

# Обучение с подкреплением

Классическое обучение с подкреплением



# Про что этот курс?

Что будет на курсе:

- Немного классической теории
- Популярные современные алгоритмы
- Некоторые продвинутые алгоритмы RL, которые будут полезны на практике
- Рассказ о том, где и как применяется RL

Что можно посмотреть:

- [Reinforcement Learning: An Introduction](#) (Sutton & Barto)
- [Spinning Up in Deep RL](#) (OpenAI)
- [Practical RL](#) (Yandex)

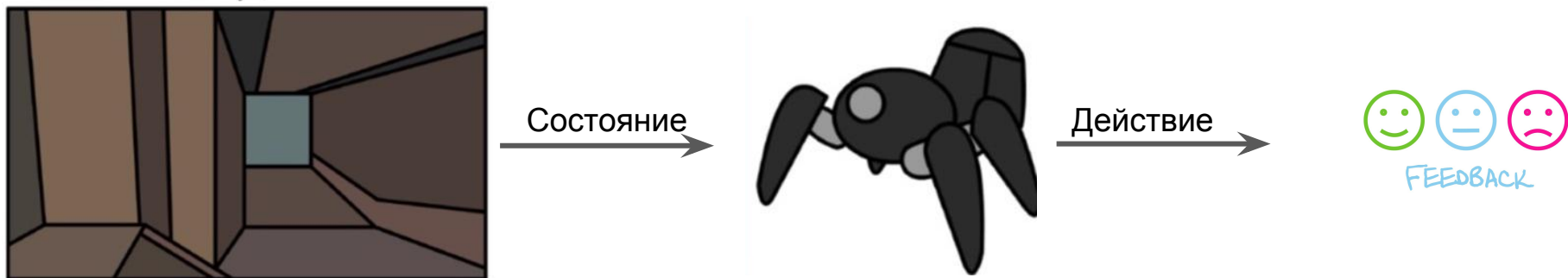
Оценка:

- Будет 4 ДЗ
- За каждое ДЗ можно получить от 0 до 10 баллов
- 32+ баллов - отлично, 24+ баллов - хорошо, 16+ баллов - удовлетворительно

# Задача обучения с учителем



# Обучение с учителем



# Обучение с подкреплением



# Обучение с подкреплением





# Supervised Learning vs Reinforcement Learning

## Обучение с учителем:

- Заранее известны ответы
- Есть фиксированный набор данных
- Результат зависит только от текущего решения

## Обучение с подкреплением:

- Набора данных нет
- Известных ответов нет
- Распределение получаемых данных сильно зависит от агента
- Результат зависит от последовательности принятых решений



# Простой случай: многорукие бандиты

Постановка задачи:

- Эпизод длится ровно  $t$  шагов.
- На каждом шаге можно совершить одно из  $k$  действий и получить награду, которая распределена случайно и зависит только от действия.
- Хотим максимизировать суммарную награду за эпизод.

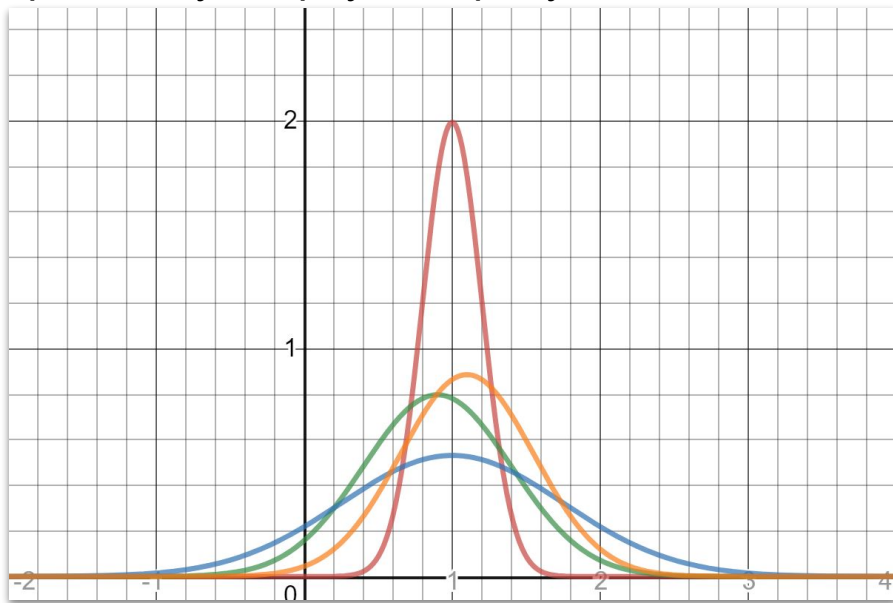




# Простой случай: многорукие бандиты

Постановка задачи:

- Эпизод длится ровно  $t$  шагов.
- На каждом шаге можно совершить одно из  $k$  действий и получить награду, которая распределена случайно и зависит только от действия.
- Хотим максимизировать суммарную награду за эпизод.



# Простой случай: многорукие бандиты

Определим value действия как математическое ожидание награды:

$$q(a) = \mathbb{E}_{r \sim p(r|a)} r$$

Если мы знаем value каждого действия, то можем просто выбрать то действие, у которого больше всего value.

Но value мы не знаем, и поэтому будем его приближать с помощью  $Q(a)$ . Хотим, чтобы  $Q(a)$  было как можно ближе к  $q(a)$ .

# Наивный алгоритм

Что будет, если мы будем всегда выбирать лучшее действие в соответствии с приближением, а затем обновлять приближение на основе наблюдений?

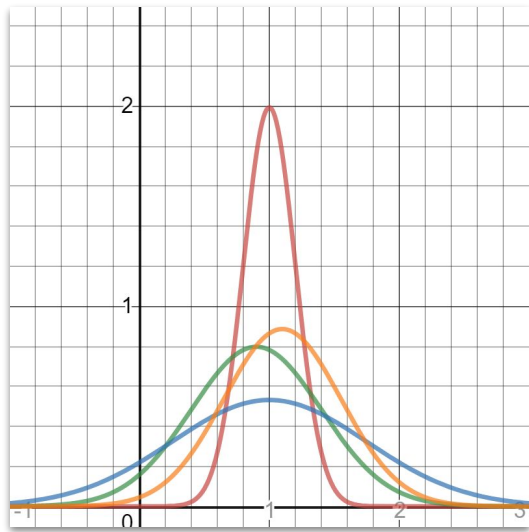
Алгоритм:

1. Инициализировать  $Q(a) \leftarrow 0$
2. Инициализировать  $N(a) \leftarrow 0$  // Счетчик количества применений действия
3. for  $t$  in range( $T$ ):
  - 3.1.  $a = \operatorname{argmax} Q(a)$
  - 3.2.  $r = \operatorname{step}(a)$  // Совершаем действие и получаем награду
  - 3.3.  $N(a) += 1$
  - 3.4.  $Q(a) = r / N(a) + (1 - 1 / N(a)) Q(a)$  // Рекуррентная формула среднего

# Наивный алгоритм

Что будет, если мы будем всегда выбирать лучшее действие в соответствии с приближением, а затем обновлять приближение на основе наблюдений?

