МИНОБРНАУКИ РОССИИ

РГУ НЕФТИ И ГАЗА (НИУ) ИМЕНИ И.М. ГУБКИНА

Факультет **Автоматики и вычислительной техники**
Кафедра **Автоматизированных систем управления**

Оценка комиссии: _____ Рейтинг: ____
Подписи членов комиссии:

_____ Волков Д. А.
(подпись) (фамилия, имя, отчество)
_____ Мухина А. Г.
(подпись) (фамилия, имя, отчество)

_____
(дата)

КУРСОВАЯ РАБОТА

по дисциплине Математическая логика и теория алгоритмов

на тему Программная реализация алгоритма

«К ЗАЩИТЕ»

Доцент к. т. н. Волков Д. А.
(должность, ученая степень; фамилия, и.о.)

_____
(подпись)

_____
(дата)

ВЫПОЛНИЛ:
Студент группы **АС-22-05**
(номер группы)

Ильичев Роман Сергеевич
(фамилия, имя, отчество)

_____
(подпись)

_____
(дата)

Москва, 20 23

МИНОБРНАУКИ РОССИИ

РГУ НЕФТИ И ГАЗА (НИУ) ИМЕНИ И.М. ГУБКИНА

Факультет **Автоматики и вычислительной техники**
Кафедра **Автоматизированных систем управления**

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

по дисциплине Математическая логика и теория алгоритмов

на тему Программная реализация алгоритма

ДАНО студенту **Ильичеву Роману Сергеевичу** (фамилия, имя, отчество в дательном падеже) группы **АС-22-05** (номер группы)

Содержание работы:

1. Описание и анализ «Алгоритм Уилсона для генерации лабиринта».
2. Выбор и обоснование необходимых для разработки средств.
3. Программная реализация «Алгоритма Уилсона».

Исходные данные для выполнения работы:

1. Материалы курса «Математическая логика и теория алгоритмов».
2.
3.

Рекомендуемая литература:

1. Колдаев, В. Д. Основы алгоритмизации и программирования: учебное пособие / В.Д. Колдаев; под ред. проф. Л.Г. Гагариной. — Москва: ФОРУМ: ИНФРА-М, 2022. — 414 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0733-7. - Текст: электронный. - URL: https://znanium.com/catalog/product/1735805 (дата обращения: 1.09.2023). – Режим доступа: по подписке.
2. Трофимов, В. В. Алгоритмизация и программирование : учебник для вузов / В. В. Трофимов, Т. А. Павловская ; под редакцией В. В. Трофимова. — 4-е изд. — Москва: Издательство Юрайт, 2023. — 118 с. — (Высшее образование). — ISBN 978-5-534-17497-7. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/533199 (дата обращения: 1.09.2023).
3. Скорубский, В. И. Математическая логика: учебник и практикум для вузов / В. И. Скорубский, В. И. Поляков, А. Г. Зыков. — Москва: Издательство Юрайт, 2023. — 211 с. — (Высшее образование). — ISBN 978-5-534-01114-2. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/511996 (дата обращения: 14.09.2023).

Графическая часть:

1. _____
2. _____

Требования к представлению результатов:

✓	Электронная версия
	Бумажный вариант и электронный образ документа

Руководитель: к.т.н. (уч.степень) доцент (должность) _____ (подпись) Волков Д.А. (фамилия, имя, отчество)

Задание принял к исполнению: студент _____ (подпись) Ильичев Р.С. (фамилия, имя, отчество)

Оглавление

Введение

Актуальность алгоритмов генерации лабиринтов

В наше актуальность генерации лабиринтов остается неоспоримой и многогранной. Этот процесс становится не просто занимательным аспектом развлекательных приложений, таких как видеоигры, но и играет ключевую роль в различных практических областях. Вот несколько сфер, подчеркивающих актуальность генерации лабиринтов в современном информационном обществе:

1. Развлекательная индустрия. Видеоигры, кино и другие форматы активно используют лабиринты как элементы сюжета или игрового процесса.
2. Обучение и тренировка. Генерация лабиринтов широко используется в обучающих средах и симуляторах для тренировки пространственного мышления, логического мышления и решения проблем. Это особенно важно в области обучения робототехнике.
3. Компьютерное зрение и искусственный интеллект.
4. Задачи планирования. Генерация лабиринтов находит применение в оптимизации маршрутов и планировании перемещения.
5. Креативное программирование и исследование алгоритмов. Для программистов и исследователей алгоритмов генерация лабиринтов представляет собой увлекательное поле для экспериментов и творчества.

С учетом перечисленных аспектов становится очевидным, что генерация лабиринтов является актуальной и перспективной областью исследований.

Алгоритм Уилсона, его особенности

Одним из эффективных и интересных алгоритмов, используемых для генерации лабиринтов, является алгоритм Уилсона. Этот алгоритм, предложенный Робертом Уилсоном в 1997 году, привлекает внимание исследователей своей уникальной концепцией и выдающейся эффективностью в создании лабиринтов различной сложности.

Алгоритм Уилсона применяет концепцию Марковских цепей, что добавляет элемент случайности и непредсказуемости в процесс генерации. Интересным аспектом алгоритма Уилсона является его способность избегать

создания тупиковых путей, называемых "шрамами". Алгоритм Уилсона, используя случайные решения, подчеркивает важность случайности в создании уникальных и разнообразных лабиринтов.

Цель данной курсовой работы - представить исчерпывающий обзор алгоритма Уилсона, рассмотреть его математические основы, принципы работы и практические применения. На основе проведенного исследования ожидается выявление сильных и слабых сторон алгоритма Уилсона

Выбор и обоснование необходимых для разработки средств

Для разработки проекта я выбрал язык программирования Python. Этот язык проще, чем другие, и в нём есть все необходимые библиотеки для написания программы. А именно мне понадобилась библиотека tkinter.

Tkinter, входящая в стандартную библиотеку Python, предоставляет простой и интуитивно понятный способ создания графических интерфейсов. Она подходит для разработки простых приложений и идеально подходит для визуализации лабиринтов.

Также Python является кроссплатформенным языком программирования, что означает, что разработанное приложение будет работать на различных операционных системах без необходимости значительных изменений.

