Introduction to

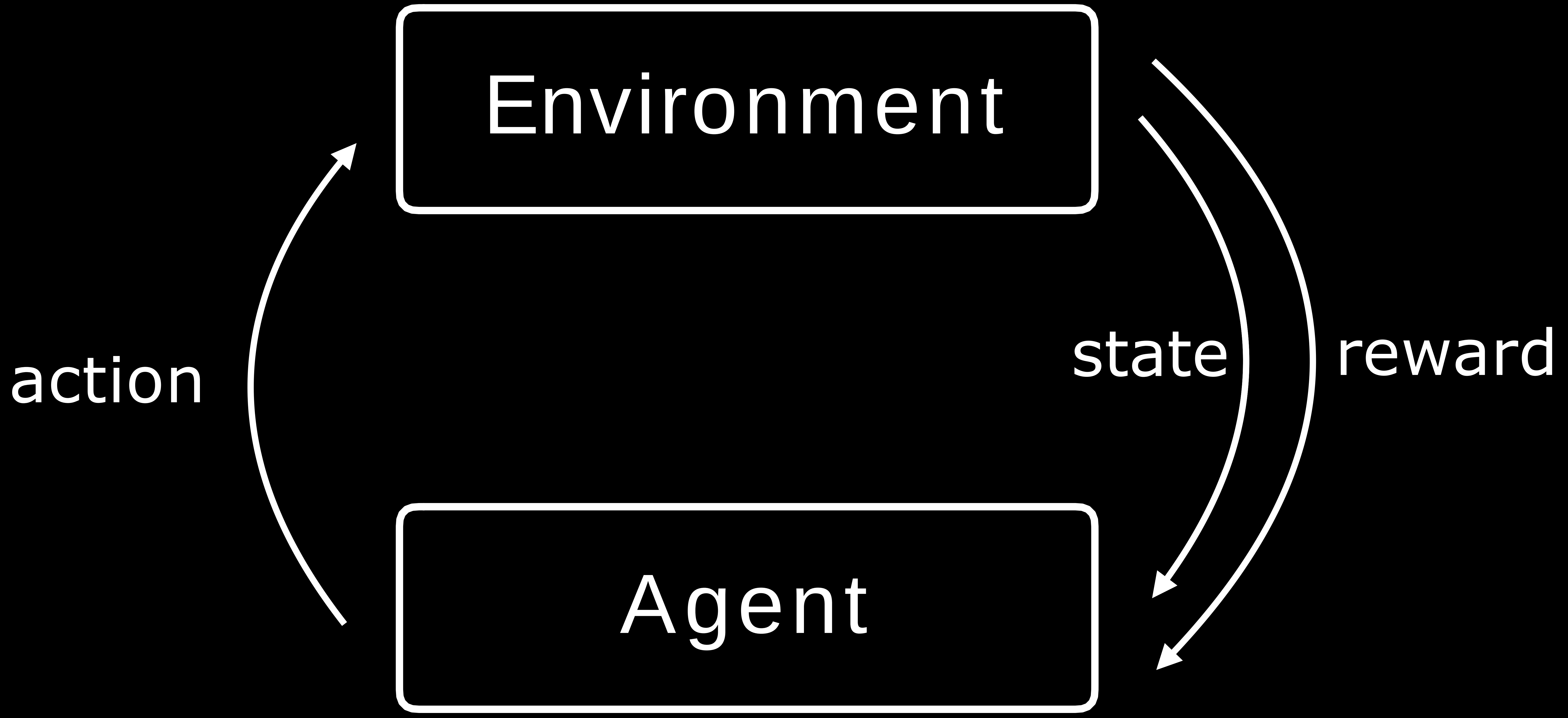
Artificial Intelligence

with Python

Reinforcement Learning

reinforcement learning

given a set of rewards or punishments, learn what actions to take in the future


Environment
action
state
reward
Agent

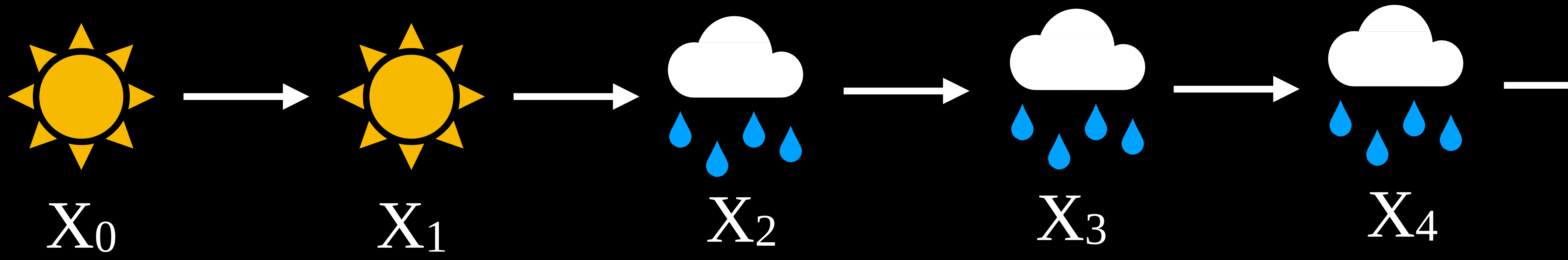
Markov Decision Process

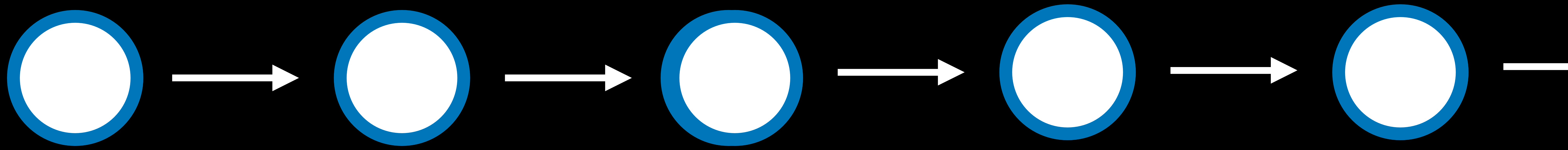
model for decision-making, representing states, actions, and their rewards

