МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА 25

КУРСОВАЯ РАБОТА (ПРОЕКТ) ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., канд. техн. наук |  |  |  | Е. М. Линский |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ |
| ИГРА В ХЕКСАГОН (ГЕКСАГОН) |
| по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |
| --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 2351 |  | Е. А. Лысов |
|  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

**Содержание**

[Постановка задачи 4](#_Toc185352770)

[Описание алгоритма 6](#_Toc185352787)

[Пошаговое выполнение алгоритма на примере 7](#_Toc185352788)

[Псевдокод 8](#_Toc185352789)

[Инструкция пользователя 10](#_Toc185352790)

[Тестовые примеры 11](#_Toc185352791)

[Список литературы 14](#_Toc185352792)

# Постановка задачи

# Задачей данной курсовой работы является разработка программы, которая моделирует игру в Гексагон, создает игровое поле и анализирует стратегии для достижения победы. Программа должна учитывать правила игры и предоставлять пользователю возможность визуализировать игровое поле, а также проводить анализ наилучших ходов

# Игра в Гексагон осуществляется на специальной ромбической доске, состоящей из шестиугольных ячеек, которая может быть любого размера, но традиционно используется размер n×n (чаще всего 11×11). На доске каждое поле в центре граничит с шестью соседними полями, а угловые поля имеют 2 или 3 соседа.

# Для игрока важным свойством является возможность построения цепи из своих фишек, соединяющей две стороны его цвета (красную или синюю). Задача программы — анализировать возможные стратегии и находить оптимальные ходы с целью создания такой цепи.

# Из литературы, которая помогает понять правила и стратегии игры, можно обратиться к книге Сильвии Назар «Игры разума», где подробно описываются принципы и аналитика игр, включая Гексагон.

# Примеры решения задачи

# Пример 1: Простой случай

# На доске 5x5 у красного игрока фишки расположены следующим образом:

# 

# Рисунок 1 – Расположение фишек красного игрока

# Где R — красные фишки и 1 — свободное поле. Красный игрок может выиграть, расположив свою следующую фишку, чтобы соединить две красные стороны.

# Пример 2: Неоптимальный ход

# На доске 5x5 у синего игрока фишки расположены так:

# 

# Рисунок 2 – Расположение фишек синего игрока

# Синему игроку нужно учесть, что размещение следующей фишки не должно дать красному игроку возможность создать цепь из своих фишек.

# Эти примеры демонстрируют важность стратегического мышления и анализа ходов, что и должно быть реализовано в разработанной программе

# Описание алгоритма

**Эти примеры демонстрируют важность стратегического мышления и анализа ходов, что и должно быть реализовано в разработанной программе.Начало формы**

**Основные идеи алгоритма:**

Алгоритм игры в Гексагон строится на стратегии поиска оптимальных ходов для обоих игроков, а также на анализе состояния игрового поля. Основные идеи включают:

1. Анализ соседей. Каждое поле имеет от 2 до 6 соседей. Игроки должны учитывать, как их ход влияет на доступные варианты для противника.
2. Поиск цепей. Необходимость находить все возможные цепи для каждого игрока, начиная с границ их сторон.
3. Минимизация ходов противника. Алгоритм выявляет наилучшие ответы на ходы соперника, блокируя их возможности.
4. Рекурсивный поиск. Использование методов поиска с возвратом (backtracking) для оценки возможных состояний.

Описание структур данных:

* Доска представляется двумерным массивом для представления игрового поля. Элементы массива могут быть пустыми, красными или синими.
* Для каждого поля находится список его соседей, что позволяет быстро изменять состояние поля и проверять возможные ходы.
* Для хранения текущих состояний во время поиска используется стек.

Шаги алгоритма:

* Инициализация доски.
* Поочередные ходы игроков.
* Проверка состояния доски после каждого хода на наличие цепей.
* Если найдено решение, завершить игру; если нет, перейти к следующему ходу.

**Пошаговое выполнение алгоритма на примере**

Допустим, у нас есть доска размером 3x3, и игроки делают свои ходы.

Начальная ситуация:



Рисунок 3 – начальная ситуация

Первый ход (Синий):

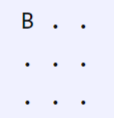


Рисунок 4 – первый ход

Второй ход (Красный):

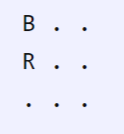


Рисунок 5 – второй ход

Третий ход (Синий):

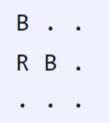


Рисунок 6 – третий ход

Четвертый ход (Красный):

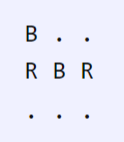


Рисунок 7 – четвертый ход

Проверка состояния доски. Алгоритм анализирует цепи и понимает, что путь от одной стороны к другой еще не закрыт.

