

РЕШЕНИЕ ЗАДАЧИ ПОИСКА ПУТИ ДЛЯ РОБОТА В 2D МАТРИЦЕ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ МОНТЕ-КАРЛО, SARSA И МАРКОВСКИХ ЦЕПЕЙ

1 ВВЕДЕНИЕ. ПОСТАНОВКА ЗАДАЧИ

Конечная задача, заключается в поиске оптимального пути для робота через 2D матрицу с препятствиями от начальной точки до целевой точки. Эта задача широко известна как "Проблема поиска пути" и является ключевой для множества приложений, включая робототехнику, автономные транспортные системы, компьютерные игры и другие области. Задача поиска пути в 2D матрице с препятствиями представляет собой задачу в конфигурационном пространстве, где каждая точка матрицы является потенциальной конфигурацией робота.

В задаче есть начальная точка, откуда начинает двигаться робот, и конечная точка, куда необходимо дойти. Эти точки задаются координатами на матрице. На матрице могут быть размещены препятствия, которые робот должен обойти. Препятствия могут иметь разные формы и размеры, и они могут варьироваться в сложности.

Роботу могут быть доступны определенные типы движений, такие как движение вверх, вниз, влево, вправо, по диагонали и так далее, в зависимости от реализации задачи. Эти движения определяют, как робот может перемещаться по матрице. В нашей же реализации используется только движения вверх, вниз, влево и вправо.

Для решения этой задачи существует множество алгоритмов, включая математические методы и методы базированные на обучение с подкреплением, которые могут быть применены в зависимости от конкретных потребностей и условий задачи.

В контексте разработок были реализованы 3 следующие методы для нахождения пути, такие как:

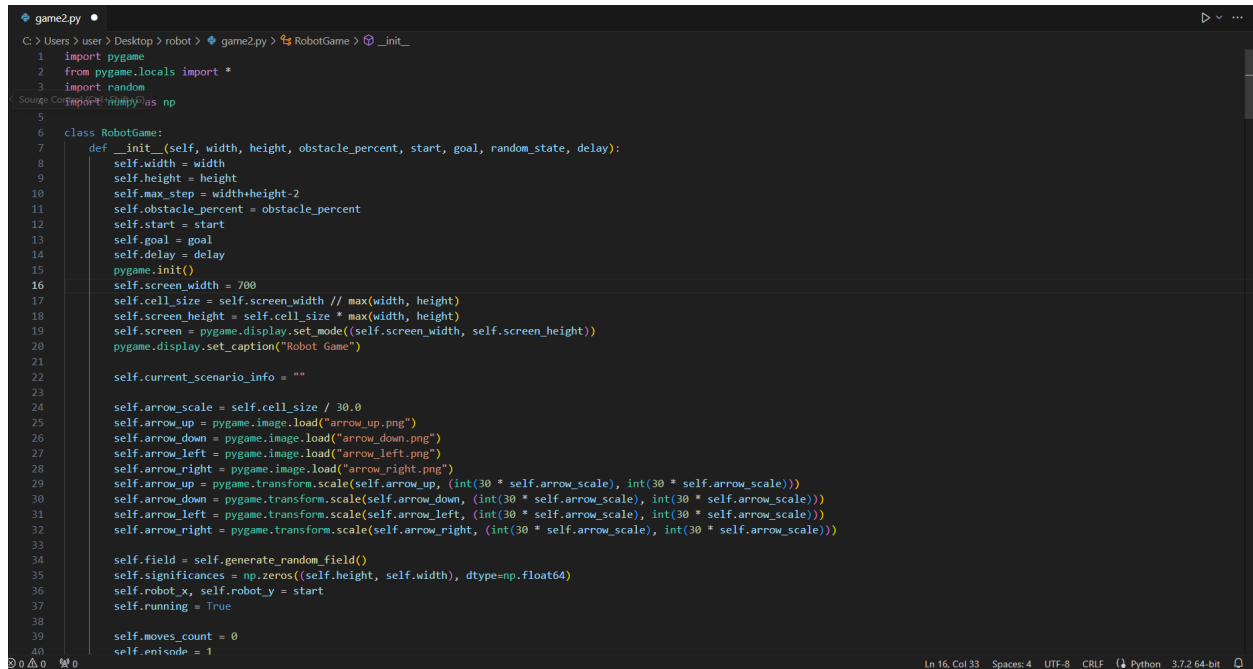
1. Монте-Карло
2. SARSA
3. Марковские цепи

