ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА ФИЗИКО-МАТЕМАТИЧЕСКИХ МЕТОДОВ УПРАВЛЕНИЯ

БАКАЛАВРСКАЯ РАБОТА

«РЕАЛИЗАЦИЯ НЕЙРОСЕТЕВОГО АЛГОРИТМА ПОИСКА ПУТИ В ЛАБИРИНТЕ»

Выполнил студент
442 группы
Юдинцев Егор Викторович

Научный руководитель:
Галяев Андрей Алексеевич

Допущен к защите

Зав.кафедрой

Москва

2019

Оглавление

ВВЕДЕНИЕ

Данная работа посвящена постановке задачи поиска пути в лабиринте и программной реализации для данной задачи с помощью метода обучения с подкреплением (Reinforcement Learning). Программная реализация включает в себя два пункта:

- Разработка среды для обучения агента;
- Разработка алгоритма обучения агента.

Построенные программные модели поиска пути в лабиринте сопоставимы с реальными физическими задачами. В данной работе рассматривается движение беспилотного летательного аппарата (БПЛА), который доставляет объекты из одной точки пространства в другую.

В результате работы был создан и протестирован алгоритм, позволяющий осуществлять управление агентом в трёхмерном лабиринте, имитирующим атмосферу. В процессе исследования алгоритма были найдены оптимальные параметры для обучения агента в двух моделях:

- Модель без учета физических параметров среды;
- Модель с учетом высоты слоя.

Алгоритм показал свою эффективность в стационарных случаях.

1. Теоретическое введение

Современные задачи науки и техники требуют применения современных методов, позволяющих быстро и корректно обрабатывать большие объёмы данных, поступающих с многочисленных датчиков. Кроме того, с увеличением сложности задач, стоящих перед кибернетическими агентами, усложняется их поведение. Классические методы программирования показывают свою неэффективность в решение современных задач. Это проявляется в нехватке доступной памяти компьютера и медленном выполнении программ.

Эффективным методом решения современных задач является метод машинного обучения (англ. Machine Learning). Машинное обучение - это область компьютерных наук, которая часто использует статистические методы, чтобы дать компьютерам возможность «учиться» (то есть постепенно улучшать производительность в конкретной задаче) с данными, не будучи заранее явно запрограммированной. Термин «машинное обучение» был придуман в 1959 году Артуром Самуэлем. Бурное развитие данного метода началось лишь в 1990-х годах вместе с ростом вычислительных мощностей компьютеров. Машинное обучение используется в ряде вычислительных задач, где проектирование и программирование явных алгоритмов с хорошей производительностью является задачей трудной или неосуществимой.

Существует две основных концепции машинного обучения [1]:

- Обучение с учителем, в котором агент обучается производить определённые действия на основании предварительно подготовленных выборок;

- Обучение без учителя, в котором агент самостоятельно формирует стратегию поведения, опираясь на изменения, производимые его действиями.

Обучение с подкреплением (англ. reinforcement learning - RL) принадлежит ко второму типу машинного обучения. Агент перебирает все варианты действий и из всех возможных действий выбирает те, которые принесут ему наибольшее итоговое вознаграждение. В основе обучения с подкреплением лежит концепция 'метода проб и ошибок'.

В данной работе для решения задачи поиска пути в лабиринте применяется метод обучения с подкреплением. Обучение агента происходит благодаря взаимодействию с окружающей средой. Лабиринт - это и есть среда, предназначенная для экспериментального исследования, в которой движется управляемый агент.

Базовое подкрепление моделируется как процесс принятия марковских решений [2]:

- Множество состояний среды и агента S;
- Множество возможных действий агента A;
- $P_a(s, s') = P(s_{t+1} = s' \mid s_t = s,\ a_t = a)$ - вероятность перехода из s в s' при действии a;
- $R_a(s, s') = R(s_{t+1} = s' \mid s_t = s,\ a_t = a)$ - вознаграждение, получаемое после перехода в состояние s' из состояния s с вероятностью $P_a(s, s')$;
- Правила, описывающие то, что наблюдает агент.

Поведение агента описывается следующей цепочкой действий:

$$\text{состояние} \rightarrow \text{действие} \rightarrow \text{поощрение} \rightarrow \text{состояние} \rightarrow$$
$$\rightarrow \text{действие} \rightarrow \text{поощрение} \rightarrow \ldots$$

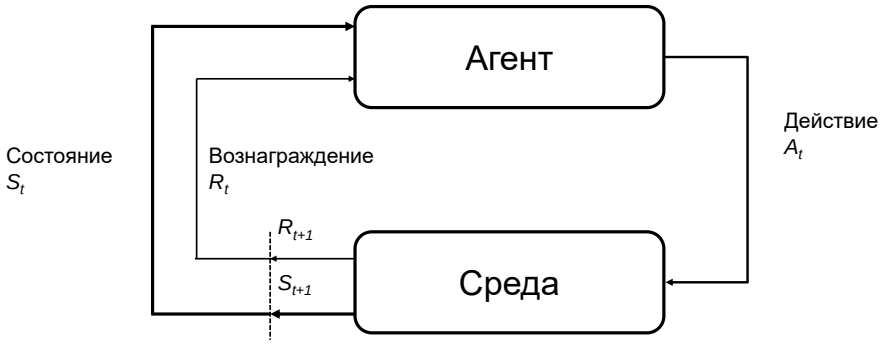


Рисунок 1: SARSA-модель

В англоязычной литературе данный процесс носит название «SARSA» («State-Action-Reward-State-Action-...»).

Вводится некоторая политика (англ. *policy*):

$$\pi \ : \ S \times A \to [0, 1]$$

$\pi(a \mid s) = P(a_t = a \mid s_t = s)$ - вероятность действия a в состоянии s.

Цель агента - выбрать такую политику π,чтобы при следовании ей сумма вознаграждений, получаемых от среды, была максимальна. Ожидаемая награда в момент времени t определяется как:

$$R_t = M\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right],$$

где $M[\cdot]$ - математическое ожидание, $\gamma \in (0, 1)$ - коэффициент дисконтирования.

Стратегия агента не стремится к тому, чтобы получать максимальную выгоду на каждом шаге. Введем функцию $Q(s, a)$, которая парам состояние-действие ставит в соответствие число. Данное число называется ценностью состояния-действия. Также на каждом временном шаге t

агент получает вознаграждение r_t:

$$Q^{\pi}(s,a) = M_{\pi}[R_t|\, s_t = s, a_t = a] = M_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,|\, s_t = s, a_t = a\right],$$

где индекс π означает выбор действий в соотвествии с некоторой политикой. Эта функция характеризует ожидаемую награду, получаемую агентом. Можно получить рекурсивную формулу для оценки данной функции:

$$Q^{\pi}_{i+1}(s,a) = M_{\pi}\left[r_t + \gamma Q^{\pi}_i(s_{t+1} = s', a_{t+1} = a') \,|\, s_t = s, a_t = a\right]$$

Целью агента является нахождение оптимальной политики π, на которой достигается максимальная ожидаемая награда. Следовательно, необходимо найти такую π^*, которая в результате нам дает максимальное значение action-value функции $Q^*(s,a)$ среди всех существующих политик. Формула для оценки оптимального значения action-value функции определяется следующим образом:

$$Q_{i+1}(s,a) = M[r_t + \gamma max_{a'} Q_i(s',a') \,|\, s,a]$$

При $i \to \infty$ следует, что $Q_i(s,a) \to Q^*(s,a)$. Данный процесс называется *алгоритмом итерации значений* (англ. *value iteration algorithm*).

2. Постановка задачи

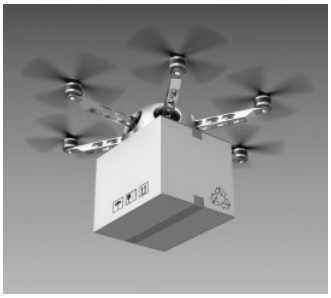


Рисунок 2: Беспилотный аппарат

Рассмотрим движение беспилотного аппарата (агента) в атмосфере (испытательной среде, лабиринте). Задачей агента является сбор грузов в различных точках пространства и их доставка до точек выгрузки с наименьшими затратами топлива.

Агент движется в трёхмерном пространстве, каждому слою атмосферы соответствует одна координата по оси Z, кроме того, происходит движение в плоскости Oxy. Каждой точке пространства соответствует определённое значение плотности атмосферы, от которой зависит расход топлива, требуемого для перемещения.

Множество действий, доступных агенту состоит из восьми элементов:

- *move south* - увеличение координаты y на 1 (движение на юг);
- *move north* - уменьшение координаты y на 1 (движение на север);
- *move east* - увеличение координаты x на 1 (движение на восток);
- *move west* - уменьшение координаты x на 1 (движение на запад);
- *move up* - увеличение координаты z на 1 (движение вверх);
- *move down* - уменьшение координаты z на 1 (движение вниз);
- *pickup* - сбор объекта;
- *dropoff* - сброс объекта.

3. Q-learning алгоритм

Поставленную задачу необходимо формализовать для применения метода обучения с подкреплением. Агентом является беспилотный аппарат, атмосфера выполняет роль внешней среды, обучающей агента. Взаимодействие со средой происходит при каждом действии агента на протяжении заданного промежутка времени или до достижения терминального состояния. На каждом временном шаге t агент получает некоторое описание состояния окружающей среды $s_t \in S$, где S — множество возможных состояний среды, и на основании этого описания выбирает действие, где $A(s_t)$ — множество действий, возможных в состоянии s_t.

Наиболее популярным алгоритмом в обучении с подкреплением является Q-learning алгоритм (Watkins, 1989). Популярность алгоритма обусловлена его простотой и эффективностью.

Название функции «Q», которая возвращает вознаграждение, используемое для обеспечения подкрепления, обозначает «качество» (англ. quality) действия, предпринимаемого агентом в определенном состоянии.

Алгоритм относится к классу TD (Temporal-Difference). В TD-методах процесс обучения основывается на опыте взаимодействия агента со средой без использования модели среды. Расчетные оценки состояний (в случае задачи управления состояний-действий) в TD-методах обновляются, основываясь на других полученных оценках, т.е. они самонастраиваются [3].

В простом случае, одношаговый Q-learning алгоритм определяется следующим образом:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)],$$

где α, γ - параметры Q-learning. R - вознаграждение. α - это темп обучения, а γ - дисконтирующий множитель. Гамма определяет, какую мы хотим придать важность вознаграждениям, ожидающим нас в перспек-

тиве [4].

