

Задача 1: Статистика текста

Прежде чем нейросеть научиться писать стихи или отвечать на вопросы, ей необходимо понять базовые закономерности языка. Анализ длины слов — первый шаг в подготовке данных для обучения языковых моделей. Эта задача погружает вас в фундаментальные принципы обработки естественного языка (NLP), которые лежат в основе таких систем, как ChatGPT.

Условие задачи

Дан текстовый файл `text.txt`, содержащий произвольный текст на русском и английском языках. Необходимо:

1. Извлечь все слова из текста согласно правилам ниже.
2. Рассчитать три статистические метрики для длин извлеченных слов:
 - Среднее арифметическое (с точностью 6 знаков после запятой)
 - Медиану (нижнюю медиану при четном числе элементов)
 - Моду (с разрешением конфликтов в пользу наименьшего значения)
3. Сохранить результаты в два файла: `words.txt` — список извлеченных слов - `result.txt` — вычисленные метрики
4. Заархивировать оба файла в `task1.zip`.



Теоретическая часть

Что такое слово?

Слово — непрерывная последовательность буквенных символов, где:

- **Буквенные символы:**

- Латинские буквы: a-z, A-Z
- Кириллические буквы: а-я, А-Я, включая ё и Ё
- **Разделители:** все остальные символы (цифры, знаки препинания, пробелы, спецсимволы).

Примеры:

- "Привет, world123! Это-тест." → Привет, world, Это, тест
- "Hello-world" → Hello, world
- "abc123def" → abc, def

Статистические метрики

1. Среднее арифметическое

Для списка длин слов $L = [l_1, l_2, \dots, l_n]$:

$$\bar{l} = \frac{1}{n} \sum_{i=1}^n l_i$$

Требование: результат выводится с точностью 6 знаков после запятой.

2. Медиана

Значение в середине упорядоченного списка. При четном n берется **нижняя медиана** (элемент с индексом $(n - 1) // 2$ в отсортированном списке).

Пример:

$[2, 3, 5, 8] \rightarrow n = 4$, индекс $(4 - 1) // 2 = 1 \rightarrow$ медиана = 3.

3. Мода

Наиболее частая длина. При равенстве частот выбирается **наименьшее** значение.

Пример:

$[2, 3, 2, 4, 3, 4] \rightarrow$ частоты: $2 \rightarrow 2, 3 \rightarrow 2, 4 \rightarrow 2 \rightarrow$ мода = 2.

Входные данные

- Файл `text.txt` в кодировке UTF-8 с произвольным текстом.

Выходные данные

Архив task1.zip, содержащий два файла:

1. words.txt
Каждая строка — слово из исходного текста в порядке появления.

Пример (для текста "Hello, world! Это тест123."):

```
Hello
world
Это
тест
```

2. result.txt
Три строки с вычисленными метриками:

```
3.833333
4
4
```

Критерии оценивания

№	Критерий	Баллы
1.1	Корректное извлечение слов (учет разделителей)	25
1.2	Правильное вычисление среднего и медианы	35
1.3	Корректный выбор моды (включая обработку равных частот)	40
	Итого	100

Технические требования

- Используйте только стандартные библиотеки Python (re, collections и др.).
- Среднее арифметическое должно иметь точность $\pm 1e-6$.
- Медиана и мода — целые числа.
- Все файлы должны быть в кодировке UTF-8.

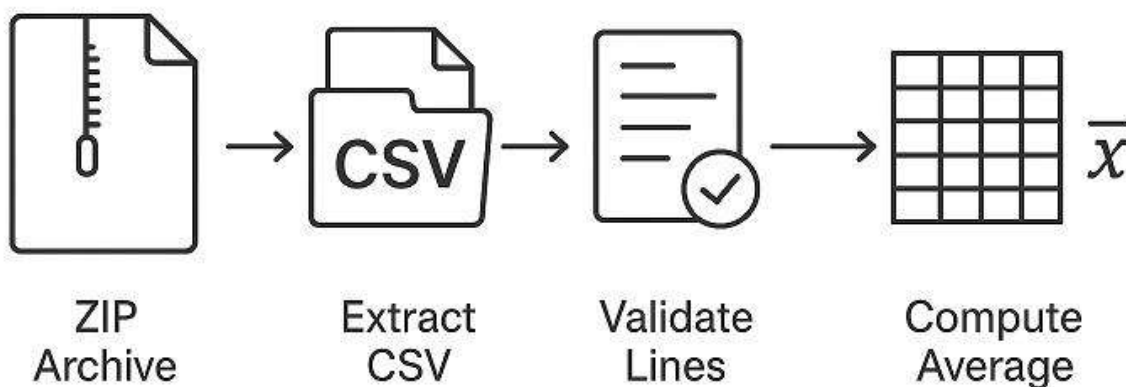
Задача 2: Чистильщик данных

В реальном мире данные редко бывают идеальными. Как археологи, вычищающие древние артефакты от пыли, дата-сайентисты тратят большую часть времени на очистку “грязных” данных. Пропущенные значения, ошибки датчиков, мусорные символы — всё это мешает анализу. Сегодня вы научитесь превращать хаотичные данные в чистые и пригодные для работы числа. Без этого ни одна модель искусственного интеллекта не сможет дать точный результат!

Условие задачи

Дан ZIP-архив `data.zip`, содержащий файлы различных форматов. Необходимо:

1. Найти все CSV-файлы в архиве (регистр расширения не важен: `.csv`, `.CSV` и т.д.).
2. Из каждой строки CSV-файлов извлечь корректные данные, удовлетворяющие правилам ниже.
3. Сохранить все корректные строки в файл `valid_lines.txt`.
4. Вычислить среднее арифметическое для каждого из трёх столбцов по всем корректным строкам и записать результаты в файл `result.txt` с точностью 6 знаков после запятой.
5. Заархивировать оба файла в `task2.zip`.



Теоретическая часть

Что такое CSV?

CSV (Comma-Separated Values) — текстовый формат хранения табличных данных, где значения разделены запятыми. Для этой задачи действуют строгие правила форматирования.

Правила корректной строки CSV:

- Каждая строка содержит ровно три вещественных числа
- Числа разделены запятыми **без пробелов**
- Все три значения должны быть корректными вещественными числами в десятичном формате

Примеры корректных строк:

1.5,2.3,4.1
-0.5,10,3.14
0.0,0.0,0.0

Примеры некорректных строк:

1.5, 2.3, 4.1 ← пробелы вокруг запятых
1.5,2.3 ← меньше трёх значений
1e3,2.3,4.1 ← экспоненциальная запись
abc,2.3,4.1 ← нечисловое значение
1.5,,4.1 ← пропущенное значение

Что такое вещественное число в задаче?

Допустимые форматы:

- Целые числа: 42, -17, 0
- Дробные числа: 3.14, -0.5, 100.0
- Числа с нулевой дробной частью: 0.0, -0.0

Запрещено использовать:

- Экспоненциальную запись (1e3, 2.5e-2)
- Спецсимволы (inf, NaN, null)
- Числа с пробелами или другими разделителями

Среднее арифметическое по столбцу

Для каждого столбца j ($j = 1, 2, 3$) среднее вычисляется как:

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{ij},$$

где N — общее количество корректных строк, x_{ij} — значение в i -й строке и j -м столбце.

Входные данные

Архив data.zip в кодировке UTF-8, содержащий: - Файлы с расширением .csv (регистр не важен) в корне архива - Другие файлы (.txt, .json, без расширения) и файлы во вложенных папках **игнорируются**

Пример структуры архива:

```
data.zip
├── file1.csv      ← обрабатываем
├── file2.txt      ← игнорируем
├── data.CSV       ← обрабатываем (регистр не важен)
├── subfolder/
│   └── more.csv   ← игнорируем (вложенные папки не обрабатываются)
```

Выходные данные

Архив task2.zip, содержащий два файла:

3. valid_lines.txt
 - Каждая строка: три корректных числа в формате {число1}, {число2}, {число3}
 - Порядок строк: сохраняется порядок обработки файлов и строк внутри них
 - Кодировка UTF-8
4. result.txt
 - Одна строка с тремя числами через пробел: {mean_col1} {mean_col2} {mean_col3}
 - Точность: ровно 6 знаков после запятой
 - Пример: -3.000000 1.000000 34.380000

Критерии оценивания

№	Критерий	Баллы
2.1	Корректная фильтрация CSV-файлов и отсеивание некорректных строк (формат файлов, количество столбцов, тип данных)	50
2.2	Правильное вычисление средних по столбцам с требуемой точностью	50
Итого		100

Технические требования

- Используйте только стандартные библиотеки Python (zipfile, csv, re и др.)
- Числовой допуск для средних: $\pm 10^{-6}$
- Порядок обработки файлов может быть произвольным
- Все файлы должны быть в кодировке UTF-8
- При обработке файлов учитывайте только файлы в корневой директории архива

Задача 3: Детектор аномалий во временном ряде

Каждую секунду миллионы датчиков собирают данные о работе критически важных систем: от сердечного ритма пациентов до состояния ядерных реакторов. Раннее обнаружение аномалий позволяет предотвратить катастрофы и спасти жизни. В основе таких систем лежит анализ локального контекста — сравнение текущего значения с характерным поведением системы в ближайшем окружении. Эта задача погружает вас в фундаментальные принципы временных рядов и статистического контроля качества, которые используются в медицине, промышленности и научных исследованиях.

Условие задачи

Дан временной ряд измерений в файле `series.txt`. Для каждого элемента ряда с индексом i (начиная с 0) необходимо определить, является ли он аномалией относительно своего локального окружения.

Аномалией считается точка x_i , если одновременно выполняются условия:

1. Для неё можно построить полное скользящее окно размером 11 точек (т.е. $5 \leq i \leq N - 6$, где N — общее количество точек).
2. Её отклонение от локального среднего превышает удвоенное стандартное отклонение: $|x_i - \mu_i| > 2 \cdot \sigma_i$.
3. Стандартное отклонение в окне положительно: $\sigma_i > 0$.

Требуется:

1. Для всех точек из диапазона $[5, N - 6]$ рассчитать μ_i (среднее) и σ_i (стандартное отклонение) для их окон.
2. Сформировать файл `checks.txt` с результатами вычислений.
3. Сформировать файл `result.txt` со списком индексов аномальных точек.
4. Заархивировать оба файла в `task3.zip`.

Теоретическая часть

Временной ряд

Временной ряд — последовательность значений, измеренных в последовательные моменты времени. Примеры: температура воздуха каждый час, пульс пациента каждую секунду, показания датчика вибрации.

Скользящее окно

Для анализа локального контекста вокруг точки x_i используется окно из 11 соседних значений: 5 точек слева, сама точка x_i и 5 точек справа. Индексы окна для точки i : $[i - 5, i - 4, \dots, i, \dots, i + 4, i + 5]$.

Важно: Окно можно построить только для индексов i , для которых существуют все соседи ($5 \leq i \leq N - 6$).

Среднее значение (μ)

Среднее арифметическое значений в окне:

$$\mu_i = \frac{1}{11} \sum_{k=i-5}^{i+5} x_k$$

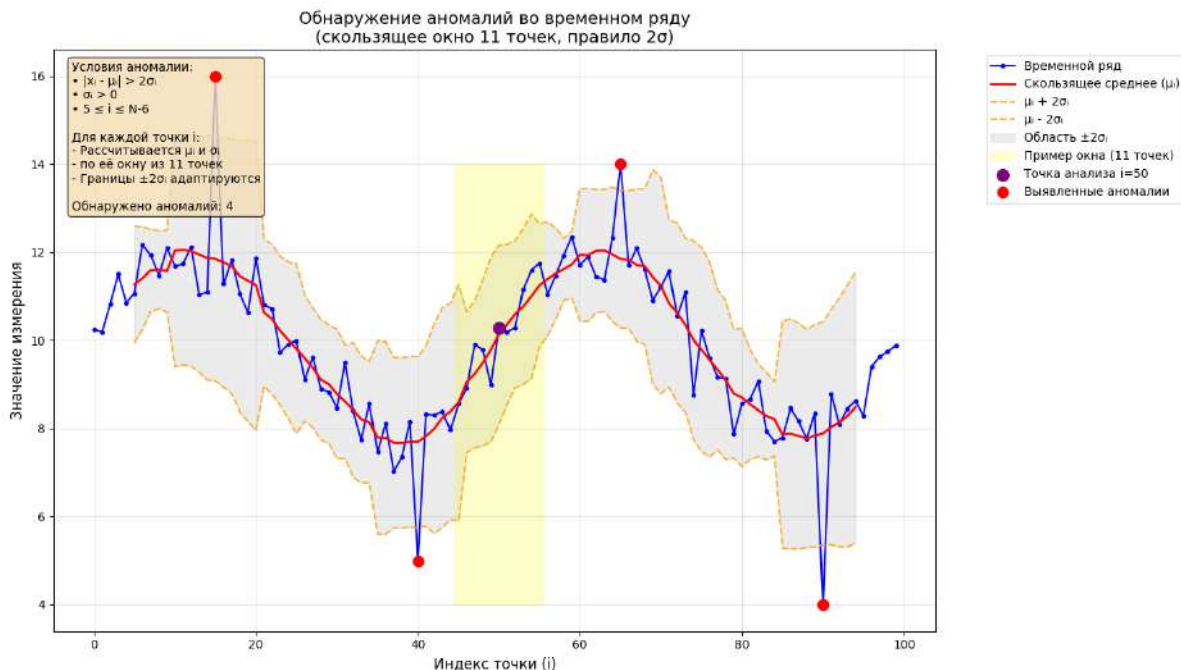
Стандартное отклонение (σ)

Мера разброса значений относительно среднего (популяционная формула):

$$\sigma_i = \sqrt{\frac{1}{11} \sum_{k=i-5}^{i+5} (x_k - \mu_i)^2}$$

Правило 2σ

В нормально распределённых данных $\sim 95\%$ значений лежат в пределах $\mu \pm 2\sigma$. Точки за этим интервалом считаются статистически значимыми аномалиями.



Входные данные

- Файл `series.txt` содержит N строк. Каждая строка — вещественное число x_i , представляющее измерение в момент времени i .
Файл в кодировке UTF-8.

Выходные данные

Архив `task3.zip`, содержащий два файла:

- `checks.txt` — для каждого индекса i из диапазона $[5, N - 6]$:
 - Строка формата: `{i} {mu_i:.8f} {sigma_i:.8f} {is_anomaly}`
 - Где `is_anomaly = 1`, если точка аномальная, иначе `0`.
- `result.txt`
 - индексы всех обнаруженных аномалий (по одному на строку, без дубликатов, в порядке возрастания). Если аномалий нет
 - файл пустой.

Пример для входных данных из 16 точек:

```
# checks.txt
5 10.00000000 0.00000000 0
6 10.00000000 0.00000000 0
7 13.63636364 11.49912255 0
8 13.63636364 11.49912255 0
9 13.63636364 11.49912255 1
10 13.63636364 11.49912255 0
```

```
# result.txt
9
```

Файлы `checks.txt` и `result.txt` должны быть заархивированы в `task3.zip`.

Критерии оценивания

№	Критерий	Баллы
3.1	Корректная реализация алгоритма анализа локального контекста (скользящее окно и вычисление статистик)	60
3.2	Верное определение аномалий по правилу 2σ и формирование выходных файлов	40
Итого		100

Технические требования

- Для стандартного отклонения использовать **популяционную формулу**.
- Погрешность вычислений μ и σ не должна превышать 10^{-6} .
- Значения μ_i и σ_i выводить с точностью не менее 8 знаков после запятой.
- Все файлы должны быть в кодировке UTF-8.
- Использовать только стандартные библиотеки Python.

Задача 4: Простая рекомендательная система

Рекомендательные системы — невидимые помощники современности. Они подбирают фильмы, музыку, товары и даже друзей. В основе многих из них лежит простая идея: «Тебе понравится то, что нравится похожим людям». В этой задаче вы реализуете классический алгоритм коллаборативной фильтрации — тот самый, с которого начиналась эпоха персонализации. Возможно, именно такие расчёты однажды помогут найти кому-то идеальную книгу... или даже решение сложной проблемы.

Условие задачи

Дан файл `ratings.txt`, содержащий матрицу оценок пользователей товарам. Ваша задача:

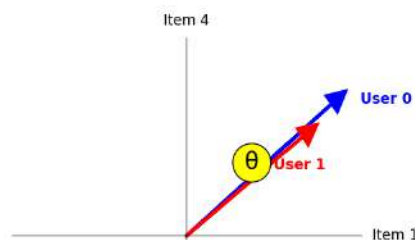
1. Для каждого пользователя $v \neq 0$ вычислить косинусную схожесть с пользователем 0 **только по общим оценённым товарам**.
2. Найти пользователя $v_{\max}$ с максимальной схожестью.
3. Среди товаров, которые пользователь 0 **не оценил** ($r_{0j} = 0$), выбрать товар, который получил **максимальную оценку** от $v_{\max}$. Если таких товаров несколько — выбрать товар с минимальным индексом.
4. Вывести результат в виде архива `task4.zip`, содержащего два файла:
 - `similarities.txt`
 - `result.txt`

User-Item Rating Matrix

User/Item	Item 1	Item 2	Item 3	Item 4	Item 5
User 0	5	3	0	4	0
User 1	4	0	5	3	2
User 2	0	4	3	5	1
User 3	3	5	4	0	3

0 = не оценено

Cosine Similarity



$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

A = вектор оценок User 0
B = вектор оценок User 1

Similarity = 1.000

Collaborative Filtering: User-Based Similarity

Теоретическая часть

Что такое матрица оценок?

Матрица оценок $R \in \mathbb{R}^{U \times I}$ — двумерная таблица, где:

- U — количество пользователей (строки),
- I — количество товаров (столбцы),
- r_{ui} — оценка пользователя u товару i (целые значения от 0 до 5, где 0 означает отсутствие оценки).

Пример матрицы 4×6:

	Товар 0	Товар 1	Товар 2	Товар 3	Товар 4	Товар 5
User 0	5	0	4	0	0	3
User 1	4	0	5	0	0	4
User 2	0	3	0	2	5	0
User 3	5	0	3	0	0	4

Косинусная схожесть пользователей

Косинусная схожесть — мера сходства двух векторов, основанная на угле между ними. Для двух пользователей u и v вычисляется **только по общим оценённым товарам** I_{uv} :

$$\text{sim}(u, v) = \frac{\sum_{j \in I_{uv}} r_{uj} \cdot r_{vj}}{\sqrt{\sum_{j \in I_{uv}} r_{uj}^2} \cdot \sqrt{\sum_{j \in I_{uv}} r_{vj}^2}}$$

где $I_{uv} = \{j \mid r_{uj} > 0 \text{ и } r_{vj} > 0\}$ — множество индексов товаров, оценённых обоими пользователями.

Пример вычисления:

Для User 0 ([5,0,4,0,0,3]) и User 3 ([5,0,3,0,0,4]):

- Общие товары: индексы {0,2,5},
- Векторы оценок по общим товарам: [5,4,3] и [5,3,4],
- Числитель: $5 \cdot 5 + 4 \cdot 3 + 3 \cdot 4 = 25 + 12 + 12 = 49$,

- Знаменатель: $\sqrt{5^2 + 4^2 + 3^2} \cdot \sqrt{5^2 + 3^2 + 4^2} = \sqrt{50} \cdot \sqrt{50} = 50$,
- $\text{sim}(0,3) = 49/50 = 0.98$.

Входные данные

Файл `ratings.txt` в кодировке UTF-8:

- Каждая строка соответствует одному пользователю.
- Числа разделены пробелами.
- Первая строка — целевой пользователь (User 0).
- Пример содержимого:

```
5 0 4 0 0 3
4 0 5 0 0 4
0 3 0 2 5 0
5 0 3 0 0 4
```

Выходные данные

Архив `task4.zip` должен содержать два файла:

1. `similarities.txt`

- Для каждого пользователя $v \neq 0$, имеющего хотя бы один общий оценённый товар с User 0, строка в формате:
- `{user_id} {similarity}`
- `similarity` выводится с точностью 8 знаков после запятой.
- Порядок строк может быть произвольным.
- Пример:

```
1 0.97402464
3 0.98000000
```

2. `result.txt`

- Одна строка, содержащая целое число — индекс рекомендуемого товара.
- Пример:

```
1
```

Критерии оценивания

№	Критерий	Баллы
4.1	Корректное чтение матрицы и определение общих товаров	20
4.2	Правильный расчёт косинусной меры для всех пользователей	40
4.3	Верный выбор рекомендации (максимум оценки, при равенстве – минимум ID)	40
Итого		100

Технические требования

- Используйте только стандартные библиотеки Python (math, json и др.).
- Числовой допуск для схожести: $\pm 10^{-6}$.
- Гарантируется наличие хотя бы одной возможной рекомендации.
- Все файлы должны быть в кодировке UTF-8.
- При одинаковой максимальной схожести между несколькими пользователями выбирается любой из них.
- При одинаковых максимальных оценках у товаров выбирается товар с наименьшим индексом.

