

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

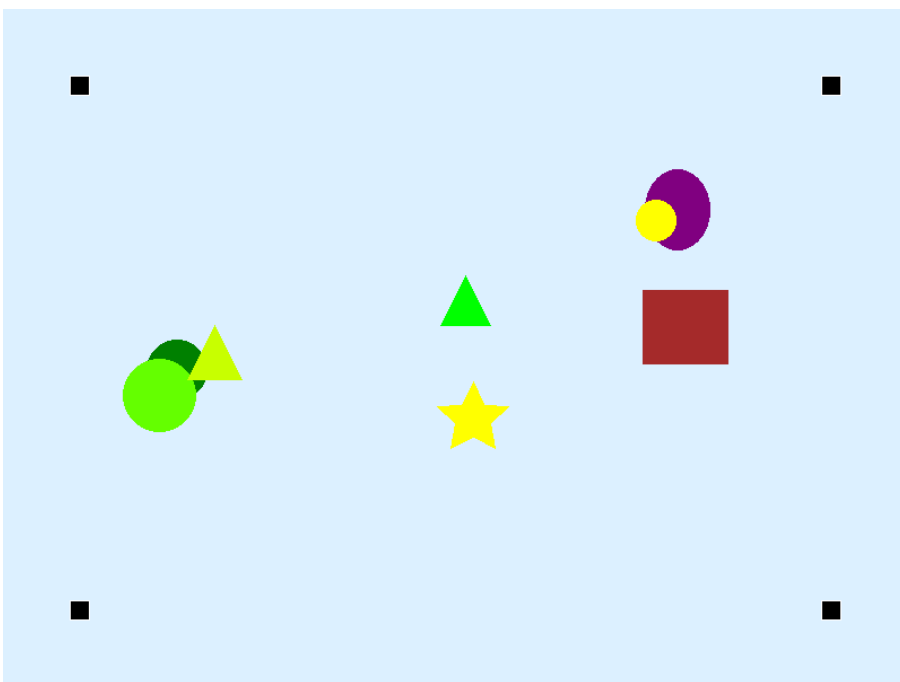
Входные данные

- 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**:

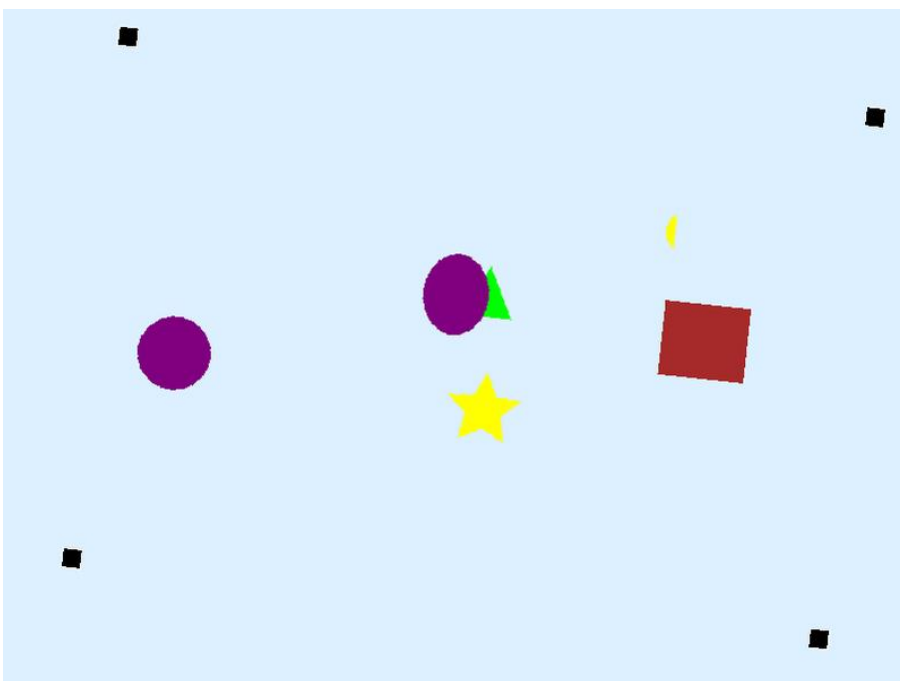
- Размер каждого маркера: **15×15 пикселей**
- Расположение: в углах изображения, на расстоянии **60 пикселей** от краев
- Координаты центров маркеров на первом изображении (без искажений):
 - Левый верхний: (67.5, 67.5)
 - Правый верхний: (732.5, 67.5)
 - Левый нижний: (67.5, 532.5)
 - Правый нижний: (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

