

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Физико-технологический институт

Кафедра «Технической физики»

Оценка

Преподаватель

Кашин И.В.

**МНОГОПОТОЧНАЯ РЕАЛИЗАЦИЯ ПОИСКА   
ВЫХОДА ИЗ ЛАБИРИНТА**

ОТЧЕТ

по лабораторной работе №7

|  |  |  |
| --- | --- | --- |
| Студент | Маркова Алёна Денисовна | ФИО студента |

|  |
| --- |
| Специальность (направление подготовки) |
| 09.03.02 Информационные системы и технологии | |

|  |  |
| --- | --- |
| Группа | Фт-420008 |

Екатеринбург

2025

**ОГЛАВЛЕНИЕ**

[ОПИСАНИЕ ЗАДАЧИ 3](#_Toc212600700)

[ПРИНЦИП ПАРАЛЛЕЛИЗАЦИИ 6](#_Toc212600701)

[РЕШЕНИЕ ЗАДАЧИ 8](#_Toc212600702)

[ОЦЕНКА ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛИЗАЦИИ 10](#_Toc212600703)

[ПРИЛОЖЕНИЕ А – ЛИСТИНГ КОДА 12](#_Toc212600704)

ОПИСАНИЕ ЗАДАЧИ

В данной задаче нам предстояло изучить один из подходов к работе с с использованием эвристических алгоритмов поиска, основанных на динамике агентов в пространстве.

Ключевая задача состояла в анализе того, как количество вычислительных процессов влияет на производительность данного алгоритма.

Эвристические методы представляют собой способы решения сложных оптимизационных задач, где перебрать все варианты невозможно из-за их огромного количества. Суть подхода заключается в том, что решения принимаются на основе локальной информации, а итоговый результат формируется из множества таких локальных решений, приводя к результату, близкому к оптимальному.

Центральная идея алгоритма заключается в том, что независимые величины становятся взаимосвязанными через процесс нормирования (преобразование в проценты или вероятности).

Иллюстративный пример:

Возьмем набор из четырех чисел: 2, 3, 6, 8. Изначально они независимы – изменение одного никак не затрагивает остальные.

Однако при преобразовании их в доли от общей суммы:

Общая сумма: ;

Получаем доли: .

Теперь изменение любого значения влияет на все остальные через изменение общей суммы в знаменателе. Этот эффект взаимного влияния и используется в алгоритме.

Алгоритм реализован через систему множественных агентов.

Агент – это вычислительная сущность со следующими свойствами:

* Находится в определенной позиции на двумерной сетке (координаты x, y);
* Способен перемещаться в одном из 8 направлений (по горизонтали, вертикали и диагоналям);
* Выбирает направление движения исходя из информации в соседних клетках;
* При посещении клетки увеличивает ее значение в динамическом распределении на единицу.

Хотя каждый агент действует самостоятельно, их совместная деятельность через общее пространство создает сложное коллективное поведение, формирующее требуемое распределение (эмергентное поведение).

Программа реализует мультиагентный подход к поиску пути в лабиринте. Лабиринт загружается из чёрно-белого изображения, где белые пиксели представляют собой проходимые клетки (значение 1), а чёрные - стены (значение 0). Вход располагается в левой части лабиринта, выход - в правой части. Каждый агент может двигаться в одну из восьми соседних клеток (включая диагональные направления), если целевая клетка является проходимой.

Для навигации агентов используется специальная метрика α - градиент направленности к выходу. Эта метрика вычисляется с помощью алгоритма поиска в ширину, начиная от точки выхода. Значение метрики отражает удалённость от цели: чем ближе клетка расположена к выходу, тем выше её значение α. При движении агент выбирает такое направление, которое максимизирует значение α, тем самым приближаясь к выходу лабиринта.

