

# Algorytmy i struktury danych

Dokumentacja zadania 14: Znajdź drogę przez labirynt zdefiniowany przez macierz sąsiedztwa  
zawartą w pliku

Daniel Kosytorz, grupa 1A

14 stycznia 2021

# Spis treści

<b>1</b>	<b>Opis problemu przedstawionego w zadaniu</b>	<b>3</b>
<b>2</b>	<b>Model matematyczny zadania</b>	<b>3</b>
2.1	Opis modelu matematycznego . . . . .	3
2.2	Przykładowe rozwiązanie . . . . .	4
<b>3</b>	<b>Algorytm</b>	<b>5</b>
3.1	Pseudokod . . . . .	5
3.2	Schemat blokowy . . . . .	6
<b>4</b>	<b>Implementacja</b>	<b>7</b>
4.1	Zasada działania programu . . . . .	7
4.2	Dodatkowe informacje . . . . .	10
4.3	Testy . . . . .	10
4.4	Kod źródłowy - main.py . . . . .	16

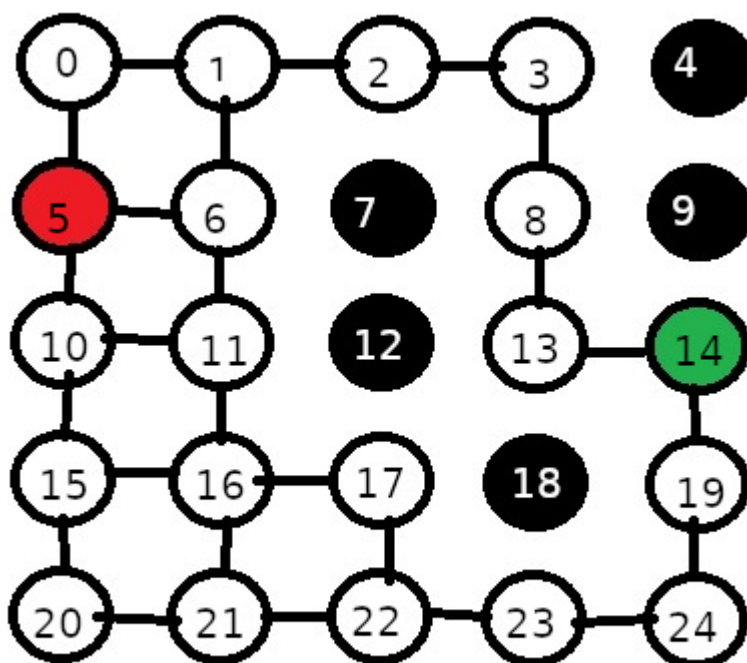
# 1 Opis problemu przedstawionego w zadaniu

Zadanie polega na znalezieniu drogi w labiryncie zdefiniowanym przez macierz sąsiedztwa. W przypadku więcej niż jednej drogi program ma znaleźć najkrótszą drogę.

## 2 Model matematyczny zadania

### 2.1 Opis modelu matematycznego

Labirynt jest przedstawiony jako graf a droga w nim to pojedyncze wierzchołki tego grafu. Koszty połączeń między wierzchołkami są wszędzie jednakowe i wynoszą 1. Jeżeli sąsiadujące wierzchołki są ze sobą bezpośrednio połączone to znaczy, że istnieje między nimi droga. Jeśli sąsiadujące wierzchołki nie są połączone oznacza to, że jeden z nich stanowi ścianę w labiryncie. Jest to przedstawione na poniższym rysunku:



Rysunek 1: Labirynt jako graf, czarne wierzchołki - ściana, czerwony - start, zielony - koniec

Do znalezienia najkrótszej drogi w labiryncie z punktu startowego do punktu końcowego użyj algorytmu grafowego A\*. Algorytm ten wykorzystuje pewną heurystykę. Służy ona do oszacowania kosztu nieznanego jeszcze drogi z pewnego wierzchołka pośredniego do celu. Dzięki temu algorytm w pierwszej kolejności analizuje wierzchołki pośrednie, które najlepiej rokuja.

Na każdym kroku minimalizowana jest wartość funkcji

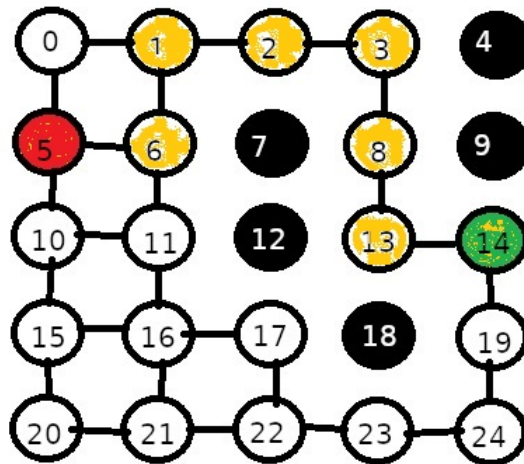
$$f(v) = g(v) + h(v)$$

gdzie  $f$  jest oszacowaniem kosztu całej drogi z  $s$  do celu, która jest sumą rzeczywistą kosztu  $g$  dotarcia do węzła pośredniego  $v$  i szacowanego kosztu  $h$  dostania się z  $v$  do celu.

