

AiSD - laboratorium

Symulator Transportu Pacjentów - specyfikacja implementacyjna

Kacper Baczyński, Michał Kiełczykowski, Marek Knosala,
Edward Sucharda

18 grudnia 2020

1 Wstęp

Niniejszy dokument jest ściśle powiązany z dokumentem dotyczącym dokumentacji funkcjonalnej projektu zespołowego z przedmiotu Algorytmy i Struktury Danych w roku akademickim 2020/2021 na Wydziale Elektrycznym Politechniki Warszawskiej. Zawiera opis implementacyjny algorytmu wykorzystanego do rozwiązania problemu postawionego w projekcie. Aby nie powielać informacji ogólnych dotyczących projektu zalecane jest zapoznanie, ze wspomnianym dokumentem, gdyż znajduje się w nim dokładny opis i założenia projektu.

2 Opis struktury projektu

2.1 Założenia wstępne

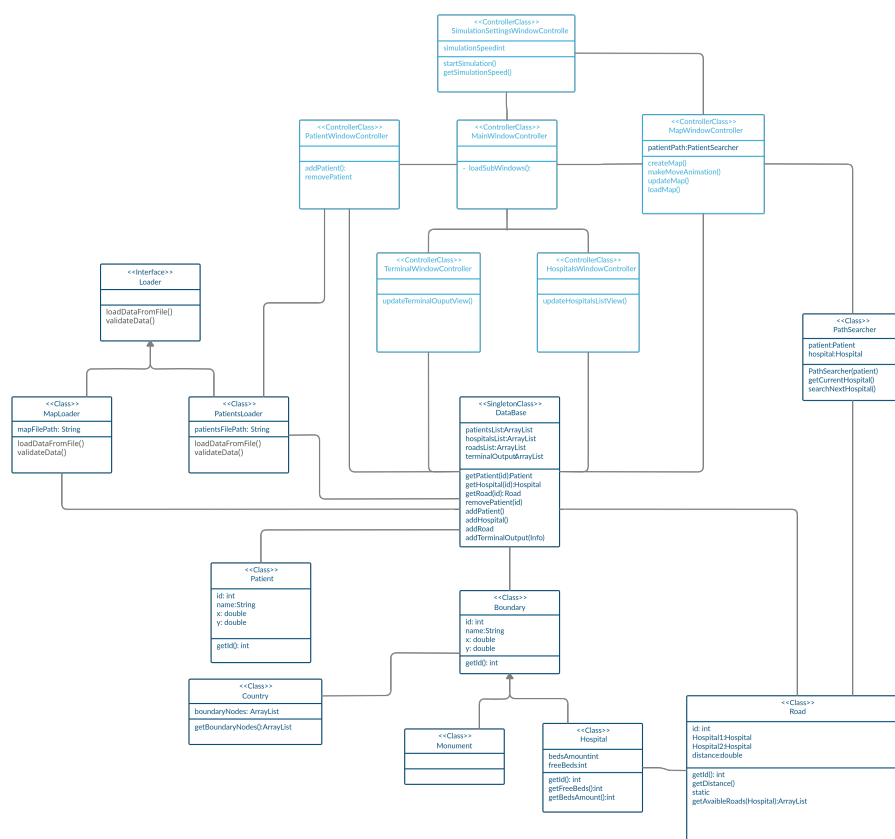
W celu realizacji problemu należało rozwiązać kwestie sprawdzenia poprawności danych, stworzenia grafu połączeń pomiędzy szpitalami, uwzględniając możliwe skrzyżowania, a następnie kwestie odpowiedniego przydziału pacjenta do najbliższego szpitala oraz ewentualne jego przemieszczenia w przypadku braku wolnego łóżka. Dodatkowo w programie należało rozwiązać kwestie graficznej prezentacji danych wejściowych oraz wyników działania algorytmu w formie czytelnej dla użytkownika.