Основная часть

Описание алгоритма Уилсона

Алгоритм Уилсона значительно сложнее всех остальных алгоритмов генерации лабиринтов как в реализации, так и в понимании. Цель алгоритма Уилсона – генерация равновероятного случайного остовного дерева.

Перемещаясь по лабиринту, мы «запоминаем» все вершины, в которых побывали до момента нахождения вершины остовного дерева. Как только мы натыкаемся на уже добавленную вершину, мы присоединяем получившийся путь к нашему генерируемому лабиринту. Если создается цикл в подграфе, то удаляем его.

После присоединения подграфа к остовному дереву, выбор следующей случайной точки происходит исключительно из еще не присоединенных вершин.

Алгоритм Уилсона генерирует абсолютно случайные лабиринты без какого-либо смещения. Алгоритм не имеет предпочтений по направленности, запутанности или ещё каким-либо характеристикам.

Шаги алгоритма:

1. Выбрать случайную вершину, не принадлежащую остовному дереву и добавить её в дерево;
2. Выбрать случайную вершину, не принадлежащую остовному дереву и начать обход графа (лабиринта), пока не придём в уже добавленную вершину дерева. Если образуется цикл, удалить его;
3. Добавить все вершины получившегося подграфа в остовное дерево;
4. Повторять шаги 2-3, пока все вершины не будут добавлены в остовное дерево.

Ниже представлена блок-схема алгоритма, где встречаются следующие функции:

- RandomCell() – выбор любой ячейки, которая еще не была посещена;
- FindAdjCells() – возвращение массива пар координат ячеек, соседних текущей.

- OpenWall() – открывает стенки между двумя ячейками лабиринта.

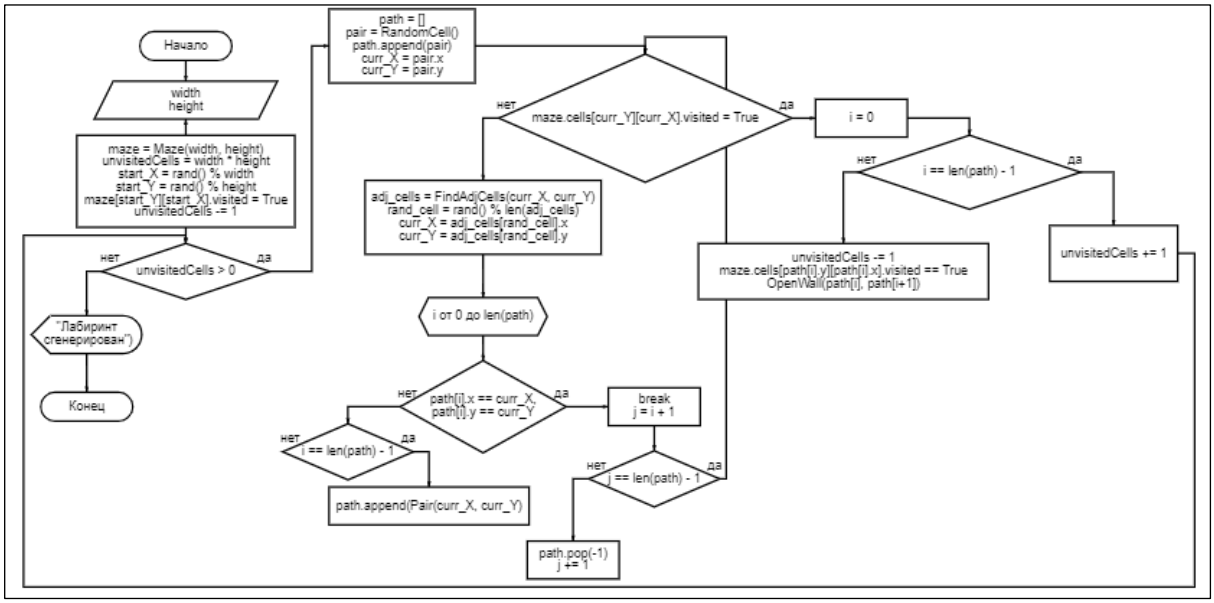

Рисунок 1. Блок-схема алгоритма

Плюсы данного алгоритма:

- Отсутствует какое-либо смещение;
- Лабиринты абсолютно случайны, поэтому невозможно создать определенный алгоритм их решения;
- Сложность решения для человека;
- Гарантированная связность (в лабиринте существует путь между любыми двумя вершинами);
- Нет бессмысленного блуждания.

Минусы данного алгоритма:

- Непростая реализация;
- Падения скорости в начале генерации;
- Неэффективен для больших графов;
- Возможность зацикливания;
- Большие требования к памяти.

Реализация алгоритма на Python

Для того, чтобы реализовать алгоритм мне понадобились, следующие библиотеки: time, tkinter, random, а также написанный мною файл Maze.py

```
import time
import tkinter as tk

from random import *
from tkinter import messagebox
from PIL import Image, ImageDraw

from Maze import *
```

Рисунок 2. Импортирование библиотек

Мною были написаны классы Cell и Maze для того, чтобы легче было работать с объектом лабиринта. Класс Cell представляет собой ячейку лабиринта. В нём есть следующие поля:

- __x и __y – координаты ячеек;
- visited – переменная типа bool, являющаяся некоторым флагом, указывающим на то, была ли ячейка посещена раннее;
- Left, Right, Top, Bottom – переменные типа bool, которые нужны для того, чтобы отмечать какие стены ячейки существуют, а какие открыты.

```
class Cell():
    def __init__(self, x: int, y: int):
        self.__x: int = x
        self.__y: int = y
        self.visited: bool = False
        self.Left: bool = False
        self.Right: bool = False
        self.Top: bool = False
        self.Bottom: bool = False
```