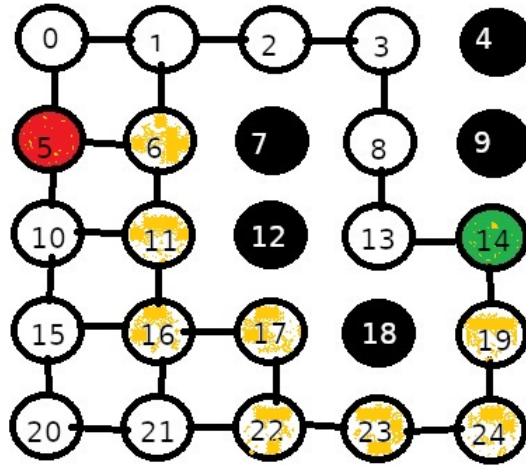
Wierzchołki w grafie interpretuje jako punkty na płaszczyźnie, więc funkcja  $h$  jest odległością euklidesową pomiędzy wierzchołkami w linii prostej. Możemy poruszać się tylko w czterech kierunkach, więc dobrym wyborem będzie metryka Manhattan.

## 2.2 Przykładowe rozwiązanie

Poniżej przedstawione są dwa rozwiązania tego labiryntu. Pierwszy rysunek zawiera najkrótszą trasę, której długość wynosi 7, a drugi rysunek przedstawia drogę, której długość wynosi 9.



Rysunek 2: Pomarańczowy kolor oznacza drogę - długość 7



Rysunek 3: Pomarańczowy kolor oznacza drogę - długość 9

## 3 Algorytm

### 3.1 Pseudokod

**Data:** Graf z wierzchołkami, wierzchołek startowy, wierzchołek końcowy

$Q, S$  - puste zbiory wierzchołków;

$n$  = rozmiar grafu;

$g, f$  = tablice o rozmiarze  $n$  wypełnione wartościami nieskończonymi;

$pred$  = tablica o rozmiarze  $n$  wypełniona zerami;

Dodaj startowy wierzchołek do  $Q$ ;

**while**  $Q$  nie jest pusty **do**

$v$  = wierzchołek ze zbioru  $Q$  o najmniejszym  $f(v)$ ;

**if**  $v$  jest wierzchołkiem końcowym **then**

        Zakończ działanie algorytmu;

**end**

    Dla każdego sąsiada  $w$  wierzchołka  $v$ :

$temp_g = g[v] + 1$ ;

**if**  $w$  znajduje się w  $Q$  i  $w$  nie znajduje się w  $S$  **then**

        Dodaj  $w$  do  $Q$ ;

$g[w] = temp_g$ ;

$f[w] = h(w, \text{wierzchołek końcowy}) + g[w]$ ;

$pred[w] = v$ ;

**end**

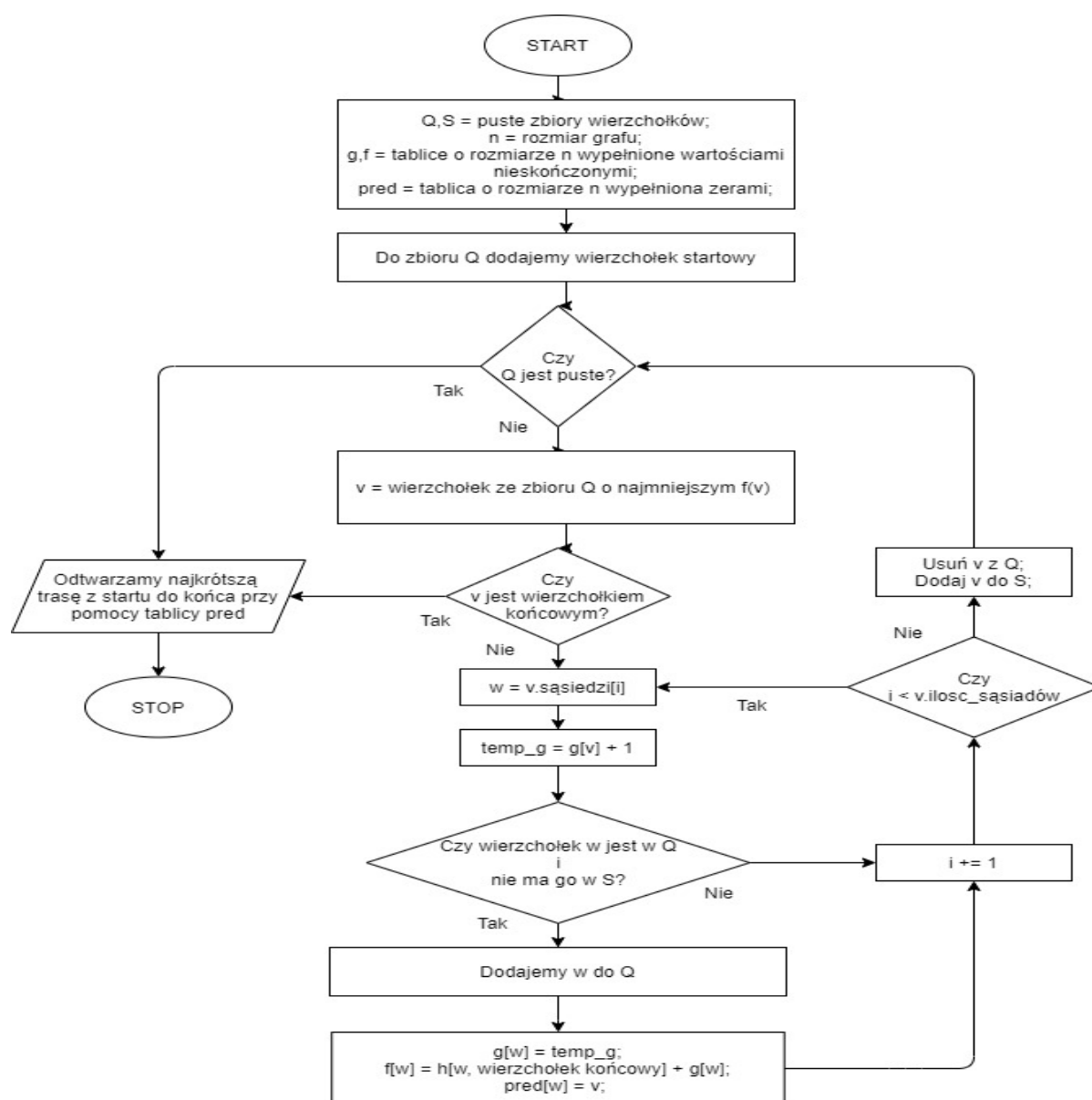
    Usuń wierzchołek  $v$  z  $Q$ ;

