МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Лабораторная работа №15
по дисциплине: Объектно-ориентированное программирование
Тема: Знакомство с библиотеками языка Python. PyGame.

Выполнил: студент группы ПВ-223
Дмитриев А.А.
Проверил:
Черников С.В.

Белгород 2024 г.

Цель работы: приобретение практических навыков создания приложений на языке Python, быстрая разработка 2d игр.

Задание:
Для выполнения лабораторной работы требуется установить интерпретатор Python версии 3.5+. Выполнить написание программы-сценария в соответствии с вариантом задания (табл. 1). Провести тестирование. Оформить отчет.

Вариант 2:
Лабиринт

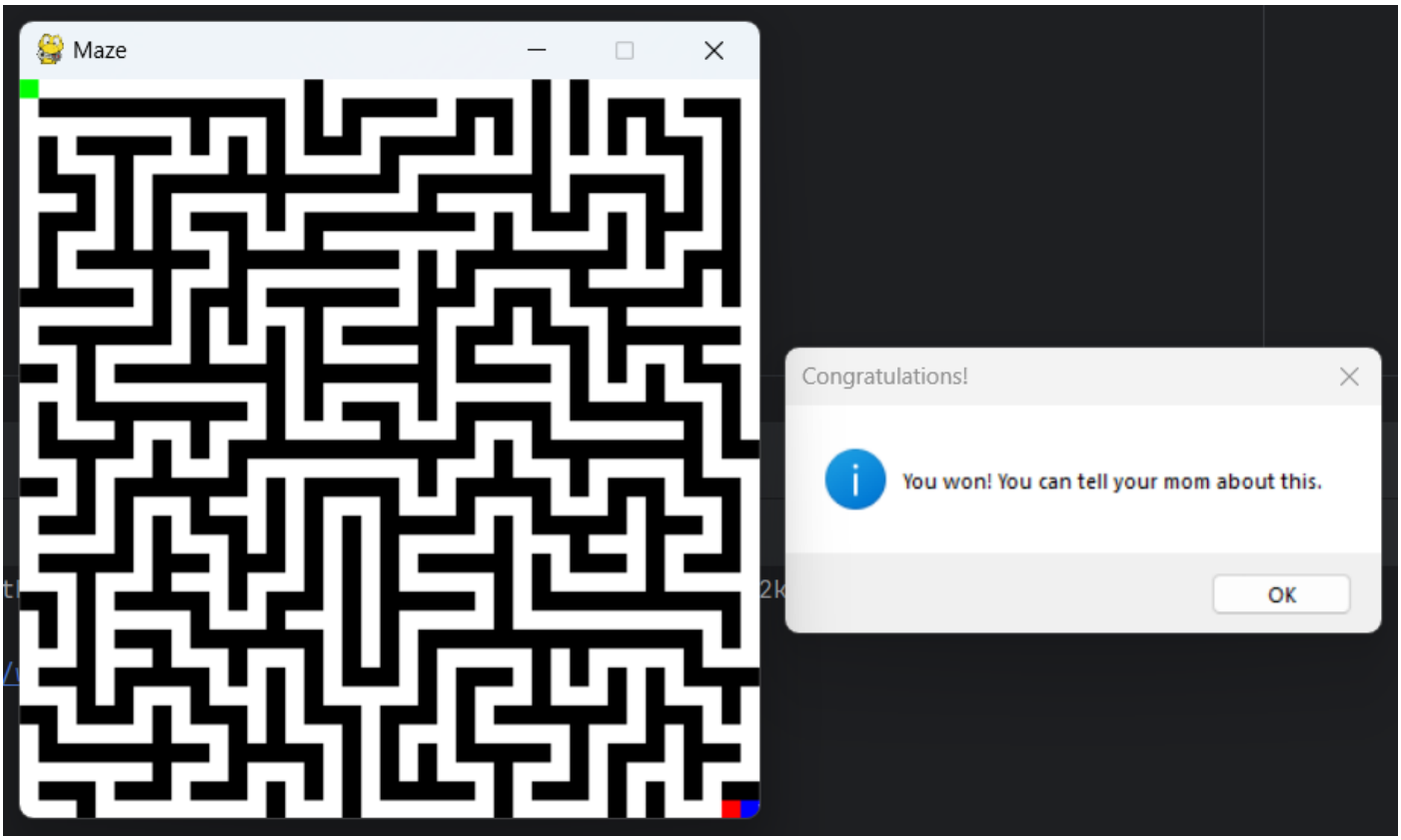
Код (Исходник 1):

```
import pygame
from random import choice

TILE = 10
COLS_FOR_GEN = 20
ROWS_FOR_GEN = 20

INDENT = 10

COLS = 2 * COLS_FOR_GEN - 1
ROWS = 2 * ROWS_FOR_GEN - 1
WIDTH = COLS * TILE
HEIGHT = ROWS * TILE

CELL_COLOR = pygame.Color('white')
CELL_START_COLOR = pygame.Color('green')
CELL_FINISH_COLOR = pygame.Color('blue')
WALL_COLOR = pygame.Color('black')
PLAYER_COLOR = pygame.Color('red')

pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Maze example")
clock = pygame.time.Clock()


class State:
    WALL = 0
    ROAD = 1


class Game:
    def __init__(self):
        self.grid = self.clean_grid()
        self.start_pos = (0, 0)
        self.finish_pos = (COLS - 1, ROWS - 1)
        self.player_pos = (0, 0)
        self.walls = []

    @staticmethod
    def clean_grid():
        return [[Cell(x, y) for x in range(COLS)] for y in range(ROWS)]

    def check_finish(self):
        return self.player_pos == self.finish_pos

    @staticmethod
    def check_pos(x, y):
        return 0 <= x < COLS and 0 <= y < ROWS

    def move_player(self, dx, dy):
        x, y = self.player_pos
        if self.check_pos(x + dx, y + dy) and self.grid[y + dy][x + dx].state ==
State.ROAD:
            self.player_pos = (x + dx, y + dy)

    def gen_maze(self):
        def check_cell_on_visited(_x, _y):
            if _x < 0 or _x >= COLS or _y < 0 or _y >= ROWS:
                return None
            return self.grid[_y][_x]

        def check_neighbours(_x, _y):
            neighbours = []

            top = check_cell_on_visited(_x, _y - 2)
```

```
            right = check_cell_on_visited(_x + 2, _y)
            bottom = check_cell_on_visited(_x, _y + 2)
            left = check_cell_on_visited(_x - 2, _y)

            if top and not top.visited:
                neighbours.append(top)
            if right and not right.visited:
                neighbours.append(right)
            if bottom and not bottom.visited:
                neighbours.append(bottom)
            if left and not left.visited:
                neighbours.append(left)

            return choice(neighbours) if neighbours else None

        self.grid = self.clean_grid()
        stack = []
        current_cell = Cell(*self.start_pos)
        current_cell.visited = True
        while True:
            self.grid[current_cell.y][current_cell.x].state = State.ROAD
            next_cell = check_neighbours(*current_cell.pair())
            if next_cell:
                next_cell.visited = True

                xdx = current_cell.x + (next_cell.x - current_cell.x) // 2