Задача 5: Анализ n-грамм

Нейросети не читают слова так, как это делают люди. Они анализируют последовательности символов и учатся на основе того, какие сочетания встречаются чаще всего. Такие пары и тройки символов как «пр», «ст», «ть» несут в себе ритм и структуру языка. Анализ -грамм — один из самых старых, но мощных методов для кодирования языка в машинном обучении. На его основе работают спам-фильтры, переводчики и системы анализа текстов. Ваша задача — расшифровать «ДНК языка», найдя наиболее частые последовательности символов.

Условие задачи

Дано: Архив `corpus.zip`, содержащий несколько текстовых файлов с расширением `.txt` (регистр расширения не важен).

Требуется:

1. Обработать все текстовые файлы в архиве в **алфавитном порядке имен файлов** (регистронезависимо), объединив их содержимое в указанном порядке.
2. Очистить объединенный текст согласно правилам предобработки.
3. Подсчитать частоты униграмм (1-грамм) и биграмм (2-грамм) в очищенном тексте.
4. Найти топ-10 наиболее частых униграмм и биграмм с учётом правил сортировки.
5. Подготовить два файла: `cleaned.txt` (очищенный текст) и `result.txt` (частоты -грамм).

Результат: Архив `task5.zip`, содержащий файлы `cleaned.txt` и `result.txt`.

Теоретическая часть

Что такое -грамма?

n -грамма — это последовательность из n подряд идущих символов в тексте. Например:

- **Униграмма** ($n = 1$): отдельные символы, например, “П”, “р”, “и”.
- **Биграмма** ($n = 2$): пары символов, например, “Пр”, “ри”, “ив”.

Из строки длины L можно получить:

- L униграмм,

- $L - 1$ биграмм.

Очистка текста

Оставить только буквы:

- Латинские буквы: a-z, A-Z
- Кириллические буквы: а-я, А-Я, ё, Ё
- Удалить всё остальное: цифры, знаки препинания, пробелы, специальные символы.

Пример очистки:

Исходный текст: "Привет, мир123!!! Это AI."
 → Только буквы: "ПриветмирЭтоAI"

Порядок обработки файлов

Для обеспечения воспроизводимости результата файлы обрабатываются **в алфавитном порядке имен файлов** (регистронезависимо). Например:

- file1.txt, file2.txt, notes.TXT, README.txt
- Порядок обработки: file1.txt → file2.txt → notes.TXT → README.txt

Подсчёт частот и сортировка

- **Частота** — количество вхождений -граммы в текст.
- **Правила сортировки топ-10:**
 1. По убыванию частоты.
 2. При равных частотах — по возрастанию ASCII-кода символов.

Порядок ASCII (важные символы):

'A' < 'B' < ... < 'Z' < 'a' < 'b' < ... < 'z' < 'А' < 'Б' < ... < 'Я' < 'а' < 'б' < ... < 'я'

Пример сортировки: Для униграмм с частотами:

'a': 10, 'б': 10, 'А': 10, 'в': 5, 'В': 5

Топ-3 после сортировки: А, а, б.

Входные данные

- Архив corpus.zip содержит файлы с расширением .txt (например, file1.txt, notes.TXT).
- В обработке участвуют **только** файлы с расширением .txt (регистронезависимо).
- Кодировка файлов: UTF-8.
- Порядок обработки: алфавитный порядок имен файлов (регистронезависимо).

Выходные данные

Архив task5.zip, содержащий два файла:

1. cleaned.txt:
 - Очищенный текст **без пробелов**.
 - Весь текст записывается в одну строку.
 - Пример содержимого: ПриветмирЭтотестHelloWorld.
2. result.txt:
 - Ровно 20 строк в формате: {n_грамма}<TAB>{частота}.
 - Первые 10 строк: топ-10 униграмм.
 - Следующие 10 строк: топ-10 биграмм.
 - Разделитель – символ табуляции (\t).
 - Пример содержимого:

```
т 4
l 3
o 3
...
He 1
ll 1
lo 1
...
```

Критерии оценивания

№	Критерий	Баллы
5.1	Корректное чтение файлов в алфавитном порядке, объединение	30

№	Критерий	Баллы
	содержимого и правильная очистка текста	
5.2	Топ-10 униграмм с правильной сортировкой	30
5.3	Топ-10 биграмм с правильной сортировкой	40
Итого		100

Технические требования

- Разрешённые библиотеки: `os`, `re`, `collections.Counter`, `glob`, `zipfile`.
- Кодировка всех файлов: UTF-8.

Задача 6: Обработка изображений фильтрами

Прежде чем нейросети смогут распознавать объекты на изображениях, им необходимо научиться извлекать из них полезные признаки. Фильтрация изображений — один из фундаментальных методов компьютерного зрения, позволяющий выделять границы, подавлять шумы и усиливать важные детали. Эта задача познакомит вас с базовыми операциями обработки изображений, которые лежат в основе современных систем компьютерного зрения.

Условие задачи

Дан файл `image.png`, содержащий полутоновое изображение. Необходимо последовательно применить к нему два фильтра:

1. **Гауссово размытие** с ядром 5×5 и $\sigma = 1.0$.
2. **Фильтр повышения четкости** методом “нерезкого маскирования” (unsharp masking).
3. **Инверсию изображения** для получения окончательного результата

Сохранить результаты всех этапов обработки в отдельные файлы и заархивировать их.

Требования к реализации

Этапы обработки и выходные файлы

Для проверки корректности выполнения каждого этапа необходимо сохранить промежуточные результаты:

1. `kernel.txt` — файл с вычисленным гауссовым ядром 5×5
 - Формат: 5 строк, в каждой строке 5 чисел с плавающей точкой (разделитель — пробел)
 - Точность: 6 знаков после запятой
 - Требование: сумма всех элементов ядра должна быть равна 1.0 с точностью $\pm 1e-6$
2. `blurred.png` — изображение после применения гауссова размытия
 - Формат: PNG, 8 бит на пиксель
 - Значения пикселей должны быть округлены до целых и ограничены диапазоном $[0, 255]$
3. `sharpened.png` — изображение после применения фильтра повышения четкости (до инверсии)
 - Формат: PNG, 8 бит на пиксель
 - Значения пикселей должны быть округлены до целых и ограничены диапазоном $[0, 255]$
4. `result.png` — финальное изображение после применения инверсии

- Формат: PNG, 8 бит на пиксель
- Формула инверсии: $I_{\text{final}}(x,y) = 255 - I_{\text{sharpened}}(x,y)$

Все файлы должны быть заархивированы в task6.zip.

Теоретическая часть

Гауссово размытие

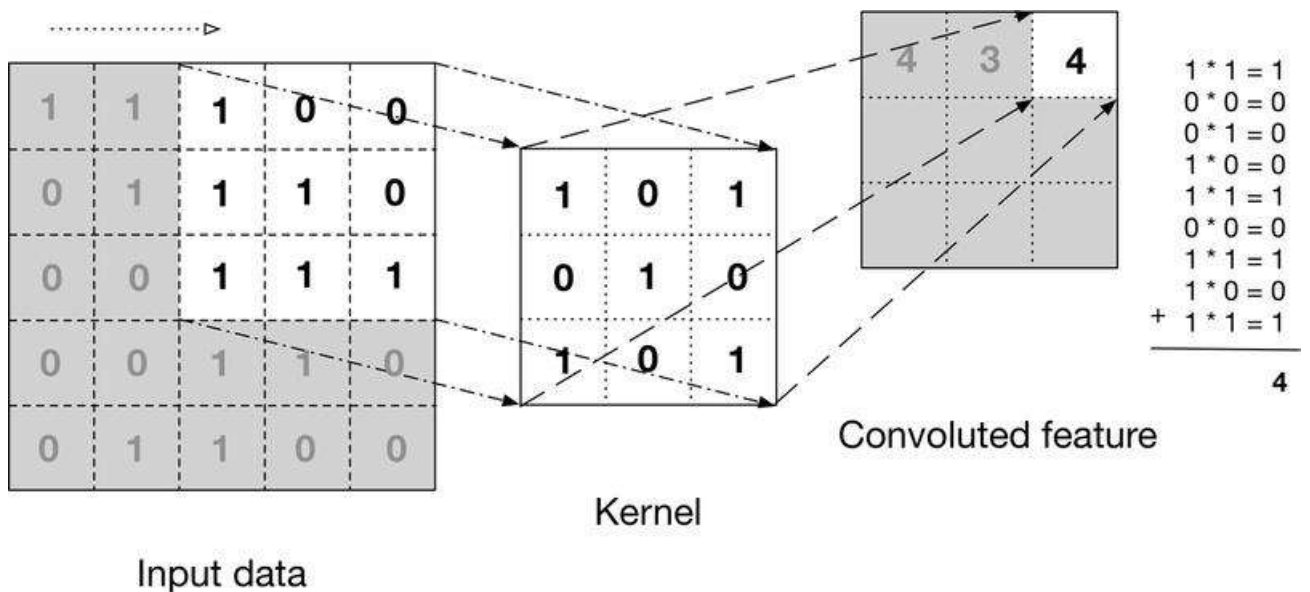
Гауссово ядро размера (k k) вычисляется по формуле:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

где (x, y) — координаты относительно центра ядра ((-2 x), (-2 y) для ядра 5×5).

Для применения фильтра выполняется операция свертки:

$$I_{\text{blurred}}(x, y) = \sum_{i=-2}^2 \sum_{j=-2}^2 I(x + i, y + j) \cdot G(i, j)$$



Пример вычисления ядра ($\sigma = 1.0$): Для центрального пикселя ($x=0, y=0$):

$$G(0,0) = \frac{1}{2\pi \cdot 1^2} e^{-\frac{0^2+0^2}{2 \cdot 1^2}} = \frac{1}{2\pi} \approx 0.159155$$

Фильтр повышения четкости (unsharp masking)

Метод основан на выделении высокочастотных компонент изображения. Алгоритм:

1. Исходное изображение: (I)
2. Размытое изображение: (B) (получено на предыдущем шаге)
3. Маска деталей: (M = I - B)
4. Усиленное изображение: ($I_{\text{sharp}} = I + M$)

где ($= 1.5$) — коэффициент усиления деталей.

Пример: Для пикселя со значением 100:

- Размытое значение: 95
- Маска деталей: $100 - 95 = 5$
- Усиленное значение: $100 + 1.5 \times 5 = 107.5 \rightarrow 108$ (после округления)

Инверсия изображения

Инверсия преобразует светлые области в темные и наоборот, что полезно для визуализации деталей:

$$I_{\text{inverted}}(x, y) = 255 - I_{\text{sharp}}(x, y)$$

Обработка границ

При применении фильтров для пикселей вблизи границ изображения используйте стратегию отражения (reflect):

$$I(x, y) = \begin{cases} I(0, y), & \text{если } x < 0 \\ I(\text{width} - 1, y), & \text{если } x \geq \text{width} \\ I(x, 0), & \text{если } y < 0 \\ I(x, \text{height} - 1), & \text{если } y \geq \text{height} \end{cases}$$

Пример отражения для изображения 3×3: Для координаты $(-1, 1) \rightarrow (0, 1)$ Для координаты $(3, 2) \rightarrow (2, 2)$

Входные данные

- Файл `image.png` в формате PNG с полутоновым изображением (8 бит на пиксель, значения 0-255)
- Размер изображения не менее 5×5 пикселей

Выходные данные

Четыре файла, заархивированных в `task6.zip`:

1. `kernel.txt` — матрица гауссова ядра 5×5 (5 строк по 5 чисел с точностью 6 знаков после запятой)
2. `blurred.png` — изображение после гауссова размытия

3. `sharpened.png` — изображение после повышения четкости
4. `result.png` — финальное изображение после инверсии

Критерии оценивания

№	Критерий	Баллы
6.1	Корректное вычисление гауссова ядра (точность, нормализация)	25
6.2	Правильная реализация свертки и обработки границ	35
6.3	Корректное применение фильтра повышения четкости и инверсии	40
	Итого	100

Технические требования

- **Разрешенные библиотеки:** `numpy`, `PIL (Pillow)`, `math`, `zipfile`
- **Точность вычислений:**
 - Гауссово ядро: не менее 6 знаков после запятой
 - Промежуточные вычисления: 32-битные числа с плавающей точкой
- **Ограничения значений:** все значения пикселей должны быть в диапазоне `[0, 255]`
- **Форматы файлов:**
 - `kernel.txt` — текстовый файл в кодировке UTF-8
 - Все изображения — PNG формат, 8 бит на пиксель
- **Структура архива:** `task2.zip` должен содержать ровно 4 файла в корневой директории

Примечания

- Гауссово ядро должно быть симметричным относительно центрального элемента
- Сумма всех элементов гауссова ядра должна быть строго равна 1.0
- При обработке границ обязательна стратегия отражения, а не заполнение нулями
- Все промежуточные результаты должны быть сохранены для проверки корректности каждого этапа
- Финальная инверсия применяется только к результату повышения четкости для создания контрастного представления деталей

Задача 7: Матричное представление свёртки с различными параметрами

Современные свёрточные нейронные сети (CNN) лежат в основе компьютерного зрения и обработки изображений. Понимание того, как операции свёртки могут быть представлены в виде матричных преобразований, является ключевым для оптимизации вычислений и глубокого понимания архитектуры нейросетей. Эта задача погружает вас в математические основы CNN, которые используются в таких системах, как ResNet, VGG и других архитектурах глубокого обучения.

Условие задачи

Дано входное изображение X размером 8×8 и три ядра свёртки разных размеров:

1. Ядро K_1 размером 2×2 со stride = 1
2. Ядро K_2 размером 3×3 со stride = 2
3. Ядро K_3 размером 4×4 со stride = 2

Для каждого ядра требуется построить соответствующую матрицу свёртки $K_{\text{mat}}^{(i)}$, которая преобразует векторизованное представление изображения $\text{vec}(X)$ в векторизованный результат свёртки $\text{vec}(Y_i)$:

$$\text{vec}(Y_i) = K_{\text{mat}}^{(i)} \cdot \text{vec}(X),$$

где Y_i — результат свёртки для i -го ядра (без padding).

Размеры выходных матриц $K_{\text{mat}}^{(i)}$ необходимо определить самостоятельно на основе параметров ядер и изображения.

Сохранить результаты в три файла и заархивировать их в task7.zip.

$$\underbrace{\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix}}_{\text{image}} * \underbrace{\begin{pmatrix} k_1 & k_2 \\ k_3 & k_4 \end{pmatrix}}_{\text{kernel}} \rightarrow \underbrace{\begin{pmatrix} k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 \\ 0 & 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 \end{pmatrix}}_{\text{transformed kernel}} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix}}_{\text{transformed image}} = \underbrace{\begin{pmatrix} k_1x_1 + k_2x_2 + k_3x_4 + k_4x_5 \\ k_1x_2 + k_2x_3 + k_3x_5 + k_4x_6 \\ k_1x_4 + k_2x_5 + k_3x_7 + k_4x_8 \\ k_1x_5 + k_2x_6 + k_3x_8 + k_4x_9 \end{pmatrix}}_{\text{result}}$$

Теоретическая часть

Что такое свёртка?

Свёртка — фундаментальная операция в CNN, которая применяет ядро (фильтр) к локальным областям изображения для выявления признаков. Ключевые параметры:

- **Ядро (kernel)** — матрица весов, определяющая, какие признаки будут обнаружены
- **Stride** — шаг перемещения ядра по изображению
- **Padding** — добавление границ (в задаче padding = 0)

Формула размера выходного изображения

Размер выходного изображения при свёртке без padding определяется по формуле:

$$\text{output_size} = \left\lfloor \frac{\text{input_size} - \text{kernel_size}}{\text{stride}} \right\rfloor + 1$$

где $\lfloor \cdot \rfloor$ — округление вниз.

Примеры:

- Для 8×8 изображения и ядра 2×2 со stride = 1: $(8 - 2)/1 + 1 = 7$
- Для ядра 3×3 со stride = 2: $(8 - 3)/2 + 1 = 3$
- Для ядра 4×4 со stride = 2: $(8 - 4)/2 + 1 = 3$

Векторизация изображения

Векторизация в порядке **row-major** (по строкам) преобразует матрицу 8×8 в вектор длины 64:

$$X = \begin{bmatrix} x_{00} & x_{01} & \dots & x_{07} \\ x_{10} & x_{11} & \dots & x_{17} \\ \vdots & \vdots & \ddots & \vdots \\ x_{70} & x_{71} & \dots & x_{77} \end{bmatrix} \xrightarrow{\text{vec}} \text{vec}(X) = [x_{00}, x_{01}, \dots, x_{07}, x_{10}, \dots, x_{77}]^T$$

Индекс элемента $X[i, j]$ в векторе: $k = i \cdot 8 + j$.

Матрица свёртки

Матрица свёртки K_{mat} имеет размер $(\text{output_size}^2) \times 64$:

- Каждая строка соответствует одному пикселю выходного изображения
- Каждый столбец соответствует одному пикселю входного изображения
- Элемент $K_{\text{mat}}[p, q]$ содержит вес ядра, применяемый к пикселю q при вычислении пикселя p

Правила построения:

1. Определите размер выходного изображения для каждого ядра
2. Для каждой позиции выходного изображения:
 - Определите соответствующее окно во входном изображении с учетом stride
 - Разместите веса ядра в соответствующих позициях строки матрицы
3. Все остальные элементы матрицы равны нулю

Входные данные

Три файла с ядрами свёртки:

- kernel1.txt — ядро 2×2
- kernel2.txt — ядро 3×3
- kernel3.txt — ядро 4×4

Каждый файл содержит значения ядра в формате:

```
k00 k01 ... k0m
k10 k11 ... k1m
...
kn0 kn1 ... knm
```

Выходные данные

Архив task7.zip, содержащий три файла:

- result1.txt — матрица свёртки для первого ядра (2×2 , stride = 1)
- result2.txt — матрица свёртки для второго ядра (3×3 , stride = 2)
- result3.txt — матрица свёртки для третьего ядра (4×4 , stride = 2)

Формат каждого файла:

```
{row0_col0} {row0_col1} ... {row0_col63}
{row1_col0} {row1_col1} ... {row1_col63}
...
{rowM_col0} {rowM_col1} ... {rowM_col63}
```

где $M = \text{output_size}^2$.

Каждое число выводится с точностью до 3 знаков после запятой.

Пример для ядра 2×2 со $\text{stride} = 1$

Для ядра:

```
1.0 0.5  
0.0 0.0
```

и изображения 8×8 , выход будет размером $7 \times 7 = 49$.

Первые две строки `result1.txt`:

```
1.000 0.500 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0  
.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.  
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.0  
00 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00  
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000  
0.000  
0.000 1.000 0.500 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0  
.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.  
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.0  
00 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00  
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000  
...
```

Объяснение: Первая строка соответствует вычислению верхнего левого пикселя выхода. Ядро 2×2 применяется к пикселям (0,0), (0,1), (1,0), (1,1) входного изображения. В векторизованном представлении эти пиксели имеют индексы 0, 1, 8, 9. Веса ядра 1.0 и 0.5 помещаются в позиции 0 и 1 соответственно.

Критерии оценивания

№	Критерий	Баллы
7.1	Правильный расчёт размеров выходных изображений для всех ядер	25
7.2	Корректное построение матрицы для одного ядра	25
7.3	Построение всех трёх матриц свёртки	35
7.4	Правильная обработка <code>stride</code> при построении матрицы	15
	Итого	100

Технические требования

- Разрешено использовать библиотеку `numpy`.
- Каждое число в выходных файлах должно быть выведено с точностью до 3 знаков после запятой.

