Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Отчет по лабораторной работе №3

По дисциплине «Системное программное обеспечение».

Выполнили:

Замотаев Егор

Карагужина Гульсария

Мингазов Артур

студенты группы R3336

Преподаватель:

Капитонов Александр Александрович

Санкт-Петербург

2018

***Задание 1 «Лабиринт»***

Используя лекционные материалы, создайте игру «лабиринт». Желательный размер лабиринта – 50 х 50 клеток (можно выбрать свой размер, но не менее 20х20), пользователь может перемещаться вверх, вниз, вправо и влево. Каждый шаг смещает пользователя на одну клетку, если не мешает стена. Вход и выход из лабиринта генерируются случайно при каждом запуске игры. При прохождении лабиринта пользователю выдается информация о затраченном количестве шагов, времени прохождения, предложение завершить игру и начать заново.

***Код программы:***

***Main\_game:***

from pygame import \*

import pygame

from pygame import \*

from blocks import Platform

from random import randrange, randint

from finish import Finish

WIN\_WIDTH = 1010 # Ширина создаваемого окнa

WIN\_HEIGHT = 1010 # Высота

DISPLAY = (WIN\_WIDTH, WIN\_HEIGHT) # Группируем ширину и высоту в одну переменную

BACKGROUND\_COLOR = "#FA8072"

PLATFORM\_WIDTH = 10

PLATFORM\_HEIGHT = 10

MAZE\_SIZE = 50

def neighbours(x\_start, y\_start, maze\_size):

neighbour = []

if x\_start + 2 <= maze\_size - 2:

neighbour.append([x\_start + 2, y\_start])

if y\_start + 2 <= maze\_size - 2:

neighbour.append([x\_start, y\_start+2])

if x\_start - 2 >= 0:

neighbour.append([x\_start - 2, y\_start])

if y\_start - 2 >= 0:

neighbour.append([x\_start, y\_start - 2])

return neighbour

def outneighbour(neighbours, visited):

out = []

for i in range(len(neighbours)):

if neighbours[i] in visited:

out.append(i)

out.reverse()

for i in range(len(out)):

neighbours.pop(out[i])

return neighbours

def maze\_generation():

maze\_size = MAZE\_SIZE \* 2 + 1

level = [[0 for i in range(maze\_size)] for i in range(maze\_size)]

x = randrange(1, maze\_size - 2, 2)

y = randrange(1, maze\_size - 2, 2)

level[x][y] = 1

visited = [[x, y]]

while len(visited) < MAZE\_SIZE \*\* 2:

neighbour = neighbours(x, y, maze\_size)

neighbour = outneighbour(neighbour,visited)

if len(neighbour) == 0:

i = 0

while len(neighbour) == 0:

i += 1

next = visited[len(visited) - i]

x = next[0]

y = next[1]

pas = neighbours(x, y, maze\_size)

neighbour = outneighbour(pas, visited)

if len(neighbour) != 0:

next = randint(0, len(neighbour) - 1)

if neighbour[next][0] - x < 0:

level[x-1][y] = 1

elif neighbour[next][0] - x > 0:

level[x+1][y] = 1

elif neighbour[next][1] - y < 0:

level[x][y-1] = 1

else:

level[x][y+1] = 1

x = neighbour[next][0]

y = neighbour[next][1]

level[x][y] = 1

visited.append([x, y])

return level, visited[len(visited)-randint(1, len(visited)-1)], visited[len(visited)-randint(1, len(visited)-1)]

def main():

ms = 'k'

while True:

pygame.init()

maze = maze\_generation()

screen = pygame.display.set\_mode(DISPLAY)

pygame.display.set\_caption('Maze game')

bg = Surface(DISPLAY)

bg.fill(Color(BACKGROUND\_COLOR))

screen.blit(bg, (0, 0))

hero = NPC(maze[1][0]\*10, maze[1][1]\*10)

finish = Finish(maze[2][0]\*10, maze[2][1]\*10)

entites = pygame.sprite.Group()

platforms = []

entites.add(hero)

