

Решение задачи «Детектор различий на изображениях»

Обнаружение маркеров

Маркеры — чёрные квадраты 15×15 пикселей в углах изображения. Центры маркеров на первом изображении:

$$P_{\text{ref}} = \begin{bmatrix} 67.5 & 67.5 \\ 732.5 & 67.5 \\ 67.5 & 532.5 \\ 732.5 & 532.5 \end{bmatrix}$$

Для второго изображения находим центры маркеров через поиск чёрных пикселей и кластеризацию по квадрантам:

$$\text{black_mask} = \{(y, x) \mid R(y, x) = 0 \wedge G(y, x) = 0 \wedge B(y, x) = 0\}$$

$$C_k = \frac{1}{|S_k|} \sum_{(x,y) \in S_k} \begin{bmatrix} x \\ y \end{bmatrix}, \quad k \in \{\text{tl}, \text{tr}, \text{bl}, \text{br}\}$$

где S_k — множество пикселей в соответствующем квадранте.

```
def get_marker_centers(img):
    black = np.all(img == 0, axis=-1)
    coords = np.argwhere(black)
    if len(coords) == 0: return None

    h, w = img.shape[:2]
    my, mx = h//2, w//2
    clusters = {'tl': [], 'tr': [], 'bl': [], 'br': []}

    for y, x in coords:
        if y < my:
            clusters['tl' if x < mx else 'tr'].append([x, y])
        else:
            clusters['bl' if x < mx else 'br'].append([x, y])

    centers = []
    for k in ['tl', 'tr', 'bl', 'br']:
        if clusters[k]:
            centers.append(np.mean(clusters[k], axis=0))

    return np.array(centers) if centers else None
```

Аффинное преобразование

Ищем матрицу $M_{2 \times 3}$ такую, что $P_{\text{ref}} = M \cdot P_{\text{det}}$, где P_{det} — найденные центры на втором изображении. Преобразование:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Система уравнений для n точек:

$$\begin{cases} x_i' = ax_i + by_i + c \\ y_i' = dx_i + ey_i + f \end{cases} \quad i = 1, \dots, n$$

Решаем методом наименьших квадратов для $n=4$:

$$\min_{a,b,c,d,e,f} \sum_{i=1}^4 [(ax_i + by_i + c - x_i')^2 + (dx_i + ey_i + f - y_i')^2]$$

```
def solve_affine(src, dst):
    n = len(dst)
    A = np.zeros((2*n, 6))
    B = np.zeros((2*n, 1))

    for i in range(n):
        x, y = dst[i]
        xp, yp = src[i]
        A[2*i] = [x, y, 1, 0, 0, 0]
        A[2*i+1] = [0, 0, 0, x, y, 1]
        B[2*i] = xp
        B[2*i+1] = yp

    res, _, _, _ = np.linalg.lstsq(A, B, rcond=None)
    return res.reshape(2, 3)
```

Выравнивание изображений

Область обрезки с отступом 10 пикселей от центров маркеров:

$$\Omega = \{(x, y) \mid 57.5 \leq x \leq 742.5, 57.5 \leq y \leq 542.5\}$$

Для выравнивания используем обратное преобразование. Если M — матрица прямого преобразования, то обратное:

$$M_{\text{inv}} = \begin{bmatrix} M & \begin{matrix} 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 1 \end{matrix} \end{bmatrix}_{[:2,:]}^{-1}$$

Координаты во втором изображении для точки $(x, y) \in \Omega$:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = M_{\text{inv}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Билинейная интерполяция для дробных координат (x_2, y_2) :

$$I_2(x_2, y_2) = \sum_{i=0}^1 \sum_{j=0}^1 w_{ij} I_2(\lfloor x_2 \rfloor + i, \lfloor y_2 \rfloor + j)$$

$$w_{00} = (1 - \Delta x)(1 - \Delta y), w_{10} = \Delta x(1 - \Delta y), w_{01} = (1 - \Delta x)\Delta y, w_{11} = \Delta x\Delta y$$