- Порядок векторизации: row-major (по строкам).
- Все файлы должны быть в кодировке UTF-8.
- Числовой допуск при проверке: $\pm 10^{-6}$.

Задача 8: Оптимальная политика в решётке

Обучение с подкреплением формирует основу для создания автономных систем, способных принимать решения в сложных средах. От шахматных программ до систем управления роботами — все они используют принципы оценки состояний и поиска оптимальных стратегий. Алгоритм Value Iteration представляет собой фундаментальный метод динамического программирования, позволяющий агенту заранее оценивать ценность каждого состояния среды. Эта задача погружает вас в суть алгоритмов планирования, которые лежат в основе современных систем искусственного интеллекта.

Условие задачи

Дана прямоугольная сетка размером $H \times W$, где:

- S — стартовая позиция агента (ровно 1 клетка)
- T — сокровище (цель, даёт награду +10)
- X — ловушки (дают штраф -5)
- . — пустые клетки (штраф -0.1 за шаг)

Агент может выполнять действия в 4 направлениях: вверх, вниз, влево, вправо. При попытке выхода за границы сетки агент остаётся в текущей позиции.

Терминальными состояниями являются T и X, из которых дальнейшие переходы невозможны.

Требуется:

1. Реализовать алгоритм Value Iteration с коэффициентом дисконтирования $\gamma = 0.9$ для 100 итераций.
2. Вычислить ценность стартового состояния $V(S)$ после 100 итераций.
3. Подготовить три файла с результатами:
 - map.txt — координаты ключевых клеток в формате тип row col
 - v1.txt — значения функции ценности после 1-й итерации
 - result.txt — значение $V(S)$ после 100 итераций
4. Заархивировать все файлы в task8.zip.

Теоретическая часть

Основные понятия

Среда — прямоугольная сетка, где каждая клетка представляет собой состояние. Координатная система: (row, col) , где $row = 0$ соответствует верхней строке, $col = 0$ — левому столбцу.

Состояние s — текущая позиция агента (row, col).

Действие a — выбор направления движения из множества $A = \{Up, Down, Left, Right\}$.

Награда $R(s, a, s')$:

- +10 при переходе в состояние с сокровищем T
- −5 при переходе в состояние с ловушкой X
- −0.1 для всех остальных переходов (включая попытки выхода за границы)

Терминальные состояния — состояния с метками T и X. Для терминальных состояний $V(T) = V(X) = 0$, так как из них нет возможных переходов.

Функция ценности и алгоритм

Функция ценности $V(s)$ — ожидаемая суммарная дисконтированная награда при оптимальной стратегии, начинающейся из состояния s .

Уравнение Беллмана для оптимальной ценности:

$$V^*(s) = \max_{a \in A} [R(s, a) + \gamma \cdot V^*(s')]$$

Алгоритм Value Iteration:

1. **Инициализация:** $V_0(s) = 0$ для всех состояний s

2. **Итерация:** Для $k = 1$ до 100:

$$V_k(s) = \max_{a \in A} [R(s, a) + \gamma \cdot V_{k-1}(s')]$$

где s' — следующее состояние при действии a из состояния s

3. **Результат:** $V_{100}(S)$ — ценность стартового состояния после 100 итераций

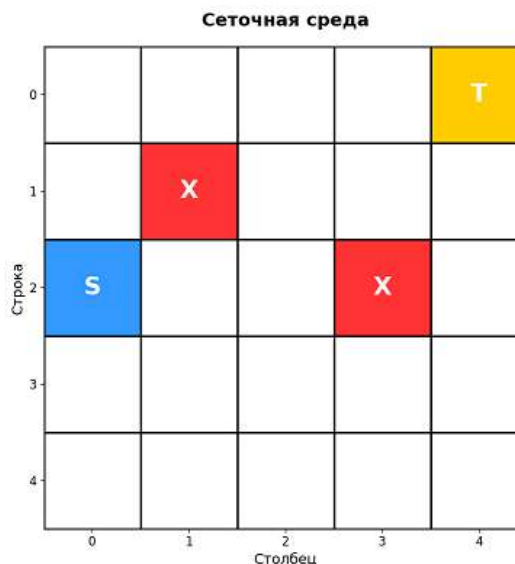
Пример

Для сетки:

....T
.X...
S..X.
.....

Координаты ключевых точек:

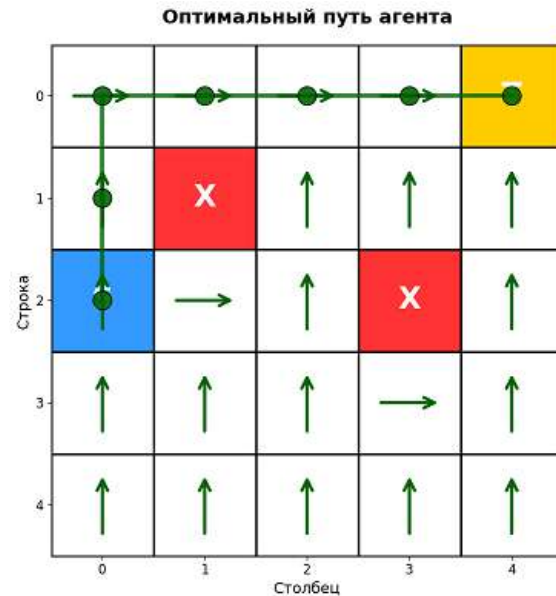
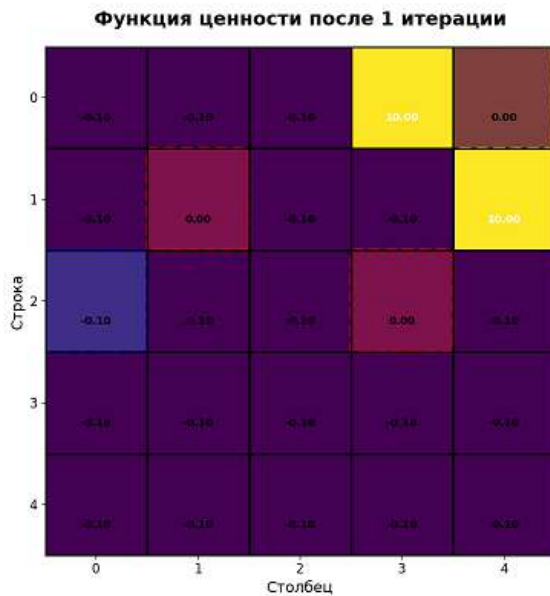
- S: (2,0)
- T: (0,4)



- $X: (1,1)$ и $(2,3)$

После 1-й итерации:

- $V(0,3) = R + \gamma \cdot V(T) = 10 + 0.9 \cdot 0 = 10.0$ (непосредственно перед сокровищем)
- $V(2,0) = \max_a [R + \gamma \cdot V(s')] = -0.1 + 0.9 \cdot 0 = -0.1$ (стартовая позиция)



Входные данные

Файл `grid.txt` в кодировке UTF-8, содержащий H строк по W символов (S, T, X, .).
Пример содержимого:

```
....T
.X...
S..X.
.....
```

Выходные данные

Архив `task8.zip`, содержащий три файла в кодировке UTF-8:

1. `map.txt`

Координаты ключевых точек в формате `row col` (каждая точка на новой строке):

```
S 2 0
T 0 4
X 1 1
X 2 3
```

2. v1.txt

Матрица значений функции ценности после 1-й итерации (H строк по W чисел с точностью 1 знак после запятой, разделённых пробелами):

```
-0.1 -0.1 -0.1 10.0 0.0  
-0.1 0.0 -0.1 -0.1 -0.1  
-0.1 -0.1 -0.1 0.0 -5.0  
-0.1 -0.1 -0.1 -0.1 -0.1
```

3. result.txt

Значение $V(S)$ после 100 итераций (6 знаков после запятой):

7.500000

Критерии оценивания

№	Критерий	Баллы
8.1	Корректный парсинг карты и определение координат	20
8.2	Корректная реализация одной итерации Value Iteration	40
8.3	Точное вычисление $V(S)$ после 100 итераций	40
Итого		100

Технические требования

- Параметры алгоритма: $\gamma = 0.9$, количество итераций = 100
- Точность вычислений: ± 0.01 для $V(S)$
- Все файлы должны быть в кодировке UTF-8
- Решение должно быть детерминированным и воспроизводимым

Задача 1: Детектор различий на изображениях

Разработайте алгоритм для автоматического обнаружения различий между двумя визуально похожими изображениями. Изображения подверглись геометрическим искажениям (сдвиг, поворот, масштабирование). Для облегчения выравнивания на изображения добавлены специальные черные маркеры в углах.

Ключевые особенности:

- Самостоятельная реализация алгоритма (без готовых библиотек компьютерного зрения)
- Обработка геометрических искажений
- Использование маркеров для точного выравнивания
- Оценка по метрике IoU (Intersection over Union) на двух наборах данных: публичном и приватном

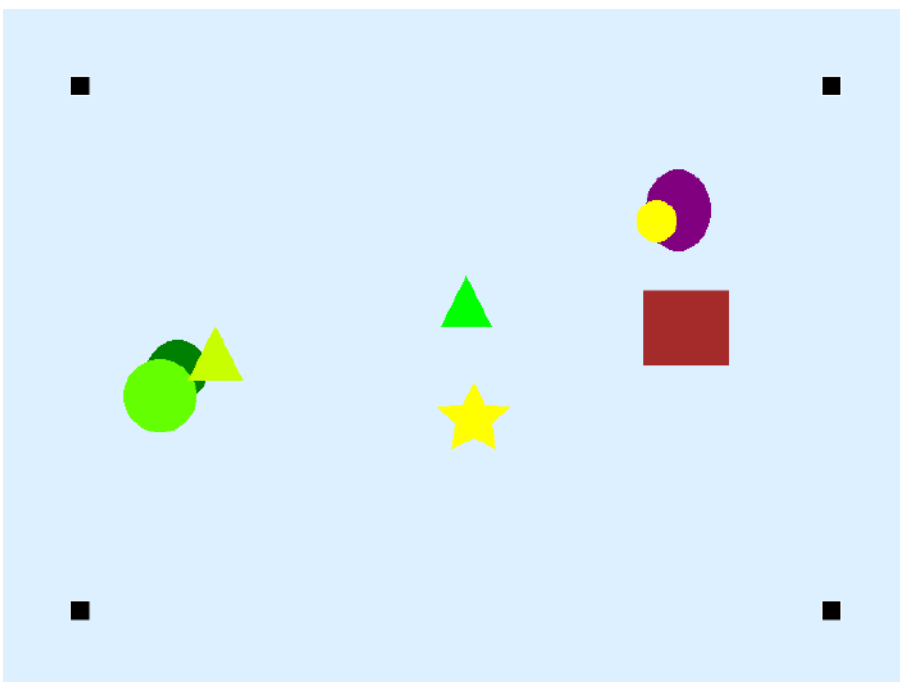
Входные данные

- Два цветных изображения в формате PNG: `image1.png` и `image2.png`
- Размер каждого изображения: **800×600 пикселей**
- Формат данных: RGB, 8 бит на канал (значения 0-255)

Маркеры для выравнивания

На каждом изображении присутствуют **4 черных квадрата-маркера**:

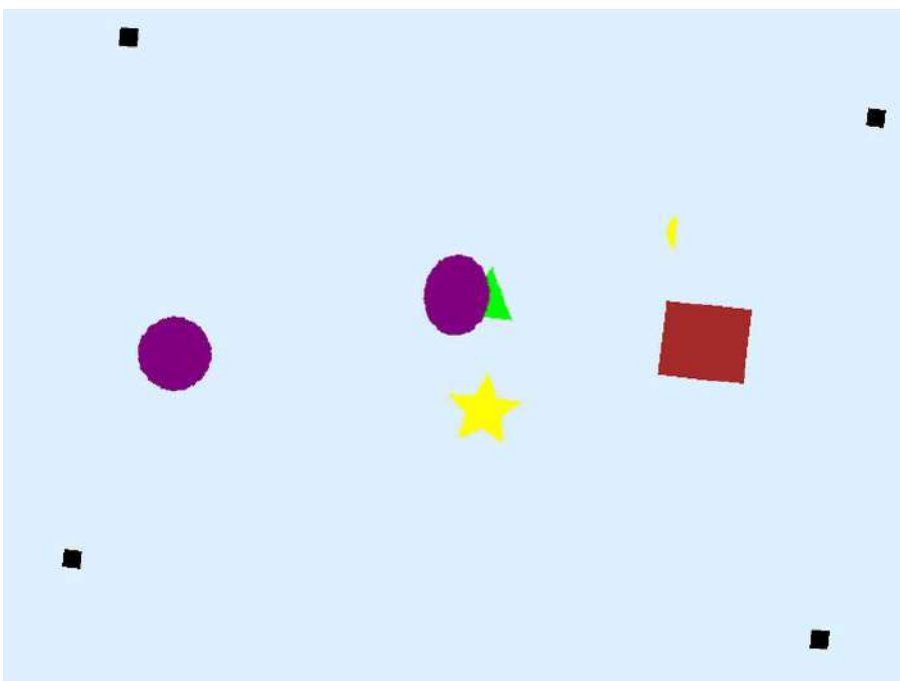
- Размер каждого маркера: **15×15 пикселей**
- Расположение: в углах изображения, на расстоянии **60 пикселей** от краев
- Координаты центров маркеров на первом изображении (без искажений):
 - Левый верхний: (67.5, 67.5)
 - Правый верхний: (732.5, 67.5)
 - Левый нижний: (67.5, 532.5)
 - Правый нижний: (732.5, 532.5)
- Цвет: абсолютно черный (0, 0, 0) с белой обводкой толщиной 1 пиксель
- **Важно:** Черный цвет используется ТОЛЬКО для маркеров, все остальные объекты имеют ненулевые значения хотя бы в одном цветовом канале



Типы искажений

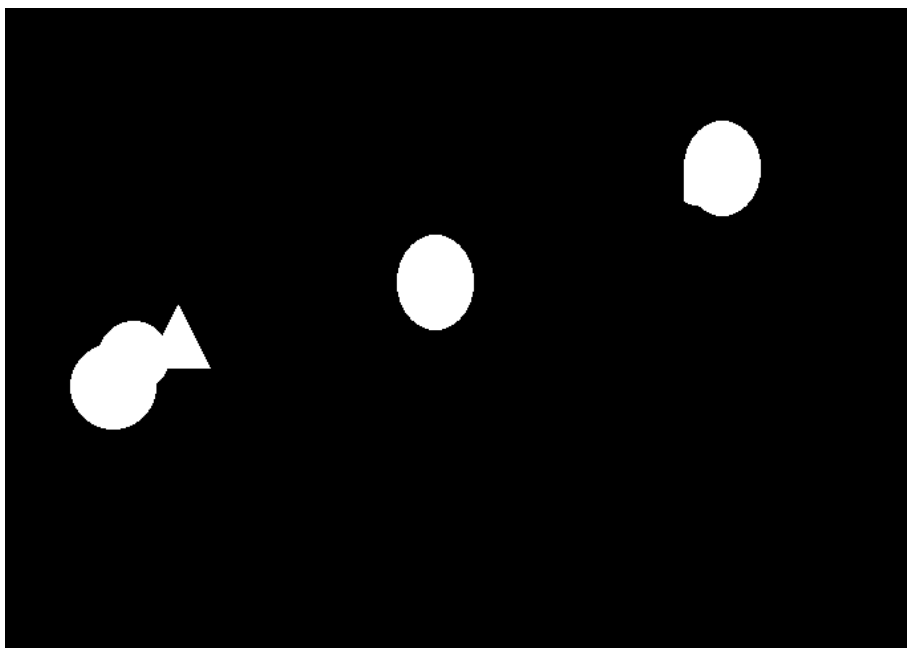
Второе изображение (`image2.png`) может быть подвергнуто следующим искажениям относительно первого:

- Случайный сдвиг: до 15 пикселей по горизонтали и вертикали
- Поворот: в диапазоне ± 8 градусов
- Масштабирование: от 92% до 108% от исходного размера



Постановка задачи

Реализуйте функцию `detect_diff(image1, image2)`, которая принимает два изображения и возвращает бинарную маску различий.



Требования к функции

```
def detect_diff(image1: np.ndarray, image2: np.ndarray) -> np.ndarray:
    """
    Находит различия между двумя изображениями.

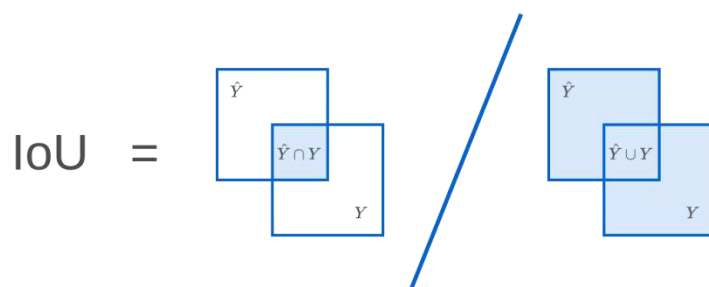
    Алгоритм должен выполнить следующие шаги:
    1. Обнаружить черные маркеры на обоих изображениях
    2. Вычислить геометрическое преобразование для выравнивания image2 к image1
    3. Применить преобразование к image2
    4. Обрезать оба изображения до области, ограниченной маркерами первого
    изображения
       с отступом 10 пикселей от центров маркеров
    5. Сравнить обрезанные изображения и создать бинарную маску различий
    6. Вернуть маску того же размера, что и обрезанные изображения (685×485
    пикселей)

    Параметры:
        image1: массив NumPy формы (600, 800, 3), тип uint8
        image2: массив NumPy формы (600, 800, 3), тип uint8

    Возвращает:
        mask: массив формы (485, 685), тип uint8, значения 0 или 255
    """
```

Метрика качества

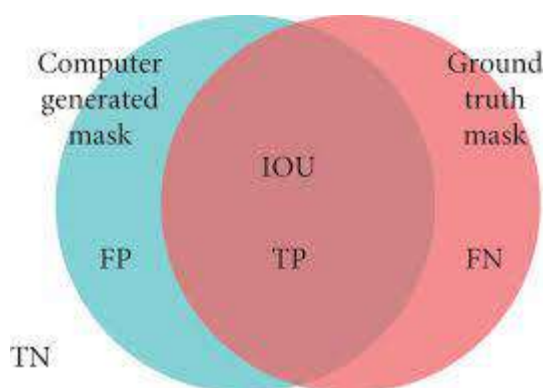
Качество решения оценивается с помощью метрики **IoU (Intersection over Union, Jaccard Index)** — стандартного показателя для задач семантической сегментации.



Формула IoU

TP (True Positives) — пиксели, правильно определенные как различия
 FP (False Positives) — пиксели, ошибочно определенные как различия
 FN (False Negatives) — пиксели различий, которые алгоритм пропустил

$$\text{IoU} = \text{TP} / (\text{TP} + \text{FP} + \text{FN})$$



Пример вычисления IoU

Предсказанная маска:	Эталонная маска:
[255, 0, 0, 0]	[255, 255, 0, 0]
[0, 255, 0, 0]	[0, 255, 0, 0]
[0, 0, 255, 0]	[0, 0, 0, 0]
[0, 0, 0, 0]	[0, 0, 0, 0]

TP = 2 (пиксели [0,0] и [1,1], где обе маски = 255)

FP = 1 (пиксель [2,2], где предсказание = 255, эталон = 0)

FN = 1 (пиксель [0,1], где предсказание = 0, эталон = 255)

$$\text{IoU} = 2 / (2 + 1 + 1) = 2 / 4 = 0.5$$

Итоговая оценка

Оценка вычисляется отдельно для **публичного** и **приватного** наборов тестов:

$$\text{Балл_публичный} = 100 \times (\text{средний IoU по публичным тестам})$$

Балл_приватный = $100 \times (\text{средний IoU по приватным тестам})$

Округление: до ближайшего целого (стандартное математическое округление).

Важно:

- При отправке решения на платформу отображается результат только на **публичных тестах**.
- После окончания конкурса финальный результат будет пересчитан на **приватных тестах**.

Требования к реализации

Разрешенные библиотеки

- `numpy` — для математических операций и работы с массивами
- `PIL (Pillow)` — **только** для загрузки и сохранения изображений
- `math` — для математических функций

Примечание: в конкурсе будет доступна краткая документация по разрешённым библиотекам.