Рисунок 3. Класс Cell

Класс Maze уже представляет собой лабиринт, который состоит из ячеек. При создании лабиринта вызывается конструктор, который принимает на вход ширину и высоту лабиринта. Создаётся двумерный массив, который заполняется ячейками. Также у данного класса есть два метода для получения высоты и ширины лабиринта.

```
class Maze():
    def __init__(self, width: int, height: int):
        self.__width = width
        self.__height = height

        cell = Cell(x = 0, y = 0)
        self.cells = [[cell] * width for i in range(height)]
        for i in range(0, height):
            for j in range(0, width):
                cell = Cell(x = j, y = i)
                self.cells[i][j] = cell
                self.cells[i][j] = cell

    16 usages
    def GetWidth(self):
        return self.__width

    16 usages
    def GetHeight(self):
        return self.__height
```

Рисунок 4. Класс Maze

В начале программы, я задаю 4 цвета, которые будут использоваться при создании графического интерфейса.

```
color1 = "#DDA0DD"
color2 = "#EEAEEE"
color3 = "#CD96CD"
color4 = "red"
```

Рисунок 5. Цвета

Основной код программы – это класс Generator. Именно этот класс отвечает за создание графической оболочки и реализацию алгоритма Уилсона. При вызове конструктора, заполняются следующие поля:

- maze – объект класса Maze.
- cells – двумерный массив для хранения ячеек, а именно для квадратиков, нарисованных на холсте tkinter.Canvas.
- wall_ver, wall_hor – двумерные массивы для хранения вертикальных и горизонтальных стенок, а именно для вертикальных и горизонтальных линий нарисованных на холсте canvas.
- root – это главное окно. При создании окна задаётся цвет фона, название и такие параметры, как высота и ширина окна. Метод geometry() используется для настройки размеров, а также расположения относительно самого экрана.
- width_wall, width_cell – это ширина стенки и ширина ячейки лабиринта соответственно. Ширина ячейки рассчитывается в зависимости от размера лабиринта.

```
class Generator():
    def __init__(self):

        self.maze = Maze(width=20, height=40)
        self.cells = [[]]
        self.wall_ver = [[]]
        self.wall_hor = [[]]

        """Настройка размеров окна, ячейки лабиринта"""
        self.root = tk.Tk()
        self.root.configure(bg=color2)
        self.root.title("Алгоритм Уилсона для генерациия лабиринта")

        self.width_root = 1000
        self.height_root = 650
        self.k = self.width_root / (self.height_root - 50)
        width_screen = self.root.winfo_screenwidth()
        height_screen = self.root.winfo_screenheight()
        self.root.geometry(f"{self.width_root}x{self.height_root}"
                           f"+{int((width_screen - self.width_root) / 2)}"
                           f"+{int((height_screen - self.height_root) / 3)}")
        self.root.resizable(width=False, height=False)


        self.width_wall = 2
        self.width_cell = 0
```

Рисунок 6. Класс Generator, конструктор

Также при создании объекта создаётся фрейм (tkinter.Frame) для меню, на котором идёт размещение двух полей для ввода ширины и высоты лабиринта (tkinter.Entry), и холст (tkinter.Canvas), на котором и будет прорисовываться лабиринт. И с помощью root.mainloop() запускается цикл обработки событий, благодаря которому окно реагирует на внешние события и воспроизводится.

```
        """Разбиение на фреймы"""
        self.frameMenu = tk.Frame(self.root, height=50, bg=color1)
        self.frameMenu.place(relx=0, relwidth=1, rely=0)

        self.canvasMaze = tk.Canvas(self.root, highlightthickness=0,
                                    highlightbackground="black", bg=color2,
                                    width=self.width_cell * self.maze.GetWidth() - 2,
                                    height=self.width_cell * self.maze.GetHeight() - 2)
        self.canvasMaze.place(anchor='center', relx=0.5, y=350)


        """Создание кнопок и полей для взаимодействия с пользователем"""
        self.entry_width = tk.Entry(self.frameMenu, bg=color1, font=("Times New Roman", 14))
        self.entry_width.place(relx=0.12, rely=0.5, anchor="center", relheight=0.7, relwidth=0.2)
        self.entry_width.insert(0, " Ширина:")
        self.entry_width.bind("<FocusIn>", self.focus_in_width)
        self.entry_width.bind("<FocusOut>", self.focus_out_width)

        self.entry_height = tk.Entry(self.frameMenu, bg=color1, font=("Times New Roman", 14))
        self.entry_height.place(relx=0.33, rely=0.5, anchor="center", relheight=0.7, relwidth=0.2)
        self.entry_height.insert(0, " Высота:")
        self.entry_height.bind("<FocusIn>", self.focus_in_height)
        self.entry_height.bind("<FocusOut>", self.focus_out_height)

        self.btn_start = tk.Button(self.frameMenu, text="Генерировать", command=self.start,
                                   font=("Times New Roman", 14), bg=color1, cursor="hand2")
        self.btn_start.place(relx=0.8, rely=0.5, anchor="center", relheight=0.7, relwidth=0.2)

        self.root.mainloop()
```

Рисунок 7. Класс Generator, конструктор

По нажатию на кнопку «Генерировать» вызывается функция start() и запускается алгоритм Уилсона. В самом начале этого блока с помощью

конструкции try – except обрабатываются исключения, а именно получение значений из полей, заполненных пользователем: ширины и высоты лабиринта. Затем идёт проверка на введённые значения ширины и высоты лабиринта, они должны иметь размер меньше 100, так как данный алгоритм работает не быстро. Создаётся объект класса Maze, используя параметры, введённые пользователем. Вызывается функция DetermineSize(), которая нужна для того, чтобы определить размер ячейки лабиринта и размеров холста (tkinter.Canvas). С помощью циклов for рисуется лабиринт, и идёт заполнение двумерных массивов cells, wall_ver, wall_hor. И в самом конце блока вызывается метод класса Generator – Wilson(). Это и есть функция, выполняющая алгоритм Уилсона.

```
def start(self):
    flag: bool = False
    width = 0
    height = 0

    try:
        width = int(self.entry_width.get())
        height = int(self.entry_height.get())
        flag = True
    except:
        messagebox.showerror("Ошибка", "Введите корректные параметры!")

    if flag and (width <= 100 and height <= 100):

        self.maze = Maze(width=width, height=height)
        self.DetermineSize()

        self.cells = [[0] * self.maze.GetWidth() for i in range(self.maze.GetHeight())]
        self.wall_ver = [[0] * (self.maze.GetWidth() - 1) for i in range(self.maze.GetHeight())]
        self.wall_hor = [[0] * self.maze.GetWidth() for i in range(self.maze.GetHeight() - 1)]


        for row in range(self.maze.GetHeight()):
            for column in range(self.maze.GetWidth()):
                rect = self.canvasMaze.create_rectangle(column * self.width_cell, row * self.width_cell,
                                                        (column + 1) * self.width_cell,
                                                        (row + 1) * self.width_cell,
                                                        fill=color2, outline="")
                self.cells[row][column] = rect
```