entites.add(finish)

level = maze[0]

clock = pygame.time.Clock()

x = y = 0

for row in level:

for col in row:

if col == 0:

pf = Platform(x, y)

entites.add(pf)

platforms.append(pf)

x += PLATFORM\_WIDTH

y += PLATFORM\_HEIGHT

x = 0

stepc = 0

while True:

for i in pygame.event.get():

if i.type == KEYDOWN and i.key == K\_LEFT:

stepc += 1

hero.rect.x -= 10

for p in platforms:

if sprite.collide\_rect(hero, p):

hero.rect.left = p.rect.right

if i.type == KEYDOWN and i.key == K\_RIGHT:

stepc += 1

hero.rect.x += 10

for p in platforms:

if sprite.collide\_rect(hero, p):

hero.rect.right = p.rect.left

if i.type == KEYDOWN and i.key == K\_UP:

stepc += 1

hero.rect.y -= 10

for p in platforms:

if sprite.collide\_rect(hero, p):

hero.rect.top = p.rect.bottom

if i.type == KEYDOWN and i.key == K\_DOWN:

hero.rect.y += 10

stepc += 1

for p in platforms:

if sprite.collide\_rect(hero, p):

hero.rect.bottom = p.rect.top

if i.type == QUIT:

pygame.quit()

exit()

pygame.display.update()

screen.blit(bg, (0, 0))

entites.draw(screen)

clock.tick(60)

ms = finish.collaid(hero, stepc)

if ms == 'kek':

break

pygame.display.update()

if \_\_name\_\_ == '\_\_main\_\_':

main()

***NPC:***

from pygame import \*

MOVE\_SPEED = 10

WIDTH = 10

HEIGHT = 10

COLOR = '#ff0033'

class NPC(sprite.Sprite):

def \_\_init\_\_(self, x, y):

sprite.Sprite.\_\_init\_\_(self)

self.xvel = 0

self.yvel = 0

self.startX = x

self.startY = y

self.rect = Rect(x, y, WIDTH, HEIGHT)

self.image = Surface((WIDTH, HEIGHT))

self.image.fill(Color(COLOR))

***Finish:***

from pygame import \*

PLATFORM\_WIDTH = 10

PLATFORM\_HEIGHT = 10

PLATFORM\_COLOR = "#ffffff"

WIN\_WIDTH = 1010 # Ширина создаваемого окнa

WIN\_HEIGHT = 1010 # Высота

DISPLAY = (WIN\_WIDTH, WIN\_HEIGHT) # Группируем ширину и высоту в одну переменную

BACKGROUND\_COLOR = "#FA8072"

BUTTON\_COLOR = "#412227"

class Finish(sprite.Sprite):

def \_\_init\_\_(self, x, y):

sprite.Sprite.\_\_init\_\_(self)

self.image = Surface((PLATFORM\_WIDTH, PLATFORM\_HEIGHT))

self.image.fill(Color(PLATFORM\_COLOR))

self.rect = Rect(x, y, PLATFORM\_WIDTH, PLATFORM\_HEIGHT)

def collaid(self, hero, stepc):

ms = 'k'

if sprite.collide\_rect(self, hero):

sc = display.set\_mode(DISPLAY)

bg = Surface(DISPLAY)

bg.fill(Color(BACKGROUND\_COLOR))

sc.blit(bg, (0, 0))

f1 = font.Font(None, 40)

text1 = f1.render('GAME OVER', 1, (64, 34, 39))

text2 = f1.render('number of steps:', 1, (64, 34, 39))

text3 = f1.render(str(stepc), 1, (64, 34, 39))

sc.blit(text1, (427, 450))

sc.blit(text2, (390, 500))

sc.blit(text3, (630, 500))

x = 422

y = 800

width = 165

height = 45

cur = mouse.get\_pos()

click = mouse.get\_pressed()

if x + width > cur[0] > x and y + height > cur[1] > y:

draw.rect(sc, (0, 0, 0), (x, y, width, height))

text4 = f1.render('NEW GAME', 1, (250, 128, 114))

sc.blit(text4, (427, 810))

if click[0] == 1:

ms = 'kek'

else:

draw.rect(sc, (65, 34, 39), (x, y, width, height))

text4 = f1.render('NEW GAME', 1, (250, 128, 114))

sc.blit(text4, (427, 810))

return ms

***Block:***

from pygame import \*

PLATFORM\_WIDTH = 10

PLATFORM\_HEIGHT = 10

PLATFORM\_COLOR = "#412227"

class Platform(sprite.Sprite):

def \_\_init\_\_(self, x, y):