Алгоритм выглядит следующим образом [4]:

- Инициализация произвольного $Q(s, a)$ $\forall s \in S, a \in A(s)$,
- Повторение для каждого эпизода:
  - Инициализация S
  - Повторение для каждого эпизода:
    * Выбор A из S, получение вознаграждения R, подсчет S'
    * Вычисление Q(S,A) по формуле выше
    * Переход к новому состоянию S'

4. Программная реализация

Основой для решения послужила библиотека Gym от OpenAI [5]. Библиотека содержала рассмотренную мной задачу в упрощённом виде: обучение с подкреплением использовалось для оптимизации обработки заказов и движения такси в двумерной плоскости. Несмотря на кажущуюся схожесть с задачей управления беспилотным аппаратом, требовалась серьёзная доработка существующего решения:

- Требовалось обобщить задачу на случай движения в трёх измерениях;
- Требовалось изменить постановку задачи так, чтобы добавить физический и прикладной смыслы.

Обе задачи были выполнены.

Основные компоненты программной реализации:

- main.py - основной файл, в котором реализовано обучение агента с помощью Q-learning, заданы параметры обучения (количество эпизодов, максимальное количество шагов в эпизоде, параметры Q-learning и т.д.)
- labyrinth.py - файл, в котором реализованы 'правила' взаимодействия агента со средой.
- map_generation.py - файл, который содержит необходимые функции для построения символьного поля среды, в которой происходит обучение агента.
- discrete.py - вспомогательный файл, который был разработан OpenAI для обучения с подкреплением.

Код каждого файла представлен в Приложении. Теперь рассмотрим каждую часть более подробно.

4.1 Создание среды

Как отмечалось выше, программа была реализована с помощью библиотеки *gym* от OpenAI, также был использован пакет *numpy* для более удобной работы с матрицами.

В моей задаче среда - это параллелепипед, задаваемый тремя параметрами (длина, ширина, высота). Создание символьного поля производится в файле map_generation.py (см. Приложение), в котором с помощью символов +, -, | и : формируется среда, а также случайным образом расставляются пункты назначения для агента (R(ed), G(green), B(lue), Y(ellow)) (см. рисунок ниже).

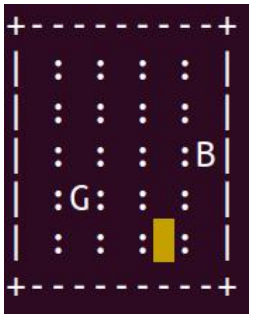


Рисунок 3: Слой в какой-то момент времени.

Основная цель данного символьного поля - провизуализировать перемещение агента в среде. Как отмечалось в разделе 'Постановка задачи', агенту доступны следующие действия (actions):

- Двигаться на юг (move south)
- Двигаться на север (move north)
- Двигаться на восток (move east)
- Двигаться на запад (move west)
- Двигаться вверх (move up)
- Двигаться вниз (move down)

- Подобрать объект (pickup)
- Положить объект (dropoff)

За каждое действие агент получает очки (rewards). Они могут быть как очками вознаграждения, когда агент доставил объект из одной точки в другую, так и штрафными очками, когда агент сделал неправильное действие, например, доставил объект не в то место или врезался в препятствие. Кроме того, за каждое перемещение агент теряет очки (топливо). И в зависимости от того, в какой ячейке находится агент, он затрачивает различное количество очков. За то, сколько необходимо потратить на перемещение, отвечает функционал, который каждому набору данных в ячейке ставит в соответствие вознаграждение. В самом простом случае в ячейке хранится уровень слоя, но в моей реализации среда также может учитывать плотность и давление воздуха, сопротивление ветра на данной высоте. При проведении различных испытаний, связанных с изменением количества эпизодов обучения агента, параметров Q-learning и т.д., учитывается только высота слоя, поэтому вознаграждение рассчитывается следующим образом:

$$\text{reward} = \text{lay\_reward}(\text{lay}),$$

где lay_reward - это структура данных 'ключ-значение', где ключ - это номер слоя, а значение - очки на этом слое. В моём случае нулевой слой соответствует вознаграждению -(n+1)/2, а (n-1)-ый слой - вознаграждению -1. Для более общего случая вводится трехмерный массив, который характеризует затраты топлива на перемещение в каждой точке слоя:

$$\text{reward} = \text{cell\_reward}[\text{lay}][\text{row}][\text{column}],$$

где, cell_reward - трехмерный массив, а lay, row, column - это индексы для слоя, строки и столбца в этом массиве. Данная общая конструкция обес-

печивает возможность ввести в модель сопротивление ветра, плотность и давление атмосферы.

Теперь рассмотрим количество возможных состояний в данной задаче. Всю среду можно представить в виде трехмерной сетки $size\_x \cdot size\_y \cdot size\_z$. Количество ячеек этой сетки равно количеству возможных расположений агента. В среде также расположены 4 возможных места назначения. Если еще учесть одно состояние объекта: объект находится у агента, то можно подсчитать общее количество состояний в нашей среде для обучения агента. Итого, четыре возможных расположения пунктов назначения и 5 возможных расположений для объекта. Следовательно, в нашей среде насчитывается

$$N = size\_x \cdot size\_y \cdot size\_z \cdot 5 \cdot 4$$

возможных состояний для агента. Агент взаимодейтсвует с одним из этих состояний и предпринимает решение, какое действие ему принять дальше.

После того, как было задано количество состояний, нужно учесть границы среды, чтобы в дальнейшем агент не смог за них выйти. Основную часть данного файла занимает шестивложенный цикл по следующим параметрам:

- 3 пространственных параметра (lay, row, column),
- 2 по состояниям объекта и пунктов назначения,
- 1 по возможным действиям.

Внутри данного шестивложенного цикла происходит заполнение первичной таблицы вознаграждений под названием P. Данная таблица является матрицей, в которой количество столбцов соответствует числу возможных действий, а количество строк соответствует количеству состояний.

На рисунке ниже представлена данная матрица P при рандомном индексе 442.

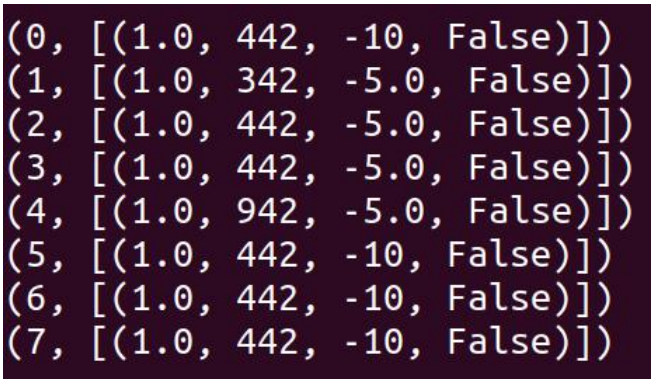


Рисунок 4: P[442]

Как интерпретировать эти данные?

(action, [(probability, nextstate, reward, done)]),

Причем,

- значения 0 - 7 соответствуют действиям (south, north, east, west, move up, move down, pickup, dropoff),
- done характеризует результат доставки объекта в пункт назначения.

Каким же образом заполняется данная таблица? В шестивложенном цикле существует проверка на то, какое действие совершается, и, в зависимости от результата действия, агент получает определенное количество очков. Новое состояние получается при помощи функции encode и пяти параметров:

- 3 пространственных (new_lay, new_row, new_col),
- расположения объекта (new_obj_idx),
- расположения пункта назначения (desc_idx).

В данном файле также представлена функция render(), которая реализует 2D-отрисовку перемещения агента в среде. В render используется decode(), которая преобразует входные данные в расположение агента, объекта и пунктов назначения при визуализации. Ниже представлено несколько последовательных расположений агента в среде в виде куба со стороной 5:

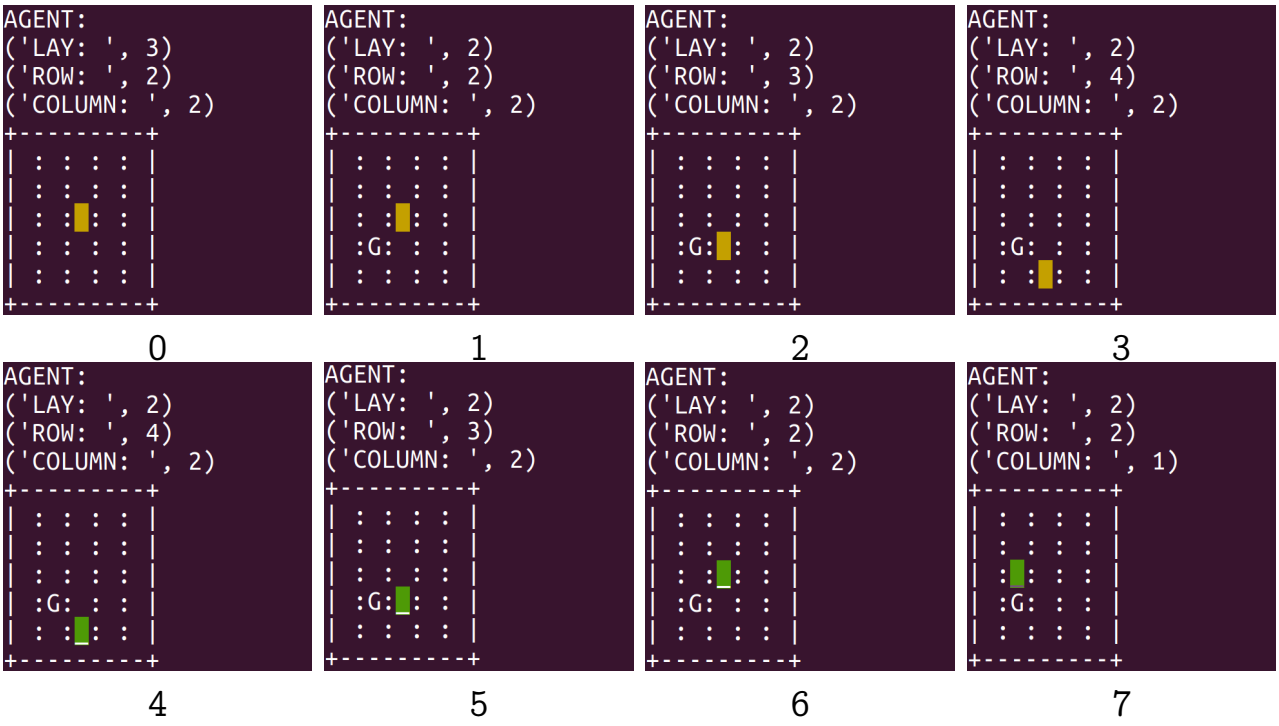


Рисунок 5: Последовательность действий агента в одном из эпизодов

Задача для агента формулируется следующим образом: "Доставить объект из пункта А в пункт Б с минимальными затратами топлива.".