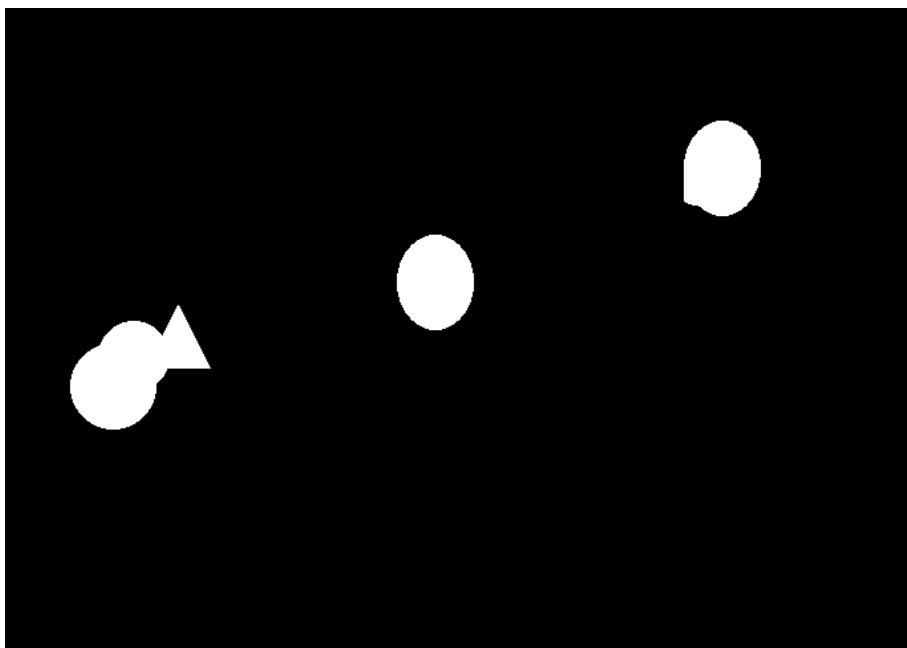
Второе изображение (`image2.png`) может быть подвергнуто следующим искажениям относительно первого:

- 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.

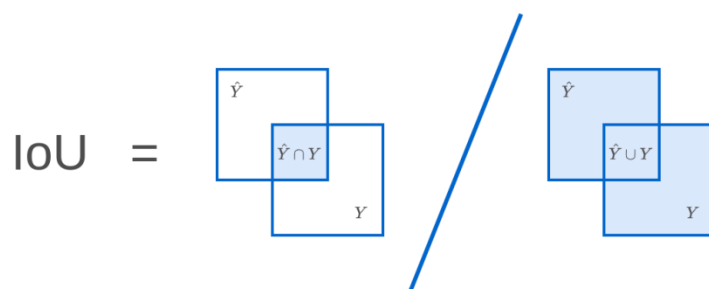
    Алгоритм должен выполнить следующие шаги:
    1. Обнаружить черные маркеры на обоих изображениях
    2. Вычислить геометрическое преобразование для выравнивания image2 к image1
    3. Применить преобразование к image2
    4. Обрезать оба изображения до области, ограниченной маркерами первого
    изображения
       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

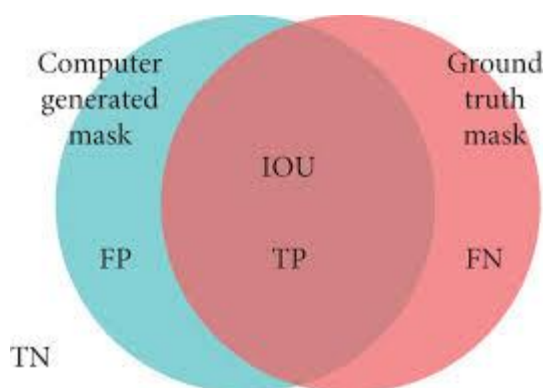
Качество решения оценивается с помощью метрики **IoU (Intersection over Union, Jaccard Index)** — стандартного показателя для задач семантической сегментации.