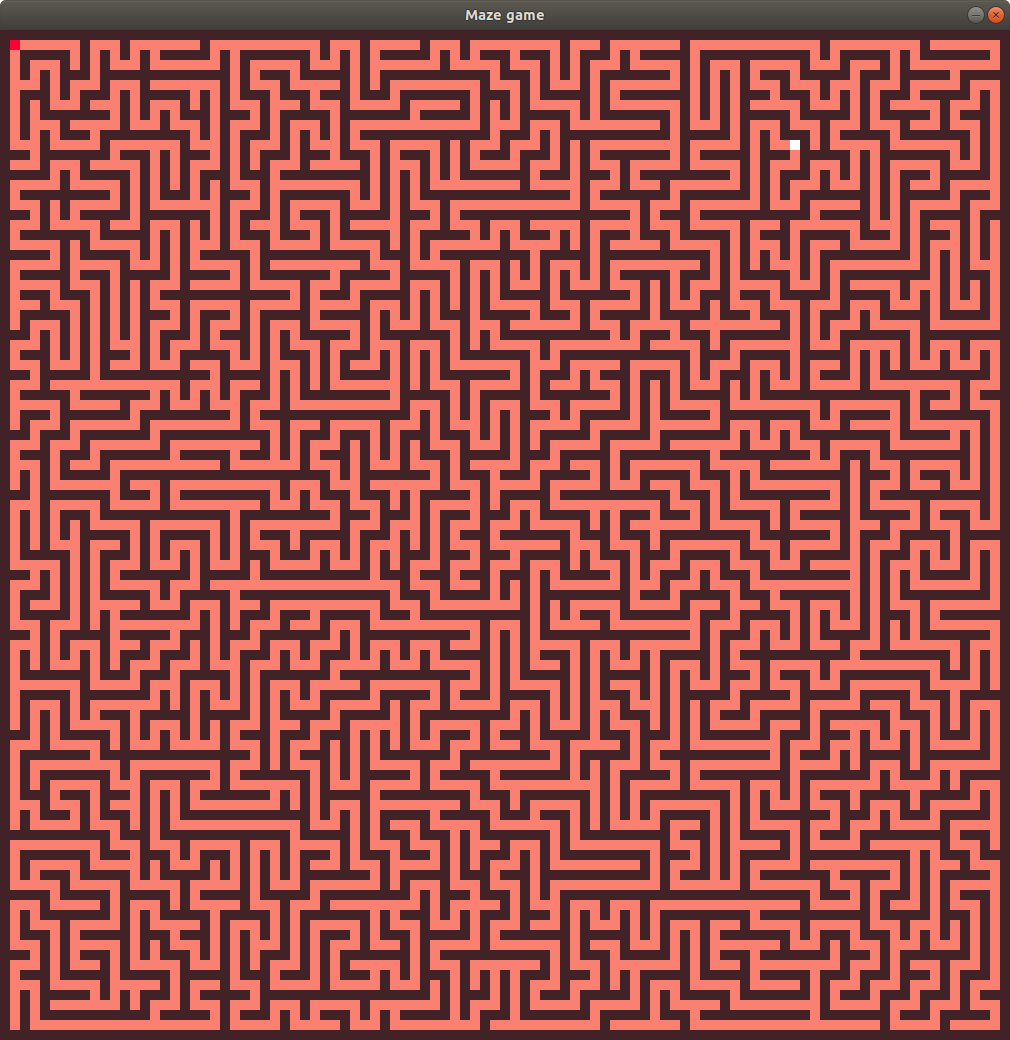
sprite.Sprite.\_\_init\_\_(self)

