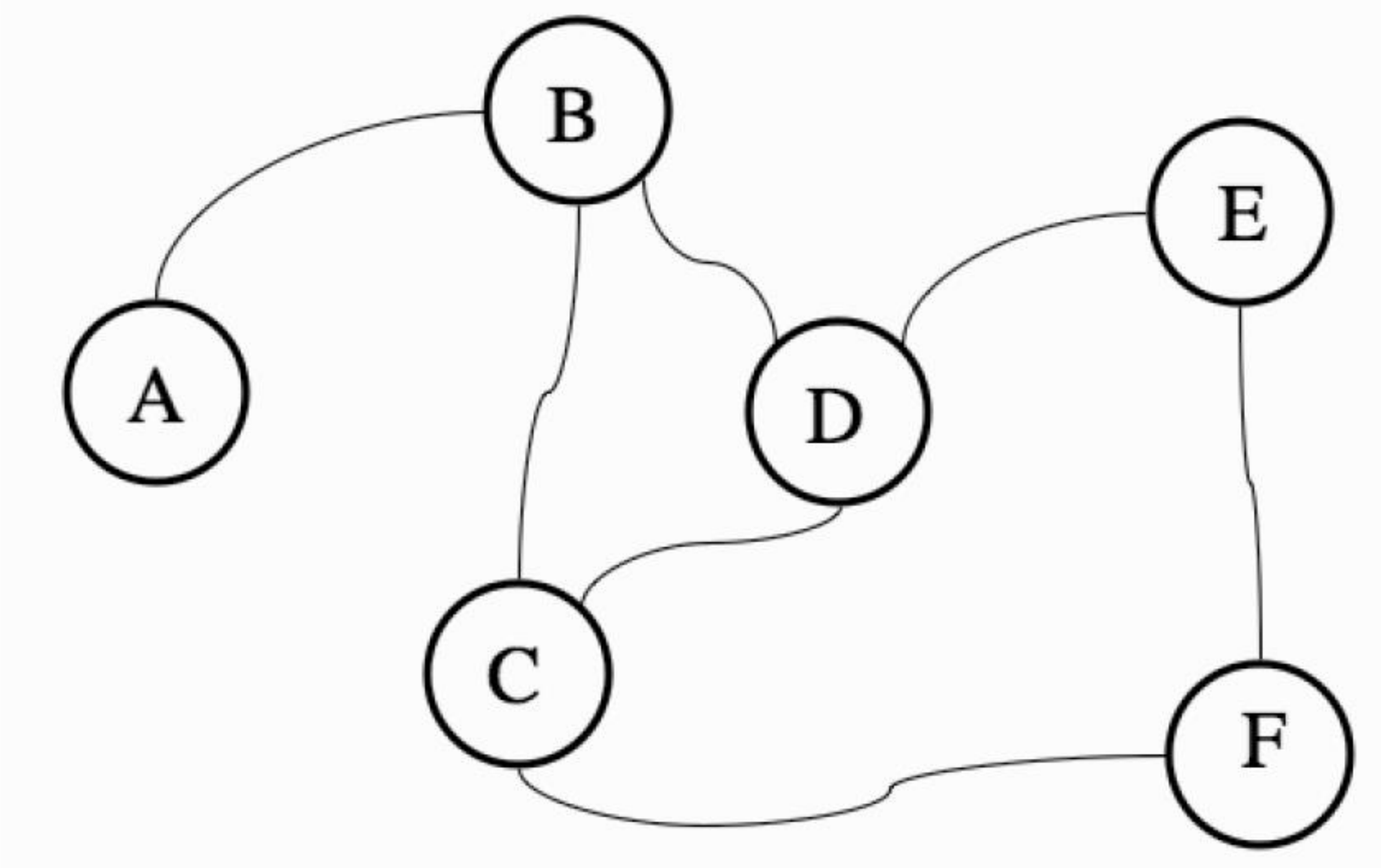
- Exploitation
  - Take the best action learnt so far
- Exploration
  - Give sub-optimal actions a chance



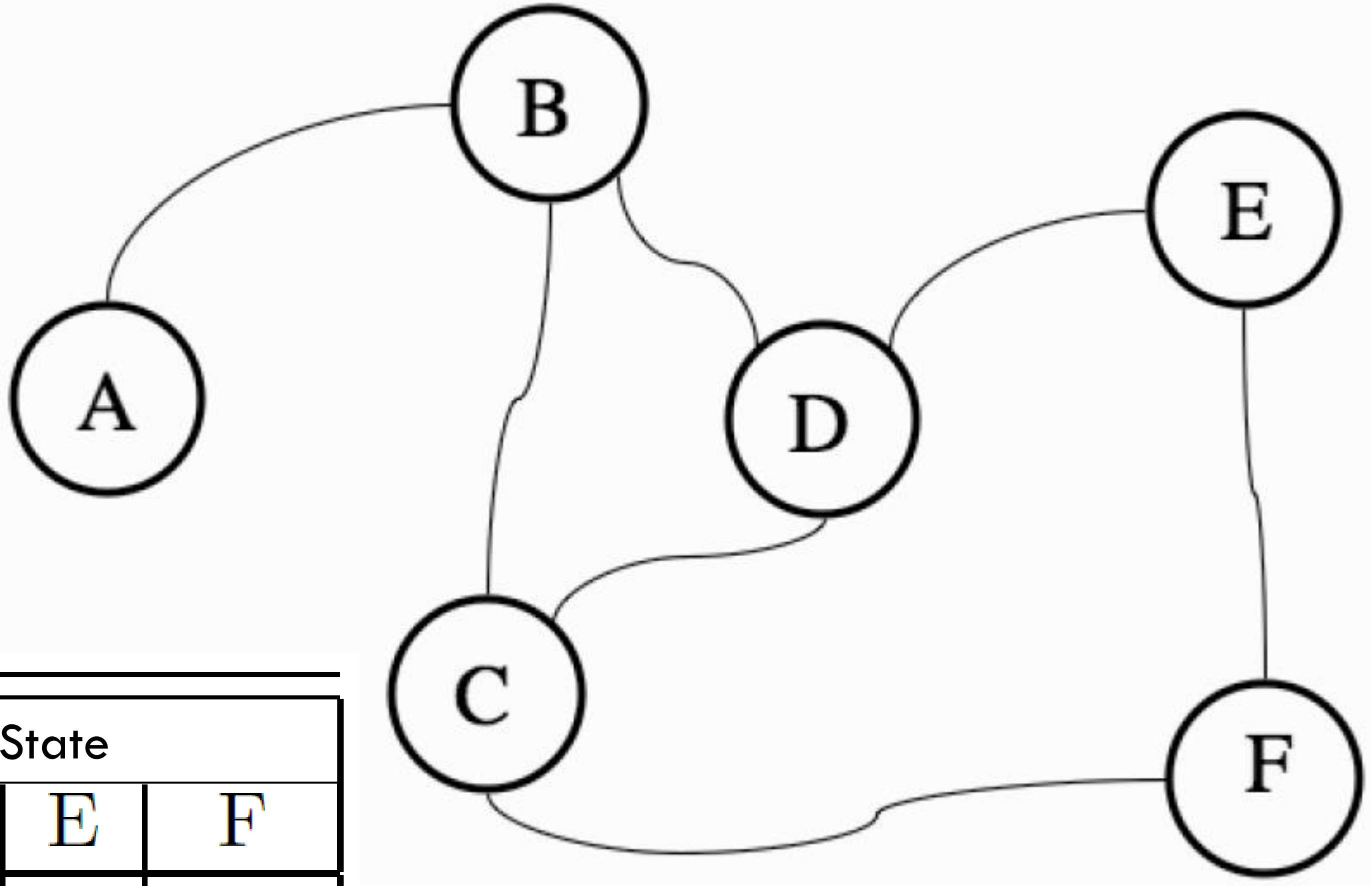
- Reward can be delayed or it can be instant
  - E.g. a robot travelling a maze vs. a robot driving a car
- ***Policy:*** Choice of action that should be taken
  - Should be a combination of exploration and exploitation
- ***Absorbing state***: your goal state, when the solution you were searching for is found
  - Also called ***Terminal*** or ***Accepting*** state.
- ***State space***: set of all states that the learner can experience
- ***Action space:*** set of all actions that the learner can take

EXAMPLE: GETTING LOST

- You know that F is the absorbing state ,because you are vaguely familiar with it
- You decide to reward yourself with chips only if you reach F



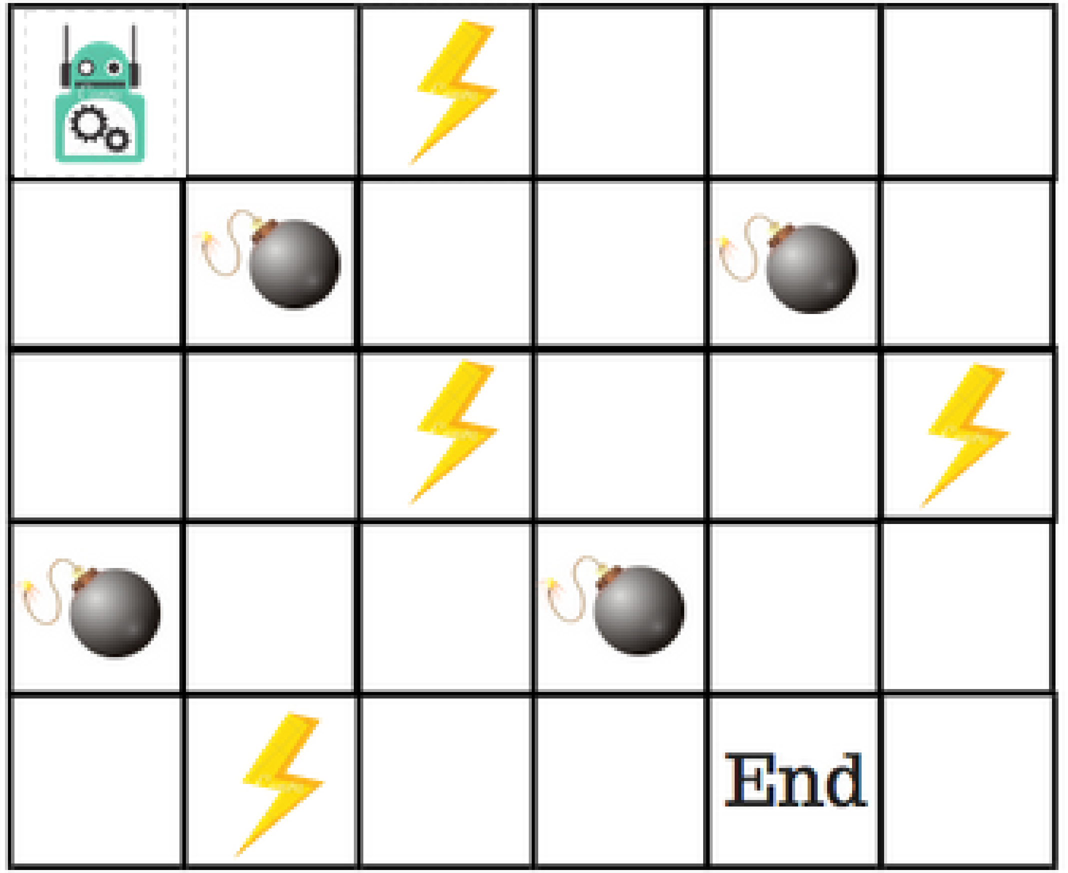
- You don't have the city map with you, hence, you are not aware of the following ***reward matrix***