2.2 Wykorzystane technologie

W celu zrealizowania założeń projektu postanowiono wykorzystać język programowania Java (OpenJDK w wersji 11 lub adekwatnej wersji OracleJDK). Jego dużą zaletą jest fakt możliwości uruchamiania na większości dostępnych obecnie systemów operacyjnych. Kolejną zaletą jest bardzo dobre wsparcie programowania współbieżnego oraz spora ilość elementów (np. kontenerów danych) zaimplementowanych przez autorów języka, co znacząco przyspiesza wykonanie projektu oraz zmniejsza podatność na błędy implementacyjne. Dodatkowo do projektu postanowiono zastosować interfejs graficzny (ang. Graphical user

interface - GUI) wykorzystując do tego celu środowisko JavaFX (w wersji co najmniej 11). Zaletą tego środowiska jest możliwość sporej separacji kodu źródłowego napisanego w języku Java realizującego mechanikę działania GUI oraz cel projektu od warstwy wizualnej interfejsu graficznego. Pozwala to na lepszą współpracę w projekcie zespołowym, gdzie dane osoby mogą realizować różne części projektu minimalizując kolizje spowodowane pracą dwóch osób nad jednym elementem projektu.

2.3 Diagram klas



Rysunek 1: Diagram klas.

Powyższy diagram prezentuje pomysł na zrealizowanie żądanej funkcjonalności. W diagramie widoczne jest wyraźne rozdzielenie na klasy dotyczące reprezentacji graficznej oraz kodu, który wykonuje algorytmy.

Na samej górze, kolorem jasnoniebieskim zostały zaznaczone klasy kontrolerów okien JavyFX. Struktura opiera się na głównym oknie MainWindows, do którego następnie ładowane są kolejne okna. Każde okno reprezentuje funkcjonalność okna przedstawioną w specyfikacji funkcjonalnej.

- PatientWindowController - odpowiedzialny za dodawanie pacjentów do symulacji (ręcznie i z pliku)

- MapWindowController - odpowiedzialny za wyświetlenie mapy oraz operacje wizualizacyjne w symulacji. Wykorzystuje do swoich celów algorytm znajdujący kolejny szpital, do którego musi zostać przesłany pacjent.
- TerminalWindowController - odpowiedzialny za przetrzymywanie i pokazywanie komunikatów programu.
- HospitalsWindowController - odpowiedzialny za reprezentację stanu każdego szpitala i pokazywanie ich zmian.
- SimulationSettingsWindowController - odpowiedzialny za wskazanie parametrów symulacji np. prędkość jej postępowania.
- MainWindowController - główne okno programu. Zarządza wszystkimi pozostałymi oknami oraz przekazuje referencje i zmiany stanów pomiędzy oknami.

W dolnej części diagramu przedstawione zostały wyszczególnione klasy związane z wykonywaniem algorytmów. Głównym ich zadaniem jest realizacja funkcjonalności projektu. Można je podzielić na klasy reprezentujące dane oraz funkcjonalności programu. Klasy reprezentujące dane:

- Patient - klasa reprezentująca pacjenta. Posiada współrzędne kartezjańskie, identyfikator oraz nazwę.
- Road - klasa reprezentująca połączenia między szpitalami. Wyposażona w referencje do szpitali, do których jest w stanie doprowadzić, identyfikator oraz czas, potrzebny na jej pokonanie.
- Boundary - klasa reprezentująca obiekty wchodzące w skład granicy państwa. Zawiera współrzędne kartezjańskie, identyfikator oraz nazwę.
- Hospital - klasa reprezentująca szpital, pochodna Boundary, ponieważ może wyznaczać granicę państwa. Posiada dodatkowo informację o ilości wszystkich i wolnych łóżek w szpitalu.
- Monument - klasa reprezentująca pomnik. Stworzona w celu odróżnienia szpitala od pomnika.
- Country - klasa reprezentująca państwo. Zawiera listę obiektów, które wyznaczają granice państwa.

Poniżej zostały przedstawione klasy odpowiedzialne za operacje algorytmiczne:

- DataBase - klasa przetrzymująca dane. Jest to klasa singleton, która będzie dostarczała dane do wszystkich klas. Taka realizacja pozwoli na spójny dostęp do danych.
- PathSearcher - klasa znajdująca szpitale, do których będziemy chcieli przewozić pacjentów. Jej wynik będzie dostarczany do kontrolerów okien w celu aktualizacji postępu symulacji.

