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.

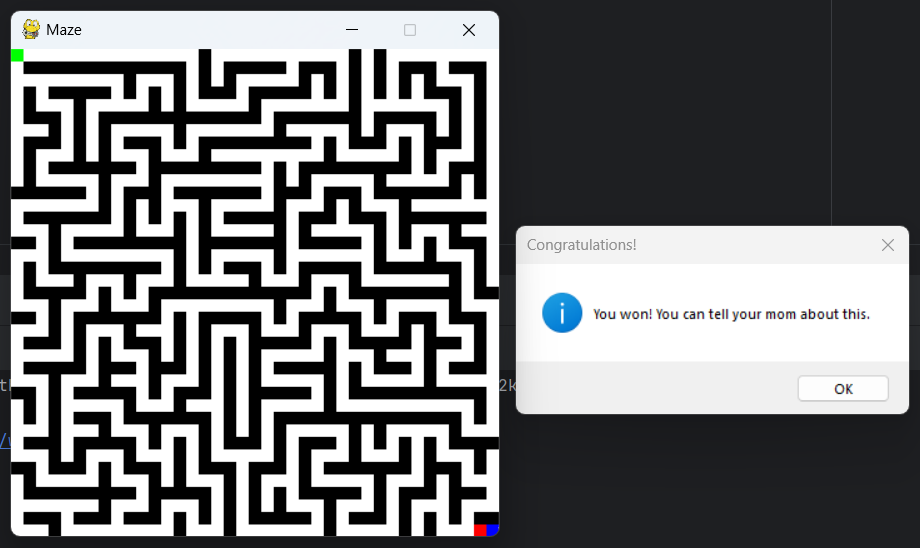
**Вариант 2:**

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.