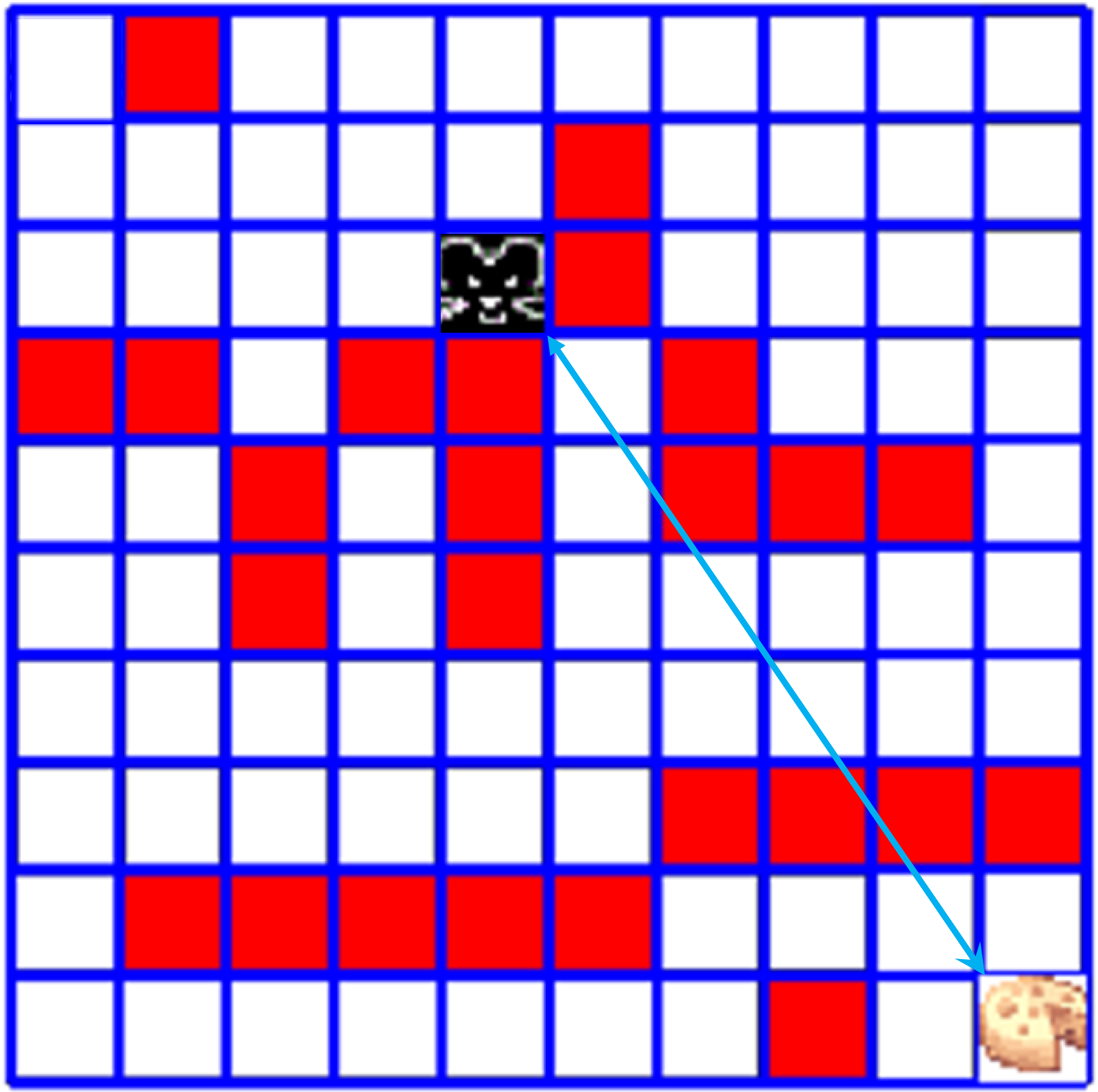
Current State	Action or Next State					
	A	B	C	D	E	F
A	-5	0	-	-	-	-
B	0	-5	0	0	-	-
C	-	0	-5	0	-	100
D	-	0	0	-5	0	-
E	-	-	-	0	-5	100
F	-	-	0	-	0	-

Carrots and Sticks: The Reward Function

- Reward Function
  - Input: Current State and Chosen Action
  - Output: Numerical reward based on them
- It is generated by the environment around the learner
  - It is not internal to the learner
- Two parts
  - An intermediate part (at every step or so)
  - A pay-off in the end



- The reward tells the learner what the goal is, not how the goal should be achieved, which would be supervised learning.
  - Usually a bad idea to include sub-goals like speed up of learning
  - Learner can find methods of achieving sub-goals without actually achieving the real goal



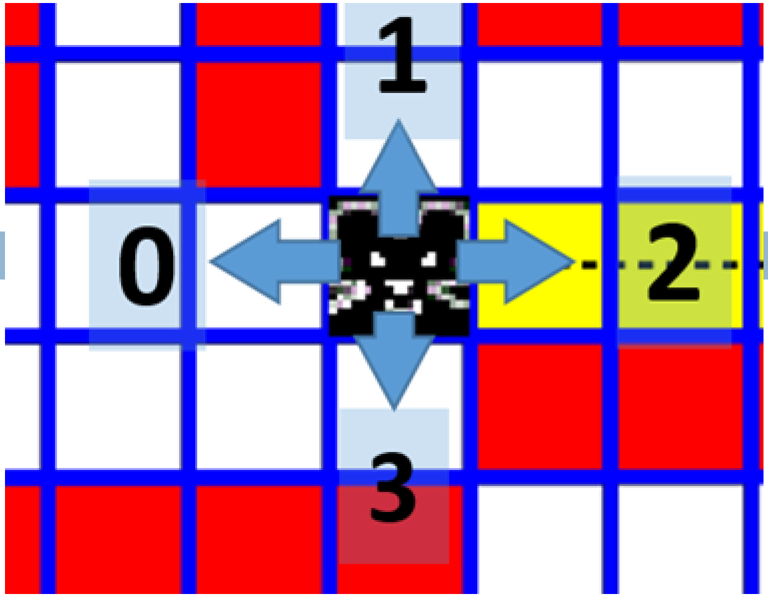
- For continual tasks (e.g., learning to walk): there is no terminal state
- In general, we want to predict the reward into the infinite future
- Solution: ***Discounting***:
  - We discount future rewards depending upon how "far" they are in the future

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{k-1} r_k + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Where, $0 < \gamma < 1$ is the discounting factor

write on desk

Action Selection



- Each action is assigned a value based on
  - The current state
  - How it has been rewarded in the past
- A reward action a has received on being taken t times form the state s: $Q_{s,t}(a)$
  - Will eventually converge to true predictions

Based on prediction of $Q_{s,t}(a)$, there are three methods of choosing a

- **Greedy**: pick action with highest $Q_{s,t}(a)$
- **ϵ-Greedy**: similar to greedy, but small probability where we pick some other action at random
  - Works better than greedy in practice.
- ***Soft-max***: refinement of ϵ-Greedy

$$P(Q_{s,t}(a)) = \frac{\exp(Q_{s,t}(a)/\tau)}{\sum_b \exp(Q_{s,t}(b)/\tau)}$$

write on desk

- $\boldsymbol{\tau}$ (temperature):
  - Large τ: all actions have similar probability
  - Small τ: individual probabilities matter more

Policy

- Choice of which action to take in each state in order to get optimal results: π
  - It is a mapping from states to actions.
- Goal: learn better π for each state s_t as we proceed.
- Two things:
  - How much past information we need to know while getting into the current state (use of Markov Decision Process)
  - How we assign a value to the current state

MARKOV DECISION PROCESSES

- Is the information of current state sufficient to compute reward for the next move (e.g., chess)
  - Such a state is called ***Markov state***

