Politechnika Warszawska
Wydział Elektryczny
Kierunek Informatyka

Specyfikacja Implementacyjna

Wykonali: Piotr Jeleniewicz, Aliaksandr Karolik

Warszawa, 24.12.2018

Spis treści

1 Wprowadzenie

Program będzie umożliwiał wyznaczenie optymalnych obszarów terenu określonego przez wcześniej podane punkty w pliku wejściowym. Optymalność obszarów, będzie określana na podstawie odległości do najbliższych punktów kluczowych, również określonych w pliku wejściowym. Aplikacja będzie także umożliwiać dodawanie obiektów należących do danego obszaru, a później wyświetlać statystki związane z obiektami wybranego obszaru.

Program będzie tworzony w języku JAVA i testowany w środowisku:

- IntelliJ IDEA 2018.2.5

Sprzęt, na którym program będzie testowany:

- Intel Core i5-7300HQ 2,5 GHz
- Pamięć RAM o pojemności 8 GB
- System Ubuntu 18.04.1

2 Opis algorytmu

Przy realizacji programu obecnie brane są pod uwagę 3 algorytmy. Wszystkie będą sprowadzać problem wyznaczania obszarów do tworzenia *diagramu Voronoi*. Są to następujące algorytmy:

1. **Algorytm wyznaczania diagramu Voronoi na podstawie półpłaszczyzn** wyznaczanych przez symetralne do odcinków utworzonych przez połączenie każdych dwóch punktów kluczowych.
2. **Algorytm Fortune'a** służący do wyznaczenia diagramu Voronoi, za pomocą przesuwania prostej (tzw. miotły) oraz linii brzegowej.

3. **Algorytm wyznaczania diagramu Voronoi poprzez obliczenie odległości każdego punktu** od punktów kluczowych i przyporządkowanie go do obszaru odpowiadającego punktowi kluczowemu, do którego odległość była najmniejsza.

Obecnie rozważamy implementację programu na podstawie algorytmu wyznaczającego obszary w oparciu o odległości każdego punktu od wszystkich punktów kluczowych. W przypadku lepszego rozeznania problemu, bierzemy pod uwagę zaimplementowanie algorytmu Fortune'a lub algorytmu opierającego się na wyznaczaniu półpłaszczyzn. Jednakże na chwilę obecną, po zapoznaniu się z problemem i przeanalizowaniu wyżej wymienionych algorytmów, oceniamy, że jedynym algorytmem, który jesteśmy w stanie zaimplementować na chwilę obecną jest algorytm przedstawiony jako 3.

Algorytm, który najprawdopodobniej zostanie zaimplementowany w programie do wyznaczania obszarów, będzie miał postać:

1. Utworzenie tablicy punktów symbolizującą cały badany obszar.
2. Dla każdego punktu zostanie wyznaczona odległość do wszystkich punktów kluczowych. Jednocześnie podczas wyznaczania tych odległości, zostanie znaleziony punkt kluczowy, do którego odległość jest najmniejsza.
3. Badanemu punktowi zostanie przypisany obszar odpowiadający najbliższemu punktowi kluczowemu.

W przypadku usunięcia danego punktu kluczowego, obszary zostaną na nowo przypisane tylko tym punktom i obiektom, które znajdowały się w danym obszarze.