    Dodaj wierzchołek  $v$  do  $S$ ;

**end**

Najkrótszą drogę z startu do końca odtwarzamy z tablicy  $pred$ .

## 3.2 Schemat blokowy



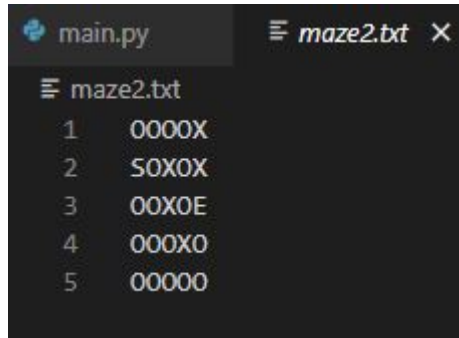
Rysunek 4: Schemat blokowy Algorytmu A\*

## 4 Implementacja

### 4.1 Zasada działania programu

Program jest wykonany z użyciem modułu pygame. Służy to lepszemu zwizualizowaniu labiryntu jak i jego rozwiązania.

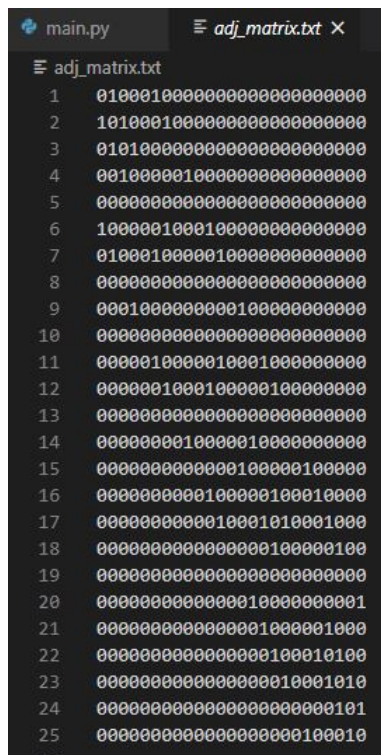
Układ labiryntu znajduje się w pliku maze[i].txt i wygląda np. następująco:



```
main.py  maze2.txt X
maze2.txt
1  0000X
2  50X0X
3  00X0E
4  000X0
5  00000
```

Rysunek 5: Plik z układem labiryntu, gdzie O - puste pole, S - start, E - koniec, X - ściana