$$Pr(r_t = r', s_{t+1} = s' | s_t, a_t)$$

write on desk

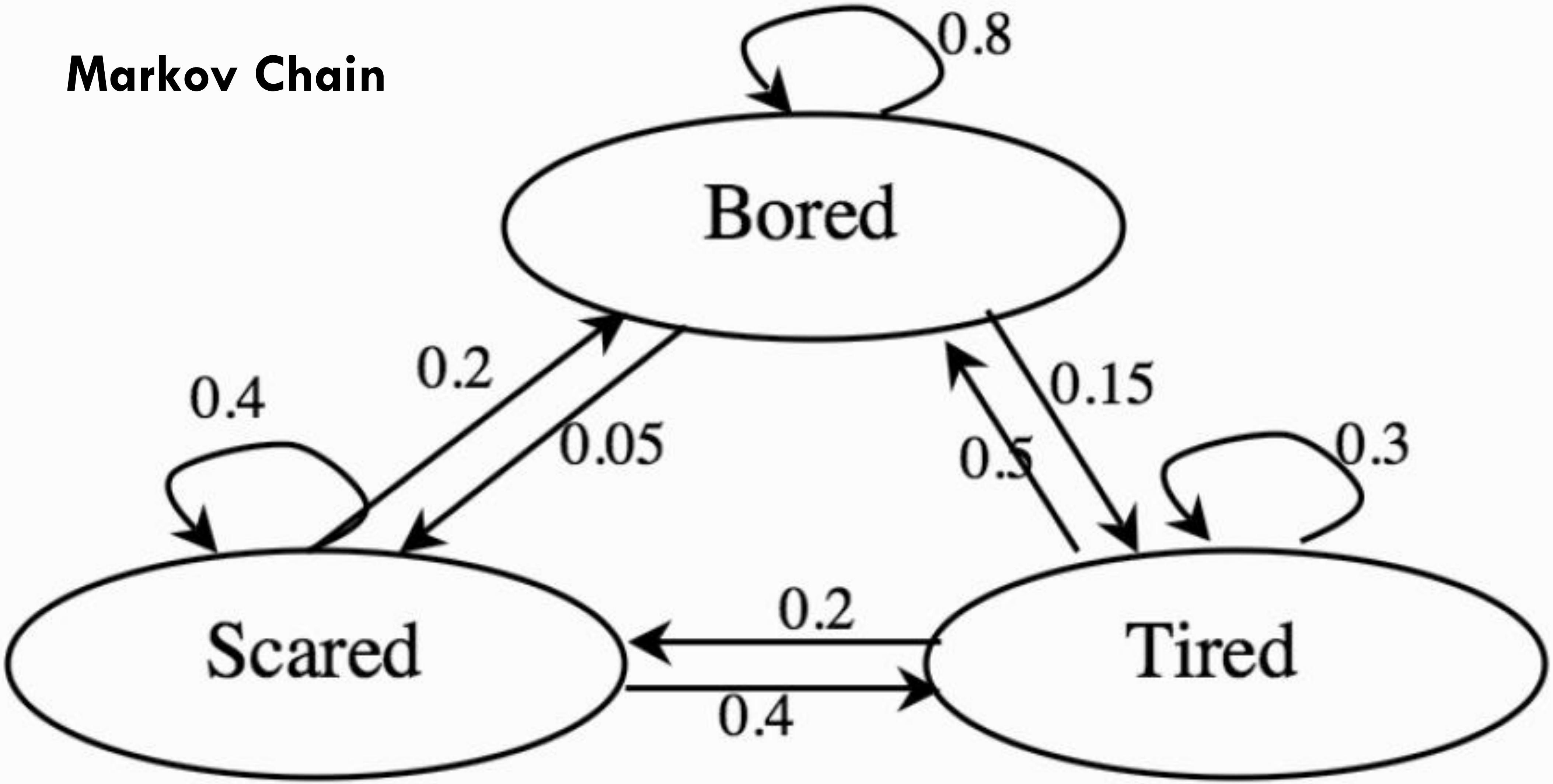
OR

- We need to consider how we got to the current state, i.e., a sense of history needs to be stored.

$$Pr(r_t = r', s_{t+1} = s' | s_t, a_t, r_{t-1}, s_{t-1}, a_{t-1}, \ldots r_1, s_1, a_1, r_0, s_0, a_0)$$

- A reinforcement learning problem that follows Markov state property is known as a ***Markov Decision Process (MDP)***
- The number of possible states and actions are assumed to be finite.
- Markov Chain: States + transition probabilities
- Markov Decision Process: Extension of Markov Chain with actions and rewards

Markov Chain

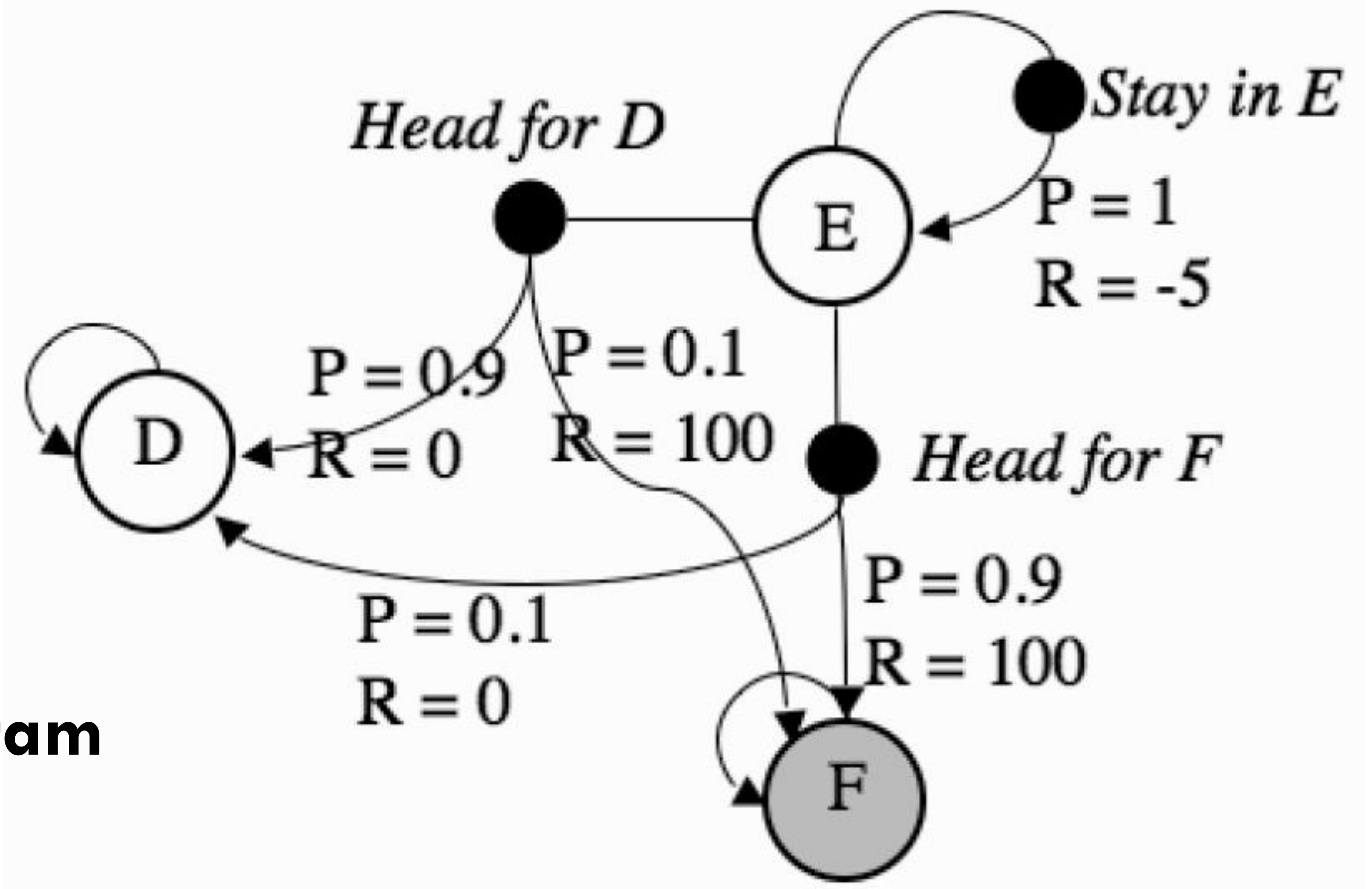


A simple example of a Markov decision process to decide on the state of your mind tomorrow given your state of mind today.

Markov Decision Process

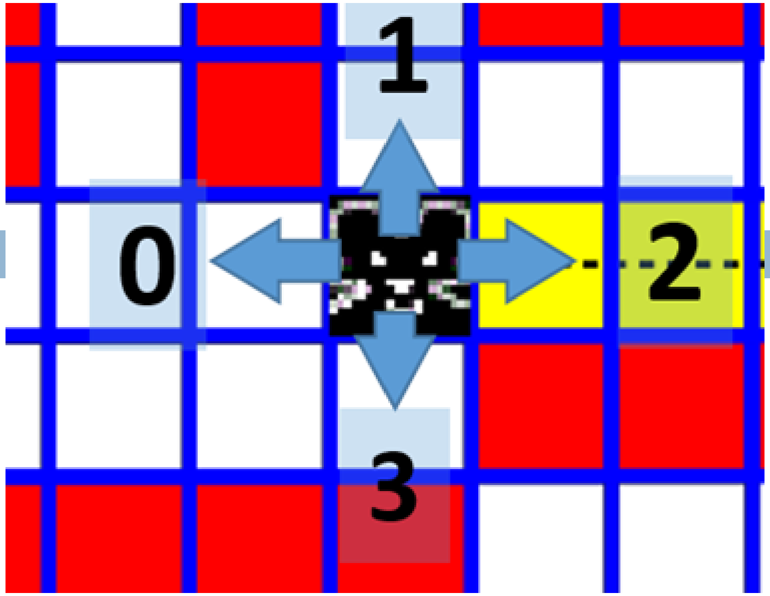
- Addition: Selecting an action does not guarantee a state.
- We add action nodes coming out of state nodes.
- Multiple state nodes can be connected to an action node.

Transition Diagram



MDP for getting lost example

VALUES



- ***Value*** denotes the expected future reward the reinforcement learner is trying to maximize
- Two ways:
  - ***State-value function*** $V(s)$: average the reward across all of the actions that can be taken

$$V(s) = E(r_t | s_t = s) = E\left\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s\right\}$$

write on desk

  - ***Action-value function*** $Q(s, a)$: consider each possible action that can taken from the current state separately.

$$Q(s,a) = E(r_t | s_t = s, a_t = a) = E\left\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s, a_t = a\right\}$$

- Two ways:
  - ***State-value function*** $V(s)$: average the reward across all of the actions that can be taken

$$V(s) = E(r_t | s_t = s) = E\left\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s\right\}$$

  - ***Action-value function*** $Q(s,a)$: consider each possible action that can taken from the current state separately.

$$Q(s,a) = E(r_t | s_t = s, a_t = a) = E\left\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s, a_t = a\right\}$$

  - State-value function is less accurate but easier to compute
  - Action-value function is more accurate but requires more information to compute

- Now, we have two problems
  - Deciding which action to take, i.e., the policy
  - Predicting the value function
- Optimal policy, π^*, is the one (not necessarily unique) in which the value function is the greatest over all possible states.
- Optimal value functions can be of two types: V^(s) = max_a(Q^(s, a))
  - Optimal state value function: write on desk

$$V^*(s) = \max_\pi V^\pi(s) \text{ for all possible states } s$$

  - Optimal action value function:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \text{ for all possible states } s \text{ and actions } a$$

- Optimal state value function:

$$V^*(s) = \max_\pi V^\pi(s) \text{ for all possible states } s$$

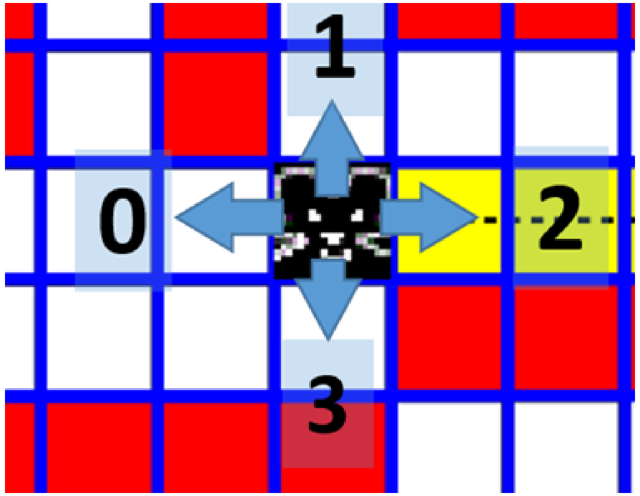
  - Assumes taking the optimal action in each case

- Optimal action value function:

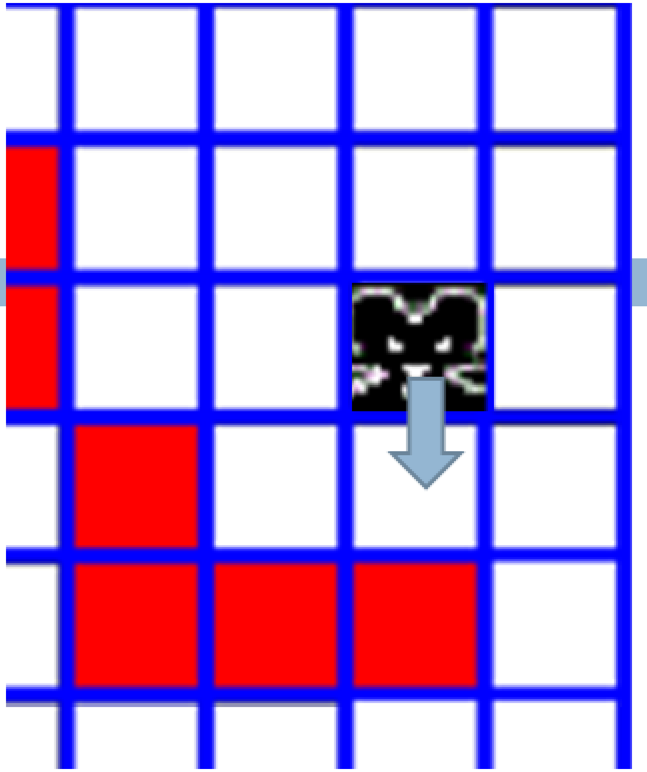
$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \text{ for all possible states } s \text{ and actions } a$$

  - Considers taking action a this time, and then following the optimal policy from then on

- Hence, optimal action value = current reward + discounted estimate of the future reward



$$\begin{aligned} Q^*(s,a) &= E(r_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \\ &= E(r_{t+1}) + \gamma V^*(s_{t+1} | s_t = s, a_t = a) \end{aligned}$$



- How do we update values?

Updated Estimate ← Current Estimate +

μ(New Estimate – Current Estimate)

Temporal Difference (TD) method

$$V(s_t) \leftarrow V(s_t) + \mu(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

$$Q(s,a) \leftarrow Q(s,a) + \mu(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Write on desk

- If we are remembering previous λ states and also updated their value functions, then the above algorithm is called $TD(\lambda)$.

The Q-Learning Algorithm TD(0)

- **Initialisation**
  - set $Q(s, a)$ to small random values for all s and a
- Repeat:
  - initialise s
  - repeat:
    - $*$ select action a using ϵ-greedy or another policy
    - $*$ take action a and receive reward r
    - $*$ sample new state s'
    - $*$ update $Q(s, a) \leftarrow Q(s, a) + \mu(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
    - $*$ set $s \leftarrow s'$
  - For each step of the current episode
- Until there are no more episodes

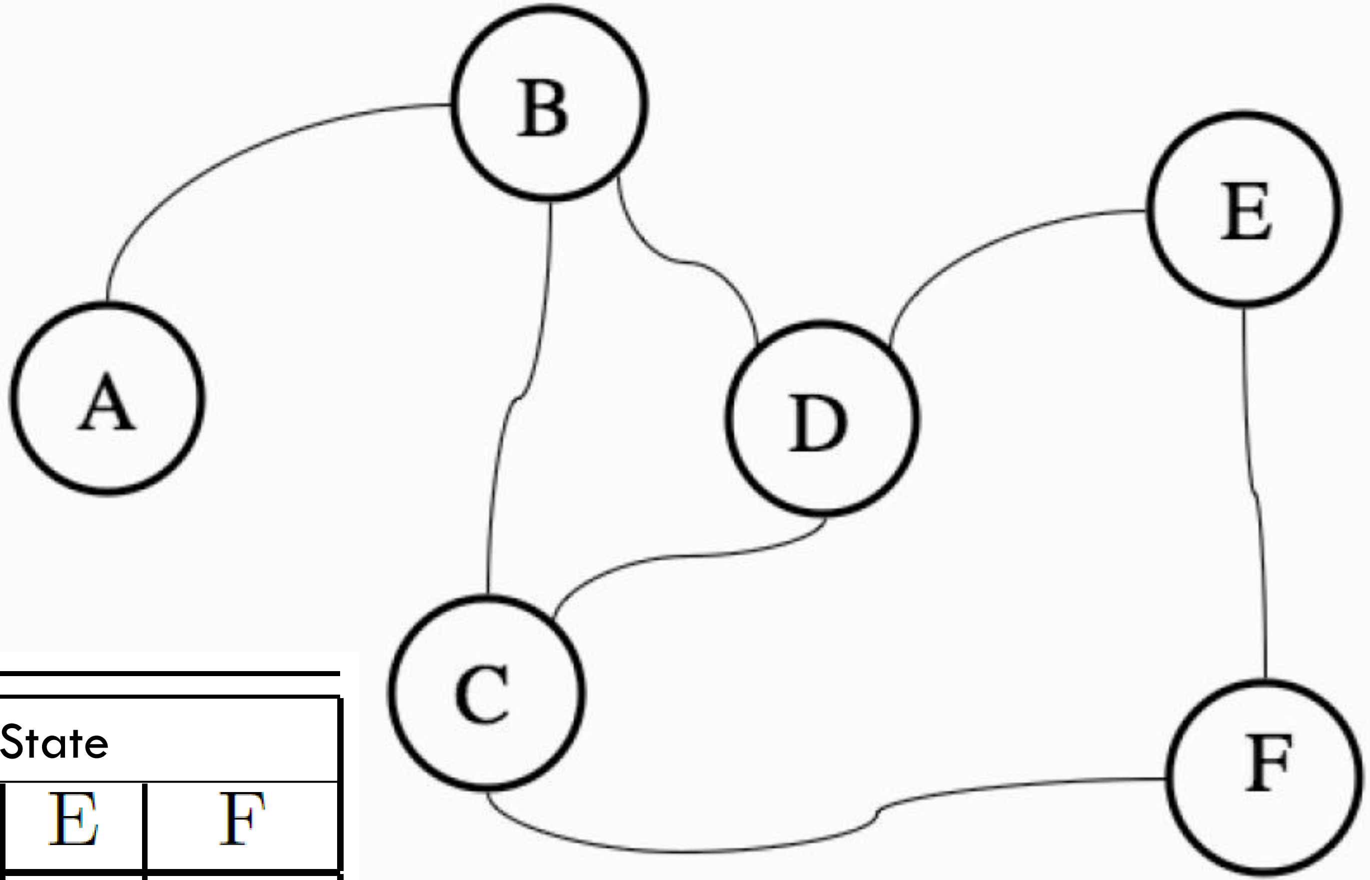
Off-policy decision

The Sarsa Algorithm

- **Initialisation**
  - set $Q(s, a)$ to small random values for all s and a
- Repeat:
  - initialise s
  - choose action a using the current policy
  - repeat:
    - take action a and receive reward r
    - sample new state s'
    - choose action a' using the current policy
    - update $Q(s, a) \leftarrow Q(s, a) + \mu(r + \gamma Q(s', a') - Q(s, a))$
    - $s \leftarrow s'$, $a \leftarrow a'$
  - for each step of the current episode
- Until there are no more episodes

On-policy decision

Recall Getting Lost Example



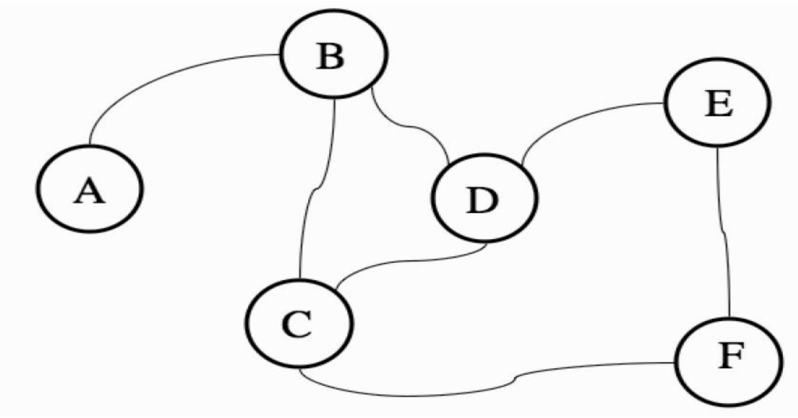
Reward Matrix

Current State	Action or Next State: A	B	C	D	E	F
A	-5	0	-	-	-	-
B	0	-5	0	0	-	-
C	-	0	-5	0	-	100
D	-	0	0	-5	0	-
E	-	-	-	0	-5	100
F	-	-	0	-	0	-

Q-learning Example

- For the getting lost example, consider the following trail of steps for an iteration: A,B,A,B,D,D,B,C,F

And the following initial Q matrix:



States	Action or Next State					
	A	B	C	D	E	F
A	-0.039	-0.028	-0.026	-0.018	0.013	0.011
B	0.028	-0.029	0.024	0.044	0.013	-0.026
C	-0.018	-0.042	-0.017	-0.002	-0.026	0.047
D	0.013	0.025	-0.004	0.043	-0.015	0.011
E	-0.018	0.011	-0.018	-0.009	-0.038	-0.033
F	0.013	-0.018	0.039	-0.026	0.011	-0.002

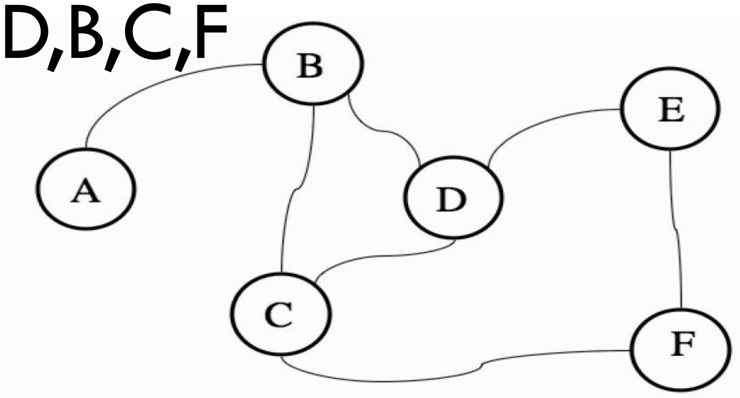
Compute matrix updates as during the iteration. Use

μ (learning rate) $= .6$ and γ (discounting factor) $= .3$

Trail: A,B,A,B,D,D,B,C,F

$\mu = .6$

$\gamma = .3$



For $A \rightarrow B: s = A, a = B, s' = B, r = 0$

$$Q(A,B) = Q(A,B)$$
$$+.6\left(0 + .3 \times \max_{a'} Q(B,a') - Q(A,B)\right)$$
$$= -0.028 + .6(.3 \times 0.044 + 0.028)$$
$$= -0.003$$

