# Playing Atari with Deep Reinforcement Learning

Jonathan Chung

# Objectives

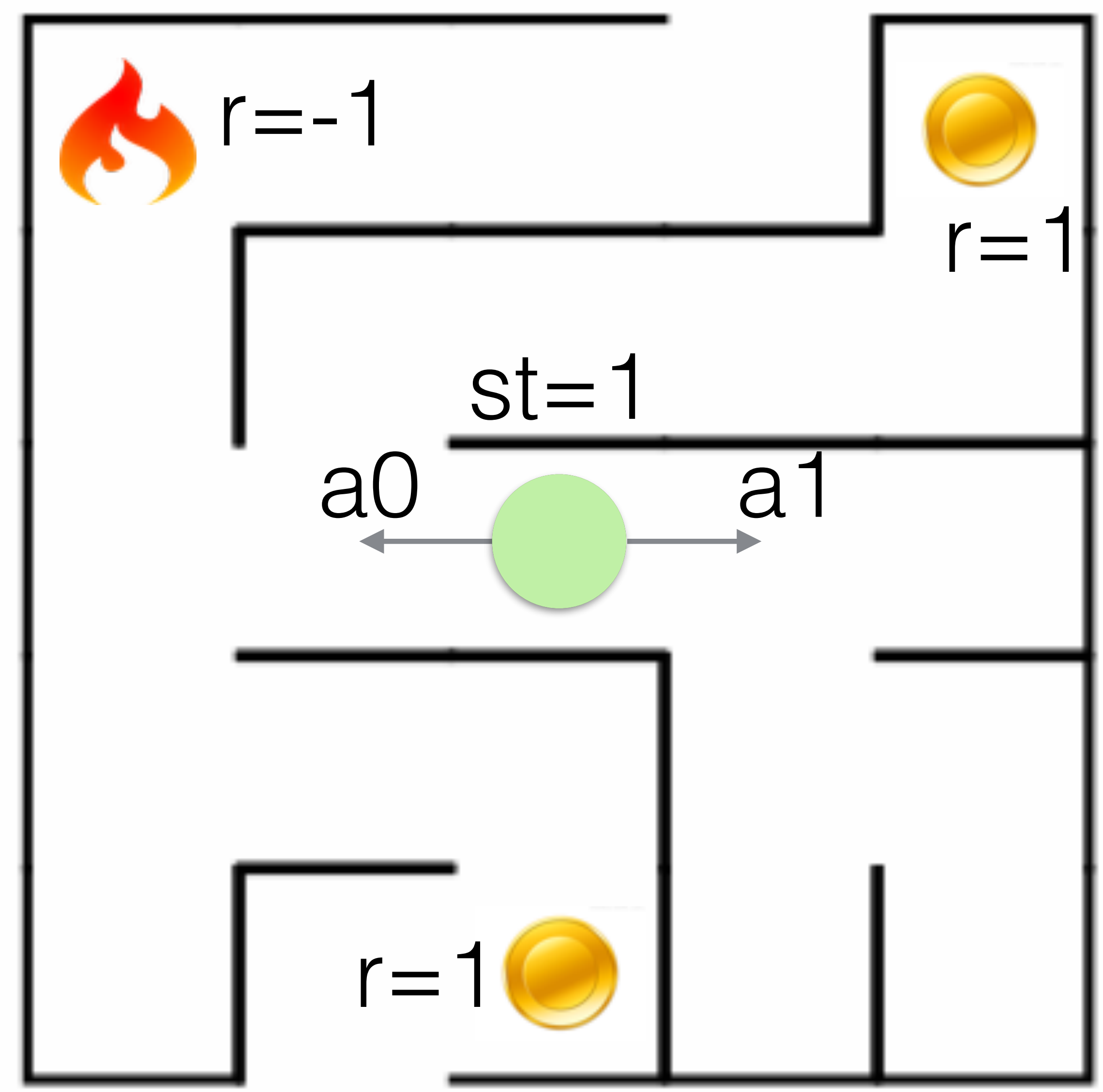
- Provide some basic understanding of RL
- Apply this understanding to the paper
- Discuss possible future directions of the paper

# Reinforcement Learning

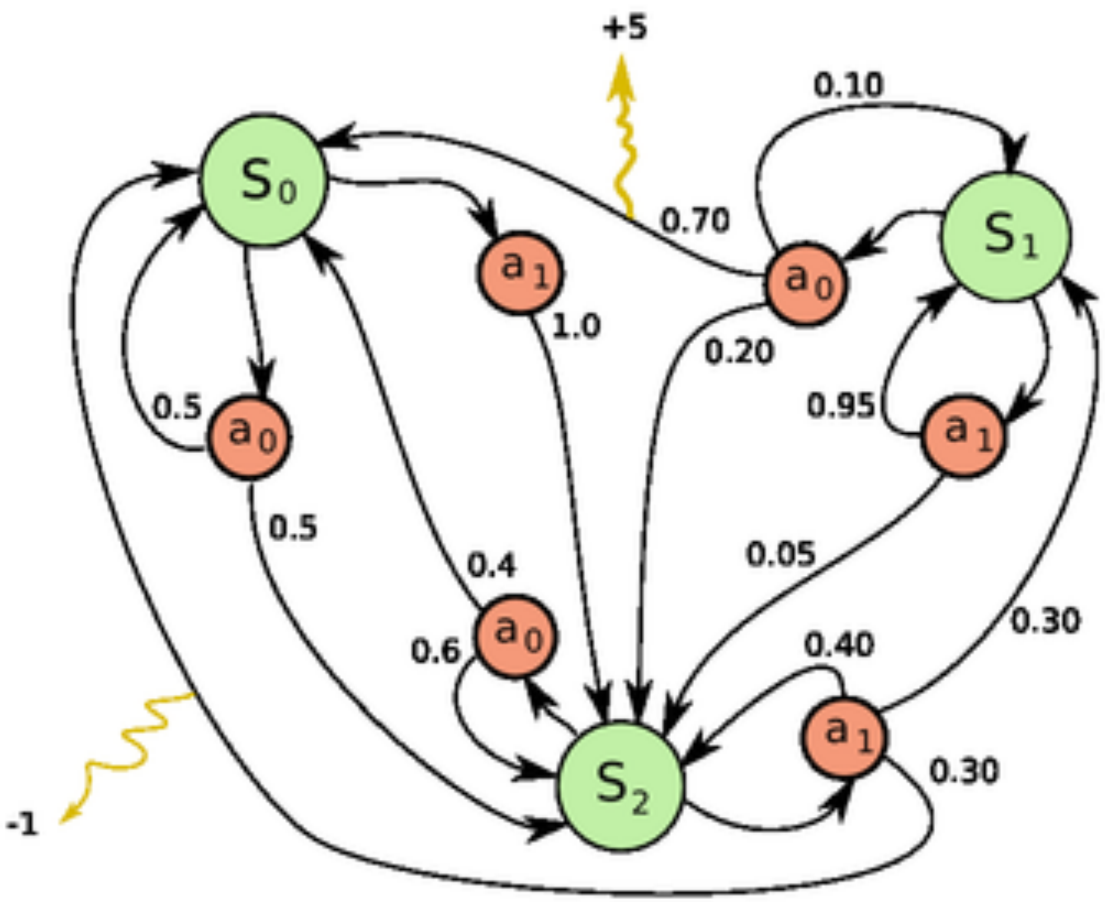
"Finding suitable actions in order to maximise the reward"