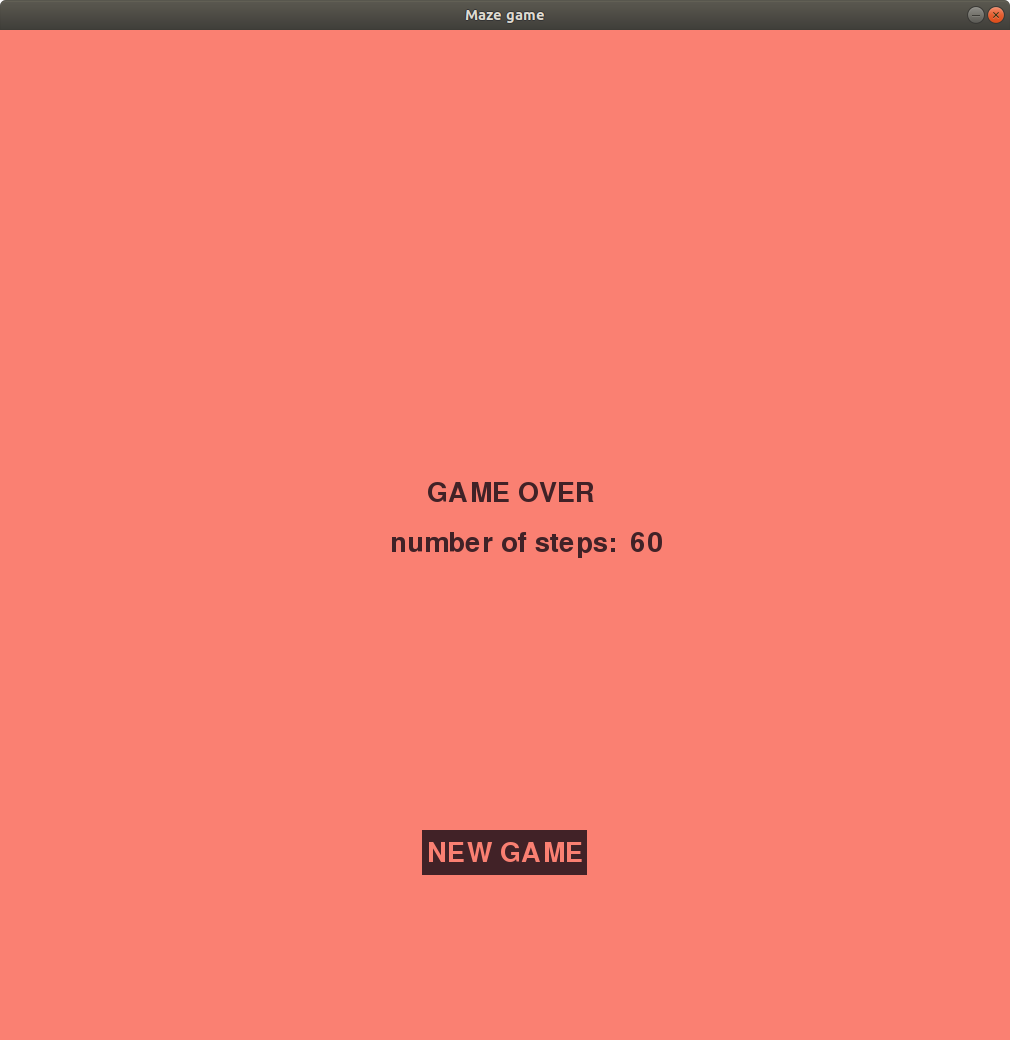
self.image = Surface((PLATFORM\_WIDTH, PLATFORM\_HEIGHT))

self.image.fill(Color(PLATFORM\_COLOR))

self.rect = Rect(x, y, PLATFORM\_WIDTH, PLATFORM\_HEIGHT)

***Результат работы программы:***

******

******

***Задание 2 «Поиск выхода»***

Напишите программу, которая будет автоматически передвигать пользователя из задания 1 и искать выход из лабиринта.