Рисунок 8. Функция start()

```
                if column != self.maze.GetWidth() - 1:
                    line_ver = self.canvasMaze.create_line((column + 1) * self.width_cell,
                                                           row * self.width_cell,
                                                           (column + 1) * self.width_cell,
                                                           (row + 1) * self.width_cell,
                                                           width=self.width_wall)
                    self.wall_ver[row][column] = line_ver

                if row != self.maze.GetHeight() - 1:
                    line_hor = self.canvasMaze.create_line(column * self.width_cell,
                                                           (row + 1) * self.width_cell,
                                                           (column + 1) * self.width_cell,
                                                           (row + 1) * self.width_cell,
                                                           width=self.width_wall)
                    self.wall_hor[row][column] = line_hor

        self.Wilson()
        # self.SaveCanvas()

    elif flag:
        messagebox.showerror("Большие значения",  "Ширина и высота должны быть меньше 100!")
```

Рисунок 9. Функция start()

```
def DetermineSize(self):
    if self.maze.GetWidth()/self.maze.GetHeight() > self.k:
        self.width_cell = self.width_root / self.maze.GetWidth()
    else:
        self.width_cell = (self.height_root - 50) / self.maze.GetHeight()

    self.canvasMaze.configure(width=self.width_cell * self.maze.GetWidth() - 2,
                              height=self.width_cell * self.maze.GetHeight() - 2,
                              highlightthickness=1)
```

Рисунок 10. Функция DetermineSize()

В функции Wilson() мы задаём такую переменную, как unvisitedCells. Она содержит информацию о количестве всех непосещённых ячеек. В самом начале её значение равно количеству всех ячеек лабиринта.

Случайным образом выбирается ячейка из лабиринта, а точнее координаты этой ячейки. И клетка обозначается на холсте другим цветом и отмечается посещённой.

Затем запускается цикл while, который каждый раз проверяет, все ли ячейки у нас посещены. В данном блоке создаётся массив path, необходимый для хранения клеток создаваемого пути, случайным образом выбирается ещё не посещённая ячейка, которая обозначается на холсте красным цветом.

После этого строится путь, пока мы не наткнёмся на уже посещённую ячейку лабиринта. Случайным образом выбирается соседняя клетка. Если эта клетка создаёт цикл, то цикл удаляется, и путь начинает формироваться заново. Иначе эта клетка на холсте перекрашивается в красный. После того, как мы пришли в посещённую ячейку, формируется путь: на холсте удаляются стенки, найденный путь перекрашивается в другой цвет. Начинает строиться другой путь.

Функция RandomCell() предназначена для выбора случайной, еще не посещённой ячейки лабиринта.

Функция FindAdjacentCells() возвращает массив пар координат ячеек, соседних той, в которой мы находимся.

Функция hasCycle() возвращает истинное значение при наличии цикла и при этом удаляет его, иначе ложное значение, если цикл отсутсвует.

Функция OpenWall() убирает стенки между двумя ячейками лабиринта.

```
def Wilson(self):

    unvisitedCells: int = self.maze.GetWidth() * self.maze.GetHeight();

    start_x: int = randint(0, self.maze.GetWidth() - 1)
    start_y: int = randint(0, self.maze.GetHeight() - 1)
    self.canvasMaze.itemconfig(self.cells[start_y][start_x], fill=color3)

    self.maze.cells[start_y][start_x].visited = True
    unvisitedCells -= 1

    while unvisitedCells > 0:

        path: list = []
        pair = self.RandomCell(curr_x=start_x, curr_y=start_y)
        path.append(pair)
        curr_x: int = pair.x
        curr_y: int = pair.y
        self.canvasMaze.itemconfig(self.cells[curr_y][curr_x], fill=color4)

        while not self.maze.cells[curr_y][curr_x].visited:

            adjacent_cells: list = self.FindAdjacentCells(curr_x=curr_x, curr_y=curr_y)
            rand_cell: int = randint(0, len(adjacent_cells) - 1)
            curr_x = adjacent_cells[rand_cell].x
            curr_y = adjacent_cells[rand_cell].y
```

Рисунок 11. Функция Wilson()

```
            if not self.hasCycle(path, curr_x, curr_y):
                path.append(Pair(curr_x, curr_y))
                self.canvasMaze.itemconfig(self.cells[curr_y][curr_x], fill=color4)
                self.canvasMaze.update()
                time.sleep(0.05)


        for i in range(len(path)):
            unvisitedCells -= 1;
            self.maze.cells[path[i].y][path[i].x].visited = True
            self.canvasMaze.itemconfig(self.cells[path[i].y][path[i].x], fill=color3)
            if i != len(path) - 1:
                self.OpenWall(path[i], path[i + 1])

        unvisitedCells += 1

        self.canvasMaze.update()
        time.sleep(0.05)

    messagebox.showinfo("Конец!", "Лабиринт сгенерирован!")
```

Рисунок 12. Функция Wilson()

```
def RandomCell(self, curr_x: int, curr_y: int):

    curr_x = randint(0, self.maze.GetWidth() - 1)
    curr_y = randint(0, self.maze.GetHeight() - 1)

    while self.maze.cells[curr_y][curr_x].visited:
        curr_x = randint(0, self.maze.GetWidth() - 1)
        curr_y = randint(0, self.maze.GetHeight() - 1)

    return Pair(curr_x, curr_y)
```