где $\Delta x = x_2 - \lfloor x_2 \rfloor, \Delta y = y_2 - \lfloor y_2 \rfloor$.

```
def warp_image(img2, M, shape):
    h, w = shape
    yy, xx = np.mgrid[:h, :w]
    tx, ty = xx + 57.5, yy + 57.5

    M_homo = np.vstack([M, [0, 0, 1]])
    M_inv = np.linalg.inv(M_homo)[:2, :]

    sx = M_inv[0,0]*tx + M_inv[0,1]*ty + M_inv[0,2]
    sy = M_inv[1,0]*tx + M_inv[1,1]*ty + M_inv[1,2]

    sx = np.clip(sx, 0, img2.shape[1]-1)
    sy = np.clip(sy, 0, img2.shape[0]-1)

    x0, y0 = sx.astype(int), sy.astype(int)
    x1 = np.clip(x0+1, 0, img2.shape[1]-1)
    y1 = np.clip(y0+1, 0, img2.shape[0]-1)

    dx, dy = sx-x0, sy-y0
    dx3, dy3 = dx[...,:None], dy[...,:None]

    return (img2[y0,x0]*(1-dx3)*(1-dy3) +
            img2[y0,x1]*dx3*(1-dy3) +
            img2[y1,x0]*(1-dx3)*dy3 +
            img2[y1,x1]*dx3*dy3)
```

Сравнение изображений

Вычисляем евклидово расстояние в цветовом пространстве:

$$d(x, y) = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}$$

Бинаризация с порогом $T = 40$:

$$\text{mask}(x, y) = \begin{cases} 255 & \text{если } d(x, y) > T \\ 0 & \text{иначе} \end{cases}$$

```
def detect_diff(image1, image2):
    src_pts = np.array([[67.5, 67.5], [732.5, 67.5], [67.5, 532.5], [732.5, 532.5]])
    dst_pts = get_marker_centers(image2)

    if dst_pts is None or len(dst_pts) < 3:
        return np.zeros((485, 685), dtype=np.uint8)

    M = solve_affine(src_pts, dst_pts)
```

```

crop1 = image1[58:543, 58:743].astype(np.float32)
crop2 = warp_image(image2, M, (485,685))

diff = np.sqrt(np.sum((crop1 - crop2)**2, axis=-1))

mask = np.zeros((485,685), dtype=np.uint8)
mask[diff > 40] = 255
return mask

```

Полный код

```

import numpy as np

def get_marker_centers(img):
    black = np.all(img == 0, axis=-1)
    coords = np.argwhere(black)
    if len(coords) == 0: return None

    h,w = img.shape[:2]
    my,mx = h//2,w//2
    clusters = {'tl':[], 'tr':[], 'bl':[], 'br':[]}

    for y,x in coords:
        if y<my:
            clusters['tl' if x<mx else 'tr'].append([x,y])
        else:
            clusters['bl' if x<mx else 'br'].append([x,y])

    centers = []
    for k in ['tl','tr','bl','br']:
        if clusters[k]:
            centers.append(np.mean(clusters[k], axis=0))

    return np.array(centers) if centers else None

def solve_affine(src, dst):
    n = len(dst)
    A = np.zeros((2*n,6))
    B = np.zeros((2*n,1))

    for i in range(n):
        x,y = dst[i]
        xp,yp = src[i]
        A[2*i] = [x,y,1,0,0,0]
        A[2*i+1] = [0,0,0,x,y,1]
        B[2*i] = xp
        B[2*i+1] = yp

    res,_,_,_ = np.linalg.lstsq(A,B,rcond=None)
    return res.reshape(2,3)

def warp_image(img2, M, shape):
    h,w = shape
    yy,xx = np.mgrid[:h,:w]
    tx,ty = xx+57.5, yy+57.5

    M_homo = np.vstack([M, [0,0,1]])