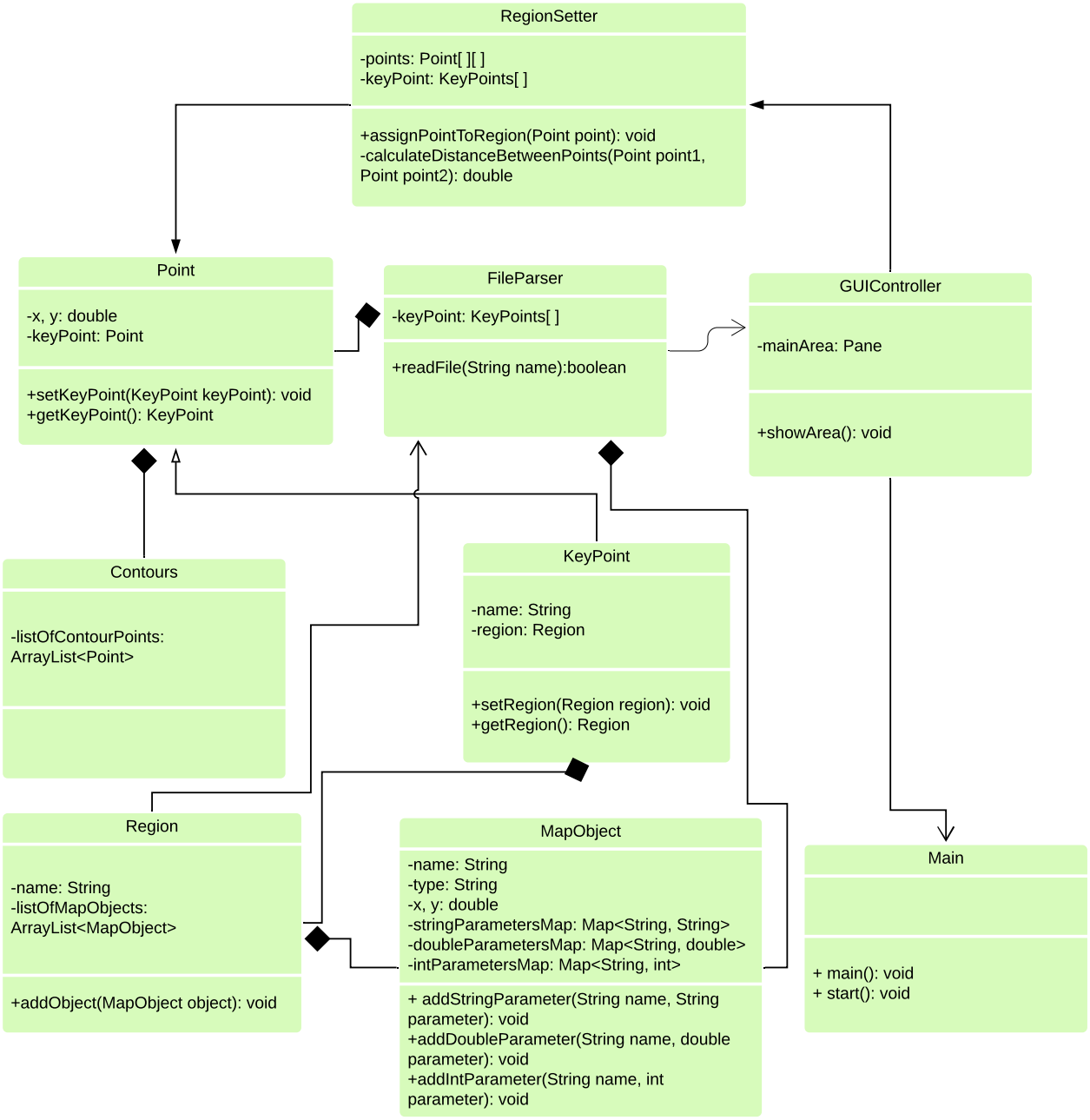
W przypadku dodania punktu kluczowego, obszary zostaną przypisane na nowo dla punktów obszaru, w którym umieszczono nowy punkt kluczowy oraz dla wszystkich obszarów sąsiadujących z tym obszarem.

Każdy umieszczany obiekt, zostanie przypisany do obszaru na podstawie punktu, w którym zostanie umieszczony.

Każdy obszar będzie posiadał listę obiektów, które znajdują się na jego obszarze.

Na tej podstawie generowane będą listy obiektów - zarówno listy wszystkich obiektów leżących na terenie danego obszaru jak i zbiorcze listy obiektów danego typu leżące w danym obszarze.

3 Diagram klas



RYS.01 DIAGRAM KLAS

4 Opis klas

4.1 Klasa Point

Klasa ta będzie przedstawieniem punktu, jako najmniejszego i niepodzielnego fragmentu badanego terenu. Będzie zawierać pola:

- `Double x;`

  Pole przechowujące współrzędną X punktu.

- `Double y;`

  Pole przechowujące współrzędną Y punktu.

- `KeyPoint keyPoint;`

  Referencja do pola kluczowego obszaru, w którym znajduje się dany punkt. W przypadku punktów kluczowych pole ma wartość null.

W klasie występować będą tylko typowe metody get i set.

4.2 Klasa KeyPoint

Klasa ta będzie przedstawieniem punktu kluczowego. Dziedziczy ona po klasie Point. Oprócz pól dziedziczonych po klasie Point, zawiera pola:

- `String name;`

  Pole przechowujące nazwę punktu kluczowego.

- `Region region;`

  Pole przechowujące referencje do obiektu opisującego obszar przypisany do punktu kluczowego.

- `KeyPoint keyPoint;`

  Referencja do pola kluczowego obszaru, w którym znajduje się dany punkt. W przypadku punktów kluczowych pole ma wartość null.

W klasie występować będą tylko typowe metody get i set.

4.3 Klasa Contours

Klasa ta będzie zawierała elementy opisujące kontury badanego terenu. Będzie posiadać jedno pole:

- `ArrayList<Point> listOfContourPoints`

  Lista zawierająca wszystkie punkty określające kontury.

