# Dokumentacja – Zadanie 1

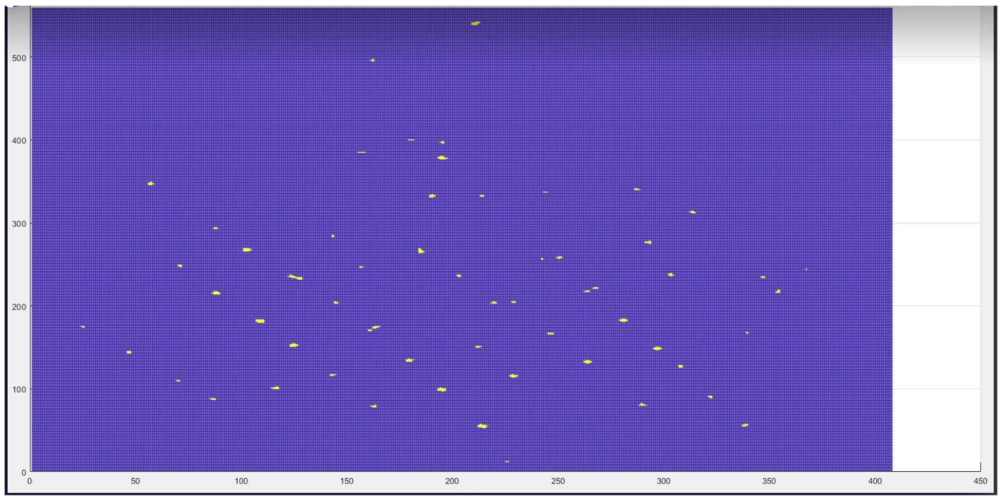
1. Finalna funkcja nie jest użytkowa, po kilkudziesięciu iteracjach wpada w nieskończoną pętlę.
2. Wykorzystano algorytm A*. Funkcja przyjmuje parametry zgodnie z wytycznymi zawartymi w pliku Task_Guidelines.

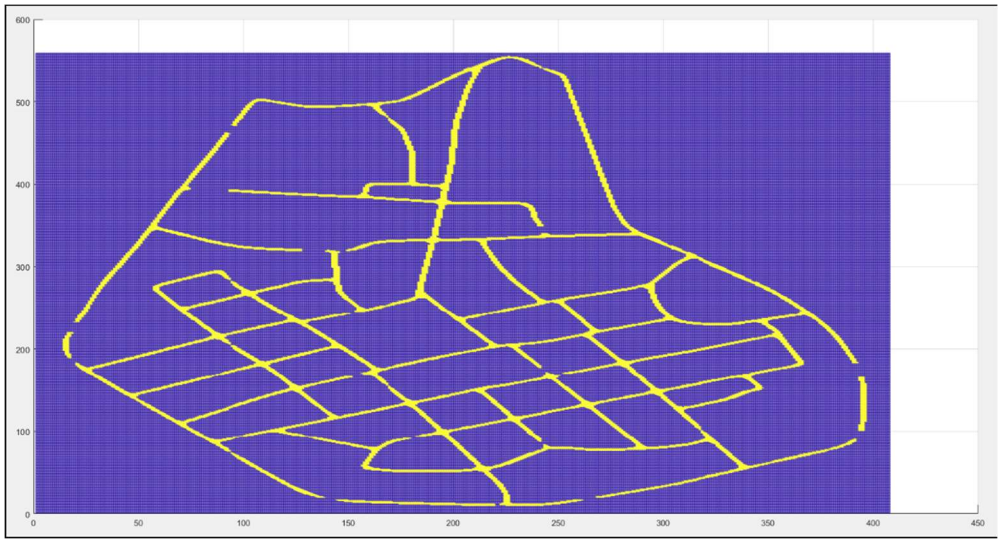
```
%Galeon
function path = task_1(map, startPoint, stopPoint)
    map1= map(:,:,1);
    cost = map(:,:,2) .* map(:,:,3);
    start = startPoint;
    stop = stopPoint;
    next_steps = [start];
    dist_to_go = sqrt(sum((start-stop).^2));
    start_next = start;
    while(next_steps(end) ~= stop)
    neigh_costs = cost(start_next(1)-1:start_next(1)+1, start_next(2)-1:start_next(2)+1);
    neigh_dirs = zeros(3);
    for i = -1:1
        for j = -1:1
            neigh_dirs(i+2,j+2) = sqrt(sum((start+[i,j]-stop).^2));
        end
    end
    neigh_dirs(2,2) = Inf;
    neigh_dirs = neigh_dirs + neigh_costs;
    minimum = min(min(neigh_dirs));
    [xmin,ymin] = find(neigh_dirs == minimum);
    next_point = start_next + [xmin, ymin] - [2,2];
    next_steps(end+1, :) = next_point;
    start_next = next_point;
end
path = next_steps;
end
```