4.2 Основной алгоритм: Q-learning

Прежде всего нужно заметить, что данную задачу можно решить без машинного обучения. С помощью цикла while можно написать алгоритм, который реализовывал бы доставку объекта из одной точки локации в другую, но, очевидно, что данный алгоритм был бы совершенно не эффективен на сетках любой размерности. Теперь перейдем к описанию самого алгоритма.

Используя среду, которую я описал в предыдущем подпункте, и gym, я реализовал Q-learning алгоритм для поставленной задачи. Перед тем, как описывать реализацию алгоритма, необходимо описать несколько полезных функций, которые были разработаны OpenAI. Прежде всего, env = gym.make() - это сердце OpenAI Gym, представляющее собой интерфейс среды. У env есть несколько полезных методов:

- env.step(action) - продвигает развитие окружающей среды на один шаг по времени.
- env.reset - обновляет среду, то есть перезапускает исходную среду и возвращает новое случайное исходное состояние.

Напомню основные детали Q-learning метода. Среда вознаграждает агента за постепенное обучение и за то, что в конкретном состоянии он совершает наиболее оптимальный шаг. В предыдущем подпункте я вводил таблицу P, по которой будет учиться агент. Опираясь на таблицу вознаграждений, он выбирает следующее действие в зависимости от того, насколько оно затратно, а затем обновляет величину, именуемую Q-значением. В результате создается новая таблица (Q-таблица), отображаемая на комбинацию (State, Action). Если Q-значения оказываются лучше, то получаются более оптимизированные вознаграждения. Например, если агент с объектом находится в точке, в которой нужно выложить объект, то Q-значение для 'dropoff' оказывается выше, чем для остальных действий [6]. При взаимодействии со средой, Q-значение в Q-таблице обновляется на основе следующей формулы:

$$Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma \max Q(s',a') - Q(s,a)],$$

где α, γ - параметры Q-learning. R(s,a) - вознаграждение, α - это темп обучения, а γ - дисконтирующий множитель. Гамма определяет, какую мы хотим придать важность вознаграждениям, ожидающим нас в перспективе. Для того, чтобы агент был 'любопытным', вводится параметр

ϵ, отвечающий за так называемый exploration, то есть за исследование среды. Также вводится параметр max_steps, который отвечает за то, чтобы агент со временем перешел к следующему эпизоду, а не зашел в тупик, из которого не сможет выйти.

Сам алгоритм выглядит довольно просто:

1. Инициализация Q-таблицы с нулями,
2. Цикл по количеству эпизодов:
   (a) Перезапускаем среду (метод env.reset()),
   (b) Внутренний цикл по количеству максимальных шагов:
      i. Взаимодействие агента со средой: выполнение доступных для агента действий, в зависимости от 'любопытности' агента,
      ii. Переход к новому состоянию по результатам взаимодействия со средой,
      iii. Обновление значений Q-таблицы по формуле, которая была представлена выше,
      iv. Новое состояние становится текущем состоянием,
      v. Если агент не выполнил поставленную задачу, то алгоритм повторяется с п.(i),
      vi. Если выполнил, то уменьшаем 'любопытность' агента и начинаем новый эпизод с п.(a).

Дальше, в файле main.py представлено обучение агента еще в нескольких эпизодах. Это было сделано для того, чтобы провизуализировать как ведет себя агент при взаимодействии со средой.

4.3 Вспомогательный файл от OpenAI

Данный файл содержит необходимые для обучения агента методы, такие как:

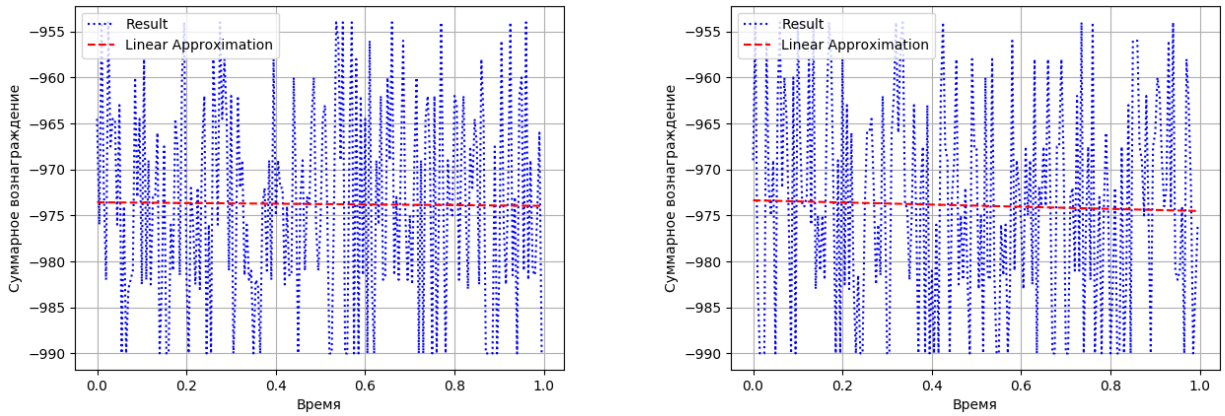
- reset() - перезапускает среду,
- step() - продвигает развитие окружающей среды на один шаг.

5. Исследование поведения алгоритма

Проведем ряд экспериментов над нашим алгоритмом. Будем варьировать количество эпизодов обучения, условия Q-learning'а (темп обучения α и дисконтирующий множитель γ), модель среды (учет высоты, плотность и давление воздуха, сопротивление ветра) и проанализируем полученные результаты.

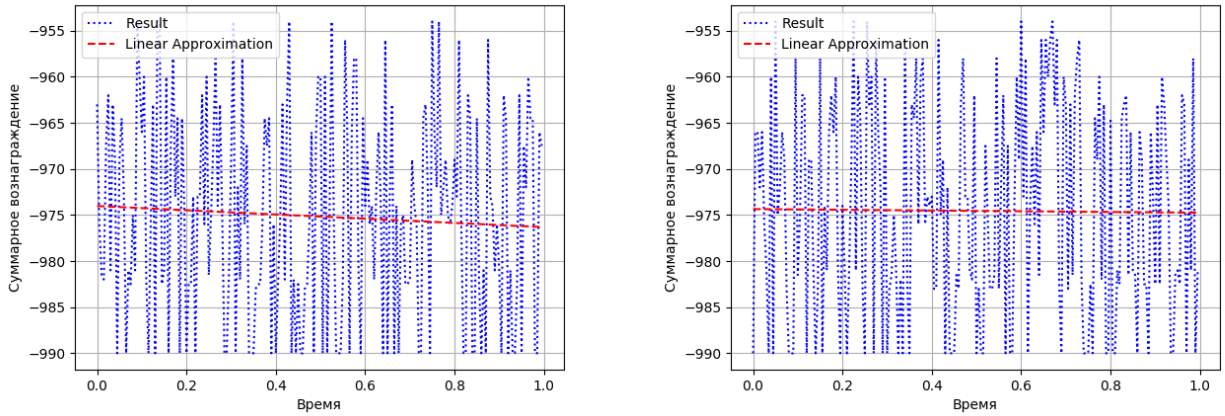
5.1 Алгоритм без учета высоты, сопротивления ветра, давления и плотности воздуха

Для начала рассмотрим, как ведет себя алгоритм при $\alpha = 0$ (темп обучения), $\gamma = 0.75$ (дисконтирующий множитель, выбрали какое-то случайное значение), $\epsilon = 1$ ('любопытность' агента, которое изменяется в процессе обучения). Кроме того, зафиксируем нашу среду как куб со стороной, равной 5. И будем варьировать количество эпизодов обучения N (total_episodes). Рассмотрим следующие значения N: 50, 500, 500, 5000. На графиках будут представлены две зависимости: 1) Полученные экспериментальные точки. 2) Линейная аппроксимация этих точек для более информативного представления.



(а) (б)

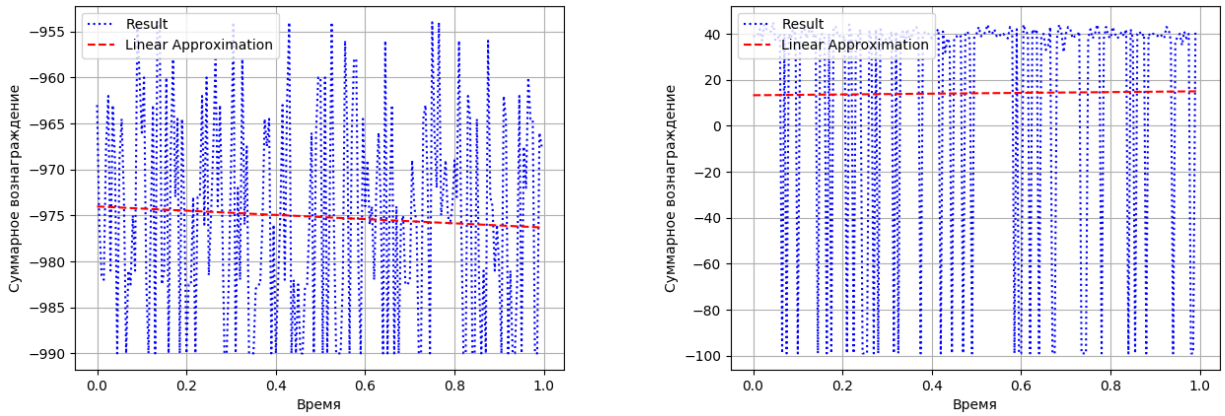
Рисунок 6: Зависимость суммарного вознаграждения от пройденного времени при α=0, γ=0.75 и: (а) N=50; (б) N=500



(а) (б)

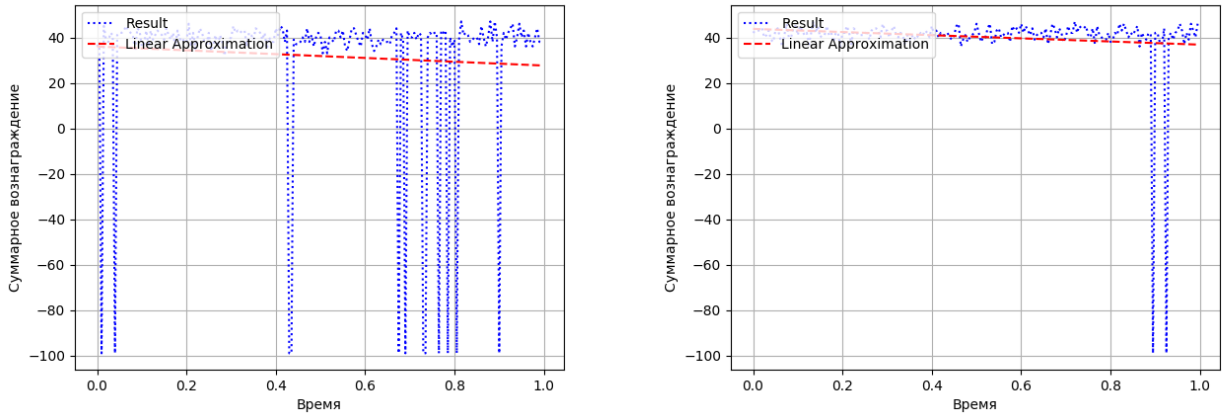
Рисунок 7: Зависимость суммарного вознаграждения от пройденного времени при α=0, γ=0.75 и: (а) N=5000; (б) N=50000

Получены достаточно ожидаемые графики. При $\alpha = 0$, обучение агента практически не происходит, и даже увеличение числа эпизодов обучения не дает адекватный результат. Поэтому, зафиксируем N=5000, оставим все остальные параметры с прежними значениями и проведем эксперименты с α, изменяющимся с шагом 0.2.



