**Лабораторная работа №3. Декомпозиция задачи поиска выхода из лабиринта в ООП парадигме.**

## Цель: Реализовать в парадигме ООП методы поиска в глубину и ширину для поиска кратчайшего пути для выхода из лабиринта.

**Ход работы**

В данной работе представлен код, в котором реализованы следующие классы:

* *Maze*: считывание из файла, хранение, информация о содержимом клетки по координатам, информация о координатах сокровища.
* *Pathfinder*: информация о личности, двигаться в 8 направлениях: S, N, E, W, E, NW, NE, SW, SE, записывать в журнал (кортеж или список) свой маршрут в формате coord = [x, y], только положительные (проверять перед шагом)
* *Tracer*: набор алгоритмов (в нашем случае - в глубину и ширину). Принимает сам лабиринт и искателя, ведет его по лабиринту. Искатель сам в это время записывает свой маршрут.
* *Trace\_maker*: умеет из лабиринта и искателя построить маршрут из точек и запятых и сохранить его в файл.

**Ход работы**

*from* collections *import* deque

POSSIBLE\_WAYS = ('N', 'S', 'W', 'E')

FILE\_TO\_READ = 'maze-for-u.txt'

*class* Maze:

    maze = []

    path\_to\_exit = []

    current\_path = []

    tresure\_is\_found = False

    start\_point = [0, 1]

    x = int(*input*("коорд х: "))

    y = int(*input*("коорд y: "))

    coord\_point = [y, x]

    t\_coord = [118, 141]

    tresure\_is\_here = (y, x)

*def* read(self):

*for* line *in* *open*(FILE\_TO\_READ, "r"):

            self.maze.append(line[:-1])

*for* i *in* *range*(*len*(self.maze)\**len*(self.maze[0])):

            self.path\_to\_exit.append(' ')

*class* Pathfinder:

*def* step(self,coord, direction):

*if* direction == 'N':

*return* self.step\_N(coord)

*elif* direction == 'S':

*return* self.step\_S(coord)

*elif* direction == 'E':

*return* self.step\_E(coord)

*elif* direction == 'W':

*return* self.step\_W(coord)

*def* step\_N(self,coord):

*return* [coord[0]-1, coord[1]]

*def* step\_E(self,coord):

*return* [coord[0], coord[1]+1]

*def* step\_S(self,coord):

*return* [coord[0]+1, coord[1]]

*def* step\_W(self,coord):

*return* [coord[0], coord[1]-1]

*def* is\_coord\_in\_maze(self, maze, coord):

        len\_y, len\_x = *len*(maze), *len*(maze[0])

*if* coord[0] < 0 or coord[0] > len\_x:

*return* False

*if* coord[1] < 0 or coord[1] > len\_y:

*return* False

*return* True

*def* is\_tresure\_clean(self,coord, tresure\_is\_here ):

*if* coord[0] == tresure\_is\_here[0] and coord[1] == tresure\_is\_here[1]:

*print*('tresure\_is\_here!')

*return* True

*else*:

*return* False

    @staticmethod

*def* split(s):

*return* [char *for* char *in* s]

*def* is\_path\_clean(self,maze, coord):

*if* maze[coord[0]][coord[1]] == '#':

*return* False

*return* True

*def* cut\_way\_back(self,direction):

*if* direction == "N":

*return* ("N", "E", "W")

*if* direction == "S":

*return* ("S", "E", "W")

*if* direction == "E":

*return* ("N", "E", "S")

*if* direction == "W":

*return* ("N", "S", "W")

*def* find\_a\_way(self, maze, coord, possible\_ways, path\_to\_exit, current\_path, tresure\_is\_found, tresure\_is\_here):

*if* not self.is\_coord\_in\_maze(maze, coord):

*print*('вне зоны!')

*return*

*if* *len*(current\_path) > *len*(path\_to\_exit):

*print*('слишком длинный путь')

*return*

*for* direction *in* possible\_ways:

*if* self.is\_tresure\_clean(self.step(coord, direction), tresure\_is\_here) or tresure\_is\_found:

                tresure\_is\_found = True

                b = self.split(maze[coord[0]])

                b[coord[1]] = "."

                maze[coord[0]] = ''.join(b)

*return*

*else*:

*if* self.is\_path\_clean(maze, self.step(coord, direction)):

                    current\_path.append(direction)

                    self.find\_a\_way(maze, self.step(coord, direction),

                    self.cut\_way\_back(direction), path\_to\_exit, current\_path, tresure\_is\_found, tresure\_is\_here)

                    current\_path.pop()

*if* tresure\_is\_found == False:

                        b = self.split(maze[coord[0]])

                        b[coord[1]] = "."