Рисунок 13. Функции RandomCell()

```
def FindAdjacentCells(self, curr_x: int, curr_y: int):

    free_cell: list = []

    if curr_x > 0:
        free_cell.append(Pair(curr_x - 1, curr_y))

    if curr_x < (self.maze.GetWidth() - 1):
        free_cell.append(Pair(curr_x + 1, curr_y))

    if curr_y > 0:
        free_cell.append(Pair(curr_x, curr_y - 1))

    if curr_y < (self.maze.GetHeight() - 1):
        free_cell.append(Pair(curr_x, curr_y + 1))

    return free_cell
```

Рисунок 14. Функция FindAdjacentCells()

```
def hasCycle(self, path: list, x: int, y: int):

    for i in range(len(path)):
        if path[i].x == x and path[i].y == y:
            for j in range(i + 1, len(path)):
                self.canvasMaze.itemconfig(self.cells[path[-1].y][path[-1].x], fill=color2)
                self.canvasMaze.update()
                path.pop(-1)
            return True

    return False
```

Рисунок 15. Функция hasCycle()

```
def OpenWall(self, first: Pair, second: Pair):

    x1: int = first.x
    x2: int = second.x
    y1: int = first.y
    y2: int = second.y

    if x1 == x2:

        if y1 < y2:
            self.maze.cells[y1][x1].Bottom = True
            self.maze.cells[y2][x2].Top = True
            self.canvasMaze.delete(self.wall_hor[y1][x1])

        else:
            self.maze.cells[y1][x1].Top = True
            self.maze.cells[y2][x2].Bottom = True
            self.canvasMaze.delete(self.wall_hor[y2][x2])

    if y1 == y2:
        if x1 < x2:
            self.maze.cells[y1][x1].Right = True
            self.maze.cells[y2][x2].Left = True
            self.canvasMaze.delete(self.wall_ver[y1][x1])

        else:
            self.maze.cells[y1][x1].Left = True
            self.maze.cells[y2][x2].Right = True
            self.canvasMaze.delete(self.wall_ver[y2][x2])
```

Рисунок 16. Функция OpenWall()

Также в программе есть функции сохранения лабиринта в файл и картинкой и загрузки лабиринта из файла.

```
def save(self):
    entry_window = tk.Toplevel(self.root)
    entry_window.title("Название файла")
    entry_window.configure(bg=color2)
    width_screen = entry_window.winfo_screenwidth()
    height_screen = entry_window.winfo_screenheight()
    entry_window.geometry(f"{300}x{200}"
                          f"+{int((width_screen - 300) / 2)}"
                          f"+{int((height_screen - 200) / 3)}")

    label = tk.Label(entry_window, font=("Times New Roman", 14), bg=color2)
    label.configure(text="Название файла и картинки")
    label.place(anchor="center", relx=0.5, rely=0.2)

    entry_widget = tk.Entry(entry_window, font=("Times New Roman", 14), bg=color2)
    entry_widget.place(anchor="center", relx=0.5, rely=0.4, relwidth=0.8)

    # Функция, которая будет вызываться при нажатии кнопки ввода
    def on_submit():

        if self.flag_save:

            file_path = "./data/" + entry_widget.get()
            image_path = "./image/" + entry_widget.get() + ".png"
            entry_window.destroy()

            s = ""
```

Рисунок 17. Функция save()

```
            with open(file_path, "w") as file:
                file.write(str(self.maze.GetHeight()) + "\n")
                file.write(str(self.maze.GetWidth()) + "\n")
                for i in range(self.maze.GetHeight()):
                    for j in range(self.maze.GetWidth()):
                        s += "1" if self.maze.cells[i][j].Bottom else "0"
                        s += "1" if self.maze.cells[i][j].Right else "0"
                    file.write(s + "\n")
                    s = ""
            time.sleep(2)
            x = self.canvasMaze.winfo_x() + self.root.winfo_rootx() - 1
            y = self.canvasMaze.winfo_y() + self.root.winfo_rooty() - 1
            width = self.canvasMaze.winfo_reqwidth() + 2
            height = self.canvasMaze.winfo_reqheight() + 2
            screenshot = ImageGrab.grab(bbox=(x, y, x + width, y + height))
            screenshot.save(image_path)

        else:
            entry_window.destroy()
            messagebox.showerror("Ошибка", "Лабиринт не сгенерирован")

    # Создаем кнопку для отправки текста
    submit_button = tk.Button(entry_window, text="Сохранить", command=on_submit,
                              font=("Times New Roman", 12), bg=color2,
                              cursor="hand2")
    submit_button.place(anchor="center", relx=0.5, rely=0.75)
```

Рисунок 18. Функция save()

```python
def load(self):
    entry_window = tk.Toplevel(self.root)
    entry_window.title("Название файла")
    entry_window.configure(bg=color2)
    width_screen = entry_window.winfo_screenwidth()
    height_screen = entry_window.winfo_screenheight()
    entry_window.geometry(f"{300}x{200}"
                          f"+{int((width_screen - 300) / 2)}"
                          f"+{int((height_screen - 200) / 3)}")

    label = tk.Label(entry_window, font=("Times New Roman", 14), bg=color2)
    label.configure(text="Название файла")
    label.place(anchor="center", relx=0.5, rely=0.2)


    entry_widget = tk.Entry(entry_window, font=("Times New Roman", 14), bg=color2)
    entry_widget.place(anchor="center", relx=0.5, rely=0.4, relwidth=0.8)

    # Функция, которая будет вызываться при нажатии кнопки ввода
    def on_submit():
        file_path = "./data/" + entry_widget.get()
        self.canvasMaze.delete("all")
        self.flag_save = False

        try:
            with open(file_path, "r") as file:
                height = int(file.readline())
                width = int(file.readline())

                self.maze = Maze(width=width, height=height)
                self.DetermineSize()

                self.cells = [[0] * self.maze.GetWidth() for i in range(self.maze.GetHeight())]
                self.wall_ver = [[0] * (self.maze.GetWidth() - 1) for i in range(self.maze.GetHeight())]
                self.wall_hor = [[0] * self.maze.GetWidth() for i in range(self.maze.GetHeight() - 1)]
```

Рисунок 19. Функция load()

Заключение

Алгоритмы генерации лабиринта играют важную роль в современной жизни. Они используются в самых разнообразных сферах. Есть множество различных алгоритмов. В данной курсовой работе было дано описание алгоритма Уилсона.