Основой алгоритма является механизм принятия решения о направлении движения. Процесс перемещения агента включает следующие этапы:

1. Оценивает соседние позиции (до 8 направлений)

2. Вычисляет коэффициент K для каждой пробной позиции ():

*где n(x, y) – количество посещений точки,   
 – степень затухания (например 0.1)*

3. Из проходимых направлений выбирается то, которое имеет наибольшее значение K

4. Агент перемещается в выбранную клетку

5. Обновляет динамическое распределение:

K близкое к 0 (недобор): В данной точке динамическое распределение меньше эталонного – агент стремится туда переместиться

K >> 0 (перебор): В данной точке динамическое распределение больше эталонного → агент избегает этой точки

Несмотря на то, что каждый агент использует только локальную информацию (значения K в непосредственно соседних клетках), коллективное поведение множества агентов приводит к успешному решению глобальной задачи - нахождению пути через весь лабиринт.

Механизмы успешной навигации:

* Градиентный спуск: следование по возрастанию метрики α обеспечивает движение в правильном направлении
* Случайность в выборе: при равных значениях случайный выбор помогает избежать зацикливания
* Множественность попыток: большое количество агентов повышает вероятность того, что хотя бы один найдёт путь к выходу

ПРИНЦИП ПАРАЛЛЕЛИЗАЦИИ

Параллелизация реализована с использованием модуля multiprocessing библиотеки Python. Основная идея заключается в разделении агентов на батчи и параллельной обработке каждого батча в отдельном процессе.

Ключевые элементы реализации:

* Использование RawArray (ctypes) для создания разделяемой памяти между процессами
* Лабиринт (maze), метрика K, позиции агентов (agents\_x, agents\_y) и флаги завершения (agents\_finished) хранятся в shared memory
* Пул процессов (Pool) создаётся один раз в конструкторе класса и переиспользуется на всех итерациях алгоритма
* Агенты делятся на батчи приблизительно равного размера пропорционально количеству процессов
* Каждый процесс независимо вычисляет новые позиции для своего батча агентов
* Случайные порядки проверки направлений также хранятся в shared memory

Такой подход минимизирует накладные расходы на создание процессов и передачу данных, поскольку данные не копируются между процессами, а используются совместно через разделяемую память.

Архитектура параллельной обработки:

1. Главный процесс загружает изображение лабиринта и преобразует его в бинарную матрицу
2. Вычисляется метрика K с помощью обхода в ширину
3. Создаются буферы в разделяемой памяти (shared memory) для всех данных
4. Инициализируется пул из N процессов-воркеров
5. На каждом шаге симуляции агенты делятся на батчи и распределяются по процессам
6. Каждый процесс параллельно обрабатывает свой батч агентов
7. Главный процесс собирает результаты и обновляет статистику
8. После завершения эксперимента пул процессов корректно закрывается

РЕШЕНИЕ ЗАДАЧИ

Исходный лабиринт представлен на рисунке 1. Он сгенерирован с помощью специализированного интернет-ресурса.

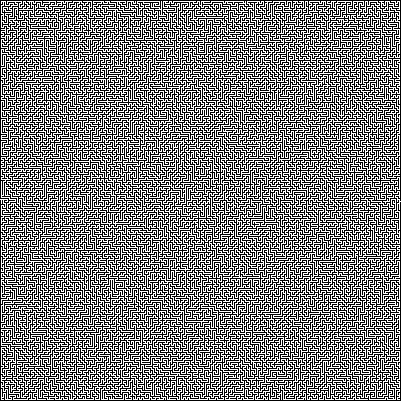


Рисунок 1 – Лабиринт

Сначала программа загружает изображение лабиринта в формате PNG. Изображение конвертируется в оттенки серого, после чего происходит бинаризация: пиксели со значением яркости выше 128 считаются проходимыми участками и получают значение 1, остальные становятся стенами со значением 0. В результате получается двумерный массив, представляющий структуру лабиринта.

Программа автоматически ищет вход и выход. Вход ищется в левом столбце лабиринта — это первая сверху проходимая клетка. Выход находится аналогично в правом столбце. Если на границах нет проходов, программа ищет ближайшие проходимые клетки к соответствующим краям.

Для ускорения вычислений программа использует параллелизм. Создаётся пул процессов, количество которых задаётся параметром (в экспериментах от 1 до 6). Все данные - лабиринт, метрика α, позиции агентов - размещаются в разделяемой памяти (shared memory) с использованием RawArray из модуля ctypes.