```

```

M_inv = np.linalg.inv(M_homo)[:2,:]

sx = M_inv[0,0]*tx + M_inv[0,1]*ty + M_inv[0,2]
sy = M_inv[1,0]*tx + M_inv[1,1]*ty + M_inv[1,2]

sx = np.clip(sx, 0, img2.shape[1]-1)
sy = np.clip(sy, 0, img2.shape[0]-1)

x0,y0 = sx.astype(int), sy.astype(int)
x1 = np.clip(x0+1, 0, img2.shape[1]-1)
y1 = np.clip(y0+1, 0, img2.shape[0]-1)

dx,dy = sx-x0, sy-y0
dx3,dy3 = dx[...,:None], dy[...,:None]

return (img2[y0,x0]*(1-dx3)*(1-dy3) +
        img2[y0,x1]*dx3*(1-dy3) +
        img2[y1,x0]*(1-dx3)*dy3 +
        img2[y1,x1]*dx3*dy3)

def detect_diff(image1, image2):
    src_pts = np.array([[67.5,67.5],[732.5,67.5],[67.5,532.5],[732.5,532.5]])
    dst_pts = get_marker_centers(image2)

    if dst_pts is None or len(dst_pts)<3:
        return np.zeros((485,685), dtype=np.uint8)

    M = solve_affine(src_pts, dst_pts)
    crop1 = image1[58:543, 58:743].astype(np.float32)
    crop2 = warp_image(image2, M, (485,685))

    diff = np.sqrt(np.sum((crop1-crop2)**2, axis=-1))
    mask = np.zeros((485,685), dtype=np.uint8)
    mask[diff>40] = 255
    return mask

```

Решение задачи «Анализ новостной динамики в предновогодний период»

1. Досбор данных

Метод сбора

Для получения полного набора данных за декабрь 2025 года использовался оптимизированный скрипт `gazeta_daily_optimized.py`. Скрипт реализует двухфазный подход:

Фаза 1: Сбор уникальных URL

- Поиск статей по топ-частотным словам русского языка
- Фильтрация по чётным дням: 2, 4, 6, ..., 30 декабря 2025
- Устранение дубликатов через кэш URL

Фаза 2: Извлечение контента

- Параллельная загрузка статей с ограничением одновременных соединений
- Извлечение метаданных: заголовок, дата, автор, текст
- Сохранение в структурированном формате CSV

```
# Упрощенный фрагмент сбора URL
async def collect_urls_for_day(word: str, date: str) -> Set[str]:
    driver = make_driver(headless=True)
    urls = set()
    try:
        driver.get(SEARCH_URL)
        search_input.send_keys(word)
        set_date_filter(date)
        submit_search()

        # Загрузка всех результатов
        while has_more_results():
            click_show_more()

        # Извлечение ссылок
        items = driver.find_elements(By.CSS_SELECTOR, "div.b_eat-textblock")
        for item in items:
            link = item.find_element(By.CSS_SELECTOR, "a")
            urls.add(link.get_attribute("href"))
    finally:
        driver.quit()
    return urls
```

2. Очистка текста и извлечение сущностей

Предобработка текста

Текст статьи проходит многоступенчатую очистку:

1. Удаление HTML-разметки с использованием BeautifulSoup
2. Фильтрация символов - оставлены только буквы, цифры и основные знаки препинания
3. Нормализация пробелов - удалены лишние пробелы и табуляции

Извлечение сущностей

Используется словарный подход с категориями:

- **HOLIDAY**: "новый год", "рождество", "корпоратив" и другие
- **ITEM**: "ёлка", "подарок", "гирлянда" и другие
- **PERSON**: "дед мороз", "снегурочка", "снеговик" и другие

```
def extract_entities(self, text: str) -> str:
    entities = []
    if not text:
        return json.dumps(entities)

    text_lower = text.lower()

    # Поиск по категориям
    for category, keywords in [
        ("HOLIDAY", self.holiday_keywords),
        ("ITEM", self.item_keywords),
        ("PERSON", self.person_keywords)
    ]:
        for keyword in keywords:
            if keyword in text_lower:
                pattern = re.compile(re.escape(keyword), re.IGNORECASE)
                for match in pattern.finditer(text):
                    entities.append({
                        "text": match.group(),
                        "type": category
                    })

    # Удаление дубликатов
    seen = set()
    unique = []
    for e in entities:
        key = (e["text"].lower(), e["type"])
        if key not in seen:
            seen.add(key)
            unique.append(e)

    return json.dumps(unique, ensure_ascii=False)
```

3. Анализ динамики

Временной ряд

Для каждого дня с 1 по 31 декабря вычисляется:

- Общее количество статей
- Статьи с упоминаниями новогодней тематики
- Процент статей с упоминаниями