                        maze[coord[0]] = ''.join(b)

*return*

*class* Tracer:

*def* bfs(maze, coord, t):

*try*:

            n, m = *len*(maze), *len*(maze[0])

            INF = 0

            p = [[None]\*m *for* \_ *in* *range*(n)]

            d = [[INF]\*m *for* \_ *in* *range*(n)]

            used = [[False]\*m *for* \_ *in* *range*(n)]

            queue = deque()

            delta = [(0, -1), (0, 1), (1, 0), (-1, 0)]

            d[coord[0]][coord[1]] = 0

            used[coord[0]][coord[1]] = True

            queue.append(coord)

*while* *len*(queue) != 0:

                x, y = queue.popleft()

*for* dx, dy *in* delta:

                    nx, ny = x + dx, y + dy

*if* 0 <= nx < n and 0 <= ny < m and not used[nx][ny] and maze[nx][ny] != "#":

                        d[nx][ny] = d[x][y] + 1

                        p[nx][ny] = (x, y)

                        used[nx][ny] = True

                        queue.append((nx, ny))

*print*(d[t[0]][t[1]])

            cur = t

            path = []

*while* cur is not None:

                path.append(cur)

                cur = p[cur[0]][cur[1]]

            path.reverse()

            count = 0

*for* i *in* path:

                ys = path[count][1]

                xs = path[count][0]

                b = Pathfinder.split(maze[xs])

                b[ys] = ","

                maze[xs] = ''.join(b)

                count += 1

*except* IndexError:

*pass*

*class* Trace\_maker:

*def* write(maze,p,x,y):

        b = Pathfinder.split(maze[y])

        b[x] = "\*"

        maze[y] = ''.join(b)

*with* *open*("maze-for-me-done\_oop.txt", "w") *as* txt\_file:

*for* line *in* maze:

                txt\_file.write(" ".join(line) + "\n")

*if* \_\_name\_\_ == "\_\_main\_\_":

    m = Maze()

    m.read()

    p = Pathfinder()

    Pathfinder.find\_a\_way(p,m.maze, m.start\_point, POSSIBLE\_WAYS, m.path\_to\_exit, m.current\_path, m.tresure\_is\_found, m.tresure\_is\_here)

    Tracer.bfs(m.maze, m.coord\_point, m.t\_coord)

    Trace\_maker.write(m.maze,p,m.x,m.y)

На рисунках 1-3 приведен лабиринт и найденный путь.