                ydy = current_cell.y + (next_cell.y - current_cell.y) // 2
                if self.check_pos(xdx, ydy):
                    self.grid[ydy][xdx].state = State.ROAD

                current_cell = next_cell
                stack.append(current_cell)
            elif stack:
                current_cell = stack.pop()

            if not stack:
                break  # todo


class Cell:
    def __init__(self, x, y, state=State.WALL):
        self.x = x
        self.y = y
        self.state = state
        self.visited = False

    def pair(self):
        return self.x, self.y


class Window:
    def __init__(self):
        pygame.init()
        self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
        pygame.display.set_caption("Maze")
        self.clock = pygame.time.Clock()

        self.game = Game()
        self.game.gen_maze()

        self.running = True

    def draw_grid(self, grid):
        for row in grid:
            for cell in row:
                if cell.state == State.ROAD:
                    pygame.draw.rect(self.screen, CELL_COLOR,
```

```
                                    (cell.x * TILE, cell.y * TILE, TILE, TILE))
                elif cell.state == State.WALL:
                    pygame.draw.rect(self.screen, WALL_COLOR,
                                     (cell.x * TILE, cell.y * TILE, TILE, TILE))

    def event_move_player(self, event):
        if event.type == pygame.KEYDOWN:
            match event.key:
                case pygame.K_DOWN:
                    self.game.move_player(0, 1)
                case pygame.K_UP:
                    self.game.move_player(0, -1)
                case pygame.K_LEFT:
                    self.game.move_player(-1, 0)
                case pygame.K_RIGHT:
                    self.game.move_player(1, 0)

    def event_quit(self, event):
        if event.type == pygame.QUIT:
            self.running = False

    def check_finish(self):
        if self.game.check_finish():
            self.running = False

            import tkinter
            from tkinter import messagebox
            tkinter.Tk().wm_withdraw()
            messagebox.showinfo("Congratulations!", "You won! You can tell your mom
about this.")

    def run(self):
        while self.running:
            for event in pygame.event.get():
                self.event_move_player(event)
                self.event_quit(event)
                self.check_finish()

            self.screen.fill(WALL_COLOR)

            self.draw_grid(self.game.grid)
            pygame.draw.rect(self.screen, CELL_START_COLOR,
                             (self.game.start_pos[0] * TILE, self.game.start_pos[1] *
TILE, TILE, TILE))
            pygame.draw.rect(self.screen, CELL_FINISH_COLOR,
                             (self.game.finish_pos[0] * TILE, self.game.finish_pos[1] *
TILE, TILE, TILE))
            pygame.draw.rect(self.screen, PLAYER_COLOR,
                             (self.game.player_pos[0] * TILE, self.game.player_pos[1] *
TILE, TILE, TILE))

            pygame.display.flip()


if __name__ == "__main__":
    window = Window()
    window.run()
```

Пример работы:



Вывод: В ходе лабораторной работы приобрели практические навыки создания приложений на языке Python, быстрая разработка 2d игр.