# Problem definition

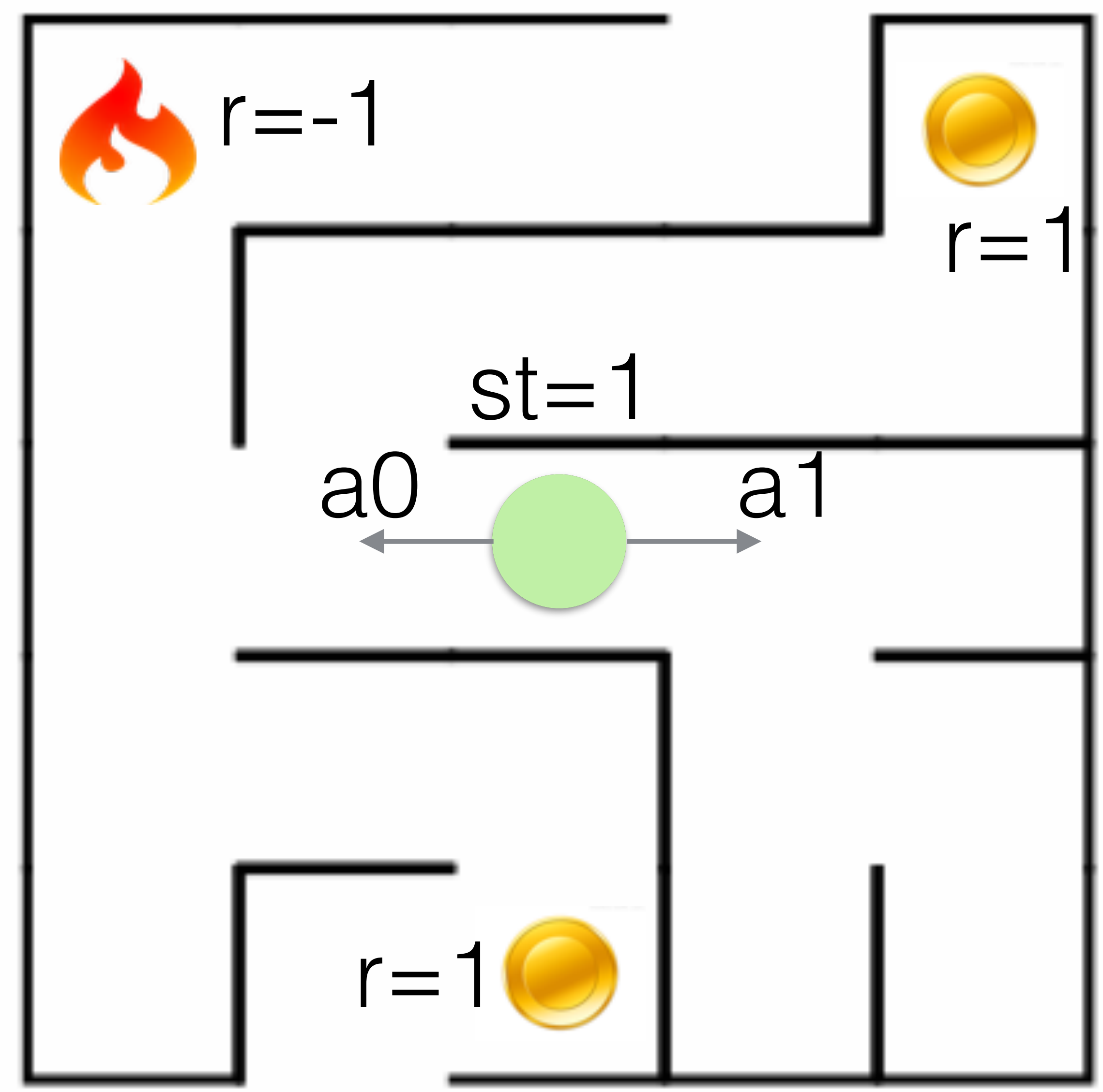
- States (st)
- Action (a)
- Reward (r)



# Problem definition




Markov decision process



# Problem definition

- States (st)

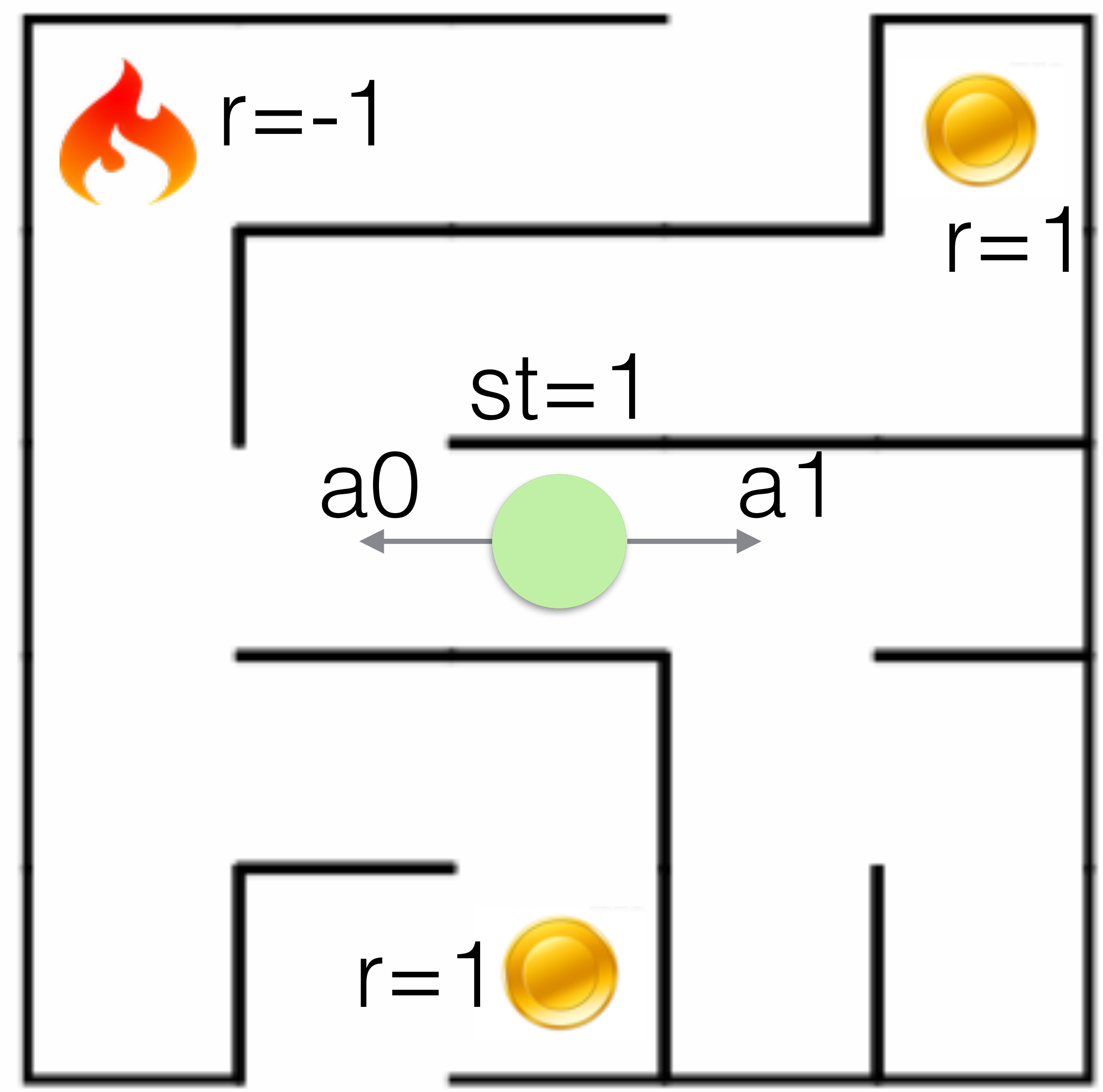
  Where you are (and where you have been)

- Action (a)

  What can you do

- Reward (r)

  What can you get




Rewards.are.Not.
Always.immediate

# Problem definition

- States (st)