***Код программы:***

import sys

sys.path.insert(0, '/home/egor/Documents/PythonLabs/Lab3/3.1')

sys.setrecursionlimit(3500)

import main\_game

import pygame

from pygame import \*

FINISH\_COLOR = '#0000ff'

NPC\_COLOR = '#ff0033'

PLATFORM\_COLOR = "#412227"

PLATFORM\_COLOR2 = "#FFFFFF"

MAZE\_SIZE = 10

WIN\_WIDTH = 1010 # Ширина создаваемого окнa

WIN\_HEIGHT = 1010 # Высота

DISPLAY = (WIN\_WIDTH, WIN\_HEIGHT) # Группируем ширину и высоту в одну переменную

BACKGROUND\_COLOR = "#FA8072"

PLATFORM\_WIDTH = 10

PLATFORM\_HEIGHT = 10

maze = main\_game.maze\_generation()

level = maze[0]

door = (maze[1][0], maze[1][1])

door1 = maze[1][0]

door2 = maze[1][1]

print(door1, door2)

out = (maze[2][0], maze[2][1])

out1 = maze[2][0]

out2 = maze[2][1]

print(out)

print(level)

for i in range(len(level)):

for j in range(len(level)):

if level[i][j] == 0:

level[i][j] = 1

else:

level[i][j] = 0

def search\_path(data, x, y, short\_path={}, full\_path={},

count=0): # data-лабиринт; х,у - точка, где мы сейчас находимся

# short\_path - {key=точка куда идём: value=откуда идём}

# full\_path - {key=точка : value=кол-во шагов до точки}

# count - номер шага

full\_path[(x, y)] = count # Записываем точку и сколько шагов до неё на данном этапе

if x == out1 and y == out2: # Точка выхода из лабиринта

return full\_path, short\_path

walks = [(-1, 0), (0, 1), (1, 0), (0, -1)] # up,left, down, right,

for walk\_X, walk\_Y in walks:

if data[x + walk\_X][y + walk\_Y] == 0 and (0 < x + walk\_X < 100 and 0 < y + walk\_Y < 100): # Если ячейка свободна

# и мы не вышли за границы лабиринта

check = full\_path.get((x + walk\_X, y + walk\_Y)) # Смотрим на точку, куда хотим пойти и сколько до неё шагов

# Если check=None, значит в точке ещё не были

if check != None and check > count: # Если мы были уже в этой точке и расстояние до неё больше, чем номер шага

# на данном этапе

full\_path[

(x + walk\_X, y + walk\_Y)] = count # Перезаписываем full\_path, т.к нашли более короткую дистанцию

short\_path[(x + walk\_X, y + walk\_Y)] = (x, y) # Пепрезаписываем short\_path, потому что нашли точку, из

# которой в данную можно попасть короче

search\_path(data, x + walk\_X, y + walk\_Y, short\_path, full\_path,

count + 1) # Увеличиваем шаг и запускаем

# рекурсивно функцию

else:

if (x + walk\_X, y + walk\_Y) not in full\_path.keys(): # Если в токе, куда собираемся пойти еще не были

short\_path[(x + walk\_X, y + walk\_Y)] = (x, y) # записываем {куда идём:откуда идём}

search\_path(data, x + walk\_X, y + walk\_Y, short\_path, full\_path, count + 1) # запускаем рекурсию

# с шагом +1

return full\_path, short\_path

def short\_path(data, path=[], start=door, end=out): # Сюда прилетает short\_path из search\_path, когда нашли выход

# data=short\_path, start - координата точки входа, end - выхода

# path - короткий путь в виде списка координат из лабиринта

*"""Здесь мы рекурсивно пробегаемся из конечной точки в начальную, восстанавливая путь по лабиринту:*

*берём {точка, куда пришли(допустим А) : точка откуда пришли (в точку А) - Б}"""*

if len(path) == 0:

path.append(end)

path.append(data[end])

if data[end] == start:

return path

else:

short\_path(data, path, start, data[end])

return path

p = search\_path(level, door1, door2)

print(p)

if p is None:

print('No Exit!!!')