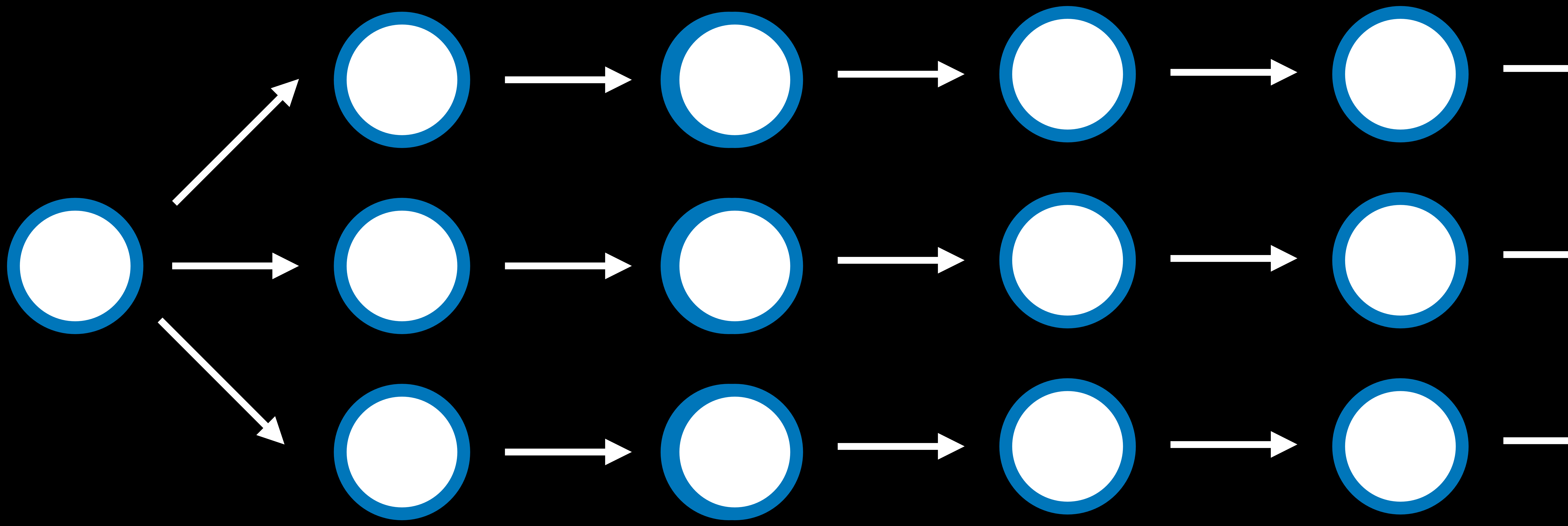
Markov Decision Process

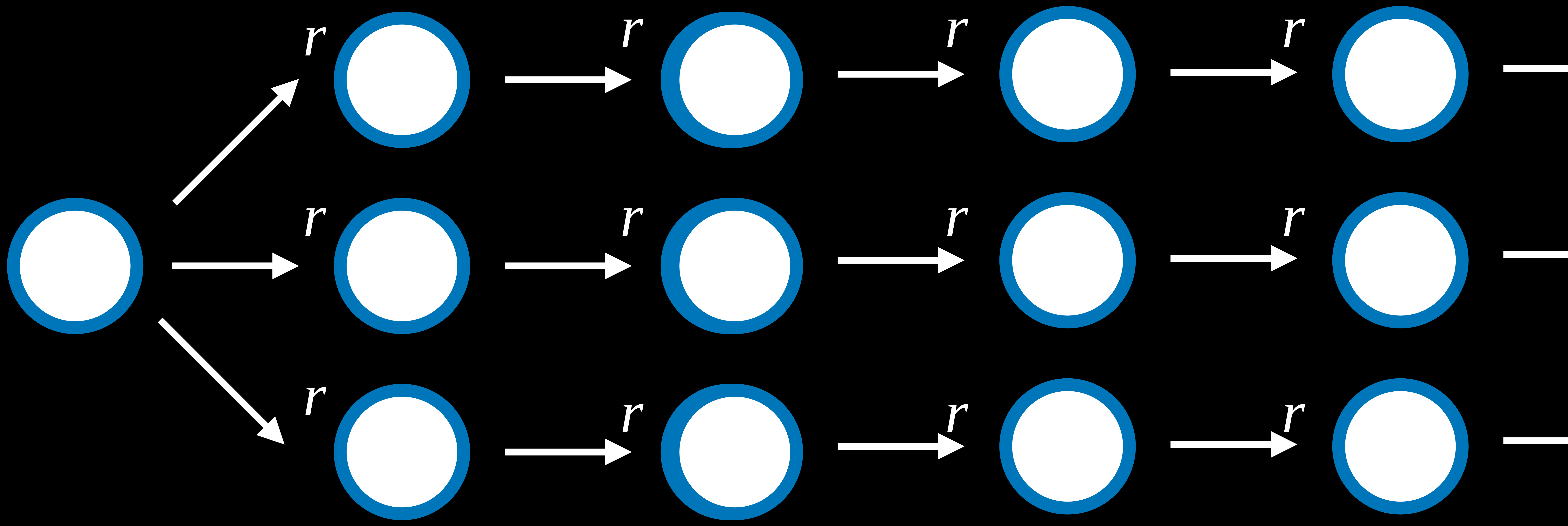
model for decision-making, representing states, actions, and their rewards