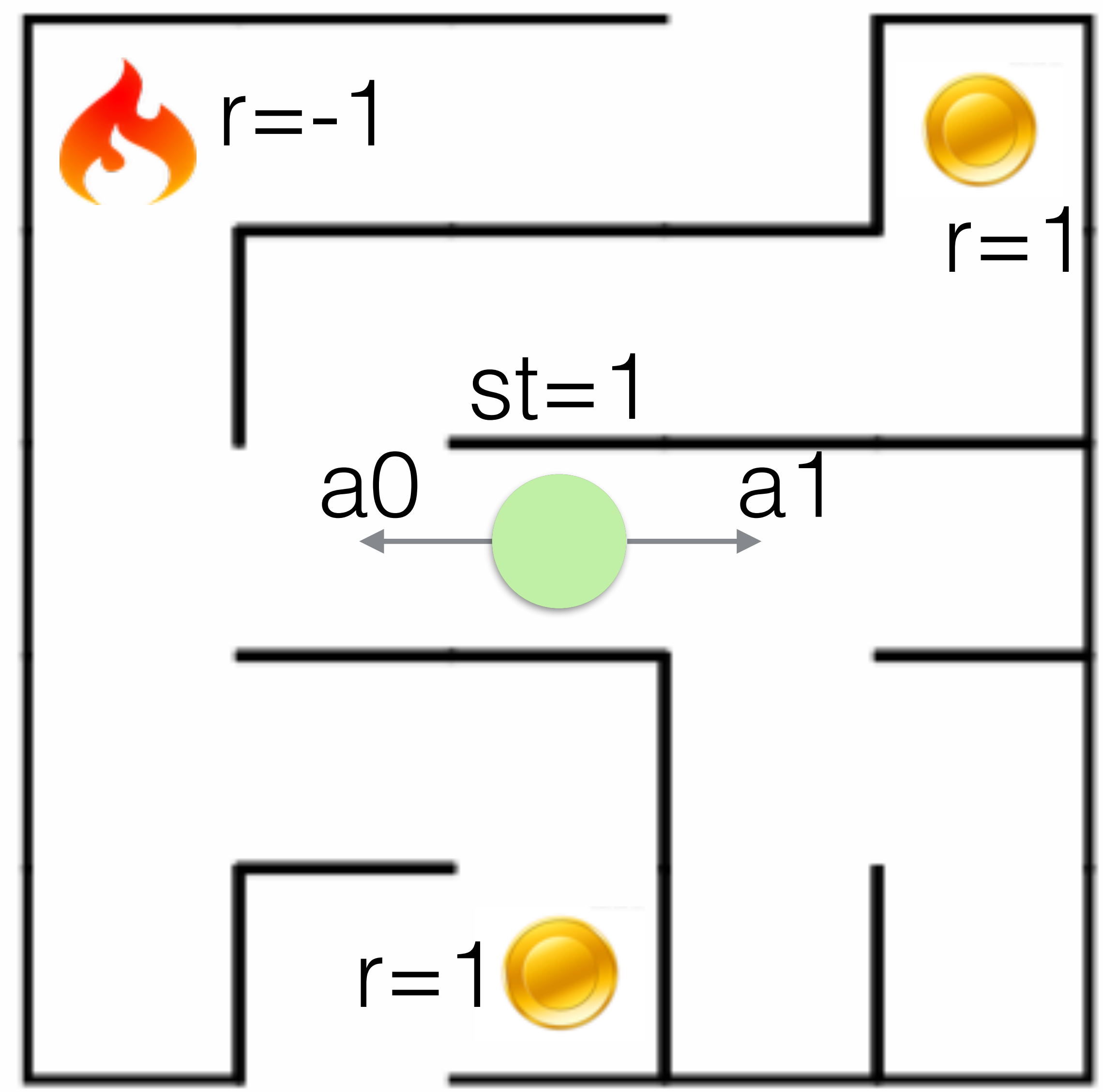
  Where you are (and where you have been)

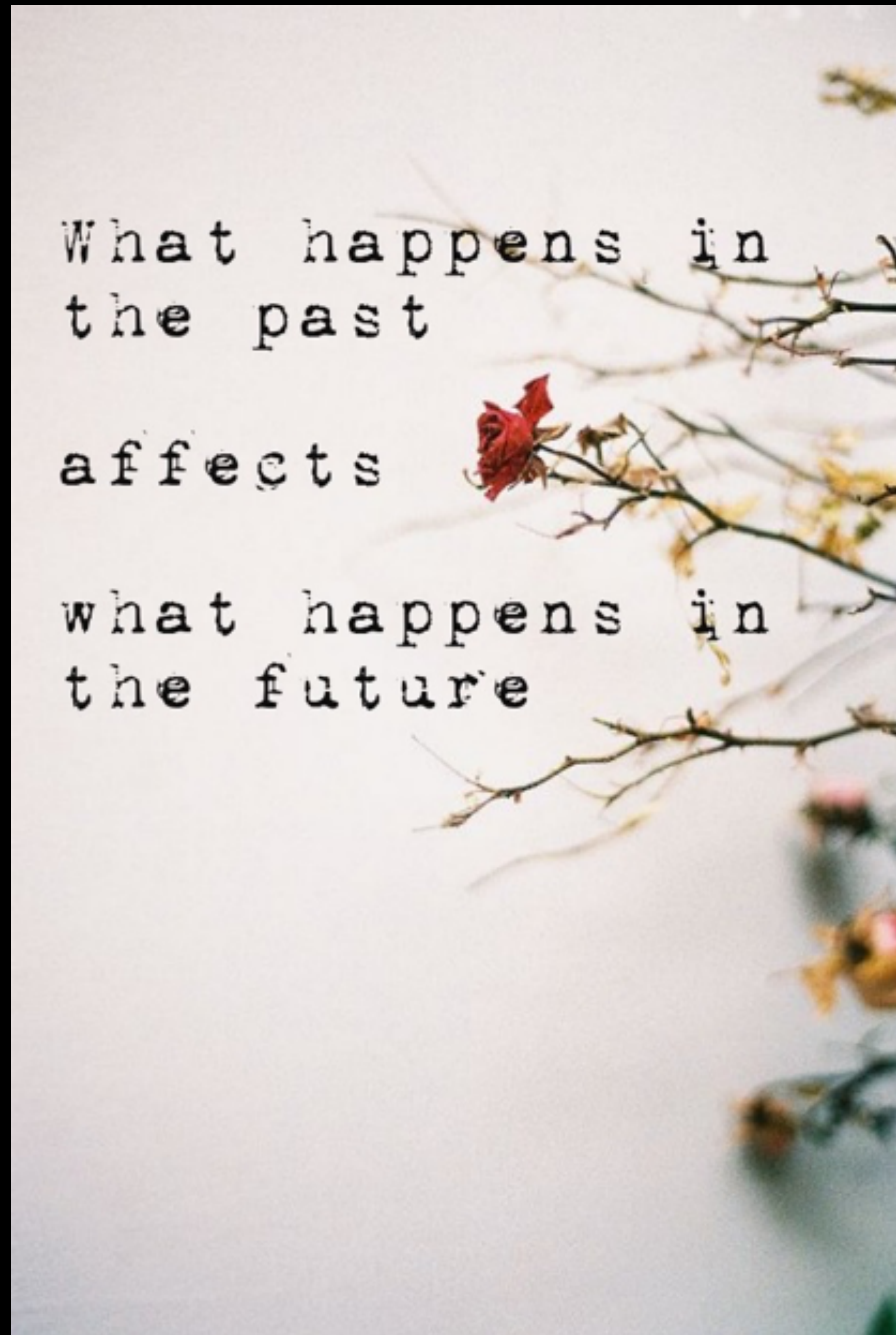
- Action (a)

  What can you do

- Reward (r)

  What can you get




What happens in the past
affects
what happens in the future

# Problem definition

- States (st)

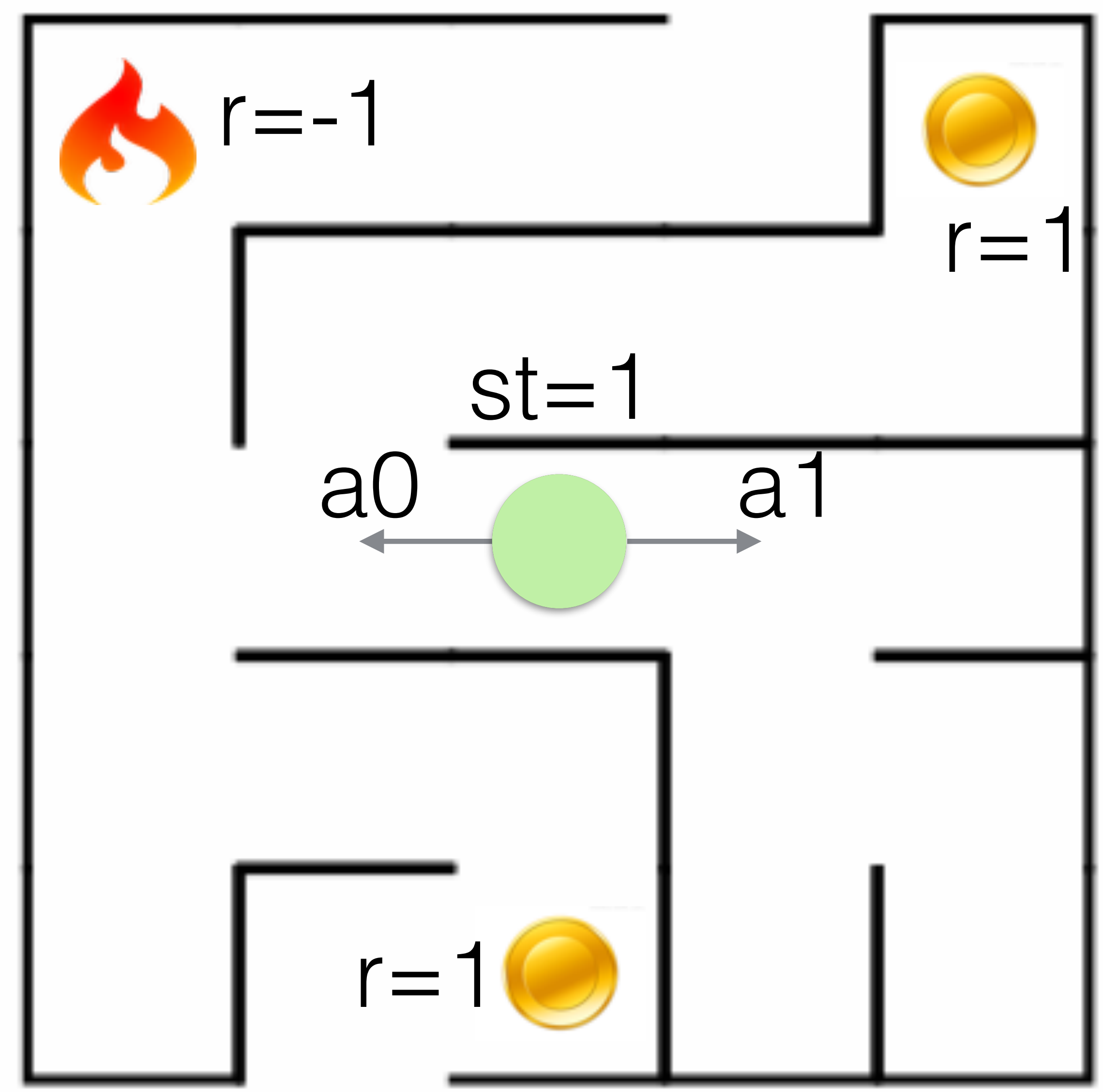
  Where you are (and where you have been)