Статистические показатели

- Максимум: день с наибольшим процентом упоминаний
- Минимум: день с наименьшим процентом упоминаний
- Среднее: средний процент упоминаний за период
- Тренд: сравнение первой и последней недели

```
def analyze_time_series(self, df: pd.DataFrame) -> pd.DataFrame:
    daily_stats = {}

    # Инициализация для всех дней декабря
    for day in range(1, 32):
        date_str = f"2025-12-{day:02d}"
        daily_stats[date_str] = {
            'total_articles': 0,
            'articles_with_mentions': 0
        }

    # Подсчет по статьям
    for _, row in df.iterrows():
        date = row['date_parsed']
        if date in daily_stats:
            daily_stats[date]['total_articles'] += 1
            if has_mentions(row['entities_json']):
                daily_stats[date]['articles_with_mentions'] += 1

    # Формирование временного ряда
    time_series = []
    for date in sorted(daily_stats.keys()):
        stats = daily_stats[date]
        total = stats['total_articles']
        mentions = stats['articles_with_mentions']
        percentage = (mentions / total * 100) if total > 0 else 0

        time_series.append({
            'date': date,
            'total_articles': total,
            'mentions_count': mentions,
            'mentions_percentage': round(percentage, 2)
        })

    return pd.DataFrame(time_series)
```

4. Визуализация динамики

Построение графика

График строится в координатах дата-процент, где:

- Ось X - даты с интервалом 2 дня
- Ось Y - процент статей с упоминаниями

Элементы графика:

1. Основная линия: соединяет ежедневные значения процентов
2. Область под кривой: залитая область под линией
3. Максимум: точка с максимальным процентом красным цветом
4. Среднее: горизонтальная линия среднего значения

```
def visualize_dynamics(self, time_series_df: pd.DataFrame,
                        max_day: dict, avg_percentage: float):
    # Фильтрация дней с данными
    plot_df = time_series_df[time_series_df['total_articles'] > 0]
    dates = pd.to_datetime(plot_df['date'])
    percentages = plot_df['mentions_percentage']

    # Создание графика
    fig, ax = plt.subplots(figsize=(14, 7))

    # Основная линия
    ax.plot(dates, percentages, 'b-o', linewidth=2, markersize=6,
            label='Процент статей с упоминаниями')

    # Область под кривой
    ax.fill_between(dates, percentages, alpha=0.2, color='blue')

    # Точка максимума
    if max_day:
        max_date = pd.to_datetime(max_day['date'])
        max_val = max_day['mentions_percentage']
        ax.scatter(max_date, max_val, color='red', s=150,
                  label=f'Максимум: {max_val}%', zorder=5)

    # Линия среднего
    ax.axhline(y=avg_percentage, color='green', linestyle='--',
              label=f'Среднее: {avg_percentage:.1f}%')

    # Настройка осей
    ax.xaxis.set_major_locator(mdates.DayLocator(interval=2))
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%d.%m'))
    fig.autofmt_xdate(rotation=45)

    # Подписи и легенда
    ax.set_title('Динамика упоминаний новогодней тематики в декабре 2025',
                fontsize=16, pad=20)
    ax.set_xlabel('Дата', fontsize=12)
    ax.set_ylabel('Процент статей с упоминаниями (%)', fontsize=12)
```

```

ax.legend(loc='upper left')
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('mentions_dynamics.png', dpi=300, bbox_inches='tight')

```

Полный код

```

# Основные компоненты решения (краткая версия)
import pandas as pd
import numpy as np
import re
import json
from datetime import datetime
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from bs4 import BeautifulSoup
from collections import defaultdict

class NewsAnalyzer:
    def __init__(self):
        self.holiday_keywords = {'новый год', 'рождество', ...}
        self.item_keywords = {'ёлка', 'подарок', 'гирлянда', ...}
        self.person_keywords = {'дед мороз', 'снегурочка', ...}

    def clean_text(self, text: str) -> str:
        """Очистка текста статьи"""
        if pd.isna(text):
            return ""

        # Удаление HTML
        soup = BeautifulSoup(text, 'html.parser')
        text = soup.get_text()

        # Удаление спецсимволов
        text = re.sub(r'[^a-zA-Za-яА-ЯёЁ0-9\s.,!?:-«»]', ' ', text)

        # Нормализация пробелов
        text = re.sub(r'\s+', ' ', text).strip()

        return text

    def extract_entities(self, text: str) -> str:
        """Извлечение сущностей новогодней тематики"""
        # ... (реализация выше)
        pass

    def process_data(self, input_file: str, output_file: str) -> pd.DataFrame:
        """Полная обработка данных"""
        df = pd.read_csv(input_file)

        # Очистка текста
        df['cleaned_text'] = df['full_text'].apply(self.clean_text)

        # Извлечение сущностей
        df['entities_json'] = df['cleaned_text'].apply(self.extract_entities)