Время, $t$	Действие, $a$	Q(1)	Q(2)	Q(3)	Q(4)	Награда, $r$
1	3	0	0	0	0	1.2050
2	3	0	0	1.2050	0	0.1138
3	3	0	0	0.6594	0	1.5140
4	3	0	0	0.9442	0	1.3417



Что пошло не так?

# Epsilon-greedy exploration

Когда мы выбираем действие с наибольшим value, мы используем уже полученные знания (**exploit**), однако чтобы получить эти знания, нам необходимо исследовать среду (**explore**).

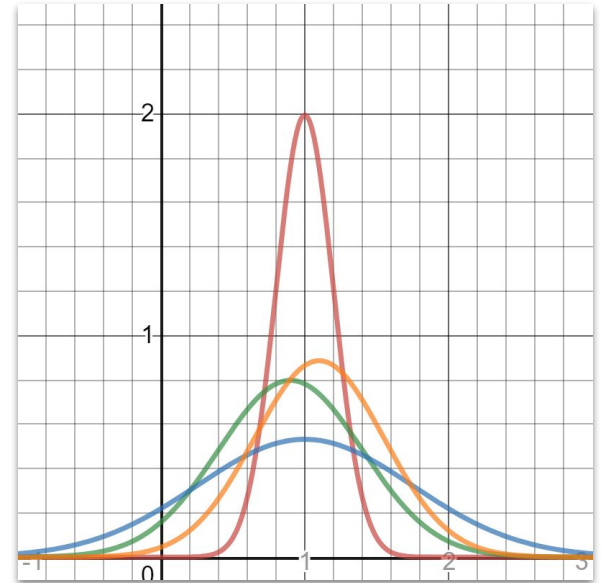
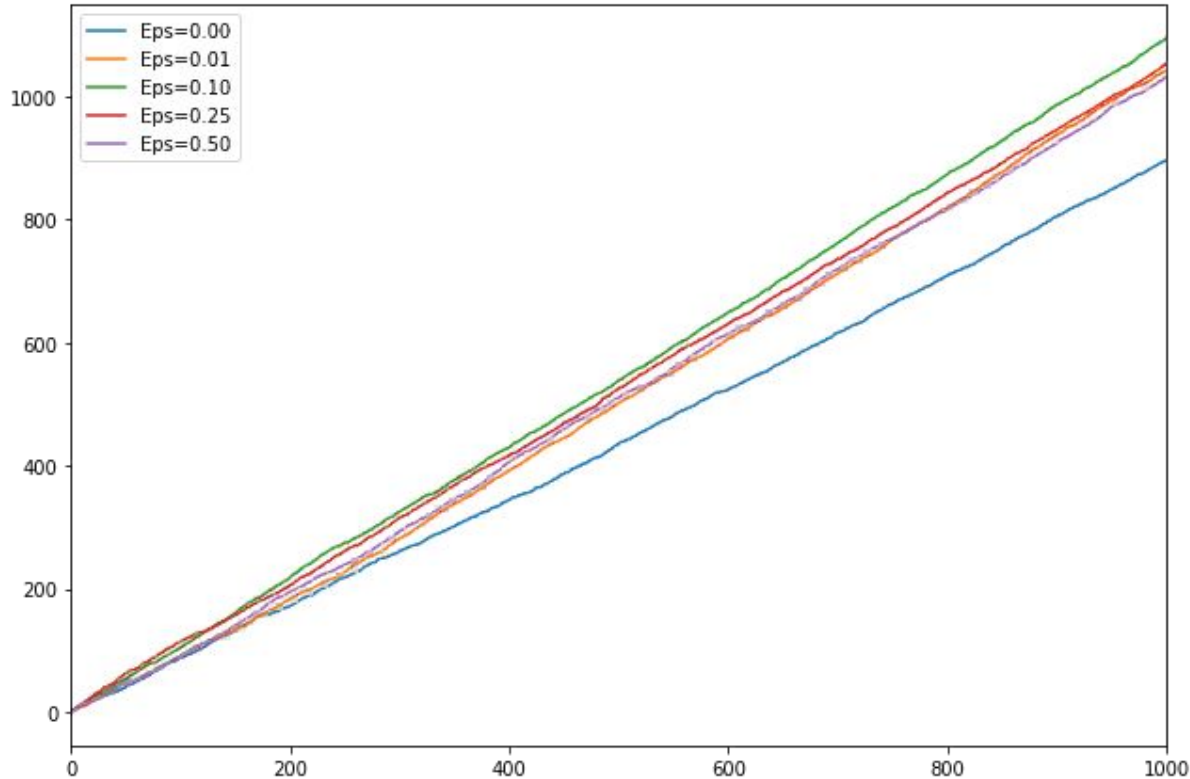
Как это делать?

Простой  $\epsilon$ -greedy алгоритм:

1. Инициализировать  $Q(a) \leftarrow 0$
2. Инициализировать  $N(a) \leftarrow 0$  // Счетчик количества применений действия
3. for  $t$  in range( $T$ ):
  - 3.1.  $a = \operatorname{argmax} Q(a)$  с вероятностью  $1 - \epsilon$ , случайное действие с вероятностью  $\epsilon$
  - 3.2.  $r = \text{step}(a)$  // Совершаем действие и получаем награду
  - 3.3.  $c(a) += 1$
  - 3.4.  $Q(a) = r / c(a) + (1 - 1 / c(a)) Q(a)$  // Рекуррентная формула среднего

# Epsilon-greedy exploration

Посмотрим на результат  $\epsilon$ -greedy алгоритма:





# Exploration via UCB sampling

Можно ли исследовать среду лучше, чем просто совершая случайные действия с определенной вероятностью?

**Upper-confidence bound** алгоритм предлагает повышать вероятность совершить действие, если мы совершаем его слишком редко:

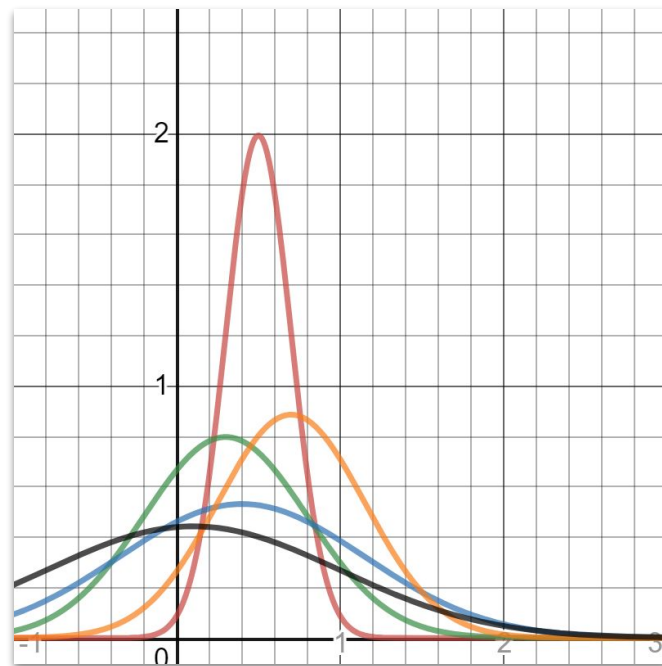
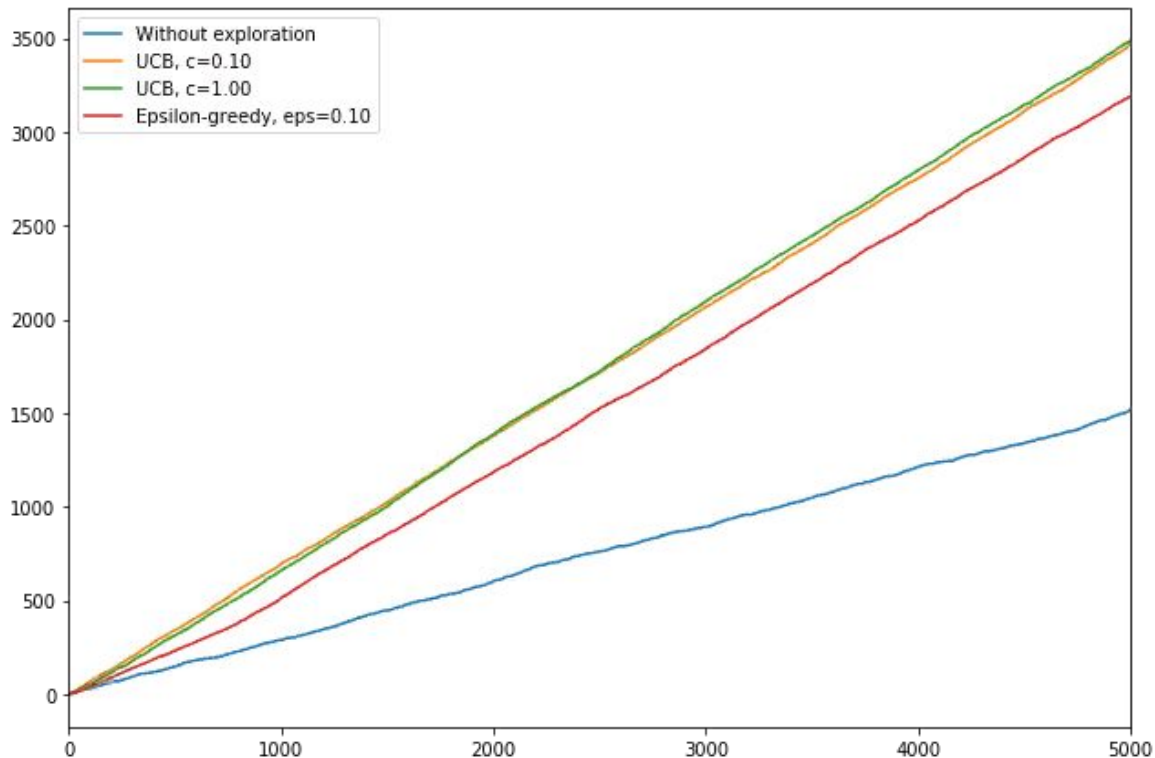
$$a = \arg \max Q(a) + c \cdot \sqrt{\frac{\ln t}{N(a)}}$$

Свойства:

- 1) Совершим каждое действие хотя бы один раз (т.к. в начале  $N(a) = 0$ )
- 2) Со временем количество неоптимальных действий убывает
- 3) Можно регулировать то, насколько часто исследуем среду с помощью константы

# Exploration via UCB sampling

Сравним два подхода:



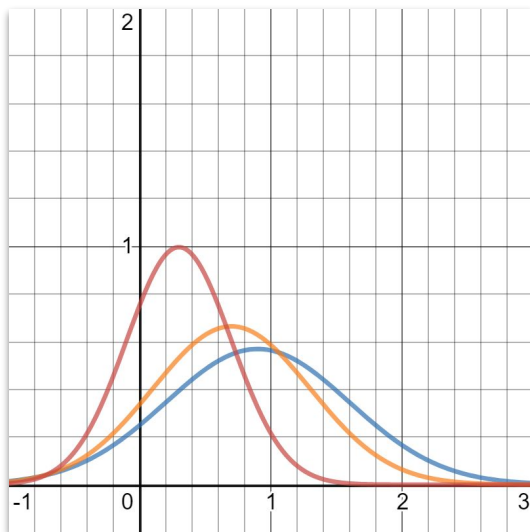
# Contextual bandits

Предположим, что у распределение наград теперь зависит от некоторого состояния, которое известно в начале.

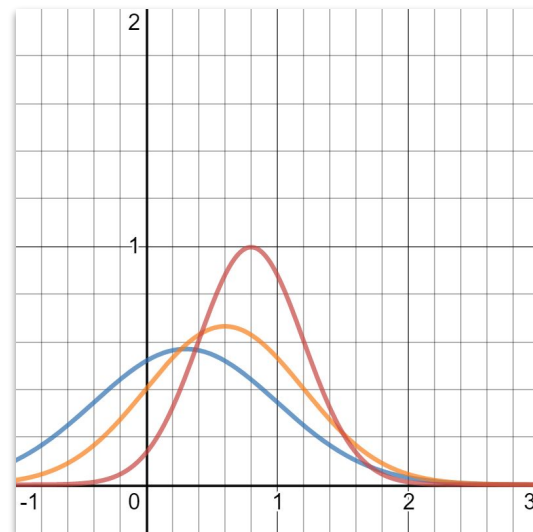
Старый подход работает плохо. Нужно учитывать **контекст**.

Можно просто приближать награду для каждого состояния независимо, но есть более сложные алгоритмы.

Состояние 1



Состояние 2



# Применение контекстуальных бандитов

## Artwork Personalization as Contextual Bandit



- **Environment:** Netflix homepage
- **Context:** Member, device, page, etc.
- **Learner:** Artwork selector for a show
- **Action:** Display specific image for show
- **Reward:** Member has positive engagement

# Markov Decision Process

**S** - множество состояний

**A** - множество действий

**T(s, a):**  $S \times A \rightarrow S$  - функция перехода (может быть случайной величиной)

**R(s, a, s'):**  $S \times A \times S \rightarrow R$  - функция награды (может быть случайной величиной)

**D(s):**  $S \rightarrow \{0, 1\}$  - функция, которая определяет, закончился ли эпизод

Задача: максимизировать  $\sum_{i=1}^T R(s_{i-1}, a_{i-1}, s_i)$

**Марковский процесс принятия решений (MDP)** - это набор (S, A, T, R, D). Чаще всего D присутствует неявно. В таком случае MDP определяется как (S, A, T, R)

# Grid World

S: множество возможных положений на сетке

A: {идти вверх, идти налево, идти направо, идти вниз}

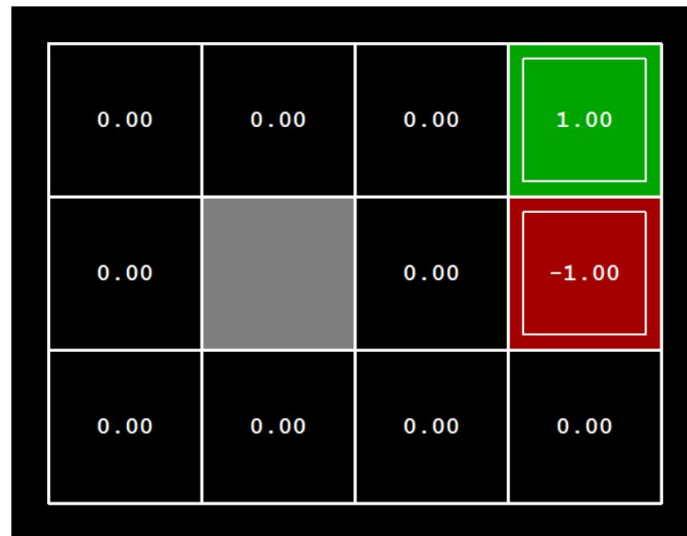
T(s, a): сместиться в нужную сторону если есть путь