- Action (a)

  What can you do

- Reward (r)

  What can you get




Intelligence is the
ability to adapt to
change.

# Problem definition

- States (st)

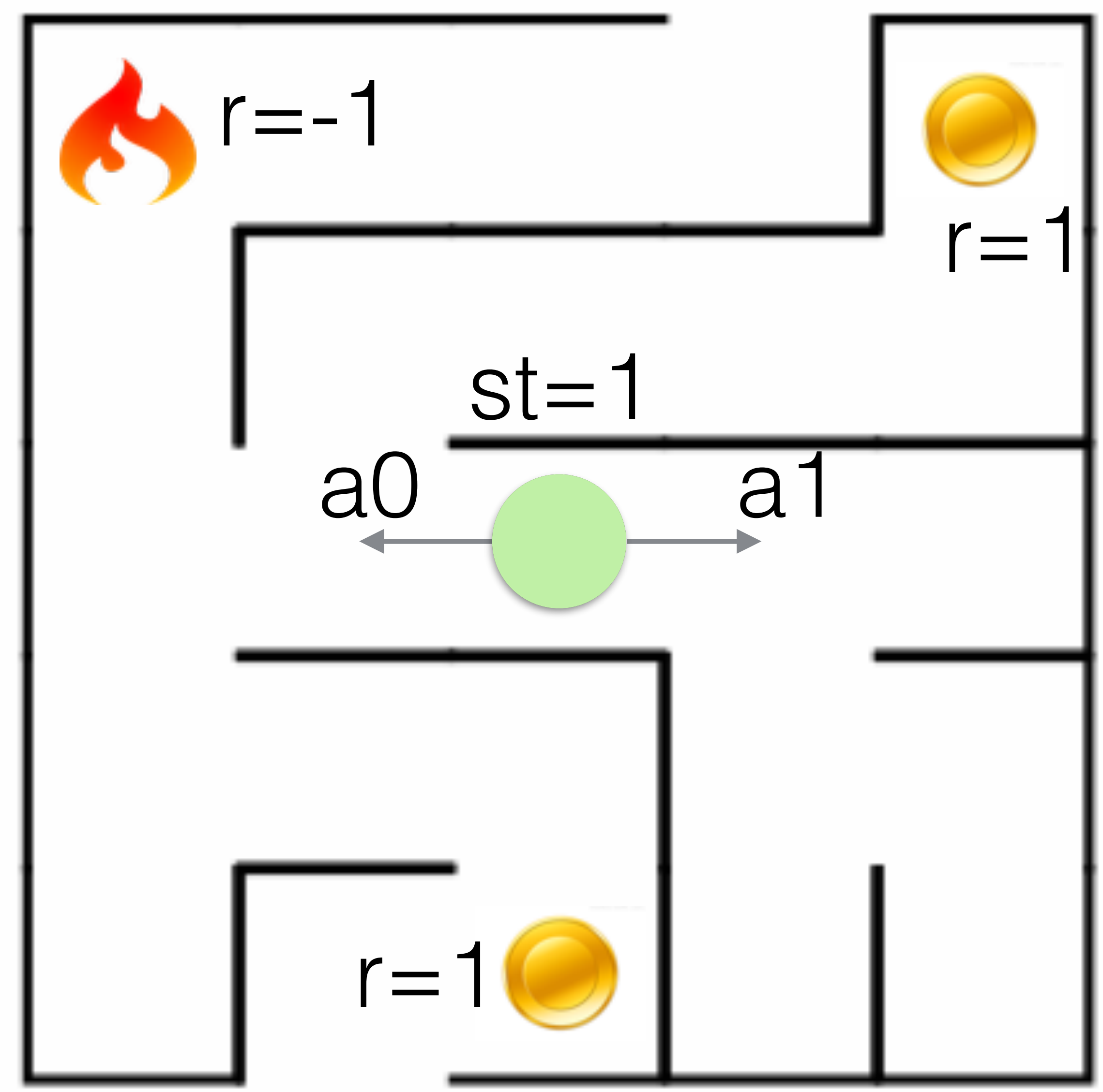
  Where you are (and where you have been)

- Action (a)

  What can you do

- Reward (r)

  What can you get



# Aim

$$Q^*(s, a) \; = \; \max_\pi \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]$$

- **Q** is defined as the maximum expected reward (**R**$_t$) after sequence **s** and taking action **a**


