Politechnika Warszawska
Wydział Elektryczny

Specyfikacja Implementacyjna

Autor:
Grzegorz Kopyt
Arkadiusz Michalak

23 grudnia 2018

Spis treści

1 Wstęp teoretyczny

Dokument ten dotyczy programu realizowanego w ramach „Projektu Zespołowego 2018/2019".

Ma on za zadanie przedstawić problem rozważany w projekcie od strony czysto praktycznej. Zagadnienie podziału płaszczyzny na optymalne obszary doczekało się wielu koncepcji rozwiązań. Opisy algorytmów użytych w naszym rozwiązaniu znajdują się w **rozdziale 2**. Odnośnie potrzeb teorii warto usystematyzować używane pojęcia:

- kontur (zamiennie obszar konturu) - wielobok wypukły wyznaczony przez podane punkty, tylko ten fragment płaszczyzny będzie rozważany, punkty kluczowe i obiekty nie mogą istnieć poza tym konturem;
- obszar punktu kluczowego (obszar) - zawiera jeden punkt kluczowy, a granice wyznaczane są na zasadzie iż, każdy punkt wchodzący w skład obszaru musi mieć bliżej do jego punktu kluczowego niż dowolnego innego.

2 Opis algorytmu

2.1 Sprawdzanie wypukłości konturu

Wszystkie punkty konturu podane przez użytkownika przechowywane będą w klasie *Contour*.

Na podstawie algorytmu Jarvisa zostaną one podzielone na te, które wejdą w skład konturu oraz na te, które zostaną zignorowane. Punkty będą ignorowane, jeśli podany kontur nie będzie wypukły. Wtedy algorytm stworzy wypukły kontur, na podstawie podanych punktów, ignorując te, które uzna za burzące wypukły kształt figury. Punkty wchodzące w skład konturu zostaną zachowane w kolejności łączenia.
Algorytm będzie działał następująco:

1. Wybierze dwa punkty P i Q.

   P to punkt o najmniejszej współrzędnej Y (oraz X, jeśli więcej punktów ma tą samą Y). Q to punkt o największej współrzędnej Y (oraz X, jeśli więcej punktów ma tą samą Y).

2. Wyznaczy prawą część konturu:

   C - obecny punkt (początkowo P), N - następny punkt konturu

   (a) Znajdzie N, dla którego cosinus kąta między wektorem CN a *[1,0]* jest największy,
   (b) C staje się N, a N to kolejny punkt,
   (c) Jeśli $N = Q$ skończy iteracje.

   Powyższe instrukcje wykona w pętli.

3. Wyznaczy lewą część konturu:

   C - obecny punkt (początkowo Q), N - następny punkt konturu

   (a) Znajdzie N, dla którego cosinus kąta między wektorem CN a *[-1,0]* jest największy,
   (b) C staje się N a N to kolejny punkt,
   (c) Jeśli $N = P$ skończy iteracje.

   Powyższe instrukcje wykona w pętli.

2.2 Sprawdzenie czy punkt leży w konturze

Przy założeniach takich, że:

- kontur jest wielokątem wypukłym;
- punkt należy do wielokąta jeśli leży w dowolnym miejscu ograniczonym przez krawędzie wielokąta, nie poza krawędziami i nie na nich.

Przy powyższych założeniach stwierdzenie czy punkt leży wewnątrz konturu opiera się na sprawdzeniu czy półprosta wytyczona w dowolnym kierunku z tego punktu ma dokładnie jeden punkty przecięcia z liniami wyznaczającymi kontur. Dla ułatwienia przyjmiemy, że wytyczamy półprostą pionową w dół z tego punktu. Mając dwa punkty definiujące krawędź możemy wyznaczyć równanie kierunkowe prostej która zawiera tą krawędź a następnie rozwiązując układ dwóch równań liniowych określić czy ma punkt przecięcia z daną krawędzią. Proces ten należy powtórzyć dla wszystkich krawędzi, lub do znalezienia przecięcia.

