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 udostępnionej nam aplikacji internetowej. Algorytm został wymyślony przez naszą grupę bazując na różnych poznanych przez nas metodach. Jest połączeniem kilku znanych algorytmów i naszych pomysłów na ich optymalizację. Algorytm operuje na strukturze `Maze` składającej się z tablicy dwuwymiarowej obiektów `Node`. Obiekty `Node` posiadają:

- informację o swoich sąsiadach,
- odległość od węzła startowego,
- wskaźnik na swojego poprzednika,
- pole informujące czy węzeł został odwiedzony,

- swoje współrzędne.

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` startowym wpisujemy dystans równy 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:

- oznaczamy go jako odwiedzonego,
- ustawiamy węzeł z którego przyszliśmy jako jego poprzednika,
- ustawiamy jego odległość od startu jako odległość poprzednika + 1,
- sprawdzamy, czy nie należy usunąć któregoś ze skrzyżowań z listy (czy pole, na które weszliśmy, nie jest nieodwiedzoną ścieżką w znalezionym już skrzyżowaniu),
- sprawdzamy czy nie należy poprawić odległości do węzła startowego korzystając z `relaxator`.

Wywołujemy `fixRelaxation`, gdy któryś z sąsiadów węzła ma mniejszą zmienną `distance` niż węzeł, na którym się znajdujemy. Może to nastąpić na przykład w sytuacji, gdy labirynt zawiera rondo. Metoda `fixRelaxation` poprawia odległości wszystkich węzłów sąsiadujących, aż nie będą poprawne (tzn. aż będą

wskazywać najmniejszą liczbę kroków potrzebnych do dotarcia do tego węzła ze startu).
Po każdym przesunięciu się do nowego węzła ustawiamy jego sąsiadów w strukturze wykorzystując odpowiednie zapytania aplikacji.

1.2.2 Jeżeli węzeł jest ślepym zaułkiem

Kierujemy się do ostatnio odwiedzonego skrzyżowania. Wykorzystujemy tutaj zmienną `distance`, aby jak najszybciej do niego dotrzeć. W tym celu wyliczamy różnicę pomiędzy `distance` ostatnio odwiedzonego skrzyżowania, a polem `distance` każdego z sąsiadów pola na którym się znajdujemy. Wybieramy wartość minimalną, zapisujemy sąsiada dla którego wystąpiła i przesuwamy się w jego kierunku. Postępujemy tak, aż dotrzemy do pola o takim samym `distance` jak ostatnie skrzyżowanie.

1.2.3 Wysłanie "giermka"

Aby uniknąć sytuacji, w której po wykonaniu algorytmu z poprzedniego podpunktu znajdziemy się na innym polu, niż ostatnio odwiedzone skrzyżowanie (sprawdzamy to za pomocą współrzędnych), a tym samym marnując kilka ruchów, tworzymy pomocnika `squire`. Jego zadaniem jest, aby wykonać wcześniej algorytm z poprzedniego podpunktu na zapisanej strukturze w programie, aby po dotarciu na miejsce ocenić, czy wykorzystując ten algorytm trafimy na odpowiedni węzeł. Jeżeli wszystko się zgadza, wykonujemy podpunkt 1.2.2 na aplikacji. Jeżeli to nie ten węzeł, uruchamiany jest warunek specjalny.

1.2.4 Warunek specjalny

Zostaje wykonany, jeżeli na skutek cofania się do ostatniego skrzyżowania trafimy na węzeł o odpowiedniej odległości, który nie jest jednak poszukiwanym przez nas skrzyżowaniem. Cofamy się wtedy po poprzednikach węzłów, począwszy od tego, na 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.