For $B \rightarrow A: s = B, a = A, s' = A, r = 0$

States	Action or Next State					
	A	B	C	D	E	F
A	-0.039	-0.028	-0.026	-0.018	0.013	0.011
B	0.028	-0.029	0.024	0.044	0.013	-0.026
C	-0.018	-0.042	-0.017	-0.002	-0.026	0.047
D	0.013	0.025	-0.004	0.043	-0.015	0.011
E	-0.018	0.011	-0.018	-0.009	-0.038	-0.033
F	0.013	-0.018	0.039	-0.026	0.011	-0.002

Updated	A	B	C	D	E	F
A	-0.039	**-0.003**	-0.026	-0.018	0.013	0.011
B	0.028	-0.029	0.024	0.044	0.013	-0.026
C	-0.018	-0.042	-0.017	-0.002	-0.026	0.047
D	0.013	0.025	-0.004	0.043	-0.015	0.011
E	-0.018	0.011	-0.018	-0.009	-0.038	-0.033
F	0.013	-0.018	0.039	-0.026	0.011	-0.002

$$Q(s,a) \leftarrow Q(s,a) + \mu(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Q matrix after iteration # 10

[[-0.0 0.0 -inf -0.0 0.0 0.0]
[-0.0 0.0 38.2 -0.0 0.0 -inf]
[-0.0 -0.0 -0.0 -0.0 0.0 99.8]
[-0.0 0.0 -0.0 -0.0 -0.0 -inf]
[-0.0 -0.0 -0.0 0.0 -0.0 91.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

Q matrix after iteration # 30

[[-0.0 16.0 -inf -0.0 -inf -inf]
[4.3 0.0 40.0 -0.0 0.0 -inf]
[-0.0 -0.0 24.5 4.4 0.0 100.0]
[-0.0 15.9 -0.0 0.8 -0.0 -inf]
[-0.0 -0.0 -0.0 0.0 -0.0 99.8]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

Q matrix after iteration # 100

[[-0.0 16.0 -inf -0.0 -inf -inf]
[5.8 7.7 40.0 -0.0 0.0 -inf]
[-0.0 -0.0 34.1 6.2 0.0 100.0]
[-0.0 16.0 -0.0 0.8 -0.0 -inf]
[-0.0 -0.0 -0.0 0.0 -0.0 100.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

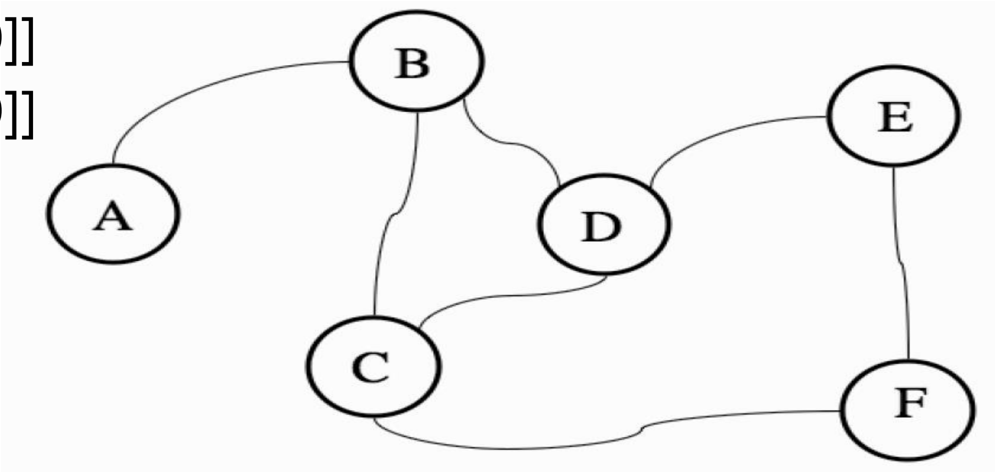
Q matrix after iteration # 500

[[1.4 16.0 -inf -0.0 -inf -inf]
[6.4 11.0 40.0 15.7 0.0 -inf]
[-0.0 16.0 35.0 15.9 0.0 100.0]
[-0.0 16.0 36.4 8.0 40.0 -inf]
[-0.0 -0.0 -0.0 15.6 34.1 100.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

Q matrix after iteration # 1000

[[1.4 16.0 -inf -0.0 -inf -inf]
[6.4 11.0 40.0 16.0 0.0 -inf]
[-0.0 16.0 35.0 16.0 0.0 100.0]
[-0.0 16.0 39.7 10.9 40.0 -inf]
[-0.0 -0.0 -0.0 16.0 35.0 100.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

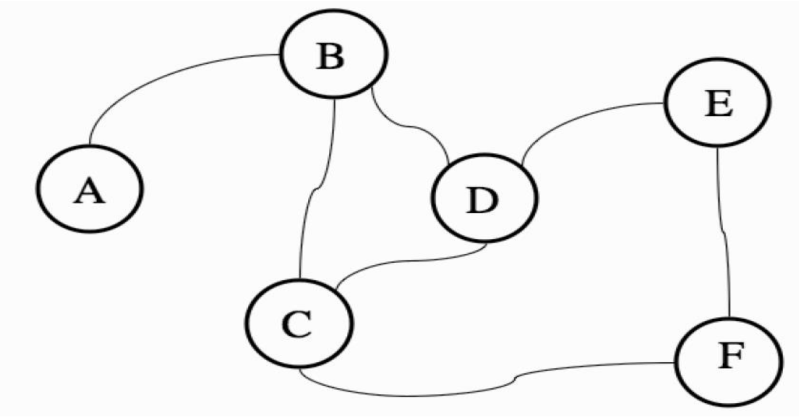
Sample run of 1000 iterations for Q-learning (TD-0) algorithm



Sarsa learning Example

- For the getting lost example, consider the following trail of steps taken by the agent: D,B,B,B,B,C,C,C,F,C

And the following initial Q matrix



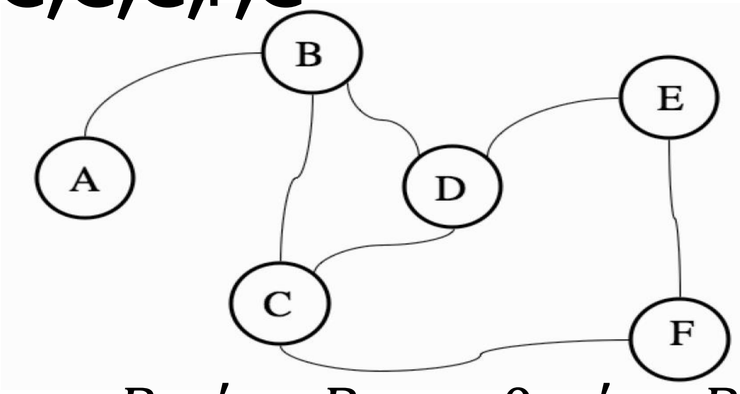
States	Action or Next State					
	A	B	C	D	E	F
A	-0.039	-0.028	-0.018	0.011	-0.026	0.011
B	0.028	-0.029	0.024	0.044	-0.018	-0.026
C	0.011	-0.042	-0.017	-0.002	-0.026	0.047
D	-0.018	0.025	-0.004	0.043	-0.015	-0.018
E	0.011	-0.018	0.011	-0.009	-0.038	-0.033
F	-0.018	-0.026	0.039	-0.026	0.011	-0.002

Compute matrix updates as during the iteration. Use
μ (learning rate) $= .6$ and γ (discounting factor) $= .3$

Trail: D,B,B,B,B,C,C,C,F,C

$\mu = .6$

$\gamma = .3$



For $D \rightarrow B: s = D, a = B, s' = B, r = 0, a' = B$

$$Q(D,B) = Q(D,B)$$
$$+.6(0 + .3 \times Q(B,B) - Q(D,B))$$
$$= 0.025 + .6(.3 \times (-0.029) - 0.025)$$
$$= 0.005$$

For $B \rightarrow B: s = B, a = B, s' = B, r = -5,$
$a' = B$

States	Action or Next State					
	A	B	C	D	E	F
A	-0.039	-0.028	-0.018	0.011	-0.026	0.011
B	0.028	-0.029	0.024	0.044	-0.018	-0.026
C	0.011	-0.042	-0.017	-0.002	-0.026	0.047
D	-0.018	0.025	-0.004	0.043	-0.015	-0.018
E	0.011	-0.018	0.011	-0.009	-0.038	-0.033
F	-0.018	-0.026	0.039	-0.026	0.011	-0.002

Updated	A	B	C	D	E	F
A	-0.039	-0.028	-0.018	0.011	-0.026	0.011
B	0.028	-0.029	0.024	0.044	-0.018	-0.026
C	0.011	-0.042	-0.017	-0.002	-0.026	0.047
D	-0.018	**0.005**	-0.004	0.043	-0.015	-0.018
E	0.011	-0.018	0.011	-0.009	-0.038	-0.033
F	-0.018	-0.026	0.039	-0.026	0.011	-0.002

$$\text{update } Q(s,a) \leftarrow Q(s,a) + \mu(r + \gamma Q(s',a') - Q(s,a))$$

Q matrix after iteration # 10
[[-4.5 0.0 -0.0 -0.0 -0.0 -0.0]
[-0.0 0.0 38.4 0.0 -0.0 -0.0]
[-0.0 -0.0 -0.0 -0.0 0.0 99.9]
[-0.0 -0.0 -0.0 0.0 -0.0 0.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 97.3]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

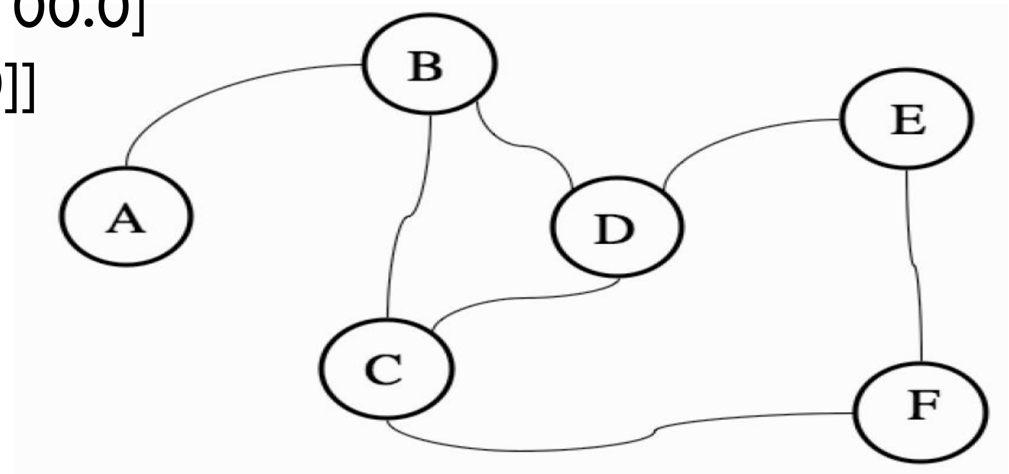
Q matrix after iteration # 30
[[-4.5 4.3 -0.0 -0.0 -0.0 -0.0]
[-0.0 7.7 40.0 0.0 -0.0 -0.0]
[-0.0 -0.0 -0.0 -0.0 0.0 100.0]
[-0.0 -0.0 36.4 -4.5 -0.0 -inf]
[-0.0 -0.0 -0.0 -0.0 -0.0 99.9]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