else:

short = short\_path(p[1])

short.reverse()

for walk in short:

level[walk[0]][walk[1]] = 3 # Здесь мы просто указываем в графе путь к выходу цифрой 3

for see in level:

print(see)

pygame.init()

screen = pygame.display.set\_mode(DISPLAY)

bg = Surface(DISPLAY)

bg.fill(Color(BACKGROUND\_COLOR))

screen.blit(bg, (0, 0))

platforms = []

x = y = 0

for row in level:

for col in row:

if col == 1:

pf = Surface((PLATFORM\_WIDTH, PLATFORM\_HEIGHT))

pf.fill(Color(PLATFORM\_COLOR))

screen.blit(pf, (x, y))

if col == 3:

pf = Surface((PLATFORM\_WIDTH, PLATFORM\_HEIGHT))

pf.fill(Color(PLATFORM\_COLOR2))

screen.blit(pf, (x, y))

x += PLATFORM\_WIDTH

y += PLATFORM\_HEIGHT

x = 0

pf = Surface((PLATFORM\_WIDTH, PLATFORM\_HEIGHT))

pf.fill(Color(NPC\_COLOR))

screen.blit(pf, (door2\*10, door1\*10))

pf = Surface((PLATFORM\_WIDTH, PLATFORM\_HEIGHT))

pf.fill(Color(FINISH\_COLOR))

screen.blit(pf, (out2\*10, out1\*10))

while True:

pygame.display.update()

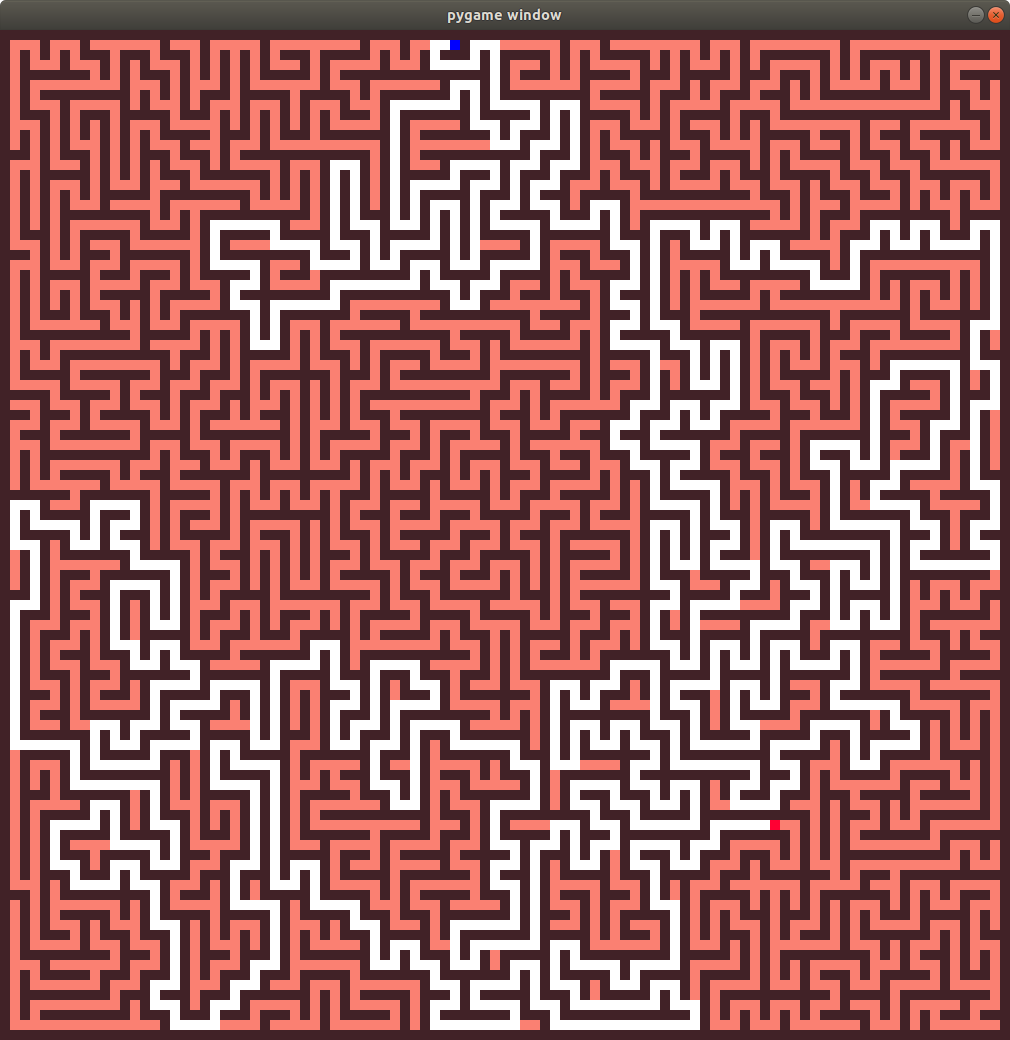
for i in pygame.event.get():