При каждой итерации агенты делятся на примерно равные группы (батчи) по количеству процессов. Каждый процесс получает свой диапазон индексов агентов и независимо обрабатывает их, вычисляя новые позиции. Все процессы работают одновременно, обращаясь к общим данным напрямую из shared memory без копирования.

После того как все процессы завершают обработку своих батчей, главная программа собирает результаты: обновляет позиции всех агентов и проверяет флаги завершения. Если хотя бы один агент достиг выхода, выполнение останавливается.

На рисунке 1 представлен вывод программы.

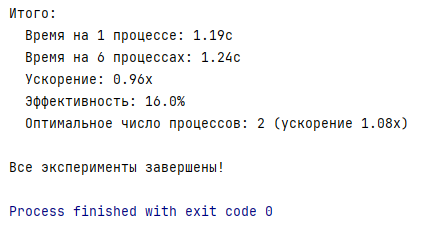
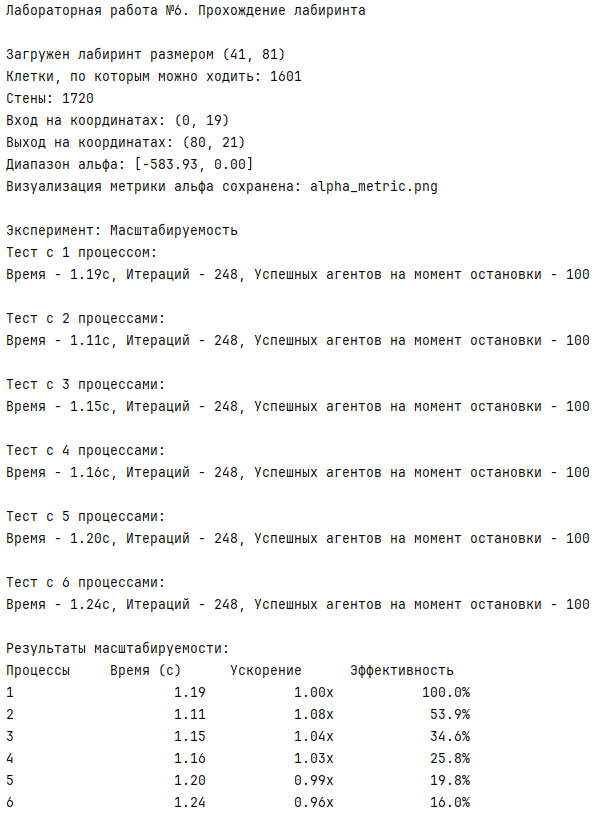


Рисунок 1 – Вывод программы

На рисунке 2 представлено решение лабиринта агентом.

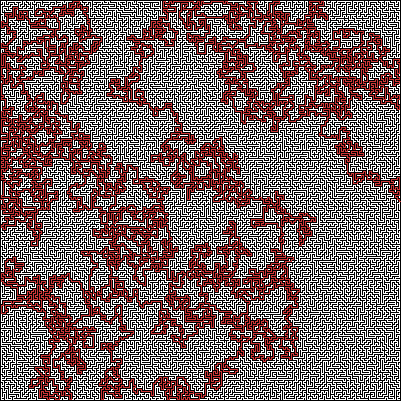


Рисунок 2 – Решение лабиринта

Распределение метрики K представлено на рисунке 3.