# Playing Atari with Deep Reinforcement Learning

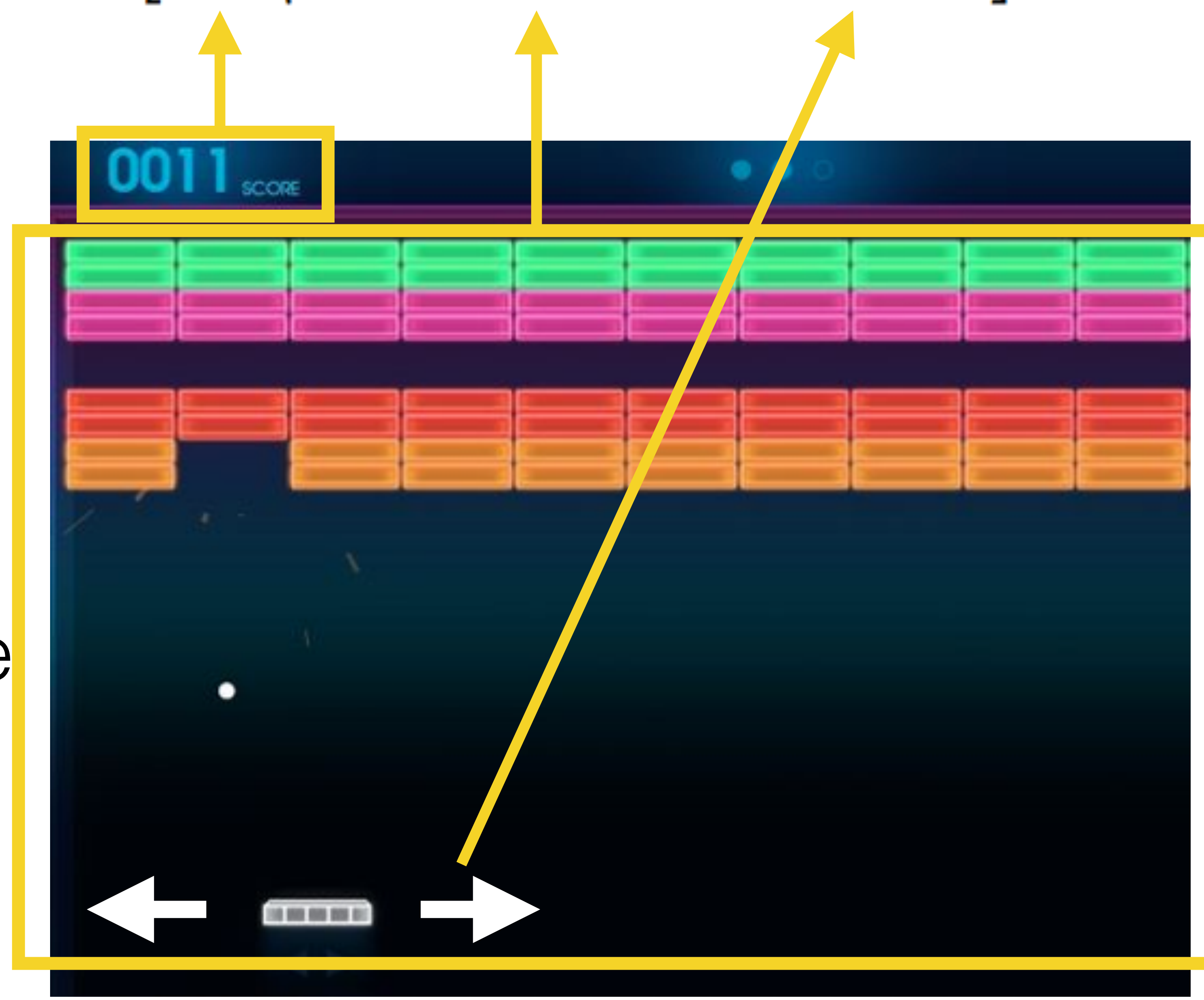
# Inputs

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]$$

- Images ~ s#
- Actions = a^
- Score ~ Reward*

#A sequence of images are used to represent the sequence s.

^Depends on the game

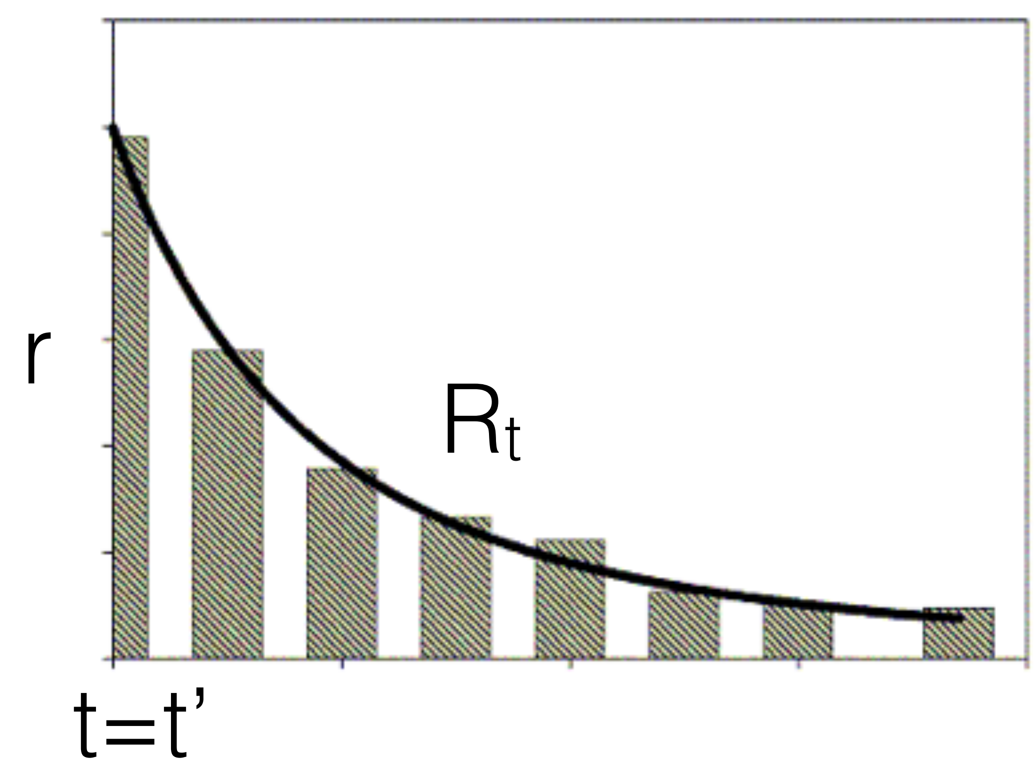


# Inputs

$$Q^*(s,a) = \max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

- Images ~ s#
- Actions = a^
- Score ~ Reward*

*All future rewards are considered but discounted based on the time



# Learning

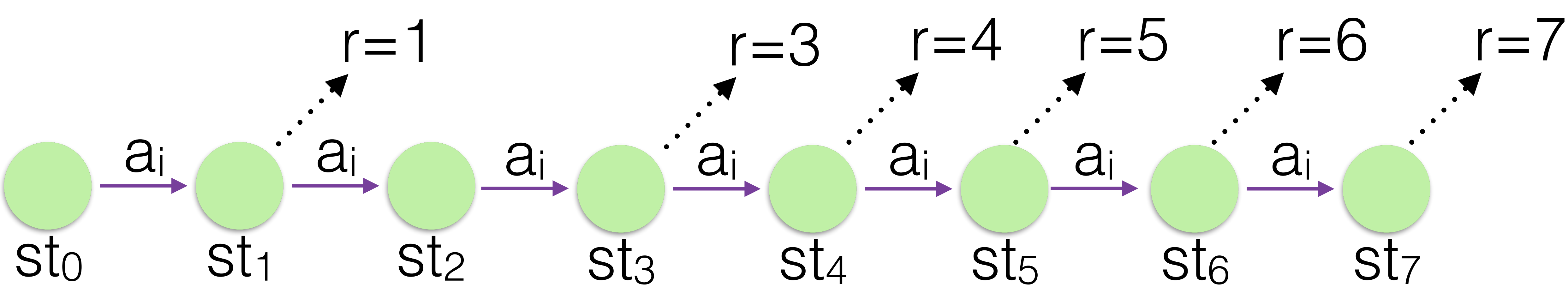
- Bellman equation

$$Q\ (s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q\ (s',a') \middle| s,a\right]$$

# Learning

- Bellman equation

$$Q\ (s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q\ (s',a') \middle| s,a\right]$$



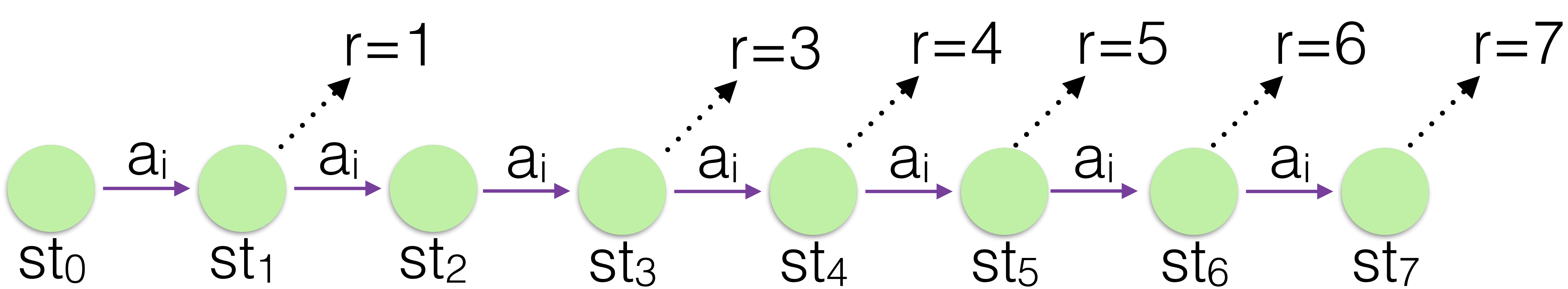
# Learning

- Bellman equation

$$Q\ (s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q\ (s',a') \middle| s,a \right]$$

