# Методы машинного обучения

ИУ-5, магистратура, 2 семестр,
весна  2023 года

# Введение в обучение с подкреплением (reinforcement learning – RL)

# Книги

- [1] Саттон Р. С., Барто Э. Дж. Обучение с подкреплением: Введение. 2-е изд. / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 552 с.
- Основополагающий учебник по обучению с подкреплением. Рассмотрена история развития обучения с подкреплением и основные методы.


Издание 2014 года

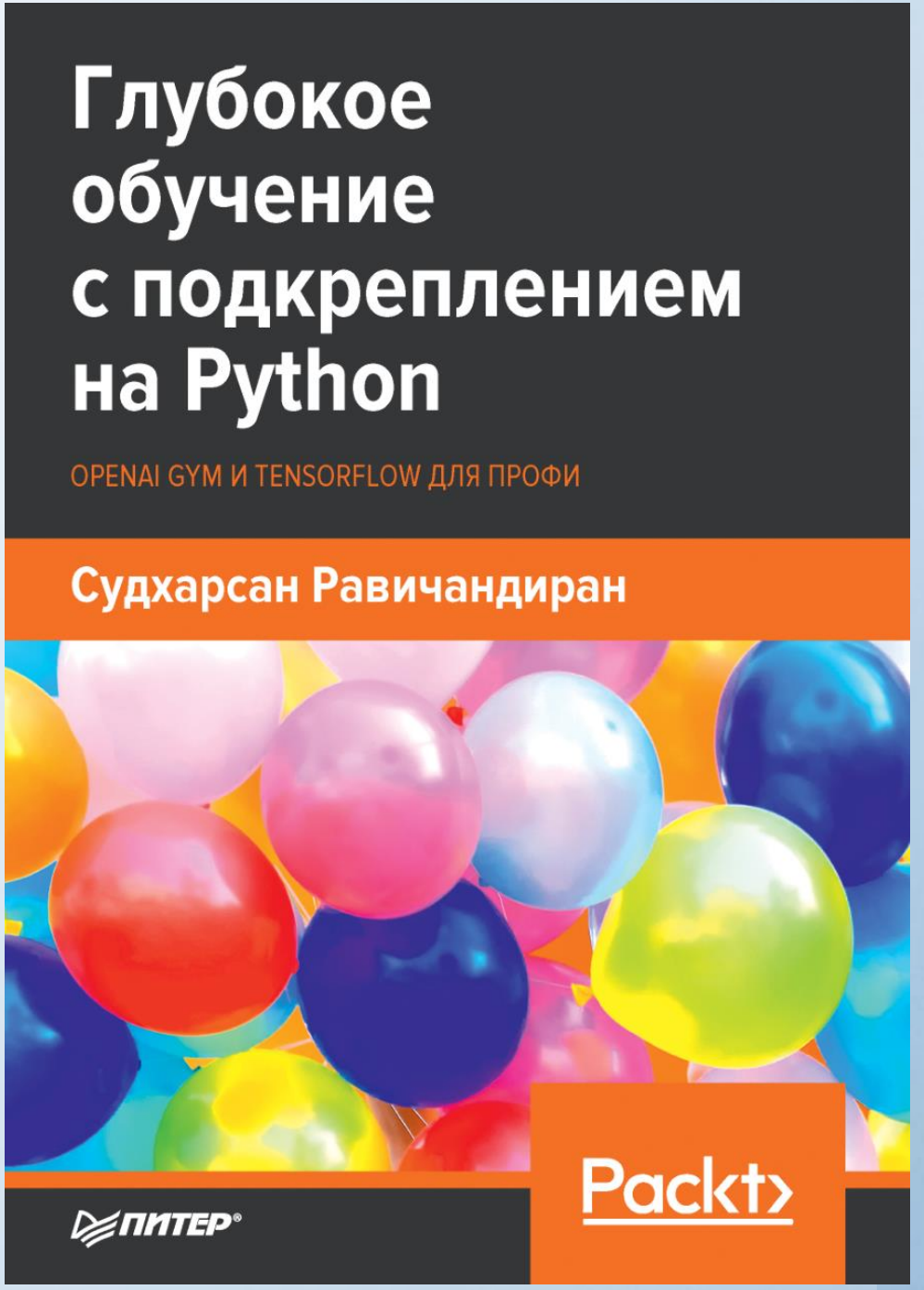

Издание 2020 года

# Книги

- [2] Лапань Максим. Глубокое обучение с подкреплением. AlphaGo и другие технологии. — СПб.: Питер, 2020. — 496 с.
- Подробно рассмотрены теоретические основы, большое количество примеров.

# Книги


- [3] Равичандиран Судхарсан. Глубокое обучение с подкреплением на Python. OpenAI Gym и TensorFlow для профи. — СПб.: Питер, 2020. — 320 с.

# Книги

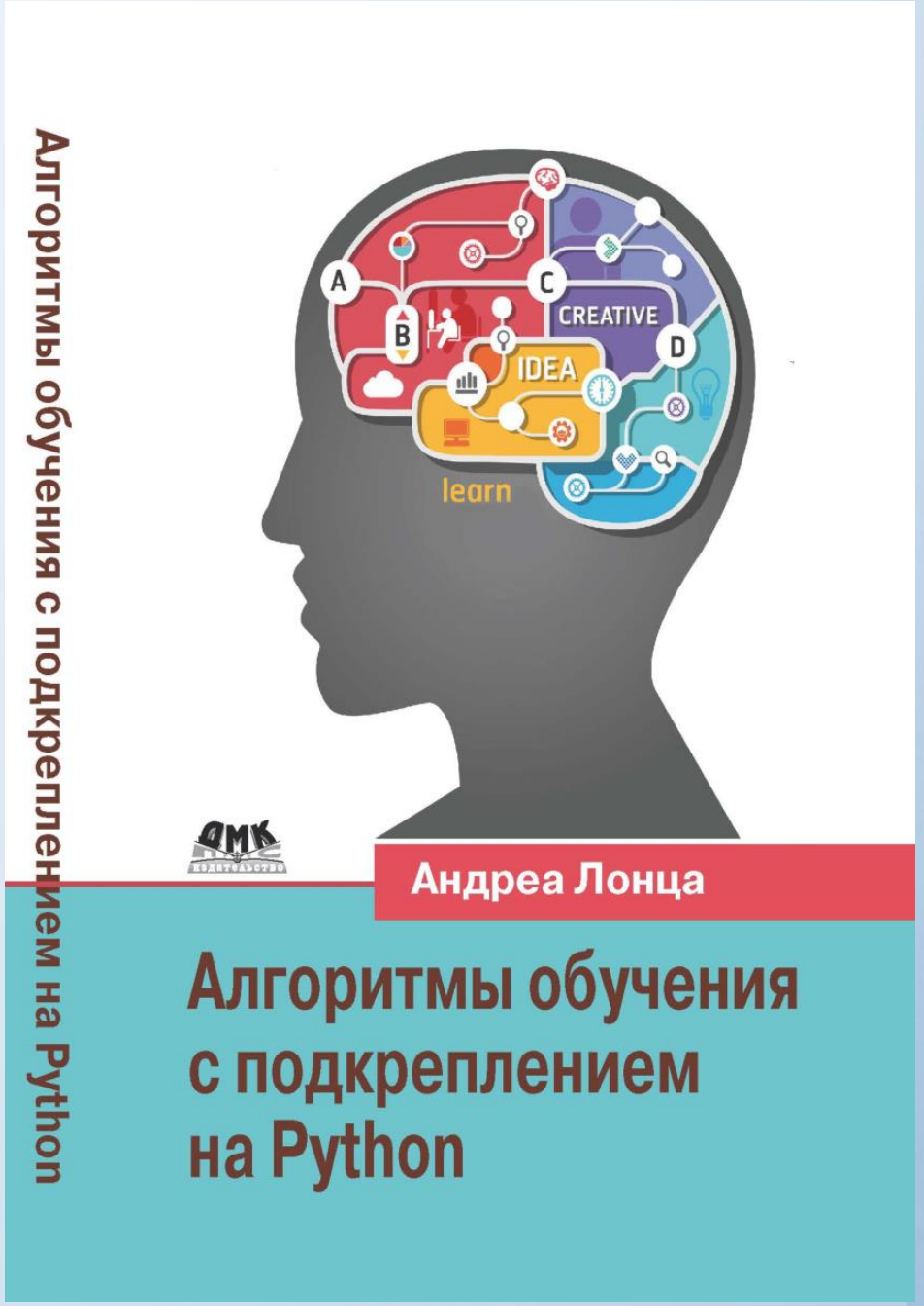
- [4] Грессер Лаура, Кенг Ван Лун. Глубокое обучение с подкреплением: теория и практика на языке Python. — СПб.:Питер, 2022. — 416 с.