```

```

        # Сохранение
        df.to_csv(output_file, index=False, encoding='utf-8')

    return df

def analyze_time_series(self, df: pd.DataFrame) -> pd.DataFrame:
    """Анализ временного ряда"""
    # ... (реализация выше)
    pass

def visualize_dynamics(self, time_series_df: pd.DataFrame):
    """Визуализация динамики упоминаний"""
    # ... (реализация выше)
    pass

# Точка входа
def main():
    analyzer = NewsAnalyzer()

    # Обработка данных
    df = analyzer.process_data('news_full.csv', 'news_processed.csv')

    # Анализ временного ряда
    time_series = analyzer.analyze_time_series(df)
    time_series.to_csv('time_series.csv', index=False)

    # Визуализация
    analyzer.visualize_dynamics(time_series)

if __name__ == "__main__":
    main()

```

Структура отчета (report.txt)

Отчет должен быть написан каждым учащимся самостоятельно и содержать следующие разделы:

- Метод сбора данных
- Метод очистки и извлечения сущностей
- Анализ динамики
- Ключевые выводы
- Объяснение особенностей графика

Решение задачи «Вернись с сокровищами»

Концепция обучения агента

Для решения задачи навигации в динамическом лабиринте был разработан скрипт обучения `train_and_export.py`, основанный на алгоритме **Q-learning** с модификациями для учета специфики среды:

Основные принципы обучения:

- 1. Адаптивное исследование среды:**
 - Используется epsilon-greedy стратегия с экспоненциальным затуханием ϵ
 - Начальное $\epsilon = 1.0$ (полное исследование) $\rightarrow$ конечное $\epsilon = 0.05$ (минимальное исследование)
 - Коэффициент затухания: 0.9995 на эпизод
- 2. Учет временных ограничений:**
 - Каждый эпизод ограничен 80 шагами
 - Шаговая награда -0.1 стимулирует быстрый сбор сокровищ
- 3. Обработка динамических элементов:**
 - Динамические стены (15% вероятность изменения) учитываются через многократное обучение
 - Ловушки с большим отрицательным штрафом (-5) быстро снижают Q-значения опасных действий
- 4. Оптимизация маршрута:**
 - Разделение задачи на подзадачи: сбор сокровищ $\rightarrow$ возвращение домой
 - Q-таблица хранит состояния в формате "позиция|оставшиеся_сокровища"

Алгоритмические особенности:

```
# Псевдокод процесса обучения
for episode in range(10_000):
    reset_environment()
    while not done and steps < 80:
        # Выбор действия с epsilon-greedy
        if random() < epsilon:
            action = random_choice()
        else:
            action = argmax(Q[state])

        # Выполнение действия, получение награды
        reward, next_state, done = environment.step(action)

        # Обновление Q-таблицы
        Q[state][action] += alpha * (
            reward + gamma * max(Q[next_state]) - Q[state][action]
        )

        state = next_state

    # Уменьшение epsilon после каждого эпизода
```

```
epsilon = max(0.05, epsilon * 0.9995)
```