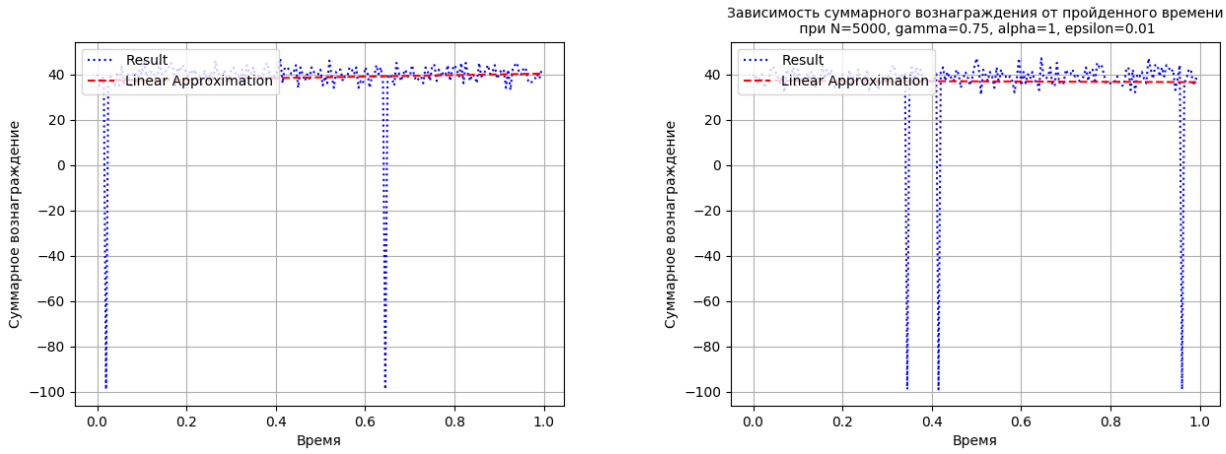
(а) (б)

Рисунок 8: Зависимость суммарного вознаграждения от пройденного времени при N=5000, γ=0.75 и: (а) α=0; (б) α=0.2



(а) (б)

Рисунок 9: Зависимость суммарного вознаграждения от пройденного времени при N=5000, γ=0.75 и: (а) α=0.4; (б) α=0.6

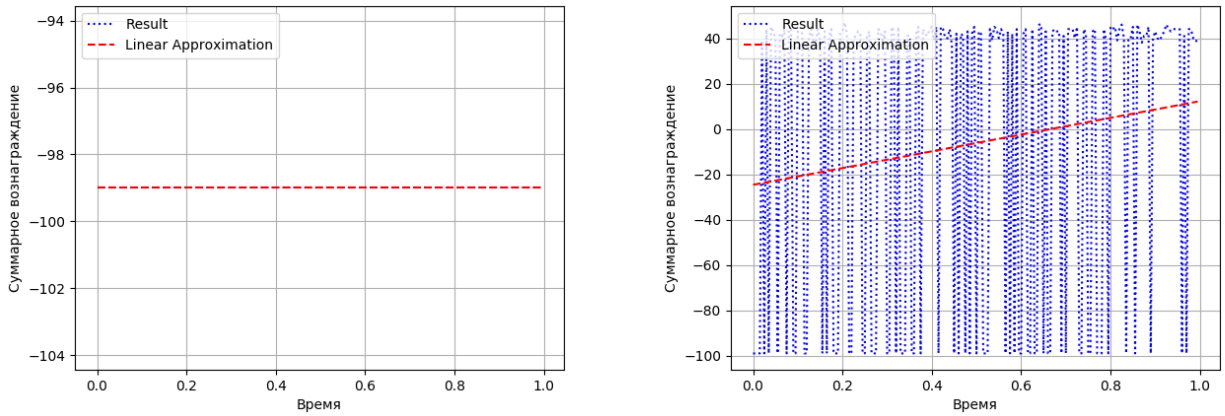


(а) (б)

Рисунок 10: Зависимость суммарного вознаграждения от пройденного времени при N=5000, γ=0.75 и: (а) α=0.8; (б) α=1.0

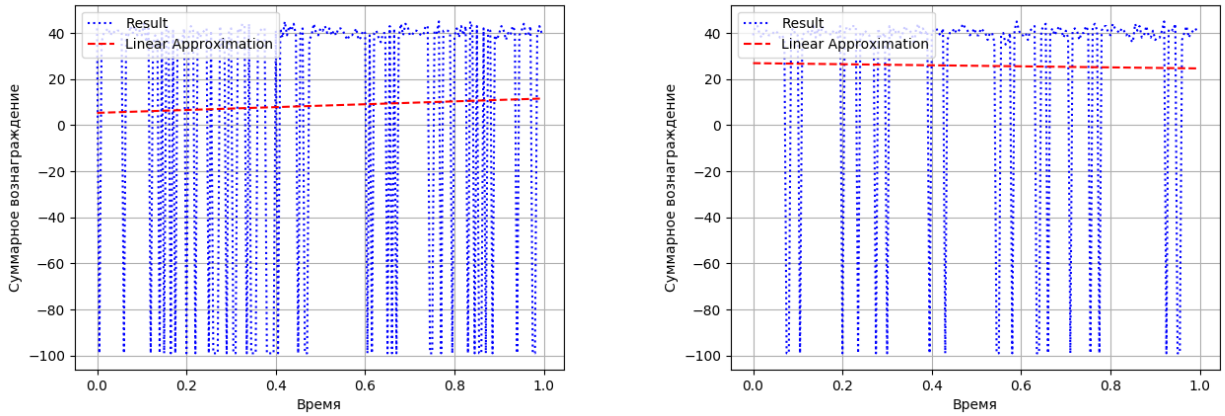
Как можно увидеть из графиков, даже при N=5000 и $\alpha = 0.2$ результат стал намного лучше: вознаграждение возросло до 20-40, а при $\alpha = 0$, оно было равно $\sim$(-1000). При увеличении параметра α до 1.0, можно наблюдать, что в какой-то момент, при α>0.6, получаемое агентом вознаграждение вышло на постоянное значение ($\sim$39). Следовательно, существует такое значение α, при котором оно уже не будет вносить вклад в обучение, как бы его не увеличивали. Проводя эксперимент при данных параметрах, я получил, что точка насыщения для α, т.е. при котором происходит выход на постоянное значение, равняется 0.63.

Зафиксируем α=0.63 и N=5000 и будем варьировать дисконтирующий множитель γ. Напомню, что данный параметр отвечает за то, какую мы хотим придать важность вознаграждениям, ожидающим нас в перспективе.



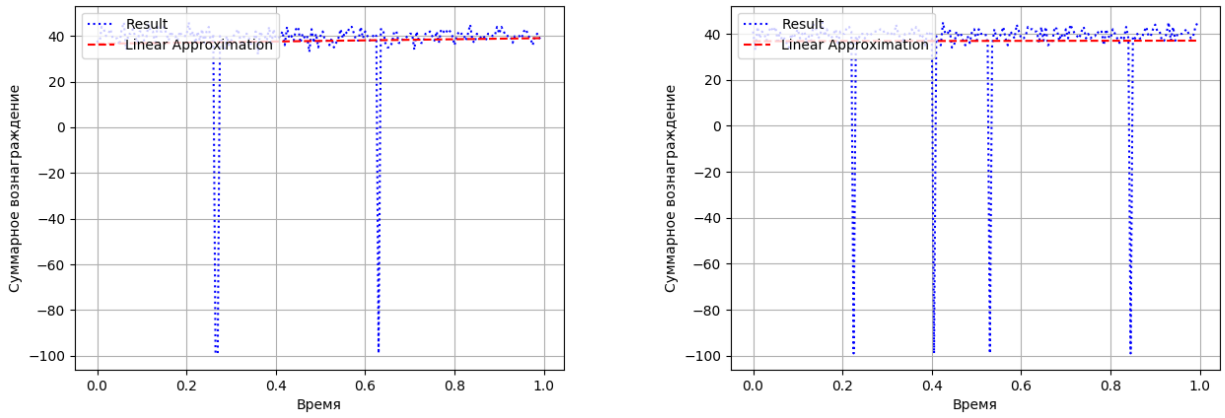
(а) (б)

Рисунок 11: Зависимость суммарного вознаграждения от пройденного времени при N=5000, α=0.63 и: (а) γ=0; (б) γ=0.2



(а) (б)

Рисунок 12: Зависимость суммарного вознаграждения от пройденного времени при N=5000, α=0.63 и: (а) γ=0.4; (б) γ=0.6

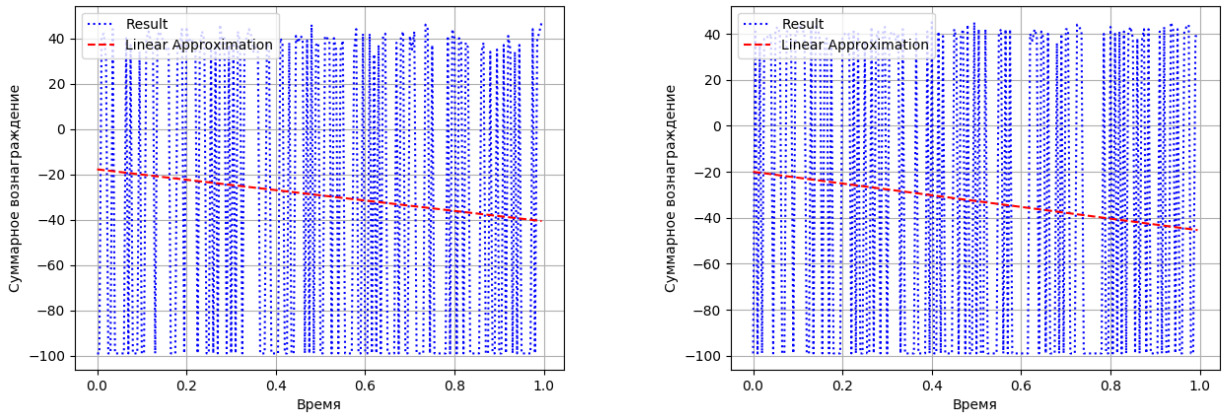


(а) (б)

Рисунок 13: Зависимость суммарного вознаграждения от пройденного времени при N=5000, α=0.63 и: (а) γ=0.8; (б) γ=1.0

Как можно заметить из приведенных графиков, при γ=0 алгоритм ведет себя неадекватно, но при увеличении γ система достигает какого-то насыщения, т.е. существует такое значение параметра γ, при котором дальнейшее увеличение данного параметра не дает вклада в обучение. Проводя ряд испытаний, я получил, что оптимальное значение для γ равняется 0.74.