# Книги

- [5] Алфимцев А.Н. Мультиагентное обучение с подкреплением: учебное пособие - Москва: Изд-во МГТУ им. Н.Э.Баумана, 2021 – 222 с.

# Книги

- [6] Лонца А. Алгоритмы обучения с подкреплением на Python / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 286 с.



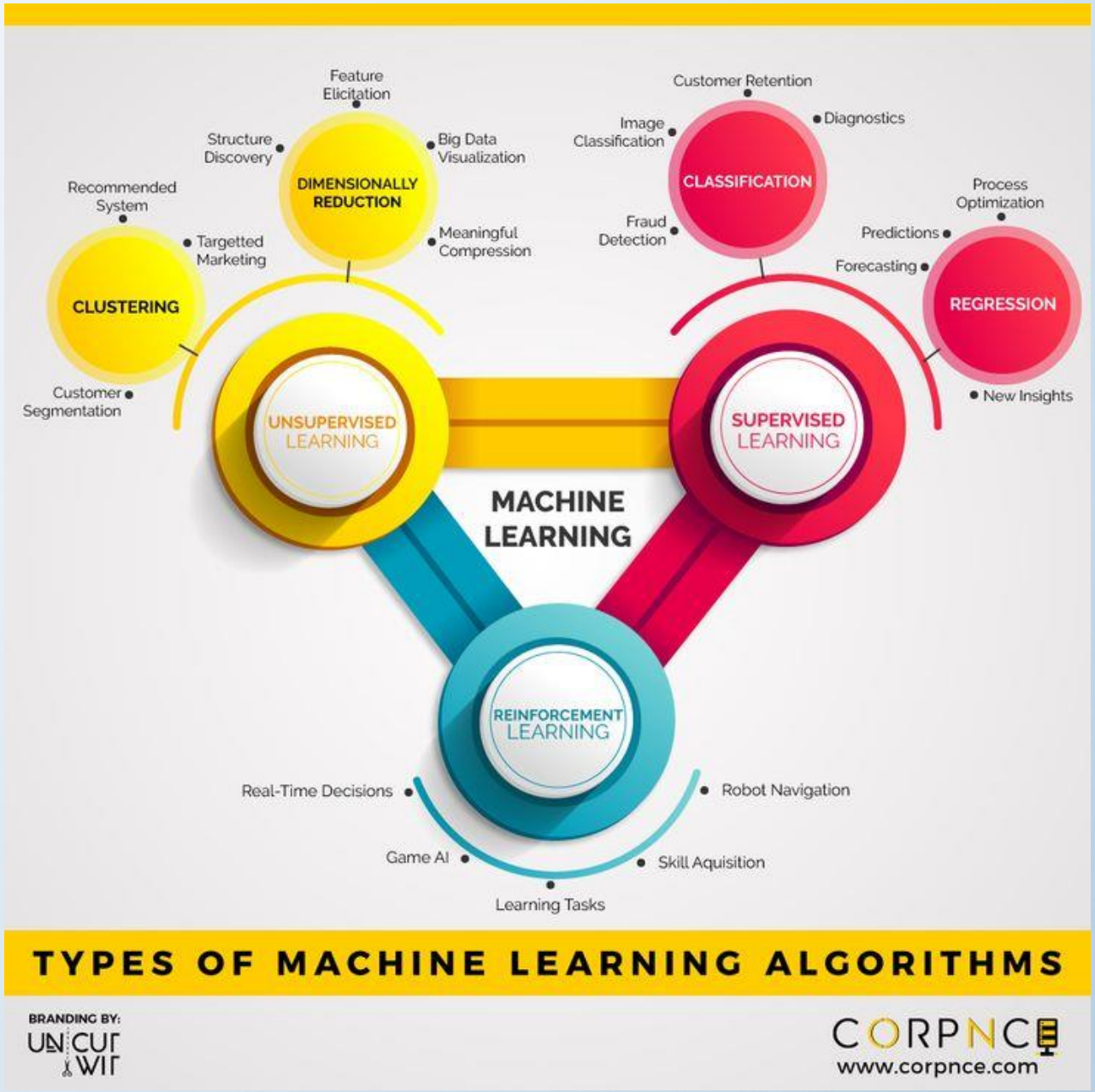
# Книги

- [7] Лю Ю. Обучение с подкреплением на PyTorch: сборник рецептов / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 282 с.

# Основные концепции RL

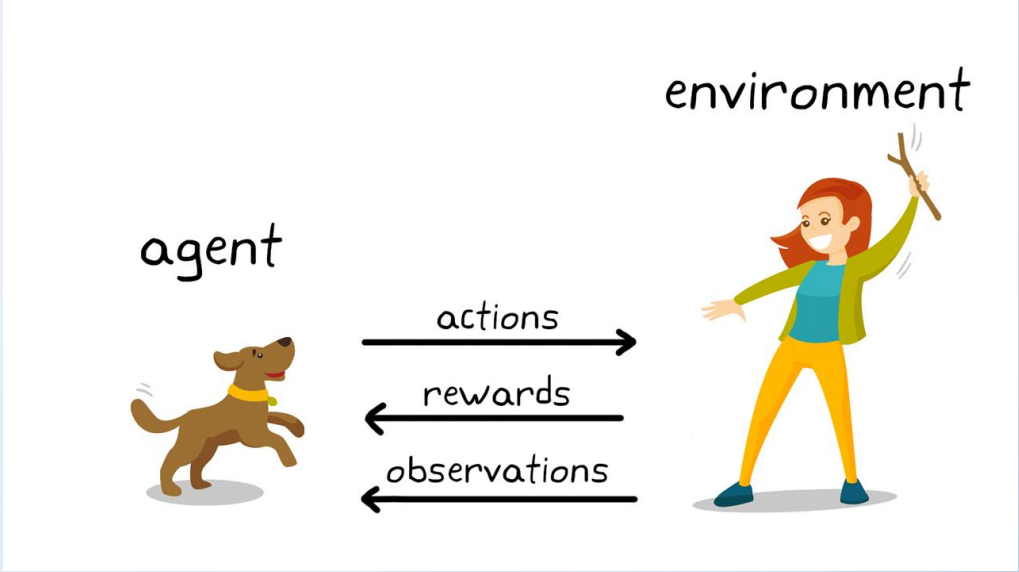
# Типы («Классификация») задач ML

- Обучение с учителем (supervised learning)
  - Классификация
  - Регрессия
  - Прогнозирование временных рядов
- Обучение без учителя (unsupervised learning)
  - Кластеризация
  - Методы понижения размерности
- Обучение с подкреплением (reinforcement learning)