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:

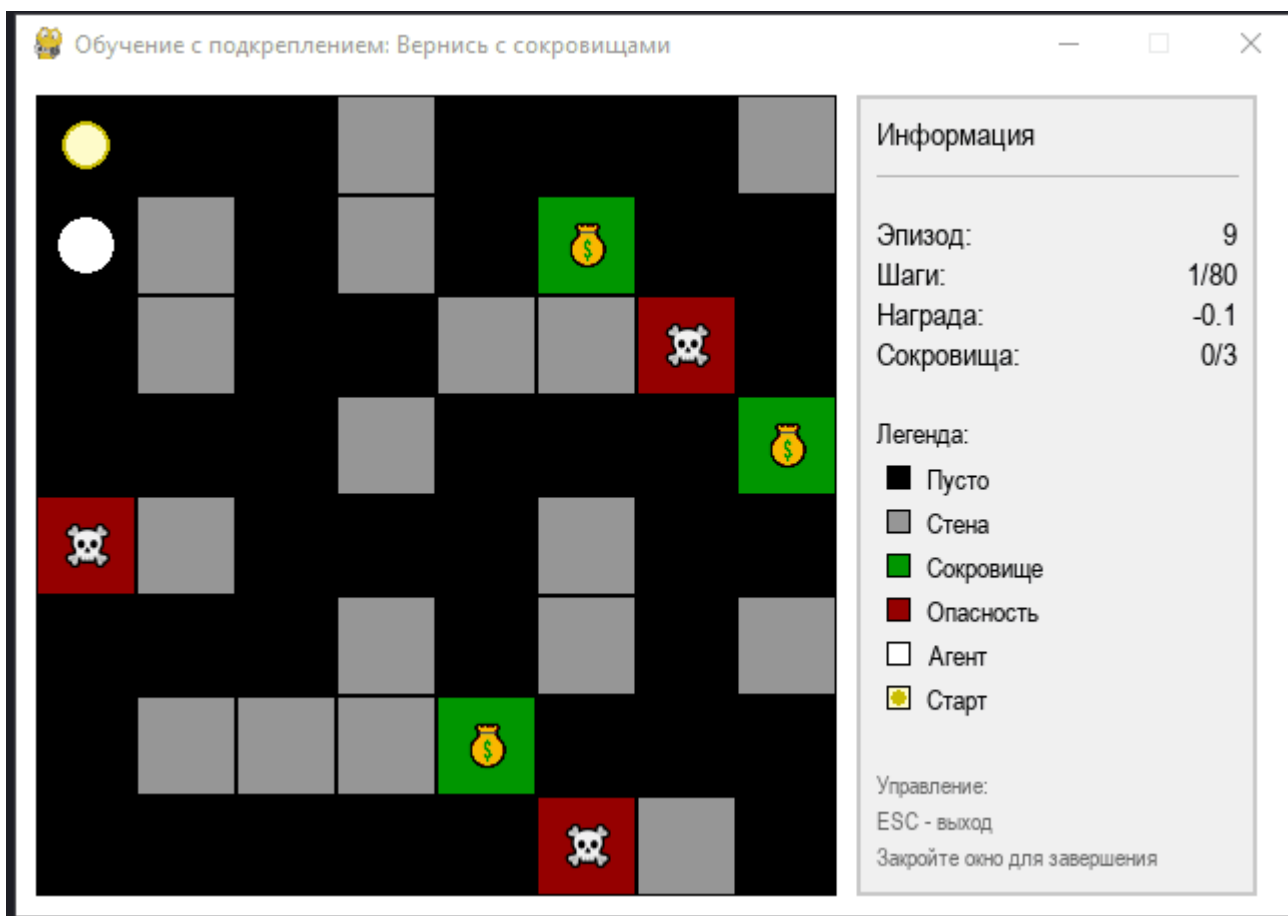
1. **Найти и собрать все 3 сокровища**, разбросанные по лабиринту
2. **Вернуться в стартовую позицию** после сбора всех сокровищ
3. **Избегать опасностей и динамических стен**, которые могут появляться и исчезать
4. **Уложиться в 80 шагов** на выполнение всей миссии