Markov Chain



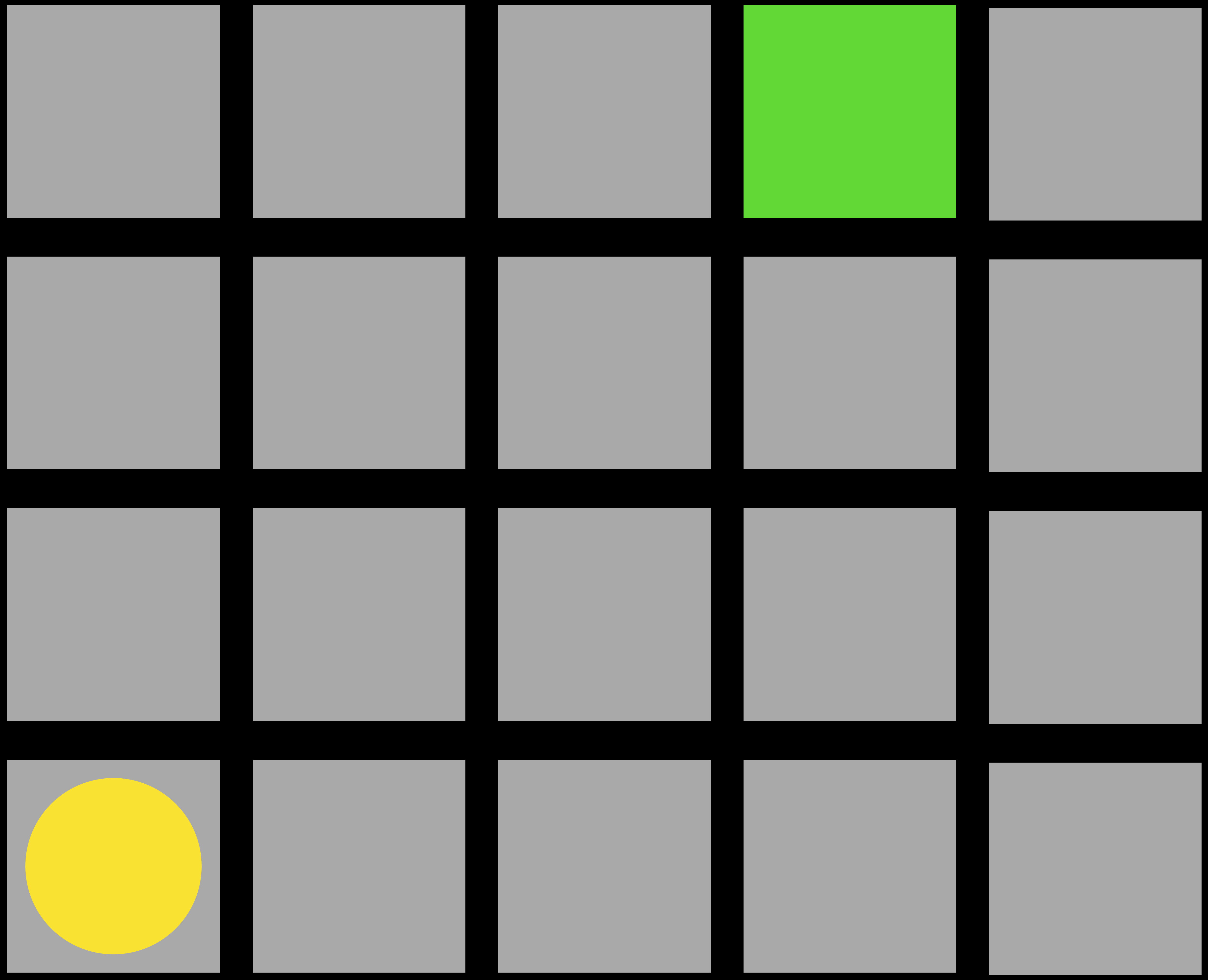


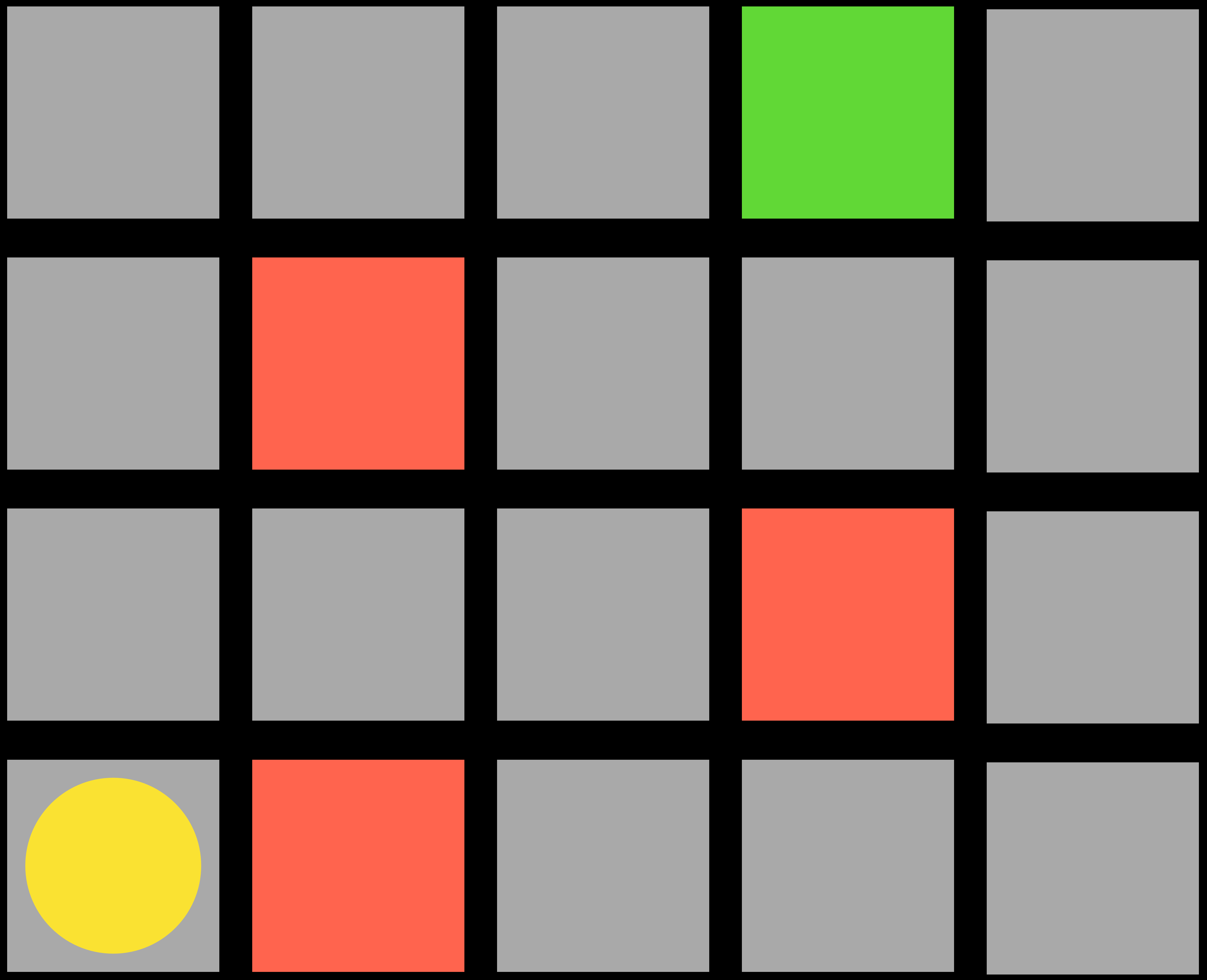


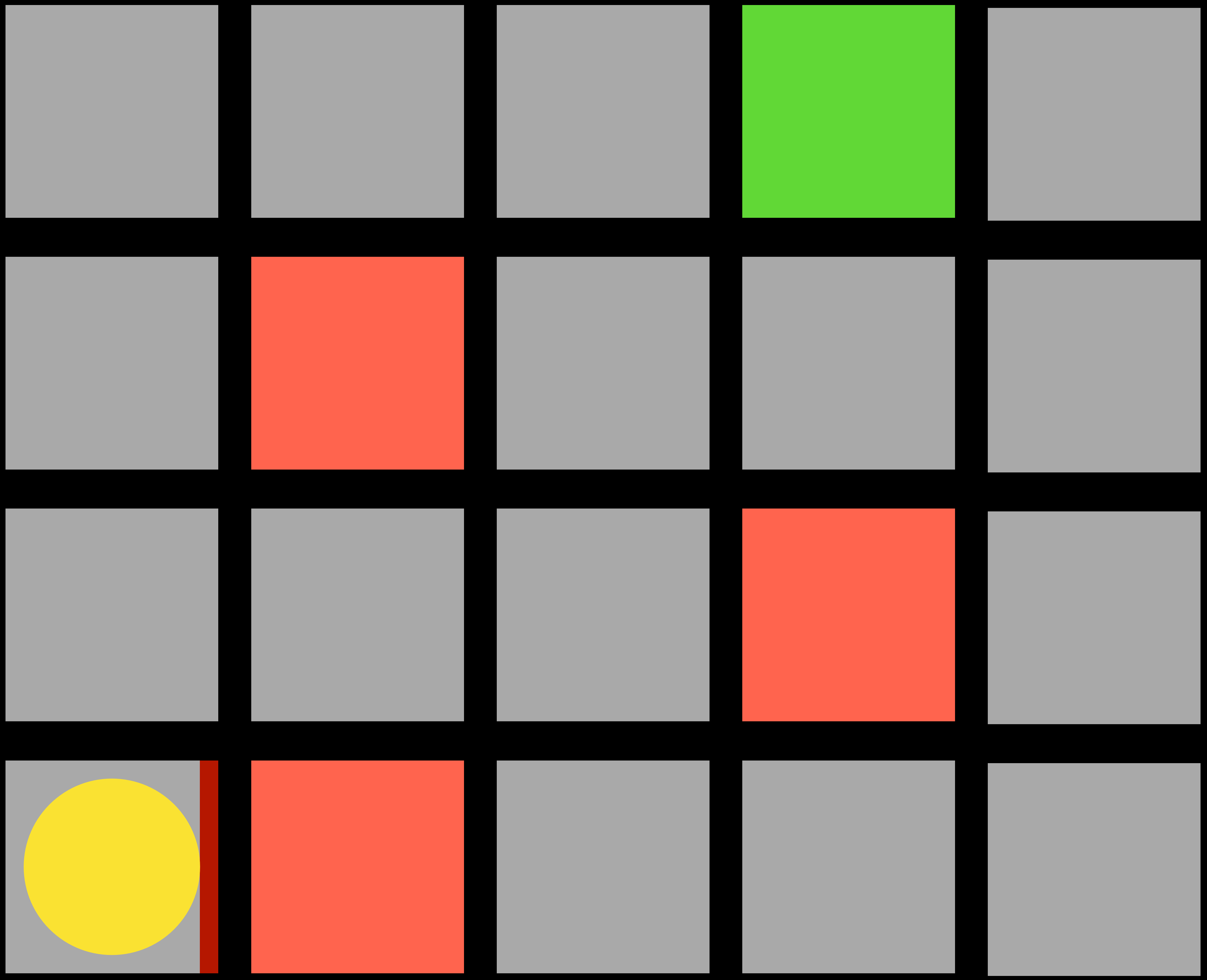

r
r
r
r
r
r
r
r
r
r
r
r

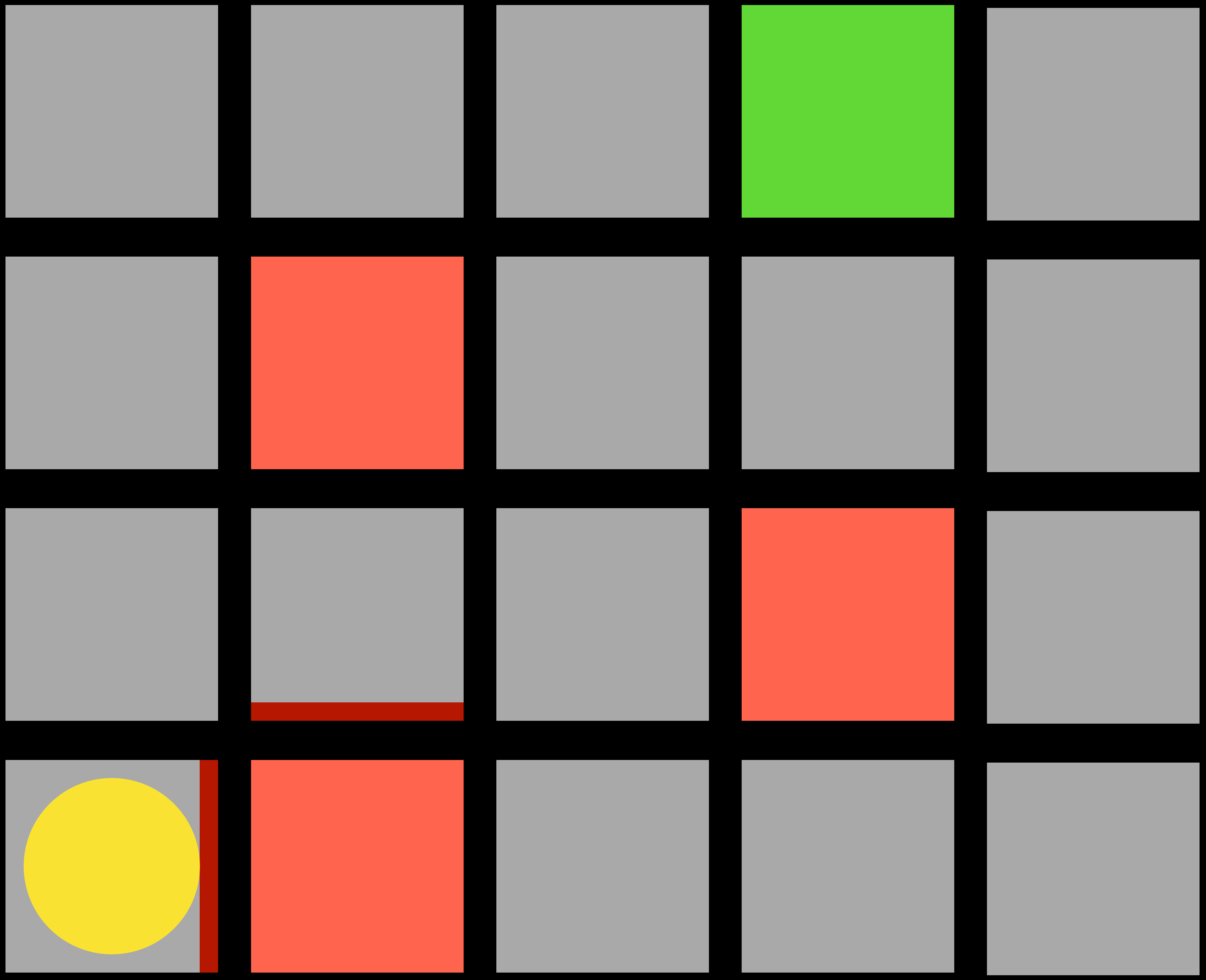