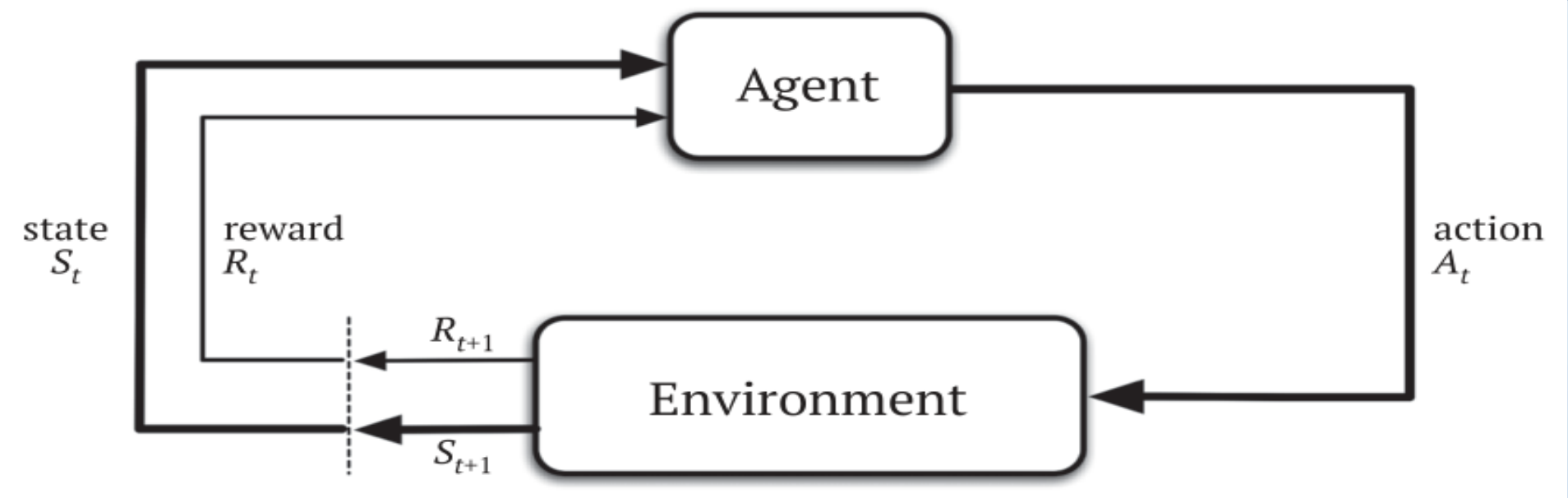


# Типичный алгоритм RL неформально [3]

1. Агент взаимодействует со средой, выполняя действие.
2. Агент выполняет действие и переходит из одного состояния в другое.
3. Агент получает награду на основании выполненного действия.
4. В зависимости от награды агент понимает, было действие хорошим или плохим.
5. Если действие было хорошим, то есть если агент получил положительную награду, то агент предпочитает выполнить это действие еще раз.
6. В противном случае агент пытается выполнить другое действие, приводящее к положительной награде. Таким образом, по сути, происходит процесс обучения методом проб и ошибок.



# Типичный алгоритм RL формально [1]




**Взаимодействие между агентом и окружающей средой в марковском процессе принятия решений.**

- Агент и среда взаимодействуют на каждом шаге дискретной последовательности временных шагов, t = 0, 1, 2, 3, …
- На каждом временном шаге t агент получает некоторое представление состояния окружающей среды $S_t$ и, исходя из него, выбирает действие $A_t$.
- На следующем шаге агент в качестве последствия своего действия получает числовое вознаграждение $R_{t+1}$ и оказывается в новом состоянии $S_{t+1}$.
- В результате порождается траектория $S_0 \rightarrow A_0 \mid R_1, S_1 \rightarrow A_1 \mid R_2, S_2 \rightarrow A_2 \ldots$

# Основные понятия RL - 1 [3]

- **Агент (agent)** — программный модуль, способный принимать осмысленные решения, играет роль обучаемого в RL. Агенты выполняют действия, контактируя со средой, и получают награды в зависимости от своих действий. Сумму наград, полученных агентом от среды, принято называть **«возвратом»**.
- **Политика (policy)** (называемая также **«стратегией»**) определяет поведение агента в среде, направленное на достижение какой-либо цели. Способ выбора агентом действия, которое он будет выполнять, зависит от политики.
  - Допустим, агенту необходимо добраться из пункта А в пункт Б. Существует множество возможных вариантов маршрута; одни пути короткие, другие длинные. Эти пути называются «политиками», потому что они представляют собой способ выполнения действия для достижения цели. Политика часто обозначается символом $\pi$ и может быть реализована таблицей соответствия или сложным процессом поиска.

# Основные понятия RL - 2 [3]

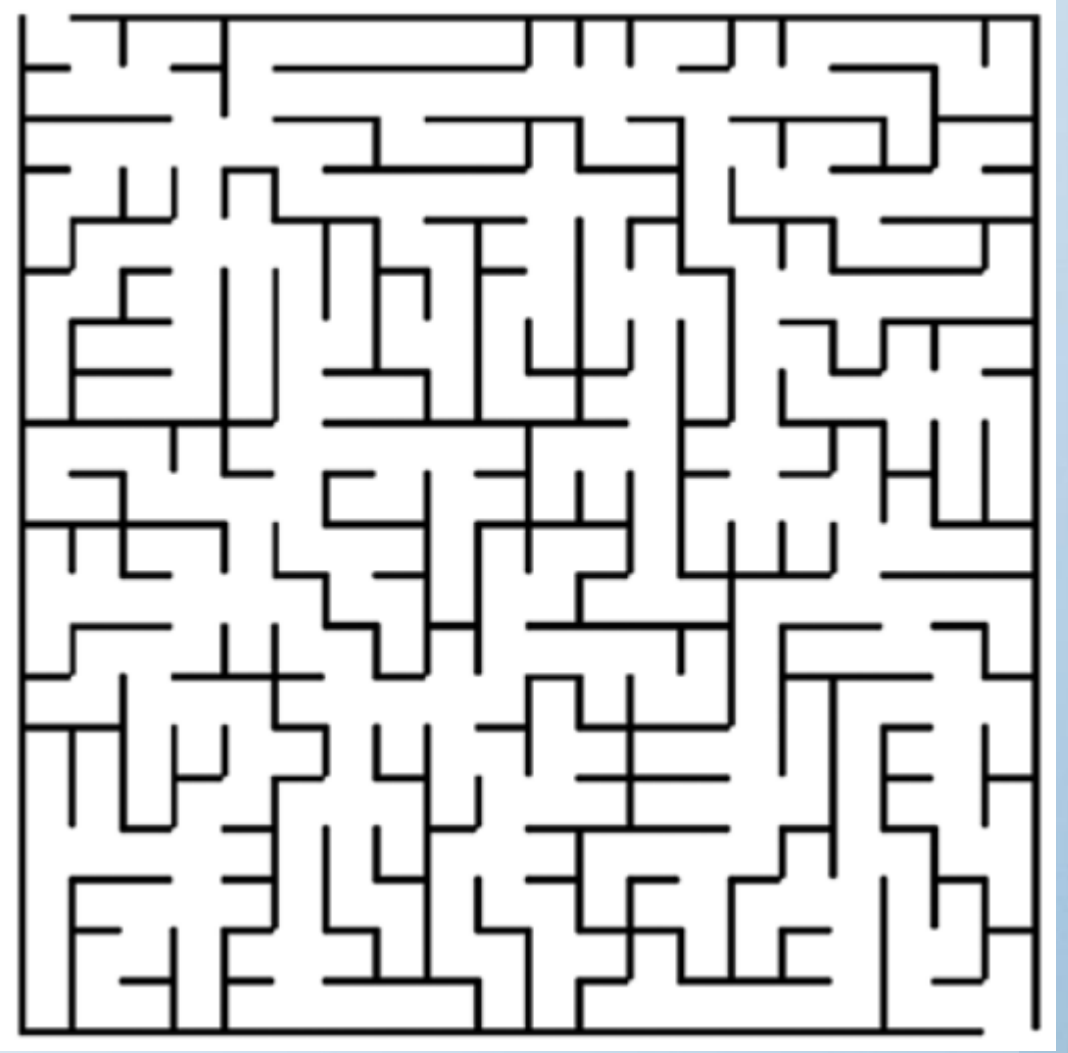
- **Функция ценности (value function)** определяет, насколько «хорошо» для агента А пребывание в конкретном состоянии. Она зависит от политики и часто обозначается v(s). Ее значение равно ожидаемому суммарному результату, получаемому агентом, начиная с исходного состояния.
  - Функций ценности может быть несколько, «оптимальной функцией ценности» $v_*(s)$ называется та, которая обеспечивает наибольшую ценность по всем состояниям по сравнению с другими функциями ценности. Аналогичным образом «оптимальной политикой» называется политика, имеющая оптимальную функцию ценности.
- **Модель (model)** является представлением среды с точки зрения агента. Обучение делится на два типа: с моделью и без модели. В обучении с моделью агент эксплуатирует ранее усвоенную информацию для выполнения задачи, а при обучении без модели агент просто полагается на метод проб и ошибок для выполнения правильного действия.
  - Допустим, агенту необходимо добраться из пункта А в пункт Б. В обучении с моделью агент использует имеющуюся у него карту, а в обучении без модели агент методом проб и ошибок пробует разные пути и выбираете самый быстрый.
  - Если в качестве модели используется глубокая нейронная сеть, то говорят о **глубоком обучении с подкреплением.**

# Основные понятия RL - 3 [1]

- **Компромисс между исследованием и использованием. Дилемма исследование-использование. Exploration-Exploitation dilemma.**
- Чтобы получить большое вознаграждение, обучающийся с подкреплением агент должен предпочитать действия, которые были испробованы в прошлом и принесли вознаграждение. Агента, который всегда выбирает наиболее выгодный на текущий момент действия принято называть «жадным».
- Но чтобы найти такие действия, он должен пробовать действия, которые раньше не выбирал. Агент должен использовать уже приобретенный опыт, чтобы получить вознаграждение, но должен продолжать исследования, чтобы выбирать более эффективные действия в будущем.
- Дилемма состоит в том, что одного лишь исследования или использования недостаточно для успешного решения задачи. Агент должен пробовать разные действия и неуклонно отдавать предпочтение тем, которые кажутся наилучшими.
- Не существует однозначно эффективного баланса между исследованием и использованием. Все зависит от длительности сессии, особенностей среды и других факторов.
    1. В RL агент должен действовать, невзирая на значительную неопределенность окружающей среды. Как правило, агент начинает с «исследовательских» ходов, чтобы получить первичную информацию о среде.
    2. В случае короткой сессии более выигрышными являются «жадные» стратегии.
    3. В случае длинной сессии более выигрышными являются стратегии со значительной «исследовательской» составляющей.