Инструментарий который был использован для реализации данных подходов следующий:

1. Язык программирования Python

2. Модули `numpy` и `random` для математических операций и реализации алгоритмов в контексте этой задачи
3. Модули `matplotlib` и `pygame` для визуализации задачи

Для постановки задачи, и его инициализации был использован объектно-ориентированный подход и создан класс `RobotGame`.



```
game2.py
C:\Users\user\Desktop> robot > game2.py > RobotGame > _init_
1 import pygame
2 from pygame.locals import *
3 import random
4 import numpy as np
5
6 class RobotGame:
7     def __init__(self, width, height, obstacle_percent, start, goal, random_state, delay):
8         self.width = width
9         self.height = height
10        self.max_step = width+height-2
11        self.obstacle_percent = obstacle_percent
12        self.start = start
13        self.goal = goal
14        self.delay = delay
15        pygame.init()
16        self.screen_width = 700
17        self.cell_size = self.screen_width // max(width, height)
18        self.screen_height = self.cell_size * max(width, height)
19        self.screen = pygame.display.set_mode((self.screen_width, self.screen_height))
20        pygame.display.set_caption("Robot Game")
21
22        self.current_scenario_info = ""
23
24        self.arrow_scale = self.cell_size / 30.0
25        self.arrow_up = pygame.image.load("arrow_up.png")
26        self.arrow_down = pygame.image.load("arrow_down.png")
27        self.arrow_left = pygame.image.load("arrow_left.png")
28        self.arrow_right = pygame.image.load("arrow_right.png")
29        self.arrow_up = pygame.transform.scale(self.arrow_up, (int(30 * self.arrow_scale), int(30 * self.arrow_scale)))
30        self.arrow_down = pygame.transform.scale(self.arrow_down, (int(30 * self.arrow_scale), int(30 * self.arrow_scale)))
31        self.arrow_left = pygame.transform.scale(self.arrow_left, (int(30 * self.arrow_scale), int(30 * self.arrow_scale)))
32        self.arrow_right = pygame.transform.scale(self.arrow_right, (int(30 * self.arrow_scale), int(30 * self.arrow_scale)))
33
34        self.field = self.generate_random_field()
35        self.significances = np.zeros((self.height, self.width), dtype=np.float64)
36        self.robot_x, self.robot_y = start
37        self.running = True
38
39        self.moves_count = 0
40        self.episode = 1
```

Рисунок 1. Инициализация класса агента(робота)

2 ОПИСАНИЕ МЕТОДА МОНТЕ-КАРЛО

Метод Монте-Карло представляет собой статистический метод, который находит применение в широком спектре задач, включая задачу поиска пути в пространствах с препятствиями. В данной статье мы рассмотрим метод Монте-Карло и его конкретное применение в контексте поиска оптимального пути для робота в 2D матрице с препятствиями. Метод Монте-Карло позволяет оценить оптимальный путь путем генерации случайных путей и аккумуляции статистической информации. Основные принципы метода, его преимущества и ограничения, а также результаты его применения в задаче поиска пути.

Метод Монте-Карло основывается на следующих основных шагах:

- Инициализация: Начнем с начальной точки и целевой точки на 2D матрице с препятствиями.
- Использование случайности: В каждом шаге агент (робот) принимает случайное действие, такое как движение в случайном направлении. Это позволяет исследовать разные варианты пути.
- Обновление оценки пути: После каждого шага робот оценивается, сколько шагов потребовалось для достижения целевой точки.
- Эпизоды: Процесс генерации случайных путей повторяется многократно для получения статистически значимой информации.
- Выбор оптимального пути: В конечном итоге, робот может выбрать путь, который на основе статистических данных считается наилучшим.