- Interfejs Loader - interfejs ujednolicing sposób tworzenia pacjentów oraz mapy z listy. Pozwoli na generyczne pobieranie danych z plików i zapewni zdolności uogólniające kodu.
- MapLoader - klasa implementująca interfejs Loader. Głównym zadaniem jest wczytanie mapy z pliku.
- MapLoader - klasa implementująca interfejs Loader. Głównym zadaniem jest wczytanie pacjentów z pliku.

Takie zaprojektowanie systemu pozwoli na oddzielenie warstwy kontroli okien od warstwy obliczeniowej. Dodatkowo powstały interfejs pozwoli na zwiększenie uogólnienia zastosowania kodu i ograniczenie powielania kodu. Dziedziczenie obiektów, pozwoli na łatwiejsze sprawdzenie granic państwa, bez rozróżnienia, czy obiekt jest szpitalem czy pomnikiem.

3 Problemy Algorytmiczne

Całe zadanie symulacji transportu pacjenta do szpitali można rozbić na pomniejszych problemy. Rozwiązując je sekwencyjnie uzyskać można rzeczywiste rozwiązanie problemu przewozu pacjentów przy pomocy karet do szpitali. Poniżej znajdują się opisy wydzielonych problemów programistycznych, których rozwiązanie pozwoli na otrzymanie końcowego wyniku.

3.1 Przedstawienie mapy państwa

Pierwszym krokiem programu jest wczytanie środowiska, w którym należy rozwiązać symulację. Granice państwa muszą być figurą wypukłą opartą na najbardziej zewnętrznych punktach. Mimo tego nie wszystkie zewnętrzne punkty wyznaczają granice państwa, ponieważ połączenie niektórych z nich powoduje powstanie bryły niewypukłej.

W tym celu zastosowany zostanie poniższy algorytm bazujący na problemie "Convex Hull" w implementacji opracowanej przez jednego z członków zespołu. Wierzchołkami przedstawionego poniżej grafu są szpitale oraz pomniki, których jedyną funkcją to poszerzenie granic państwa. Poniżej znajduje się opis algorytmu:

1. Szukaj wierzchołka o najmniejszej współrzędnej x .
2. Dodaj go do nowo utworzonej listy wierzchołków.
3. Stwórz początkowo pustą listę współczynników kierunkowych, która będzie gromadzić nachylenie prostej przechodzącej przez wybrane dwa wierzchołki grafu.
4. Szukaj kolejnego wierzchołka o najmniejszej możliwej współrzędnej x (może to być również wierzchołek, który ma taką samą współrzędną w osi x o ile ma większą wartość w osi y).
5. Po znalezieniu wierzchołka wyznacz współczynnik kierunkowy prostej łączącej go i poprzedni punkt. Jeśli jest on mniejszy od ostatniego elementu na liście współczynników to dodaj ten wierzchołek do listy wierzchołków,

a współczynnik do listy współczynników. Jeśli lista współczynników jest pusta to warunek jest zawsze spełniony. Jeżeli warunek nie jest spełniony to usuń z obu list ostatni element i sprawdź analogiczny warunek dla nowych ostatnich elementów obu list. Jeśli warunek jest spełniony to dodaj elementy do listy, jeśli nie to ponownie usuwaj tak długo aż warunek będzie spełniony.

6. Kroki od 3 do 5 powtarzaj tak długo aż zostaną sprawdzone wszystkie wierzchołki - czy nie ma nowego wierzchołka o współrzędnej y większej niż ostatni z listy.
7. Wykonaj kroki 1-6 z tym, że:
 - a) zacznij od ostatniego punktu z listy wierzchołków z punktów 1-6,
 - b) szukaj kolejnych wierzchołków malejąco wzdłuż osi y takich, żeby ich współrzędne x były coraz większe,
 - c) postępuj analogicznie jak w punkcie 5.
8. Wykonaj kroki 1-6 z tym, że:
 - a) zacznij od ostatniego punktu z listy wierzchołków z punktu 7,
 - b) szukaj kolejnych wierzchołków malejąco wzdłuż osi x takich, żeby ich współrzędne y były coraz mniejsze,
 - c) postępuj jak w punkcie 5.
9. Wykonaj kroki 1-6 z tym, że:
 - a) zacznij od ostatniego punktu z listy wierzchołków z punktu 8,
 - b) szukaj kolejnych wierzchołków rosnąco wzdłuż osi y takich, żeby ich współrzędne x były coraz mniejsze,
 - c) postępuj analogicznie jak w punkcie 5.