# Пример задачи RL - лабиринт [3]

- Цель игры — добраться до выхода и не заблудиться в лабиринте.
- Агент — тот, кто перемещается по лабиринту.
- Среда — лабиринт.
- Состояние — позиция в лабиринте, на которой в данный момент находится агент.
- Агент выполняет действие, перемещаясь из одного состояния в другое.
- Агент получает положительный результат, если при выполнении действия он не наталкивается на препятствие, и отрицательный результат, если при своем действии он наталкивается на препятствие и не может добраться до конечной точки.



# Типы сред в RL - 1 [3]

- **Детерминированные и стохастические среды.**
- Среда называется **детерминированной**, если последствия действий полностью известны по текущему состоянию. Например, в шахматной партии известен точный результат хода каждой фигурой.
- Среда называется **стохастической**, если результат не может быть определен по текущему состоянию из-за повышенной неопределенности. Пример подбрасывание кубика 1-6.

# Типы сред в RL - 2 [3]

- **Среды с полной и неполной информацией.**
- В среде **с полной информацией** агент может определить состояние системы в любой момент времени. Например, в шахматной партии состояние системы (то есть положение всех фигур на доске) известно в любой момент времени, так что игрок может принять оптимальное решение.
- В среде **с неполной информацией** агент не может определить состояние системы в любой момент времени. (Типичный пример – игра в карты, когда карты противника неизвестны)
  - Среды с неполной информаций в настоящий момент особенно активно изучаются в RL. Большинство практических задач связанно именно с такими средами.
  - Пример задачи – подбор параметра станка по частично известным характеристикам брака детали. Фактически, в этом случае мы используем обучение на основе датасета (как в обучении с учителем).

# Типы сред в RL - 3 [3]

- **Дискретные и непрерывные среды.**
- Если существует конечный набор доступных действий для перехода из одного состояния в другое, среда называется **дискретной**. Например, в шахматной партии набор ходов конечен.
- Если существует бесконечный набор доступных действий для перехода из одного состояния в другое, среда называется **непрерывной**. Например мы подаем силу или скорость в качестве параметра физической модели.
- *А если определяем угловой курс корабля?*

# Типы сред в RL - 4 [3]

- **Эпизодические и неэпизодические среды.**
- **Эпизодические** среды также называются непоследовательными. В таких средах текущее действие агента не влияет на будущее действие. В эпизодической среде агент выполняет независимые задачи.
- В **неэпизодических** (или последовательных) средах, будущее действие зависит от текущего. В неэпизодической среде все действия агента связаны между собой.

# Типы сред в RL - 5 [3]

- **Одноагентные и многоагентные среды.**
- В **одноагентной** среде используется только один агент, а в **многоагентной** среде агентов несколько.
- Многоагентные среды в основном являются стохастическими, так как они обладают повышенным уровнем неопределенности.

# Марковские процессы принятия решений (МППР)

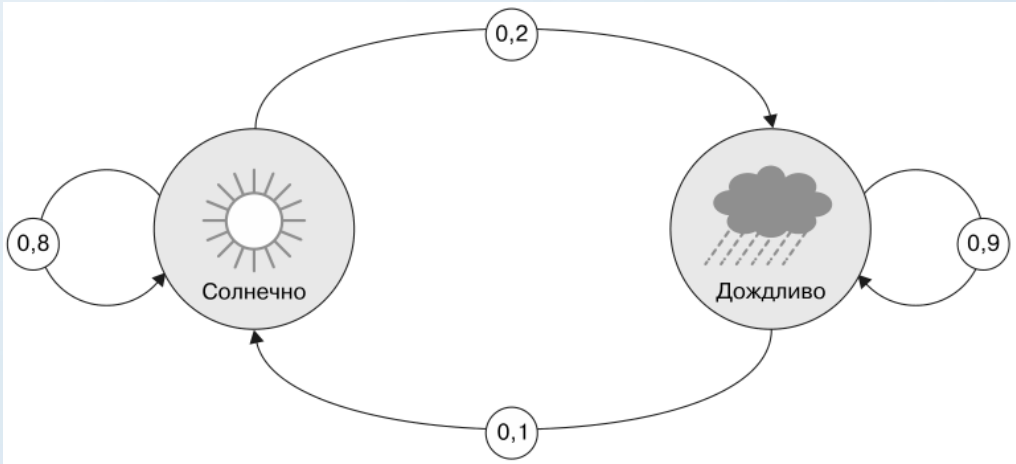
# МППР – 1 [3]

- Марковское свойство гласит, что «будущее зависит только от настоящего, но не от прошлого» («будущее связано с прошлым через настоящее»).
- Марковская цепь представляет собой вероятностную модель, которая для прогнозирования следующего состояния зависит только от текущего состояния, но не от предыдущих состояний, будущее условно независимо от прошлого. Марковская цепь соответствует марковскому свойству.
- Пример марковского процесса «пасмурно» → «дождь» (состояния до «пасмурно» не учитываются).
- Пример немарковского процесса – бросок кубика (не зависит от предыдущего состояния).

# МППР – 2 [2,3]

- Переход из одного состояния марковской цепи в другое называется «переходом», а его вероятность называется «вероятностью перехода».
- Вероятности перехода можно свести в таблицу, которую называют «марковской таблицей», а также представить в виде графа.

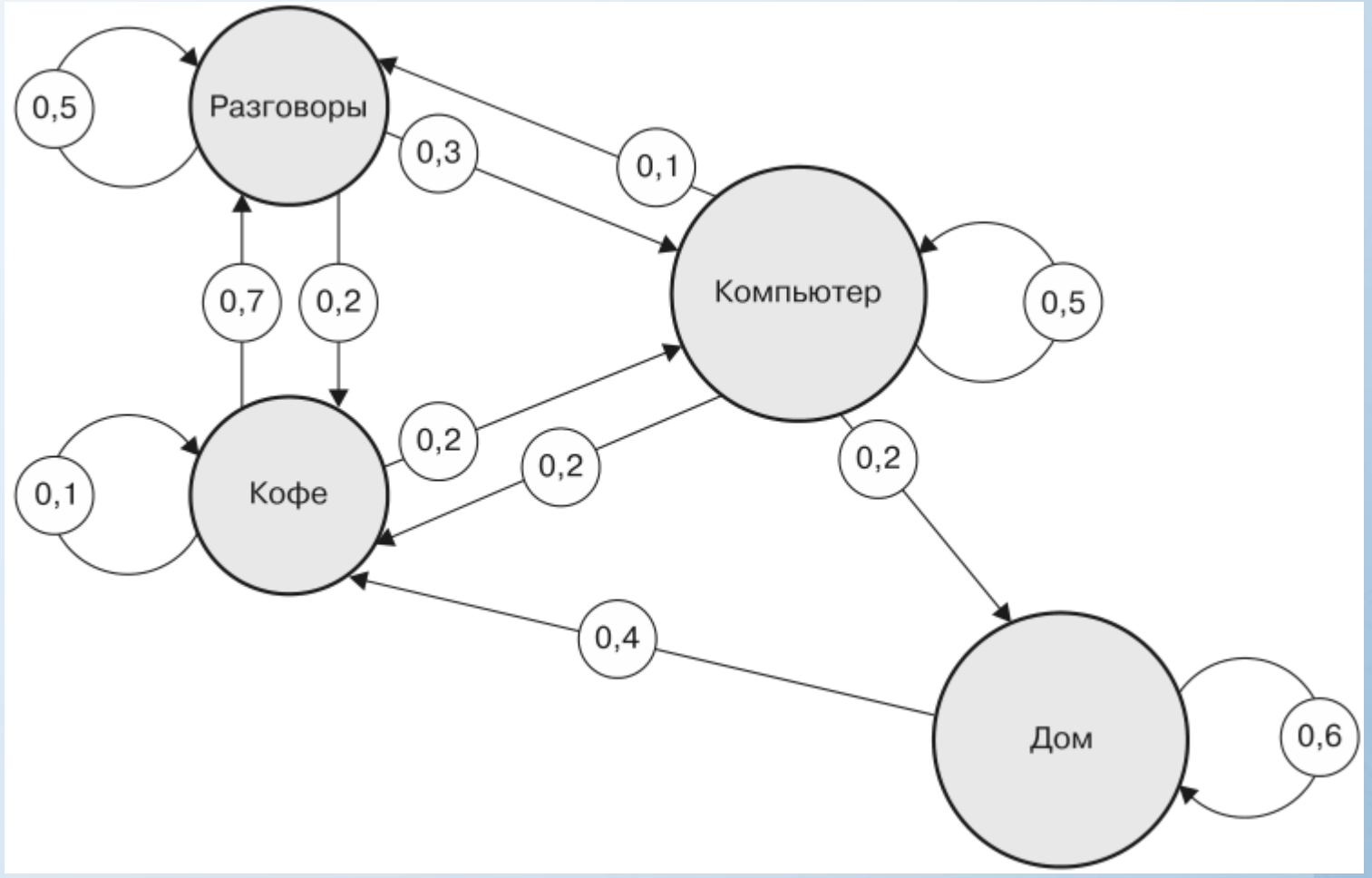
| | **Солнечно** | **Дождливо** |
|---|---|---|
| **Солнечно** | 0,8 | 0,2 |
| **Дождливо** | 0,1 | 0,9 |