Также хотелось бы привести примеры графиков при значения $\alpha_{cr} = 1.45$, $\gamma = 0.74$ и $\gamma_{cr} = 1.05$, $\alpha = 0.63$. То есть, при таких параметрах, при которых происходят заметные ухудшения при обучении.



(а) (б)

Рисунок 14: Зависимость суммарного вознаграждения от пройденного времени при N=5000 и: (а) γ_{cr}=1.05, α=0.63; (б) γ=0.74, α_{cr}=1.45

Как можно заметить, при данных значениях параметров обучение ведет себя сильно хуже.

5.2 Алгоритм с учетом высоты слоя

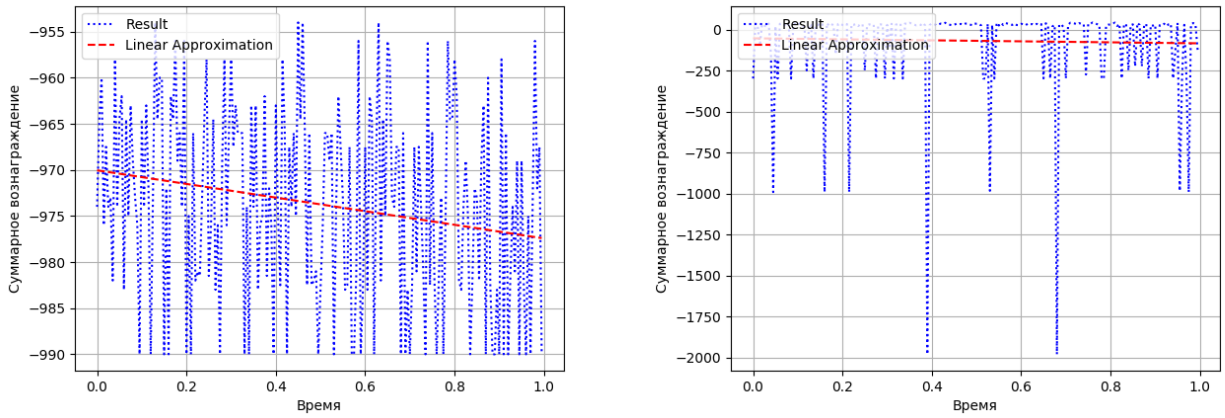
В предыдущем пункте была рассмотрена работа алгоритма без учета высоты, т.е. не была учтена разреженность атмосферы при увеличении слоя, что было грубым допущением. Попробуем усложнить модель (в данном случае, среду). Добавим (в грубом приближении), что с увеличением слоя, будет затрачено меньше топлива на перемещение в плоскости Oxy.

Как отмечалось выше, за вознаграждение на каждом слое в модели, в которой учитывается только высота слоя, отвечает следующая конструкция:

$$\text{reward} = \text{lay\_reward}(\text{lay}),$$

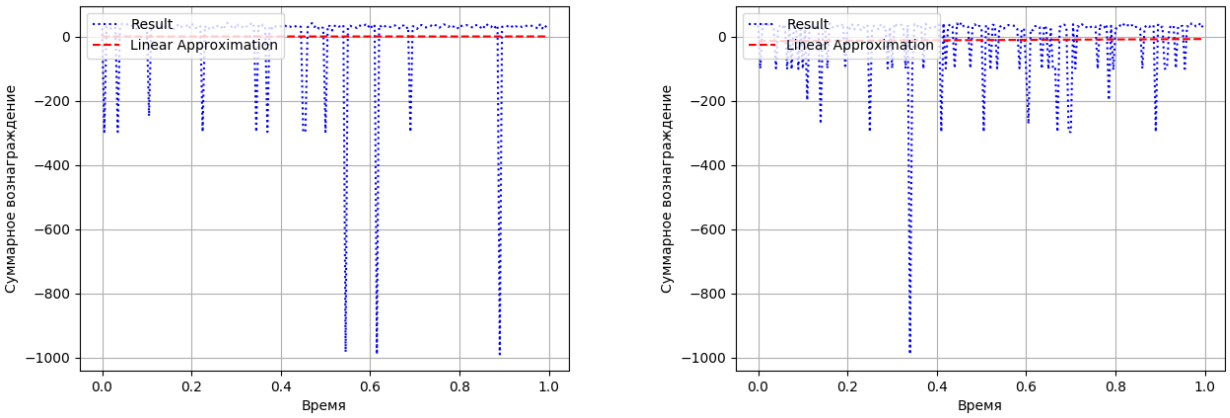
где lay_reward - это структура данных 'ключ-значение', где ключ - это номер слоя, а значение - очки на этом слое. В моём случае, нулевой слой соответствует вознаграждению - (n+1)/2, а (n-1)-ый слой - вознаграждению -1.

В предыдущем пункте было показано, что происходит с алгоритмом, если параметр α=0. Поэтому, сразу перейдем к рассмотрению случая, при котором мы будем варьировать α при фиксированном N. Пусть γ=0.75, N=5000, а α изменяется в пределах от 0 до 1.0 с шагом 0.2.



(а) (б)

Рисунок 15: Зависимость суммарного вознаграждения от пройденного времени при N=5000, γ=0.85 и: (а) α=0; (б) α=0.2

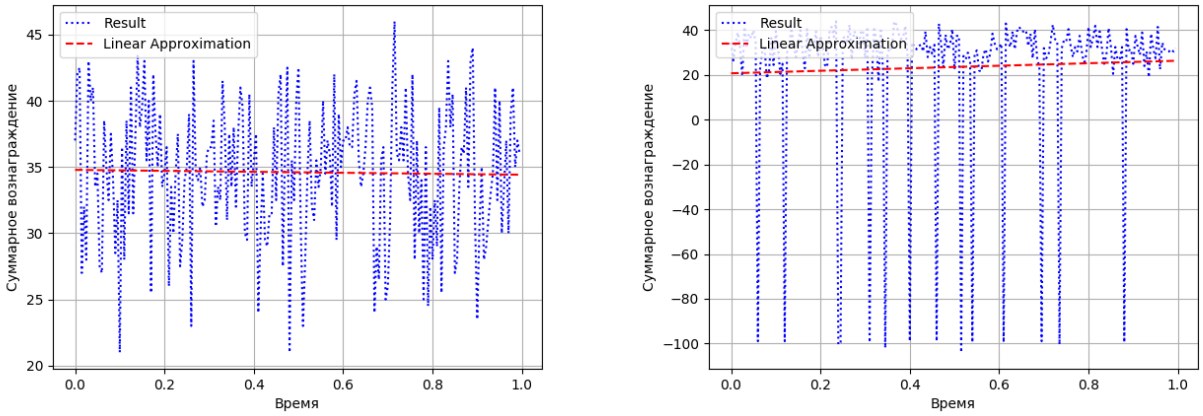


(а) (б)

Рисунок 16: Зависимость суммарного вознаграждения от пройденного времени при N=5000, γ=0.85 и: (а) α=0.4; (б) α=0.6

Из графиков выше можно заметить, что аппроксимирующая зависимость лежит в зоне отрицательных значений вознаграждений. Можно задаться вопросом: "Адекватно ли работает алгоритм?". Да, алгоритм работает адекватно, так как в - ажно смотреть не только на то, сколько

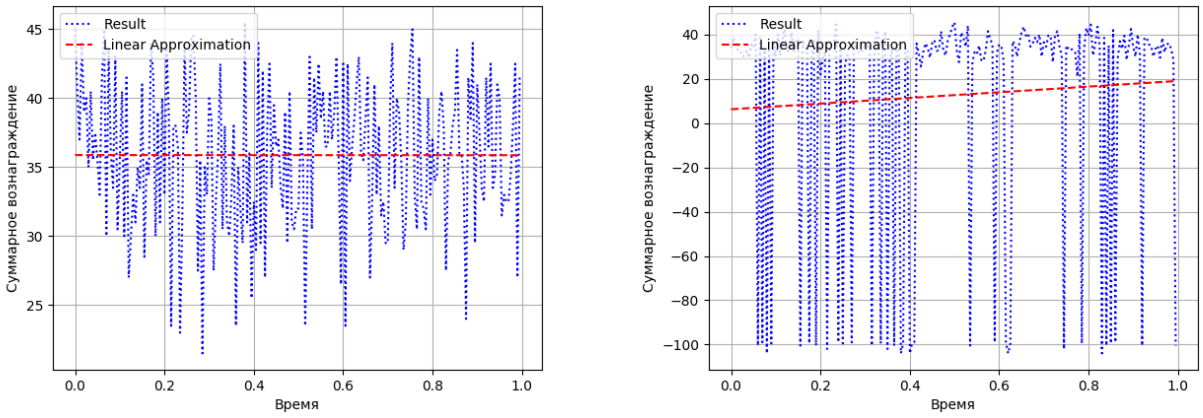
получает агент при обучении, а ещё на то, улучшается ли тенденция при изменении параметров в 'нужную' сторону, если мы ожидаем увидеть там улучшение. Подбор нужных штрафов и вознаграждений довольно трудная задача, поэтому в данном эксперименте я больше ориентировался на тенденцию улучшения. Помимо увеличения α, увеличим и N до 50000.