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:

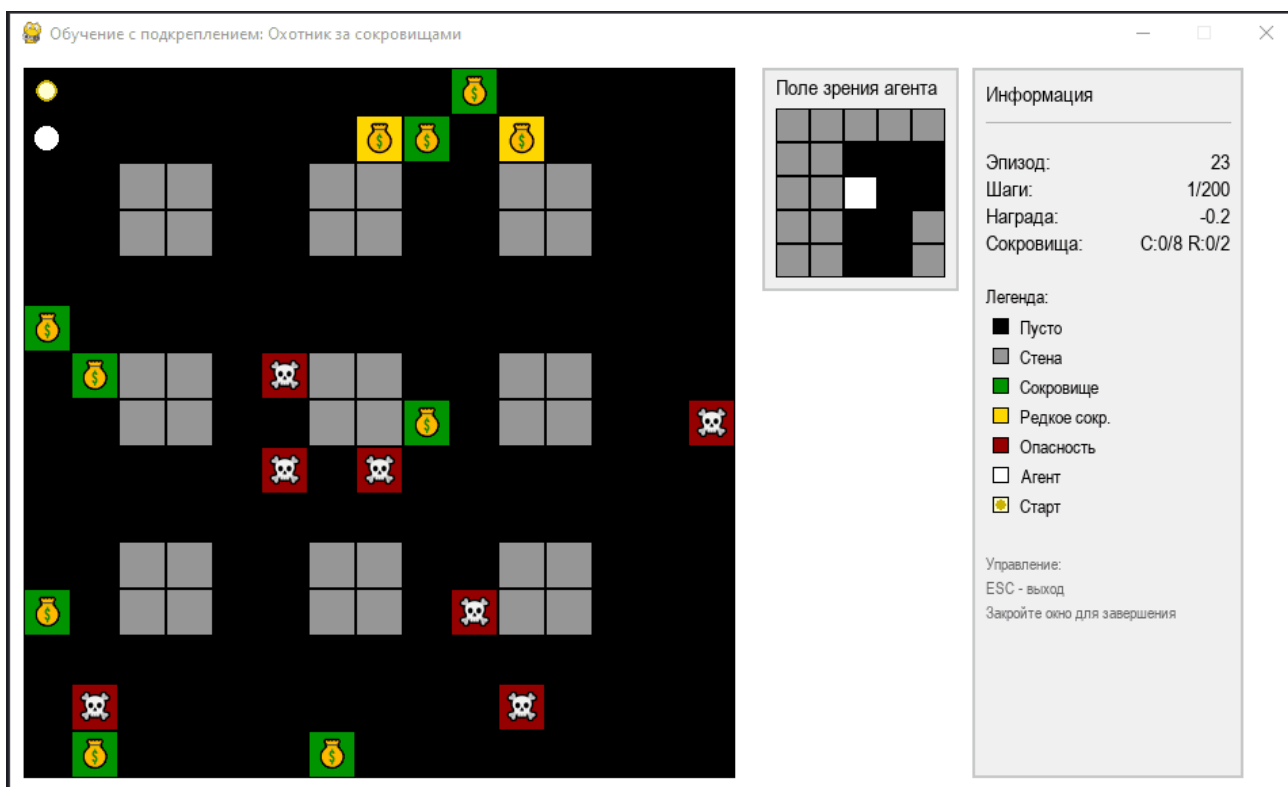
1. **Найти и собрать все 10 сокровищ:** 8 обычных (+10) и 2 редких (+25)
2. **Вернуться на стартовую позицию** после сбора всех сокровищ (+50)
3. **Избегать охотящихся ловушек**, которые активно преследуют агента
4. **Уложиться в 200 шагов** на выполнение всей миссии
5. **Работать с ограниченным полем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