Q matrix after iteration # 100
[[-4.5 16.0 -0.0 -0.0 -0.0 -0.0]
[-0.0 7.7 40.0 0.0 -0.0 -0.0]
[-0.0 11.2 31.8 6.6 0.0 100.0]
[-0.0 -0.0 40.0 -4.5 -0.0 -inf]
[-0.0 -0.0 -0.0 -0.0 -0.0 100.0]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

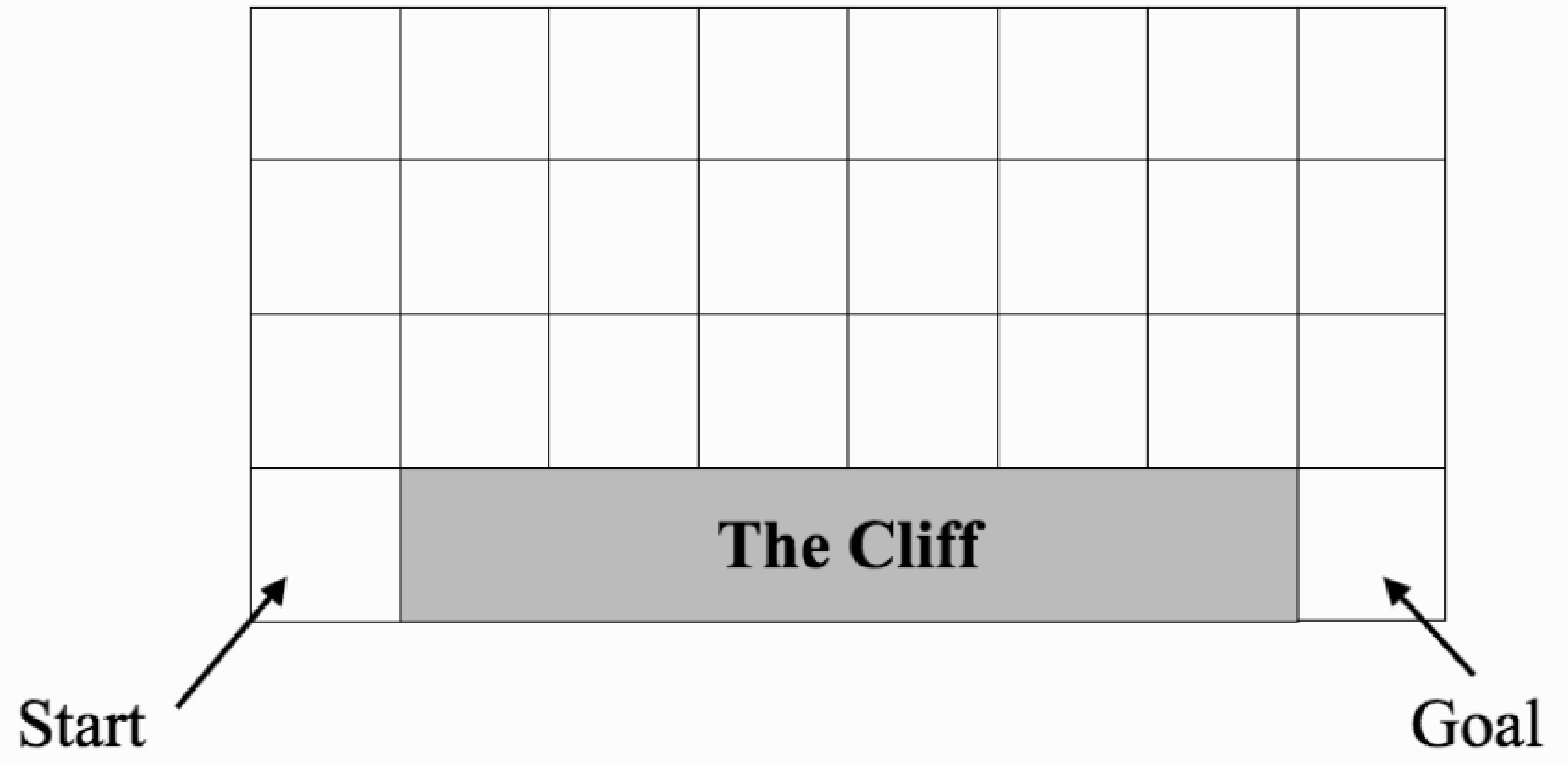
Q matrix after iteration # 500
[[0.3 16.0 -0.0 -0.0 -0.0 -0.0]
[3.0 10.4 40.0 14.0 -0.0 -0.0]
[-0.0 6.6 35.0 12.9 0.0 100.0]
[-0.0 8.7 21.2 -4.5 40.0 -inf]
[-0.0 -0.0 -0.0 15.8 -0.0 100.0]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

Q matrix after iteration # 1000
[[1.4 16.0 -0.0 -0.0 -0.0 -0.0]
[5.4 4.4 40.0 7.4 -0.0 -0.0]
[-0.0 9.1 35.0 11.9 0.0 100.0]
[-0.0 7.3 14.5 10.9 38.2 -inf]
[-0.0 -0.0 -0.0 14.0 32.8
100.0]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

Sample run of 1000 iterations for SARSA algorithm

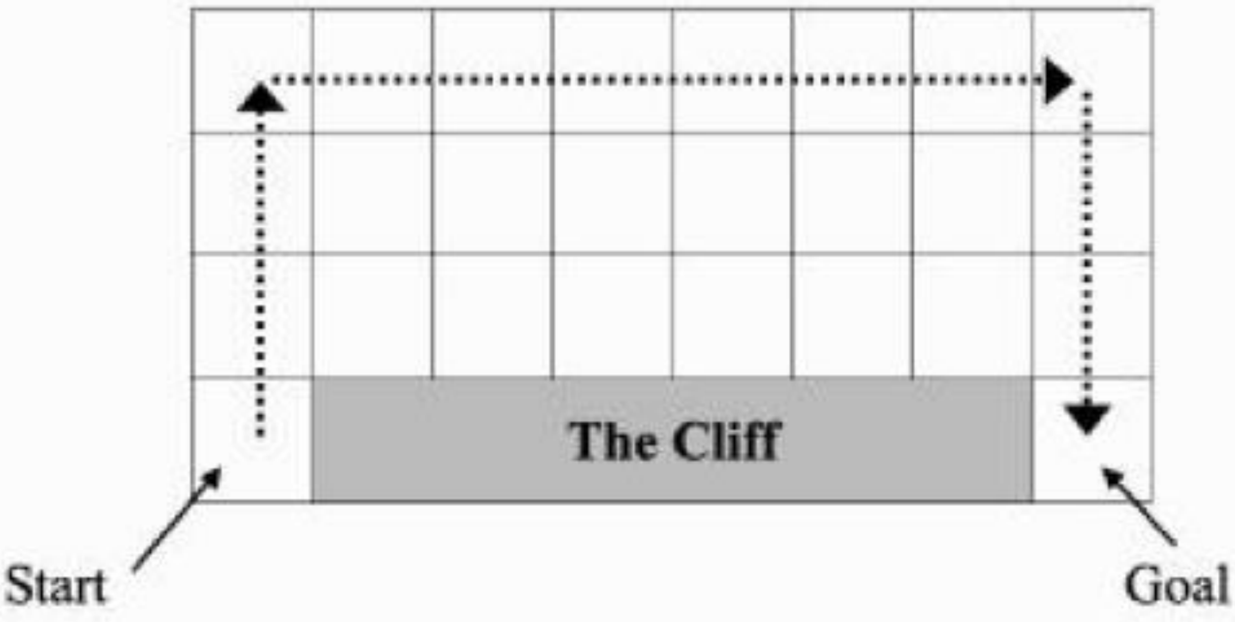


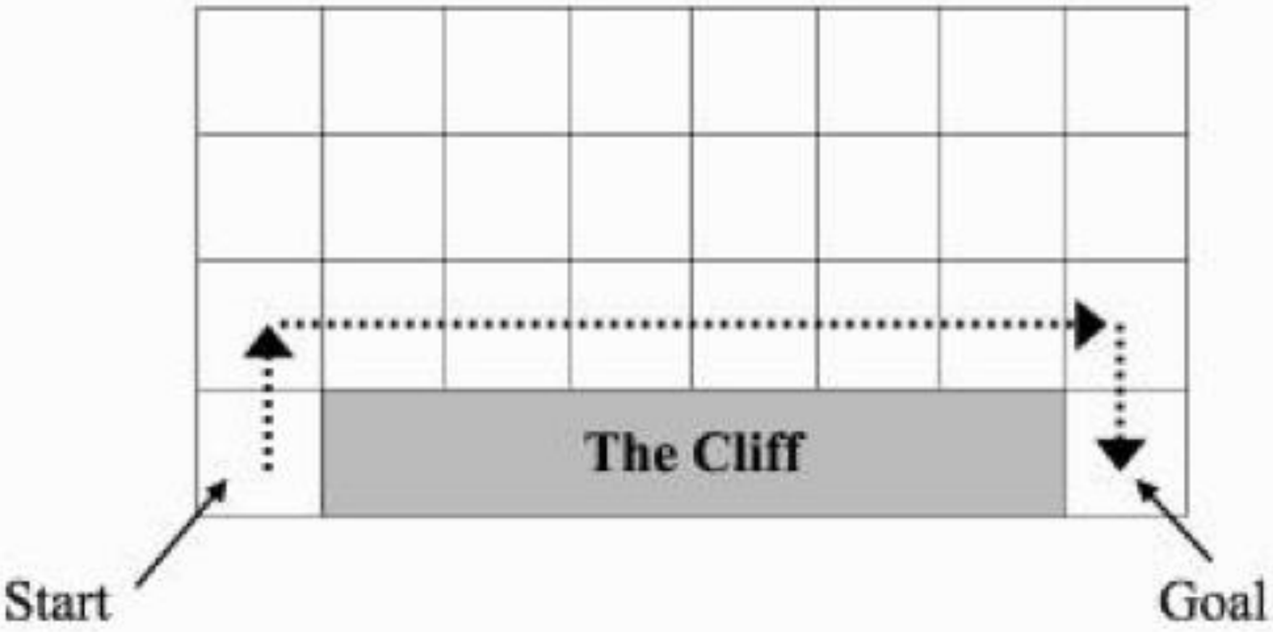
SARSA vs. Q



Every move gets a reward of -1

Moves that end up on cliff get a reward of -100





The SARSA solution is far from optimal, but it is safe.