Запрещенные библиотеки

- Любые библиотеки компьютерного зрения: `opencv (cv2)`, `scikit-image`, `scipy`
- Готовые реализации детекторов особенностей, дескрипторов, алгоритмов поиска соответствий
- Библиотеки для вычисления гомографии, аффинных преобразований
- Любые нейросетевые фреймворки

Ограничения

- **Время выполнения:** функция `detect_diff` должна завершаться за ≤ 30 секунд на одном тестовом примере. При превышении лимита тест считается проваленным ($\text{IoU} = 0$).
- **Использование памяти:** ≤ 2 ГБ. Решения, превышающие лимит, могут быть отклонены системой.
- **Детерминированность:** при одинаковых входных данных должен возвращаться одинаковый результат.
- **Самостоятельность:** все вычисления должны быть реализованы участником. Использование готовых решений из интернета запрещено.

Формат отправки решения

Структура решения

Решение должно состоять из одного файла Python с именем `solution.py`. Файл должен содержать реализацию функции `detect_diff` и может содержать вспомогательные функции.

Шаблон решения

```
import numpy as np
from PIL import Image
import math

def detect_diff(image1: np.ndarray, image2: np.ndarray) -> np.ndarray:
    """
    Находит различия между двумя изображениями.

    Параметры:
        image1: массив формы (600, 800, 3), тип uint8
        image2: массив формы (600, 800, 3), тип uint8

    Возвращает:
        mask: массив формы (485, 685), тип uint8, значения 0 или 255
    """
    # Ваша реализация здесь
    # ...

    return mask
```

Baseline решение

Для локального тестирования предоставляется простое baseline решение. Оно демонстрирует правильный формат выходных данных, но имеет низкое качество детектирования.

```
import numpy as np
from PIL import Image
import math

def detect_diff(image1: np.ndarray, image2: np.ndarray) -> np.ndarray:
    """
    Baseline решение — простой пример, демонстрирующий правильный формат
    выходных данных.
    Это решение НЕ является оптимальным и служит только для проверки формата.
    """
    # Простейший подход: преобразование в градации серого и пороговая обработка
    gray1 = np.mean(image1, axis=2).astype(np.float32)
    gray2 = np.mean(image2, axis=2).astype(np.float32)

    # Вычисление абсолютной разности
    diff = np.abs(gray1 - gray2)

    # Простая пороговая обработка
    threshold = 50
    mask = (diff > threshold).astype(np.uint8) * 255

    # Приведение маски к требуемому размеру
    # ВАЖНО: В реальном решении необходимо обрезать изображения до области
    маркеров
    # Этот код только демонстрирует правильный формат вывода
    if mask.shape != (485, 685):
```

```

        mask = mask[:485, :685] if mask.shape[0] >= 485 and mask.shape[1] >= 685
    else mask

    return mask

```

Локальное тестирование

Для локального тестирования вашего решения вам предоставляется архив с 5 парами тестовых изображений. Каждый тестовый пример содержит:

- `image1.png` — первое изображение
- `image2.png` — второе изображение с искажениями
- `true_mask.png` — эталонная маска различий (размер 485×685)

Пример кода для локальной проверки:

```

import numpy as np
from PIL import Image
from solution import detect_diff

def compute_iou(pred_mask, true_mask):
    """Вычисление IoU между предсказанной и эталонной маской"""
    intersection = np.logical_and(pred_mask == 255, true_mask == 255)
    union = np.logical_or(pred_mask == 255, true_mask == 255)

    if np.sum(union) == 0:
        return 0.0
    return np.sum(intersection) / np.sum(union)

# Пример тестирования на одной паре изображений
image1 = np.array(Image.open('test1/image1.png'))
image2 = np.array(Image.open('test1/image2.png'))
true_mask = np.array(Image.open('test1/true_mask.png'))

pred_mask = detect_diff(image1, image2)
iou_score = compute_iou(pred_mask, true_mask)
print(f"IoU: {iou_score:.4f}")

```

Проверка на сервере

При отправке решения на сервер:

1. Ваш файл `solution.py` будет импортирован.
2. Будет вызвана функция `detect_diff` для каждой тестовой пары.
3. Результат будет сравнен с эталонной маской.
4. Будет вычислен средний IoU отдельно для публичных и частных тестов.
5. **Важно:** Если возвращаемая маска имеет размер, отличный от 485×685, система автоматически изменит её размер до 485×685 (методом изменения размера изображения). Это может исказить результат, поэтому рекомендуется возвращать маску точно указанного размера.

Примечание: Убедитесь, что ваше решение работает в изолированном окружении и не требует дополнительных файлов или настроек.

Критерии оценивания

Критерий	Баллы	Описание
Корректность работы на публичных тестах	0–100	$100 \times \text{средний_IoU_публичный}$ (округляется до целого)
Корректность работы на частных тестах	0–100	$100 \times \text{средний_IoU_частный}$ (округляется до целого)

Итоговый балл определяется по результатам на **частных тестах** после окончания конкурса.

Примечание: Решения, не соответствующие техническим требованиям (использование запрещенных библиотек, превышение времени выполнения или потребления памяти) получают 0 баллов.

Особенности тестовых данных

Структура данных

- Тестовые данные разделены на **публичные** (5 примеров) и **частные** (неизвестное количество).
- При локальном тестировании вы можете проверить решение только на публичных примерах.
- Частные тесты становятся доступны только после окончания конкурса.

Характеристики изображений

- Все тестовые изображения имеют размер 800×600 пикселей.
- На каждом изображении гарантированно присутствуют 4 черных маркера.
- Маркеры всегда находятся в пределах изображения, даже после искажений.
- Эталонные маски созданы на основе неискаженных изображений, обрезанных до области маркеров.
- Область маркеров первого изображения (с отступом 10 пикселей от центров) используется для обрезки.
- Все эталонные маски имеют размер 485×685 пикселей.

Технические детали

Алгоритм создания эталонных масок (для понимания задачи):

1. Создаются два изображения без геометрических искажений.
2. На первом изображении размещаются маркеры и набор объектов (фигур).

3. На втором изображении вносятся изменения (добавление, удаление, перемещение или изменение цвета объектов).
4. Оба изображения обрезаются до области, ограниченной маркерами первого изображения с отступом 10 пикселей от центров маркеров.
5. На основе обрезанных изображений создается эталонная маска различий.
6. Ко второму изображению применяются геометрические искажения (поворот, масштабирование, сдвиг).

Ваша задача — выровнять искаженное второе изображение относительно первого, выполнить обрезку до той же области и обнаружить различия.

Работа проверочной системы

1. Система измеряет время выполнения для каждого теста. При превышении 30 секунд тест считается проваленным.
2. Маска, возвращаемая функцией `detect_diff`, приводится к размеру 485×685 , если её размер отличается.
3. IoU вычисляется отдельно для публичных и приватных тестов.
4. Итоговый балл вычисляется как $100 \times \text{средний_IoU_приватный}$ и округляется до целого числа.

Задание 2: Анализ новостной динамики в предновогодний период

Условие задачи

Вы работаете в аналитическом отделе новостного портала.

Задача — проанализировать, как меняется освещение новогодней тематики в СМИ в декабре 2025 года.

Дано:

Архив `news_raw.zip`, содержащий файлы:

- `news_raw.csv` — статьи за нечётные дни декабря 2025 года
- `gazeta_simple.py` — скрипт для сбора данных с новостного портала

Требуется:

1. Дособрать статьи за чётные дни декабря 2025 года.
2. Очистить тексты статей и извлечь сущности новогодней тематики.
3. Проанализировать динамику упоминаний новогодней тематики в течение декабря.
4. Подготовить файлы: `news_processed.csv`, `time_series.csv`, `mentions_dynamics.png`, `report.txt`.

Результат: Архив `solution.zip`, содержащий все требуемые файлы.

Подзадача 1: Досбор данных и первичная обработка

Цель: Получить полный набор данных за декабрь 2025 года.

Действия:

1. Собрать статьи за **чётные дни** декабря 2025 года (2, 4, 6, ..., 30 декабря).
2. Использовать модифицированный скрипт `gazeta_simple.py` с ключевыми словами новогодней тематики.
3. Объединить данные с `news_raw.csv`.
4. Сохранить результат в `news_full.csv`.

Подзадача 2: Очистка текста и извлечение сущностей

Цель: Подготовить тексты для анализа и выделить ключевые сущности.

Действия:

1. Очистить текст: удалить HTML, спецсимволы, оставить только буквы, цифры, основные знаки препинания.

2. Извлечь сущности трёх категорий:
 - **HOLIDAY** — праздники и мероприятия.
 - **ITEM** — новогодние атрибуты и подарки.
 - **PERSON** — сказочные персонажи.
3. Рекомендуется использовать лемматизацию для точного определения сущностей.

Подзадача 3: Анализ динамики и визуализация

Цель: Проанализировать интенсивность освещения новогодней тематики.

Действия:

1. Для каждого дня рассчитать:
 - Общее количество статей.
 - Количество статей с упоминаниями новогодней тематики.
 - Процент статей с упоминаниями.
2. Провести статистический анализ: максимум, минимум, тренд.
3. Построить график динамики упоминаний.
4. Подготовить текстовый отчёт с выводами.

Теоретическая часть

Основные понятия

- **Неструктурированные данные** — информация без формальной структуры (тексты, сообщения).
- **Предобработка текста** — приведение текста к стандартному виду для анализа.
- **Лемматизация** — приведение слова к начальной форме.
- **Извлечение сущностей (NER)** — автоматическое выделение именованных сущностей (персоны, организации и т.д.).
- **Временной ряд** — последовательность данных, измеренных в последовательные моменты времени.

Входные данные

- `news_raw.csv` — CSV-файл со статьями за нечётные дни декабря 2025 года.
- `gazeta_simple.py` — скрипт для сбора данных.

Важно: Все статьи должны быть собраны с реального сайта.
Генерация текстов ИИ запрещена.

Выходные данные

Архив `solution.zip` должен содержать:

1. `news_processed.csv`

Колонки:

`id, title, url, date, author, original_text, cleaned_text, entities_json`

2. `time_series.csv`

Колонки:

`date, total_articles, mentions_count, mentions_percentage`

3. `mentions_dynamics.png`

График динамики упоминаний новогодней тематики.



4. `report.txt` (или `report.md`)

Отчёт, содержащий:

- Описание метода сбора данных.
- Описание метода очистки и извлечения сущностей.
- Ключевые выводы из анализа.
- Объяснение особенностей графика.

Критерии оценивания

Подзадача	Критерий	Баллы	Тип проверки
1	Наличие данных за все 31 день декабря	10	Автоматическая
	Минимум 3 статьи на каждый чётный день	10	Автоматическая
	Корректность формата файла	10	Автоматическая
	Отсутствие пустых обязательных полей	5	Автоматическая
	Качество данных (реальные ссылки, уникальность)	5	Ручная
2	Полное удаление HTML-разметки	8	Автоматическая
	Корректная очистка от спецсимволов	7	Автоматическая

Подзадача	Критерий	Баллы	Тип проверки
	Валидность JSON с сущностями	5	Автоматическая
	Наличие сущностей всех трёх категорий	5	Автоматическая
	Качество извлечения сущностей	10	Ручная
3	Корректность расчётов временного ряда	5	Автоматическая
	Качество графика (оформление, читаемость)	5	Ручная
	Логичность аналитических выводов	8	Ручная
	Полнота и структурированность отчёта	7	Ручная
Итого		100	

Технические требования

Разрешённые библиотеки Python:

- **Сбор данных:** requests, BeautifulSoup4, selenium, lxml
- **Обработка текста:** re, json, pymorphy2, nltk, natasha, pandas, numpy
- **Анализ и визуализация:** matplotlib, seaborn, plotly

Запрещено:

- Использовать готовые API для сбора новостей (кроме предоставленного скрипта).
- Генерировать тексты статей с помощью ИИ.
- Использовать предобученные модели без указания в отчёте.

Задача 3: Вернись с сокровищами

Условие задачи

Агент находится в лабиринте размером 8×8 клеток и должен выполнить сложную миссию:

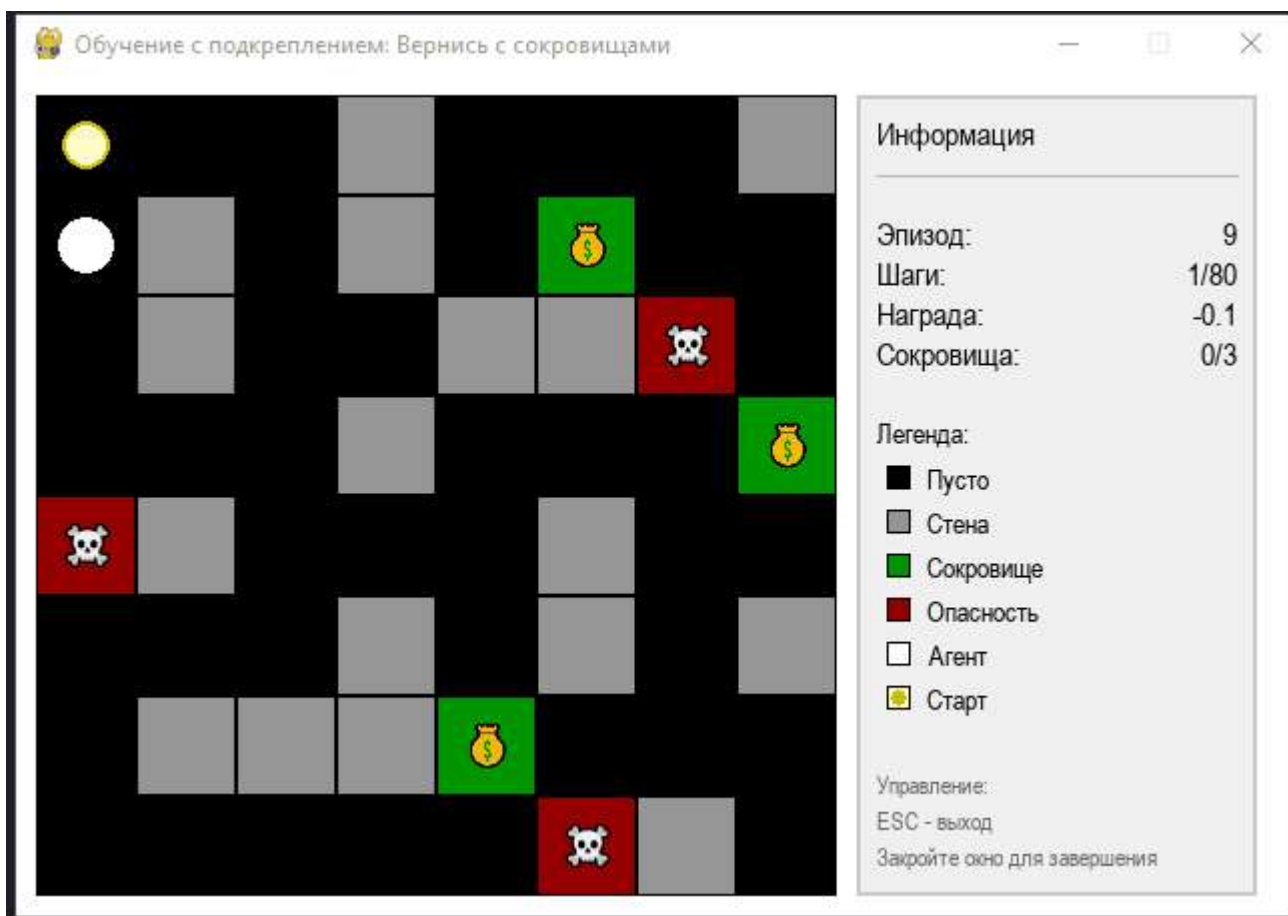
1. **Найти и собрать все 3 сокровища**, разбросанные по лабиринту
2. **Вернуться в стартовую позицию** после сбора всех сокровищ
3. **Избегать опасностей и динамических стен**, которые могут появляться и исчезать
4. **Уложиться в 80 шагов** на выполнение всей миссии

Карта содержит несколько типов клеток с разными свойствами:

Значение	Тип клетки	Описание	Награда/Штраф
-1	Старт	Начальная позиция агента	+50 за успешное возвращение
0	Пустая клетка	Проходимое пространство	-0.1 за каждый шаг
1	Стена	Непроходимое препятствие	-1 за попытку движения
2	Сокровище	Цель для сбора	+10 за каждое сокровище
3	Ловушка	Опасная зона	-5 за попадание (конец эпизода)
4	Агент	Текущая позиция агента	Отображается в наблюдении

Особенности среды:

- **Динамические стены** (15% вероятность появления/исчезновения на каждом шагу)
- **Фиксированные ловушки** (3 неподвижные опасные клетки)
- **Полная видимость** карты 8×8 в каждый момент времени
- **Сокровища исчезают** после сбора
- **Эпизод завершается** при:
 - Сборе всех сокровищ и возвращении домой
 - Попадании в ловушку
 - Превышении 80 шагов
 - Попытке пройти сквозь стену



Теоретическая часть

Марковский процесс принятия решений (MDP)

Среда представляет собой MDP, определённый кортежем (S, A, P, R, γ) , где:

- S — множество состояний (полная карта 8×8 + позиция агента + количество собранных сокровищ)
- $A = \{\text{вверх}(0), \text{вправо}(1), \text{вниз}(2), \text{влево}(3)\}$ — дискретный набор действий
- P — функция переходов с учётом динамических стен
- R — функция наград, заданная таблицей выше
- $\gamma = 0.9$ — коэффициент дисконтирования (по умолчанию)

Структура состояния

Наблюдение представляет собой матрицу 8×8 с кодировкой клеток. Состояние s_t в момент времени t включает:

- Позицию агента на карте
- Расположение всех объектов (стен, сокровищ, ловушек)
- Количество оставшихся сокровищ

- Количество выполненных шагов

Функция наград

$$r_t = \begin{cases} +10 & \text{при сборе сокровища} \\ +50 & \text{при возвращении домой после сбора всех сокровищ} \\ -5 & \text{при попадании в ловушку (эпизод завершается)} \\ -0.1 & \text{за каждый шаг по пустой клетке} \\ -1 & \text{при попытке движения в стену} \\ 0 & \text{в остальных случаях} \end{cases}$$

Цель агента

Агент должен максимизировать суммарную дисконтированную награду за эпизод:

$$\max \mathbb{E} \left[\sum_{t=0}^T \gamma^t r_t \right]$$

где $T \leq 80$ — максимальная длина эпизода.

Описание файлов проекта

1. `environment.py` *(редактировать запрещено)*

Роль: Моделирует игровую среду и её динамику.

Ключевые компоненты:

- Класс `Environment` — ядро симуляции
- Метод `reset()` — сбрасывает среду в начальное состояние
- Метод `step(action)` — выполняет действие и возвращает новое состояние, награду и флаг завершения
- Метод `get_observation()` — возвращает полное наблюдение карты 8×8
- Метод `update_dynamic_elements()` — управляет появлением/исчезновением динамических стен
- Метод `get_state_data()` — предоставляет данные для визуализации

Важно: Этот файл содержит всю логику игры — правила движения, систему наград, генерацию динамических стен и условия завершения эпизода.

2. `game.py` *(редактировать запрещено)*

Роль: Управляет взаимодействием между агентом и средой.

Ключевые компоненты:

- Класс `Game` — связующее звено между агентом и средой
- Метод `reset()` — инициализирует новый эпизод
- Метод `one_step_simulate()` — выполняет один шаг симуляции (выбор действия → выполнение → обучение)
- Метод `run_episode()` — запускает полный эпизод от начала до конца
- Метод `get_state_data()` — собирает данные для визуализации

Важно: Этот файл обеспечивает корректную интеграцию вашего агента с игровой средой, вызывая методы `choose_action` и `learn` в правильной последовательности.

3. `solution.py` *(единственный файл для редактирования)*

Роль: Содержит интеллект агента — стратегию принятия решений.

Требуемая реализация:

- Класс `Agent` — должен быть реализован участниками
- Метод `__init__()` — инициализация параметров обучения
- Метод `choose_action(state)` — **основной метод**, выбирает действие на основе текущего состояния
- Метод `learn(state, action, reward, next_state, done)` — опциональный метод для обучения агента

4. `visualizer.py` *(редактировать запрещено)*

Роль: Визуализация процесса обучения и тестирования.

Ключевые возможности:

- Графическое отображение карты 8×8 с разными цветами для типов клеток
- Анимация движения агента
- Панель информации с текущей статистикой (шаги, награда, сокровища)
- Легенда с обозначениями
- Управление скоростью симуляции

Использование: Для отладки и понимания поведения агента. Запускается через `python local_tester.py --visualize`.

5. `local_tester.py` *(редактировать запрещено)*

Роль: Автоматическое тестирование и оценка решения.

Ключевые функции:

- Запуск множества эпизодов для оценки стабильности агента
- Расчет финального балла по четырём критериям:
 1. **Сбор сокровищ** (0-50 баллов) — за количество собранных сокровищ

2. **Возвращение домой** (0-30 баллов) — за успешное возвращение после сбора всех сокровищ
 3. **Эффективность** (0-10 баллов) — за минимальное количество шагов на сокровище
 4. **Избегание ловушек** (0-10 баллов) — за минимизацию попаданий в ловушки
- Вывод подробной статистики и рекомендаций по улучшению
 - Поддержка быстрого теста (100 эпизодов) и полного теста (1000 эпизодов)

Важно: Именно этот тестировщик будет использоваться для финальной оценки решения.

6. requirements.txt

Роль: Список зависимостей Python.

Содержимое:

```
numpy>=1.19.0
pygame>=2.0.0
```

Входные данные

Среда предоставляется в виде набора файлов в архиве `task3.zip`. Структура архива:

```
task3.zip/
├── environment.py    # Класс Environment (редактировать запрещено)
├── game.py          # Класс Game (редактировать запрещено)
├── visualizer.py    # PyGame визуализация (редактировать запрещено)
├── local_tester.py  # Локальный тестировщик (редактировать запрещено)
├── solution.py      # Ваш класс Agent (единственный файл для редактирования)
└── requirements.txt # Зависимости Python
```

Исходная карта 8×8 имеет следующую структуру:

Старт (-1)	Пусто (0)	Пусто (0)	Стена (1)	Пусто (0)	Пусто (0)	Пусто (0)	Стена (1)
Пусто (0)	Стена (1)	Пусто (0)	Стена (1)	Пусто (0)	Сокр. (2)	Пусто (0)	Пусто (0)
Пусто (0)	Стена (1)	Пусто (0)	Пусто (0)	Пусто (0)	Стена (1)	Ловуш. (3)	Пусто (0)
Пусто (0)	Пусто (0)	Пусто (0)	Стена (1)	Пусто (0)	Пусто (0)	Пусто (0)	Сокр. (2)
Стена (1)	Стена (1)	Пусто (0)	Пусто (0)	Пусто (0)	Стена (1)	Пусто (0)	Пусто (0)
Пусто (0)	Пусто (0)	Пусто (0)	Стена (1)	Пусто (0)	Стена (1)	Пусто (0)	Стена (1)
Пусто (0)	Стена (1)	Стена (1)	Стена (1)	Сокр. (2)	Пусто (0)	Пусто (0)	Пусто (0)
Пусто (0)	Пусто (0)	Пусто (0)	Пусто (0)	Пусто (0)	Ловуш. (3)	Стена (1)	Пусто (0)

Координаты ключевых объектов:

- Стартовая позиция: (0, 0)
- Сокровища: (1, 5), (3, 7), (6, 4)
- Ловушки: (2, 6), (4, 0), (7, 5)

Выходные данные

Участники должны предоставить файл `solution.py`, содержащий класс `Agent` с реализованным методом `choose_action`. Файл должен быть совместим с предоставленным тестировщиком.

Пример минимальной реализации:

```
import numpy as np

class Agent:
    def __init__(self):
        self.actions = [0, 1, 2, 3] # 0=up, 1=right, 2=down, 3=left

    def choose_action(self, state):
        """
        state: словарь с информацией о состоянии
        state['observation'] – матрица 8×8 с полной картой
        Возвращает целое число от 0 до 3
        """
        # Ваша стратегия здесь
        return np.random.choice(self.actions)

    def learn(self, state, action, reward, next_state, done):
        """
        Опциональный метод для обучения агента
        """
        pass
```

Критерии оценивания

Система баллов (0-100):

Критерий	Макс. баллов	Метод расчета
Базовый балл (сокровища)	50	$(\text{среднее_сокровищ} / 3) \times 50$
Возвращение домой	30	$\%_возвращений \times 30$
Бонус эффективности	10	Градация по шагам на сокровище: ≤ 20 шагов $\rightarrow$ 10 баллов ≤ 25 шагов $\rightarrow$ 8 баллов ≤ 30 шагов $\rightarrow$ 6 баллов ≤ 40 шагов $\rightarrow$ 4 баллов ≤ 60 шагов $\rightarrow$ 2 баллов
Бонус за избегание ловушек	10	Градация по частоте попаданий: $< 10\%$ $\rightarrow$ 10 баллов $< 20\%$ $\rightarrow$ 8 баллов $< 30\%$ $\rightarrow$ 6 баллов $< 40\%$ $\rightarrow$ 4 баллов

Критерий	Макс. баллов	Метод расчета
		<50% → 2 баллов
Штраф за случайное блуждание	-20	Если среднее сокровищ < 0.5 и возвратов < 10%

Технические требования

1. **Единственный редактируемый файл** — `solution.py`
2. **Запрещено изменять** другие файлы проекта
3. **Максимальное время выполнения** одного эпизода — 1 секунда
4. **Допустимые библиотеки** — только стандартные библиотеки Python и `pymru`
5. **Детерминированность** — решение должно давать стабильные результаты при многократном запуске
6. **Объём решения** — не более 500 строк кода

Инструкции для участников

Установка зависимостей:

```
pip install -r requirements.txt
```

Тестирование решения:

```
# Быстрый тест (100 эпизодов)
python local_tester.py --quick

# Полный тест (1000 эпизодов)
python local_tester.py

# Тест с визуализацией (1 эпизод в реальном времени)
python local_tester.py --visualize

# Тест с другим количеством эпизодов
python local_tester.py --episodes 500
```

Рекомендации по разработке:

1. **Начните с простой стратегии** — сначала доберитесь до одного сокровища
2. **Используйте визуализацию** для понимания поведения агента
3. **Проанализируйте карту** — запомните расположение ловушек и оптимальные маршруты
4. **Реализуйте алгоритм Q-learning** как базовый подход
5. **Оптимизируйте возвращение домой** — запоминайте путь к стартовой позиции
6. **Тестируйте на 1000 эпизодах** для оценки стабильности

Задача 4: Охотник за сокровищами

Условие задачи

Агент-охотник находится в опасном лабиринте размером 15×15 клеток. Его миссия состоит из нескольких этапов сложности:

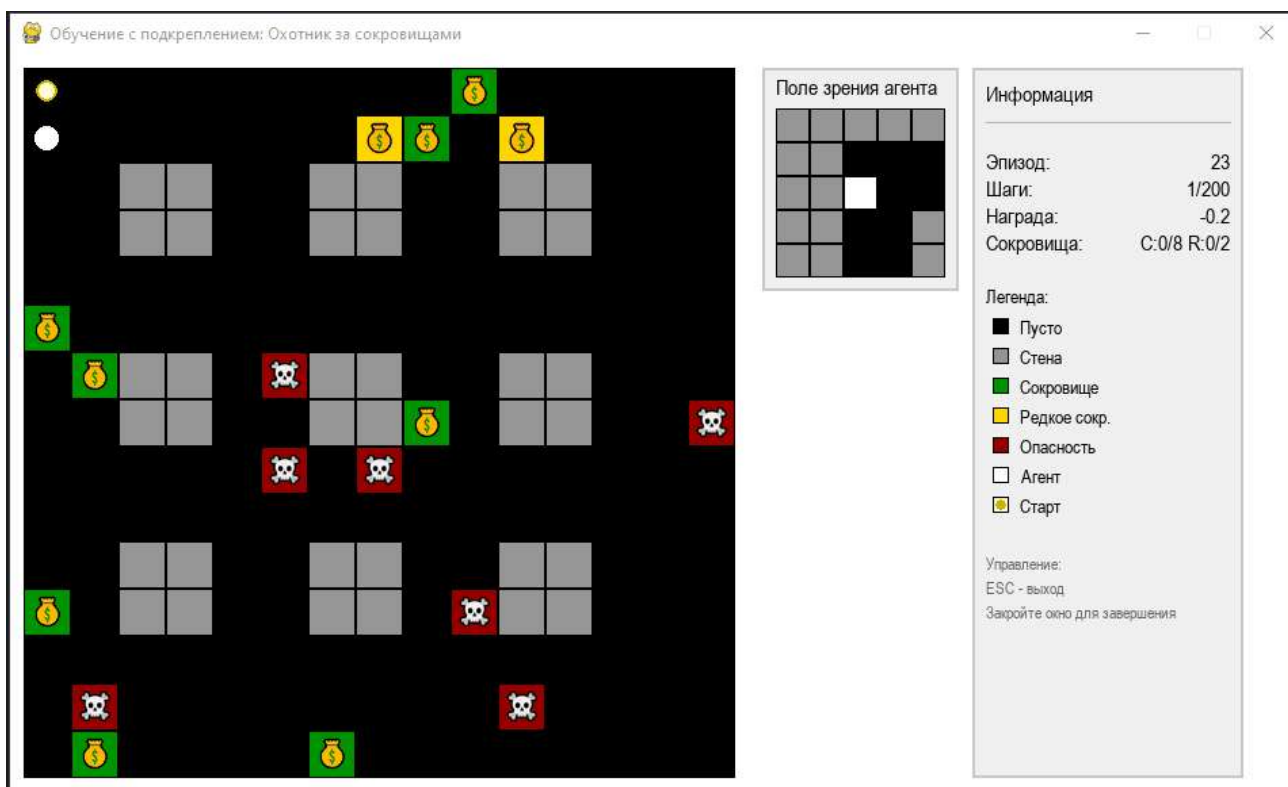
1. **Найти и собрать все 10 сокровищ:** 8 обычных (+10) и 2 редких (+25)
2. **Вернуться на стартовую позицию** после сбора всех сокровищ (+50)
3. **Избегать охотящихся ловушек**, которые активно преследуют агента
4. **Уложиться в 200 шагов** на выполнение всей миссии
5. **Работать с ограниченным полем зрения** (5×5 клеток вокруг агента)

Среда представляет собой сложный динамический мир с несколькими типами объектов:

Значение	Тип объекта	Описание	Награда/Штраф
-1	Старт	Начальная позиция агента	+50 за успешное возвращение
0	Пустая клетка	Проходимое пространство	-0.2 за каждый шаг
1	Стена	Непроходимое препятствие	-1 за попытку движения
2	Обычное сокровище	Цель для сбора	+10 за каждое
5	Редкое сокровище	Ценная цель	+25 за каждое
3	Ловушка	Опасная зона (движется!)	-5 за попадание (конец эпизода)
4	Агент	Текущая позиция агента	Отображается в наблюдении

Особенности среды:

- **Охотящиеся ловушки** (7 штук) двигаются к агенту с вероятностью 70% каждые 2 шага
- **Два типа сокровищ** с разной ценностью
- **Ограниченная видимость** (5×5 клеток вокруг агента)
- **Динамические препятствия** в виде стен определенной структуры
- **Сокровища исчезают** после сбора
- **Эпизод завершается** при:
 - Сборе всех сокровищ и возвращении на старт
 - Попадании в ловушку
 - Превышении 200 шагов
 - Попытке пройти сквозь стену



Теоретическая часть

Частично наблюдаемый марковский процесс (POMDP)

Среда представляет собой POMDP, определённый кортежем $(S, A, P, R, \Omega, O, \gamma)$, где:

- S — множество полных состояний (вся карта 15×15 + все объекты)
- $A = \{\text{вверх}(0), \text{вправо}(1), \text{вниз}(2), \text{влево}(3)\}$ — дискретный набор действий
- P — функция переходов с учётом движущихся ловушек
- R — функция наград, заданная таблицей выше
- Ω — множество наблюдений (5×5 область вокруг агента)
- O — функция наблюдения (ограниченная видимость)
- $\gamma = 0.99$ — коэффициент дисконтирования

Структура наблюдения

Агент получает только локальную информацию — матрицу 5×5 с центром в его текущей позиции. Наблюдение o_t включает:

- Позицию агента в центре матрицы (значение 4)
- Типы клеток в пределах видимости
- Объекты, попавшие в поле зрения

Функция наград

$$r_t = \begin{cases} +10 & \text{при сборе обычного сокровища} \\ +25 & \text{при сборе редкого сокровища} \\ +50 & \text{при возвращении на старт после сбора всех сокровищ} \\ -5 & \text{при попадании в ловушку (эпизод завершается)} \\ -0.2 & \text{за каждый шаг по пустой клетке} \\ -1 & \text{при попытке движения в стену} \\ 0 & \text{в остальных случаях} \end{cases}$$

Цель агента

Агент должен максимизировать суммарную дисконтированную награду за эпизод:

$$\max \mathbb{E} \left[\sum_{t=0}^T \gamma^t r_t \right]$$

где $T \leq 200$ — максимальная длина эпизода.

Описание файлов проекта

1. `environment.py` *(редактировать запрещено)*

Роль: Моделирует сложную игровую среду с динамическими угрозами и ограниченной видимостью.

Ключевые компоненты:

- Класс `Environment` — ядро симуляции с движущимися ловушками
- Метод `reset()` — сбрасывает среду, случайно размещая сокровища и ловушки
- Метод `step(action)` — выполняет действие и возвращает наблюдение 5×5 , награду и флаг завершения
- Метод `get_observation()` — возвращает **ограниченное наблюдение** (5×5 область вокруг агента)
- Метод `_move_traps()` — управляет интеллектуальным движением ловушек (70% к агенту)
- Метод `_place_treasures()` — случайное размещение двух типов сокровищ
- Метод `get_state_data()` — предоставляет данные для визуализации полной карты

Важно: Этот файл содержит всю сложную логику игры — движение охотящихся ловушек, случайную генерацию сокровищ, ограниченную видимость агента и все условия завершения эпизода. Ловушки двигаются целенаправленно, что делает среду особенно сложной.

2. game.py *(редактировать запрещено)*

Роль: Управляет взаимодействием между агентом и сложной динамической средой.

Ключевые компоненты:

- Класс `Game` — связующее звено между агентом и средой с частичной наблюдаемостью
- Метод `reset()` — инициализирует новый эпизод с новой расстановкой объектов
- Метод `one_step_simulate()` — выполняет один шаг симуляции (выбор действия → выполнение → обучение)
- Метод `run_episode()` — запускает полный эпизод от начала до конца (до 200 шагов)
- Метод `get_state_data()` — собирает данные для визуализации (полная карта + поле зрения)

Важно: Этот файл обеспечивает корректную интеграцию вашего агента со средой с **частичной наблюдаемостью**. Он вызывает метод `choose_action`, передавая только локальное наблюдение 5×5 , и при наличии метода `learn` передает опыт для обучения.

3. solution.py *(единственный файл для редактирования)*

Роль: Содержит интеллект агента, способного действовать в условиях неполной информации и динамических угроз.

Требуемая реализация:

- Класс `Agent` — должен быть реализован участниками
- Метод `__init__()` — инициализация параметров обучения и внутренних структур
- Метод `choose_action(state)` — **основной метод**, выбирает действие на основе **ограниченного наблюдения 5×5**
- Метод `learn(state, action, reward, next_state, done)` — опциональный метод для онлайн-обучения

4. visualizer.py *(редактировать запрещено)*

Роль: Визуализация процесса в сложной среде с двумя типами информации.

Ключевые возможности:

- **Двойное отображение:** полная карта 15×15 + поле зрения агента 5×5
- Графическое представление **двух типов сокровищ** (обычные и редкие)
- Анимация **движения охотящихся ловушек**
- Отображение **агента**
- Панель информации с текущей статистикой (счет, шаги, собранные сокровища)
- **Динамическая легенда** с обозначениями всех типов объектов
- Поддержка **масштабирования** для разных размеров экрана

Использование: Критически важен для отладки стратегий в условиях частичной наблюдаемости. Запускается через `python local_tester.py --visualize`.

5. local_tester.py (редактировать запрещено)

Роль: Автоматическое тестирование и оценка решения в сложных условиях.

Ключевые функции:

- Запуск **1000 эпизодов** для оценки стабильности и адаптивности агента
- Расчет финального балла по **трем критериям**:
 1. **Сбор сокровищ** (0-40 баллов) — за эффективность сбора обоих типов сокровищ
 2. **Эффективность** (0-40 баллов) — за оптимальность маршрутов (шаги на сокровище)
 3. **Успешность миссии** (0-20 баллов) — за полное выполнение (все сокровища + возврат)
- Вывод **подробной статистики** по каждому типу сокровищ отдельно
- **Прогрессивный отчет** каждые 100 эпизодов
- Поддержка **быстрого теста** (100 эпизодов) и **полного теста** (1000 эпизодов)
- **Визуализация процесса тестирования** в реальном времени

Важно: Именно этот тестировщик будет использоваться для финальной оценки решения. Он оценивает не только результат, но и **эффективность** и **стабильность** агента.

6. requirements.txt

Роль: Список минимальных зависимостей Python для работы проекта.

Содержимое:

```
numpy>=1.19.0
pygame>=2.0.0
```

Установка: Все зависимости устанавливаются одной командой `pip install -r requirements.txt`.

Входные данные

Среда предоставляется в виде набора файлов в архиве `task4.zip`. Структура архива:

```
task4.zip/
├── environment.py    # Класс Environment (редактировать запрещено)
├── game.py          # Класс Game (редактировать запрещено)
├── visualizer.py    # PyGame визуализация (редактировать запрещено)
├── local_tester.py  # Локальный тестировщик (редактировать запрещено)
├── solution.py      # Ваш класс Agent (единственный файл для редактирования)
└── requirements.txt # Зависимости Python
```

Базовая структура карты 15×15 представляет собой решетку из блоков 2×2:

- Черные квадраты (значение 1) — непроходимые стены
- Белые области — проходимое пространство
- Стартовая позиция находится в левом верхнем углу (0, 0)

Особенности генерации:

- 8 обычных сокровищ размещаются случайно в проходимых клетках
- 2 редких сокровища размещаются случайно в проходимых клетках
- 7 ловушек размещаются случайно в проходимых клетках
- Все объекты гарантированно находятся в разных клетках
- Расположение меняется при каждом сбросе среды

Выходные данные

Участники должны предоставить файл `solution.py`, содержащий класс `Agent` с реализованными методами `__init__` и `choose_action`. Файл должен быть совместим с предоставленным тестировщиком.

Пример минимальной реализации:

```
import numpy as np

class Agent:
    def __init__(self):
        self.actions = [0, 1, 2, 3] # 0=вверх, 1=вправо, 2=вниз, 3=влево

    def choose_action(self, state):
        """
        state: словарь с информацией о состоянии
        state['observation'] — матрица 5×5 с локальной видимостью
        Возвращает целое число от 0 до 3
        """
        # Ваша стратегия здесь
        return np.random.choice(self.actions)

    def learn(self, state, action, reward, next_state, done):
        """
        Опциональный метод для обучения агента
        """
        pass
```

Критерии оценивания

Система баллов (0-100):

Критерий	Макс. баллов	Метод расчета
Базовый балл	40	(средние_обычные/8)×24 + (средние_редкие/2)×16

Критерий (сокровища)	Макс. баллов	Метод расчета
Балл за эффективность	40	Градация по шагам на сокровище: ≤ 25 шагов $\rightarrow$ 40 баллов 25-50 шагов $\rightarrow$ линейное уменьшение > 50 шагов $\rightarrow$ 0 баллов
Балл за успешность миссии	20	$\min(20, \%_эпизодов_со_всеми_сокровищами \times 0.1 + \%_успешных_возвратов \times 0.1)$

Расшифровка критериев:

1. **Сбор сокровищ** — оценивает способность агента находить и собирать оба типа сокровищ
2. **Эффективность** — оценивает оптимальность маршрутов (меньше шагов на сокровище = лучше)
3. **Успешность миссии** — оценивает способность завершать миссию полностью (все сокровища + возврат)

Итоговый балл вычисляется как сумма трех компонентов с ограничением от 0 до 100.

Технические требования

1. **Единственный редактируемый файл** — `solution.py`
2. **Запрещено изменять** другие файлы проекта
3. **Максимальное время выполнения** одного эпизода — 2 секунды
4. **Допустимые библиотеки** — только стандартные библиотеки Python и `pymru`
5. **Детерминированность** — решение должно давать стабильные результаты при многократном запуске
6. **Объём решения** — не более 500 строк кода
7. **Ограничение памяти** — не более 512 МБ

Инструкции для участников

Установка зависимостей:

```
pip install -r requirements.txt
```

Тестирование решения:

```
# Быстрый тест (100 эпизодов)
python local_tester.py --quick
```

```
# Полный тест (1000 эпизодов)
python local_tester.py
```

```
# Тест с визуализацией (1 эпизод в реальном времени)
```

```
python local_tester.py --visualize  
  
# Тест с другим количеством эпизодов  
python local_tester.py --episodes 500
```

Рекомендации по разработке:

1. **Начните с исследования карты** — используйте визуализатор для понимания структуры лабиринта
2. **Реализуйте алгоритм запоминания карты** — так как видимость ограничена, агент должен строить внутреннюю карту
3. **Используйте стратегию избегания ловушек** — помните, что ловушки двигаются к вам!
4. **Приоритизируйте редкие сокровища** — они дают больше очков, но могут быть опаснее
5. **Планируйте возвращение** — запоминайте путь к старту во время исследования
6. **Балансируйте исследование и эксплуатацию** — в начале нужно исследовать, в конце — эффективно собирать

Задача 1: Очистка данных и корреляционный анализ

Условие задачи

Перед проведением любого анализа данных необходимо убедиться в их качестве. Реальные данные часто содержат пропуски, выбросы, нечисловые значения и дубликаты. В этой задаче вам предстоит выполнить первичную обработку набора данных, объединив информацию из нескольких файлов, очистив её от ошибок и выявив статистические взаимосвязи между признаками.

Вам необходимо написать программу, которая распакует архив с данными, произведёт очистку таблиц согласно строгим правилам, выполнит сортировку и вычислит коэффициенты корреляции между столбцами. Результат работы должен быть сохранён в виде нового архива с двумя файлами: очищенными данными и списком наиболее коррелирующих пар признаков.

Теоретическая справка

Коэффициент корреляции Пирсона

Для двух столбцов X и Y , содержащих по n значений, коэффициент корреляции вычисляется по формуле:

$$r_{XY} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

где $\bar{x}$ и $\bar{y}$ — средние арифметические соответствующих столбцов. Значение r лежит в интервале $[-1, 1]$.

Входные данные

Вам предоставлен ZIP-архив `task1_data.zip`.

- **Состав архива:** Несколько файлов с расширением `.csv` (регистр расширения не важен, например, `.csv`, `.CSV`).
- **Расположение:** Обрабатываются только файлы, находящиеся в корневой директории архива (файлы во вложенных папках игнорируются).
- **Формат файлов:**
 - Таблица с четырьмя столбцами вещественных чисел, разделённых запятыми.
 - Заголовки столбцов отсутствуют.
 - Кодировка: UTF-8.
- **Особенности данных:**
 - Числа могут быть окружены пробелами.
 - Разделитель десятичной части — точка.

- Возможны пустые поля, специальные значения (NaN, inf, -inf), нечисловые символы и строки с неверным количеством столбцов.

Выходные данные

Результат должен быть сохранён в архив `task1.zip` (без вложенных папок), содержащий два файла в кодировке UTF-8:

1. `cleaned.csv`

- Объединённые очищенные данные.
- Каждая строка содержит четыре вещественных числа, разделённых запятыми без пробелов.
- Строки отсортированы по возрастанию (численно).
- Числа сохраняются с полной точностью (без округления).
- Заголовок отсутствует.
- *Особый случай:* Если после очистки не осталось ни одной строки, файл должен быть пустым (0 байт).

2. `correlations.csv`

- Топ-5 пар столбцов с наибольшей абсолютной корреляцией.
- Формат строки: `{col_i},{col_j},{correlation}`
- Столбцы нумеруются с 0 (первый – 0, ..., четвёртый – 3). В паре $i < j$.
- Число `correlation` выводится с 6 знаками после запятой.
- Заголовок отсутствует.
- *Особый случай:* Если после очистки осталось менее 2 строк, корреляция не может быть вычислена. В этом случае файл должен быть пустым (0 байт). В противном случае файл содержит ровно 5 строк.

Требования

Выполните следующие шаги строго в указанном порядке:

1. **Извлечение и объединение:** Прочитайте все строки из всех подходящих CSV-файлов архива и объедините их в единый набор данных.
2. **Очистка данных:**
 - **Удаление строк с ошибками:** Удалите строку полностью, если выполняется хотя бы одно условие:
 - Содержится пустое поле (два разделителя подряд или пробелы между ними).
 - Содержится значение NaN, inf или -inf.

- Содержится хотя бы одно поле, которое невозможно интерпретировать как вещественное число.
 - Количество столбцов в строке отличается от 4.
- **Удаление дубликатов:** Удалите полные дубликаты строк. Строки считаются одинаковыми, если все четыре числа в них совпадают с точностью до машинного представления (`float64`).
3. **Сортировка:** Отсортируйте очищенный набор данных по возрастанию числового значения. Сортировка многоключевая: сначала по первому столбцу, при равенстве — по второму, затем по третьему, затем по четвёртому.
 4. **Анализ:** Для очищенного набора данных вычислите коэффициенты корреляции Пирсона между всеми уникальными парами столбцов (всего 6 пар для 4 столбцов). Выберите 5 пар с наибольшей абсолютной величиной корреляции $|r|$.
 5. **Сохранение результатов:** Сформируйте архив `task1.zip`, содержащий два файла описанного выше формата.

Правила сортировки файла `correlations.csv`: * По убыванию модуля корреляции $|correlation|$. * При равенстве модулей — по возрастанию `col_i`. * При равенстве `col_i` — по возрастанию `col_j`.

Критерии оценивания

№	Критерий	Баллы	Проверка
1.1	Корректность очистки данных (удаление пропусков, бесконечностей, нечисловых значений, дубликатов)	30	Сравнение результирующего набора с эталонным
1.2	Правильная сортировка очищенных данных (численная, многоключевая)	15	Проверка порядка строк в <code>cleaned.csv</code>
1.3	Корректное вычисление корреляций Пирсона	30	Сравнение значений корреляций с эталоном (погрешность $\leq 10^{-6}$)
1.4	Правильный формат и сортировка файла <code>correlations.csv</code>	15	Анализ структуры файла, порядка пар и форматирования чисел
1.5	Соответствие выходных файлов формату (кодировка, разделители, отсутствие заголовков)	10	Проверка технических требований
Итого		100	

Технические ограничения

- **Разрешённые библиотеки:** `zipfile`, `pandas`, `numpy`, `csv`, `re`, `os`.
- **Точность вычислений:** Не хуже 10^{-6} (для корреляции).
- **Обработка файлов:** Только `.csv` (регистронезависимо), только в корне архива.
- **Кодировка:** UTF-8 (вход и выход).
- **Выходной архив:** `task1.zip` (без вложенных папок).

Задача 2: Рекомендательная система на основе матричной факторизации

Условие задачи

Рекомендательные системы анализируют предпочтения пользователей для предложения релевантного контента (фильмов, музыки, товаров). Одним из классических подходов является матричная факторизация, позволяющая выявить скрытые признаки пользователей и объектов на основе известных оценок.

Вам предстоит реализовать алгоритм обучения модели матричной факторизации методом стохастического градиентного спуска (SGD). На вход подаются известные оценки пользователей объектам. Необходимо обучить модель, сохранить её промежуточные состояния для проверки корректности обучения и сделать предсказания для тестовых пар.

Теоретическая справка

Матричная факторизация

Модель аппроксимирует матрицу оценок R (размера $U \times I$) произведением двух матриц латентных факторов:

$$R \approx P \cdot Q^T$$

где P — матрица факторов пользователей ($U \times K$), Q — матрица факторов объектов ($I \times K$), K — размерность латентного пространства.

Предсказание и ошибка

Предсказание оценки для пользователя u и объекта i :

$$\hat{r}_{ui} = \mathbf{p}_u \cdot \mathbf{q}_i^T$$

Ошибка предсказания для известной оценки r_{ui} :

$$e_{ui} = r_{ui} - \hat{r}_{ui}$$

Обновление векторов (SGD с L2-регуляризацией)

Обновление векторов факторов производится по следующим формулам:

$$\mathbf{p}_u \leftarrow \mathbf{p}_u + \eta \cdot (e_{ui} \cdot \mathbf{q}_i - \lambda \cdot \mathbf{p}_u)$$

$$\mathbf{q}_i \leftarrow \mathbf{q}_i + \eta \cdot (e_{ui} \cdot \mathbf{p}_u^{\text{old}} - \lambda \cdot \mathbf{q}_i)$$

Важно: При обновлении $\mathbf{q}_i$ следует использовать значение $\mathbf{p}_u$ до его обновления на текущем шаге ($\mathbf{p}_u^{\text{old}}$).

Входные данные

Вам предоставлен ZIP-архив `task2_data.zip`.

- **Состав архива:** Два файла в кодировке UTF-8.
- **`ratings.csv`** — обучающая выборка.
 - Формат: `user_id, item_id, rating` (без заголовка).
 - `user_id, item_id` — целые числа (индексация с 0).
 - `rating` — целое число от 1 до 5.
 - Разделитель: запятая.
- **`test_pairs.csv`** — тестовые пары для предсказания.
 - Формат: `user_id, item_id` (без заголовка).
 - Гарантия: все пользователи и объекты из теста присутствуют в обучающих данных.

Выходные данные

Результат должен быть сохранён в архив `task2.zip` (без вложенных папок), содержащий три файла:

1. **`predictions.txt`**
 - Предсказанные оценки для пар из `test_pairs.csv`.
 - Каждая строка содержит одно целое число (от 1 до 5).
 - Порядок строк строго соответствует порядку пар в тестовом файле.
 - Файл не содержит заголовка и лишних пробелов.
 - Кодировка: UTF-8.
2. **`checkpoint_epoch_5.npy`**
 - Состояние матрицы пользовательских факторов P после завершения 5-й эпохи обучения.
 - Формат: бинарный файл NumPy (сохранение через `np.save`).
 - Матрица имеет размер (U, K) , тип `float64`.
3. **`checkpoint_epoch_10.npy`**
 - Состояние матрицы пользовательских факторов P после завершения 10-й эпохи обучения.
 - Формат: бинарный файл NumPy.
 - **Важно:** Нумерация эпох в именах файлов начинается с 1 (`epoch_5` = после 5-й эпохи).

Требования

Выполните следующие шаги строго в указанном порядке. Все параметры обучения фиксированы и не подлежат изменению.

1. Инициализация

- Определите число пользователей $U = \max(\text{user_id}) + 1$ и число объектов $I = \max(\text{item_id}) + 1$ по данным из `ratings.csv`.
- Задайте размерность латентного пространства $K = 10$.
- Установите фиксированный генератор случайных чисел с `seed 42` для воспроизводимости.
- Инициализируйте матрицу пользовательских факторов P размера (U, K) и матрицу объектных факторов Q размера (I, K) значениями из нормального распределения $\mathcal{N}(0, 0.1^2)$.

2. Обучение

- Параметры обучения: скорость обучения $\eta = 0.005$, коэффициент регуляризации $\lambda = 0.1$, количество эпох `epochs = 20`.
- Для каждой эпохи `epoch = 1 ... 20`:
 - Перемешайте все строки обучающего набора (используя установленный `seed`).
 - Для каждой тройки $(\text{user}, \text{item}, \text{rating})$ из перемешанных данных выполните обновление векторов согласно формулам в разделе «Теоретическая справка».
 - После завершения всех обновлений эпохи, если `epoch == 5` или `epoch == 10`, сохраните текущее состояние матрицы P в соответствующий файл.

3. Предсказание для тестовых пар

- Для каждой пары $(\text{user}, \text{item})$ из файла `test_pairs.csv` вычислите предсказание $\hat{r}_{ui} = \mathbf{p}_u \cdot \mathbf{q}_i^T$.
- Округлите результат до ближайшего целого числа (стандартное округление).
- Ограничьте результат диапазоном $[1, 5]$: $\max(1, \min(5, \text{rounded_pred}))$.

4. Сохранение результатов

- Сформируйте архив `task2.zip`, содержащий три файла описанного выше формата.

Критерии оценивания

Решение оценивается по двум независимым компонентам: точность реализации (совпадение чекпоинтов) и качество предсказаний.

№	Критерий	Баллы	Проверка
2.1	Совпадение чекпоинта эпохи 5 с эталоном	20	Сравнение файлов <code>checkpoint_epoch_5.npy</code> (погрешность $\leq 10^{-9}$)
2.2	Совпадение чекпоинта эпохи 10 с эталоном	20	Сравнение файлов <code>checkpoint_epoch_10.npy</code> (погрешность $\leq 10^{-9}$)
2.3	Точность предсказаний (RMSE) на приватном тесте	0–60	Вычисление RMSE между предсказанными и истинными оценками
Итого		100	

Шкала оценки RMSE:

RMSE	Баллы
≤ 0.20	60
≤ 0.30	50
≤ 0.40	40
≤ 0.50	30
≤ 0.60	20
> 0.60	0

Технические ограничения

- **Разрешённые библиотеки:** `numpy`, `pandas` (только для чтения CSV), `random`, `zipfile`.
- **Запрещено:** Использовать готовые реализации матричной факторизации (например, `surprise`, `implicit`), изменять параметры алгоритма.
- **Типы данных:** Вычисления с плавающей точкой (`float64`).
- **Воспроизводимость:** Обязательно использование фиксированного `seed` перед инициализацией весов и перемешиванием.
- **Кодировка:** UTF-8 для текстовых файлов.
- **Выходной архив:** `task2.zip` (без вложенных папок).

Задача 3: Классификация текстов (определение тональности отзывов)

Условие задачи

Анализ тональности текстов (sentiment analysis) является ключевой задачей обработки естественного языка. Вам предстоит разработать систему классификации англоязычных отзывов о фильмах на позитивные (1) и негативные (0).

Необходимо реализовать конвейер обработки данных, включающий предобработку текста, векторизацию методом «мешок слов» и обучение наивного байесовского классификатора. Модель должна быть обучена на предоставленной размеченной выборке и применена для предсказания меток на тестовом наборе.

Теоретическая справка

Предобработка текста

Для корректной работы моделей машинного обучения текст необходимо привести к единому формату:

- Приведение к нижнему регистру устраняет различия между словами с разным регистром
- Удаление специальных символов оставляет только значимые токены (буквы, цифры, пробелы)

Векторизация

CountVectorizer Метод преобразует коллекцию текстов в матрицу частот терминов. Параметр `stop_words` позволяет исключить часто встречающиеся слова, не несущие смысловой нагрузки.

Наивный байесовский классификатор

MultinomialNB — вероятностный классификатор, основанный на теореме Байеса. Эффективен для задач классификации текстов с мешком слов.

Входные данные

Вам предоставлен ZIP-архив `task3_data.zip`.

- **Состав архива:** Три файла в кодировке UTF-8.
- **`train_texts.txt`** — обучающие тексты.
 - Формат: 1000 строк, по одному отзыву в строке.

- **train_labels.txt** — метки классов для обучающей выборки.
 - Формат: 1000 строк, числа 0 или 1.
 - Гарантия: Порядок строк соответствует порядку текстов в train_texts.txt.
- **test_texts.txt** — тестовые тексты для предсказания.
 - Формат: 500 строк, отзывы без меток.

Выходные данные

Результат должен быть представлен в виде функции `predict`, возвращающей список предсказаний.

- **Тип возвращаемого значения:** `List[int]` (список целых чисел).
- **Длина списка:** 500 элементов (соответствует количеству тестовых текстов).
- **Значения элементов:** 0 или 1 (предсказанная метка тональности).
- **Порядок:** Строго соответствует порядку текстов в `test_texts.txt`.

Сигнатура функции:

```
from typing import List

def predict(train_texts: List[str], train_labels: List[int],
            test_texts: List[str]) -> List[int]:
    # реализация
    return predictions
```

Требования

Выполните следующие шаги строго в указанном порядке.

1. **Предобработка текста** Для каждого текста (как обучающего, так и тестового) выполните последовательно:
 - Приведите все символы к нижнему регистру.
 - Удалите все символы, кроме букв латинского алфавита (a–z, A–Z), цифр (0–9) и пробелов.
 - **Важно:** Знаки пунктуации, спецсимволы и апострофы должны быть удалены. Разбиение на слова осуществляется векторизатором по пробелам после очистки.
2. **Векторизация**
 - Используйте `CountVectorizer` из библиотеки `sklearn.feature_extraction.text`.

- Обязательно укажите параметр `stop_words='english'` для удаления английских стоп-слов.
- Вызовите метод `fit_transform` только на предобработанных обучающих текстах.
- Для тестовых текстов вызовите только метод `transform` (без повторного обучения).

3. Обучение классификатора

- Используйте `MultinomialNB` из библиотеки `sklearn.naive_bayes` с параметрами по умолчанию.
- Обучите модель на векторизованных обучающих данных и соответствующих метках.

4. Предсказание

- Для тестовых текстов получите предсказанные метки (целые числа 0 или 1) с помощью обученной модели.
- Верните список предсказаний в том же порядке, что и входные тестовые тексты.

Критерии оценивания

Решение оценивается по трём независимым компонентам.

№	Критерий	Баллы	Проверка
3.1	Корректность предобработки текста (регистр, удаление спецсимволов)	30	Статический анализ кода (наличие <code>lower()</code> и удаления символов)
3.2	Корректность использования моделей (<code>CountVectorizer</code> со стоп-словами, <code>MultinomialNB</code>)	20	Статический анализ кода
3.3	Точность классификации (Ассурасу) на приватном тесте	0–50	Динамическое тестирование на скрытом наборе
Итого		100	

Шкала оценки Accuracy:

Accuracy	Баллы
≥ 0.78	50
≥ 0.74	40
≥ 0.70	30
≥ 0.66	20
≥ 0.62	10
< 0.62	0

Технические ограничения

- **Разрешённые библиотеки:** `re`, `numpy`, `pandas` (опционально), `sklearn.feature_extraction.text.CountVectorizer`, `sklearn.naive_bayes.MultinomialNB`.
- **Запрещено:** Другие классификаторы (SVM, логистическая регрессия, нейросети); NLP-библиотеки (NLTK, SpaCy, TextBlob); стемминг и лемматизация; обращение к внешним файлам или сети.
- **Время выполнения:** Функция `predict` должна выполняться не более 60 секунд на предоставленных данных.
- **Память:** Не более 512 МБ.
- **Версия sklearn:** 1.0.0 или выше (гарантируется проверяющей системой).
- **Кодировка:** UTF-8 (входные файлы).
- **Тип возвращаемого значения:** Список целых чисел (`List[int]`), не `numpy`-массив.

Задача 4: Сегментация изображений методом пороговой обработки

Условие задачи

Сегментация изображений — один из фундаментальных этапов компьютерного зрения. Она позволяет выделить объекты, границы или другие значимые области на изображении. В этой задаче необходимо реализовать конвейер обработки изображений, основанный на градиентных методах и адаптивной пороговой обработке.

Вам дан архив `task4_data.zip`, содержащий 10 полутоновых изображений в формате PNG. Каждое изображение имеет размер 256×256 пикселей. На изображениях присутствуют объекты на контрастном фоне. Необходимо для каждого изображения выполнить последовательную обработку и сохранить результирующие бинарные маски.

Теоретическая справка

Фильтр Собеля

Оператор Собеля аппроксимирует градиент яркости изображения. Свёртка изображения с ядрами Собеля даёт приближённые значения производных G_x и G_y . Магнитуа градиента вычисляется по формуле:

$$G = \sqrt{G_x^2 + G_y^2}$$

Там, где яркость резко меняется (граница объекта), магнитуа градиента высока.

Метод Оцу

Метод Оцу автоматически выбирает порог бинаризации T , максимизируя межклассовую дисперсию:

$$\sigma_{between}^2(T) = \omega_0(T)\omega_1(T)(\mu_0(T) - \mu_1(T))^2 \rightarrow \max$$

где ω_0, ω_1 — доли пикселей классов, μ_0, μ_1 — их средние яркости.