Markov Decision Process

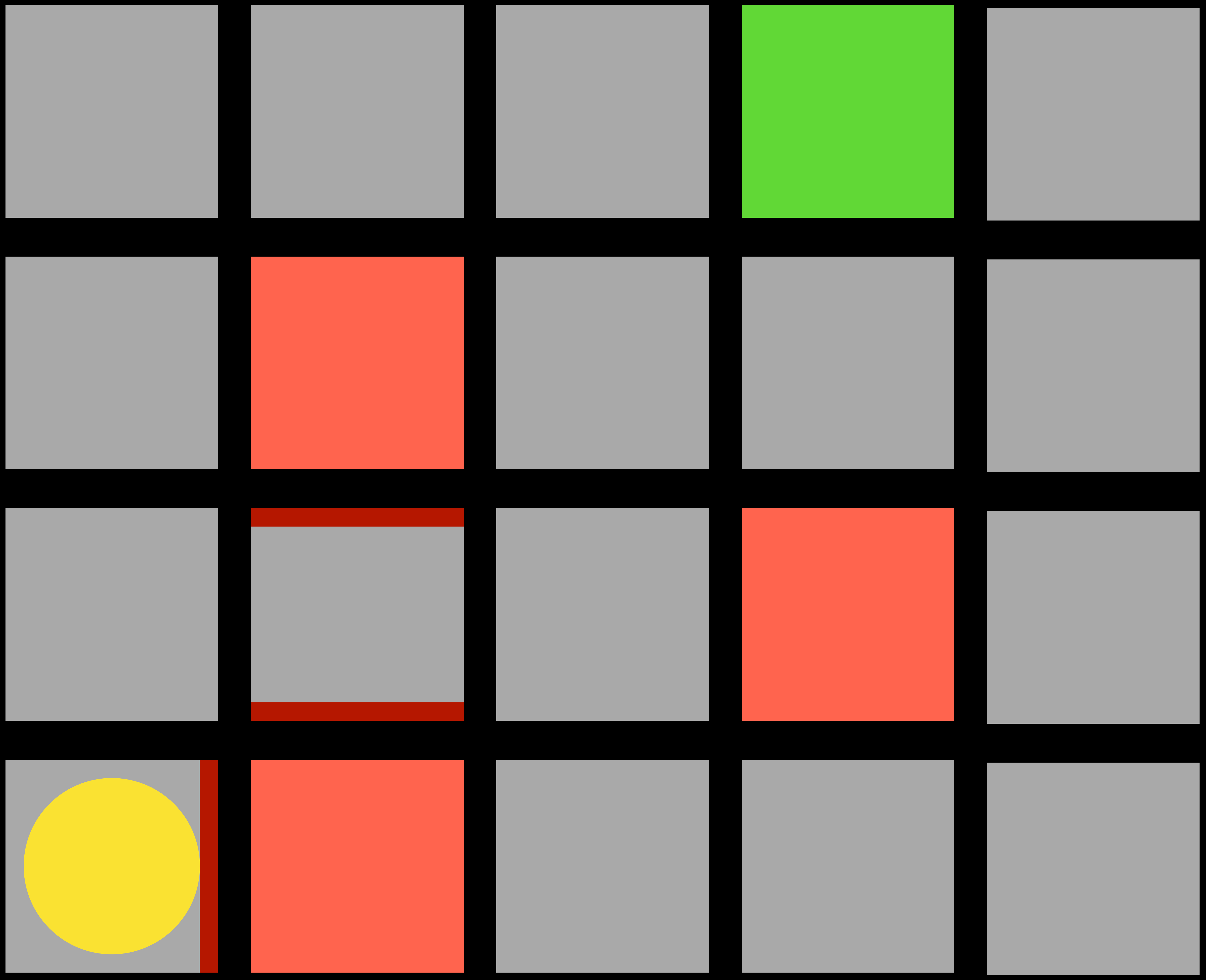
- Set of states S
- Set of actions $\textsc{Actions}(s)$
- Transition model $P(s' \mid s, a)$
- Reward function $R(s, a, s')$