- s - sequence, a series of states

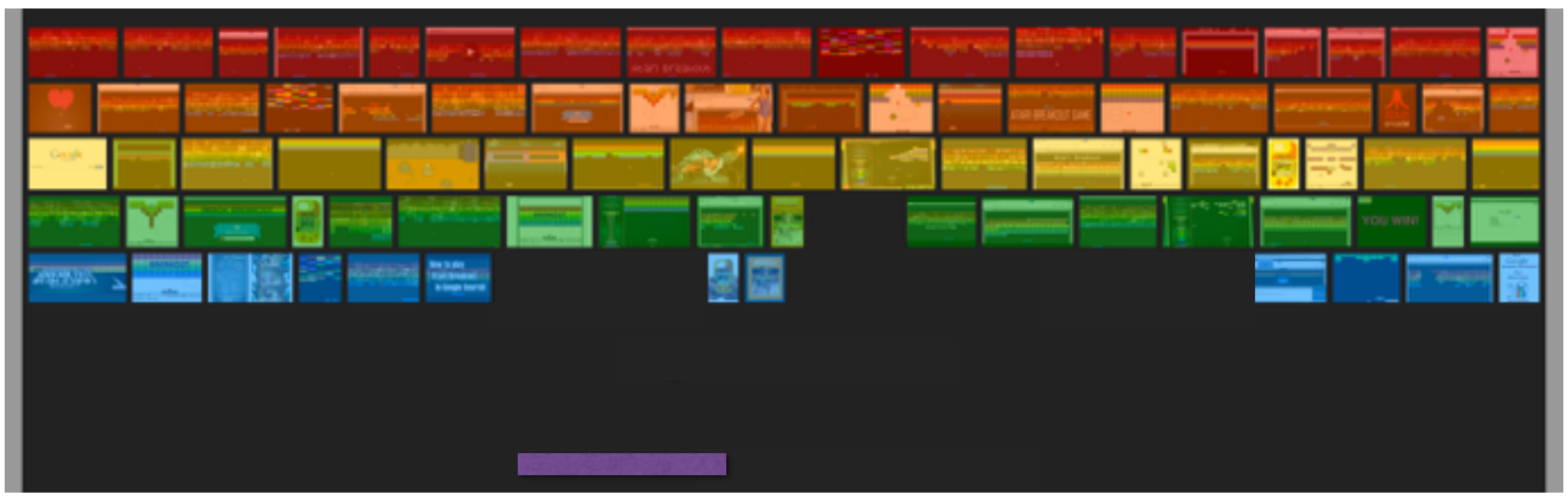
- a - action

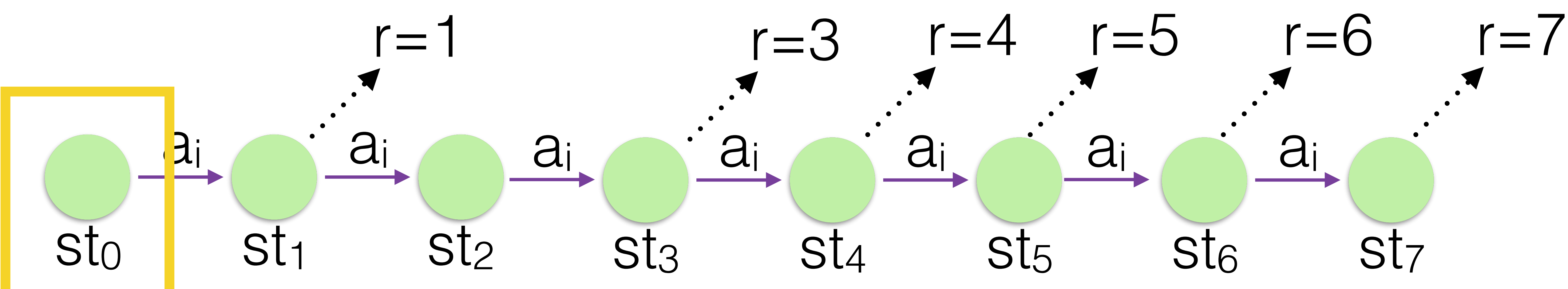


# Learning

- Bellman equation

$$Q(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a') \middle| s,a\right]$$





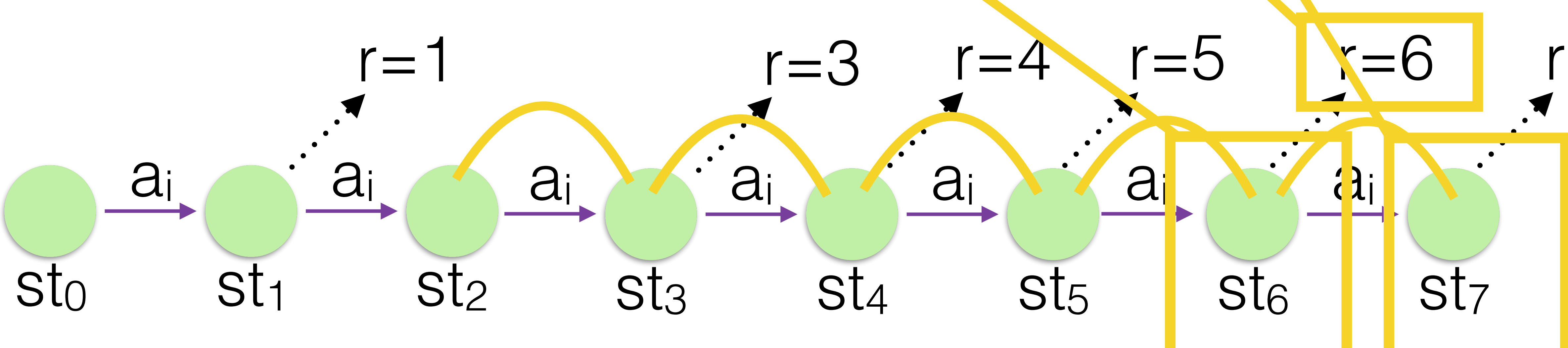
# Learning

- Bellman equation

$$Q(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[ r + \gamma \max_{a'} Q(s',a') \middle| s,a \right]$$