# МППР – 3 [2]

- **Пример работы офисного сотрудника.**
- Считается, что его рабочий день обычно начинается из состояния «дом» и всегда с «кофе», без исключений (нет переходов «дом → компьютер» и «дом → разговоры»).
- Рабочие дни всегда заканчиваются (то есть происходит переход в состояние «дом» из состояния «компьютер»).

| | Дом | Кофе | Разговоры | Компьютер |
|---|---|---|---|---|
| **Дом** | 60 % | 40 % | 0 % | 0 % |
| **Кофе** | 0 % | 10 % | 70 % | 20 % |
| **Разговоры** | 0 % | 20 % | 50 % | 30 % |
| **Компьютер** | 20 % | 20 % | 10 % | 50 % |



# МППР – 4 [3]

Марковский процесс принятия решений (Markov Decision Process – MDP) (в некоторых источниках также называемый «марковский процесс с вознаграждением») является расширением марковских цепей. Включает следующие параметры:

1. Набор состояний (S), в которых может находиться агент.
2. Набор действий (A), которые могут выполняться агентом для перехода из одного состояния в другое.
3. Вероятность перехода $P_{ss}^{a}$, из состояния s в состояние s' посредством выполнения действия a.
4. Вероятность награды $R_{ss}^{a}$, получаемой агентом при переходе из состояния s в состояние s' посредством выполнения действия a.
5. Поправочный коэффициент $\gamma$ (гамма), управляющий соотношением важности немедленных и будущих наград.

# МППР – 5 [2,3]

- Агент пытается максимизировать сумму наград (накопленную награду), полученную от среды, а не каждую немедленную награду. Сумма наград, полученных агентом от среды, называется **«возвратом»** или **«доходом» (G – gain):**

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

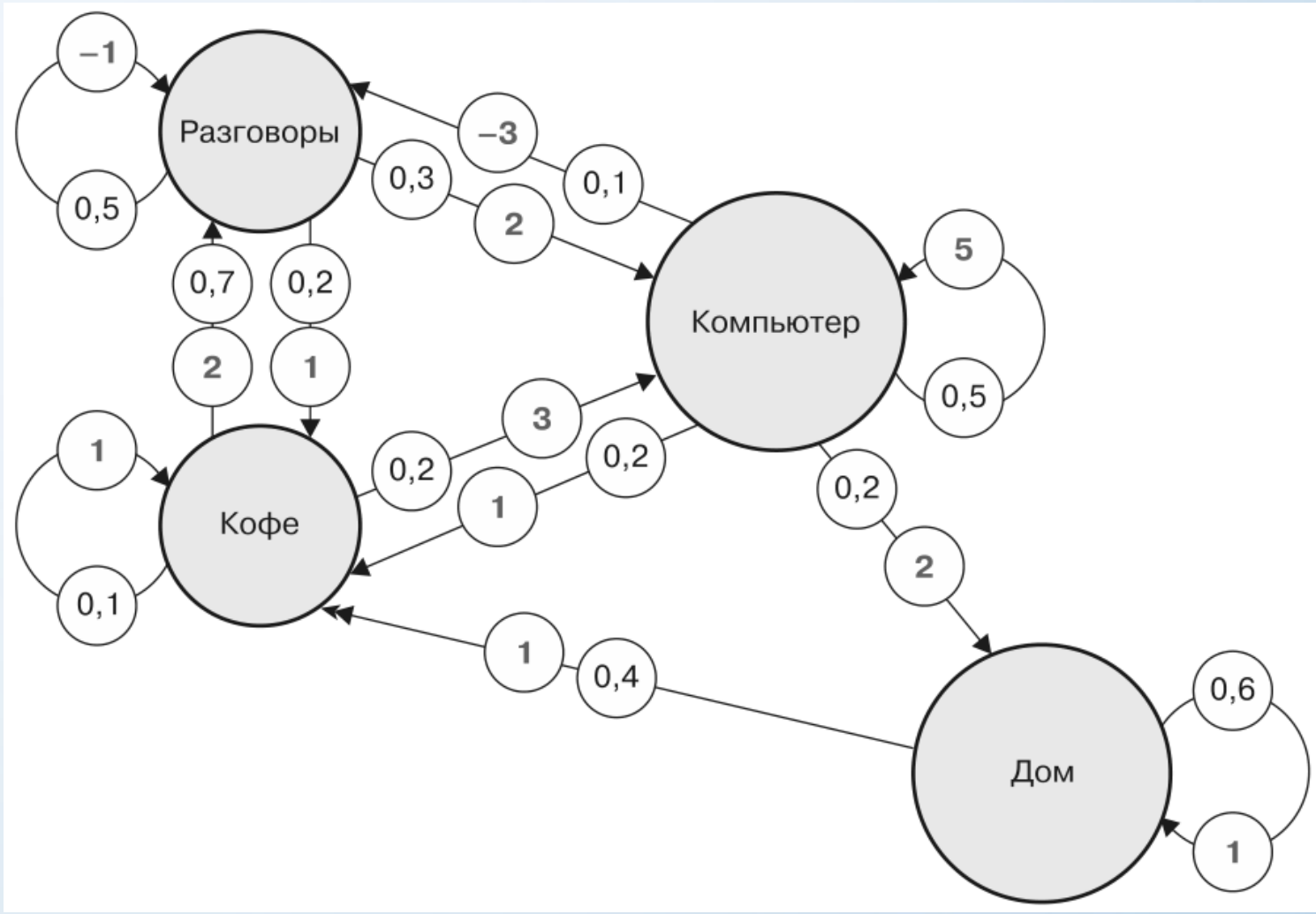
- Для каждого момента времени t вычисляется доход как сумма последующих вознаграждений, но более отдаленные из них умножаются на поправочный коэффициент $\gamma$ (коэффициент дисконтирования), возведенный в степень, равную числу шагов, на которое мы отстоим от начальной точки в момент времени t.
- Поправочный коэффициент определяет относительную важность будущих и немедленных наград. Его значение лежит в диапазоне от 0 до 1. Поправочный коэффициент 0 означает, что немедленные награды более важны, а поправочный коэффициент 1 означает, что будущие награды важнее немедленных.
- На основе дохода можно определить ценность состояния:

$$\mathrm{V(s)} = \mathbb{E}[G_t | S_t = s]$$

- Ценность состояния определяется как средняя (или ожидаемая) выгода, которую агент получает находясь в состоянии s.
- В случае $\gamma = 1$ для многих графов $\mathrm{V(s)} \to \infty$. Этот бесконечный результат является одной из причин введения поправочного коэффициента $\gamma$ в марковский процесс с вознаграждением вместо простого суммирования всех будущих вознаграждений. В данном случае мы хотим ограничить горизонт, до которого будем проводить вычисления. Такое ограничение дает коэффициент $\gamma$ со значением меньше 1.