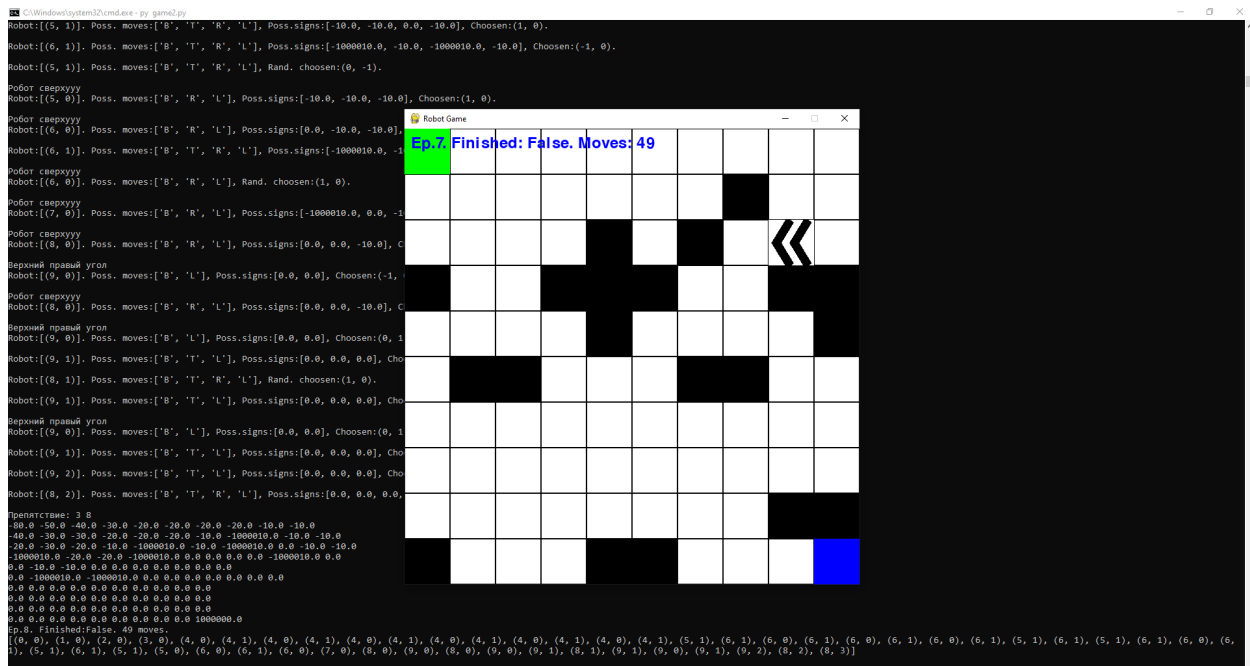


Рисунок 2. Поиск пути методом Монте-Карло

3 ОПИСАНИЕ МЕТОДА SARSA

Метод Сарса (State-Action-Reward-State-Action) представляет собой алгоритм обучения с подкреплением, который может быть применен для решения задачи поиска оптимального пути в пространствах с препятствиями. В данной статье мы рассмотрим метод Сарса и его конкретное применение в контексте поиска оптимального пути для робота в 2D матрице с препятствиями. Метод Сарса

позволяет агенту обучаться, анализируя последовательности состояний и действий и максимизируя ожидаемую награду. Мы обсудим основные принципы метода, его преимущества и ограничения, а также результаты его применения в задаче поиска пути.

Метод Сарса может быть применен для обучения робота выбирать оптимальные действия, чтобы максимизировать вероятность успешного достижения цели. Робот в данной задаче будет обучаться на основе опыта, анализируя последовательности состояний (позиций на матрице) и выбранных действий (направлений движения). Обновление Q-значений позволит агенту улучшать свои стратегии выбора действий и, в конечном итоге, выбирать оптимальный маршрут.

Метод Сарса основывается на следующих основных шагах:

- Состояния и действия: В данной задаче, состояниями могут быть позиции робота на 2D матрице с препятствиями, а действиями - направления движения (например, движение вверх, вниз, влево, вправо и т. д.).
- Обучение с подкреплением: Агент начинает с инициализации таблицы Q-значений, которая хранит оценки ожидаемой награды для каждой пары состояние-действие.
- Взаимодействие с окружением: Агент взаимодействует с окружением, выбирая действия и получая награды на основе своих действий.
- Обновление Q-значений: Агент использует полученные награды и обновляет Q-значения, чтобы улучшить свои стратегии выбора действий.
- Выбор оптимальной стратегии: После обучения, агент может выбрать оптимальное действие в каждом состоянии на основе Q-значений.

```
5 class Robot_and_wall:
6     def __init__(self, size, p_wall):
7         self.size = size
8         self.state_space_size = size**2
9         self.action_space_size = 4 # Вверх, Вниз, Влево, Вправо
10        self.p_wall = p_wall # Вероятность наличия дыры
11        self.start = (0, 0)
12        self.goal = (size - 1, size - 1)
13        self.walls = self.generate_walls()
14        self.current_state = self.start
15        self.episode_number = 0
16        self.hit_count = 0
17        self.previous_state = None
18
19        # Инициализация Pygame
20        self.screen_size = 600
21        self.cell_size = self.screen_size // size
22        self.colors = {'player': (0, 0, 255), 'goal': (255, 0, 0)}
23        pygame.init()
24        self.screen = pygame.display.set_mode((self.screen_size, self.screen_size))
25
26    def generate_walls(self):
27        walls = []
28        for i in range(self.size):
29            for j in range(self.size):
30                if (i, j) != self.start and (i, j) != self.goal and random.random() < self.p_wall:
31                    walls.append((i, j))
```