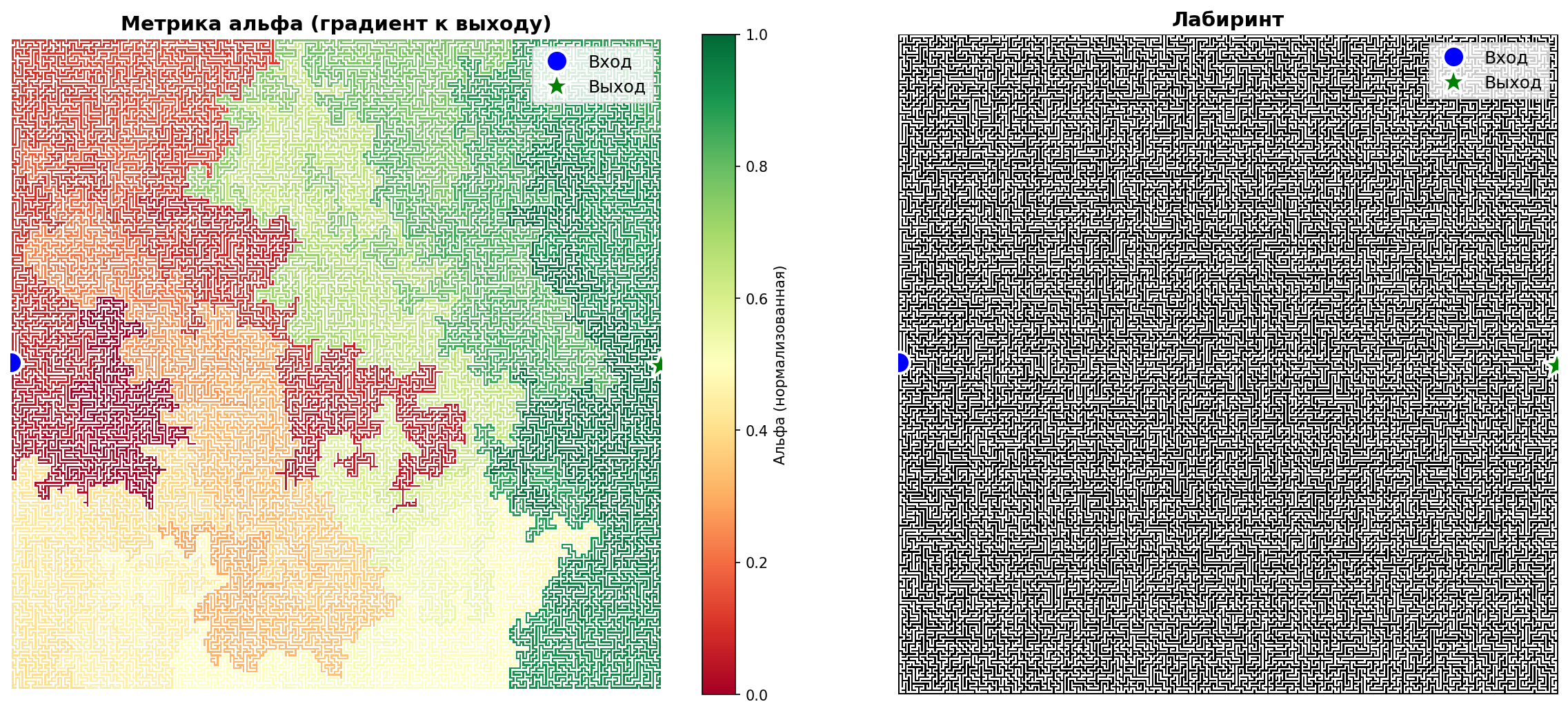


Рисунок 3 – Градиент к выходу

ОЦЕНКА ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛИЗАЦИИ

Если мы будем брать в качестве количества потоков число, равное количеству логических процессоров, то заметим, что после 6 потоков рост производительности практически прекращается. Это связано с тем, что мой процессор имеет 6 физических ядер, а использование 12 логических потоков (за счёт технологии Hyper-Threading) не даёт двукратного увеличения скорости, а лишь частично улучшает распределение нагрузки.

Оптимальное количество потоков для данной задачи соответствует числу физических ядер процессора. Увеличение числа потоков сверх этого значения не даёт существенного выигрыша и может даже ухудшить результат. График, показывающий использование 6 процессов показан на рисунке \*.

