Autorzy: Jakub Guzek, Kacper Stachyra, Marcin Jarząbek
Prowadzący: dr inż. Paweł Wawrzyński

PODSTAWY SZTUCZNEJ INTELIGENCJI
NAWIGATOR

OPIS PROBLEMU

Napisać program znajdujący drogę przy użyciu algorytmu A*. Dla zadanych przez użytkownika parametrów n, r program generuje Graf_losowy(n,r), który prezentowany jest użytkownikowi. Użytkownik wskazuje myszką dwa punkty, między którymi wyznaczana jest najkrótsza droga.
Sprawdzić n=100, r=0.1, 0.2, 0.5;
Przez Graf_losowy(n,r) dla całkowitego n>1 oraz rzeczywistego r>0 jest poniżej rozumiana następująca struktura: Na kwadracie o boku 1 losowanych jest n punktów, stanowiących węzły grafu. Następnie, ustanawiane są krawędzie grafu w następujący sposób:

1. brana jest każda kolejna para węzłów, takich par jest 0.5n(n+1);
2. niech współrzędne tych węzłów to wektory v oraz z,
   - z rozkładu jednostajnego na przedziale [0,1] losowana jest liczba losowa, x,
   - jeśli x < exp(-||v-z||/r), wówczas jest ustanawiana krawędź między tymi węzłami.

Parametr r stanowi w przybliżeniu taką wartość, że jeśli węzły są w odległości mniejszej niż r, to prawdopodobnie będzie krawędź między nimi, a jeśli większej – to jej nie będzie.

ZAŁOŻENIA PROJEKTOWE

- Projekt zostanie napisany w języku Java, z wykorzystaniem wzorca projektowego MVC
- Model będzie stanowić:
  - implementacja algorytmu A*, operującego na abstrakcyjnych stanach,
  - implementacja grafu (reprezentującego siatkę miast), który posłuży do zaprezentowania działania algorytmu
- Program będzie udostępniał graficzny interfejs użytkownika, pozwalający na wprowadzenie zadanych wartości i wskazanie dwóch węzłów (miast) grafu, pomiędzy którymi ma być wyznaczona ścieżka. Jeżeli ścieżka istnieje – zostanie wyświetlona na ekranie (w przeciwnym wypadku zostanie wyświetlony komunikat o braku rozwiązania).

INSTRUKCJA UŻYTKOWA

Do uruchomienia programu potrzebne jest środowisko Java Runtime Enviroment (JRE).
Po uruchomieniu programu należy:

- wpisać w odpowiednie pola wartość parametru r (pole ***Rate***) oraz liczbę wierzchołków grafu (pole ***Number***),
- kliknąć przycisk ***Generate!***
- kliknąć przycisk ***Start***, a następnie kliknąć na wybranym węźle, mającym stać się węzłem początkowym,
- kliknąć przycisk ***End*** i kliknąć na wybranym węźle, mającym stać się węzłem końcowym,
- kliknąć przycisk ***Solve!***
- Na ekranie – jeśli istnieje – zostanie wyświetlona ścieżka pomiędzy wskazanymi węzłami, oraz w panelu ***Logs*** zostanie wyświetlona informacja o liczbie kroków, w której algorytm znalazł rozwiązanie,
- jeśli ścieżka pomiędzy zadanymi węzłami nie istnieje w panelu ***Logs*** zostanie wyświetlona informacja o braku takiej ścieżki

Pomoc jest dostępna również w menu programu (Menu → Help).

STRUKTURA PROGRAMU

W modelu zostały zaimplementowane następujące, związane z algorytmem A* i grafem, klasy:

class Node

Klasa reprezentująca wierzchołek grafu. Obiekt tej klasy przechowuje listę referencji na swoich sąsiadów (inne wierzchołki w grafie) – odpowiada to ideowo siatce dróg, które wychodzą z tego miasta

class Graph

Klasa grafu, przechowująca listę wierzchołków. Udostępnia metodę *init()* zwracającą stan początkowy s_0, którym inicjowany jest algorytm A*.

class State (implementing ProblemState)

Klasa, której obiekty reprezentują stany, na których operuje algorytm A*.
Implementuje interfejs *ProblemState* – dzięki czemu udostępnia metody:

- *getEstimatedLength() - zwraca wartość f(s) = C(s) + h(s), s – badany stan*
- *isFinish() - zwraca informację, czy stan jest terminalny*
- *extendStates() - rozwija stan w zbiór stanów*

wykorzystywane przez obiekt klasy AStar. Taka struktura rozwiązania pozwala algorytmowi A* operować na abstrakcyjnych stanach – dzięki czemu sam algorytm można wykorzystać do rozwiązania innego problemu, niekoniecznie związanego z grafem. W tym celu należy napisać klasę stanu implementującą interfejs *ProblemState*.

class AStar

Klasa stanowiąca implementację algorytmu A*. Operuje na listach stanów otwarych i zamkniętych. Udostępnia metodę *solve()*, w której do momentu znalezienia stanu terminalnego (lub stwierdzenia że jest to niemożliwe) rozwijany jest stan s_i, minimalizujący wartość funkcji

$$f(s) = C(s) + h(s)$$

gdzie

C(s) - koszt dotarcia od stanu początkowego do stanu s - w przypadku grafu badanego grafu jest to przebyta droga od wierzchołka początkowego do aktualnie badanego.

h(s) - heurystycznie oszacowany koszt dotarcia od stanu s do końca - w przypadku grafu jest to odległość w linii prostej do wierzchołka.

Po wprowadzeniu przez użytkownika parametrów, wygenerowaniu grafu, wybraniu punktów początkowego i końcowego i kliknięciu przycisku Solve tworzony jest obiekt klasy AStar, inicjowany stanem początkowym, a następnie wywoływana jest metoda solve() tej klasy – zwracająca w wypadku sukcesu stan terminalny (zawierający ścieżkę między węzłem początkowym a końcowym), bądź wartość null w wypadku porażki.

WNIOSKI

Udało się osiągnąć założenia projektowe, polegające na oddzieleniu struktury informacyjnej (graf reprezentujący miasta) od algorytmu A*, przez co sam algorytm może być wykorzystany do rozwiązania innego problemu.

Zaimplementowany w projekcie algorytm A* (pod postacią klasy AStar) pozwala na odnalezienie, jeśli istnieje, najkrótszej ścieżki pomiędzy dwoma wskazanymi przez użytkownika węzłami – miastami.

Realizacja projektu pozwoliła na zapoznanie się z algorytmem A*, oraz uniwersalnością metod przeszukiwania przestrzeni stanów.