Этот алгоритм был разработан еще в 20 веке, и по сей день много используется. Его принцип основан на равновероятном генерировании случайного остовного дерева. Плюсом данного алгоритма является то, что лабиринты всегда получаются связными и запутанными, не похожими друг на друга. Большими минусами являются требовательность к памяти, и долгое время генерации.

В данной работе наглядно описан алгоритм Уилсона, показаны его шаги, а также его блок-схема. Представлена реализация алгоритма с помощью языка программирования Python, и даётся подробное объяснения кода.

Список литературы

1. Джемис Б., Лабиринты для программистов: под редакцией Жаклин К. – США: The Pragmatic Programmers, LLC, 2015 – 286 с. – ISBN-13: 978-1-68050-055-4.
2. Колдаев, В. Д. Основы алгоритмизации и программирования: учебное пособие / В.Д. Колдаев; под ред. проф. Л.Г. Гагариной. — Москва: ФОРУМ: ИНФРА-М, 2022. — 414 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0733-7. - Текст: электронный. - URL: https://znanium.com/catalog/product/1735805 (дата обращения: 14.12.2023). – Режим доступа: по подписке.
3. Скорубский, В. И. Математическая логика: учебник и практикум для вузов / В. И. Скорубский, В. И. Поляков, А. Г. Зыков. — Москва: Издательство Юрайт, 2022. — 211 с. — (Высшее образование). — ISBN 978-5-534-01114-2. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/490017 (дата обращения: 14.12.2023).
4. Трофимов, В. В. Алгоритмизация и программирование : учебник для вузов / В. В. Трофимов, Т. А. Павловская ; под редакцией В. В. Трофимова. — Москва: Издательство Юрайт, 2022. — 137 с. — (Высшее образование). — ISBN 978-5-534-07834-3. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/491215 (дата обращения: 14.12.2023).