Псевдокод

Класс HexGame

Переменные:

board: двумерный массив Player (размер BOARD\_SIZE x BOARD\_SIZE)

currentPlayer: Player (RED или BLUE)

firstMoveDone: Логическая переменная

isBoardFlipped: Логическая переменная

Конструктор HexGame

board = [Создать двумерный массив размером BOARD\_SIZE x BOARD\_SIZE, заполнить NONE]

currentPlayer = BLUE

firstMoveDone = ЛОЖЬ

isBoardFlipped = ЛОЖЬ

Метод displayBoard

Вывести " " // Заголовок столбцов

Для i от 0 до BOARD\_SIZE

Вывести i + " " // Номер строки

Для j от 0 до BOARD\_SIZE

Если isBoardFlipped == ИСТИНА

player = getFlippedPlayer(board[j][i]) // Получить перевернутого игрока

Иначе

player = board[i][j]

Если player == RED

Вывести "R "

Иначе Если player == BLUE

Вывести "B "

Иначе

Вывести ". "

Вывести новую строку

Вывести "RED зона: Row 0 до " + (BOARD\_SIZE / 2 - 1) + ", BLUE зона: Row " + (BOARD\_SIZE / 2) + " до " + (BOARD\_SIZE - 1)

Метод makeMove(row, col)

Если row и col валидны и board[row][col] == NONE

board[row][col] = currentPlayer // Установить ход текущего игрока

Если firstMoveDone == ЛОЖЬ

firstMoveDone = ИСТИНА

Вывести "Игрок " + (currentPlayer == RED ? "RED" : "BLUE") + " совершил первый ход. Повернуть доску? (y/n): "

choice = ВводИспользователя()

Если choice == 'y' или choice == 'Y'

isBoardFlipped = ИСТИНА

currentPlayer = (currentPlayer == BLUE) ? RED : BLUE

Вывести "Доска перевернута. Игрок сменился на " + (currentPlayer == RED ? "RED" : "BLUE")

currentPlayer = (currentPlayer == BLUE) ? RED : BLUE

Иначе

Вывести "Некорректный ход. Попробуйте снова."

Метод getCurrentPlayer

Вернуть currentPlayer

Метод checkWin

Вернуть (checkWinForPlayer(RED) или checkWinForPlayer(BLUE)) // Проверка победы

Метод winner

Если checkWinForPlayer(RED)

Вернуть RED

Если checkWinForPlayer(BLUE)

Вернуть BLUE

Вернуть NONE

Метод getFlippedPlayer(player)

Если isBoardFlipped == ИСТИНА

Если player == RED

Вернуть BLUE

Иначе Если player == BLUE

Вернуть RED

Вернуть NONE

Вернуть player

Метод checkWinForPlayer(player)

Создать массив visited (размер BOARD\_SIZE x BOARD\_SIZE, заполненный значениями ЛОЖЬ)

Создать очередь q

Если player == RED

Для i от 0 до BOARD\_SIZE

Если board[0][i] == RED

q.Добавить((0, i))

visited[0][i] = ИСТИНА

Иначе Если player == BLUE

Для i от 0 до BOARD\_SIZE

Если board[i][0] == BLUE

q.Добавить((i, 0))

visited[i][0] = ИСТИНА

Определить направления для проверки соседей (направления = [(0, 1), (1, 0), (1, -1), (0, -1), (-1, 0), (-1, 1)])

Пока q не пуста

p = q.УдалитьПервый() // Получить координаты из очереди

Если (player == RED и p.первая\_координата == BOARD\_SIZE - 1) или (player == BLUE и p.вторая\_координата == BOARD\_SIZE - 1)

Вернуть ИСТИНА

Конец Если

Для каждого dir в направлениях

nx = p.первая\_координата + dir.первая\_координата

ny = p.вторая\_координата + dir.вторая\_координата

Если (nx валиден и ny валиден и board[nx][ny] == player и visited[nx][ny] == ЛОЖЬ)

visited[nx][ny] = ИСТИНА

q.Добавить((nx, ny))

Конец Для

Конец Пока

Вернуть ЛОЖЬ

Сложность алгоритма:

Анализ сложности алгоритма будет зависеть от размеров доски и количества возможных ходов. Наиболее простой расчет дает сложность:

* Проверка всех соседей для каждого хода требует O(6) времени, так как у каждого поля не более 6 соседей. Однако O(6) не используется, так как 6 — константа, а значит, сложность этой операции относится к O(1).
* Рекурсивный поиск состояния в худшем случае ведет к O(n2) состояниям на доске n×n

Таким образом, после оптимизации, общая сложность алгоритма O(n2).

## Инструкция пользователя

Программа запускается пользователем из командной строки.

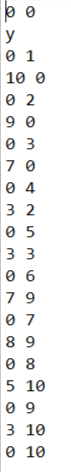
Командная строка принимает три параметра: имя проекта с кодом (например, cursovaya.exe), имя файла, в котором находятся исходные данные, то есть поочередные ходы игроков в формате первая координата x, а вторая координата y, (например, input.txt); имя файла, в который будет записан результат после выполнения программы методом полного перебора (например: output.txt).