if i.type == QUIT:

pygame.quit()

exit()

***Результат работы программы:***

******

***Задание 3 «ТАУ»***

Напишите свои алгоритмы для вычисления управляемости, наблюдаемости и ранга матрицы

***Код программы:***

import numpy as np

def contr\_check(A, control):

r\_contr = rank(control)

if size\_m(A) == r\_contr:

print("Система управляема (ранг матрицы управляемости ", r\_contr, ")")

else:

print("система не управляема, ранг матрицы управляемости ", r\_contr)

def obs\_check(A, obs):

r\_obs = rank(obs)

if size\_m(A) == r\_obs:

print("Система наблюдаема (Ранг матрицы наблюдаемости равен", r\_obs, ")")

else:

print("Система не наблюдаема, ранг равен матрицы наблюдаемости равен ", r\_obs)

def size\_n(matrix):

n = len(matrix[0])

return n

def size\_m(matrix):

m = len(matrix)

return m

def rank(matrix, n=0):

def check(matrix):

list = []

for i in range(0, size\_m(matrix)):

flag = 0

for j in range(0, size\_n(matrix)):

if matrix[i][j] != 0:

flag = 1

if flag == 0:

list.append(i)

step = 0

for i in list:

del matrix[i - step]

step += 1

list = []

for j in range(0, size\_n(matrix)):

flag = 0

for i in range(0, size\_m(matrix)):

if matrix[i][j] != 0:

flag = 1

if flag == 0:

list.append(j)

step = 0

for j in list:

for i in range(0, size\_m(matrix)):

del matrix[i][j-step]

step += 1

return matrix

def swap(matrix, i):

k = 0

if i == size\_n(matrix) or i == size\_m(matrix):

pass

else:

for step in range(0, size\_n(matrix) - i-1):

if matrix[i][i] == 0:

matrix[i], matrix[i + step] = matrix[i + step], matrix[i]

else:

k = 1

return matrix, k

ch\_matr = check(matrix)

f\_matr, k = swap(ch\_matr, n)

minim = size\_m(f\_matr) if (size\_m(f\_matr)) <= size\_n(f\_matr) else size\_n(f\_matr)

global r

r = size\_m(matrix)

if (n < minim) or k != 0:

for i in range(n + 1, size\_m(f\_matr)):

koef = -f\_matr[i][n] / f\_matr[n][n]

for j in range(n, size\_n(f\_matr)):

f\_matr[i][j] = koef \* f\_matr[n][j] + f\_matr[i][j] #

rank(f\_matr, n + 1)

return r

if \_\_name\_\_ == '\_\_main\_\_':

matrix = [[1, 2, 3, 4], [1, 2, 3, 4]]

np\_matrix = np.array(matrix)

print(rank(matrix))

print(np.linalg.matrix\_rank(np\_matrix))

print(contr\_check(matrix, matrix))

***Результат***

 ***работы программы:***