(а) (б)

Рисунок 17: Зависимость суммарного вознаграждения от пройденного времени при N=50000, γ=0.85 и: (а) α=0.4; (б) α=0.6

Как можно заметить, улучшение произошло значительное: зависимость вышла из зоны отрицательных значений. Можно задаться вопросом: "Почему для данного алгоритма адекватный результат получился для N=50000, а в предыдущем пункте насыщение произошло при N=5000?". Ответ на этот вопрос достаточно прост. Так как мы усложнили модель с помощью учета слоя, то агенту требуется больше времени на обучение. Дальнейшие эксперименты будем проводить с N=50000.

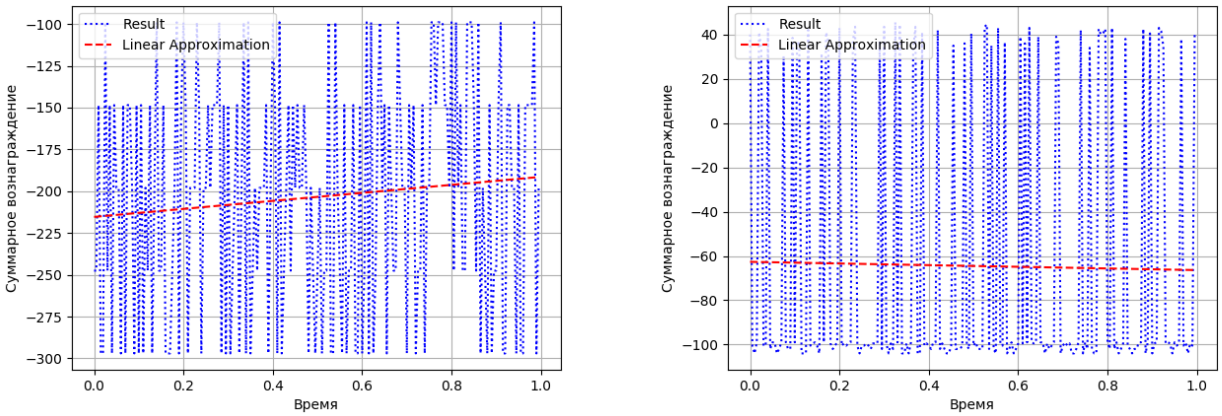


(а) (б)

Рисунок 18: Зависимость суммарного вознаграждения от пройденного времени при N=50000, γ=0.85 и: (а) α=0.8; (б) α=1.0

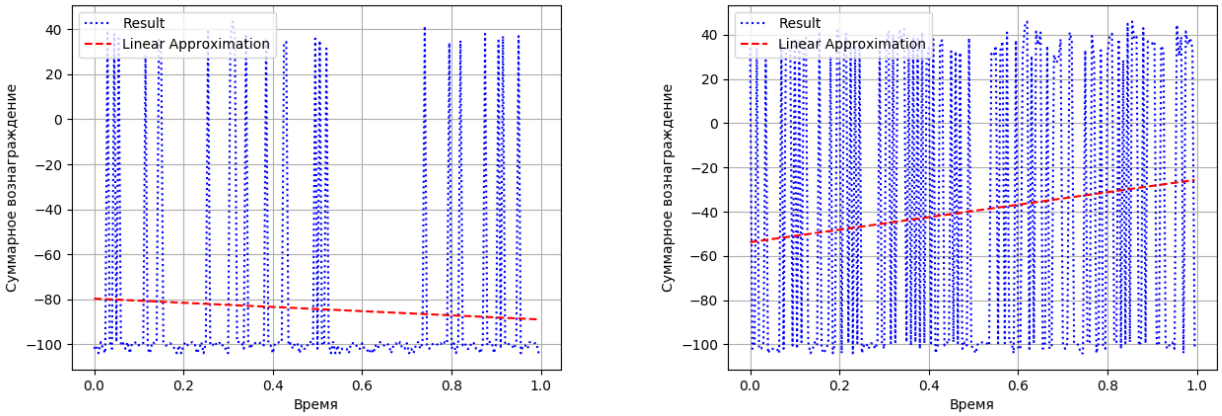
Из приведенных графиков можно сделать вывод, что при увеличении α тенденция улучшается, но можно заметить, что при α=1.0 зависимость выглядит хуже, чем для α=0.8. Поэтому, необходимо пояснить, что алгоритм не всегда работает одинаково при каждом повторном запуске. Существуют отклонения, похожие на те, что продемонстрированы выше. Так как эта модель учитывает больше факторов, чем предыдущая, то данные флуктуации требуют дополнительных исследований, но в данной работе я не буду останавливаться на этом и для дальнейшего обучения зафиксирую α=0.77 (при данном параметры были получены адекватные результаты).

Теперь проведем ряд экспериментов с фиксированными α=0.77 и N=50000. А варьировать будем γ.



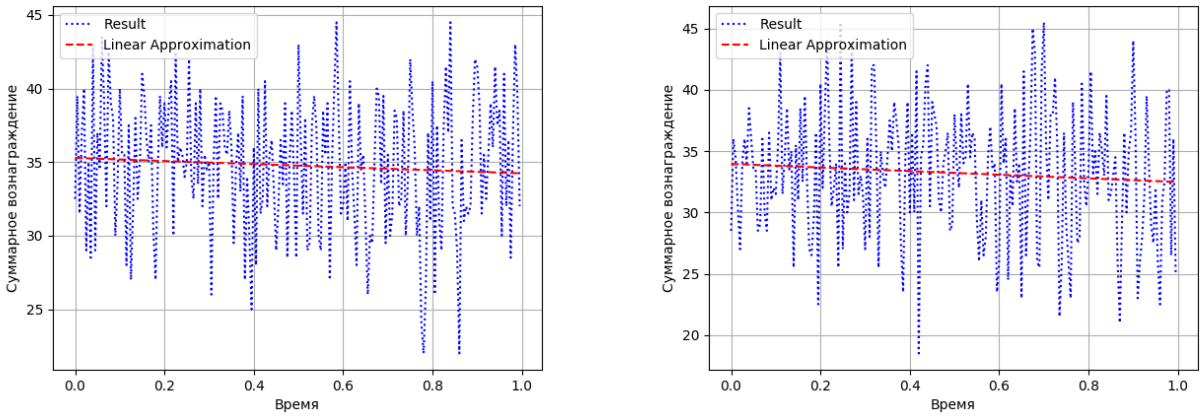
(а) (б)

Рисунок 19: Зависимость суммарного вознаграждения от пройденного времени при N=50000, α=0.77 и: (а) γ=0; (б) γ=0.2



(а) (б)

Рисунок 20: Зависимость суммарного вознаграждения от пройденного времени при N=50000, α=0.77 и: (а) γ=0.4; (б) γ=0.6



(а) (б)

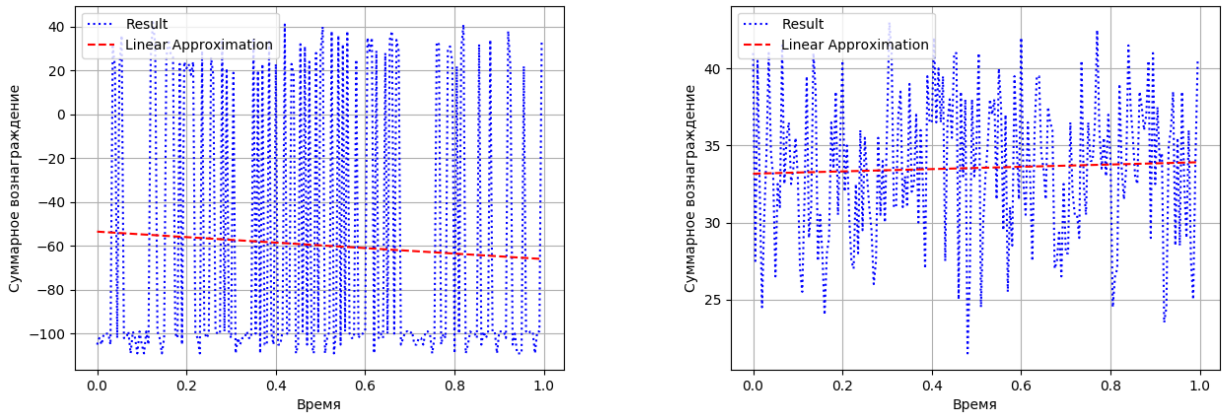
Рисунок 21: Зависимость суммарного вознаграждения от пройденного времени при N=50000, α=0.77 и: (а) γ=0.8; (б) γ=1.0

Можно заметить, что с увеличением γ, результаты при обучении получаются лучше с каждым разом. Но опять же, алгоритм выходит на насыщение. Оптимальный параметр γ в данной модели равняется 0.86.

5.3 Неудачные эксперименты

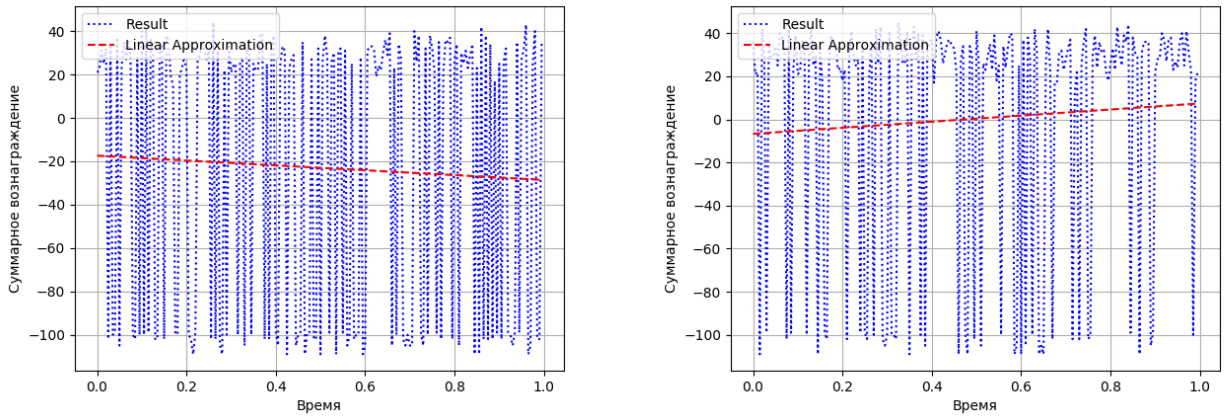
При исследовании алгоритма иногда попадались не совсем адекватные результаты эксперимента. Многое зависит от того, как ввести систему штрафов при обучении. Рассмотрим поведение алгоритма из пункта 6.2, если расход топлива на каждом слое вводится следующим образом: (а) нулевой слой соответствует вознаграждению -n, а (n-1)-ый слой вознаграждению -1 (шаг 1 между слоями); (б) нулевой слой соответствует вознаграждению -(n+1)/2, а (n-1)-ый слой вознаграждению -1 (шаг 0.5 между слоями). Рассмотрим два случая:

- При оптимальном α, N=50000 и γ чуть выше оптимального .
- При оптимальном γ, N=50000 и α чуть выше оптимального .