Морфологическое закрытие

Закрытие (closing) — это дилатация, за которой следует эрозия:

$$A \bullet B = (A \oplus B) \ominus B$$

Оно заполняет небольшие отверстия и соединяет близкие фрагменты.

Входные данные

Вам предоставлен ZIP-архив `task4_data.zip`.

- **Состав архива:** 10 файлов с расширением `.png`.
- **Имена файлов:** `image1.png ... image10.png`.
- **Расположение:** Все файлы находятся в корневой директории архива (вложенные папки отсутствуют).
- **Формат файлов:**
 - PNG, размер 256×256 пикселей.
 - 8 бит, полутоновые (одноканальные, режим 'L').
 - Значения пикселей в диапазоне 0–255.

Выходные данные

Результат должен быть сохранён в архив `task4.zip` (без вложенных папок), содержащий 10 файлов:

- **Имена файлов:** `image1_mask.png ... image10_mask.png`.
- **Формат файлов:**
 - PNG, размер 256×256 пикселей.
 - Режим 'L' (полутоновое).
 - Значения пикселей: 0 (фон) и 255 (объект).
- **Кодировка имён файлов:** UTF-8.
- **Структура архива:** Все файлы должны находиться в корне архива (без папок).

Требования

Выполните следующие шаги строго в указанном порядке для каждого изображения:

1. **Загрузка и подготовка:**
 - Загрузите изображение как массив NumPy.
 - Преобразуйте данные к типу `float64` для дальнейших вычислений.
2. **Вычисление градиента:**
 - Используйте фильтр Собеля (Sobel filter) для вычисления градиента.
 - Вычислите магнитуду градиента: $G = \sqrt{G_x^2 + G_y^2}$.
 - Режим обработки границ: `mode='reflect'`.
3. **Бинаризация:**
 - Примените метод Оцу (Otsu's method) к изображению магнитуды градиента G .

- Создайте бинарную маску: пиксели выше порога → объект (1), пиксели ниже или равны порогу → фон (0).
- 4. Морфологическая обработка:**
 - Выполните морфологическое закрытие (closing) полученной бинарной маски.
 - Используйте квадратное структурное ядро размером 3×3.
- 5. Сохранение результата:**
 - Преобразуйте итоговую бинарную маску в тип `uint8` со значениями 0 (фон) и 255 (объект).
 - Сохраните изображение в формате PNG с режимом 'L'.
 - Имя выходного файла: {исходное_имя}_mask.png.
- 6. Упаковка:**
 - После обработки всех 10 изображений упакуйте полученные файлы в архив `task4.zip`.

Критерии оценивания

Решение оценивается по качеству сегментации на основе метрики IoU (Intersection over Union) между полученными масками и эталонными масками.

№	Критерий	Баллы	Проверка
4.1	Соответствие формата выходных файлов (размер 256×256, тип 'L', значения 0 и 255)	0	Автоматическая валидация всех 10 масок
4.2	Качество сегментации (средний IoU по 10 изображениям)	0–100	Сравнение с эталонными масками
Итого		100	

Важно: Если хотя бы одна маска не соответствует требованиям формата (размер, тип, значения пикселей), работа получает 0 баллов за задачу независимо от метрики IoU.

Шкала оценки среднего IoU:

Средний IoU	Баллы
≥ 0.90	100
≥ 0.85	80
≥ 0.80	60
≥ 0.75	40
≥ 0.70	20
< 0.70	0

Технические ограничения

- **Разрешённые библиотеки:** `numpy`, `PIL (Pillow)`, `scikit-image` (модули `filters`, `morphology`), `zipfile`, `os`.
- **Запрещено:** Использовать готовые реализации полной сегментации (например, `skimage.segmentation`), нейросетевые фреймворки (`TensorFlow`, `PyTorch`).
- **Режим границ для Собеля:** Обязательно `mode='reflect'`.
- **Ядро для закрытия:** 3×3 из единиц.
- **Тип данных для вычислений:** `float64` (преобразовать изображение перед расчётом градиента).
- **Формат выходных масок:** `PNG`, режим `'L'`, значения 0 и 255.
- **Выходной архив:** `task4.zip` (без вложенных папок).

Задача 5: Интерпретируемость модели дерева решений

Условие задачи

В областях, где важна прозрачность принятия решений (медицина, финансы, кредитование), интерпретируемость модели машинного обучения становится критически важной. Деревья решений позволяют объяснить предсказание, проследив путь от корня до листа, и оценить важность используемых признаков.

Вам предоставлен набор данных о клиентах банка и предобученная модель `DecisionTreeClassifier`, предсказывающая отток клиентов (`churn`: 1 — уход, 0 — остаётся). Необходимо проанализировать модель: выявить наиболее значимые признаки и объяснить предсказание для конкретного клиента на основе структуры дерева.

Результат работы должен быть сохранён в виде архива с двумя файлами: ранжированным списком признаков по важности и текстовым объяснением пути принятия решения для тестового клиента.

Теоретическая справка

Важность признаков

Важность признака в дереве решений вычисляется как нормированное суммарное уменьшение неопределённости (критерий Джини или энтропии) во всех узлах, где использовался данный признак. Сумма важностей всех признаков равна 1 (с точностью до машинной погрешности).

Путь решения (Decision Path)

Метод `decision_path` возвращает разрежённую матрицу в формате CSR (Compressed Sparse Row). Для извлечения индексов узлов для одного объекта используйте атрибуты `.indices` и `.indptr` разрежённой матрицы. Индексы узлов перечисляются в порядке прохождения от корня к листу.

```
path = model.decision_path(X_client)
node_indices = path.indices[path.indptr[0]:path.indptr[1]]
```

Количество узлов в пути

Включает корневой и листовый узлы. Для объекта, прошедшего через дерево от корня до листа, путь содержит все промежуточные узлы решения.

Входные данные

Вам предоставлен ZIP-архив `task5_data.zip`.

- **Состав архива:** Три файла в кодировке UTF-8.
- **`tree.pkl`** — предобученная модель.
 - Формат: бинарный pickle-файл (`DecisionTreeClassifier` из `sklearn`).
 - **Важно:** Класс `DecisionTreeClassifier` должен быть импортирован перед загрузкой модели.
- **`train.csv`** — обучающая выборка.
 - Формат: CSV с заголовком.
 - Последний столбец — целевая переменная `churn`.
 - Остальные столбцы — признаки (имена признаков в заголовке).
- **`sample_client.csv`** — данные одного клиента для анализа.
 - Формат: CSV с заголовком.
 - Те же признаки, что в `train.csv` (кроме столбца `churn`).
 - Одна строка данных (один клиент).

Выходные данные

Результат должен быть сохранён в архив `task5.zip` (без вложенных папок), содержащий два файла в кодировке UTF-8:

1. **`feature_ranking.csv`**
 - Ранжированный список признаков по важности.
 - Формат: CSV с заголовком `feature, importance`.
 - Разделитель — запятая (без пробелов).
 - Значения важности выводятся с 6 знаками после запятой.
 - Строки отсортированы по убыванию важности (от наиболее важного к наименее важному).
 - Файл не содержит лишних пустых строк.
2. **`explanation.txt`**
 - Объяснение предсказания для конкретного клиента.
 - Количество строк: строго 3 строки.

- Формат: Predicted class: {0 или 1} Number of nodes in path: {целое число} Nodes indices: {список индексов через запятую без пробелов}
- Между числами в списке индексов не должно быть пробелов (только запятые).
- Файл не содержит лишних пустых строк в начале или конце.

Требования

Выполните следующие шаги строго в указанном порядке:

1. Загрузка модели

- Импортируйте класс `DecisionTreeClassifier` из `sklearn.tree` до вызова `pickle.load()`.
- Загрузите модель из файла `tree.pkl` с помощью модуля `pickle`.

2. Важность признаков

- Прочитайте файл `train.csv`.
- Извлеките имена признаков из заголовков, исключив последний столбец `churn`.
- Получите значения важности признаков из атрибута модели `feature_importances_`.
- Сформируйте список пар (признак, важность) и отсортируйте его по убыванию важности.
- Сохраните результат в файл `feature_ranking.csv` согласно требованиям раздела «Выходные данные».

3. Анализ предсказания для клиента

- Прочитайте файл `sample_client.csv` (одна строка данных, признаки без целевой переменной).
- С помощью загруженной модели получите предсказанный класс: `model.predict(X_client)`.
- Получите путь решения (decision path) для данного клиента, используя метод `model.decision_path(X_client)`.
- Извлеките индексы узлов, через которые прошёл объект, с помощью атрибутов `.indices` и `.indptr` разрежённой матрицы.
- Подготовьте объяснение согласно шаблону в разделе «Выходные данные».

4. Сохранение результатов

- Сформируйте архив `task5.zip`, содержащий два файла описанного выше формата.
- Архив не должен содержать вложенных папок.

Критерии оценивания

Решение оценивается по трём независимым компонентам.

№	Критерий	Баллы	Проверка
5.1	Корректное извлечение и сортировка важности признаков	40	Сравнение с эталонным файлом <code>feature_ranking.csv</code> (порядок и значения с погрешностью $\leq 10^{-6}$)
5.2	Правильное предсказание для заданного клиента	20	Сравнение предсказанного класса с эталонным значением
5.3	Корректное восстановление пути решения (индексы узлов)	40	Сравнение последовательности индексов узлов с эталоном (полное совпадение)
Итого		100	

Важно: Любое несоответствие формата выходных файлов (имена столбцов, количество знаков, структура `explanation.txt`, пробелы в списке индексов) приводит к снижению баллов по соответствующему критерию.

Технические ограничения

- **Разрешённые библиотеки:** `numpy`, `pandas`, `pickle`, `sklearn.tree.DecisionTreeClassifier`.
- **Запрещено:** Сторонние библиотеки интерпретации (`eli5`, `shap`, `lime`); переобучение модели.
- **Загрузка модели:** Обязательно импортировать `DecisionTreeClassifier` перед `pickle.load()`.
- **Точность вычислений:** `float64` (стандартно для `sklearn`).
- **Кодировка файлов:** UTF-8 (вход и выход).
- **Выходной архив:** `task5.zip` (без вложенных папок, файлы в корне архива).
- **Формат `feature_ranking.csv`:** Заголовок `feature,importance`, 6 знаков после запятой для важности.
- **Формат `explanation.txt`:** Строго 3 строки, без пробелов в списке индексов, без лишних пустых строк.

Задача 6: Сжатие изображений с помощью метода главных компонент (PCA)

Условие задачи

Анализ главных компонент (PCA) — стандартный метод уменьшения размерности данных, позволяющий сжимать информацию с минимальными потерями. Этот метод широко применяется для сжатия изображений, выделения признаков и визуализации многомерных данных.

В этой задаче необходимо применить PCA для сжатия изображений лиц, оценить качество восстановления и определить оптимальное количество компонент для сохранения заданной доли дисперсии. Вам предстоит обучить модель на предоставленной выборке, восстановить тестовое изображение и вычислить метрики качества сжатия.

Результат работы должен быть сохранён в виде архива с двумя файлами: значением ошибки восстановления и оптимальным количеством главных компонент.

Теоретическая справка

Метод главных компонент (PCA)

PCA находит новые оси (главные компоненты), вдоль которых дисперсия данных максимальна. Первые компоненты содержат наибольшую информацию о данных. Модель обучается на матрице данных X размера (n, d) , где n — число объектов, d — число признаков.

Объяснённая дисперсия

Доля дисперсии, объясняемая каждой главной компонентой, хранится в атрибуте `explained_variance_ratio_`. Сумма всех значений равна 1 (с точностью до машинной погрешности).

Среднеквадратичная ошибка (MSE)

Для оригинального вектора x и восстановленного вектора $\hat{x}$ длины $d = 1024$:

$$\text{MSE} = \frac{1}{d} \sum_{i=1}^d (x_i - \hat{x}_i)^2$$

Чем меньше MSE, тем точнее восстановление изображения.

Порог дисперсии

Необходимо найти минимальное целое число k , удовлетворяющее условию:

$$\sum_{j=1}^k \text{explained_variance_ratio}_j \geq 0.95$$

Входные данные

Вам предоставлен ZIP-архив `task6_data.zip`.

- **Состав архива:** Два файла в формате NumPy `.npy`.
- **`faces.npy`** — обучающая выборка.
 - Форма массива: `(200, 1024)`.
 - Тип данных: `float32`.
 - Описание: 200 изображений лиц, векторизованных в 1024 признака (изображения 32×32 пикселя).
- **`test_face.npy`** — тестовое изображение.
 - Форма массива: `(1024,)` или `(1, 1024)`.
 - Тип данных: `float32`.
- **Гарантии:**
 - Все значения нормализованы к диапазону `[0, 1]`.
 - Сумма объяснённой дисперсии достигает 95% при разумном количестве компонент.

Выходные данные

Результат должен быть сохранён в архив `task6.zip` (без вложенных папок), содержащий два файла в кодировке UTF-8:

1. **`mse.txt`**
 - Одно вещественное число — значение MSE, вычисленное между исходным и восстановленным тестовым изображением.
 - Формат: число с 6 знаками после запятой (например, `0.012345`).
 - Файл не содержит заголовков, лишних пробелов или пустых строк.
2. **`n_components.txt`**
 - Одно целое число — минимальное количество компонент k , обеспечивающее сохранение $\geq 95\%$ дисперсии.
 - Формат: целое число (например, `42`), без десятичной части.
 - Файл не содержит заголовков, лишних пробелов или пустых строк.

Требования

Выполните следующие шаги строго в указанном порядке:

1. Обучение модели сжатия

- Загрузите обучающие данные из файла `faces.npy`.
- Создайте объект PCA из `sklearn.decomposition` с параметрами:
 - `n_components=25`
 - `svd_solver='full'` (обеспечивает детерминированность вычислений).
- Обучите модель на данных `faces.npy` (центрирование выполняется автоматически).

2. Восстановление тестового изображения

- Загрузите тестовое изображение из файла `test_face.npy`.
- Если массив имеет форму `(1024, )`, преобразуйте его к форме `(1, 1024)`.
- Спроецируйте тестовое изображение на 25 главных компонент (метод `transform`).
- Выполните обратное преобразование в исходное пространство (метод `inverse_transform`).
- Получите восстановленное изображение (массив той же формы, что и входное тестовое).

3. Вычисление ошибки восстановления

- Вычислите среднеквадратичную ошибку (MSE) между исходным тестовым изображением и восстановленным изображением по формуле из раздела «Теоретическая справка».
- Сохраните результат в файл `mse.txt` согласно требованиям раздела «Выходные данные».

4. Оценка объяснённой дисперсии

- Создайте отдельный объект PCA без ограничения числа компонент (параметр `n_components` не указывайте или установите `None`).
- Обучите эту модель на тех же данных `faces.npy`.
- Извлеките массив `explained_variance_ratio_`.
- Найдите минимальное целое число k , такое что сумма первых k значений ≥ 0.95 .
- Сохраните найденное k в файл `n_components.txt` согласно требованиям раздела «Выходные данные».

5. Сохранение результатов

- Сформируйте архив `task6.zip`, содержащий два файла описанного выше формата.
- Архив не должен содержать вложенных папок.

Критерии оценивания

Решение оценивается по точности вычислений. Баллы начисляются только за полное совпадение с эталонными значениями (с учётом допустимой погрешности для MSE).

№	Критерий	Баллы	Проверка
6.1	Корректное вычисление MSE восстановления	50	Сравнение с эталонным значением (погрешность $\leq 10^{-6}$)
6.2	Корректное определение количества компонент для 95% дисперсии	50	Точное совпадение целого числа k с эталоном
Итого		100	

Важно: Любое несоответствие формата выходных файлов (количество знаков, тип числа, наличие лишних символов) приводит к снижению баллов по соответствующему критерию.

Технические ограничения

- **Разрешённые библиотеки:** `numpy`, `sklearn.decomposition.PCA`.
- **Запрещено:** Использовать другие методы уменьшения размерности; реализовывать PCA вручную.
- **Параметры PCA:**
 - Для сжатия: `n_components=25`, обязательно `svd_solver='full'`.
 - Для поиска k : без ограничения (все компоненты).
- **Центрирование:** Использовать стандартное поведение `sklearn` (автоматическое, не вычитая среднее вручную).
- **Точность вычислений:** `float64` (стандартно для PCA в `sklearn`).
- **Кодировка файлов:** UTF-8.
- **Выходной архив:** `task6.zip` (без вложенных папок, файлы в корне архива).
- **Формат `mse.txt`:** Вещественное число с 6 знаками после запятой, без лишних пробелов.
- **Формат `n_components.txt`:** Целое число, без десятичной части, без лишних пробелов.

Задача 7: Многоагентное обучение с подкреплением (Тактический бой)

Введение

Вам предстоит разработать искусственный интеллект для управления командой из трёх автономных агентов в среде командного боя с частичной наблюдаемостью. Агенты вашей команды (синие) используют единую нейронную сеть с общими весами (parameter sharing), но принимают решения независимо на основе общей карты видимости. Ваше решение будет участвовать в турнире против агентов, разработанных другими участниками. Итоговая оценка зависит от результатов матчей, эффективности боя, координации и стабильности.

Параметры среды

Параметр	Значение
Размер карты	12×12 клеток
Команды	2 (Синие против Жёлтых)
Агентов в команде	3
Максимум шагов	200
Здоровье агента (HP)	3 единицы
Поле зрения агента	квадрат 5×5 вокруг себя
Видимость команды	объединение зон видимости всех живых агентов
Количество ловушек	4
Интервал движения ловушек	каждые 2 шага
Вероятность движения ловушки	0.7

Объекты на карте (кодировка наблюдения)

Каждому агенту доступно наблюдение — карта 12×12, где числа обозначают:

Код	Тип объекта	Описание
0	Туман войны	клетка не видна ни одному союзнику
1	Пустая клетка	свободное пространство, разведано
2	Стена	непроходимое препятствие
3	Этот агент	позиция самого агента
4	Союзник	другой агент своей команды

Код	Тип объекта	Описание
5	Враг	агент противника
6	Ловушка	смертельная ловушка

Важно: `team_obs` строится на основе объединённой видимости всех живых агентов команды. Если хотя бы один союзник видит клетку, она отображается на карте (код > 0). Ловушки и враги видны только в пределах общей видимости команды.

Действия агента

Каждый агент на каждом шаге выбирает одно действие:

Код	Действие	Описание
0	Движение вверх	переместиться на 1 клетку вверх ($row - 1$)
1	Движение вправо	переместиться на 1 клетку вправо ($col + 1$)
2	Движение вниз	переместиться на 1 клетку вниз ($row + 1$)
3	Движение влево	переместиться на 1 клетку влево ($col - 1$)
4	Атака	атаковать врага в соседней клетке

Система координат: Карта представляет собой матрицу `[row, col]`.

- Row 0 — верхний край карты, Row 11 — нижний край.
- Col 0 — левый край карты, Col 11 — правый край.
- Действие 0 (**Вверх**) уменьшает индекс строки ($row - 1$).
- Действие 2 (**Вниз**) увеличивает индекс строки ($row + 1$).

Действие «ожидание» отсутствует. Если агент не может выполнить выбранное движение (стена, занятая клетка), он остаётся на месте и получает штраф.

Важно: Действие «Атака» (код 4), выбранное когда рядом нет врагов, **штрафуется** (подробнее см. раздел наград). Агент при этом не перемещается и не наносит урон.

Механика боя

1. **Атака:** если агент выбрал действие 4 и в соседней клетке (4 направления) есть враг, тот теряет 1 НР. Если рядом несколько врагов, атакуется враг с наименьшим НР (при равенстве — первый по порядку).
2. **Смерть:** при $НР \leq 0$ агент мгновенно исчезает с карты и больше не может действовать.
3. **Ловушка:**

- Если агент оказывается на клетке с ловушкой (путём движения), он немедленно погибает (НР становится 0).
 - Если ловушка перемещается на клетку, занятую агентом, агент также мгновенно погибает.
 - Ловушки каждые 2 шага могут перемещаться на одну клетку (с вероятностью 0.7). При перемещении они стремятся к ближайшему агенту (или движутся случайно, если агентов нет).
4. **Стены:** движение в стену или в клетку, занятую другим агентом, невозможно (действие игнорируется, агент остаётся на месте и получает штраф).