# МППР – 6 [2]

- Граф переходов с вероятностями перехода и вознаграждениями (целые числа).
- Чтобы вычислить ценность состояния $V(s)$ для случая $\gamma = 0$, нужно суммировать ценности всех переходов, умножив их на вероятности.
- В случае $\gamma = 1$ для данного графа $V(s) \to \infty$ для любого состояния.




$$V(Разговоры) = -1 \cdot 0{,}5 + 2 \cdot 0{,}3 + 1 \cdot 0{,}2 = 0{,}3;$$

$$V(Кофе) = 2 \cdot 0{,}7 + 1 \cdot 0{,}1 + 3 \cdot 0{,}2 = 2{,}1;$$

$$V(Дом) = 1 \cdot 0{,}6 + 1 \cdot 0{,}4 = 1{,}0;$$

$$V(Компьютер) = 5 \cdot 0{,}5 + (-3) \cdot 0{,}1 + 1 \cdot 0{,}2 + 2 \cdot 0{,}2 = 2{,}8.$$

# МППР – 7 [2,3]

- Формально политикой (стратегией) $\boldsymbol{\pi}$ называется отображение состояний на вероятности выбора каждого возможного действия.
- Если агент следует стратегии $\boldsymbol{\pi}$ в момент t, то $\boldsymbol{\pi(a|s) = P[A_t = a|S_t = s]}$ – вероятность того, что $A_t = a$ при условии $S_t = s$ (вероятность выбора действия a и дальнейшего перехода в состояние s' при условии нахождения в состоянии s).
- Рассмотрение выше функции ценности V(s) также предполагало нахождение в рамках определенной стратегии $\mathrm{V(s)} \equiv V_{\boldsymbol{\pi}}(\mathrm{s})$.
- Функция ценности действия (функция ценности состояния-действия, q-функция):

$$q_{\boldsymbol{\pi}}(\mathrm{s}, a) = \mathbb{E}_{\boldsymbol{\pi}}[G_t|S_t = s, A_t = a] =$$
$$= \mathbb{E}_{\boldsymbol{\pi}}[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} |S_t = s, A_t = a]$$

- Таким образом, q-функция задает ожидаемый доход, начиная с состояния s с действием a в соответствии с политикой $\pi$.

# Уравнения Беллмана [1,2,3]

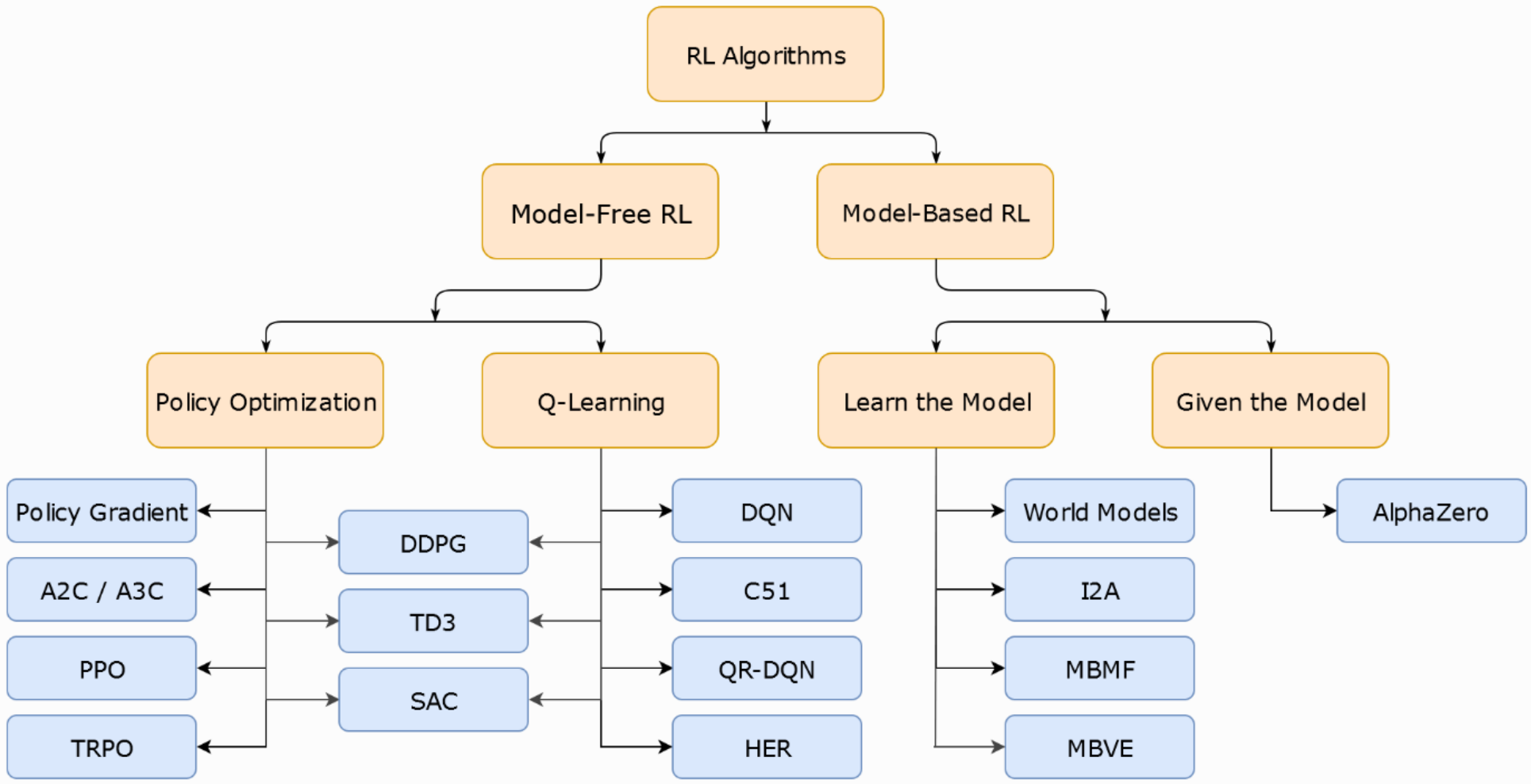
- Вернемся к определению дохода: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
- Для момента времени t+1: $G_{t+1} = R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots$
- Тогда $G_t = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) = R_{t+1} + \gamma G_{t+1}$
- Уравнение Беллмана для функции ценности действия $V_{\boldsymbol{\pi}}(\mathrm{s}) = \mathbb{E}_{\boldsymbol{\pi}}[G_t | S_t = s] = \mathbb{E}_{\boldsymbol{\pi}}[R_{t+1} + \gamma G_{t+1} | S_t = s]$
- Уравнение оптимальности Беллмана (где * означает наилучшее значение или наилучшую стратегию):

$$V_{\boldsymbol{\pi}}(\mathrm{s}) = max_a\big(q_{\boldsymbol{\pi}*}(\mathrm{s}, a)\big) = max_a(\mathbb{E}_{\boldsymbol{\pi}*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a])$$

- Физический смысл: оптимальная ценность состояния соответствует действию, которое дает максимально возможное ожидаемое немедленное вознаграждение плюс дисконтированное отдаленное вознаграждение (с помощью коэффициента γ) для следующего состояния.
- Другая формулировка принципа Беллмана: на каждом шаге следует выбирать оптимальное управление в предположении об оптимальности управлений для всех последующих шагов.
- Таким образом, значения ценности дают нам не только наилучшее из возможных вознаграждений, но и в основном оптимальную стратегию для получения этой награды: если агенту известны ценности для каждого состояния, то он автоматически знает, как собрать все эти вознаграждения. Благодаря принципу оптимальности Беллмана агенту в любом его состоянии достаточно выбрать действие с максимальным ожидаемым вознаграждением, которое представляет собой сумму немедленного вознаграждения и отдаленного вознаграждения, дисконтированного на один шаг.
- Принцип оптимальности Беллмана является важной теоретической концепцией, но решение уравнений Беллмана напрямую практически не используется, так как связано с большими вычислительными затратами.