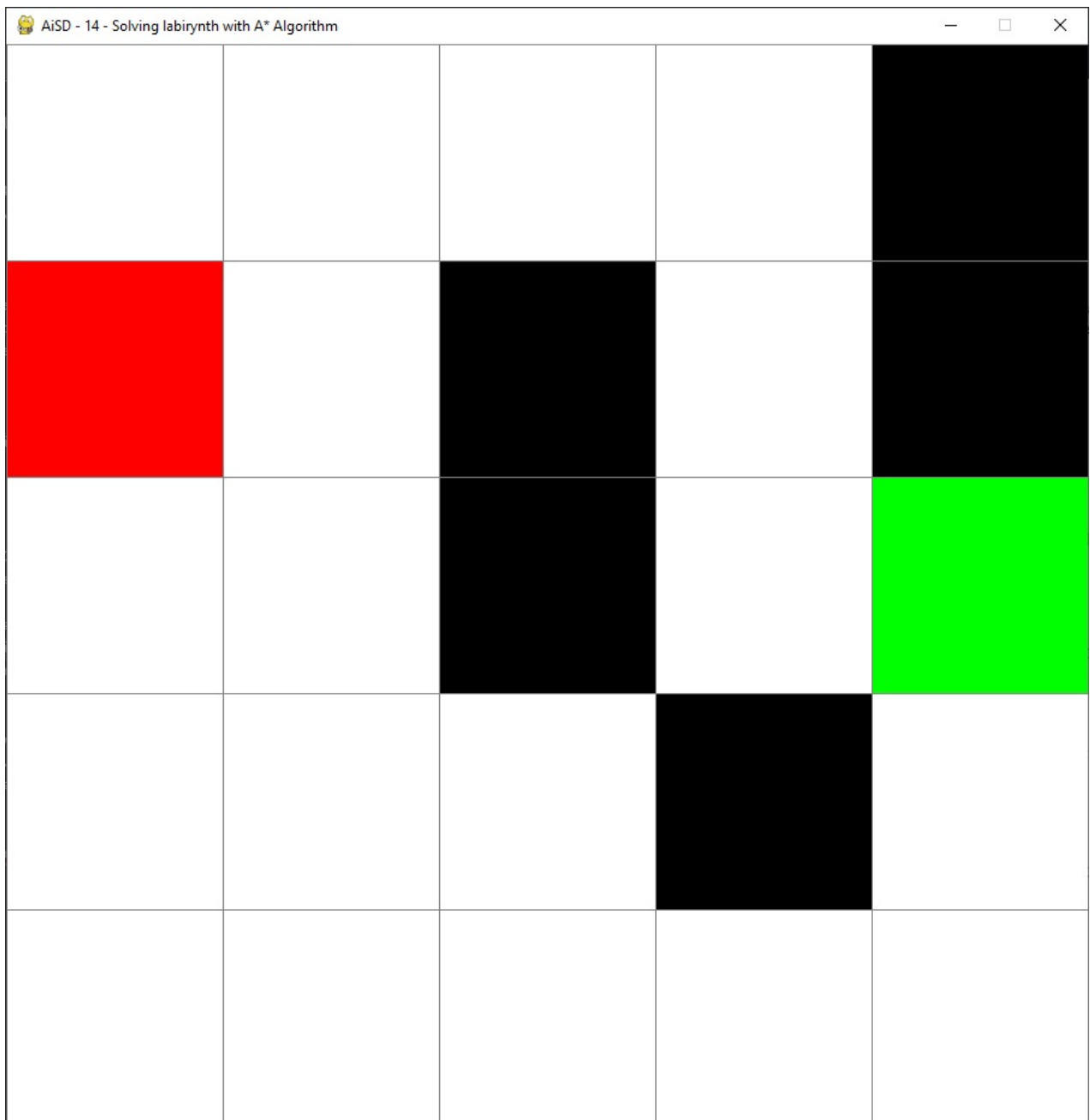
Program konwertuje ten układ labiryntu na macierz sąsiedztwa i powstaje plik takiej postaci:



```
main.py  adj_matrix.txt X
adj_matrix.txt
1  01000100000000000000000000000000
2  10100010000000000000000000000000
3  01010000000000000000000000000000
4  00100000100000000000000000000000
5  00000000000000000000000000000000
6  10000010001000000000000000000000
7  01000100000100000000000000000000
8  00000000000000000000000000000000
9  00010000000010000000000000000000
10 00000000000000000000000000000000
11 00000100000100010000000000000000
12 00000010001000001000000000000000
13 00000000000000000000000000000000
14 00000000100000100000000000000000
15 00000000000010000010000000000000
16 00000000001000001000100000000000
17 00000000000100010100010000000000
18 00000000000000001000001000000000
19 00000000000000000000000000000000
20 00000000000001000000000010000000
21 00000000000000100000100000000000
22 00000000000000001000101000000000
23 00000000000000000010001010000000
24 00000000000000000000000010100000
25 00000000000000000000001000100000
```

Rysunek 6: Macierz sąsiedztwa wygenerowana na podstawie układu labiryntu, 1 - wierzchołki są połączone, 0 - nie są

Z wygenerowanej macierzy sąsiedztwa tworzone są odpowiednie wierzchołki i układane na narysowanej siatce. Dla podanego wyżej układu labiryntu jak i wygenerowanej macierzy sąsiedztwa otrzymamy następujący efekt.