Приложение

```
import time
import tkinter as tk

from random import *
from tkinter import messagebox
from PIL import Image, ImageDraw

class Cell():
    def __init__(self, x: int, y: int):
        self.__x: int = x
        self.__y: int = y
        self.visited: bool = False
        self.Left: bool = False
        self.Right: bool = False
        self.Top: bool = False
        self.Bottom: bool = False

class Pair():
    def __init__(self, x: int, y: int):
        self.x = x
        self.y = y

class Maze():
    def __init__(self, width: int, height: int):
        self.__width = width
        self.__height = height

        cell = Cell(x = 0, y = 0)
        self.cells = [[cell] * width for i in range(height)]
        for i in range(0, height):
            for j in range(0, width):
                cell = Cell(x = j, y = i)
                self.cells[i][j] = cell
                self.cells[i][j] = cell

    def GetWidth(self):
        return self.__width

    def GetHeight(self):
        return self.__height

color1 = "#DDA0DD"
color2 = "#EEAEEE"
color3 = "#CD96CD"
```

```
color4 = "red"

class Generator():
    def __init__(self):

        self.maze = Maze(width=20, height=40)
        self.cells = [[]]
        self.wall_ver = [[]]
        self.wall_hor = [[]]

        self.root = tk.Tk()
        self.root.configure(bg=color2)
        self.root.title("Алгоритм Уилсона для генерациия лабиринта")

        self.width_root = 1000
        self.height_root = 650
        self.k = self.width_root / (self.height_root - 50)
        width_screen = self.root.winfo_screenwidth()
        height_screen = self.root.winfo_screenheight()
        self.root.geometry(f"{self.width_root}x{self.height_root}"
                           f"+{int((width_screen - self.width_root) / 2)}"
                           f"+{int((height_screen - self.height_root) / 3)}")
        self.root.resizable(width=False, height=False)

        self.width_wall = 2
        self.width_cell = 0

        if self.maze.GetWidth()/self.maze.GetHeight() > self.k:
            self.width_cell = self.width_root / self.maze.GetWidth()
        else:
            self.width_cell = (self.height_root - 50) / self.maze.GetHeight()

        self.frameMenu = tk.Frame(self.root, height=50, bg=color1)
        self.frameMenu.place(relx=0, relwidth=1, rely=0)

        self.canvasMaze = tk.Canvas(self.root, highlightthickness=0,
highlightbackground="black", bg=color2, width=self.width_cell *
self.maze.GetWidth() - 2, height=self.width_cell * self.maze.GetHeight() - 2)
        self.canvasMaze.place(anchor='center', relx=0.5, y=350)



        self.entry_width = tk.Entry(self.frameMenu, bg=color1, font=("Times New
Roman", 14))
        self.entry_width.place(relx=0.12, rely=0.5, anchor="center", relheight=0.7,
relwidth=0.2)
        self.entry_width.insert(0, " Ширина:")
```

```
        self.entry_width.bind("<FocusIn>", self.focus_in_width)
        self.entry_width.bind("<FocusOut>", self.focus_out_width)

        self.entry_height = tk.Entry(self.frameMenu, bg=color1, font=("Times New
Roman", 14))
        self.entry_height.place(relx=0.33, rely=0.5, anchor="center", relheight=0.7,
relwidth=0.2)
        self.entry_height.insert(0, " Высота:")
        self.entry_height.bind("<FocusIn>", self.focus_in_height)
        self.entry_height.bind("<FocusOut>", self.focus_out_height)

        self.btn_start = tk.Button(self.frameMenu, text="Генерировать",
command=self.start, font=("Times New Roman", 14), bg=color1, cursor="hand2")
        self.btn_start.place(relx=0.8, rely=0.5, anchor="center", relheight=0.7,
relwidth=0.2)

        self.root.mainloop()



def save(self):
    entry_window = tk.Toplevel(self.root)
    entry_window.title("Название файла")
    entry_window.configure(bg=color2)
    width_screen = entry_window.winfo_screenwidth()
    height_screen = entry_window.winfo_screenheight()
    entry_window.geometry(f"{300}x{200}"
                          f"+{int((width_screen - 300) / 2)}"
                          f"+{int((height_screen - 200) / 3)}")

    label = tk.Label(entry_window, font=("Times New Roman", 14), bg=color2)
    label.configure(text="Название файла и картинки")
    label.place(anchor="center", relx=0.5, rely=0.2)

    entry_widget = tk.Entry(entry_window, font=("Times New Roman", 14),
bg=color2)
    entry_widget.place(anchor="center", relx=0.5, rely=0.4, relwidth=0.8)

    def on_submit():

        if self.flag_save:

            file_path = "./data/" + entry_widget.get()
            image_path = "./image/" + entry_widget.get() + ".png"
            entry_window.destroy()
```

```
            s = ""

            with open(file_path, "w") as file:
                file.write(str(self.maze.GetHeight()) + "\n")
                file.write(str(self.maze.GetWidth()) + "\n")
                for i in range(self.maze.GetHeight()):
                    for j in range(self.maze.GetWidth()):
                        s += "1" if self.maze.cells[i][j].Bottom else "0"
                        s += "1" if self.maze.cells[i][j].Right else "0"
                    file.write(s + "\n")
                    s = ""
            time.sleep(2)
            x = self.canvasMaze.winfo_x() + self.root.winfo_rootx() - 1
            y = self.canvasMaze.winfo_y() + self.root.winfo_rooty() - 1
            width = self.canvasMaze.winfo_reqwidth() + 2
            height = self.canvasMaze.winfo_reqheight() + 2
            screenshot = ImageGrab.grab(bbox=(x, y, x + width, y + height))
            screenshot.save(image_path)

        else:
            entry_window.destroy()
            messagebox.showerror("Ошибка", "Лабиринт не сгенерирован")

    submit_button = tk.Button(entry_window, text="Сохранить",
command=on_submit,
                          font=("Times New Roman", 12), bg=color2,
                          cursor="hand2")
    submit_button.place(anchor="center", relx=0.5, rely=0.75)

def load(self):
    entry_window = tk.Toplevel(self.root)
    entry_window.title("Название файла")
    entry_window.configure(bg=color2)
    width_screen = entry_window.winfo_screenwidth()
    height_screen = entry_window.winfo_screenheight()
    entry_window.geometry(f"{300}x{200}"
                          f"+{int((width_screen - 300) / 2)}"
                          f"+{int((height_screen - 200) / 3)}")

    label = tk.Label(entry_window, font=("Times New Roman", 14), bg=color2)
    label.configure(text="Название файла")
    label.place(anchor="center", relx=0.5, rely=0.2)


    entry_widget = tk.Entry(entry_window, font=("Times New Roman", 14),
```

```
bg=color2)
    entry_widget.place(anchor="center", relx=0.5, rely=0.4, relwidth=0.8)



    def on_submit():
        file_path = "./data/" + entry_widget.get()
        self.canvasMaze.delete("all")
        self.flag_save = False

        try:
            with open(file_path, "r") as file:
                height = int(file.readline())
                width = int(file.readline())

                self.maze = Maze(width=width, height=height)
                self.DetermineSize()

                self.cells = [[0] * self.maze.GetWidth() for i in
range(self.maze.GetHeight())]
                self.wall_ver = [[0] * (self.maze.GetWidth() - 1) for i in
range(self.maze.GetHeight())]
                self.wall_hor = [[0] * self.maze.GetWidth() for i in
range(self.maze.GetHeight() - 1)]

                for row in range(self.maze.GetHeight()):
                    for column in range(self.maze.GetWidth()):
                        rect = self.canvasMaze.create_rectangle(column * self.width_cell,
row * self.width_cell,
                                                                (column + 1) * self.width_cell,
                                                                (row + 1) * self.width_cell,
                                                                fill=color2, outline="")
                        self.cells[row][column] = rect

                        if column != self.maze.GetWidth() - 1:
                            line_ver = self.canvasMaze.create_line((column + 1) *
self.width_cell,
                                                                   row * self.width_cell,
                                                                   (column + 1) * self.width_cell,
                                                                   (row + 1) * self.width_cell,