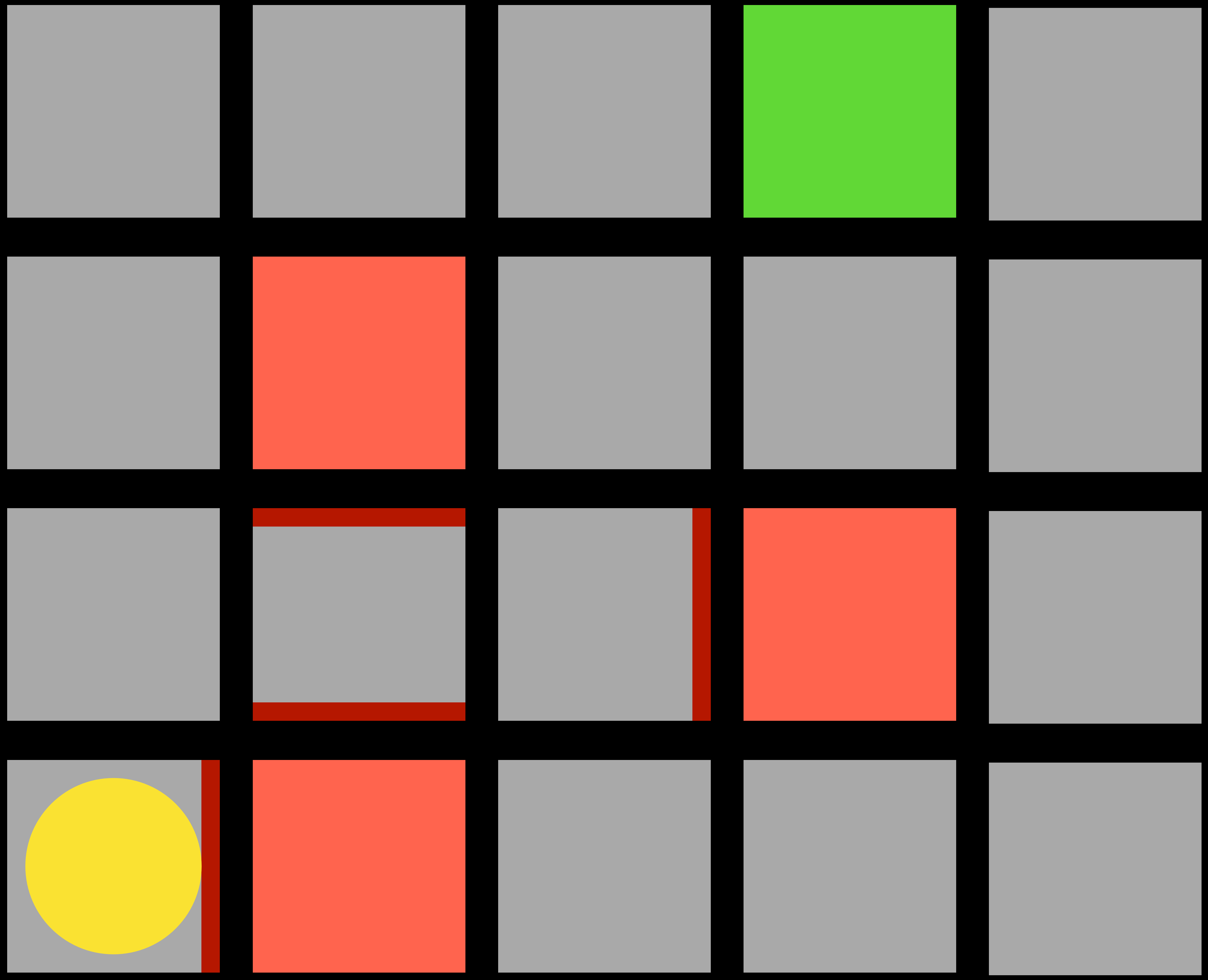


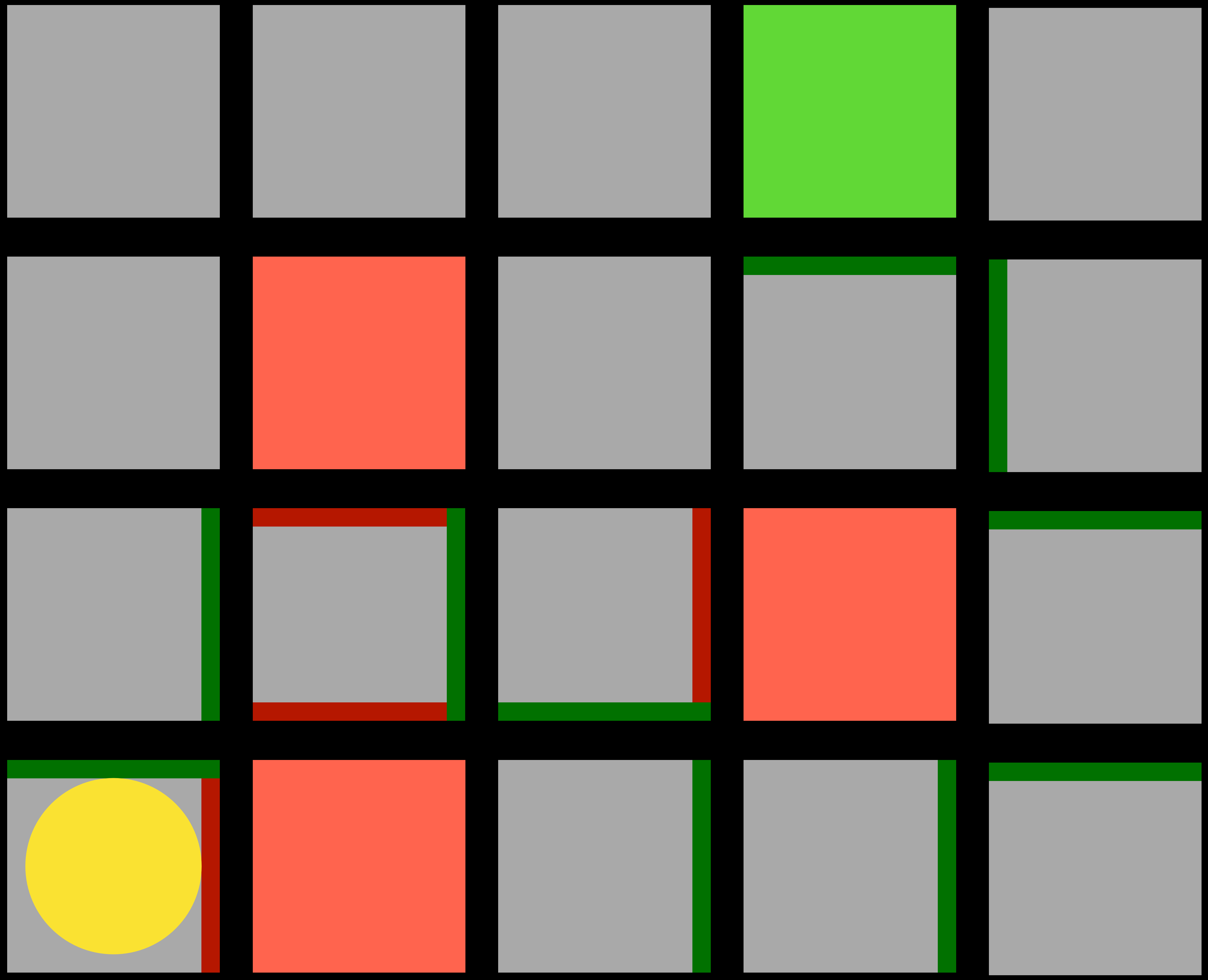


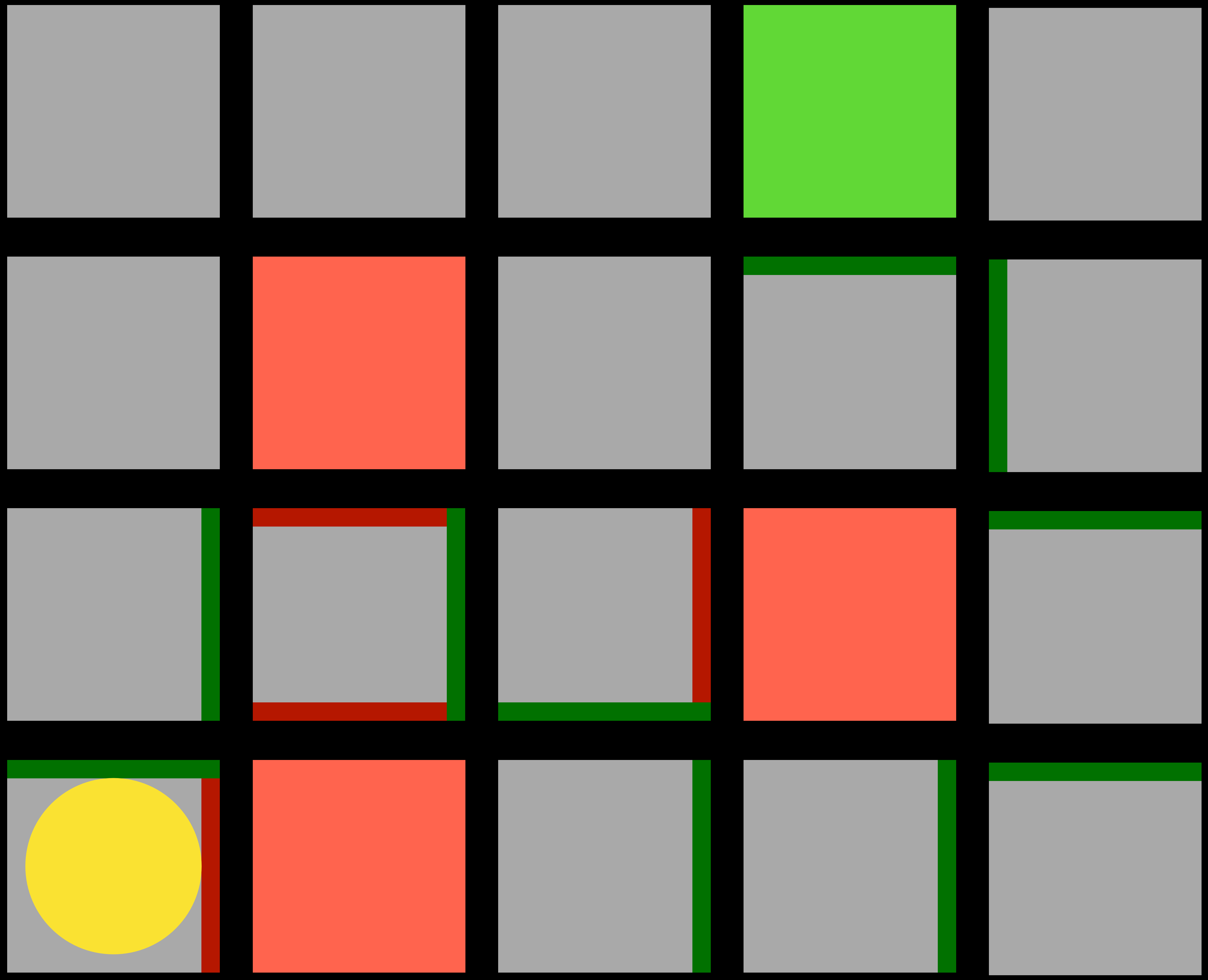












Q-learning

method for learning a function $Q(s, a)$, estimate of the value of performing action a in state s

Q-learning Overview

- Start with $Q(s, a) = 0$ for all s, a
- When we taken an action and receive a reward:
  - Estimate the value of $Q(s, a)$ based on current reward and expected future rewards
  - Update $Q(s, a)$ to take into account old estimate as well as our new estimate

Q-learning

- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r, we update:

$$Q(s, a) \leftarrow \mathrm{Q}(s, a) + \alpha(\text{new value estimate - old value estimate})$$

Q-learning

- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r, we update:

$$Q(s, a) \leftarrow \mathrm{Q}(s, a) + \alpha(\text{new value estimate} - \mathrm{Q}(s, a))$$

Q-learning

- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r, we update:

$$Q(s, a) \leftarrow \mathrm{Q}(s, a) + \alpha((\mathrm{r} + \text{future reward estimate}) - \mathrm{Q}(s, a))$$

Q-learning

- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r, we update:

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

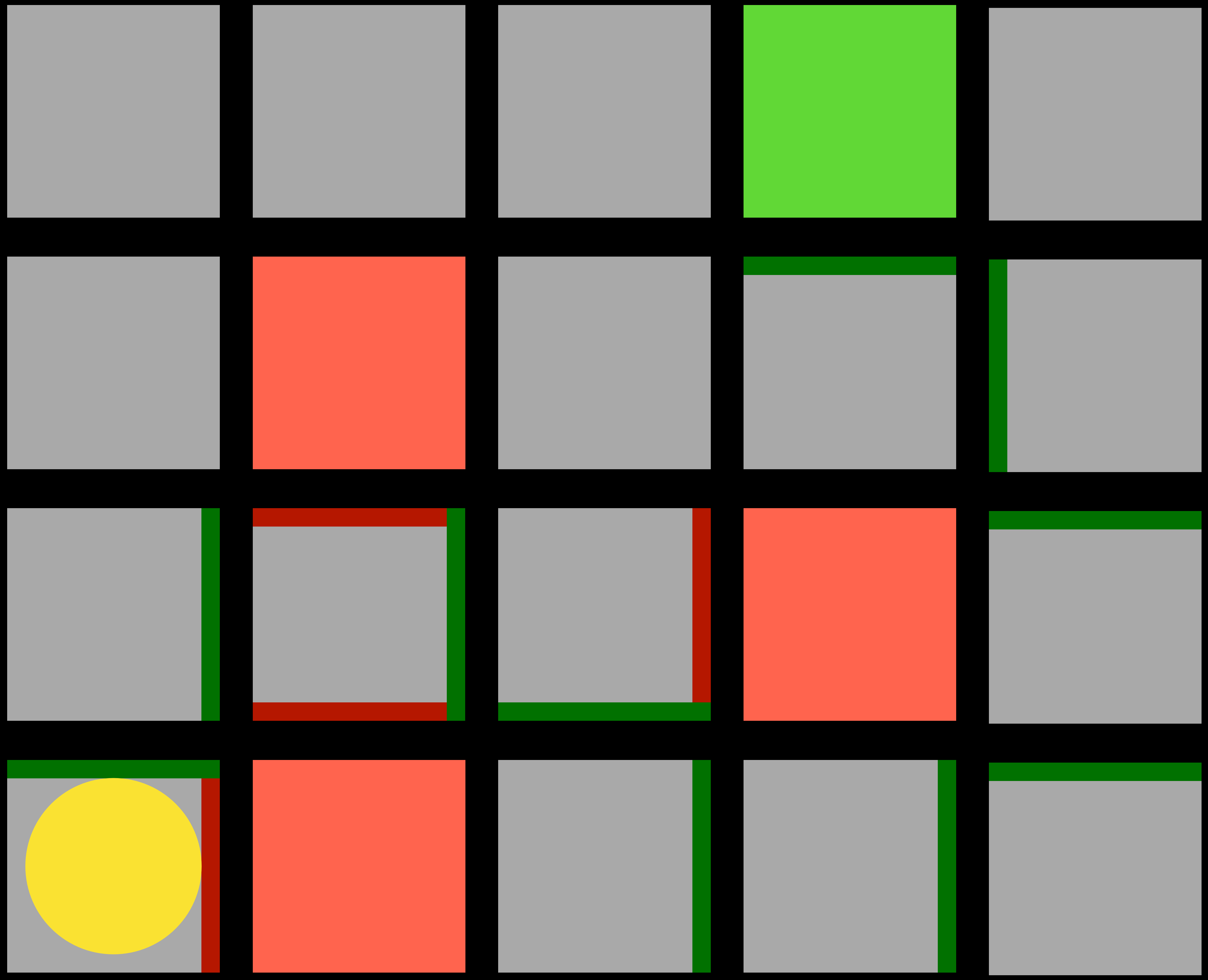
Q-learning

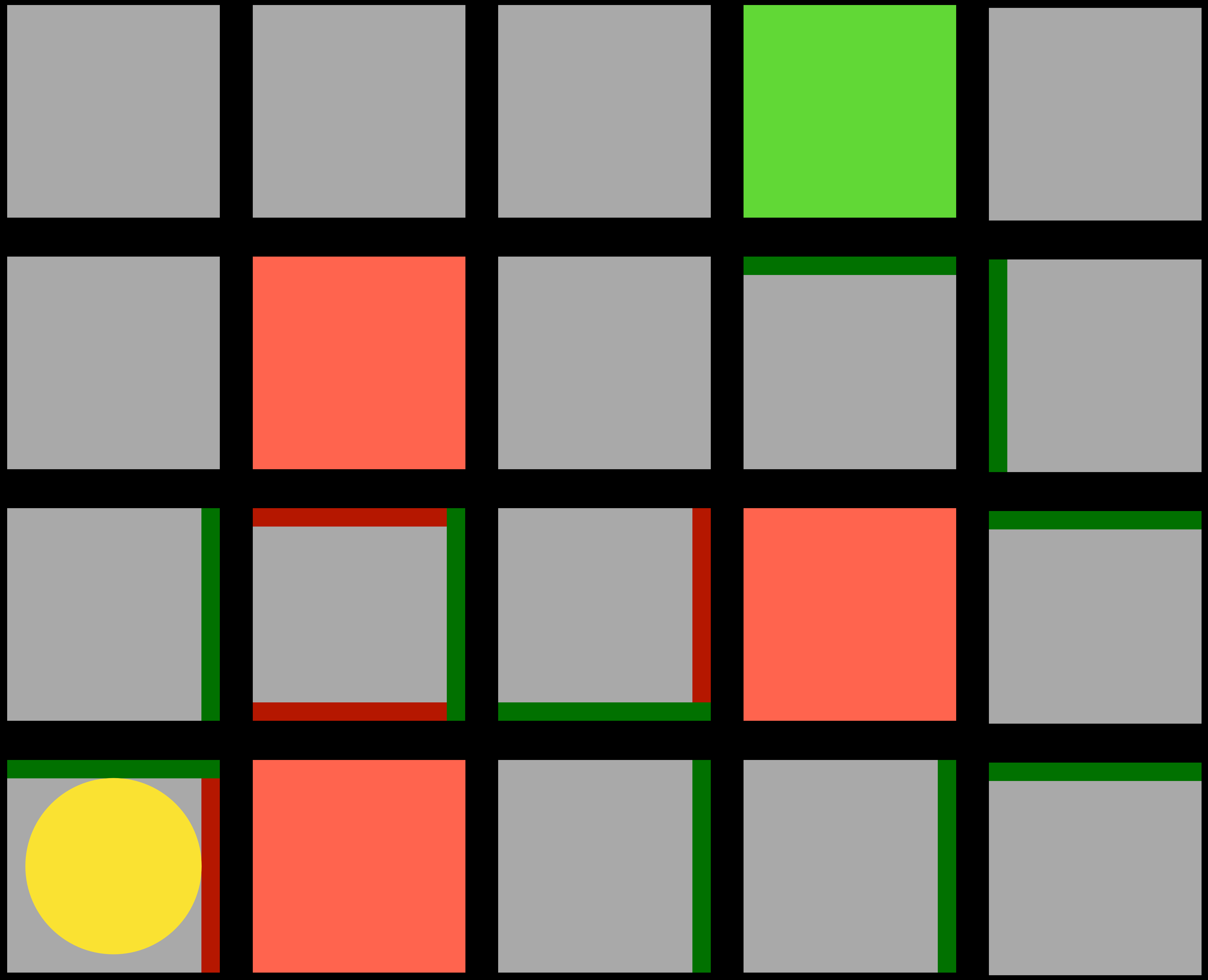
- Start with $Q(s, a) = 0$ for all s, a
- Every time we take an action a in state s and observe a reward r, we update:

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

Greedy Decision-Making

- When in state s, choose action a with highest $Q(s, a)$



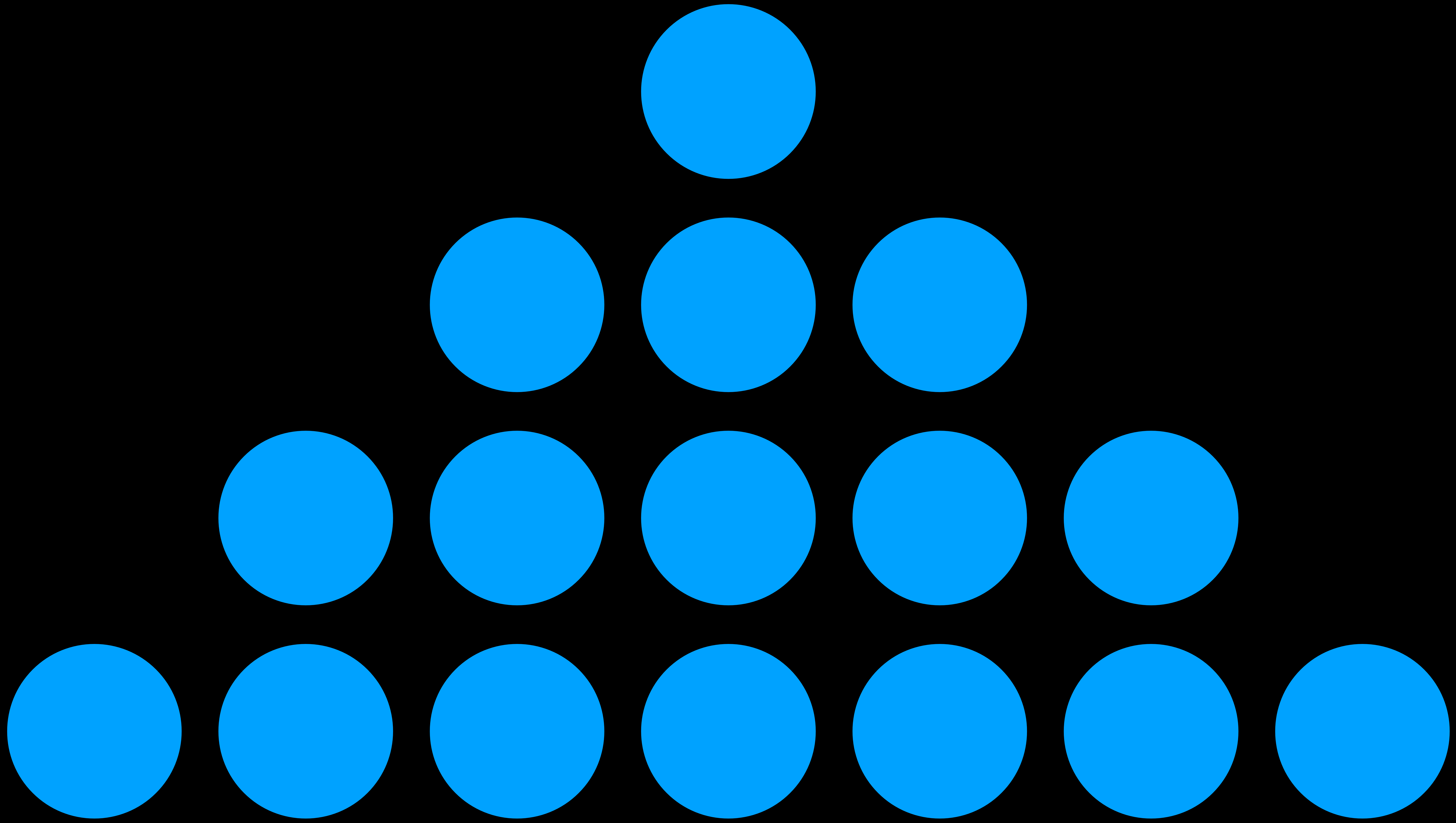


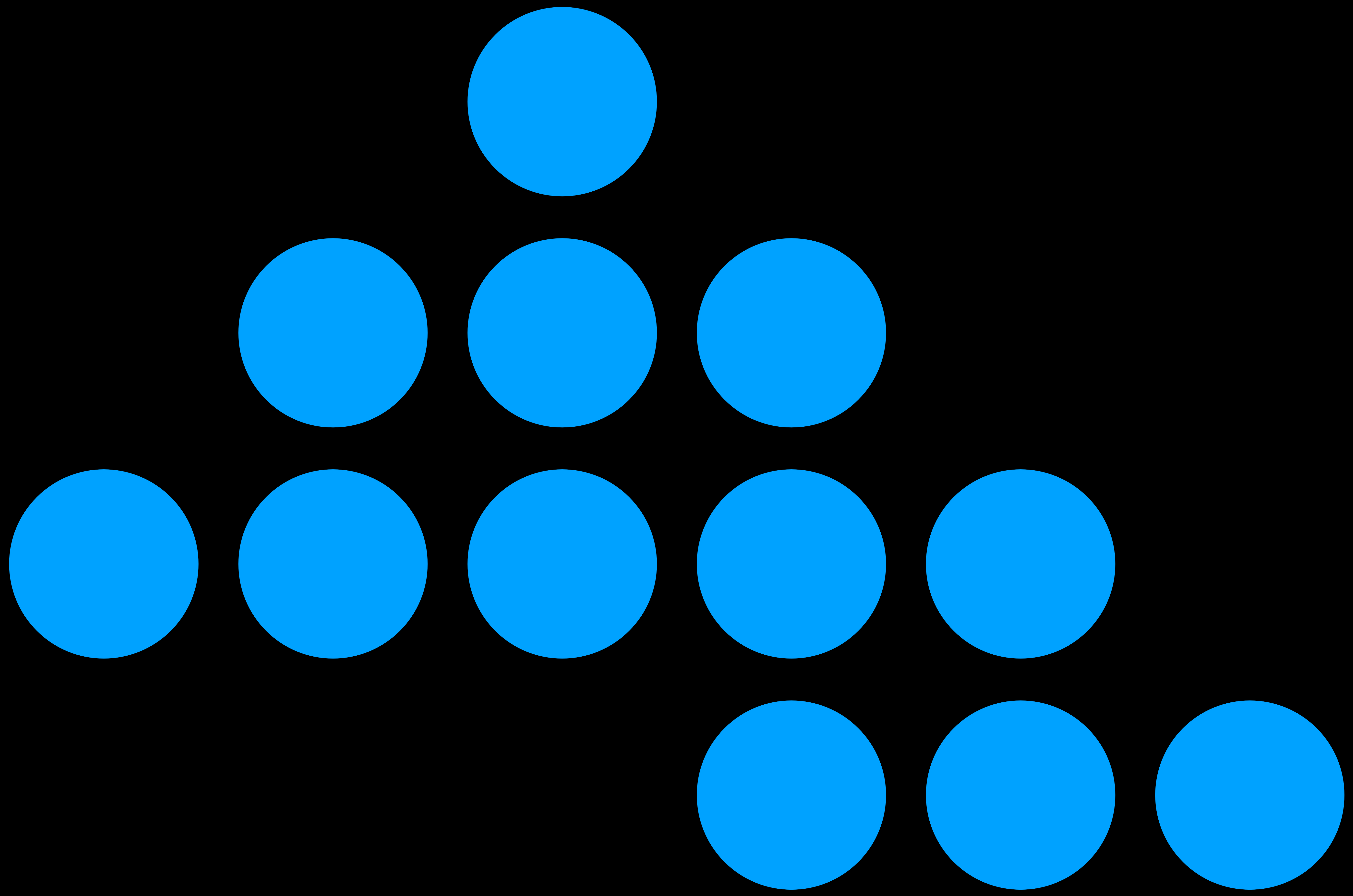
Explore vs. Exploit

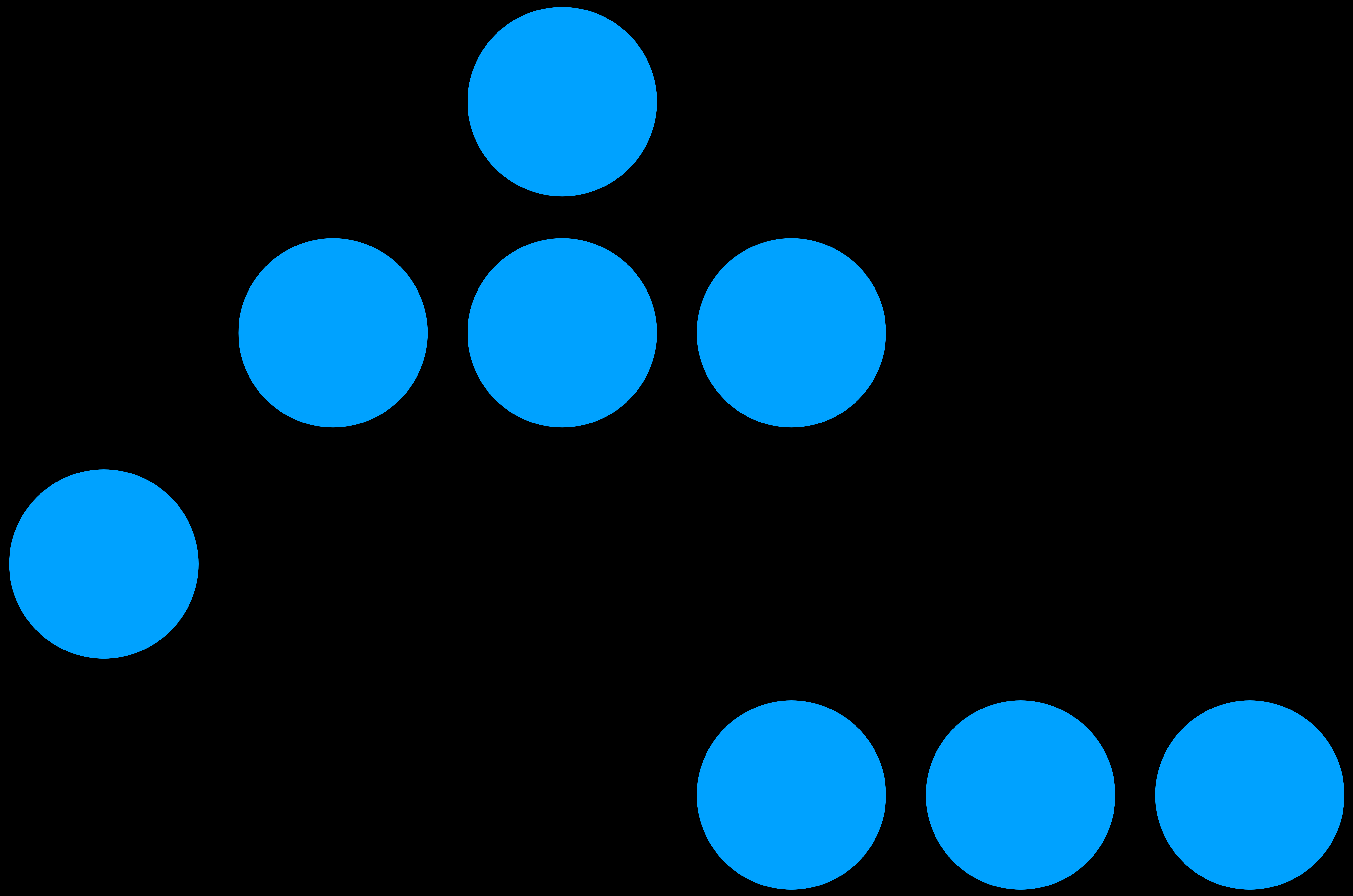
ε-greedy

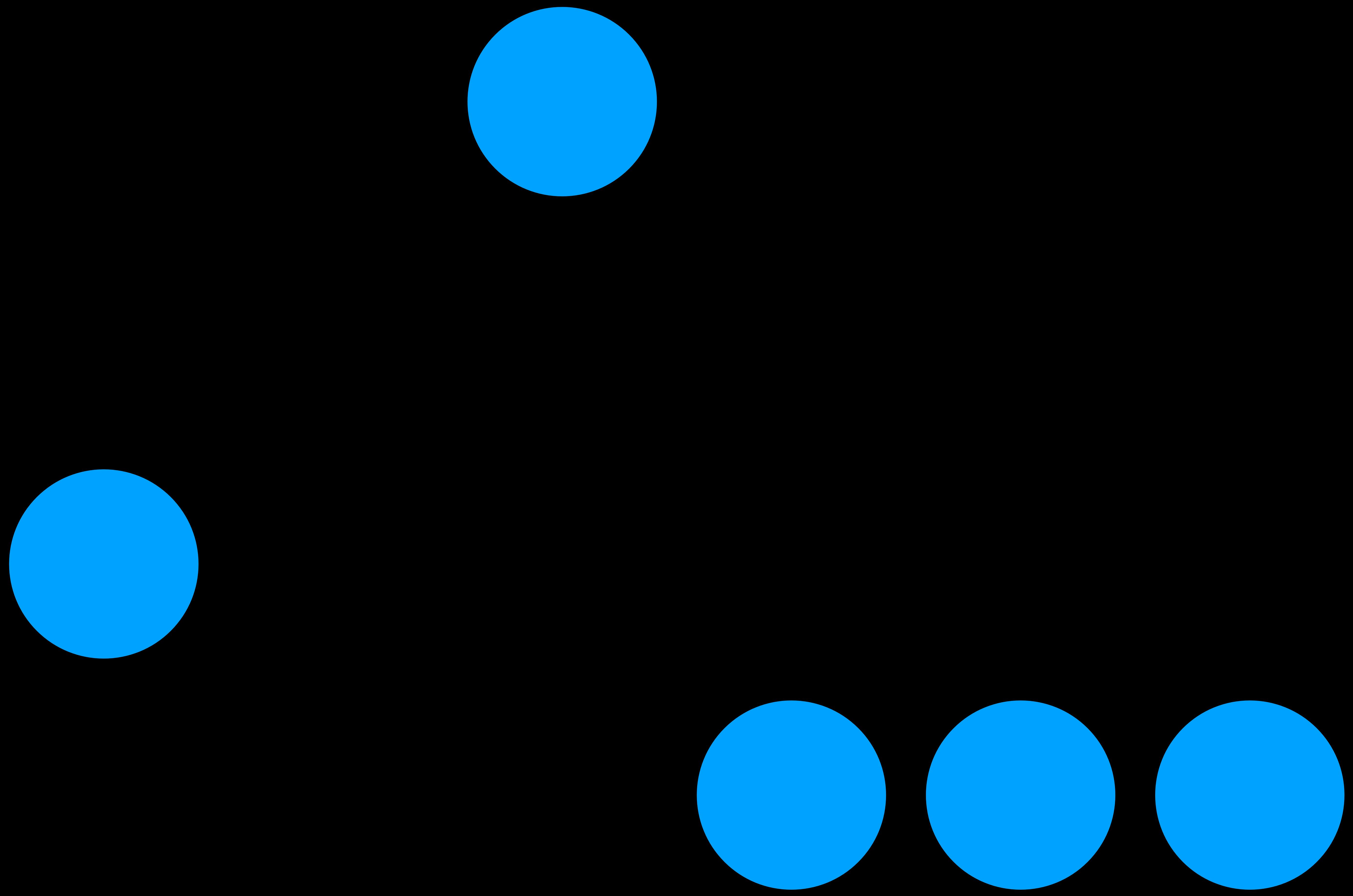
- Set ε equal to how often we want to move randomly.
- With probability 1 - ε, choose estimated best move.
- With probability ε, choose a random move.