3.2 Znalezienie skrzyżowań

W postawionym zadaniu istnieje założenie, że jeżeli drogi przecinają się to w miejscu przecięcia powstaje skrzyżowanie. Powoduje to, że od pewnego szpitala do innego szpitala można dojechać okrężną, lecz szybszą drogą, mimo że nie istnieje ona w pliku wejściowym. Biorąc pod uwagę, że wszystkie obiekty mapy posiadają współrzędne kartezjańskie, punkty przecięć można wyznaczyć w sposób czysto matematyczny. Przedstawiając drogę od szpitala do szpitala jako odcinek, można przedstawić wszystkie drogi jako odcinki i znaleźć ich punkty przecięcia. Jest to metoda naiwna, ponieważ wymaga ona przeanalizowania każdej drogi z każdą inną drogą ze zbioru dróg. O ile dla jednej pary złożoność obliczeniowa to $O(1)$, tak dla n dróg jest to już $O(n^2)$.

Istnieje także bardziej przemyślany algorytm, który poprzez "omiecienie wiązką" przez wszystkie odcinki jest w stanie wykryć ich punkty przecięć. Implementacja algorytmu Bentley–Ottmann wymaga przedstawienia dróg jako posortowanych punktów względem osi X oraz odcinków. Algorytm można przedstawić następująco:

- Pionowa linia "przemiatą" wszystkie punkty od lewej do prawej.

- Po natknięciu się na lewy(początkowy) punkt odcinka, odcinek oznaczany jest jako aktywny.
- Następnie sprawdzane są przecięcia z najbliższymi, aktywnymi odcinkami powyżej i poniżej aktualnego odcinka.
- Jeżeli odcinki aktywne się przecinają wtedy jest znajdowany i zapamiętywany punkt przecięcia między tymi odcinkami.
- W momencie gdy pionowa linia napotka końcowy punkt odcinka, wtedy dezaktywuje ona odcinek. Sprawia to, że odcinek nie jest brany dalej do analizy przecięć z innymi odcinkami.

Taki sposób pozwala na ograniczenie zbioru sprawdzanych odcinków do sąsiedztwa potencjalnie przecinających się odcinków. Dzięki temu nie trzeba sprawdzać każdej pary odcinków ze sobą, co skutkuje złożonością obliczeniową rzędu $O((n+k)\log n)$, gdzie n to liczba odcinków, a k liczba przecięć.

Rozwijając ten algorytm, punkty przecięć zostaną skrzyżowaniami z punktu widzenia działania programu, a drogi na przecięciach zostaną zmodyfikowane w zależności od tego w jakiej proporcji punkt podzielił odcinki.

3.3 Określenie czy pacjent znajduje się na terytorium państwa

Założeniem projektu jest obsługa pacjentów znajdujących się w obszarze państwa. Jeżeli pacjent znajdujący się poza granicami państwa wymaga pomocy, wtedy karetka nie jest w stanie jej udzielić. Algorytm wygląda następująco:

1. Dla pacjenta o współrzędnych (x_p, y_p) i każdego boku mapy państwa (wielokąta wypukłego) $M = \{W_1 = (x_1, y_1), \dots, W_n = (x_n, y_n)\}$, oblicz i sprawdź znak:

$$h = (y_{i+1} - y_i) * (x_p - x_i) - (x_{i+1} - x_i) * (y_p - y_i)$$

2. Jeśli h dla wszystkich boków mapy ma taki sam znak, to pacjent znajduje się na terytorium państwa. Jeśli nie, to pacjent znajduje się poza granicami państwa.