(а) (б)

Рисунок 22: Зависимость суммарного вознаграждения от пройденного времени при N=50000, α=0.63, γ=0.8. (а) Случай при шаге 1. (б) Случай при шаге 0.5



(а) (б)

Рисунок 23: Зависимость суммарного вознаграждения от пройденного времени при N=50000, α=0.8, γ=0.73. (а) Случай при шаге 1. (б) Случай при шаге 0.5

Из графиков выше видно, что в случае (а) результаты заметно хуже. Если исследовать визуализацию алгоритма, то можно заметить, почему это происходит. Агент стремится попасть на самый разреженный слой (верхний), где затраты по топливу на перемещение минимальные. Как только он туда попадает, он начинает перемещаться только по двум клеткам (агент заходит в тупик). Агент забывает о своей истинной цели и просто теряет очки.

ЗАКЛЮЧЕНИЕ

В результате работы был разработан алгоритм для поиска пути в лабиринте. Задаче была поставлена в соответствие другая физическая задача - доставка дроном объектов из точки <А> в точку <Б>. Алгоритм был реализован на основе Q-learning с помощью языка Python. Были получены и интерпретированы графики зависимостей суммарного вознаграждения агента при обучении в среде от времени обучения. На основе исследования алгоритма, были получены оптимальные параметры для обучения α (отвечает за темп обучения) и γ (дисконтирующий множитель) в двух моделях:

- Модель, в которой не учитываются физические параметры, такие как сопротивление ветра, разреженность, давление и плотность воздуха.
- Модель, в которой учитывается разреженность воздуха.

Подводя итоги работы, можно сказать, что применение машинного обучения, нейронных сетей, в особенности Q-learning, в теории управления является одной из самых перспективных областей. В данной работе были наглядно продемонстрированы возможности искусственного интеллекта к самообучению, используя метод «проб и ошибок».

Дальнейшее развитие данной задачи выглядит весьма многообещающим:

- Усложнить модель. Внести в неё более полный учет параметров воздуха, таких как давление, плотность и сопротивление ветра.
- Реализовать алгоритм Deep Q-learning, который является более эффективным по сравнению с Q-learning.
- Реализовать метод сопряженных градиентов для более эффективного поиска параметров Q-learning.

- Использовать реальные физические данные атмосфер, которые бы преобразовывались с помощью функционала в пространство вознаграждений для агента.
- Реализовать 3D-визуализацию перемещения агента в среде, где будут отмечены области, в которых агенту перемещаться менее затратно, чем в других.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] - Сайт англоязычной википедии. Интернет ресурс: https://en.wikipedia.org/wiki/Machine_learning

[2] - Саттон Р., Барто Э. Обучение с подкреплением – Бином. Лаборатория знаний, 2012. – 400 с.

[3] - Князятов С.А., Малинецкий Г.Г., Решение задачи распознавания блефа в игре «верю – не верю» с помощью алгоритмов обучения с подкреплением // Препринты ИПМ им. М.В.Келдыша. 2018. No 170. 21 с.

[4] - Richard S. Sutton and Andrew G.Barto - Reinforcement Learning: An Introduction

[5] - Библиотека gym от OpenAi. Интернет ресурс: https://gym.openai.com

[6] - Обучение с подкреплением на языке Python. Интернет ресурс: https://habr.com/ru/company/piter/blog/434738/

ПРИЛОЖЕНИЕ

Листинг 1: labyrinth.py

```
1  import sys
2  from contextlib import closing
3  from six import StringIO
4  import os
5  import time
6  from gym import utils
7  from gym.envs.toy_text import discrete
8  #from gym.envs.toy_text import map_generation as mg
9  import map_generation as mg
10 import numpy as np
11
12 x_size = 5
13 y_size = x_size
14 z_size = x_size
15
16 #functional that sets of paramenters in accordance with rewa
17 rd
18 def functional(inPut):
19     out = 0
20     for val in inPut:
21         out += inPut[val]
22     return out
23
24 lay_reward = {}
25 counter = -1
26 fine_step = 0.5
27 for a in range(z_size-1,-1,-1):
28     lay_reward[a] = counter
```

```
29      counter = counter - fine_step
30
31  cell_reward = np.zeros((x_size, y_size, z_size))
32  for z, lay in enumerate(cell_reward):
33      struct = {
34          'Lay Reward': lay_reward[z],
35          'Density': 0,
36          'Wind': 0,
37          }
38      for y, col in enumerate(lay):
39          for x, row in enumerate(col):
40              result = functional(struct)
41              cell_reward[z][y][x] = result
42
43  taxi_map = mg.Map()
44  taxi_map.map_creation(x_size, y_size, z_size)
45  d = taxi_map.loc_creation()
46
47  colors = ['R', 'G', 'Y', 'B']
48
49  def get_minimum(new_location, limit):
50      boom = False
51      if new_location > limit:
52          return limit, True
53      return new_location, boom
54
55  def get_maximum(new_location, limit):
56      boom = False
57      if new_location < limit:
58          return limit, True
```

```
59      return new_location, boom
60
61  class TaxiEnv(discrete.DiscreteEnv):
62      metadata = {'render.modes': ['human', 'ansi']}
63
64      def __init__(self):
65          self.desc = np.asarray(taxi_map.MAP, dtype='c')
66          self.locs = []
67
68          for c in colors:
69              self.locs.append(d[c])
70
71          num_states = taxi_map.volume()*5*4
72          num_rows = y_size
73          num_columns = x_size
74          num_lays = z_size
75          max_row = num_rows - 1
76          max_col = num_columns - 1
77          max_lay = num_lays - 1
78
79          initial_state_distrib = np.zeros(num_states)
80          num_actions = 8
81          P = {state: {action: []
82                       for action in range(num_actions)} for state in
                             range(num_states)}
83          # Running through layers (Z-axis)
84          for lay in range(num_lays):
85              # Running through rows (Y-axis)
86              for row in range(num_rows):
87                  # Running through columns (X-axis)
```

```
88                  for col in range(num_columns):
89                      # +1 for being inside taxi; below
90                      for obj_idx in range(len(self.locs) + 1):
91                          for dest_idx in range(len(self.locs)):
92                              state = self.encode(lay, row, col,
                                    obj_idx, dest_idx)
93                              if obj_idx < 4 and obj_idx != dest_idx:
94                                  initial_state_distrib[state] += 1
95                              for action in range(num_actions):
96                                  new_lay, new_row, new_col,
                                        new_obj_idx = lay, row, col,
                                        obj_idx
97                                  #reward = lay_reward[lay]
98                                  reward = cell_reward[lay][col][row]
99                                  done = False
100                                 taxi_loc = (lay, row, col)
101                                 # 0 - south
102                                 if action == 0:
103                                     new_row, boom = get_minimum(row
                                            + 1, max_row)
104                                     if boom:
105                                         reward = -10
106                                 # 1 - north
107                                 elif action == 1:
108                                     new_row, boom = get_maximum(row
                                            - 1, 0)
109                                     if boom:
110                                         reward = -10
111                                 # 2 - east
112                                 if action == 2 and self.desc[lay, 1
```

```
                + row, 2 * col + 2] == b":":
113                 new_col, boom = get_minimum(col
                        + 1, max_col)
114                 if boom:
115                     reward = -10
116             # 3 - west
117             elif action == 3 and self.desc[lay,
                    1 + row, 2 * col] == b":":
118                 new_col, boom = get_maximum(col
                        - 1, 0)
119                 if boom:
120                     reward = -10
121             # 4 - move up
122             if action == 4:  # and self.desc[lay
                    , 1 + row, 2 * col + 2] == b":":
123                 new_lay, boom = get_minimum(lay
                        + 1, max_lay)
124                 if boom:
125                     reward = -10
126             # 5 - move down
127             elif action == 5:  # and self.desc[
                    lay, 1 + row, 2 * col] == b":":
128                 new_lay, boom = get_maximum(lay
                        - 1, 0)
129                 if boom:
130                     reward = -10
131             # 6 - pickup
132             elif action == 6:  # pickup
133                 if (obj_idx < 4) and (taxi_loc
                        == self.locs[obj_idx]):
```

```
134                                         new_obj_idx = 4
135                                     else:  #object not at location
136                                         reward = -10
137                                 # 7 - dropoff
138                                 elif action == 7:  # dropoff
139                                     if (taxi_loc == self.locs[
                                            dest_idx]) and obj_idx == 4:
140                                         new_obj_idx = dest_idx
141                                         done = True
142                                         reward = 50
143                                     elif (taxi_loc in self.locs) and
                                            obj_idx == 4:
144                                         new_obj_idx = self.locs.
                                                index(taxi_loc)
145                                     else:  # dropoff at wrong
                                            location
146                                         reward = -20
147                                 new_state = self.encode(
148                                     new_lay, new_row, new_col,
                                            new_obj_idx, dest_idx)
149                                 P[state][action].append(
150                                     (1.0, new_state, reward, done))
151         initial_state_distrib /= initial_state_distrib.sum()
152         discrete.DiscreteEnv.__init__(
153             self, num_states, num_actions, P, initial_state_distrib)
154
155     def encode(self, taxi_lay, taxi_row, taxi_col, obj_loc, dest_idx
            ):
156         i = taxi_lay
157         i *= z_size
```

```
158         i += taxi_row
159         i *= y_size
160         i += taxi_col
161         i *= x_size
162         i += obj_loc
163         i *= 4
164         i += dest_idx
165         return i
166 
167     def decode(self, i):
168         out = []
169         out.append(i % 4)
170         i = i // 4
171         out.append(i % x_size)
172         i = i // x_size
173         out.append(i % y_size)
174         i = i // y_size
175         out.append(i % z_size)
176         i = i // z_size
177         out.append(i)
178         assert 0 <= i < 5
179         return reversed(out)
180 
181     def render(self, mode='human'):
182         outfile = StringIO() if mode == 'ansi' else sys.stdout
183 
184         out = self.desc.copy().tolist()
185         out = [[[c.decode('utf-8') for c in line] for line in lay]
               for lay in out]
186         taxi_lay, taxi_row, taxi_col, obj_idx, dest_idx = self.
```

```
            decode(self.s)
187         def ul(x): return "_" if x == " " else x
188         if obj_idx < 4:
189             out[taxi_lay + 1][1 + taxi_row][2 * taxi_col + 1] =
                    utils.colorize(
190                 out[taxi_lay + 1][1 + taxi_row][2 * taxi_col + 1], '
                        yellow', highlight=True)
191             pk, pj, pi = self.locs[obj_idx]
192             out[pk + 1][pi + 1][2 * pj + 1] = utils.colorize(out[pk
                    + 1][pi + 1][2 * pj + 1], 'blue', bold=True)
193         else: #agent with object
194             out[taxi_lay + 1][1 + taxi_row][2 * taxi_col + 1] =
                    utils.colorize(
195                 ul(out[taxi_lay + 1][1 + taxi_row][2 * taxi_col +
                        1]), 'green', highlight=True)
196
197         dk, di, dj = self.locs[dest_idx]
198         out[dk + 1][di + 1][2 * dj + 1] = utils.colorize(out[dk +
                1][di + 1][2 * dj + 1], 'magenta')
199         os.system('clear')
200         print("AGENT:")
201         print("LAY: ", taxi_lay)
202         print("ROW: ", taxi_row)
203         print("COLUMN: ", taxi_col)
204         outfile.write("\n".join(["".join(row) for row in out[
                taxi_lay + 1]]) + "\n")
205         time.sleep(5)
206         #print all lays below
207         #print("ALL LAYS")
208         #for item in out:
```

```
209         #   outfile.write("\n".join(["".join(row) for row in item])
                + "\n")
210         if self.lastaction is not None:
211             outfile.write("  ({})\n".format(["South", "North", "East
                    ", "West", "MoveUp", "MoveDown", "Pickup", "Dropoff"
                    ][self.lastaction]))
212         else:
213             outfile.write("\n")
214         if mode != 'human':
215             with closing(outfile):
216                 return outfile.getvalue()
217
218 print('labyrinth.py launched successfully')
219 print('\n')
```