The Q-learning solution is optimal, but occasionally the random search will tip it over the cliff.

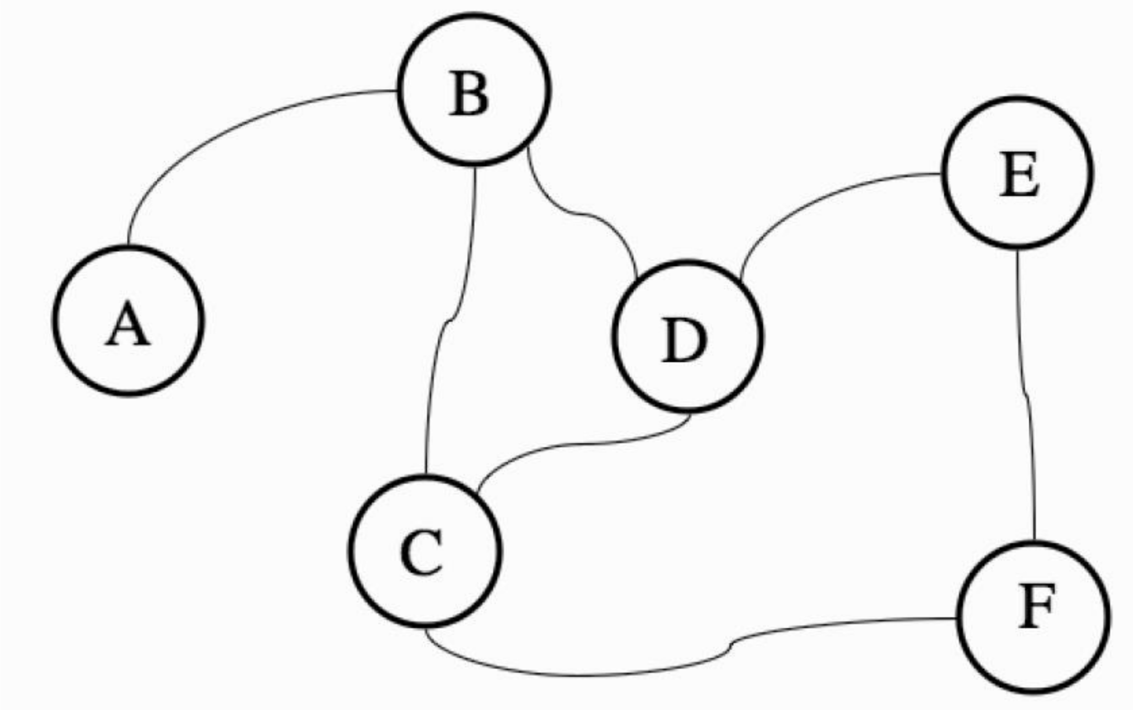
- SARSA vs. Q
  - It is our choice depending upon what we want
  - Depends upon how serious a problem is falling from the cliff.

    The On-Policy uses the same policy to evaluate and improve; however, the off-Policy uses behavioral policy to explore and learn and the target policy to improve.

- Getting lost example… again!

Reward matrix used:

```
[[  -5.0    0.0   -inf   -inf   -inf   -inf]
 [   0.0   -5.0    0.0    0.0   -inf   -inf]
 [  -inf    0.0   -5.0    0.0   -inf  100.0]
 [  -inf    0.0    0.0   -5.0    0.0   -inf]
 [  -inf   -inf   -inf    0.0   -5.0  100.0]
 [  -inf   -inf    0.0   -inf   -inf    0.0]]
```



Q matrix after iteration # 10

[[-0.0 0.0 -inf -0.0 0.0 0.0]
[-0.0 0.0 38.2 -0.0 0.0 -inf]
[-0.0 -0.0 -0.0 -0.0 0.0 99.8]
[-0.0 0.0 -0.0 -0.0 -0.0 -inf]
[-0.0 -0.0 -0.0 0.0 -0.0 91.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

Q matrix after iteration # 30

[[-0.0 16.0 -inf -0.0 -inf -inf]
[4.3 0.0 40.0 -0.0 0.0 -inf]
[-0.0 -0.0 24.5 4.4 0.0 100.0]
[-0.0 15.9 -0.0 0.8 -0.0 -inf]
[-0.0 -0.0 -0.0 0.0 -0.0 99.8]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

Q matrix after iteration # 100

[[-0.0 16.0 -inf -0.0 -inf -inf]
[5.8 7.7 40.0 -0.0 0.0 -inf]
[-0.0 -0.0 34.1 6.2 0.0 100.0]
[-0.0 16.0 -0.0 0.8 -0.0 -inf]
[-0.0 -0.0 -0.0 0.0 -0.0 100.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

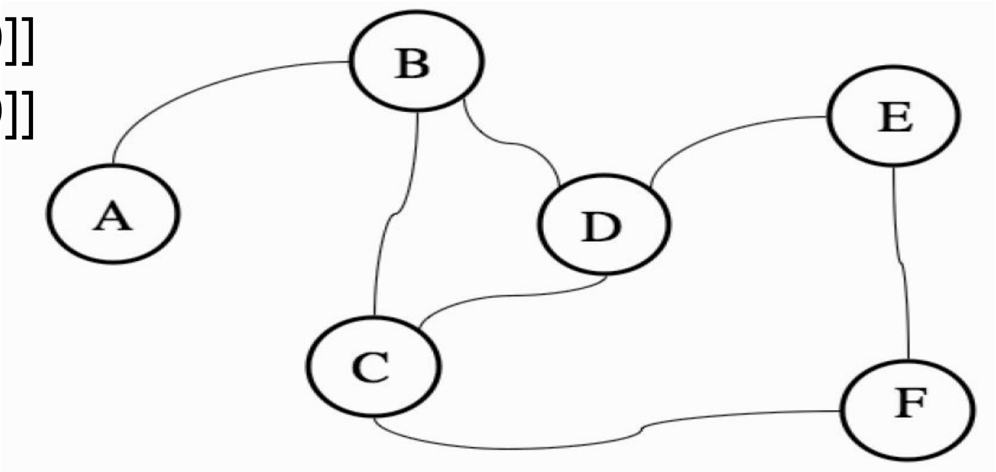
Q matrix after iteration # 500

[[1.4 16.0 -inf -0.0 -inf -inf]
[6.4 11.0 40.0 15.7 0.0 -inf]
[-0.0 16.0 35.0 15.9 0.0 100.0]
[-0.0 16.0 36.4 8.0 40.0 -inf]
[-0.0 -0.0 -0.0 15.6 34.1 100.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

Q matrix after iteration # 1000

[[1.4 16.0 -inf -0.0 -inf -inf]
[6.4 11.0 40.0 16.0 0.0 -inf]
[-0.0 16.0 35.0 16.0 0.0 100.0]
[-0.0 16.0 39.7 10.9 40.0 -inf]
[-0.0 -0.0 -0.0 16.0 35.0 100.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 0.0]]

Q-learning (TD-0) algorithm – note the number of -inf



Q matrix after iteration # 10

[[-4.5 0.0 -0.0 -0.0 -0.0 -0.0]
[-0.0 0.0 38.4 0.0 -0.0 -0.0]
[-0.0 -0.0 -0.0 -0.0 0.0 99.9]
[-0.0 -0.0 -0.0 0.0 -0.0 0.0]
[-0.0 -0.0 -0.0 -0.0 -0.0 97.3]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

Q matrix after iteration # 30

[[-4.5 4.3 -0.0 -0.0 -0.0 -0.0]
[-0.0 7.7 40.0 0.0 -0.0 -0.0]
[-0.0 -0.0 -0.0 -0.0 0.0 100.0]
[-0.0 -0.0 36.4 -4.5 -0.0 -inf]
[-0.0 -0.0 -0.0 -0.0 -0.0 99.9]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

Q matrix after iteration # 100

[[-4.5 16.0 -0.0 -0.0 -0.0 -0.0]
[-0.0 7.7 40.0 0.0 -0.0 -0.0]
[-0.0 11.2 31.8 6.6 0.0 100.0]
[-0.0 -0.0 40.0 -4.5 -0.0 -inf]
[-0.0 -0.0 -0.0 -0.0 -0.0 100.0]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

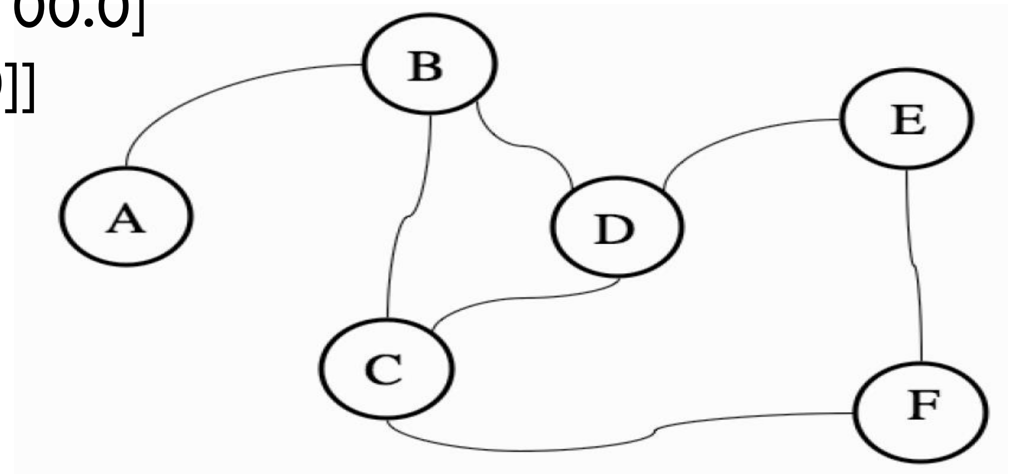
Q matrix after iteration # 500

[[0.3 16.0 -0.0 -0.0 -0.0 -0.0]
[3.0 10.4 40.0 14.0 -0.0 -0.0]
[-0.0 6.6 35.0 12.9 0.0 100.0]
[-0.0 8.7 21.2 -4.5 40.0 -inf]
[-0.0 -0.0 -0.0 15.8 -0.0 100.0]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

Q matrix after iteration # 1000

[[1.4 16.0 -0.0 -0.0 -0.0 -0.0]
[5.4 4.4 40.0 7.4 -0.0 -0.0]
[-0.0 9.1 35.0 11.9 0.0 100.0]
[-0.0 7.3 14.5 10.9 38.2 -inf]
[-0.0 -0.0 -0.0 14.0 32.8
100.0]
[-0.0 -0.0 0.0 -0.0 0.0 0.0]]

SARSA algorithm – note the number of -inf



USES OF REINFORCEMENT LEARNING

- Most popular is in intelligent robotics
  - The robot can be left to attempt to solve the task without human intervention.
- Also popular at learning how to play games

Modelling Sample Problems

- To model sample problems, we need to model
  - Q matrix (state-action matrix)
  - Reward

- Remaining things like choice of policy, choice of algorithm, etc., are simply choices, whose options are well established.
  - They don't need to be modelled.