R(s, a, s'): +1 если в (4, 3), -1 если в (4, 2), иначе 0

D(s): 1 если совершаем любое действие в (4, 3) или в (4, 2), иначе 0

Игру начинаем в (1, 1)

Задача: максимизировать  $\sum_{i=1}^T R(s_{i-1}, a_{i-1}, s_i)$





# Markov Decision Process

$S$  - множество состояний

$A$  - множество действий

$T(s, a): S \times A \rightarrow S$  - функция перехода (может быть случайной величиной)

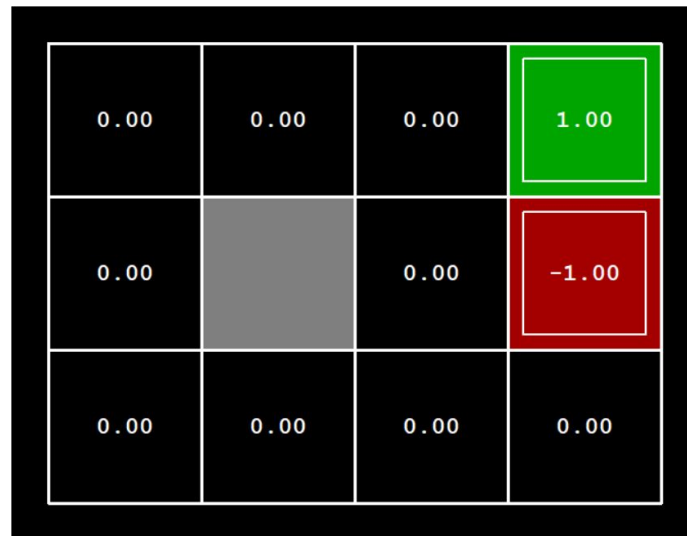
$R(s, a, s'): S \times A \times S \rightarrow R$  - функция награды (может быть случайной величиной)

$D(s): S \rightarrow \{0, 1\}$  - функция, которая определяет, закончился ли эпизод

Задача: максимизировать  $\sum_{i=1}^T R(s_{i-1}, a_{i-1}, s_i)$

Проблема: можем решать бесконечно долго

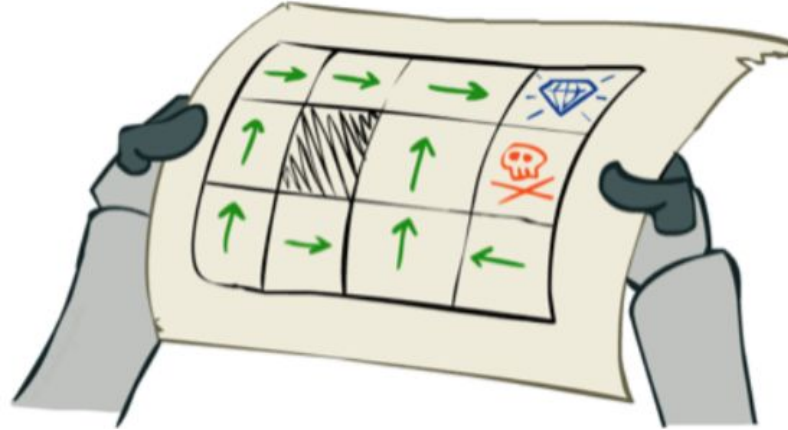
Новая задача:  $\sum_{i=0}^T \gamma^i R(s_i, a_i, s_{i+1}), 0 < \gamma < 1$



0.00	0.00	0.00	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

# Policy

$\pi$ :



**Политика/стратегия**  $\pi(s) : S \rightarrow A$  - правило, по которому агент принимает решения.

**Оптимальная стратегия:**  $\pi^*(s) : \sum_{i=0}^T \gamma^i R(s_i, a_i, s_{i+1}) \xrightarrow{a} \max$

На самом деле хотим найти оптимальную политику!

# Value Function

**Policy Value Function** определяет то, какую награду получит агент, который находится в заданном состоянии

$$V_{\pi}(s) = \mathbb{E}_{a \sim \pi(s)} \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')} [r + \gamma V_{\pi}(s')]$$

Действие политики

Ожидаемая суммарная награда

Реакция среды на действие

**Value Function** определяет то, насколько большую суммарную награду мы можем получить если находимся в определенном состоянии

$$V(s) = V_{\pi^*}(s) = \max_a \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')} [r + \gamma \cdot V_{\pi^*}(s')]$$

А оптимальную стратегию можно записать как:

$$\pi^*(s) = \arg \max_a \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')} [r + \gamma \cdot V_{\pi^*}(s')]$$

# Value Function



# Policy Evaluation

Дана политика  $\pi$ . Хотим посчитать value function для нее.

Алгоритм:

1. Инициализируем вектор  $V$
2. Пока не сошлись:

2.1. Для всех состояний  $s \in S$ :

2.1.1. В стохастическом случае

$$V_{\pi}(s) = \mathbb{E}_{a \sim \pi(s)} \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')} [r + \gamma V_{\pi}(s')]$$

В детерминированном случае

$$V_{\pi}(s) = R(s, a, T(s, a)) + \gamma V_{\pi}(T(s, a))$$

# Policy Iteration

Алгоритм:

1. Пока политика улучшается:
  - 1.1. Получаем  $V$  с помощью алгоритма Policy Evaluation
  - 1.2. Для всех состояний  $s \in S$ :
    - 1.2.1.  $\pi_{new}(s) = \arg \max_a \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')} [r + \gamma V_{\pi}(s')]$
    - 1.2.2. Если новое действие отличается от старого, то вернуться к шагу 1.1

Замечание 1:

В алгоритме Policy Evaluation можно инициализировать значение  $V$  ранее посчитанными значениями. Для многих сред это значительно ускорит алгоритм.

Замечание 2:

Полученная политика будет оптимальной.