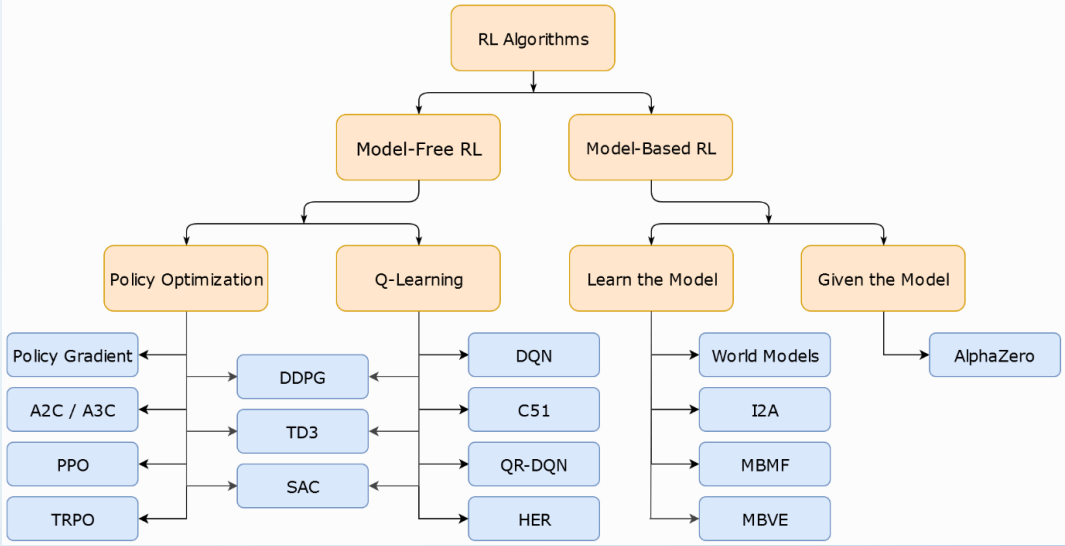
# Таксономия алгоритмов обучения с подкреплением

# Таксономия RL-алгоритмов - 1




1. Краткая таксономия
2. Более детальная таксономия

# Таксономия RL-алгоритмов - 2




- **Model-Free RL** – у агента нет модели среды.
- **Model-Based RL** – у агента есть модель среды (но создание исчерпывающей модели, как правило, невозможно).
- Model-Based RL предполагает использование техник автоматизированного планирования.
- **Given the Model** означает, что модель полностью задана для агента.
- **Learn the Model** означает, что агент изучает среду в процессе решения задачи.
- Методы **Policy Optimization** направлены на решение задачи оптимизации применительно к политике $\boldsymbol{\pi(a|s)}$. Оптимизация политики является основной задачей обучения, поэтому данные методы рассматриваются как стабильные и надежные.
- Методы **Q-обучения** построены на аппроксимации q-функции $\boldsymbol{q_\pi(s,a)}$. Обычно используется целевая функция, основанную на уравнениях Беллмана. Как правило, в этом подходе не используется информация о политике. Поэтому можно использовать данные, собранные в любой момент обучения, независимо от того, как агент изучал среду, на каком шаге были получены данные.
- Также методы **Policy Optimization** и **Q-обучения** не являются взаимоисключающими, поэтому существуют алгоритмы, комбинирующие оба подхода.

# Среды для разработки алгоритмов обучения с подкреплением

Список сред - https://github.com/clvrai/awesome-rl-envs

conda env create -f environment.yml

# OpenAI Gym (Gymnasium)

- Gym – старая версия - https://github.com/openai/gym
  - Документация - https://www.gymlibrary.dev/
- Gymnasium – новая версия - https://github.com/Farama-Foundation/Gymnasium
  - Документация - https://gymnasium.farama.org/
  - Примеры среды - https://gymnasium.farama.org/environments/toy_text/
  - Базовый пример - https://gymnasium.farama.org/content/basic_usage/

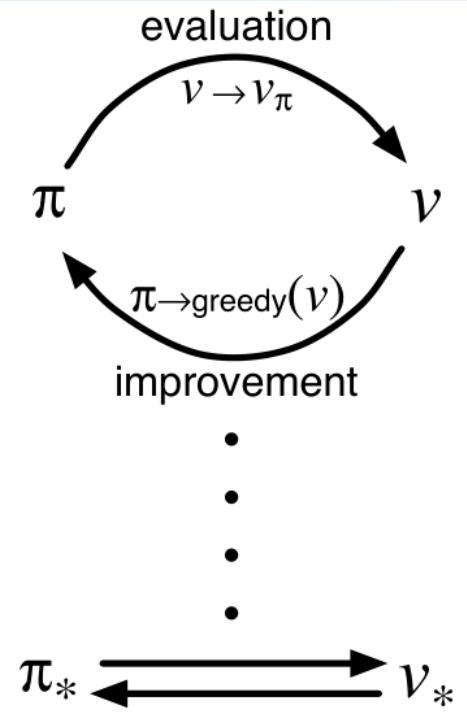
# Алгоритмы с использованием динамического программирования

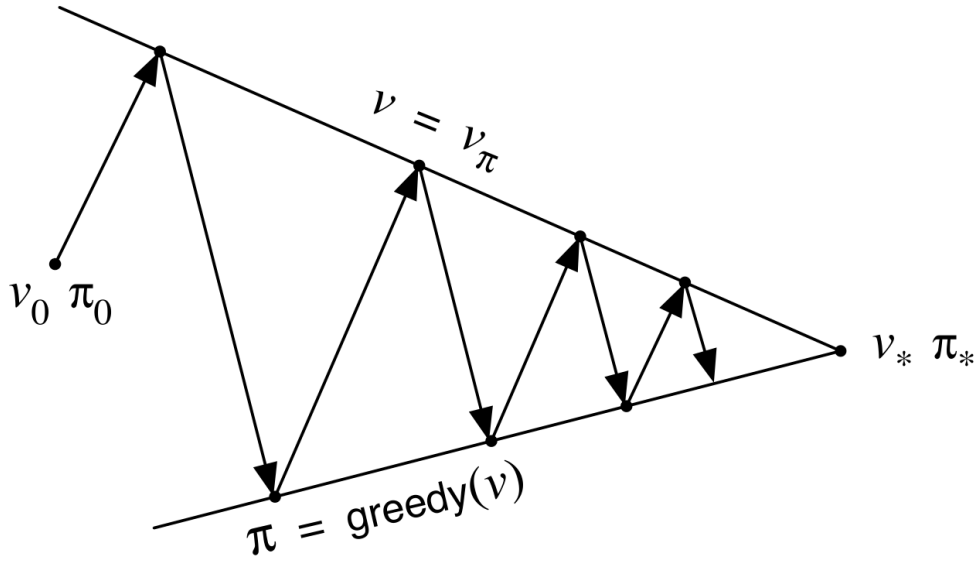
# Policy Iteration -1 [1]

- Итерация по стратегиям (политикам). Метод предполагает итеративное выполнение двух шагов:

1. Оценивание стратегии (Policy Evaluation)
2. Улучшение стратегии (Policy Improvement)

- Таким образом получается последовательность монотонно улучшающихся стратегий и функций ценности:




$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

где $\xrightarrow{\text{E}}$ обозначает *оценивание* стратегии, а $\xrightarrow{\text{I}}$ – *улучшение* стратегии

# Policy Iteration -2 [1]

- **Оценивание стратегии (Policy Evaluation).**
- Вероятность перехода из состояния **s** в состояние **s'** с действием **a** и наградой **r**:

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

- Для состояния **s** и действия **a** функция ценности состояния при стратегии $\pi$:

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

где $\pi(a|s)$ – вероятность предпринять действие $a$ в состоянии $s$ при стратегии $\pi$

- Начальное приближение $v_0$ выбирается произвольно.
- Заключительному состоянию, если оно существует, должна быть сопоставлена ценность 0.
- Каждое следующее приближение получается применением уравнения Беллмана для **v** в качестве правила обновления:

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

# Policy Iteration -3 [1]

- **Оценивание стратегии (Policy Evaluation).**
- Для вычисления следующего приближения $\boldsymbol{v_{k+1}}$ по $\boldsymbol{v_k}$ алгоритм итеративного оценивания стратегии применяет одну и ту же операцию к каждому состоянию **s**: заменяет старую ценность **s** новой ценностью, вычисленной по старым ценностям следующих за s состояний и ожидаемым немедленным вознаграждениям на всех одношаговых переходах при следовании оцениваемой стратегии.
- На каждой итерации итеративного оценивания стратегии ценность каждого состояния обновляется один раз с целью получить новую приближенную функцию ценности $\boldsymbol{v_{k+1}}$.
- Рассмотрим алгоритм итеративного оценивания стратегий. Алгоритм проверяет величину $\Delta = \boldsymbol{max}|\boldsymbol{v_{k+1}(s)} - \boldsymbol{v_k(s)}|$ после каждого прохода и останавливается, если она достаточно мала.