                                                                   width=self.width_wall)
                            self.wall_ver[row][column] = line_ver

                        if row != self.maze.GetHeight() - 1:
                            line_hor = self.canvasMaze.create_line(column * self.width_cell,
                                                                   (row + 1) * self.width_cell,
```

```
                                        (column + 1) * self.width_cell,
                                        (row + 1) * self.width_cell,
                                        width=self.width_wall)
                    self.wall_hor[row][column] = line_hor

            for i in range(height):
                s = file.readline()
                for index, sym in enumerate(s):
                    if index % 2 == 0 and sym == "1" and i != (height - 1):
                        self.canvasMaze.delete(self.wall_hor[i][index//2])

                    if index % 2 == 1 and sym == "1" and (index // 2) != width - 1:
                        self.canvasMaze.delete(self.wall_ver[i][index//2])

            entry_window.destroy()
        except FileNotFoundError:
            messagebox.showerror("Ошибка", f"Файл '{entry_widget.get()}' не
найден.")
            entry_window.destroy()

    # Создаем кнопку для отправки текста
    submit_button = tk.Button(entry_window, text="Загрузить",
command=on_submit,
                              font=("Times New Roman", 12), bg=color2,
                              cursor="hand2")
    submit_button.place(anchor="center", relx=0.5, rely=0.75)


    def start(self):
        flag: bool = False
        width = 0
        height = 0

        try:
            width = int(self.entry_width.get())
            height = int(self.entry_height.get())
            flag = True
        except:
            messagebox.showerror("Ошибка", "Введите корректные параметры!")

        if flag and (width <= 100 and height <= 100):

            self.maze = Maze(width=width, height=height)
            self.DetermineSize()

            self.cells = [[0] * self.maze.GetWidth() for i in range(self.maze.GetHeight())]
```

```
        self.wall_ver = [[0] * (self.maze.GetWidth() - 1) for i in
range(self.maze.GetHeight())]
        self.wall_hor = [[0] * self.maze.GetWidth() for i in
range(self.maze.GetHeight() - 1)]

        for row in range(self.maze.GetHeight()):
            for column in range(self.maze.GetWidth()):
                rect = self.canvasMaze.create_rectangle(column * self.width_cell, row *
self.width_cell, (column + 1) * self.width_cell, (row + 1) * self.width_cell,
fill=color2, outline="")
                self.cells[row][column] = rect

                if column != self.maze.GetWidth() - 1:
                    line_ver = self.canvasMaze.create_line((column + 1) *
self.width_cell, row * self.width_cell, (column + 1) * self.width_cell, (row + 1) *
self.width_cell, width=self.width_wall)
                    self.wall_ver[row][column] = line_ver

                if row != self.maze.GetHeight() - 1:
                    line_hor = self.canvasMaze.create_line(column * self.width_cell,
(row + 1) * self.width_cell, (column + 1) * self.width_cell, (row + 1) *
self.width_cell, width=self.width_wall)
                    self.wall_hor[row][column] = line_hor

        self.Wilson()

    elif flag:
        messagebox.showerror("Большие значения",  "Ширина и высота должны
быть меньше 100!")

    def DetermineSize(self):
        if self.maze.GetWidth()/self.maze.GetHeight() > self.k:
            self.width_cell = self.width_root / self.maze.GetWidth()
        else:
            self.width_cell = (self.height_root - 50) / self.maze.GetHeight()

        self.canvasMaze.configure(width=self.width_cell * self.maze.GetWidth() - 2,
height=self.width_cell * self.maze.GetHeight() - 2, highlightthickness=1)

    def focus_out_width(self, event):
        if self.entry_width.get() == "":
            self.entry_width.insert(0, " Ширина:")

    def focus_in_width(self, event):
        if self.entry_width.get() == " Ширина:":
            self.entry_width.delete(0, 'end')
```

```
    def focus_out_height(self, event):
        if self.entry_height.get() == "":
            self.entry_height.insert(0, " Высота:")

    def focus_in_height(self, event):
        if self.entry_height.get() == " Высота:":
            self.entry_height.delete(0, 'end')

    def RandomCell(self, curr_x: int, curr_y: int):

        curr_x = randint(0, self.maze.GetWidth() - 1)
        curr_y = randint(0, self.maze.GetHeight() - 1)

        while self.maze.cells[curr_y][curr_x].visited:
            curr_x = randint(0, self.maze.GetWidth() - 1)
            curr_y = randint(0, self.maze.GetHeight() - 1)

        return Pair(curr_x, curr_y)

    def FindAdjacentCells(self, curr_x: int, curr_y: int):

        free_cell: list = []

        if curr_x > 0:
            free_cell.append(Pair(curr_x - 1, curr_y))

        if curr_x < (self.maze.GetWidth() - 1):
            free_cell.append(Pair(curr_x + 1, curr_y))

        if curr_y > 0:
            free_cell.append(Pair(curr_x, curr_y - 1))

        if curr_y < (self.maze.GetHeight() - 1):
            free_cell.append(Pair(curr_x, curr_y + 1))

        return free_cell


    def hasCycle(self, path: list, x: int, y: int):

        for i in range(len(path)):
            if path[i].x == x and path[i].y == y:
                for j in range(i + 1, len(path)):
                    self.canvasMaze.itemconfig(self.cells[path[-1].y][path[-1].x],
fill=color2)
```

```
            self.canvasMaze.update()
            path.pop(-1)
        return True

    return False


def OpenWall(self, first: Pair, second: Pair):

    x1: int = first.x
    x2: int = second.x
    y1: int = first.y
    y2: int = second.y

    if x1 == x2:

        if y1 < y2:
            self.maze.cells[y1][x1].Bottom = True
            self.maze.cells[y2][x2].Top = True
            self.canvasMaze.delete(self.wall_hor[y1][x1])

        else:
            self.maze.cells[y1][x1].Top = True
            self.maze.cells[y2][x2].Bottom = True
            self.canvasMaze.delete(self.wall_hor[y2][x2])

    if y1 == y2:
        if x1 < x2:
            self.maze.cells[y1][x1].Right = True
            self.maze.cells[y2][x2].Left = True
            self.canvasMaze.delete(self.wall_ver[y1][x1])

        else:
            self.maze.cells[y1][x1].Left = True
            self.maze.cells[y2][x2].Right = True
            self.canvasMaze.delete(self.wall_ver[y2][x2])


def Wilson(self):

    unvisitedCells: int = self.maze.GetWidth() * self.maze.GetHeight();
    start_x: int = randint(0, self.maze.GetWidth() - 1)
    start_y: int = randint(0, self.maze.GetHeight() - 1)
    self.canvasMaze.itemconfig(self.cells[start_y][start_x], fill=color3)

    self.maze.cells[start_y][start_x].visited = True
```

```
        unvisitedCells -= 1

        while unvisitedCells > 0:

            path: list = []
            pair = self.RandomCell(curr_x=start_x, curr_y=start_y)
            path.append(pair)
            curr_x: int = pair.x
            curr_y: int = pair.y
            self.canvasMaze.itemconfig(self.cells[curr_y][curr_x], fill=color4)

            while not self.maze.cells[curr_y][curr_x].visited:

                adjacent_cells: list = self.FindAdjacentCells(curr_x=curr_x,
curr_y=curr_y)
                rand_cell: int = randint(0, len(adjacent_cells) - 1)
                curr_x = adjacent_cells[rand_cell].x
                curr_y = adjacent_cells[rand_cell].y

                if not self.hasCycle(path, curr_x, curr_y):
                    path.append(Pair(curr_x, curr_y))
                    self.canvasMaze.itemconfig(self.cells[curr_y][curr_x], fill=color4)
                    self.canvasMaze.update()
                    time.sleep(0.05)

            for i in range(len(path)):
                unvisitedCells -= 1;
                self.maze.cells[path[i].y][path[i].x].visited = True
                self.canvasMaze.itemconfig(self.cells[path[i].y][path[i].x], fill=color3)
                if i != len(path) - 1:
                    self.OpenWall(path[i], path[i + 1])

            unvisitedCells += 1

            self.canvasMaze.update()
            time.sleep(0.05)

        messagebox.showinfo("Конец!", "Лабиринт сгенерирован!")

if __name__ == "__main__":
    g = Generator()
```