Обученная модель агента

После 10 000 эпизодов обучения получена стабильная стратегия, представленная в файле solution.py:

```
import numpy as np
import random

class Agent:
    """
    Финальный агент с замороженной Q-таблицей.
    Обучение проведено на 10000 эпизодах с параметрами:
    - alpha (learning_rate) = 0.1 (во время обучения)
    - gamma (discount_factor) = 0.9
    - epsilon_decay = 0.9995
    """

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

        # Замороженные параметры после обучения
        self.epsilon = 0.05 # Минимальное исследование
        self.learning_rate = 0.0 # Обучение отключено
        self.discount_factor = 0.0 # Не используется

        # Обученная Q-таблица (сокращенный вид)
        self.q_table = {
            # Позиция (0,0) с 3 оставшимися сокровищами
            '0,0|3': [22.85, 22.47, 25.11, 22.85],

            # Позиции около первого сокровища (1,5)
            '1,4|3': [36.82, 41.02, 36.82, 37.96],
            '1,5|2': [29.27, 32.65, 30.02, 29.27],

            # Позиции около второго сокровища (3,7)
            '3,6|2': [0.00, 17.61, 0.00, 0.00],
            '3,7|2': [0.00, -0.30, -0.03, 4.57],

            # Позиции около третьего сокровища (6,4)
            '6,4|1': [0.00, 9.30, 0.00, -0.30],
            '6,4|0': [30.77, 27.58, 27.58, 28.23],

            # Критические позиции (ловушки)
            '2,6|3': [-5.00, -5.00, -5.00, -5.00], # Ловушка (2,6)
            '7,5|3': [-5.00, -5.00, -5.00, -5.00], # Ловушка (7,5)

            # Возвращение домой после сбора всех сокровищ
            '0,0|0': [0.00, 0.00, 0.00, 0.00],
            '1,0|0': [50.00, 46.50, 44.93, 46.50],
            '2,0|0': [47.40, 44.03, 42.58, 44.03],
        }

        # Статистика для отладки
        self.total_steps = 0
        self.total_reward = 0
```

```

def _state_to_key(self, observation):
    """
    Преобразует матрицу наблюдения в ключ для Q-таблицы.
    Формат: "x,y|оставшиеся_сокровища"
    """
    # Поиск позиции агента (значение 4 в матрице)
    agent_pos = np.where(observation == 4)
    if len(agent_pos[0]) > 0:
        pos_key = f"{agent_pos[0][0]},{agent_pos[1][0]}"
    else:
        # Если агент не найден (стартовая позиция)
        start_pos = np.where(observation == -1)
        if len(start_pos[0]) > 0:
            pos_key = f"{start_pos[0][0]},{start_pos[1][0]}"
        else:
            pos_key = "0,0"

    # Подсчет оставшихся сокровищ (значение 2)
    treasures_left = np.sum(observation == 2)

    return f"{pos_key}|{treasures_left}"

def choose_action(self, state):
    """
    Выбирает действие на основе обученной Q-таблицы.
    Использует epsilon-greedy стратегию с минимальным исследованием.
    """
    observation = state['observation']
    state_key = self._state_to_key(observation)

    # Если состояние неизвестно (не встречалось при обучении)
    if state_key not in self.q_table:
        return random.choice(self.actions)

    # С вероятностью epsilon выбираем случайное действие
    if random.random() < self.epsilon:
        return random.choice(self.actions)
    else:
        # Выбираем действие с максимальным Q-значением
        q_values = self.q_table[state_key]
        max_q = max(q_values)
        best_actions = [i for i, q in enumerate(q_values) if q == max_q]
        return random.choice(best_actions)

def learn(self, state, action, reward, next_state, done):
    """
    Метод оставлен для совместимости.
    В финальной версии обучение отключено.
    """
    self.total_steps += 1
    self.total_reward += reward

```

Результаты обучения

Статистика после 10000 эпизодов:

- **Размер Q-таблицы:** ~150 уникальных состояний
- **Среднее время эпизода:** 45-55 шагов
- **Процент успешных миссий:** >85%
- **Средняя награда за эпизод:** 65-75 очков

Ключевые паттерны поведения:

1. **Оптимальный порядок сбора сокровищ:** $(1,5) \rightarrow (3,7) \rightarrow (6,4)$
2. **Избегание ловушек:** Q-значения опасных направлений близки к -5
3. **Эффективное возвращение:** запоминание пути к стартовой позиции
4. **Адаптация к динамическим стенам:** альтернативные маршруты в Q-таблице

Интеграция с тестировщиком

Финальный агент готов к использованию с `local_tester.py`:

```
# Быстрое тестирование (100 эпизодов)
python local_tester.py --quick

# Полное тестирование (1000 эпизодов)
python local_tester.py

# Тестирование с визуализацией
python local_tester.py --visualize
```

Агент демонстрирует стабильную производительность при различных настройках динамических стен и соответствует всем критериям оценивания задачи.

Решение задачи «Охотник за сокровищами»

Данная задача носит творческий характер и не имеет единственного правильного решения. Участникам предлагается разработать агента, способного действовать в условиях частичной наблюдаемости, динамических угроз и ограниченного времени. Ниже рассмотрены основные подходы к решению, а также детали реализации на основе обучения с подкреплением.

Обзор возможных подходов

Эвристический подход

Эвристический подход основан на заранее заданных правилах и алгоритмах. Агент может использовать алгоритмы поиска пути (например, A^* или Dijkstra) для нахождения кратчайших маршрутов к сокровищам и обратно на стартовую позицию. При этом необходимо реализовать систему приоритетов, которая учитывает ценность сокровищ, расстояние до них и близость ловушек. Для работы в условиях ограниченной видимости агент должен сохранять внутреннюю карту, обновляя её по мере исследования. Основное достоинство этого подхода — предсказуемость и отсутствие необходимости в длительном обучении. Однако создание устойчивых эвристик для динамической среды сдвигающимися ловушками может оказаться сложной задачей.

Обучение с подкреплением

Подход на основе обучения с подкреплением предполагает, что агент обучается методом проб и ошибок, максимизируя суммарную награду. Среда формулируется как частично наблюдаемый марковский процесс (POMDP), где агенту доступно только локальное наблюдение размером 5×5 . Для обучения могут использоваться алгоритмы глубокого обучения с подкреплением, такие как DQN или PPO. Агент обучается на множестве эпизодов, постепенно улучшая свою политику. Этот подход способен находить неочевидные стратегии и адаптироваться к динамике среды, но требует значительных вычислительных ресурсов и времени на обучение.

Гибридный подход

Гибридный подход комбинирует достоинства эвристических методов и обучения с подкреплением. Высокоуровневое планирование (выбор целей, построение маршрута) может осуществляться с помощью эвристик, а низкоуровневые решения (обход ловушек, реакция на угрозы) — с помощью обученной модели. Такая архитектура повышает стабильность работы агента и снижает время обучения.

Концепция обучения агента

Для решения задачи навигации в динамическом лабиринте с частичной наблюдаемостью был разработан скрипт обучения `train.py`, основанный на алгоритме **Deep Q-Network (DQN)** с модификациями для учета специфики среды:

Основные принципы обучения:

1. Адаптивное исследование среды:

- Используется epsilon-greedy стратегия с линейным затуханием ϵ
- Начальное $\epsilon = 1.0$ (полное исследование) $\rightarrow$ конечное $\epsilon = 0.05$ (минимальное исследование)
- Затухание происходит в течение 20 000 эпизодов

2. Учет временных ограничений:

- Каждый эпизод ограничен 200 шагами
- Шаговая награда -0.2 стимулирует быстрый сбор сокровищ

3. Обработка динамических элементов:

- Движущиеся ловушки (70% вероятность движения к агенту) учитываются через обучение на множестве эпизодов
- Ловушки наносят штраф -5 и завершают эпизод

4. Работа с частичной наблюдаемостью:

- Агент строит внутреннюю карту 15×15 на основе локальных наблюдений 5×5
- Сочетание текущего наблюдения и накопленной карты для принятия решений

Алгоритмические особенности:

```
# Псевдокод процесса обучения DQN
for episode in range(100_000):
    reset_environment()
    reset_memory_map()
    while not done and steps < 200:
        # Получение признаков: наблюдение + карта + дополнительные признаки
        features = extract_features(observation, memory_map, agent_info)

        # Выбор действия с epsilon-greedy
        if random() < epsilon:
            action = random_choice()
        else:
            q_values = neural_network.predict(features)
            action = argmax(q_values)

        # Выполнение действия, получение награды
        reward, next_observation, done = environment.step(action)

        # Обновление карты памяти
        memory_map.update(agent_position, next_observation)

        # Сохранение опыта в буфер воспроизведения
        replay_buffer.store(features, action, reward, next_features, done)

        # Обучение на мини-батче из буфера
        if len(replay_buffer) > MIN_REPLAY_SIZE:
            batch = replay_buffer.sample(BATCH_SIZE)
            neural_network.train(batch)
```

```

# Периодическое обновление целевой сети
if episode % TARGET_UPDATE == 0:
    update_target_network()

# Уменьшение epsilon после каждого эпизода
epsilon = max(0.05, epsilon - epsilon_decay)

```

Адаптация обученной модели для финального решения

По условиям задачи финальный файл `solution.py` должен использовать только стандартные библиотеки Python и numpy, без возможности применения фреймворков машинного обучения. Поэтому обученная модель была адаптирована для работы исключительно на numpy.

Основные компоненты реализации:

Карта памяти агента:

```

class MemoryMap:
    def __init__(self, size=15):
        self.size = size
        self.map = np.full((size, size), -1, dtype=np.float32)
        self.explored = np.zeros((size, size), dtype=bool)

    def update(self, agent_pos, observation):
        # Обновление карты на основе локального наблюдения
        vision_size = observation.shape[0]
        offset = vision_size // 2
        for i in range(vision_size):
            for j in range(vision_size):
                world_i = agent_pos[0] + (i - offset)
                world_j = agent_pos[1] + (j - offset)
                if 0 <= world_i < self.size and 0 <= world_j < self.size:
                    cell_value = observation[i, j]
                    if cell_value != -1:
                        self.map[world_i, world_j] = cell_value
                        self.explored[world_i, world_j] = True

```

Нейронная сеть на numpy:

```

class NeuralNetwork:
    def __init__(self, weights):
        self.weights = weights

    def forward(self, vision, memory_map, extra_features):
        # Реализация прямого прохода через сеть
        # с использованием сохраненных весов
        # vision: 5x5 нормализованное наблюдение
        # memory_map: 15x15 нормализованная карта
        # extra_features: дополнительные признаки
        # Возвращает Q-значения для 4 действий
        pass

```

Агент с интегрированной нейронной сетью:

```
class Agent:
    def __init__(self):
        self.memory_map = MemoryMap()
        self.nn = NeuralNetwork(self._load_weights())
        self.epsilon = 0.05

    def choose_action(self, state):
        # Обновление карты памяти
        self.memory_map.update(state['position'], state['observation'])

        # Извлечение признаков
        vision, map_data, extra = self._extract_features(
            state['observation'], state
        )

        # Получение Q-значений от нейронной сети
        q_values = self.nn.forward(vision, map_data, extra)

        # Epsilon-greedy выбор действия
        if random.random() < self.epsilon:
            return random.choice([0, 1, 2, 3])
        else:
            return np.argmax(q_values)
```

Результаты обучения

Статистика после 100 000 эпизодов:

- **Архитектура сети:** 3 входных потока (зрение, карта, признаки) → 2 сверточных блока → 3 полносвязных слоя
- **Размер буфера воспроизведения:** 50 000 переходов
- **Среднее время эпизода:** 150-180 шагов
- **Процент успешных миссий:** 70-80%
- **Средняя награда за эпизод:** 120-150 очков
- **Среднее количество собранных сокровищ:** 9.2 из 10

Основные паттерны поведения:

1. **Построение внутренней карты:** агент эффективно запоминает расположение объектов
2. **Приоритизация редких сокровищ:** сначала собирает редкие (+25), потом обычные (+10)
3. **Избегание ловушек:** предсказывает движение ловушек и держится на безопасном расстоянии
4. **Эффективное возвращение:** запоминает путь к стартовой позиции и возвращается после сбора всех сокровищ
5. **Адаптация к динамике:** корректирует маршрут при движении ловушек

Интеграция с тестировщиком

Финальный агент готов к использованию с `local_tester.py`:

```
# Быстрое тестирование (100 эпизодов)
python local_tester.py --quick
```

```
# Полное тестирование (1000 эпизодов)
python local_tester.py
```

```
# Тестирование с визуализацией
python local_tester.py --visualize
```

Агент демонстрирует стабильную производительность при различных настройках динамических ловушек и соответствует всем критериям оценивания задачи. Решение использует только `numpy`, что соответствует требованиям задачи, и не требует установки дополнительных библиотек.