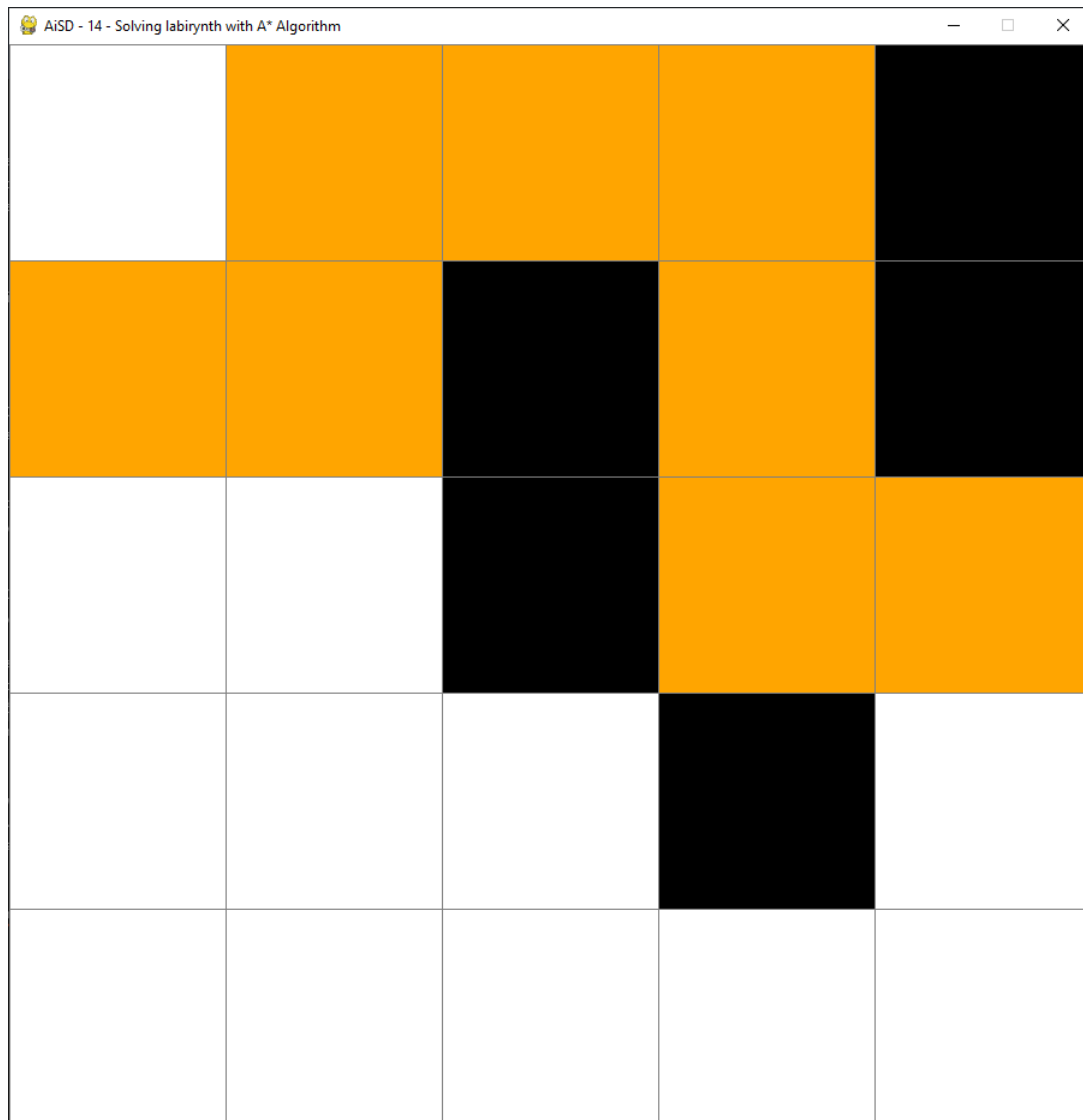


Rysunek 7: Wizualizacja wygenerowanego labiryntu

Czerwony - start, zielony - koniec, czarny - ściana, biały - wolne



Po wygenerowaniu powyższego labiryntu aby uzyskać efekt działania algorytmu, czyli znaleźć najkrótszą drogę w labiryncie wystarczy wcisnąć przycisk SPACE. Efekt po wciśnięciu SPACE:



Rysunek 8: Pomarańczowe pola oznaczają najkrótszą trasę z punktu startowego do końcowego

Dodatkowo w konsoli generowany jest układ pól po wyznaczający trasę z startu do końca.

```
5 -> 6 -> 1 -> 2 -> 3 -> 8 -> 13 -> 14
```

## 4.2 Dodatkowe informacje

Wymagania:

1. Python 3.6 lub nowszy
2. Zainstalowany moduł pygame (wizualizacja rozwiązania labiryntu)
3. Windows 7 lub nowszy

## 4.3 Testy

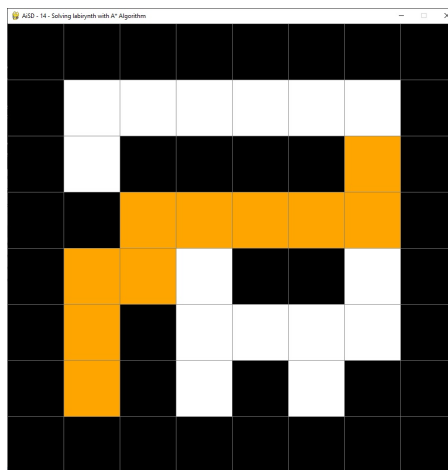
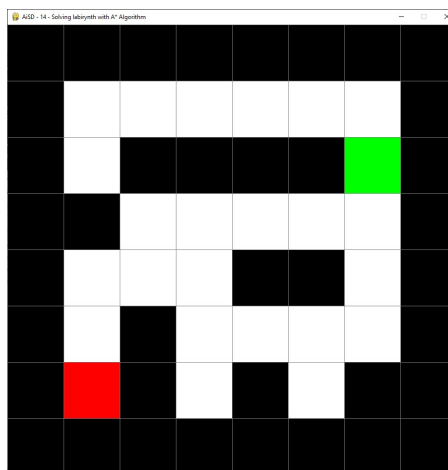
Aby zmienić plik z labiryntem wystarczy wpisać jego ścieżkę i nazwę w tym miejscu w skrypcie main.py :

```
223 # algorithm main
224 maze = open("maze2.txt",'r')
```

Rysunek 9: Miejsce w pliku main.py do zmiany labiryntu

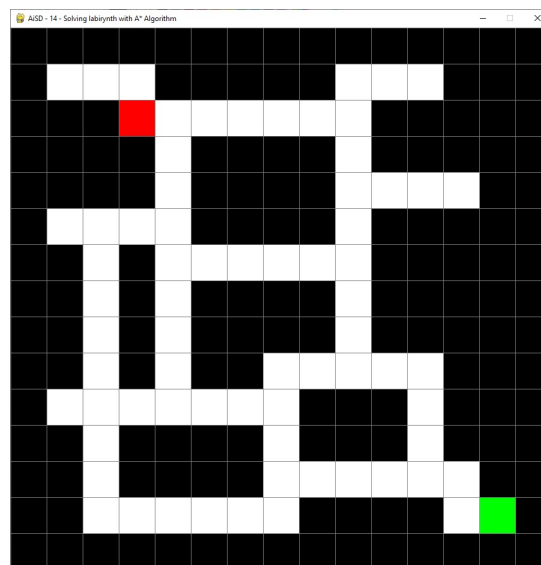
Test dla labiryntu 8x8:

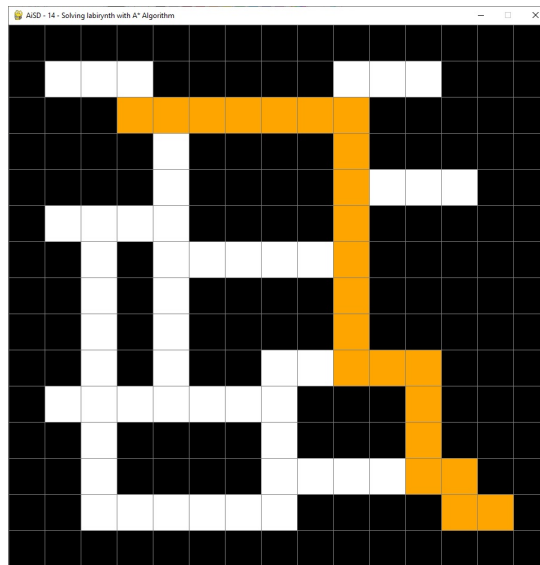
```
maze3.txt
1  XXXXXXXX
2  X000000X
3  X0XXXEX
4  XX00000X
5  X000XX0X
6  X0X0000X
7  XSX0X0XX
8  XXXXXXXX
```



Test dla labiryntu 15x15:

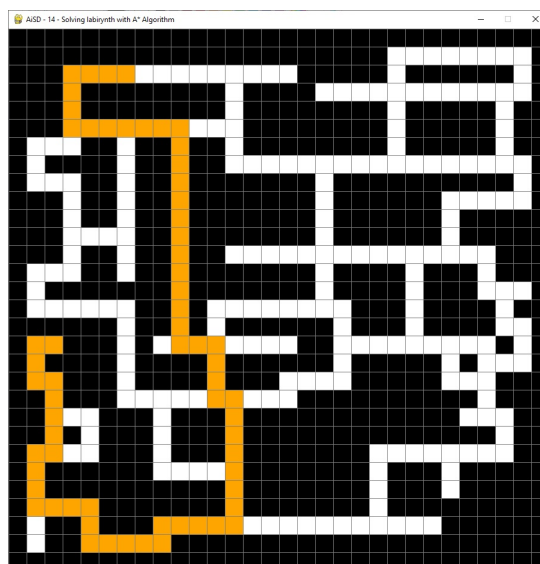
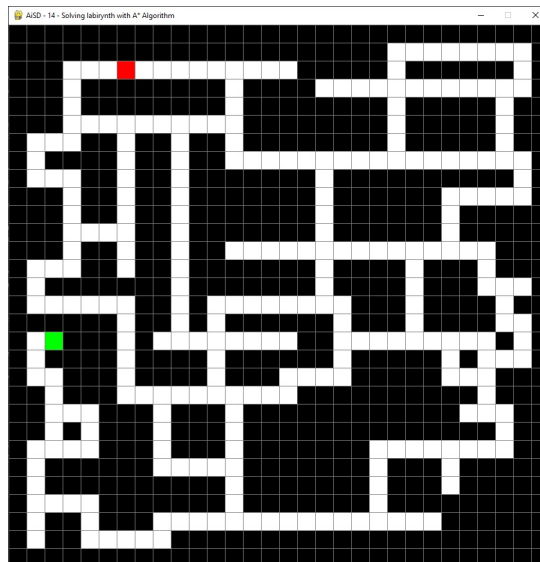
```
maze4.txt
1  XXXXXXXXXXXXXXXX
2  X000XXXXX000XX
3  XXX5000000XXXXX
4  XXXX0XXXX0XXXXX
5  XXXX0XXXX0000XX
6  X0000XXXX0XXXXX
7  XX0X000000XXXXX
8  XX0X0XXXX0XXXXX
9  XX0X0XXXX0XXXXX
10 XX0X0XX00000XX
11 X0000000XXX0XXX
12 XX0XXX0XXX0XXX
13 XX0XXX000000XX
14 XX000000XXX0EX
15 XXXXXXXXXXXXXXXX
```





Test dla labiryntu 30x30:

```
maze5.txt
1  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2  XXXXXXXXXXXXXXXXXXXXXXX0000000X
3  XXX0005000000000XXXXX0XXXXX0X
4  XXX0XXXXXXXXXX0XXX00000000000X
5  XXX0XXXXXXXXXX0XXXXXXXX0XXXX0XX
6  XXX0000000000XXXXXXX0XXXX0XX
7  X000XX0XX0XX0XXXXXXXX0XXXX0XX
8  X0XXX0XX0XX0000000000000000X
9  X000XX0XX0XXXXXXXX0XXXXXXXXXX0X
10 XXX0XX0XX0XXXXXX0XXXXXX00000X
11 XXX0XX0XX0XXXXXX0XXXXXX0XXXXX
12 XXX0000XX0XXXXXX0XXXXXX0XXXXX
13 XXX0XX0XX0X000000000000000XXX
14 X000XX0XX0XXXXXX0XXX0XXX0XXX
15 X0XXXXXX0XXXXXX0XXX0XXX000X
16 X000000XX0X00000000XX0XXX0XX
17 XXXXX0XX0X0XXXXXX0XXX0XXX00X
18 X0EXXX0X00000000XX00000000X0X
19 X0XXX0XXX0XXXXXX0XXXXX0X000X
20 X00XX0XXX0XX0000XXXXX000XXX
21 XX0XX0000000000XXXXXXX0XXX
22 XX000XX0XX0XXXXXXXXXXXX000XX
23 XX0X0XX0XX0XXXXXXXXXXXX0XX
24 X0000XX0XX0XXXXXX00000000XX
25 X0XXXXXX00000XXXXXX0XXX0XXXXX
26 X0XXXXXXX0XXXXXX0XXX0XXXXX
27 X0000XXXXXX0XXXXXX0XXXXXXXXX
28 X0XX0XX000000000000000XXXXX
29 X0X00000XXXXXXX0XXXXXXXXXXXXX
30 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```



## 4.4 Kod źródłowy - main.py