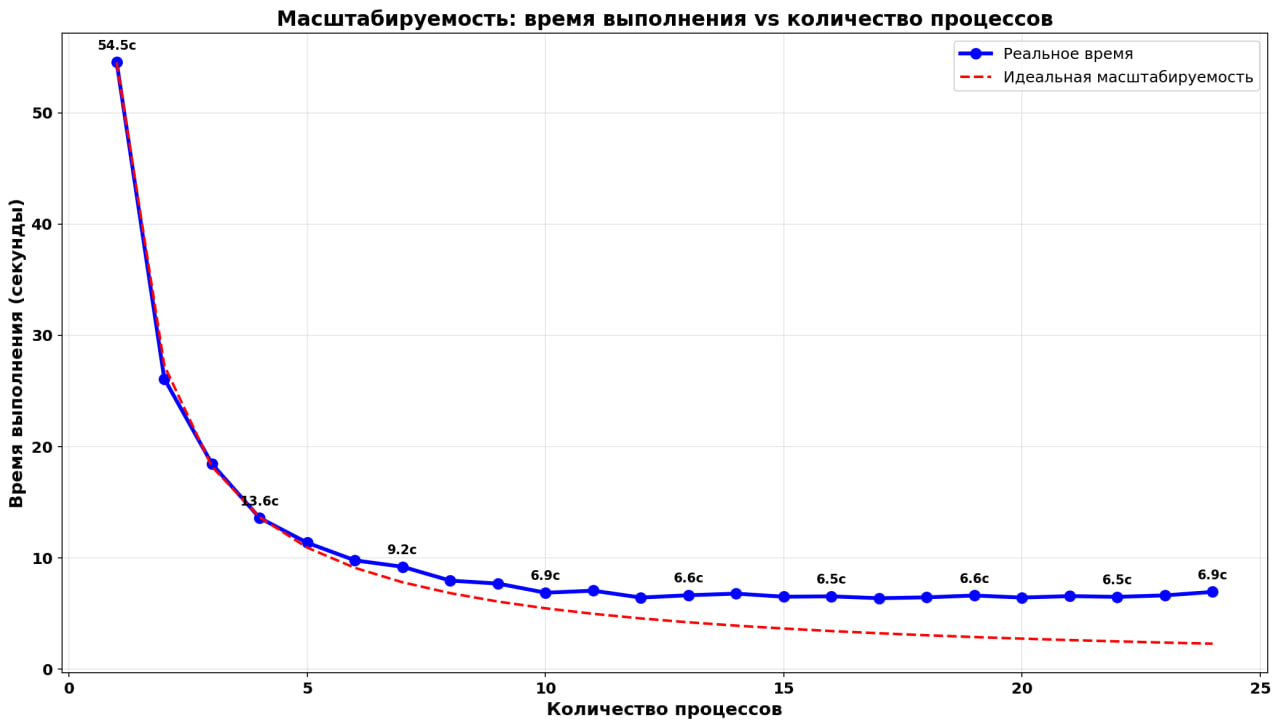


Рисунок 2 – График эксперимента 1

По графику для эксперимента 1 видно, что реальное время достаточно близко к идеальному. Отсюда можно понять, что можно выбрать оптимальное количество потоков для распараллеливания данного алгоритма сортировки (здесь это 12 потоков).