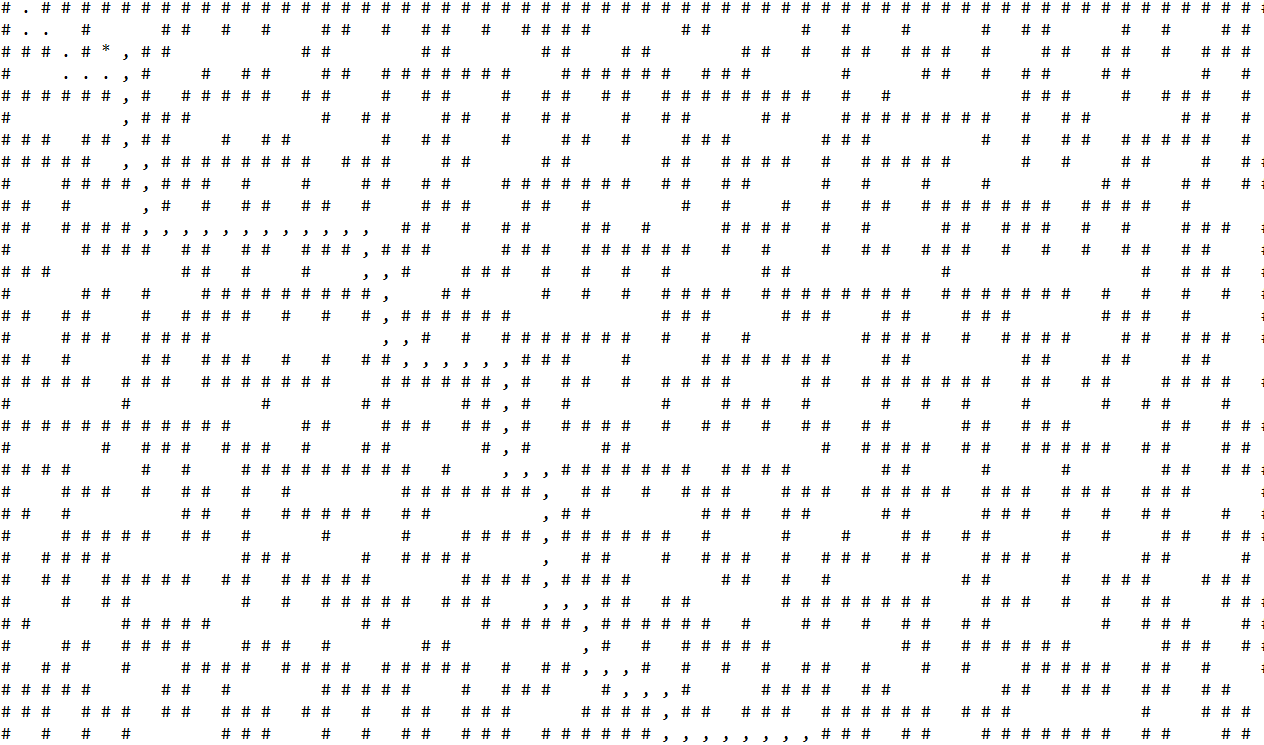


Рисунок 1

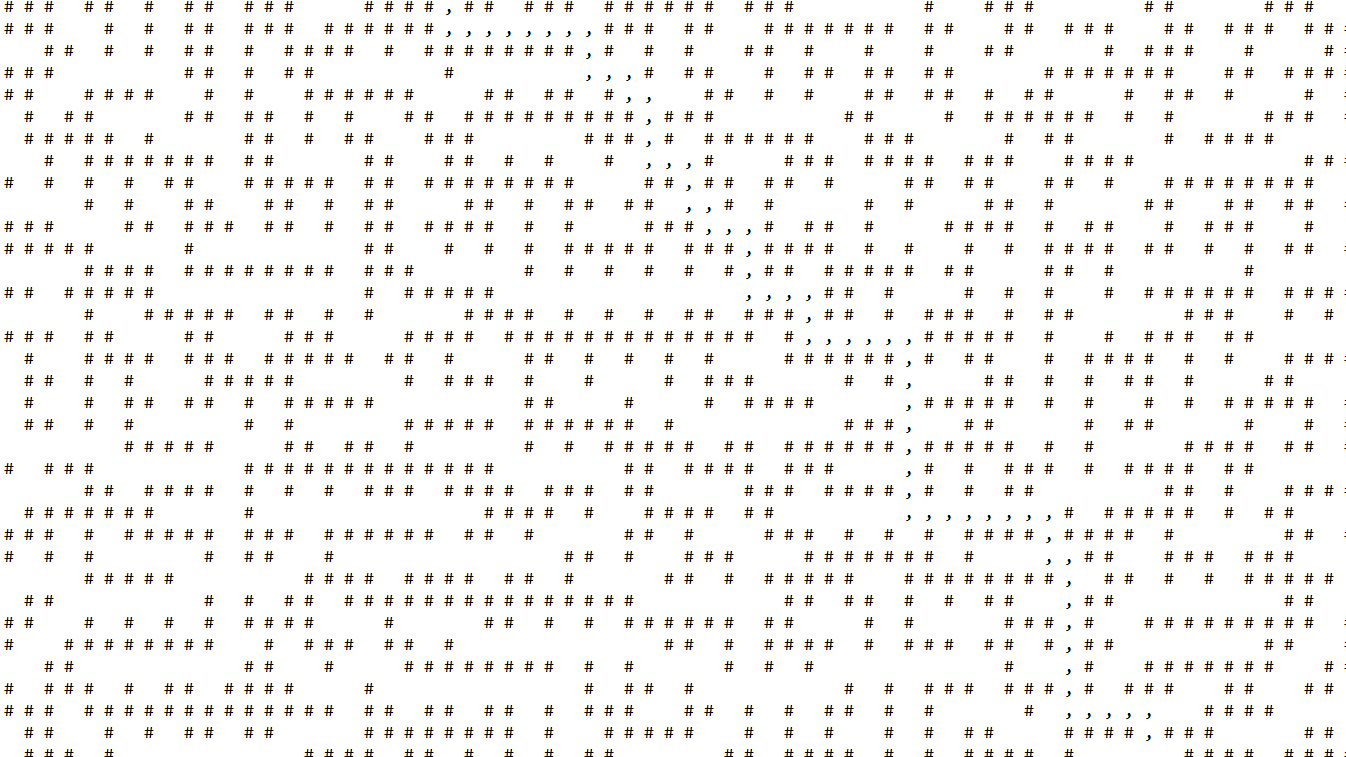


Рисунок 2



Рисунок 3

**Вывод:** В лабораторной работе рассмотрены и реализованы алгоритмы поиска в глубину и ширину в парадигме ООП, для кратчайшего нахождения пути в лабиринте.