```
1 from math import sqrt
2 import pygame
3
4 WIDTH = 900
5 WIN = pygame.display.set_mode((WIDTH, WIDTH))
6 pygame.display.set_caption("AiSD - 14 - Solving labirynt with A*
   Algorithm")
7 RED = (255, 0, 0)
8 GREEN = (0, 255, 0)
9 WHITE = (255, 255, 255)
10 BLACK = (0, 0, 0)
11 ORANGE = (255, 165, 0)
12 GREY = (128, 128, 128)
13
14
15
16 class Wierzcholek:
17     def __init__(self, position, symbol, start=None, end=None, wall=None
18         , free=None):
19         self.position = position
20         self.symbol = symbol
21         self.connections = []
22         self.start = start
23         self.end = end
24         self.wall = wall
25         self.free = free
26
27     def show_info(self):
28         if self.start:
29             print(f"Wierzcholek {self.position} - START - {self.symbol}
30                 - {self.connections}")
31         elif self.end:
32             print(f"Wierzcholek {self.position} - END - {self.symbol} -
33                 {self.connections}")
34         else:
35             print(f"Wierzcholek {self.position} - {self.symbol} - {self.
36                 connections}")
37
38     def add_connections(self, graph):
39         n = len(graph)
40         row_len = int(sqrt(n))
41         if self.symbol == 'X':
42             return
43         # first row
44         if self.position >= 0 and self.position < row_len:
45             if self.position % row_len == 0:
46                 if graph[self.position + 1].symbol != 'X':
47                     self.connections.append(self.position + 1)
48                 if graph[self.position + row_len].symbol != 'X':
49                     self.connections.append(self.position + row_len)
50             elif self.position % row_len == row_len - 1:
51                 if graph[self.position - 1].symbol != 'X':
52                     self.connections.append(self.position - 1)
```



```

49         if graph[self.position + row_len].symbol != 'X':
50             self.connections.append(self.position + row_len)
51     else:
52         if graph[self.position - 1].symbol != 'X':
53             self.connections.append(self.position - 1)
54         if graph[self.position + 1].symbol != 'X':
55             self.connections.append(self.position + 1)
56         if graph[self.position + row_len].symbol != 'X':
57             self.connections.append(self.position + row_len)
58     # last row
59     elif self.position >= n - row_len:
60         if self.position % row_len == 0:
61             if graph[self.position + 1].symbol != 'X':
62                 self.connections.append(self.position + 1)
63             if graph[self.position - row_len].symbol != 'X':
64                 self.connections.append(self.position - row_len)
65         elif self.position % row_len == row_len - 1:
66             if graph[self.position - 1].symbol != 'X':
67                 self.connections.append(self.position - 1)
68             if graph[self.position - row_len].symbol != 'X':
69                 self.connections.append(self.position - row_len)
70     else:
71         if graph[self.position - 1].symbol != 'X':
72             self.connections.append(self.position - 1)
73         if graph[self.position + 1].symbol != 'X':
74             self.connections.append(self.position + 1)
75         if graph[self.position - row_len].symbol != 'X':
76             self.connections.append(self.position - row_len)
77     # mid rows
78     else:
79         if self.position % row_len == 0:
80             if graph[self.position + 1].symbol != 'X':
81                 self.connections.append(self.position + 1)
82             if graph[self.position + row_len].symbol != 'X':
83                 self.connections.append(self.position + row_len)
84             if graph[self.position - row_len].symbol != 'X':
85                 self.connections.append(self.position - row_len)
86         elif self.position % row_len == row_len - 1:
87             if graph[self.position - 1].symbol != 'X':
88                 self.connections.append(self.position - 1)
89             if graph[self.position + row_len].symbol != 'X':
90                 self.connections.append(self.position + row_len)
91             if graph[self.position - row_len].symbol != 'X':
92                 self.connections.append(self.position - row_len)
93     else:
94         if graph[self.position - 1].symbol != 'X':
95             self.connections.append(self.position - 1)
96         if graph[self.position + 1].symbol != 'X':
97             self.connections.append(self.position + 1)
98         if graph[self.position + row_len].symbol != 'X':
99             self.connections.append(self.position + row_len)
100         if graph[self.position - row_len].symbol != 'X':
101             self.connections.append(self.position - row_len)
102
103 class Node():

```

```

104     def __init__(self, id, neighbors_id, matrix_pos, width, color=WHITE,
105                 block=None):
106         self.id = id
107         self.neighbors_id = neighbors_id
108         self.neighbors = []
109         self.block = block
110         self.matrix_pos = matrix_pos
111         self.color = color
112         self.width = width
113
114     def get_info(self):
115         print(f"Node {self.id}, {self.matrix_pos}", end=" - ")
116         for node in self.neighbors:
117             print(node.id, end=" ")
118         if self.block != None:
119             print(self.block, end=" ")
120         print()
121
122     def draw(self, win, gap):
123         pygame.draw.rect(win, self.color, (self.matrix_pos[1] * gap,
124             self.matrix_pos[0] * gap, gap, gap))
125
126 # end class
127 def create_adjacency_matrix(graph):
128     n = len(graph)
129     A = [[0 for x in range(n)] for y in range(n)]
130     i = 0
131     j = 0
132     for wierzcholek in graph:
133         if len(wierzcholek.connections) > 0:
134             for j in wierzcholek.connections:
135                 A[i][j] = 1
136             i += 1
137     ad_matrix = open("adj_matrix.txt", 'w')
138     for row in A:
139         str_row = ""
140         for char in row:
141             str_row += str(char)
142         ad_matrix.write(str_row+'\n')
143
144     return A
145
146 def h(n1, n2):
147     x1, y1 = n1.matrix_pos
148     x2, y2 = n2.matrix_pos
149     return abs(x1 - x2) + abs(y1 - y2)
150
151 def astar_algorithm(draw, graph):
152     for node in graph:
153         if node.block == "START":
154             start_node = node
155         elif node.block == "END":
156             end_node = node
157
158     Q = []

```