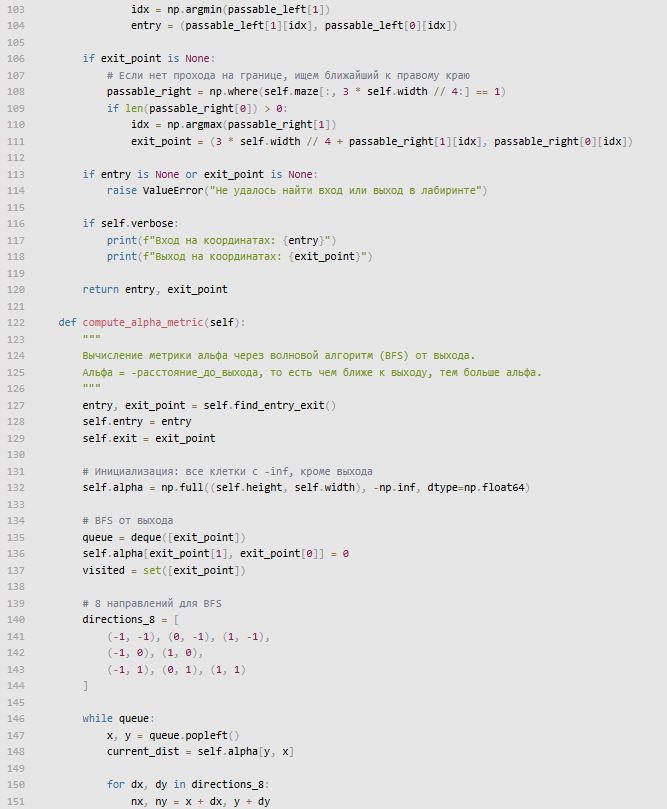
ПРИЛОЖЕНИЕ А – ЛИСТИНГ КОДА



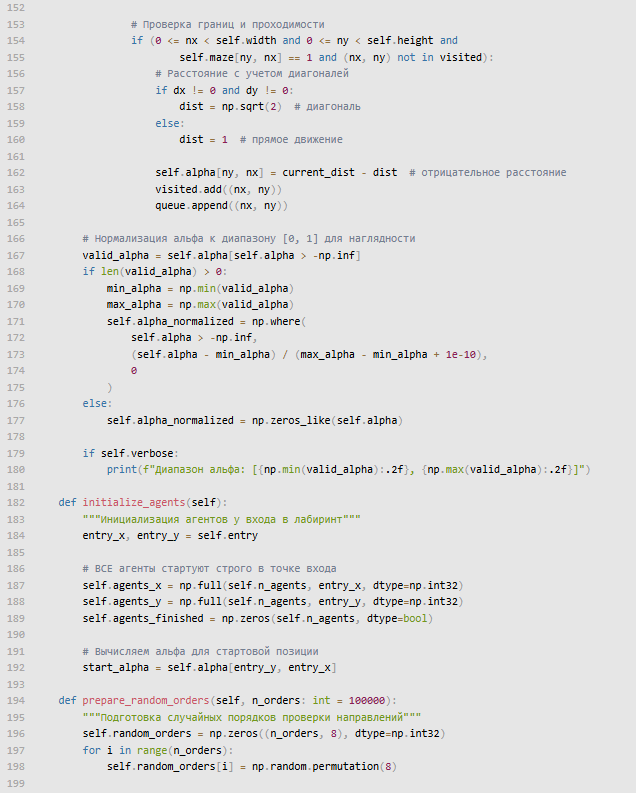
ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



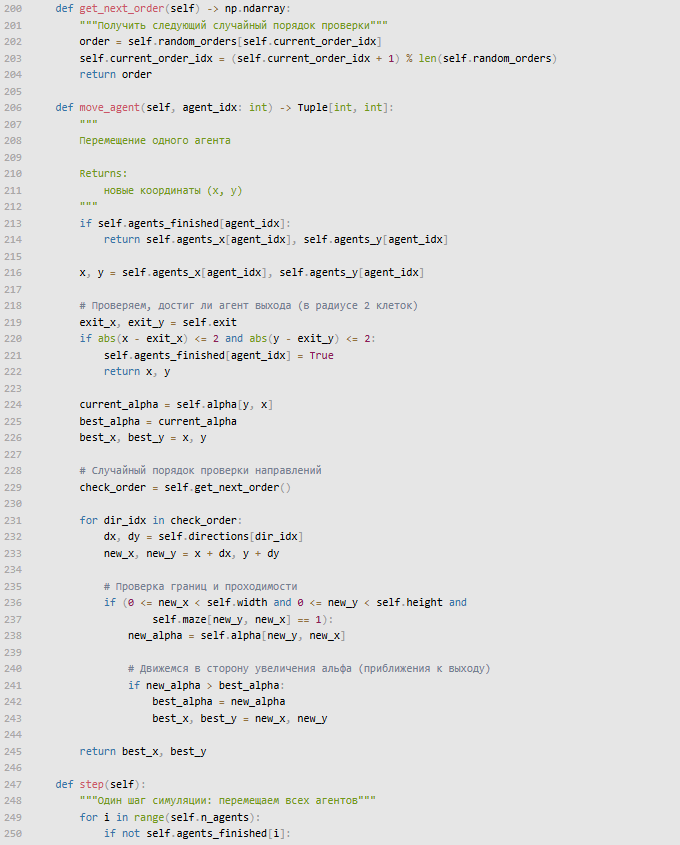
ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



