OPIS ALGORYTMU DO PROJEKTU ZESPOŁOWEGO - GRUPA 4

12 stycznia 2020

Krzysztof Anderson i Michał Malinowski

Spis treści

1 Opis algorytmu

Algorytm służy do mapowania labiryntu, korzystając z komend HTTP wywoływanych na udostępnionym nam serwerze. Algorytm został wymyślony przez naszą grupę bazując na różnych poznanych przez nas metodach.
Algorytm operuje na strukturze `Maze` składającej się z tablicy dwuwymiarowej obiektów `Node`. Obiekty `Node` posiadają:

- informację o swoich sąsiadach,
- odległość od węzła startowego,
- swojego poprzednika,
- pole informujące czy węzeł został odwiedzony,
- swoje koordynaty.

Ważnym elementem struktury wykorzystywanej przez algorytm jest lista skrzyżowań - `ArrayList<Node> junctions` zawierająca węzły w których pozostały niesprawdzone ścieżki.

1.1 Działania początkowe

Na podstawie informacji otrzymanych z serwera, program tworzy strukturę `Maze` o zadanych wymiarach. Odległości wszystkich `Node` wypełniamy -1, a w `Node` startowy wpisujemy 0 i uzupełniamy jego sąsiadów.

1.2 Pętla algorytmu

Zliczamy nieodwiedzonych sąsiadów węzła i jeżeli jest ich 2 lub więcej, dodajemy go do listy skrzyżowań. Następnie sprawdzamy warunki.

1.2.1 Jeżeli w węźle pozostały niesprawdzone ścieżki

Spośród niesprawdzonych ścieżek wybieramy losowo tę, w którą wejdziemy. Przesuwamy się na wybranego sąsiada, po czym:

- ustawiamy węzeł z którego przyszliśmy jako jego poprzednika,
- oznaczamy go jako odwiedzonego,
- ustawiamy jego odległość od startu jako odległość poprzednika + 1,
- sprawdzamy czy nie należy usunąć któregoś ze skrzyżowań z listy,
- sprawdzamy czy nie należy poprawić odległości korzystając z `relaxator`.

Wywołujemy `fixRelaxation` gdy któryś z sąsiadów węzła ma mniejszą odległość niż węzeł który sprawdzamy. Może to nastąpić na przykład w sytuacji gdy labirynt zawiera rondo. Metoda `fixRelaxation` poprawia odległości wszystkich węzłów sąsiadujących aż nie będą poprawne.
Po każdym przesunięciu się do nowego węzła, ustawiamy dzięki zapytaniu do serwera jego sąsiadów w strukturze.

1.2.2 Jeżeli węzeł jest ślepym zaułkiem

Cofamy się do ostatniego skrzyżowania z listy `junctions`. Do dotarcia do skrzyżowania korzystamy z wartości odległości zawartej w każdym obiekcie klasy `Node`. Przesuwamy się na węzeł o odległości najbliższej do poszukiwanego skrzyżowania, aż na nie nie trafimy.

1.2.3 Warunek specjalny

Jeżeli na skutek cofania się do ostatniego skrzyżowania trafimy na węzeł o odpowiedniej odległości, który nie jest jednak poszukiwanym przez nas skrzyżowaniem, uruchamiamy ten warunek. Cofamy się po poprzednikach węzła w którym się znajdujemy, aż nie trafimy na poszukiwane skrzyżowanie.

2 Pozostałe elementy programu

2.1 Main

Klasa sterująca, tutaj podajemy labirynt do mapowania oraz plik do zapisu.

2.2 MazeWriter

Klasa odpowiadająca za zapisywanie zmapowanego labiryntu do pliku tekstowego o zadanym formacie.

2.3 HttpConnector

Klasa odpowiadająca za wykonywanie poleceń HTTP.

2.4 RequestParser

Klasa tłumacząca odpowiedzi oraz zapytania HTTP na format potrzebny w programie.