```

157     S = []
158     n = len(graph)
159     g = [float("inf") for x in range(n)]
160     g[start_node.id] = 0
161     f = [float("inf") for x in range(n)]
162     f[start_node.id] = h(start_node, end_node)
163     pred = [0 for x in range(n)]
164     Q.append(start_node)
165
166     while len(Q) > 0:
167         min_f = Q[0]
168         for node in Q:
169             if f[node.id] < f[min_f.id]:
170                 min_f = node
171         if min_f.block == "END":
172             reconstruct_path(draw, graph, pred, start_node, end_node)
173             return True
174         for neighbor in min_f.neighbors:
175             temp_g = g[min_f.id] + 1
176             if neighbor not in Q and neighbor not in S:
177                 Q.append(neighbor)
178                 g[neighbor.id] = temp_g
179                 f[neighbor.id] = h(neighbor, end_node) + g[neighbor.id]
180                 pred[neighbor.id] = min_f.id
181         Q.remove(min_f)
182         S.append(min_f)
183
184     def reconstruct_path(draw, graph, pred, start_node, end_node):
185         path = []
186         path.append(end_node.id)
187         current_node = pred[end_node.id]
188         while current_node != start_node.id:
189             path.append(current_node)
190             current_node = pred[current_node]
191         path.append(start_node.id)
192         path.reverse()
193
194         for node in path:
195             graph[node].color = ORANGE
196             draw()
197
198     for i, node in enumerate(path):
199         if i != len(path) - 1:
200             print(f"{node} -> ", end="")
201         else:
202             print(f"{node}")
203
204
205
206 # pygame functions
207 def draw_grid(win, rows, width):
208     gap = width // rows
209     for i in range(rows):
210         pygame.draw.line(win, GREY, (0, i * gap), (width, i * gap))
211     for j in range(rows):

```

```

212         pygame.draw.line(win, GREY, (j * gap, 0), (j * gap, width))
213
214 def draw(win, grid, rows, width):
215     gap = width // rows
216     win.fill(WHITE)
217     for node in grid:
218         node.draw(win, gap)
219
220     draw_grid(win, rows, width)
221     pygame.display.update()
222
223 # algorithm main
224 maze = open("maze2.txt", 'r')
225 maze_matrix = []
226 graph = []
227 for row in maze.readlines():
228     maze_row = ""
229     for char in row.replace('\n', ""):
230         maze_row += char
231     maze_matrix.append(maze_row)
232
233 i = 0
234 for row in maze_matrix:
235     for char in row:
236         if char == 'S':
237             graph.append(Wierzcholek(i, char, start=True))
238             i += 1
239         elif char == "E":
240             graph.append(Wierzcholek(i, char, end=True))
241             i += 1
242         elif char == "X":
243             graph.append(Wierzcholek(i, char, wall=True))
244             i += 1
245         elif char == "0":
246             graph.append(Wierzcholek(i, char, free=True))
247             i += 1
248
249 for wierzcholek in graph:
250     wierzcholek.add_connections(graph)
251     # wierzcholek.show_info()
252
253 create_adjacency_matrix(graph)
254 adj_matrix = open("adj_matrix.txt", 'r')
255
256 final_graph = []
257 n = len(graph)
258 row_len = int(sqrt(n))
259 ROWS = row_len
260 gap = WIDTH // ROWS
261 i = 0
262 k = 0
263 for row in adj_matrix.readlines():
264     j = 0
265     current_neighbors = []
266     for char in row.replace("\n", ''):

```

```

267         if char != '0':
268             current_neighbors.append(j)
269         j += 1
270     if graph[i].start:
271         final_graph.append(Node(i, current_neighbors, matrix_pos=(k, i%
            row_len), width=gap, color=RED, block="START"))
272     elif graph[i].end:
273         final_graph.append(Node(i, current_neighbors, matrix_pos=(k, i%
            row_len), width=gap, color=GREEN, block="END"))
274     elif graph[i].wall:
275         final_graph.append(Node(i, current_neighbors, matrix_pos=(k, i%
            row_len), width=gap, color=BLACK))
276     elif graph[i].free:
277         final_graph.append(Node(i, current_neighbors, matrix_pos=(k, i%
            row_len), width=gap, color=WHITE))
278     i += 1
279     if i % row_len == 0:
280         k += 1
281
282 for node in final_graph:
283     for neighbors_id in node.neighbors_id:
284         for n in final_graph:
285             if neighbors_id == n.id:
286                 node.neighbors.append(n)
287
288
289 # main pygame program
290 grid = 0
291 run = True
292 while run:
293     draw(WIN, final_graph, ROWS, WIDTH)
294     for event in pygame.event.get():
295         if event.type == pygame.QUIT:
296             run = False
297         if event.type == pygame.KEYDOWN:
298             if event.key == pygame.K_SPACE:
299                 astar_algorithm(lambda: draw(WIN, final_graph, ROWS,
                    WIDTH), final_graph)

```

---