# Value Iteration

Алгоритм:

1. Инициализируем вектор значений функции  $V$
2. В течении  $T$  шагов:
  - 2.1. Для каждого состояния среды  $s$ :
    - 2.1.1.  $V(s) = \max_a \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')} [r + \gamma V_\pi(s')]$

Замечание 1:

Политики в явном виде нигде нет. Ее можно получить с помощью  $\operatorname{argmax}$

Замечание 2:

Обычно мы не хотим считать мат. ожидание честно, поэтому давайте от него избавимся

# Value Iteration

Алгоритм:

1. Инициализируем вектор значений функции  $V$
2. В течении  $T$  шагов:

2.1. Для каждого состояния среды  $s$ :

$$2.1.1. \underbrace{V(s)}_{\text{Learning rate}} = \underbrace{(1 - \alpha)}_{\text{Learning rate}} \cdot \underbrace{V(s)}_{\text{Learning rate}} + \underbrace{\alpha}_{\text{Learning rate}} \max_a \left[ \underbrace{R(s, a)}_{\text{Здесь используются семплы из}} + \underbrace{\gamma V_{\pi}(T(s, a))}_{\text{распределения}} \right]$$

Learning rate

Здесь используются семплы из  
распределения

Замечание 1:

Политики в явном виде нигде нет. Ее можно получить с помощью  $\text{argmax}$

Замечание 2:

Теперь нет никакого мат. ожидания. В случае стохастической среды приближение может быть не точным, но алгоритм работает значительно быстрее.

# Ограничения

Алгоритм:

1. Инициализируем вектор значений функции  $V$
2. В течении  $T$  шагов:

2.1. Для каждого состояния среды  $s$ :

$$2.1.1. \underbrace{V(s)}_{\text{Learning rate}} = \underbrace{(1 - \alpha)}_{\text{Learning rate}} \cdot \underbrace{V(s)}_{\text{Learning rate}} + \underbrace{\alpha}_{\text{Learning rate}} \max_a \left[ \underbrace{R(s, a, T(s, a))}_{\text{Здесь используются семплы из распределения}} + \underbrace{\gamma V_{\pi}(T(s, a))}_{\text{Здесь используются семплы из распределения}} \right]$$

Learning rate

Здесь используются семплы из  
распределения

Ограничения:

1. Должны знать  $T$  и  $R$
2. Должны уметь перебирать все состояния
3. Должны уметь перебирать все действия

# Перерыв

# Value Iteration

Алгоритм:

1. Инициализируем вектор значений функции  $V$
2. В течении  $T$  шагов:
  - 2.1. Для каждого состояния среды  $s$ :
    - 2.1.1. 
$$V(s) = (1 - \alpha) \cdot V(s) + \alpha \max_a [R(s, a, T(s, a)) + \gamma V_\pi(T(s, a))]$$

Ограничения:

1. Должны знать  $T$  и  $R$
2. Должны уметь перебирать все состояния
3. Должны уметь перебирать все действия

# First-visit Monte Carlo method

Дана политика  $\pi$ . Хотим посчитать value function для нее. Динамика среды не известна

Алгоритм:

1. Инициализируем вектор  $\mathbf{V}$
2. Инициализируем вектор массивов  $\mathbf{G}$  пустыми массивами
3. В течении  $\mathbf{N}$  итераций:
  - 3.1. Получаем эпизод  $\tau$  следуя политике  $\pi$
  - 3.2. Для каждого состояния  $\mathbf{s}$  в  $\tau$ :
    - 3.2.1. Добавить в  $\mathbf{G}[\mathbf{s}]$  суммарную дисконтированную награду для этого состояния, полученную начиная с первого посещения этого состояния
    - 3.2.2.  $\mathbf{V}[\mathbf{s}] = \text{mean}(\mathbf{G}[\mathbf{s}])$



# Value Function

**Policy Value Function** определяет то, какую награду получит агент, который находится в заданном состоянии

$$V_{\pi}(s) = \mathbb{E}_{a \sim \pi(s)} \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')} [r + \gamma V_{\pi}(s')]$$

Действие политики

Ожидаемая суммарная награда

Реакция среды на действие

**Value Function** определяет то, насколько большую суммарную награду мы можем получить если находимся в определенном состоянии

$$V(s) = V_{\pi^*}(s) = \max_a \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')} [r + \gamma \cdot V_{\pi^*}(s')]$$

А оптимальную стратегию можно записать как:

$$\pi^*(s) = \arg \max_a \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')} [r + \gamma \cdot V_{\pi^*}(s')]$$

# Quality Function

**Policy Quality Function** определяет то, какую награду получит агент, который находится в заданном состоянии **при совершении определенного действия**

$$Q_{\pi}(s, a) = \mathbb{E}_{\substack{s' \sim T(s, a), \\ r \sim R(s, a, s')}} [r + \gamma \mathbb{E}_{a' \sim \pi(s')} Q_{\pi}(s', a')]$$

Реакция среды на действие

Ожидаемая суммарная награда

Действие политики

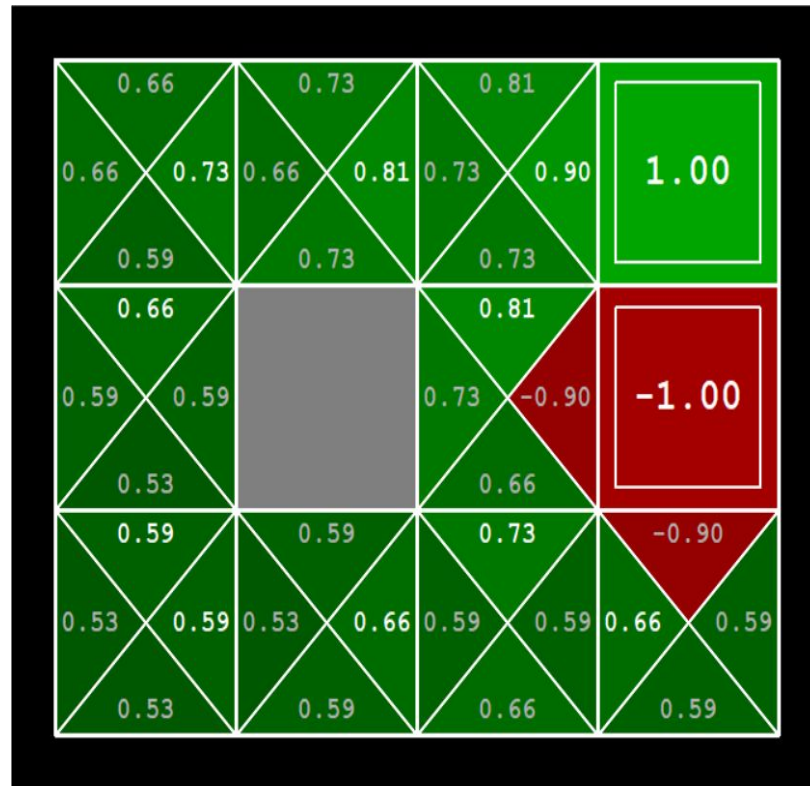
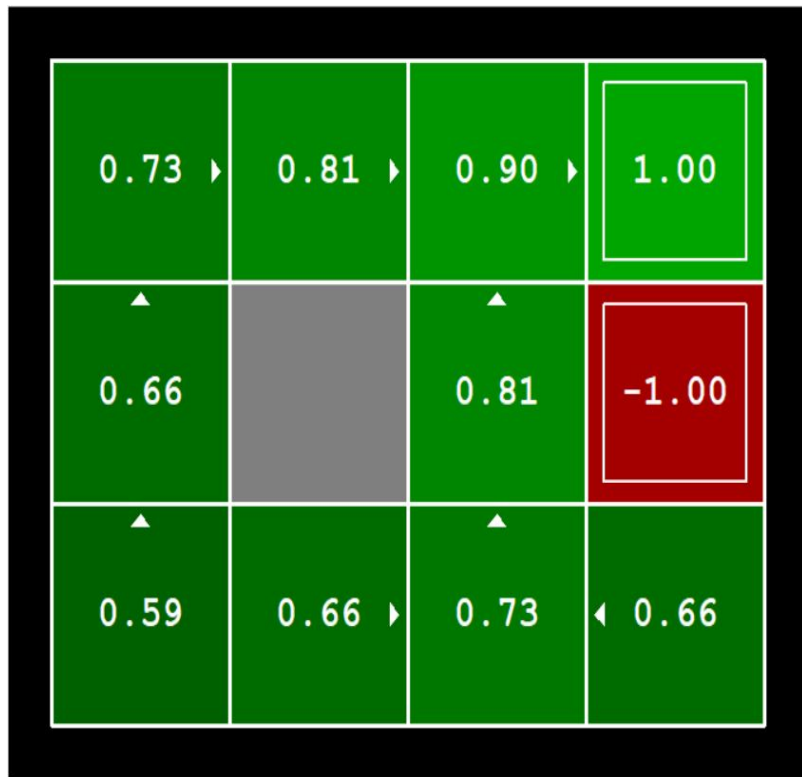
**Quality Function** определяет то, насколько большую суммарную награду мы можем получить если находимся в определенном состоянии **совершив действие**