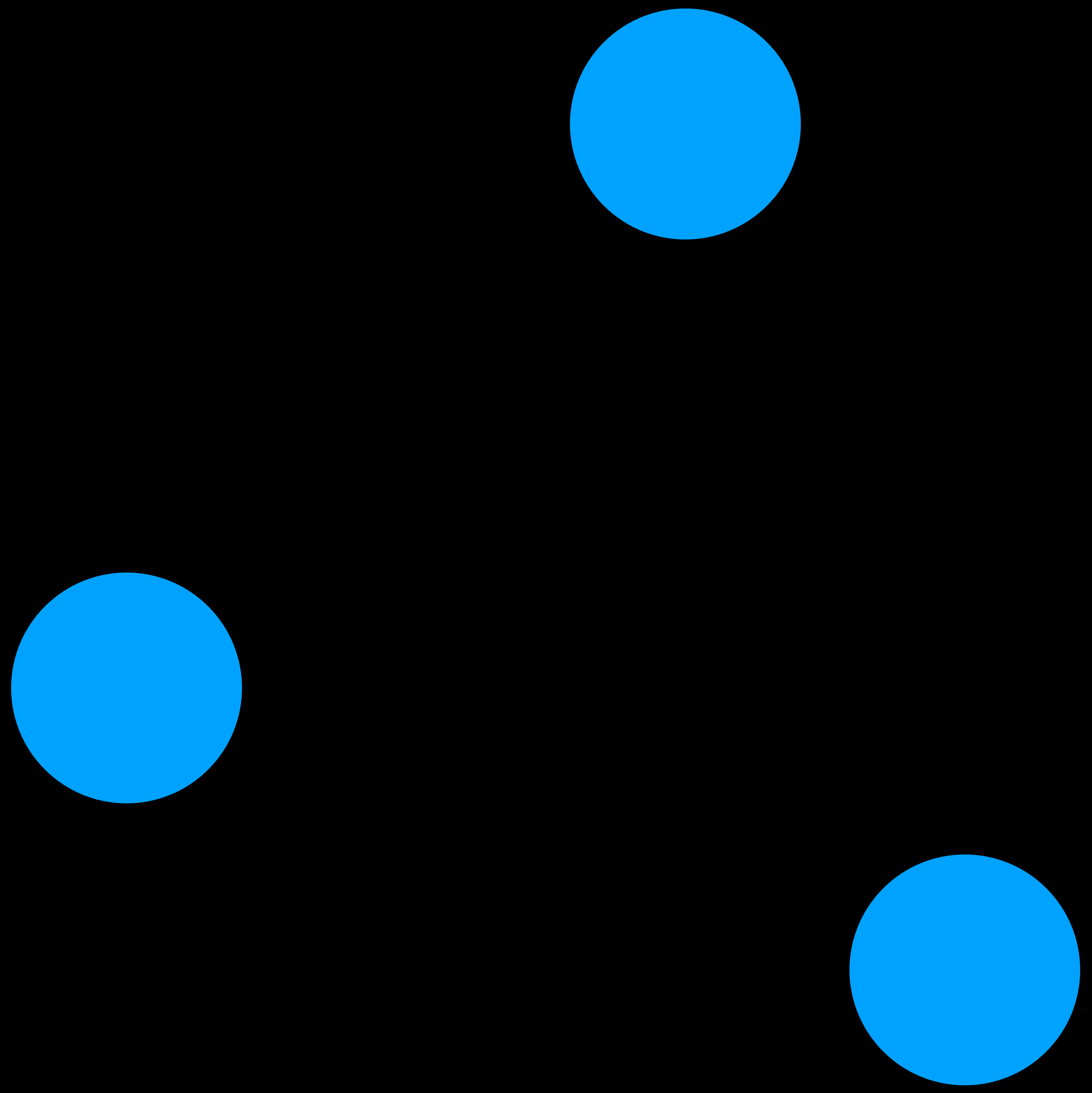
Nim

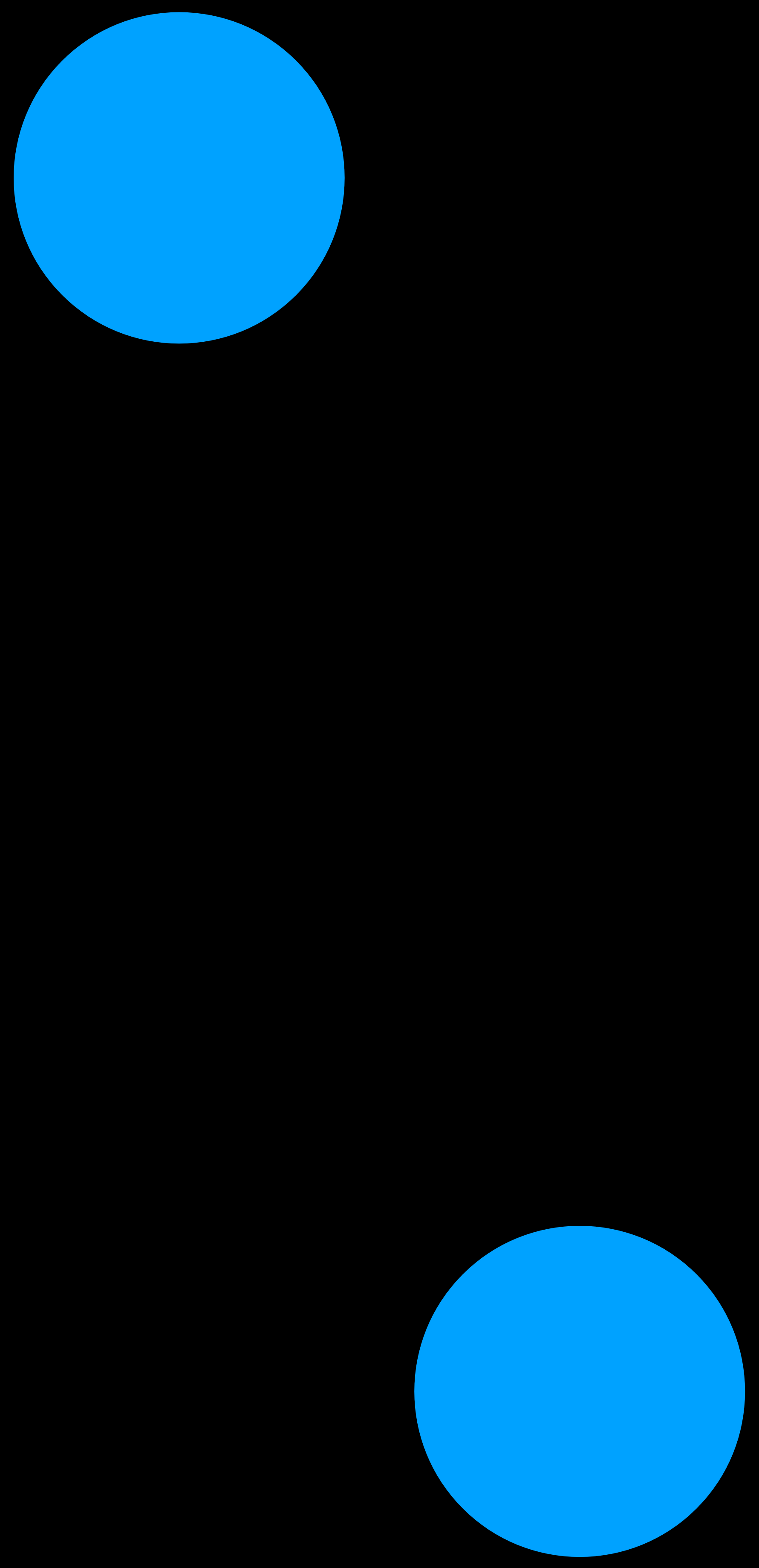


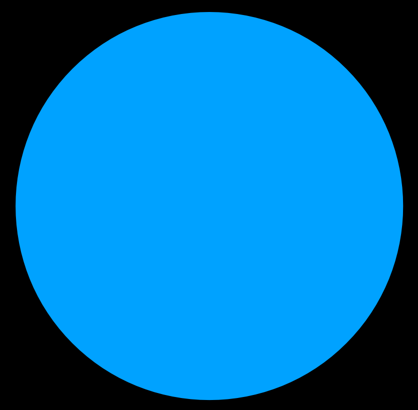












function approximation

approximating $Q(s, a)$, often by a function combining various features, rather than storing one value for every state-action pair

Reinforcement Learning

Introduction to

Artificial Intelligence

with Python