3.4 Transport pacjenta do najbliższego szpitala

Transport pacjenta jest to szereg operacji mających na celu doprowadzić pacjenta do szpitala, w którym może zostać mu udzielona pomoc. Składa się on z operacji przewiezienia pacjenta do najbliższego szpitala, a jeżeli nie będzie w nim miejsca, znalezienie takiego szpitala, który jest w stanie tą pomoc zaoferować. Poszukiwanie szpitali odbywa się poprzez przewożenia pacjenta do szpitala i dopiero sprawdzania czy są w nim miejsca. Algorytm dostarczenia pacjenta do placówki prezentuje się następująco:

1. Oblicz odległość pacjenta do wszystkich szpitali wg wzoru:

$$d_i = \sqrt{(x_{si} - x_p)^2 + (y_{si} - y_p)^2}$$

2. Wybierz szpital, do którego odległość d jest najmniejsza
3. Udać się do wybranego szpitala i po dojeździe (gdy pozycja pacjenta=pozycji szpitala) sprawdzić czy w wybranym szpitalu jest wolne łóżko. Jeśli TAK – KONIEC transportu pacjenta, zostaw pacjenta w szpitalu. Jeśli NIE - oznacz szpital jako odwiedzony i przejdź do punktu 4.
4. Dla szpitala, w którym obecnie znajduje się pacjent wyznacz inny szpital, do którego droga jest najkrótsza:
 - 4.1. Dla każdego szpitala i wierzchołka bezpośrednio połączonego z obecnym szpitalem (skrzyżowania) wyznacz długości połączeń i zapamiętaj je.
 - 4.2. Zawsze jeśli sprawdziłeś już jakiś wierzchołek zapamiętaj ten fakt (zapamiętaj, że udało się do niego wyznaczyć jakąkolwiek drogę).
 - 4.3. Jeśli w danym szpitalu nie było jeszcze karetki, to zakończ przeszukiwanie ścieżek przechodzących przez ten szpital.
 - 4.4. Jeśli w danym miejscu była już karetka, to przeszukuj dalej graf, zapamiętując sumaryczną drogę i trasę do każdego wierzchołka.
 - 4.5. Kończ przeszukiwanie danej ścieżki, gdy dotrzesz do wierzchołka, który był już sprawdzony.
 - 4.6. Po przeszukaniu całego grafu z obecnego szpitala, wybierz nieodwiedzony szpital, do którego sumaryczna droga jest najkrótsza. Z tak wybranym szpitalem przejdź do punktu 3.

4 Testy oprogramowania

Do testowania oprogramowania użyte będzie narzędzie JUnit.

Dla plików wejściowych - pliku z mapą oraz pliku z pacjentami przeprowadzone zostaną następujące testy:

1. Pusty plik.
2. Pusta sekcja.
3. Brak tytułu sekcji (znaku "#").
4. Nieprawidłowa liczba znaków strumienia "|".
5. Nieprawidłowy typ danych.
6. Niepoprawne wartości danych - niedopuszczalna wartość zerowa, ujemna lub wartość spoza zakresu danego typu danych.
7. Liczba wolnych łóżek większa niż liczba wszystkich łóżek szpitala w sekcji "Szpitale".
8. Powtarzające się id lub para id (sekcja "Drogi").
9. Wykorzystanie nieistniejącego id szpitala w sekcji "Drogi".

Inne (niedotyczące plików wejściowych) testy programu:

1. Próby wywołania funkcji z niepoprawnym parametrem (np dodanie do klasy DataBase obiektu null)
2. Próby odwoływania się do poprawnych typów i zakresów danych ale nieistniejących (np odwoływanie się do pacjenta o nieistniejącym id)
3. Testy algortmu dla kilku testowych danych i porównanie ich z poprawnym wyjściem określonym wcześniej (np testy na klasie PathSearcher)
4. Testy GUI pod kątem przypadków szczególnych (np brak granic państwa czy pusta lista szpitali)

5 Źródła

- [1] http://0x80.pl/articles/point_in_polygon.html
(data i godzina dostępu: 16.12.2020 21:39)
- [2] https://eduinf.waw.pl/inf/utils/011_2011/0105.php
(data i godzina dostępu: 17.12.2020 15:21)
- [3] <https://www.geeksforgeeks.org/convex-hull-set-2-graham-scan/>
(data i godzina dostępu: 17.12.2020 13:36)
- [4] https://en.wikipedia.org/wiki/Bentley-Ottmann_algorithm
(data i godzina dostępu: 17.12.2020 11:05)