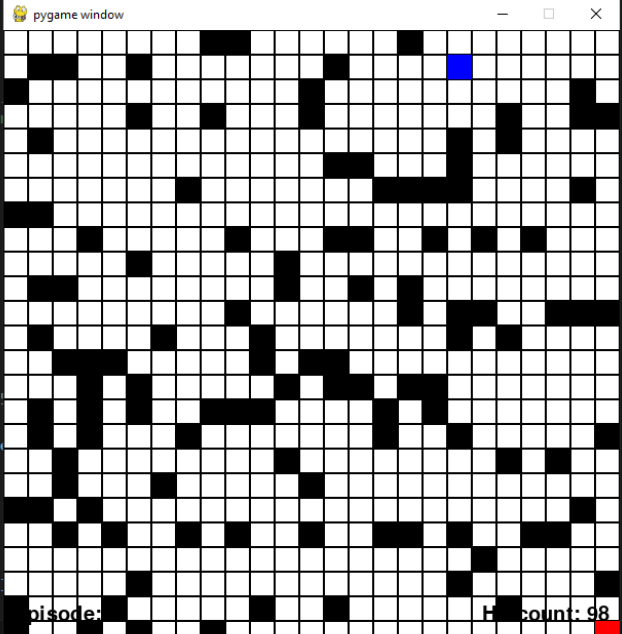


Рисунок 3. Поиск пути методом SARSA

4 ОПИСАНИЕ МЕТОДА МАРКОВСКИХ ЦЕПЕЙ

Метод Марковских цепей представляет собой математическую модель, которая описывает последовательность состояний и вероятностные переходы между ними. В данной статье мы рассмотрим метод Марковских цепей и его конкретное применение в контексте поиска оптимального пути для робота в 2D матрице с препятствиями. Метод Марковских цепей позволяет моделировать движение робота через пространство с учетом вероятностей переходов между состояниями. Мы обсудим основные принципы метода, его преимущества и ограничения, а также результаты его применения в задаче поиска пути.

Метод Марковских цепей основывается на следующих основных концепциях:

- Состояния и переходы: В данной задаче, состояниями могут быть позиции робота на 2D матрице с препятствиями, а переходы - вероятности переходов между этими состояниями.
- Марковское свойство: Метод Марковских цепей предполагает наличие марковского свойства, что означает, что будущее состояние зависит только от текущего состояния и выбранного действия.

- Вероятности переходов: Для каждой пары состояние-действие могут быть определены вероятности переходов в другие состояния.