**Алгоритм итеративного оценивания стратегии для оценивания $V \approx v_\pi$**

Вход: $\pi$, стратегия, подлежащая оцениванию
Параметр алгоритма: небольшая пороговая величина $\theta > 0$, определяющая точность оценки
Инициализировать $V(s)$ для всех $s \in \mathcal{S}^+$ произвольным образом с той оговоркой, что $V(terminal) = 0$
Повторять:
    $\Delta \leftarrow 0$
    Повторять для каждого $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\,[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
пока не окажется $\Delta < \theta$

# Policy Iteration -4 [1]

- **Улучшение стратегии (Policy Improvement).**
- Предположим, что мы определили функцию ценности $v_\pi$ для произвольной детерминированной стратегии $\pi$. Для некоторого состояния **s** мы хотели бы знать, стоит или не стоит изменять стратегию, так чтобы она детерминировано выбирала действие **a**, не относящееся к текущей стратегии. Один из способов ответить на этот вопрос – рассмотреть, что будет, если выбрать **a** в состоянии **s**, а затем следовать существующей стратегии $\pi$. Ценность при таком поведении равна:

$$q_\pi(s,a) \doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')].$$

- В этом помогает **теорема об улучшении стратегии.** Пусть $\pi$ и $\pi'$ любая пара детерминированных стратегий, такая, что для всех состояний **s**:

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

- Тогда стратегия $\pi'$ должна быть не хуже, чем $\pi$. Иначе говоря, она должна приносить не меньший ожидаемый доход для всех состояний **s**:

$$v_{\pi'}(s) \geq v_\pi(s)$$

# Policy Iteration -5 [1]

- **Улучшение стратегии (Policy Improvement).**
- Зная стратегию и ее функцию ценности, мы можем оценить изменение стратегии, состоящее в замене одного действия в одном состоянии. Обобщение – рассмотреть изменение всех возможных действий во всех состояниях, выбирая в каждом состоянии то действие, которое кажется наилучшим согласно функции $\boldsymbol{q_\pi(s,a)}$. То есть рассмотреть новую жадную стратегию $\boldsymbol{\pi'}$, определенную следующим образом:

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

- где argmax(a) обозначает значение **a**, при котором следующее далее выражение достигает максимума (возможные неоднозначности разрешаются произвольным образом). Жадная стратегия выбирает действие, которое кажется наилучшим в краткосрочной перспективе – после заглядывания вперед на один шаг – согласно функции $\boldsymbol{v_\pi}$. По построению, жадная стратегия удовлетворяет условиям теоремы об улучшении стратегии, поэтому она заведомо не хуже исходной стратегии.

# Policy Iteration -6 [1]

- **Улучшение стратегии (Policy Improvement).**
- Процесс конструирования новой стратегии, улучшающей исходную путем жадного выбора относительно функции ценности исходной стратегии, называется улучшением стратегии.
- Предположим, что новая жадная стратегия $\boldsymbol{\pi'}$ столь же хороша, но не лучше старой стратегии $\boldsymbol{\pi}$ . Тогда $\boldsymbol{v_{\pi} = v_{\pi'}}$ и для всех состояний **s**:

$$v_{\pi'}(s) \doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi'}(s')].$$

- Но это то же самое, что уравнение оптимальности Беллмана, и потому $\boldsymbol{v_{\pi}}$ должна совпадать с $\boldsymbol{v_*}$, а обе стратегии $\boldsymbol{\pi}$ и $\boldsymbol{\pi'}$ должны быть оптимальными. Таким образом, улучшение стратегии обязано давать строго лучшую стратегию всегда, кроме случая, когда исходная стратегия уже оптимальна.

# Policy Iteration -7 [1]

**Алгоритм итерации по стратегиям (с использованием итеративного оценивания стратегии) для оценивания $\pi \approx \pi_*$**

1. Инициализация
   $V(s) \in \mathbb{R}$ и $\pi(s) \in \mathcal{A}(s)$ выбираются произвольно для всех $s \in \mathcal{S}$

2. Оценивание стратегии
   Повторять:
   $\Delta \leftarrow 0$
   Повторять для каждого $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) [r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   пока не окажется $\Delta < \theta$ (небольшое положительное число, определяющее точность оценки)
3. Улучшение стратегии
   *policy-stable* $\leftarrow$ *true*
   Для каждого $s \in \mathcal{S}$:
   *old-action* $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
   Если *old-action* $\neq \pi(s)$, то *policy-stable* $\leftarrow$ *false*
   Если *policy-stable*, то остановиться и вернуть $V \approx v_*$ и $\pi \approx \pi'$; иначе перейти к 2

# Policy Iteration (реализация)-1

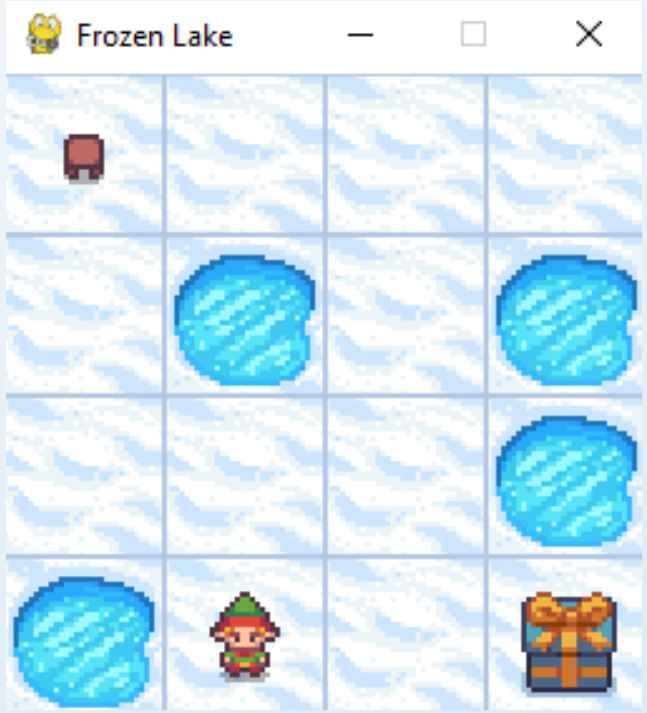
- Реализация использует фрагменты кода
  - https://github.com/escape-velocity-labs/beginner_master_rl
  - https://aleksandarhaber.com/policy-iteration-algorithm-in-python-and-tests-with-frozen-lake-openai-gym-environment-reinforcement-learning-tutorial/

- Файл «flake.py» - информация о среде:

```
Пространство состояний:
Discrete(16)

Пространство действий:
Discrete(4)

Диапазон наград:
(0, 1)

Вероятности для 0 состояния и 0 действия:
[(0.3333333333333333, 0, 0.0, False),
 (0.3333333333333333, 0, 0.0, False),
 (0.3333333333333333, 4, 0.0, False)]

Вероятности для 0 состояния:
{0: [(0.3333333333333333, 0, 0.0, False),
     (0.3333333333333333, 0, 0.0, False),
     (0.3333333333333333, 4, 0.0, False)],
 1: [(0.3333333333333333, 0, 0.0, False),
     (0.3333333333333333, 4, 0.0, False),
     (0.3333333333333333, 1, 0.0, False)],
 2: [(0.3333333333333333, 4, 0.0, False),
     (0.3333333333333333, 1, 0.0, False),
     (0.3333333333333333, 0, 0.0, False)],
 3: [(0.3333333333333333, 1, 0.0, False),
     (0.3333333333333333, 0, 0.0, False),
     (0.3333333333333333, 0, 0.0, False)]}
```

# Policy Iteration (реализация)-2

- Файл «policy_iteration.py» - реализация алгоритма




```
Стратегия:
array([[0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25]])
Алгоритм выполнился за 1000 шагов.
Стратегия:
array([[1.  , 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , 1.  ],
       [0.  , 0.  , 0.  , 1.  ],
       [0.  , 0.  , 0.  , 1.  ],
       [1.  , 0.  , 0.  , 0.  ],
       [0.25, 0.25, 0.25, 0.25],
       [0.5 , 0.  , 0.5 , 0.  ],
       [0.25, 0.25, 0.25, 0.25],
       [0.  , 0.  , 0.  , 1.  ],
       [0.  , 1.  , 0.  , 0.  ],
       [1.  , 0.  , 0.  , 0.  ],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.  , 0.  , 1.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  ],
       [0.25, 0.25, 0.25, 0.25]])
```