Discount factor

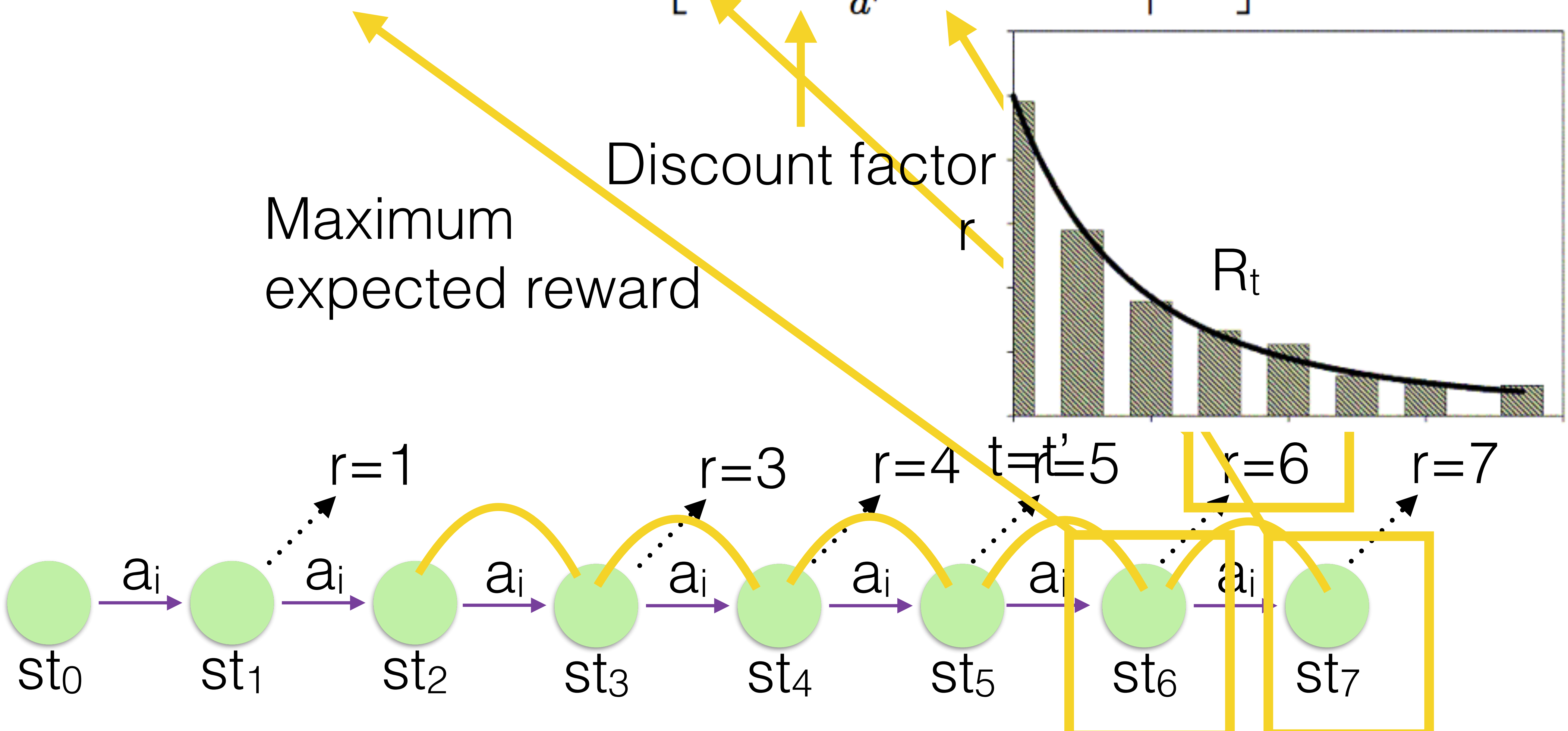
Maximum
expected reward



# Learning

- Bellman equation

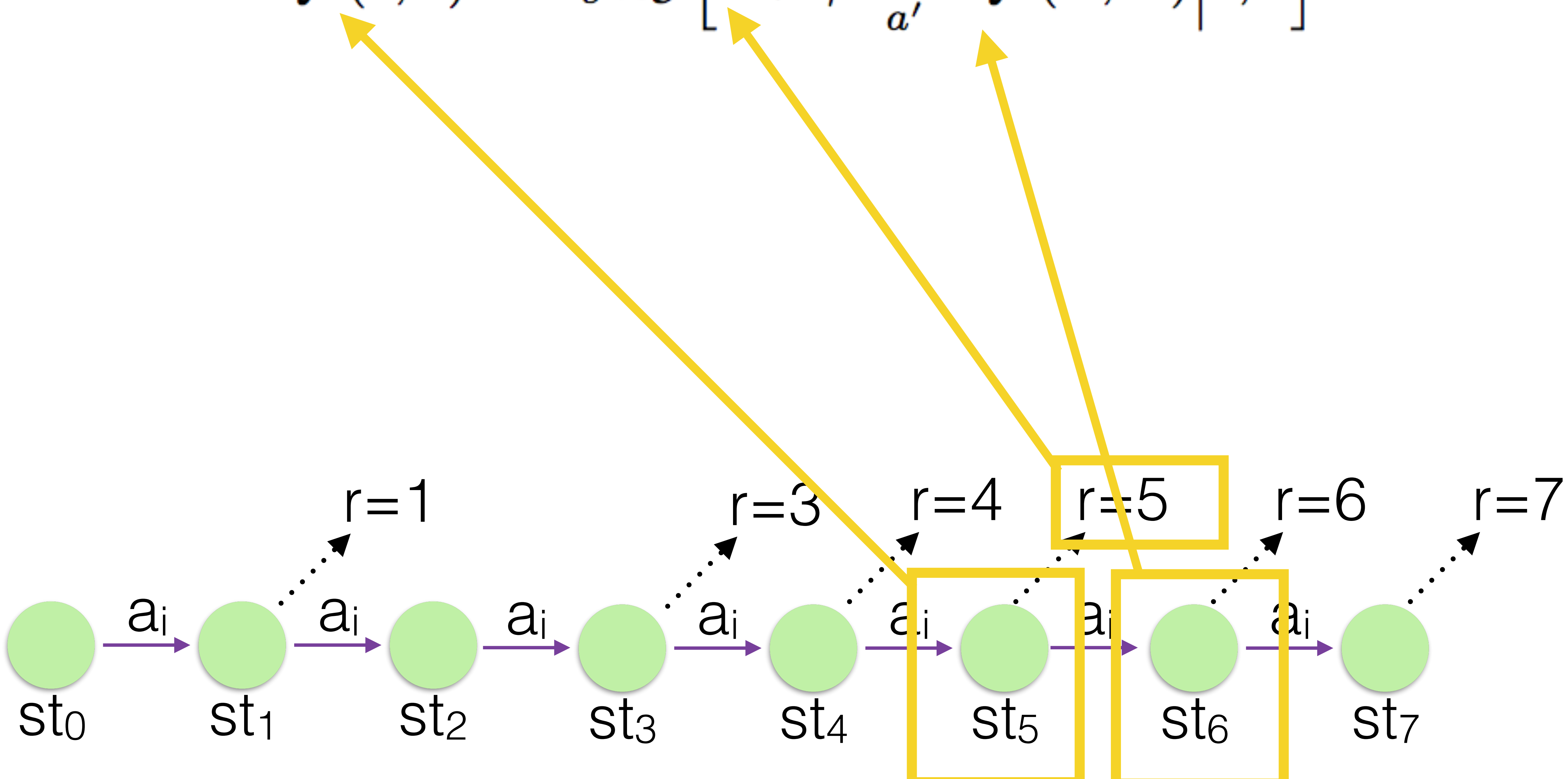
$$Q(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a') \middle| s,a\right]$$



# Learning

- Bellman equation

$$Q\ (s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[ r + \gamma \max_{a'} Q\ (s',a') \middle| s,a \right]$$

