**Automatyczny kelner - planowanie ruchu**

**Raport na przedmiot Sztuczna Inteligencja**

Aby kelner mógł poruszyć się jak najszybciej z punktu A do punktu B, musi wyznaczyć najszybszą możliwą ścieżkę.

Do naszego projektu użyliśmy algorytmu A\*, ze względu na jego szybkość i dokładność. Postanowiliśmy użyć go zamiast algorytmu Djikistry, który kalkuluje każdą możliwą ścieżkę do celu w restauracji, przez co traci na czasie działania.

W projekcie wykorzystaliśmy fragmenty kodu użytkownika Nicholas Swift <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2>

Kod został napisany w Pythonie 3.7, użyty został moduł PyGame. Wykorzystujemy stworzone przez nas środowisko zapisane w postaci tabeli liczb binarnych (0 – wolne miejsce do poruszania się, 1 – przeszkoda). Pole startowe to pozycja kelnera (obecnie statyczna [3,3]), pole końcowe wyznaczamy na planszy.

**Działanie:**

1. Dodajemy początkowy węzeł do listy węzłów nieodwiedzonych.

open\_list.append(start\_node)

2. Powtarzamy:

while len(open\_list) > 0:

A) Szukamy pola z najniższym możliwym kosztem F na liście węzłów nieodwiedzonych.

current\_node = open\_list[0]

current\_index = 0

for index, item in enumerate(open\_list):

if item.f < current\_node.f:

current\_node = item

current\_index = index

B) Przeniosimy go na listę miejsc odwiedzonych

open\_list.pop(current\_index)

closed\_list.append(current\_node)

C) Dla wszystkich ośmiu pól obok obecnie używanego pola:

for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]:

-Jeżeli jest to stół, lub pole na liście odwiedzonych, ignoruj go.

if restaurant[node\_position[0]][node\_position[1]] != 0:

continue

-Jeżeli nie jest na liście odwiedzonych, dodaj go do listy odwiedzonych. Pole, wokół którego interpretujemy pola staje się jego rodzicem. Zapisujemy koszty F, G i H tego pola.

for child in children:

child.g = current\_node.g + 1

child.h = ((child.position[0] - end\_node.position[0]) \*\* 2) + ((child.position[1] - end\_node.position[1]) \*\* 2)

child.f = child.g + child.h

open\_list.append(child)

-Jeżeli jest już na liście pól nieodwiedzonych, sprawdzamy, czy ścieżka do tego pola jest lepsza porównując koszty G. Niższy koszt G oznacza lepszą ścieżkę. Jeżeli tak jest, zmieniamy rodzica sąsiedniego pola na pole wokół którego interpretujemy pola, i obliczamy ponownie koszty F i H.

for open\_node in open\_list:

if child == open\_node and child.g > open\_node.g:

continue

D) Przerywamy, gdy:

-Dodamy nasz cel do listy pól odwiedzonych

if current\_node == end\_node:

path = []

current = current\_node

-Algorytm nie znajdzie szukanego pola. Oznacza to, że ścieżka nie może zostać wytyczona, bądź nie istnieje

3. Zapisujemy ścieżkę. Powtarzamy obliczenia wyznaczoną ścieżką z powrotem do naszego pola początkowego. Zostaje wyznaczona najszybsza ścieżka

while current is not None:

path.append(current.position)

current = current.parent

return path[::-1]

Źródła:

-[https://pl.wikipedia.org/wiki/Algorytm\_A\*](https://pl.wikipedia.org/wiki/Algorytm_A*)

-<https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2>