Листинг 2: main.py

```
1  import numpy as np
2  import scipy as sp
3  import gym
4  import random
5  import time
6  import os
7  import matplotlib
8  import matplotlib.pyplot as plt
9
10 env = gym.make("Labyrinth")
11 env.render()
12
13 action_size = env.action_space.n
14 state_size = env.observation_space.n
```

```
15
16 qtable = np.zeros((state_size, action_size))
17
18 # Total episodes
19 total_episodes = 50000
20 # Total test episodes
21 total_test_episodes = 200
22 # Max steps per episode
23 max_steps = 99
24
25 #0 < ... <= 1
26 # Learning rate
27 alpha = 0.63
28 # Discounting rate
29 gamma = 0.75
30
31 # Exploration parameters
32 # Exploration rate
33 epsilon = 1.0
34
35 # Exploration probability at start
36 max_epsilon = 1.0
37 # Minimum exploration probability
38 min_epsilon = 0.01
39 decay_rate = 0.01
40
41 def plotting(action, step, done, info):
42     print("ACTION: ", action)
43     print("STEP: ", step)
44     print("DONE: ", done)
```

```
45      print("INFO: ", info)
46      print("TOTAL REWARD: ", total_rewards)
47
48  score_ov_time =[]
49  x_steps = []
50  for episode in range(total_episodes):
51      state = env.reset()
52      step = 0
53      done = False
54
55      for step in range(max_steps):
56          exp_exp_tradeoff = random.uniform(0,1)
57
58          #If this number > greater than epsilon --> exploitation
59          #(taking the biggest Q value for this state)
60          if exp_exp_tradeoff > epsilon:
61              action = np.argmax(qtable[state,:])
62
63          # doing a random choice --> exploration
64          else:
65              action = env.action_space.sample()
66
67          new_state, reward, done, info = env.step(action)
68
69          qtable[state, action] = qtable[state, action] + alpha*(
                reward + gamma *
70                                      np.max(qtable[new_state,:]) -
                                            qtable[state, action])
71          state = new_state
72
```

```
73          if done == True:
74              break
75
76      epsilon = min_epsilon + (max_epsilon - min_epsilon)*np.exp(-
            decay_rate*episode)
77
78 env.reset()
79 rewards = []
80
81 for episode in range(total_test_episodes):
82      state = env.reset()
83      x_steps.append(episode / total_test_episodes)
84      step = 0
85      done = False
86      total_rewards = 0
87      #os.system("clc||clear")
88      #print("****************")
89      #print("EPISODE: ", episode)
90      for step in range(max_steps):
91          #FOR VISUALIZATION
92          #env.render()
93          #Take the action (index) that have the maximum
94          #expected future reward given that state
95          action = np.argmax(qtable[state,:])
96          new_state, reward, done, info = env.step(action)
97          total_rewards += reward
98          #plotting(action, step, done, info)
99          if done:
100             rewards.append(total_rewards)
101             #print ("Score", total_rewards)
```

```
102                 break
103             state = new_state
104         score_ov_time.append(total_rewards)
105 env.close()
106 print ("Score over time: " +  str(sum(rewards)/total_test_episodes))
107
108 # For graphics
109 poly = sp.polyfit(x_steps, score_ov_time, 1)
110 pol_1d = sp.poly1d(poly)
111 line_1, line_2 = plt.plot(x_steps, score_ov_time, 'b:', x_steps,
       pol_1d(x_steps), 'r—')
112 plt.legend((line_1, line_2), (u'Result', u'Linear Approximation'),
       loc='upper left')
113 plt.xlabel('Time', fontsize=10)
114 plt.ylabel('Result Reward', fontsize=10)
115 graphic_name = 'test'
116 #graphic_name = 'withoutlayreward_N' + str(total_episodes)+
117 'gamma' + str(gamma)+ 'alpha'+str(alpha)+'epsilon'+str(epsil
118 on)
119 plt.grid()
120 plt.savefig(graphic_name)
121 plt.show()
```

Листинг 3: map_generation.py

```
1 colors = ['R', 'G', 'Y', 'B']
2
3 class Map:
4     def __init__(self):
5         self.size = []
6         self.MAP = []
```

```
7           self.x_size = 0
8           self.y_size = 0
9           self.z_size = 0
10
11      def map_creation(self, x_size, y_size, z_size):
12          self.x_size = x_size
13          self.y_size = y_size
14          self.z_size = z_size
15
16          for k in range(0, self.z_size+2):
17              map_layer = []
18              for j in range(0, self.y_size+2):
19                  new_str = ""
20                  for i in range(0, 2*self.x_size+1):
21                      if (j == 0) or (j == self.y_size+1):
22                          if(i == 0) or (i == 2*self.x_size):
23                              new_str += "+"
24                          else:
25                              new_str += "-"
26                      if (k == 0) or (k == self.z_size+1):
27                          if (j != 0) and (j != self.y_size + 1):
28                              if(i == 0) or (i == 2*self.x_size):
29                                  new_str += "|"
30                              else:
31                                  if (i+1) % 2 == 0:
32                                      new_str += "-"
33                                  else:
34                                      new_str += "|"
35                      if (k != 0) and (k != self.z_size+1):
36                          if (j != 0) and (j != self.y_size + 1):
```

```
37                              if(i == 0) or (i == 2*self.x_size):
38                                  new_str += "|"
39                              else:
40                                  if (i+1) % 2 == 0:
41                                      new_str += " "
42                                  else:
43                                      new_str += ":"
44                  map_layer.extend([new_str])
45              self.MAP += [map_layer]
46
47      def loc_creation(self):
48          row_size = self.y_size
49          col_size = self.x_size
50          h_size = self.z_size
51          count = -1
52          randomR = Map.getRandom(self, h_size, row_size, col_size)
53          randomG = Map.getRandom(self, h_size, row_size, col_size)
54          randomY = Map.getRandom(self, h_size, row_size, col_size)
55          randomB = Map.getRandom(self, h_size, row_size, col_size)
56          d = dict(R=randomR, G=randomG, B=randomB, Y=randomY)
57          for item in self.MAP:
58              count = count + 1
59              if count == 0 or count == len(self.MAP) - 1:
60                  continue
61              for c in colors:
62                  lay, row, col = d[c]
63                  if lay == count:
64                      item[row] = item[row][:2*col+1] + c + item[row
                            ][2*col+1 + 1:]
65          return d
```

```
66
67      def getRandom(self, h_size, row_size, col_size):
68          import random
69          lay = random.randrange(1, h_size, 1)
70          row = random.randrange(1, row_size, 1)
71          column = random.randrange(1, col_size, 1)
72          return lay, row, column
73
74      def printmap(self, layer):
75          if layer < 0:
76              count = -1
77              for item in self.MAP:
78                  count = count + 1
79                  print(count)
80                  for it in item:
81                      print(it)
82          else:
83              for row in range(0, self.y_size+2):
84                  print(self.MAP[layer][row])
85
86      def shape(self):
87          return self.x_size+2, self.y_size+2, self.z_size+2
88
89      def volume(self):
90          return self.x_size*self.y_size*self.z_size
```

Листинг 4: discrete.py

```
1 import numpy as np
2 from gym import Env, spaces
3 from gym.utils import seeding
```

```
4
5  def categorical_sample(prob_n, np_random):
6      """
7      Sample from categorical distribution
8      Each row specifies class probabilities
9      """
10     prob_n = np.asarray(prob_n)
11     csprob_n = np.cumsum(prob_n)
12     return (csprob_n > np_random.rand()).argmax()
13
14
15 class DiscreteEnv(Env):
16     """
17     Has the following members
18     - nS: number of states
19     - nA: number of actions
20     - P: transitions (*)
21     - isd: initial state distribution (**)
22
23     (*) dictionary dict of dicts of lists, where
24       P[s][a] == [(probability, nextstate, reward, done), ...]
25     (**) list or array of length nS
26     """
27     def __init__(self, nS, nA, P, isd):
28         self.P = P
29         self.isd = isd
30         self.lastaction = None # for rendering
31         self.nS = nS
32         self.nA = nA
33
```

```
34          self.action_space = spaces.Discrete(self.nA)
35          self.observation_space = spaces.Discrete(self.nS)
36
37          self.seed()
38          self.s = categorical_sample(self.isd, self.np_random)
39          print("self.s in discrete.py", self.s)
40          self.lastaction=None
41
42      def seed(self, seed=None):
43          self.np_random, seed = seeding.np_random(seed)
44          return [seed]
45
46      def reset(self):
47          self.s = categorical_sample(self.isd, self.np_random)
48          self.lastaction = None
49          return self.s
50
51      def step(self, a):
52          transitions = self.P[self.s][a]
53          i = categorical_sample([t[0] for t in transitions], self.
                np_random)
54          p, s, r, d= transitions[i]
55          self.s = s
56          self.lastaction = a
57          return (s, r, d, {"prob" : p})
```