Окончание матча

Матч заканчивается, если выполнено одно из условий: - Все агенты одной команды уничтожены. - Достигнут лимит шагов (200).

Определение победителя: - Если одна команда полностью уничтожена — побеждает другая. - Если по истечении 200 шагов остались живые с обеих сторон, побеждает команда, у которой больше живых агентов. - Если живых поровну — сравнивается суммарное НР живых агентов. - Если и НР равны — объявляется ничья.

Теоретическая справка

Многоагентное обучение с подкреплением (MARL)

Задача относится к классу кооперативных многоагентных систем с частичной наблюдаемостью (Cooperative MARL with Partial Observability). Формально её можно описать как марковскую игру (Markov game). Каждый агент стремится максимизировать ожидаемую суммарную дисконтированную награду:

$$\max \mathbb{E} \left[\sum_{t=0}^T \gamma^t r_t \right]$$

где $T \leq 200$ — максимальная длина эпизода, γ — коэффициент дисконтирования (обычно 0.95–0.99), а r_t — награда, полученная на шаге t .

Роль нейронных сетей в обучении с подкреплением

В отличие от классического обучения с учителем, где для каждого входного примера известна правильная метка (например, класс объекта), в обучении с подкреплением агент учится на собственном опыте взаимодействия со средой. У него нет прямых указаний, какое

действие правильно в каждом состоянии. Вместо этого он получает **сигнал награды** — скалярное значение, которое оценивает качество его действий. Цель агента — максимизировать суммарную накопленную награду за эпизод.

Нейронная сеть в RL выступает в роли **аппроксиматора функции**, которая связывает состояния среды с действиями. В зависимости от используемого алгоритма сеть может предсказывать:

- **Q-функцию** (ценность действия) — ожидаемую суммарную награду, которую агент получит, если в данном состоянии выполнит определённое действие, а затем будет действовать оптимально. В методе **DQN (Deep Q-Network)** сеть обучается минимизировать **TD-ошибку** (Temporal Difference error):

$$\mathcal{L} = \mathbb{E} \left[\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)^2 \right]$$

где $Q(s, a)$ — текущая оценка сети для состояния s и действия a , r — полученная награда, s' — следующее состояние. Разница между «целевым» значением $r + \gamma \max_{a'} Q(s', a')$ и текущим предсказанием и есть ошибка, которая затем распространяется обратно через сеть, корректируя веса.

- **Политику** (вероятности действий) — напрямую выдаёт распределение вероятностей действий. В методах градиента политики (policy gradient) сеть обновляется в сторону увеличения вероятности действий, которые привели к высокой награде (например, с использованием REINFORCE или PPO).
- **Функцию ценности состояния** $V(s)$ — ожидаемую награду из состояния s , что используется в алгоритмах типа Actor-Critic.

Как награды управляют обучением

На каждом шаге агент получает от среды награду r_t (которая может быть как положительной, так и отрицательной). Эта награда агрегируется в возврат $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$. Нейронная сеть настраивается так, чтобы её предсказания (например, $Q(s, a)$) как можно точнее соответствовали ожидаемому возврату при выборе действия a в состоянии s .

Процесс обучения итеративен:

1. Агент взаимодействует со средой, сохраняя переходы (s, a, r, s') в буфер опыта (replay buffer).
2. Случайная выборка из буфера используется для вычисления целевых значений и функции потерь.
3. Градиент функции потерь вычисляется и применяется для обновления весов сети (обратное распространение).

4. Периодически целевая сеть (target network) синхронизируется с основной для стабилизации обучения.

Таким образом, нейронная сеть постепенно учится предсказывать долгосрочные последствия действий, опираясь только на получаемые награды, без внешних «правильных ответов».

Архитектура Parameter Sharing

Все агенты команды используют единую нейронную сеть с общими весами. Это позволяет эффективно обучать политику на опыте всех трёх агентов одновременно, не увеличивая число параметров. Опыт каждого агента добавляется в общий буфер, и сеть обновляется с учётом всех переходов, что ускоряет сходимость.

Идентификатор агента (agent_id): В наблюдение передаётся agent_id (0, 1, 2).

- Участник **может использовать** этот идентификатор как входной признак для создания специализированных ролей (асимметричная политика).

- Участник **может игнорировать** этот идентификатор для создания универсальных агентов (симметричная политика).

Выбор стратегии зависит от архитектуры решения.

Структура наблюдения (State)

Каждый агент получает словарь state со следующими полями:

Поле	Тип	Описание
team_obs	np.array(12, 12)	карта видимости команды (коды 0–6)
agent_id	int	ID агента в команде (0, 1, 2)
hp	int	текущее здоровье (1–3)
step	int	номер текущего шага (0–199)
alive_teammates	int	количество живых союзников
alive_enemies	int	количество живых врагов

Предобработка данных (Рекомендация)

Для эффективного обучения рекомендуется следующая предобработка входных данных:

1. **One-Hot Encoding** для карты наблюдения: Коды объектов (0–6) являются **категориальными**, а не порядковыми. Для нейронной сети значение «Ловушка (6)» не должно восприниматься как «больше» или «лучше», чем «Стена (2)».
Рекомендация: Преобразуйте карту 12×12 в тензор [7, 12, 12], где каждый

канал соответствует бинарной маске одного типа объекта. Это позволит свёрточной сети учиться уникальным пространственным паттернам для каждого типа объекта.

2. **Объединение с скалярными признаками:** Скалярные признаки (hp, step, agent_id и т.д.) следует нормализовать (например, разделить на максимальное значение) и объединить с выходом свёрточных слоёв через операцию concat (конкатенацию) перед полносвязными слоями.

Пример архитектуры (Референс):

- **Вход карты:** 7 каналов (One-Hot) $\times 12 \times 12$.
- **Свёрточный блок:** 2 слоя Conv2d (32 и 64 фильтра) + MaxPool.
- **Вход скаляров:** 5 признаков (нормализованных).
- **Объединение:** Flatten карты + Concat со скалярами.
- **Полносвязный блок:** MLP (например, $512 \rightarrow 256$ нейронов).
- **Выход:** 5 значений (Q-значения для действий).

Комбинирование нейросетевых политик с эвристиками

В практических задачах игрового ИИ часто используется гибридный подход: нейросетевая политика, обученная с подкреплением, дополняется детерминированными эвристиками. Это позволяет объединить адаптивность обучения с гарантированной безопасностью и ускорением принятия решений.

Зачем это нужно:

1. **Защита от грубых ошибок:** Нейронная сеть может ошибаться даже после обучения, например, предлагать движение в стену или ловушку, если эти объекты видны на карте. Эвристический фильтр может заблокировать такое действие и выбрать безопасную альтернативу (например, случайное допустимое направление или атаку). Это повышает выживаемость агентов в критических ситуациях.
2. **Ускорение обучения:** Сети не нужно тратить ресурсы на изучение очевидных правил среды (например, «в стены нельзя ходить»), если эти правила уже заложены в эвристику. Это позволяет сконцентрировать вычислительные мощности на освоении сложных тактических приёмов.
3. **Приоритет важных действий:** В некоторых ситуациях эвристика может переопределять решение сети, чтобы гарантированно достичь тактического преимущества. Например, если враг находится в соседней клетке и имеет всего 1 HP, действие «атака» может быть выполнено безусловно (добивание), даже если сеть предлагает движение.
4. **Стабильность и надёжность:** Эвристики снижают variance поведения, делая агента менее подверженным случайным сбоям сети.

Рекомендуемые области применения эвристик:

- **Фильтрация действий:** перед выполнением действия, предложенного сетью, проверьте, не приведёт ли оно к столкновению со стеной или ловушкой, если целевая клетка видна в `team_obs`. Если действие недопустимо, можно:
 - выбрать случайное безопасное направление из доступных;
 - выбрать действие «атака», если оно возможно и есть враг;
 - оставить агента на месте (хотя это может привести к штрафу за застой).
- **Приоритет атаки:** если в соседних клетках есть враг с $HP=1$, и действие «атака» доступно, можно всегда выбирать атаку, игнорируя выход сети. Это гарантирует устранение угрозы.
- **Избегание ловушек:** если в зоне видимости есть ловушка, можно временно повысить штраф за движение в её сторону (через внутреннюю мотивацию) или явно запретить такие движения.

Важно: Полное отсутствие нейросетевой компоненты (чисто эвристическое решение) допускается, но может снизить итоговые баллы за критерии «**Стабильность**» и «**Координация**». Эвристики, как правило, не способны адаптироваться к непредвиденным ситуациям и вырабатывать сложные кооперативные стратегии так же эффективно, как обученная сеть. Однако разумное сочетание нейросети и эвристик является стандартной практикой в современных игровых ИИ и позволяет достичь наилучших результатов.

Реализация: Эвристики должны быть реализованы внутри метода `choose_action` класса `Agent` в файле `solution.py`. Они могут анализировать входное состояние и модифицировать выходное действие, полученное от нейросети, или полностью заменять его. Важно, чтобы эвристики не нарушали временные ограничения (не более 0.1 с на вызов) и не использовали информацию, недоступную агенту (например, полную карту).

Функция наград среды

Среда возвращает общую командную награду за каждый шаг. Награды предназначены для ускорения сходимости обучения, но **не являются прямым критерием оценки решения** (см. раздел «Критерии оценивания»).

Полный список наград с точными значениями приведён в файле `environment.py`.
Ключевые события:

Категория	Примеры событий	Назначение
Боевые действия	Урон врагу (+2/ед.), убийство (+10), получение урона (−2/ед.)	Стимулирование боя
Пустая атака	Попытка атаки без цели (−1.5)	Штраф за бесполезные действия
Координация	Совместная атака ≥ 2 агентов (+5)	Поощрение командной работы

Категория	Примеры событий	Назначение
Навигация	Посещение новых клеток (+2), штраф за стены (-2), штраф за застой (-1)	Исследование карты
Итоговые	Победа в матче (+50), поражение (-50)	Глобальная цель

Важно: Награды среды являются плотными (dense rewards) — агент получает обратную связь на каждом шаге, а не только в конце эпизода. Однако оптимизация под эти награды может не совпадать с турнирными метриками, поэтому рекомендуется использовать **внутреннюю мотивацию**.

Внутренняя мотивация агента

Поскольку файл среды (`environment.py`) редактировать запрещено, участники могут реализовать **дополнительную функцию наград внутри своего агента** для лучшей оптимизации под критерии турнира.

Внутренняя мотивация (Intrinsic Motivation) — это дополнительные награды, которые агент генерирует внутри своей архитектуры, независимо от наград среды. Они добавляются к сигналу среды в процессе обучения (но не используются в инференсе).

Тип внутренней мотивации	Описание	Пример реализации
Curiosity-driven	Награда за исследование новых состояний	Бонус за посещение клеток с низкой частотой
Goal-conditioned	Награда за достижение подцелей	+1 за сближение с врагом, даже без атаки
Hierarchical	Награды для разных уровней иерархии	Высокоуровневая политика ставит цели низкоуровневой
Reward shaping	Определяемые участником награды в <code>train.py</code>	<code>internal_reward = f(state, action)</code>

Рекомендации: - Используйте внутреннюю мотивацию для выравнивания наград среды с критериями оценки. - Избегайте конфликта между наградой среды и внутренней наградой (суммируйте их). - Внутренняя мотивация особенно полезна для иерархических архитектур.

Приоритет критериев оценки

Обратите внимание: **окончательная оценка решения определяется критериями из раздела «Критерии оценивания»**, а не суммарной наградой от среды.

Приоритет	Критерий	Вес в итоговой оценке
1	Победы в матчах	50 баллов
2	Эффективность боя (kill_diff)	25 баллов
3	Координация	15 баллов
4	Стабильность	10 баллов

Вывод: Оптимизация под награды среды может не привести к победе в турнире. Рекомендуется использовать награды среды как вспомогательный сигнал, а основную оптимизацию проводить под критерии оценки через внутреннюю мотивацию и архитектуру агента.

Входные данные

Вам предоставлен ZIP-архив `task7_data.zip`.

Состав архива:

Файл	Описание	Редактирование
<code>environment.py</code>	Моделирование среды (класс <code>Environment</code>)	Запрещено
<code>game.py</code>	Управление циклом эпизода (класс <code>Game</code>)	Запрещено
<code>visualizer.py</code>	Визуализация процесса боя (<code>PyGame</code>)	Запрещено
<code>local_tester.py</code>	Автоматическое тестирование и оценка	Запрещено
<code>solution.py</code>	Шаблон для реализации агента (класс <code>Agent</code>)	Разрешено
<code>requirements.txt</code>	Список зависимостей	Запрещено

Генерация карты:

При каждом сбросе среды (`reset`) происходит следующее:

1. Создаётся пустая карта 12×12.
2. Случайно размещаются от 8 до 15 стен с проверкой связности всех проходимых клеток.
3. Размещаются 4 ловушки на свободных клетках (не стены).
4. Агенты обеих команд размещаются случайно с ограничениями:
 - Агенты не могут появляться на клетках с ловушками или стенами.
 - Агенты разных команд не могут быть соседями (клетки, смежные по стороне или углу, т.е. расстояние Чебышева ≤ 1) в начальный момент.
 - Агенты не могут быть на клетках, соседних с ловушками (8 направлений).

Локальное тестирование:

Для отладки и оценки решения предоставляется скрипт `local_tester.py`. Он позволяет запускать матчи против встроенного бота-противника (жёлтая команда) и

рассчитывает итоговый балл в соответствии с критериями задачи (победы, эффективность боя, координация, стабильность). Бот-противник обладает следующими особенностями:

- Видит полную карту (без тумана войны).
- Случайно выбирает одну из стратегий: aggressive, passive, fleeing, explorer.
- Принимает решения на основе полной информации, что делает его непредсказуемым и позволяет проверить адаптивность вашего решения.

Скрипт поддерживает несколько режимов работы и параметров командной строки:

Параметр	Описание
--episodes N или -e N	Запустить N эпизодов (матчей) в фоновом режиме без графики. По умолчанию N = 10. По окончании выводится детальная статистика и итоговый балл (0–100).
--visualize или -v	Запустить один эпизод с графической визуализацией (требуется установленный pygame). Позволяет наблюдать за поведением агентов в реальном времени.
--fullmap или -f	Используется только вместе с --visualize. Отображает полную карту без тумана войны, что полезно для отладки (показывает истинное положение врагов, ловушек и т.д.).

Примеры использования:

```
# Запустить 100 тестовых эпизодов и получить оценку
python local_tester.py --episodes 100
```

```
# Запустить один эпизод с визуализацией (стандартный вид)
python local_tester.py --visualize
```

```
# Запустить визуализацию с отключённым туманом войны
python local_tester.py -v -f
```

Интерпретация результатов:

После выполнения серии тестов скрипт выводит:

- Количество побед, ничьих и поражений.
- Детализацию баллов по каждому критерию (победы, эффективность боя, координация, стабильность) с пороговыми значениями.
- Итоговый балл (максимум 100).

На основе полученных результатов скрипт даёт рекомендации по улучшению решения. Например, если эффективность боя (kill_diff) ниже 1.5, рекомендуется улучшить тактику атаки; если координация (coord_rate) ниже 0.20 – усилить командные взаимодействия.

Перед отправкой решения обязательно убедитесь, что оно корректно проходит локальные тесты без ошибок.

Выходные данные

Для участия в соревновании участники отправляют один архив `task7.zip`, содержащий только два файла:

`solution.py`

Реализация класса `Agent`.

Обязательные методы:

Метод	Назначение
<code>__init__</code>	Инициализация модели, загрузка весов из <code>weights.pt</code>
<code>choose_action(state)</code>	Принятие решения на основе наблюдения

- Кодировка: UTF-8.
- В `__init__` должна присутствовать загрузка весов из файла `weights.pt`, если он существует. Для совместимости с сервером используйте `torch.load(..., map_location='cpu')`.
- Метод `choose_action` должен работать в режиме инференса (`model.eval()`) и возвращать целое число от 0 до 4.
- **Важно:** Файл `solution.py` предназначен только для инференса. Методы обучения не должны вызываться средой во время тестирования.

`weights.pt`

Сохранённые веса обученной модели.

- Формат: бинарный файл PyTorch (`state_dict`).
- Ограничение размера: не более 15 МБ.
- Должен загружаться в методе `__init__` класса `Agent`.

Важно: При загрузке используйте параметр `map_location='cpu'` для обеспечения совместимости.

Обучение проводится локально участником. В финальный архив для соревнования входят только файлы инференса (`solution.py` + `weights.pt`).

Требования

Выполните следующие шаги строго в указанном порядке:

1. **Локальная отладка:**
 - Запустите локальный тест: `python local_tester.py`.
 - Убедитесь, что решение работает без ошибок.
2. **Создание обучающего скрипта (опционально):**
 - Создайте файл `train.py` для обучения модели.
 - Реализуйте цикл обучения с сохранением весов в `weights.pt`.
 - Обучите модель до достижения удовлетворительных результатов.
 - Используйте внутреннюю мотивацию для выравнивания с критериями оценки.
3. **Реализация класса агента:**
 - Создайте класс `Agent` в файле `solution.py`.
 - Реализуйте метод `__init__` для инициализации модели и загрузки весов.
4. **Предобработка наблюдения:**
 - Преобразуйте словарь `state` в тензор, пригодный для подачи в нейронную сеть (например, One-Hot карты + дополнительные признаки).
 - При необходимости примените эвристики безопасности для фильтрации действий.
5. **Инференс:**
 - В методе `choose_action` выполните прямой проход через сеть (forward pass).
 - Верните действие с максимальным значением (`argmax`).
 - Режим модели должен быть `eval()` (градиенты не вычисляются).
6. **Финальная проверка:**
 - Запустите полный локальный тест: `python local_tester.py -- episodes 100`.
 - Убедитесь в отсутствии ошибок перед отправкой.
7. **Подготовка к отправке:**
 - Убедитесь, что `solution.py` содержит только код для инференса.
 - Файл `weights.pt` должен загружаться корректно.

Критерии оценивания

Локальное тестирование: Решение оценивается на основе N эпизодов (в локальном тестировании N=100).

№	Критерий	Баллы	Проверка
7.1	Победы в матчах	50	$\text{win_score} = (\text{победы} + 0.5 \cdot \text{ничьи}) / N * 50$
7.2	Эффективность боя	25	$\text{kill_diff} = (\text{убито синими} - \text{убито жёлтыми}) / N$
7.3	Координация	15	Доля шагов с ≥ 2 атаками синих агентов (любых попыток атаки)
7.4	Стабильность	10	Дисперсия итоговой командной награды за матч по всем эпизодам

Шкала оценки эффективности (kill_diff):

kill_diff × 10	Баллы
≥ 2.5	25
≥ 1.5	20
≥ 0.5	10
< 0.5	0

Шкала оценки координации (coord_rate):

coord_rate	Баллы
≥ 0.30	15
≥ 0.20	10
≥ 0.10	5
< 0.10	0

Финальное соревнование:

Окончательные баллы за задачу начисляются по результатам матчей против команд других участников на сервере проверки: - Каждый участник играет против всех остальных участников. - Итоговый рейтинг определяется нормированной суммой баллов (600 баллов - максимум, при победе одного игрока всех других).

Технические ограничения

1. **Разрешённые библиотеки:**
 - `numpy`, `torch`, `pygame` (только для визуализации).
 - Стандартные модули Python (`collections`, `random`, `math`, `os`).
2. **Запрещено:**
 - Изменение файлов среды (`environment.py`, `game.py`, `visualizer.py`, `local_tester.py`).
3. **Веса модели:**
 - Файл `weights.pt` не должен превышать 15 МБ.
4. **Фреймворк для нейросетей:**
 - Разрешён только PyTorch.
5. **Ошибки запуска:**
 - Если решение содержит ошибки при локальном запуске, оно не будет проверяться на сервере.
 - Участник получает 0 баллов за задачу при невозможности запуска решения.
6. **Эвристики:**
 - Разрешено использование эвристик в комбинации с нейронной сетью.
 - Эвристики должны быть реализованы внутри `solution.py`.
 - Эвристики не должны нарушать другие технические ограничения.
7. **Загрузка модели:**
 - В методе `__init__` обязательно используйте `torch.load(..., map_location='cpu')` для обеспечения кросс-платформенной совместимости.