Входной и выходной файлы должен быть в формате .txt.

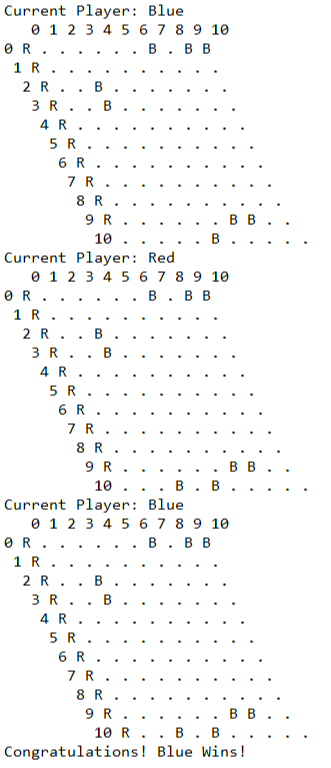
## Тестовые примеры

**Тест 1**

Входные данные:

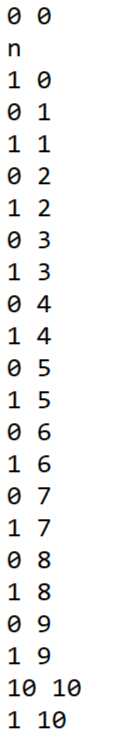


Выходные данные:

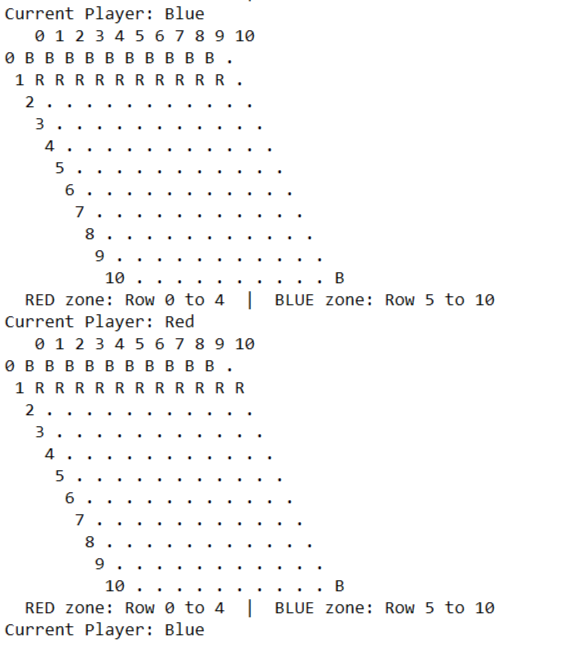


**Тест 2**

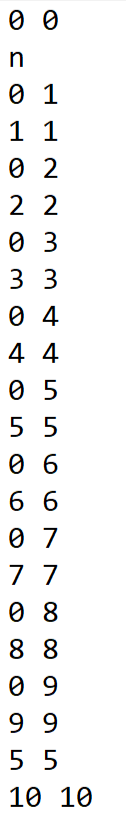
Входные данные:



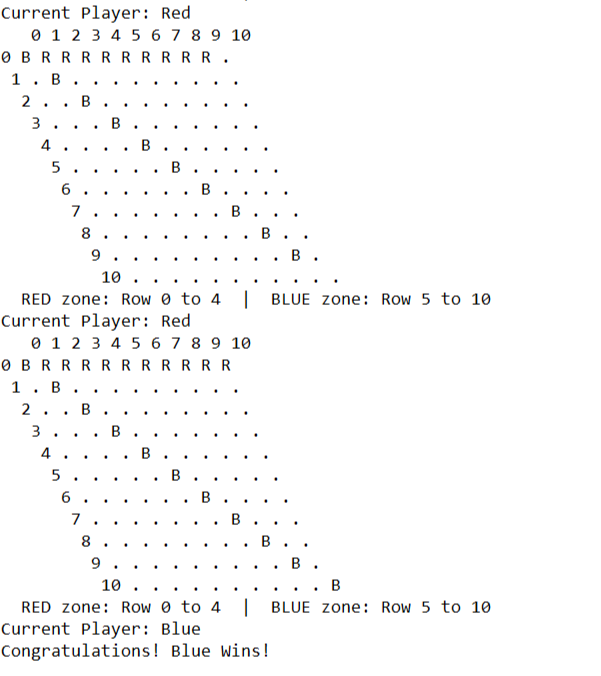
Выходные данные:



Тест 3 Входные данные:



Выходные данные



## Список литературы

1. Уэзерелл Ч. Этюды для программистов. – 1982.
2. Седжвик Р. Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка //Поиск–СПб.: ДиаСофтЮП. – 2001.
3. Назар С. Игры разума. История жизни Джона Нэша, гениального математика и лауреата Нобелевской премии //М.: АСТ: CORPUS. – 2017. – Т. 747.