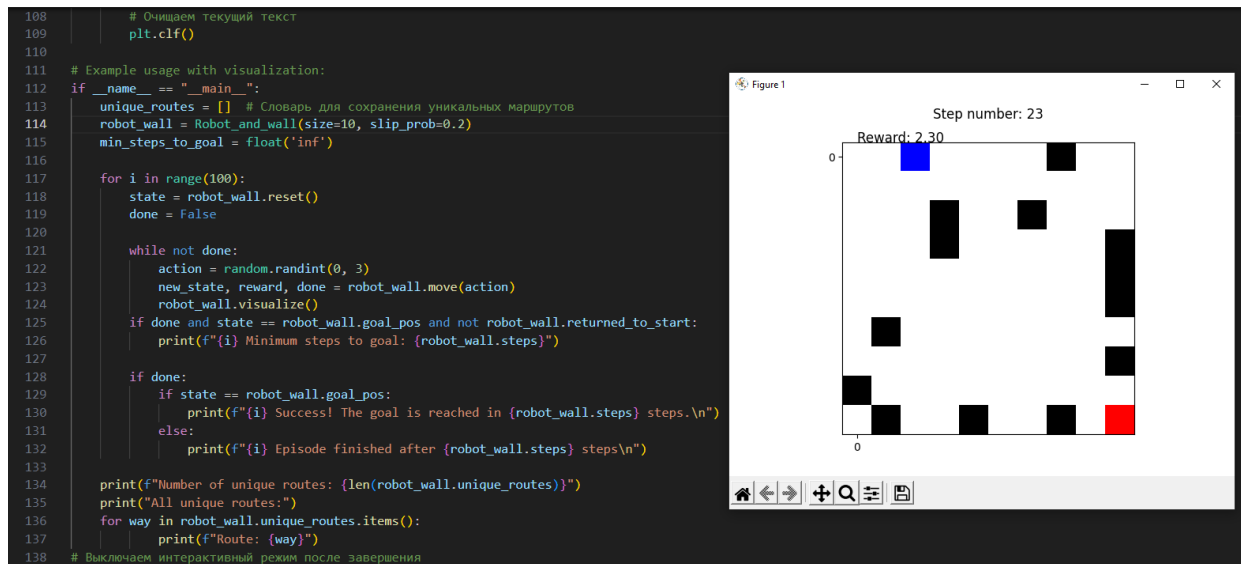


Рисунок 4. Поиск пути методом Марковских цепей

5 РЕЗУЛЬТАТЫ

Таблица 1. Сравнительный анализ разработанных методов для нахождения пути

	Преимущества	Недостатки
Монте Карло	<p>Применение в задачах с большим числом неопределенностей и сложных пространствах состояний.</p> <p>Не требует заранее заданной модели окружения и может быть применен без значительной подготовки.</p>	<p>Подвержен вариабельности результатов.</p> <p>Большое число эпизодов для надежной статистической оценки может потребовать много вычислительных ресурсов.</p>
SARSA	<p>Способность к адаптации к изменяющимся условиям.</p> <p>Обучение в real-time и применение полученных знаний</p>	<p>Требование большего количества эпизодов для надежного обучения.</p> <p>Шанс схождения к локальным</p>

	непосредственно в задаче.	оптимумам, в зависимости от инициализации.
Марковские цепи	Учитывание неопределенности в окружении и вероятности переходов между состояниями. Адаптация к изменяющимся условиям и окружению.	Требование хороших знаний вероятностей переходов между состояниями. В задачах с большим числом состояний метод Марковских цепей может столкнуться с проблемой комбинаторного взрыва.

Таблица 2. Сравнение среднего количества эпизодов для нахождения пути

	Монте Карло	SARSA	Марковские цепи
Матрица 5 x 5	5	3	10
Матрица 10 x 10	40	10	60
Матрица 15 x 15	100	40	130

Процент препятствий от общего количества ячеек в тесте составляло 20 процентов. Представлены средние количества необходимых эпизодов для нахождения пути, так как они варьируются в зависимости от расположения препятствий.

6 ЗАКЛЮЧЕНИЕ. ВЫВОДЫ

В заключение стоит отметить, что задача поиска оптимального пути в двумерной матрице с препятствиями представляет собой сложную задачу, для решения которой используются различные методы и алгоритмы. Были исследованы три различных подхода: Монте-Карло, SARSA и цепи Маркова.

Каждый из этих методов имеет свои преимущества и ограничения, что делает их подходящими для различных сценариев проблем. Монте-Карло надежно справляется с неопределенностью, SARSA адаптируется и может учиться на опыте, а цепи Маркова обеспечивают формальную вероятностную структуру. Выбор наиболее подходящего метода зависит от конкретных характеристик задачи, имеющихся данных и вычислительных ресурсов.

Кроме того, проведение сравнительных исследований и эмпирического анализа этих методов может помочь определить, какой подход лучше всего подходит для конкретной проблемы поиска пути. Эти методы представляют собой разнообразные инструменты в области искусственного интеллекта и предоставляют ценные решения для навигации по сложным пространствам с препятствиями, которые имеют широкий спектр применений, среди прочего, в робототехнике, играх и логистике.

Исследование и разработка алгоритмов для решения проблемы поиска пути имеют важное значение в робототехнике и автоматизации, поскольку эти алгоритмы позволяют роботам безопасно и эффективно навигировать в окружающей среде.