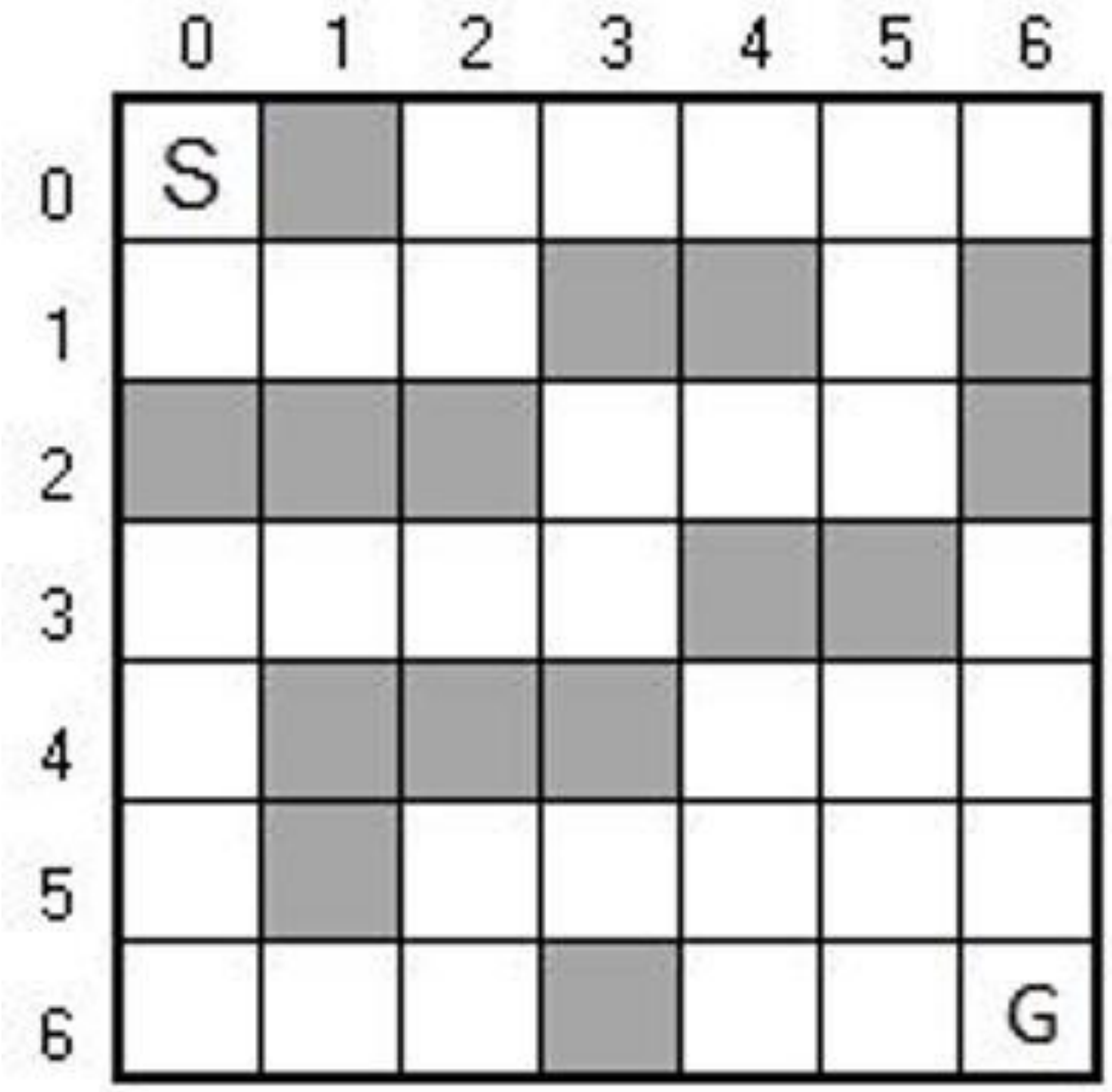
Wading Through Maze

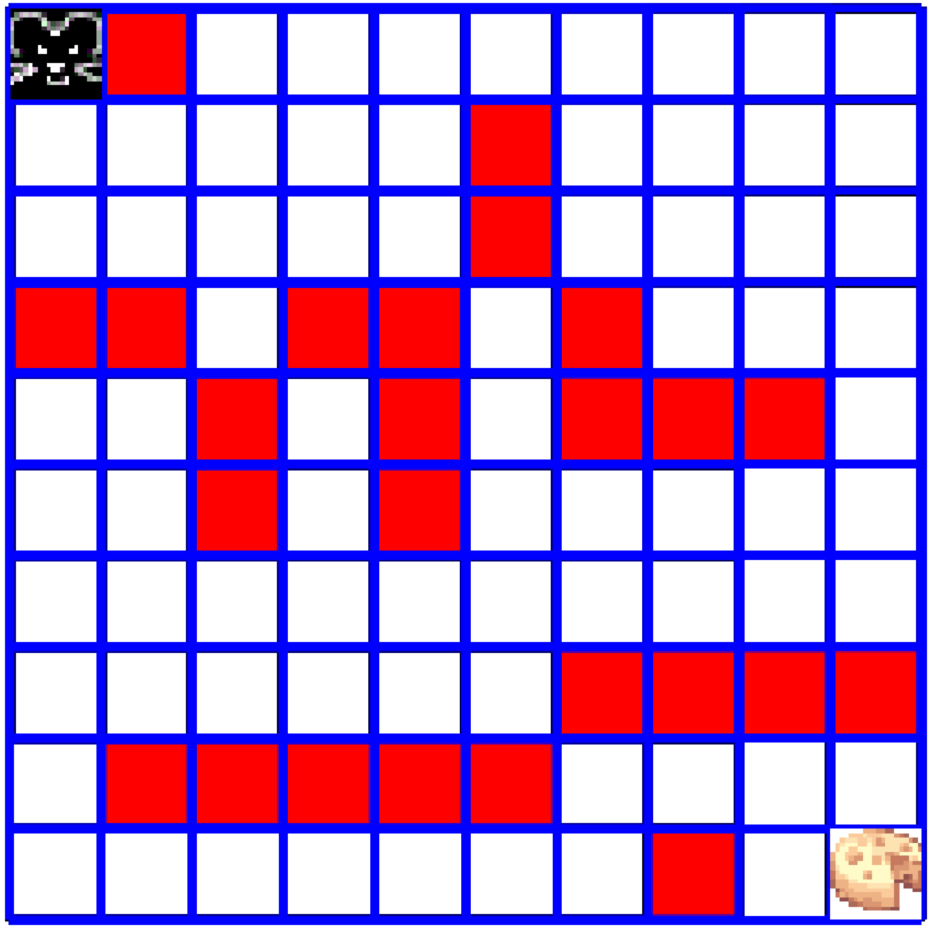
- Taken from

https://towardsdatascience.com/the-other-type-of-machine-learning-97ab81306ce9

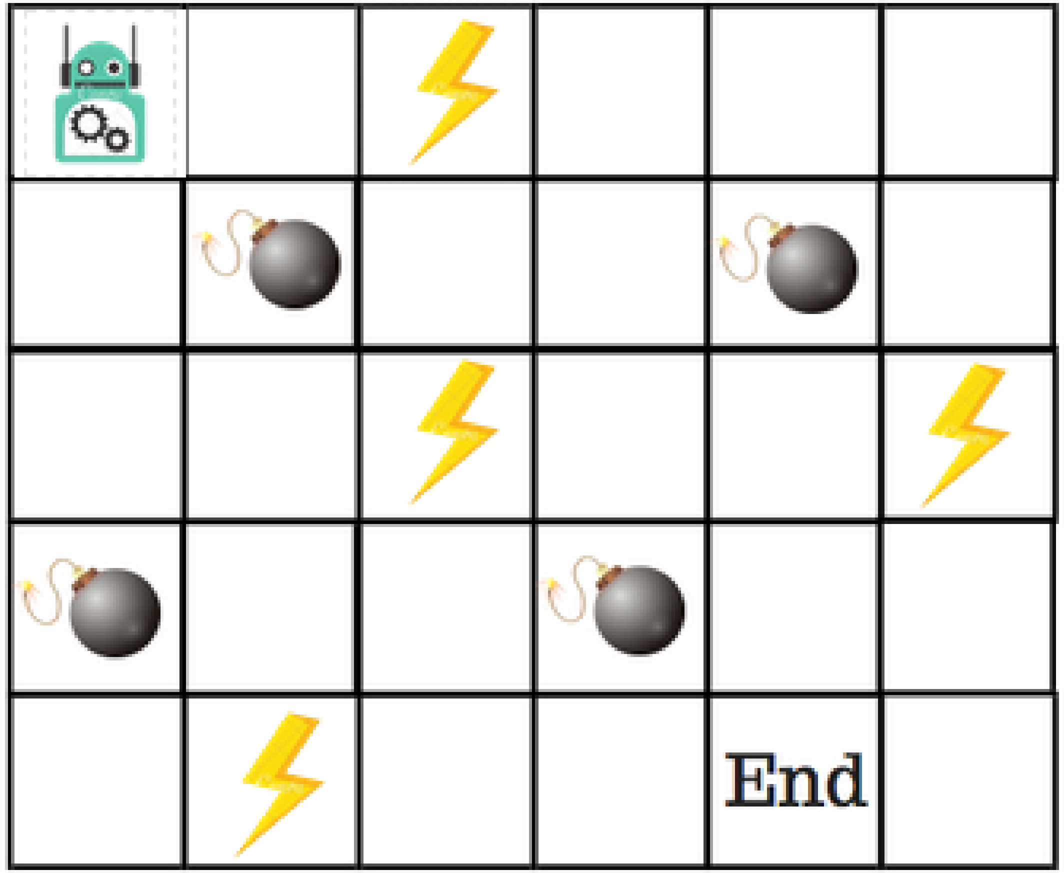
Also see:

https://samyzaf.com/ML/rl/qmaze.html





Wading Through Maze - II



Actions :	↑	→	↓	←
Start				
Nothing / Blank				
Power				
Mines				
END				

Q-Table

Credit: https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

Tic-Tac-Toe