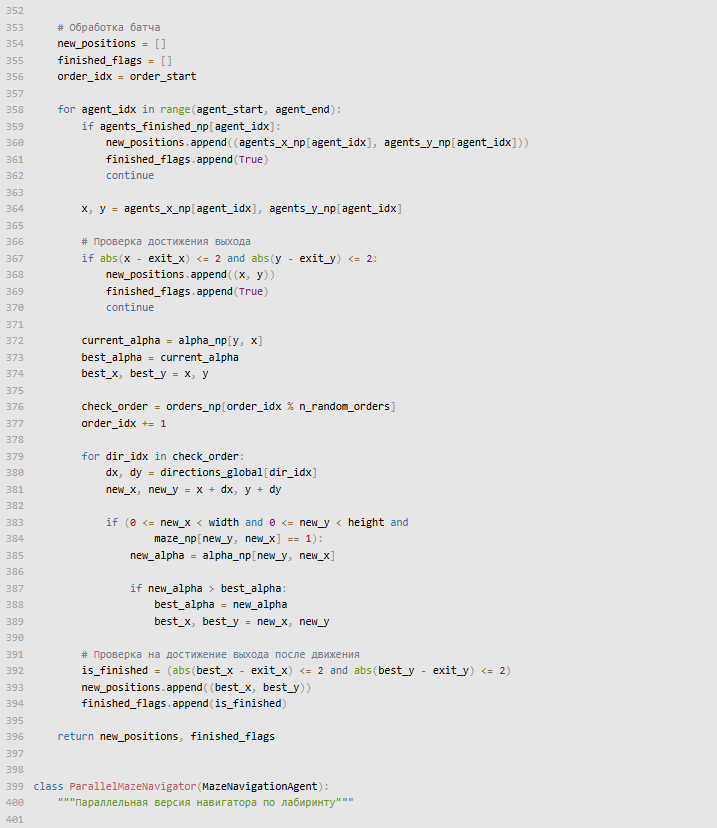
ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



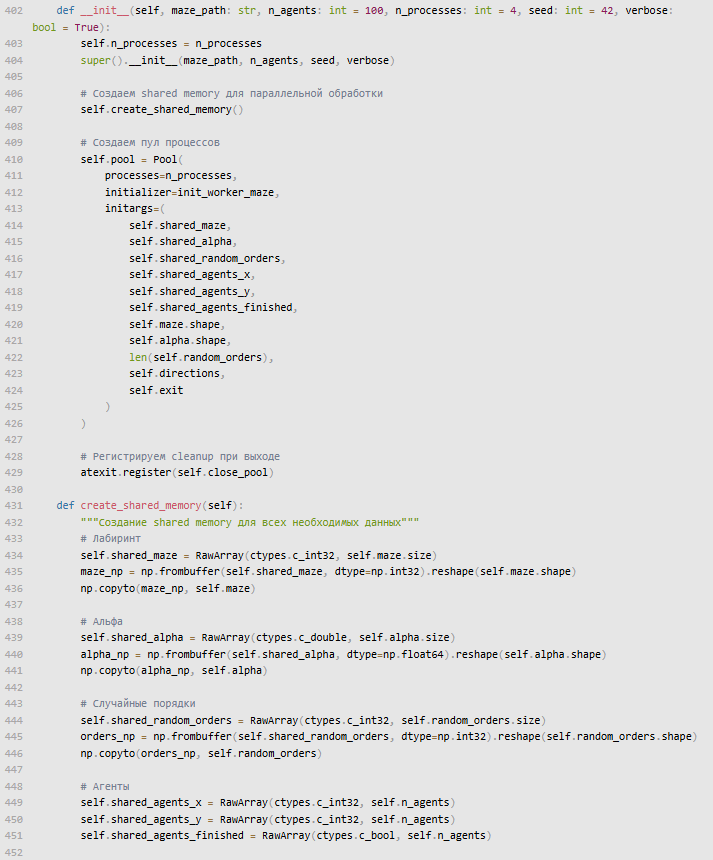
ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА



ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА