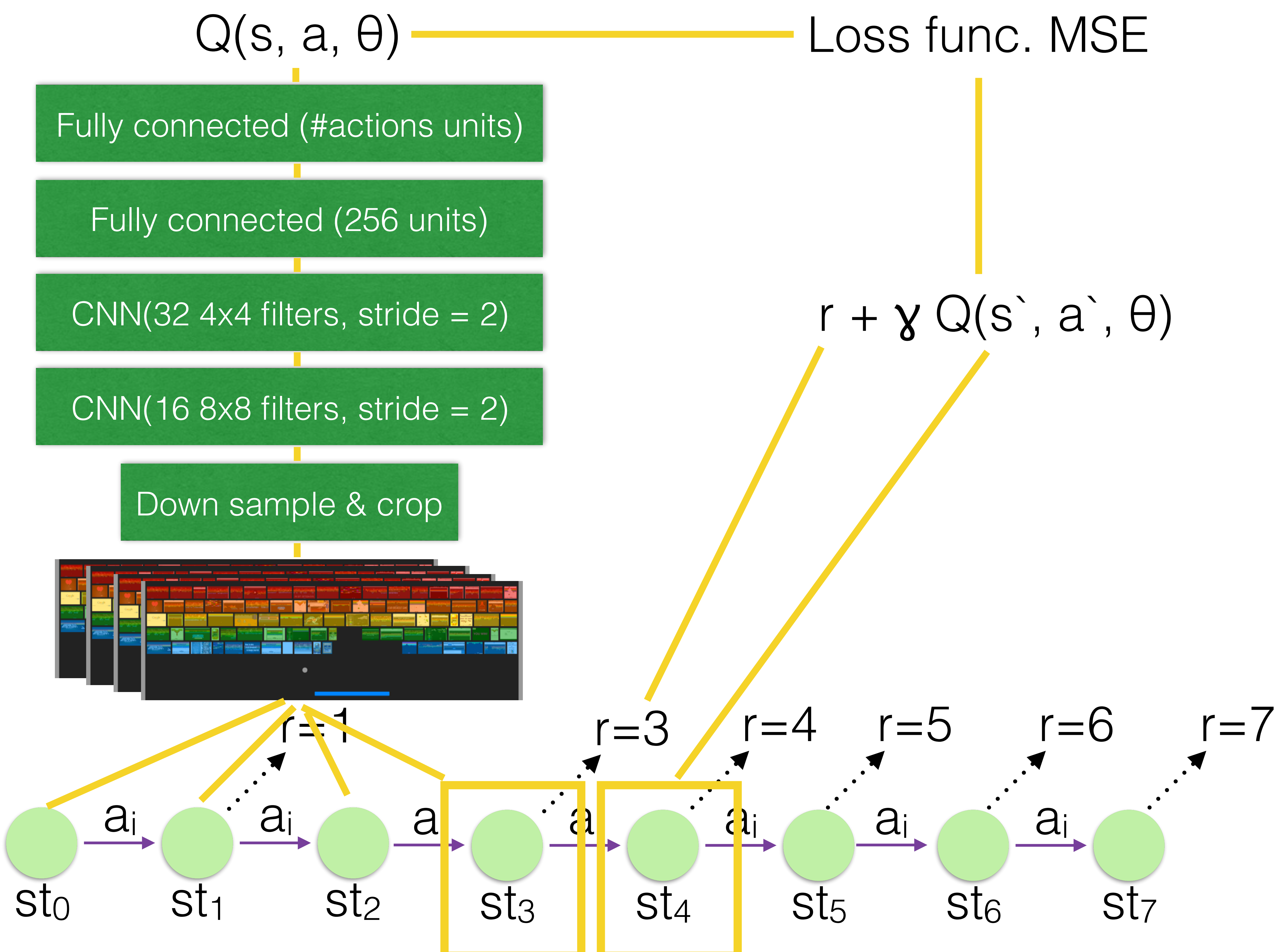


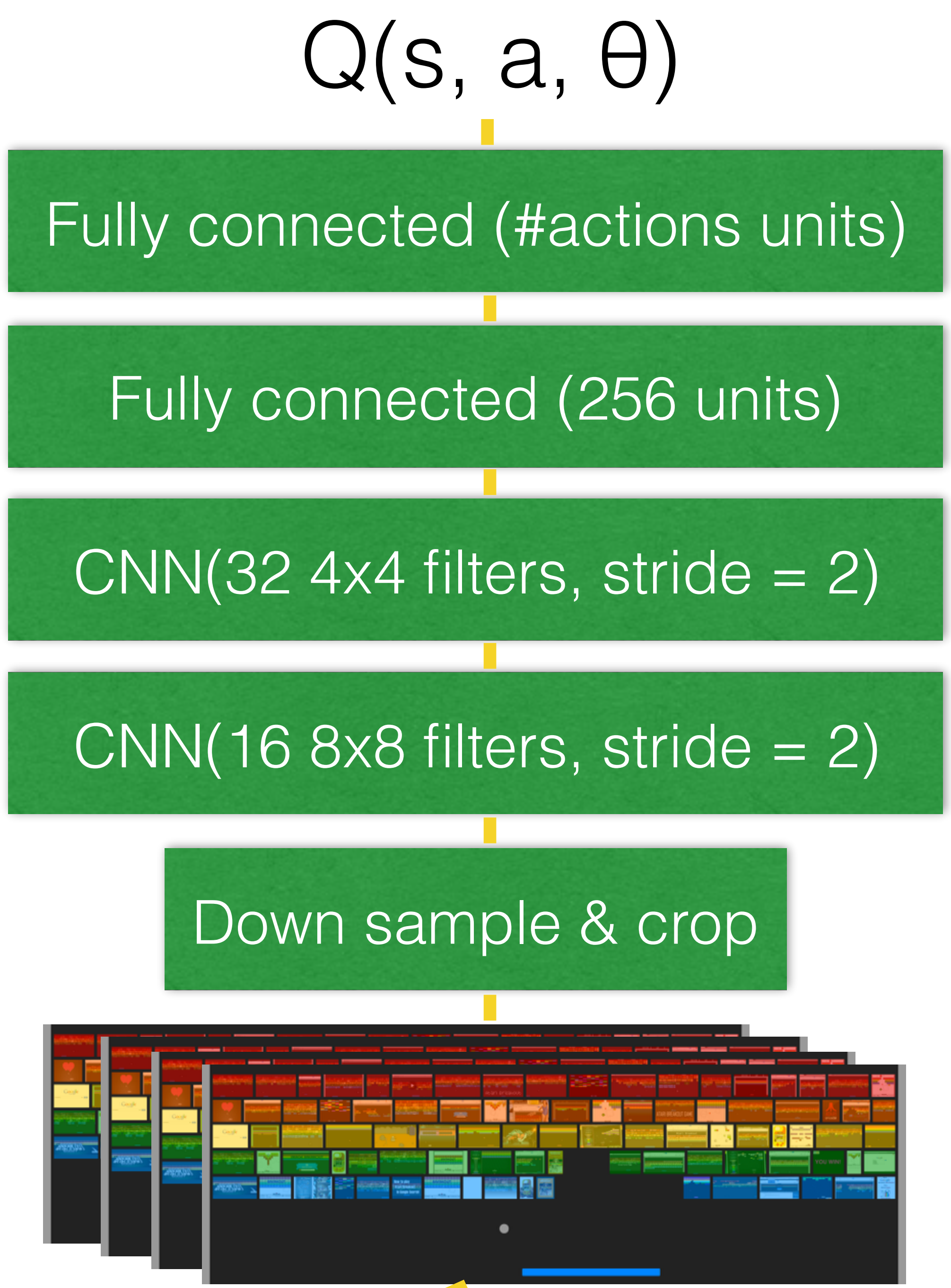
# Learning

- Bellman equation

$$Q\ (s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[ r + \gamma \max_{a'} Q\ (s', a') \middle| s, a \right]$$

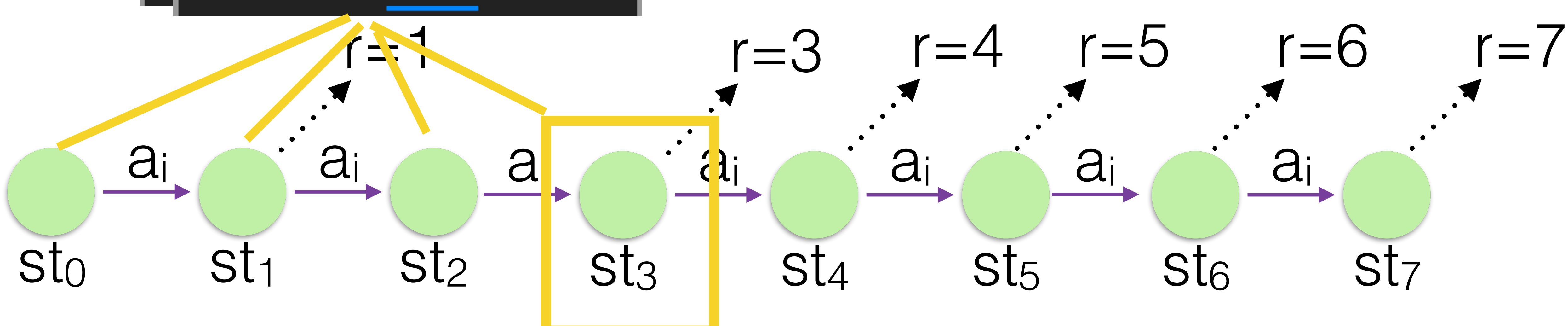
- $Q_i \rightarrow Q^*$ as $i \rightarrow \infty$
- $Q(s, a; \theta) \cong Q^*(s, a)$
- where $Q(s, a; \theta)$ is modelled with a deep neural network called a “Q-network”


Q(s, a, θ)
Loss func. MSE
Fully connected (#actions units)
Fully connected (256 units)
CNN(32 4x4 filters, stride = 2)
CNN(16 8x8 filters, stride = 2)
Down sample & crop
r + γ Q(s`, a`, θ)
r=1
r=3
r=4
r=5
r=6
r=7
st0
st1
st2
st3
st4
st5
st6
st7




# Deep Q-learning

- Initialise data and Q weights
- For each episode:
  - Init. and preprocess sequence $\phi(s_t)$
  - For t in T
    - Select the best action $a_t$ according to $Q(\phi(s_t), a, \theta)$
    - Execute action to get reward $r_t$ and image $x_{t+1}$
    - Store $\phi(s_t), a_t, r_t, \phi(s_{t+1}) \rightarrow D$
    - Sample a mini batch ^ from D then perform gradient decent to update weights



# Experiments

- 7 ATARI game (Beam rider, Breakout, Enduro, Pong, Q*bert, Seaquest, Space Invaders)

- Each trained on the same network (except actions, and scaled rewards according)

- Sequences contained the actions and states from the last 4 frames

# Experiments

- Randomly sampled s, a and s', a'
- RMSProp algorithm with mini batch of size 32
- Total of 10 million frames while training on every 4 (or 3) frames

# Experiments

- Outperformed previous state of the art Sarsa and Contingency
- Preformed better that humans in Breakout, Enduro, Pong

| | B. Rider | Breakout | Enduro | Pong | Q*bert | Seaquest | S. Invaders |
|---|---|---|---|---|---|---|---|
| **Random** | 354 | 1.2 | 0 | −20.4 | 157 | 110 | 179 |
| **Sarsa** [3] | 996 | 5.2 | 129 | −19 | 614 | 665 | 271 |
| **Contingency** [4] | 1743 | 6 | 159 | −17 | 960 | 723 | 268 |
| **DQN** | **4092** | **168** | **470** | **20** | **1952** | **1705** | **581** |
| **Human** | 7456 | 31 | 368 | −3 | 18900 | 28010 | 3690 |
| **HNeat Best** [8] | 3616 | 52 | 106 | 19 | 1800 | 920 | **1720** |
| **HNeat Pixel** [8] | 1332 | 4 | 91 | −16 | 1325 | 800 | 1145 |
| **DQN Best** | **5184** | **225** | **661** | **21** | **4500** | **1740** | 1075 |

# Future direction

AlphaGo

StarCraft II

# Future direction


units killed = 30
territory = 30%

units killed = 50
territory = 33%

WIN

$$st_0 \xrightarrow{a_i} st_1 \xrightarrow{a_i} st_2 \xrightarrow{a_i} st_3 \xrightarrow{a_i} st_4 \xrightarrow{a_i} st_5 \xrightarrow{a_i} st_6 \xrightarrow{a_i} st_7$$