$$Q(s, a) = Q_{\pi^*}(s, a) = \mathbb{E}_{s' \sim T(s, a), r \sim R(s, a, s')} [r + \gamma \max_{a'} Q(s', a')]$$

А оптимальную стратегию можно записать как:

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

# Q-Function



# First-visit Monte Carlo method

Дана политика  $\pi$ . Хотим посчитать value function для нее. Динамика среды не известна

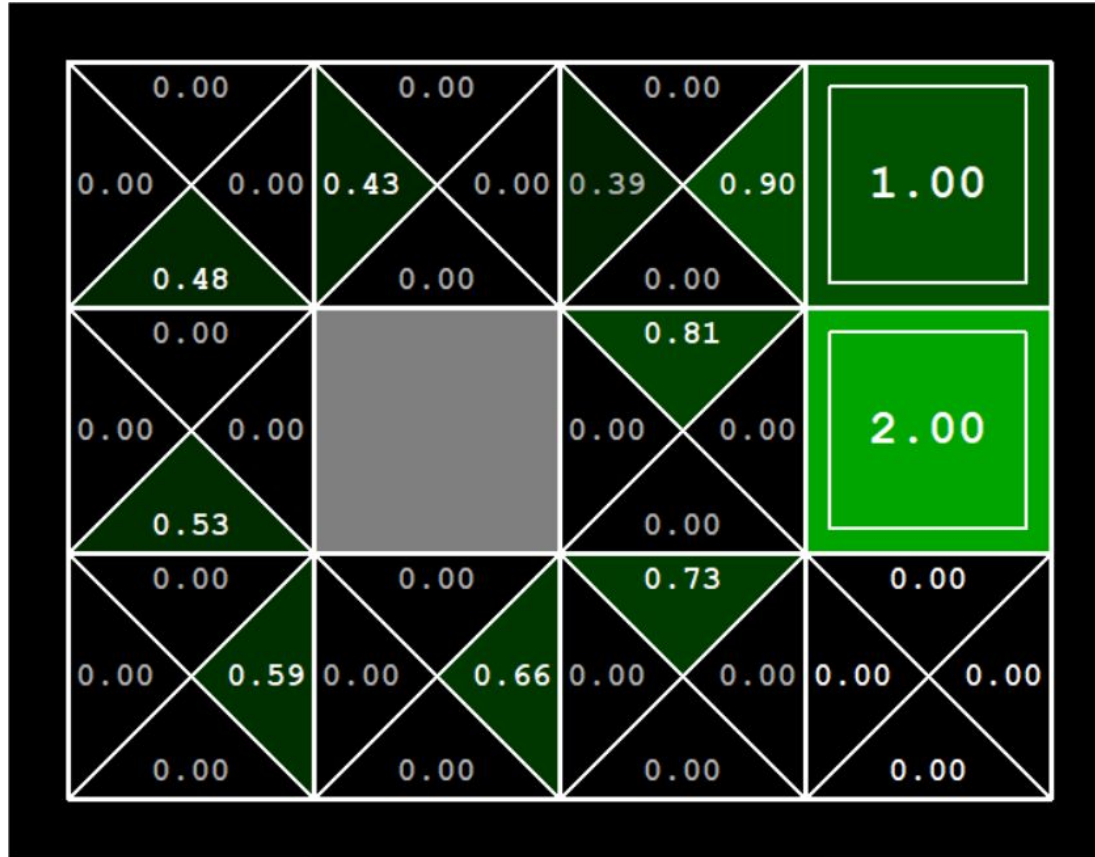
Алгоритм:

1. Инициализируем вектор  $\mathbf{Q}$
2. Инициализируем вектор массивов  $\mathbf{G}$  пустыми массивами
3. В течении  $\mathbf{N}$  итераций:
  - 3.1. Получаем эпизод  $\tau$  следуя политике  $\pi$
  - 3.2. Для каждого состояния  $\mathbf{s}$  в  $\tau$ :
    - 3.2.1. Добавить в  $\mathbf{G}[\mathbf{s}, \mathbf{a}]$  суммарную дисконтированную награду для этого состояния, полученную начиная с первого посещения этого состояния
    - 3.2.2.  $\mathbf{Q}[\mathbf{s}, \mathbf{a}] = \text{mean}(\mathbf{G}[\mathbf{s}, \mathbf{a}])$

# Есть ли проблема?

0.00	0.00	0.00	1.00
0.00		0.00	2.00
0.00	0.00	0.00	0.00

Для большинства пар нет оценки



# First-visit Monte Carlo method

Дана политика  $\pi$ . Хотим посчитать value function для нее. Динамика среды не известна

Алгоритм:

1. Инициализируем вектор  $\mathbf{Q}$
2. Инициализируем вектор массивов  $\mathbf{G}$  пустыми массивами
3. В течении  $\mathbf{N}$  итераций:
  - 3.1. Получаем эпизод  $\tau$  следуя политике  $\pi$
  - 3.2. Для каждой пары  $(\mathbf{s}, \mathbf{a})$  в  $\tau$ :
    - 3.2.1. Добавить в  $\mathbf{G}[\mathbf{s}, \mathbf{a}]$  суммарную дисконтированную награду для этого состояния, полученную начиная с первого посещения этого состояния
    - 3.2.2.  $\mathbf{Q}[\mathbf{s}, \mathbf{a}] = \text{mean}(\mathbf{G}[\mathbf{s}, \mathbf{a}])$

Предполагаем, что можем начать эпизод с любой парой  $(\mathbf{s}_0, \mathbf{a}_0)$  с вероятностью больше нуля, либо считаем, что политика имеет ненулевую вероятность достичь любой пары  $(\mathbf{s}, \mathbf{a})$  (например, является epsilon-greedy)

# Monte Carlo Control

Алгоритм:

1. Инициализируем вектор  $\mathbf{Q}$
2. Инициализируем вектор массивов  $\mathbf{G}$  пустыми массивами
3. В течении  $\mathbf{N}$  итераций:
  - 3.1. Получаем эпизод  $\tau$  следуя политике  $\pi$
  - 3.2. Для каждой пары  $(\mathbf{s}, \mathbf{a})$  в  $\tau$ :
    - 3.2.1. Добавить в  $\mathbf{G}[\mathbf{s}, \mathbf{a}]$  суммарную дисконтированную награду для этого состояния, полученную начиная с первого посещения этого состояния
    - 3.2.2.  $\mathbf{Q}[\mathbf{s}, \mathbf{a}] = \text{mean}(\mathbf{G}[\mathbf{s}, \mathbf{a}])$
  - 3.3. Для каждого состояния  $\mathbf{s}$  в  $\tau$ :
    - 3.3.1. Обновить “жадное” действие в политике  $\pi$

Замечание 1: Используем старые награды. Иногда так делать плохо

Замечание 2: Оценка Q-функции учитывает то, что наша политика epsilon-greedy



# Q-learning

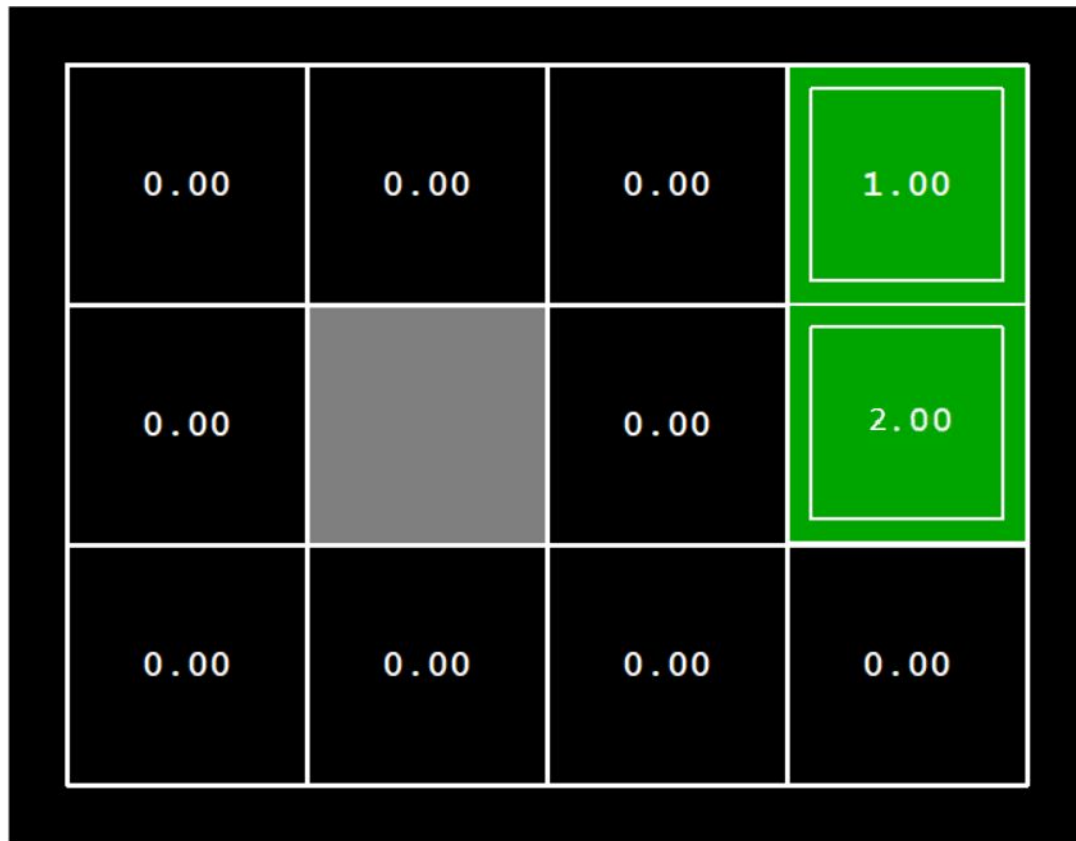
Алгоритм:

1. Инициализируем  $Q(s, a)$
2. Инициализируем среду, получаем начальное состояние  $s$
3. В течении  $T$  шагов:
  - 3.1.  $a := \operatorname{argmax} Q(s, .)$
  - 3.2. Совершаем действие  $a$ , получаем  $s', r, d$  из среды
  - 3.3.  $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \gamma \cdot \max_{a'} Q(s', a'))$
  - 3.4.  $s := s'$  если не  $d$ , иначе реинициализируем среду

Рассмотрим 3.3 подробнее:

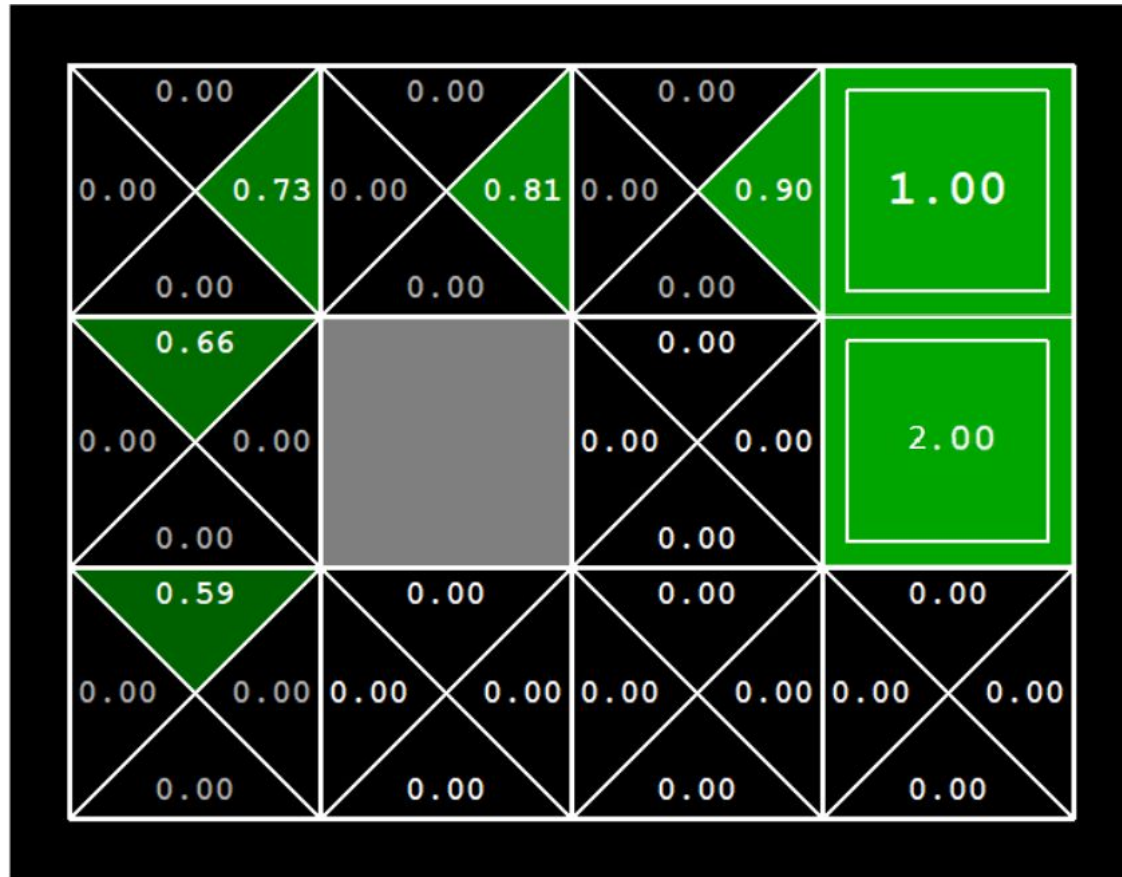
$$Q(s, a) = \underbrace{(1 - \alpha)}_{\text{Learning rate}} Q(s, a) + \underbrace{\alpha}_{\text{Получено из среды, знать R не нужно!}} (r + \gamma \cdot \max_{a'} Q(s', a'))$$