W klasie występować będą tylko typowe metody get i set.

4.4 Klasa Region

Klasa opisująca obszar przypisany do punktu kluczowego.

- `String name;`

  Pole przechowujące nazwę regionu.

- `ArrayList<MapObject> listOfMapObjects;`

  Lista zawierające wszystkie obiekty znajdujące się na danym obszarze.

W klasie oprócz metod get i set znajdować się będzie metoda:

- `void addObject(MapObject object);`

  Metoda dodająca obiekt leżący na mapie do listy obiektów danego obszaru.

4.5 Klasa RegionSetter

Klasa służąca do przypisywania punktów terenu do odpowiednich obszarów. Klasa ta pole:

- `Point[][] points;`

  Tablica wszystkich punktów mapy.

- `KeyPoint[] keyPoints;`

  Tablica zawierająca punkty kluczowe.

Posiada metody:

- `void assignPointToRegion(Point point);`

  Metoda przypisująca dany punkt do odpowiedniego regionu za pomocą przypisania mu odpowiedniego punktu kluczowego.

- `void calculateDistanceBetweenPoints(Point point1, Point point2);`

  Metoda obliczająca odległość między dwoma dowolnymi punktami.

4.6 Klasa MapObject

Klasa służąca do tworzenia obiektów przedstawiających obiekty możliwe do umieszczenia na mapie.

- `String name;`

  Pole przechowujące nazwę obiektu.

- `String type;`

  Pole przechowujące typ obiektu.

- `Double x;`

  Pole przechowujące współrzędną X punktu.

- `Double y;`

  Pole przechowujące współrzędną Y punktu.

- `Map<String, String> stringParametersMap;`

  Mapa przechowująca parametry obiektu typu String.

- `Map<String, double> doubleParametersMap;`

  Mapa przechowująca parametry obiektu typu double.

- `Map<String, int> intParametersMap;`

  Mapa przechowująca parametry obiektu typu integer.

W klasie oprócz metod get i set znajdować się będą metody:

- `void addStringParameter(String name, String parameter);`

  Metoda dodająca parametr typu String do obiektu.

- `void addDoubleParameter(String name, double parameter);`

  Metoda dodająca parametr typu double do obiektu.

- `void addIntParameter(String name, int parameter);`

  Metoda dodająca parametr typu integer do obiektu.

4.7 Klasa GUIController

Klasa będącą kontrolerem interfejsu użytkownika. Będzie m.in. posiadać pola:

- `Pane mainArea;`

  Obszar interfejsu użytkownika, w którym wyświetlana będzie mapa terenu wraz z podziałem na obszary.

W klasie występować będzie m.in metoda:

- `void showArea(String name, String parameter);`

  Metoda pokazująca wygenerowaną wcześniej mapę terenu.

4.8 Klasa fileParser

Klasa przetwarzająca plik wejściowy na odpowiednie struktury danych. Będzie między innymi posiadać pole:

- `KeyPoint[] keyPoints;`

  Tablica zawierająca punkty kluczowe.

W klasie występować będzie m.in metoda:

- `boolean readFile(String name);`

  Metoda parsująca plik o podanej w argumencie *name* nazwie.

5 Testy jednostkowe

5.1 Klasa Point.

W tej klasie występują wyłącznie metody typu set i get, więc dla tych metod nie będą stworzone testy.

5.2 Klasa KeyPoint

W tej klasie występują wyłącznie metody typu set i get, więc dla tych metod nie będą stworzone testy.

5.3 Klasa Contours

W tej klasie występują wyłącznie metody typu set i get, więc dla tych metod nie będą stworzone testy.

5.4 Klasa Region

W tej klasie testom będzie podlegała następująca metoda:

- `void addObject(MapObject object);`

  Dla tej metody będą przeprowadzone następujące testy:

  1. Wywołanie metody z argumentem null. Oczekiwanym wynikiem wywołania metody z argumentem null jest to dodanie do listy o nazwie *listOfMapObjects* warości null jako element listy. Test zostanie zaakceptowany gdy elementem listy *listOfMapObjects* będzie null.
  2. Wywołanie metody z argumentem typu *MapObject*. Oczekiwanym wynikiem wywołania metody z danym argumentem jest dodanie do listy o nazwie *listOfMapObjects* obiektu o typie *MapObject*. Test zostanie zaakceptowany gdy elementem listy będzie obiekt podany jako argument do metody.