3. Podczas hackathonu były też testowane inne podejścia, oto niektóre z nich:
   3.1 Jeden z naszych eksperymentów obejmował stworzenie grafu na podstawie kosztów ujętych w mapie. Przed tym, by ułatwić działanie na macierzy drogi były zwężane różnymi metodami. Pierwsza z nich to tresholding pikseli na podstawie średniej wartości w oknie.

```
Layer1 = map(:,:,1);

windowSize = 2;

xSize = length(Layer1(1,:));
ySize = length(Layer1(:,1));
newCity = zeros()
crosses = [];

for j = windowSize + 1 : xSize - windowSize
    for i = windowSize + 1 : ySize - windowSize
        % pixel = Layer1(i,j);
        neighbour = Layer1(i-windowSize : i+windowSize,
...                        j-windowSize : j+windowSize);
        newPixel = mean(neighbour, 'all');
        if newPixel < 0.85
            newCity(i,j) = 0;
        else
            newCity(i,j) = 1;
        end
    end
end
mesh(newCity)
```

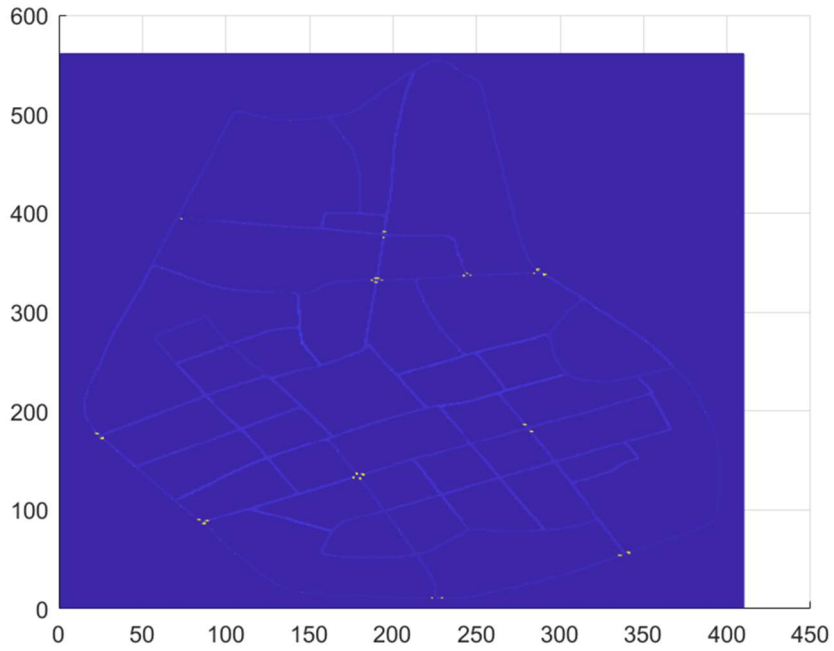
Efekty metody:




Inna metoda wyłuskiwania skrzyżowań i zawężania ścieżek obejmowała erozję obrazu:

```
kernel=3;

map1 = map(:,:,1)
se = [0 1 0; 1 1 1; 0 1 0];
nh =  [0 1 0; 1 1 1; 0 1 0];
eroded = imerode(map1, nh)

cost = map(:,:,2) .* map(:,:,3) + map(:,:,4);
map_act = cost .* eroded;

mesh(map_act)
```




Zwiększeni progu umożliwiło identyfikacje skrzyżowań. Na podstawie tych współrzędnych planowane było utworzenie grafu. Struktura grafu obejmująca całą mapę miała być przekazywana do funkcji. Dzięki temu możliwe byłoby wykorzystanie algorytmów grafowych na znacznie mniejszym zbiorze wierzchołków co korzystnie wpłynęłoby na czas działania algorytmu.

Opracowany został też algorytm, który znajdywał takie miejsce na „pomniejszonych ścieżkach", które było najbliższe wylosowanemu punktowi przekazywanemu do algorytmu:

```
randomPoint = [321, 187];
reducedStartingPoint = [0, 0];

while true
    neighbourhood = newCityRoads(randomPoint(1) - windowSize : randomPoint(1) + windowSize, ...
                                 randomPoint(2) - windowSize : randomPoint(2) + windowSize);
    if any(neighbourhood, 'all')
        neighbourhoodSize = length(neighbourhood(1,:));

        for j = 1 : neighbourhoodSize
            for i = 1 : neighbourhoodSize
                if neighbourhood(i, j) == 1
                    reducedStartingPoint = [randomPoint(1) - windowSize + i, ...
                                            randomPoint(2) - windowSize + j];
                end
            end
        end
        break
    else
        windowSize = windowSize + 1;
    end
end
```

3.2 Powstał też pomysł, by utworzyć graf z większą szczegółowością. Na takim grafie zastosowała wykorzystana funkcja pathminsearch(), niestety ze względu na powstałe cykle i dużą gęstość grafu funkcja nie radziła sobie ze znalezieniem ścieżki. Jednak by struktura mogła zostać przekazana do funkcji powstał algorytm generujący wektor: s –

początków ścieżek w grafie, t – końców ścieżek w grafie i weight – wag poszczególnych ścieżek.

```
cost = map(:,:,2) .* map(:,:,3) + map(:,:,4);
minimap = eroded;
[X, Y] = size(minimap);
s = [0];
t = [0];
weight = [0];
nodesXY = {[0,0]};

for j = 2:(X-1)
    for i = 2:(Y-1)
        if minimap(j,i) ~= 0 &  ~ismember([j, i], cell2mat(nodesXY(:)),"rows")
            nodesXY{end+1} = [j,i];
            nodeNeighbours = minimap(j-1:j+1, i-1:i+1);
            [~, s_tmp] = size(nodesXY);
            for k = -1:1
                for l = -1:1
                    if minimap(j+k, i+l) ~= 0 & ~ismember([j+k, i+l], cell2mat(nodesXY(:)),"rows")
                        elem = minimap(j+k, i+l);
                        nodesXY{end+1} = [j+k,i+l];
                        [~, t_tmp] = size(nodesXY);
                        s(end+1) = s_tmp;
                        t(end+1) = t_tmp;
                        weight(end+1) = minimap(nodesXY{s_tmp}(1), nodesXY{s_tmp}(2)) +
minimap(nodesXY{t_tmp}(1), nodesXY{t_tmp}(2));
                    end
                end
            end
        end
    end
end
nodesXY = nodesXY(2:end);
s = s(2:end)-1;
t = t(2:end)-1;
weight = weight(2:end);

minimap
s
t
nodesXY
weight

G = graph(s,t,weight) go(f, seed, [])
}
```

3.3. Ostatni pomysł obejmował użycie gradienty funkcji kosztów. Tam, gdzie funkcja byłą stała następowało zerowanie kosztu, dzięki czemu ograniczona została dziedzina problemu i przestrzeń rozwiązań.