# Есть ли в Q-learning проблема?



0.00	0.00	0.00	1.00
0.00		0.00	2.00
0.00	0.00	0.00	0.00

# Снова не исследуем среду



# Epsilon-greedy Q-learning

Алгоритм:

1. Инициализируем  $\mathbf{Q}(\mathbf{s}, \mathbf{a})$
2. Инициализируем среду, получаем начальное состояние  $\mathbf{s}$
3. В течении  $\mathbf{T}$  шагов:
  - 3.1.  $\mathbf{a} := \operatorname{argmax} \mathbf{Q}(\mathbf{s}, \cdot)$  с вероятностью  $1-\epsilon$ , иначе  $\mathbf{a} := \operatorname{random}(\mathbf{A})$
  - 3.2. Совершаем действие  $\mathbf{a}$ , получаем  $\mathbf{s}', \mathbf{r}, \mathbf{d}$  из среды
  - 3.3.  $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \gamma \cdot \max_{a'} Q(s', a'))$
  - 3.4.  $\mathbf{s} := \mathbf{s}'$  если не  $\mathbf{d}$ , иначе реинициализируем среду

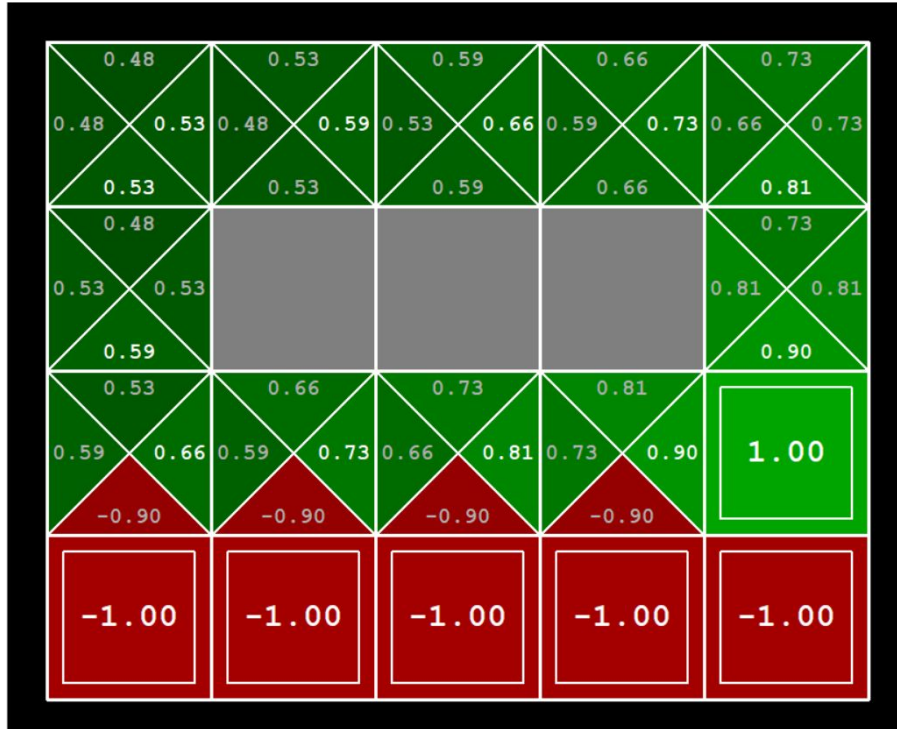
Замечание 1: Оценка  $Q$  не учитывает то, что политика epsilon-greedy

# Cliff World



# Cliff World

Что выучит Q-learning:



Оптимальная epsilon-greedy политика:



# SARSA

Алгоритм:

1. Инициализируем  $Q(s, a)$
2. Инициализируем среду, получаем начальное состояние  $s$
3. В течении  $T$  шагов:
  - 3.1.  $a := \operatorname{argmax} Q(s, \cdot)$  с вероятностью  $1 - \epsilon$ , иначе  $a := \operatorname{random}(A)$
  - 3.2. Совершаем действие  $a$ , получаем  $s', r, d$  из среды
  - 3.3.  $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \gamma \cdot Q(s', a'))$ , т.е. смотрим на реальное следующее действие
  - 3.4.  $s := s'$  если не  $d$ , иначе реинициализируем среду

# On-policy Off-policy

## On-policy

- При обучении агент оценивает реальную суммарную награду
- После обучения агент использует ту же политику, что и во время обучения
- Во время не использует дополнительных методов среды, весь exploration является частью агента

Примеры: **SARSA, MC Control**

$$r + \gamma \cdot Q(s', a')$$

## Off-policy

- При обучении агент оценивает суммарную награду для оптимальной (жадной) политики
- После обучения агент использует жадную политику, убирая дополнительные exploration методы
- Во время обучения использует дополнительные методы исследования среды

Пример: **Q-learning**

$$r + \gamma \cdot \max_{a'} Q(s', a')$$



# Ограничения

- ~~1) Должны знать  $T$  и  $R$~~
- 2) Дискретное пространство состояний
- 3) Дискретное пространство действий

Пусть все оценки храним во float. Размер 4 байта. Пусть есть 16 Гб оперативной памяти. Сколько всего можем хранить пар состояние-действие?

# Ограничения

- 1) ~~Должны знать T и R~~
- 2) Дискретное пространство состояний
- 3) Дискретное пространство действий

Пусть все оценки храним во float. Размер 4 байта. Пусть есть 16 Гб оперативной памяти. Сколько всего можем хранить пар состояние-действие?

$$\frac{16 \cdot 2^{30}}{4} = 4 \cdot 2^{30} \approx 4 \cdot 10^9$$

Сколько всего возможных позиций в игре в шахматы?



# Ограничения

- 1) ~~Должны знать T и R~~
- 2) Дискретное пространство состояний
- 3) Дискретное пространство действий

Пусть все оценки храним во float. Размер 4 байта. Пусть есть 16 Гб оперативной памяти. Сколько всего можем хранить пар состояние-действие?

$$\frac{16 \cdot 2^{30}}{4} = 4 \cdot 2^{30} \approx 4 \cdot 10^9$$

Сколько всего возможных позиций в игре в шахматы?

Оценка сверху (число Шеннона):  $10^{43}$

После первых 10 ходов (точное число):  $69352859712417 \approx 69 \cdot 10^{12}$