5.5 Klasa RegionSetter

Do przetestowania tej klasy będą stworzone punkty kluczowe jako obiekty klasy *KeyPoint* oraz punkty typu *Point*. W tej klasie będą przeprowadzone następujące testy:

- `void assignPointToRegion(Point point);`

  Test metody będzie polegał na sprawdzeniu poprawności przypisywania punktów do obszarów, które są zdefiniowane przez punkty kluczowe. Do przetestowania poprawności metody będą zbadane następujące przypadki:

  1. Punkt będzie posiadał równe odległości do punktów kluczowych które definiują obszary. Przykładowe punkty dla testu:
     - punkty kluczowe:
       `B (11,9), C(5,9)`

– punkt:

`A(8,6)`

Test zostanie zaakceptowany gdy punkt A zostanie przypisany do dwóch obszarów, co będzie oznaczać, że znajduje się on na granicy dwóch obszarów.

2. Punkt będzie posiadał różne odległości do punktów kluczowych. Przykładowe punkty dla testu:

   – punkty kluczowe:

   `B (11,9), C(4,10)`

   – punkt:

   `A(8,6)`

   Test zostanie zaakceptowany w przypadku gdy metoda poprawnie rozróżni do którego punktu kluczowego trzeba przepisać podany jako argument punkt. W danym przypadku punktem kluczowem do którego powinien być przypisany punkt A będzie punkt B.

- `void calculateDistanceBetweenPoints(Point point1, Point point2);`

  Test metody będzie polegał na sprawdzeniu poprawności wyliczenia odległości pomiędzy dwoma punktami. Przykładowe punkty dla testu:

  – `B (11,9), C(4,10)`

  Test zostanie zaakceptowany w przypadku gdy metoda obliczy poprawnie odległość pomiędzy punktami.

5.6 Klasa MapObject

W tej klasie będą przeprowadzone testy dla następujących metod:

- `void addStringParameter(String name, String parameter);`
- `void addDoubleParameter(String name, double parameter);`
- `void addIntParameter(String name, int parameter);`

  Test metod będą polegały na wywołaniu metod dla różnych parametrów oraz sprawdzeniu czy parametry zostały poprawnie dodane do map zawartych w klasie.

5.7 Klasa GUIController

Testy interfejsu graficznego zostaną przeprowadzone podczas implementowania interfejsu graficznego.

5.8 Klasa fileParser

W tej klasie będą przeprowadzone następujące testy:

- `boolean readFile(String name);`

  Dla tej metody będą przeprowadzone następujące testy:

  1. Wywołanie metody z argumentem null oraz dla zmiennej *name* o wartość " ". Test zostanie zaakceptowany gdy metoda zwróci *false*.
  2. Wywołanie metody dla zmiennej *name* o wartości, która nie jest poprawną ścieżką do pliku. Test zostanie zaakceptowany gdy metoda zwróci *false*.
  3. Wywołanie metody dla zmiennej *name*, która zawiera poprawną ścieżką do pliku. Test zostanie zaakceptowany gdy metoda zwróci *true*. Plik testowy będzie miał postać:

```
# Kontury terenu (wymienione w kolejności łączenia): Lp. x y
1. 0 0
2. 0 20
3. 20 30.5

# Punkty kluczowe: Lp. x y Nazwa
1. 1 1 KOK Krawczyka
2. 1 19 KOK Kaczmarskiego

# Definicje obiektów:
Lp. Typ_obiektu (Nazwa_zmiennej Typ_zmiennej)*
1. SZKOŁA Nazwa String X double Y double
2. DOM X double Y double L_MIESZKAŃCÓW int
3. NIEDŹWIEDŹ X double Y double

# Obiekty: Typ_obiektu (zgodnie z definicją)
1. SZKOŁA "Szkoła robienia dużych pieniędzy" 4 1
2. DOM 4 3 100
3. NIEDŹWIEDŹ 20 20
```

Po wykonaniu metody, do listy *listOfContorPoints* powinny trafić obiekty klasy *Point* o podanych współrzędnych z pliku. Dodatkowo będą stworzone będą odpowiednie obiekty klasy *KeyPoint*. Oczekiwane jest także stworzenie obiektów mapy klasy *MapObject* na podstawie danych z pliku wejściowego.

4. Wywołanie metody dla zmiennej *name* o wartości, która jest poprawną ścieżką do pliku. Plik jedna, będzie pusty, bądź będzie zawierać niekompletne lub niepoprawne dane. Test zostanie zaakceptowany gdy metoda zwróci *false*. Plik testowy będzie miał następującą postać:

```
# Kontury terenu (wymienione w kolejności łączenia): Lp. x y
1. 0 0

# Obiekty: Typ_obiektu (zgodnie z definicją)
1. SZKOŁA "Szkoła robienia dużych pieniędzy" 4 1
2. DOM 4 3 100
3. NIEDŹWIEDŹ 20 20
```