2.3 Wyznaczanie optymalnych obszarów

2.3.1 Algorytm Fortune'a

Do wyznaczenia optymalnych obszarów zastosowany zostanie algorytm Fortune'a, który wyznaczy diagram Voronoi. Obszary powstaną na podstawie zbioru punktów, uporządkowanych według współrzędnej y (malejąco). Diagram Voronoi zapewni nam podział terenu na optymalne części, zawierające po jednym punkcie kluczowym. Algorytm Fortune'a ma kilka podstawowych pojęć:

1. Miotła (*sweep line*)

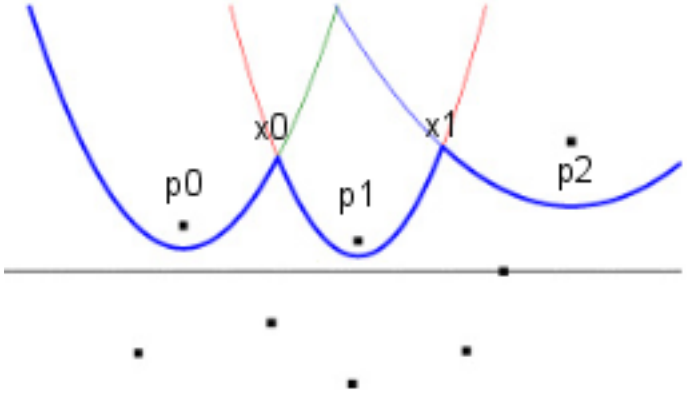
   Jest to abstrakcyjna nazwa, którą określamy prostą równoległą do osi x w kartezjańskim układzie współrzędnych. Prosta ta początkowo znajduje się na wysokości współrzędnej y większej niż y „najwyższego" punktu z naszego zbioru. Miotła porusza się w dół i napotyka kolejne punkty z naszego zbioru.

2. Zdarzenie punktowe (*site event*)

   Jest to sytuacja, w której miotła napotyka punkt. Powoduje to powstanie paraboli o skupieniu w danym punkcie, która przyczyni się do określenia optymalnych obszarów oraz wpłynie na kształt linii brzegowej.

3. Linia brzegowa (*beach line*)

   Jest to linia (niebieska), która jest dolną granicą naszego powstającego diagramu. Jest modyfikowana na podstawie pozycji miotły oraz napotkanych przez nią punktów.



   (źródło obrazu: http://blog.ivank.net)

4. Zdarzenie okręgu (*circle event*)

   Jest to sytuacja, w której zanika jedna z paraboli linii brzegowej, „wyparta" przez dwie inne parabole. Miejsce jej ostatecznego zniknięcia będące jednocześnie miejscem spotkania tych dwóch paraboli jest kolejnym punktem diagramu Voronoi.

Szczegółowy opis pracy algorytmu Fortune'a znajduje się w 7. rozdziale książki *Computational Geometry: Algorithms and Applications*.

Animacja pracy algorytmu znajduje się pod tym linkiem: `https://www.youtube.com/watch?v=rvmREoyL2F0`

2.3.2 Struktura danych

Przy implementacji tego algorytmu wykorzystane zostaną następujące struktury danych:

- kolejka priorytetowa:

  Przechowywać będzie zdarzenia jakie napotka miotła i na ich podstawie modyfikowana będzie linia brzegowa, a docelowo diagram Voronoi.

- drzewo binarne:

  Przechowywać będzie linie brzegową oraz krawędzie diagramu. W jego liściach znajdować się będą łuki (parabole) linii brzegowej, a wewnętrzne węzły przechowywać będą informacje o krawędziach. Drzewo to posłuży jako podstawa do tworzenia diagramu.

  Obiekty należące do tego drzewa będą miały dwie tożsamości będą mogły być:

  – przecięciem (reprezentując punkt przecięcia dwóch łuków/paraboli);
  – łukiem/parabolą (reprezentując część linii brzegowej będącą kawałkiem paraboli o skupieniu w konkretnym punkcie kluczowym).

  Taki zabieg usprawni modyfikacje struktury drzewa.

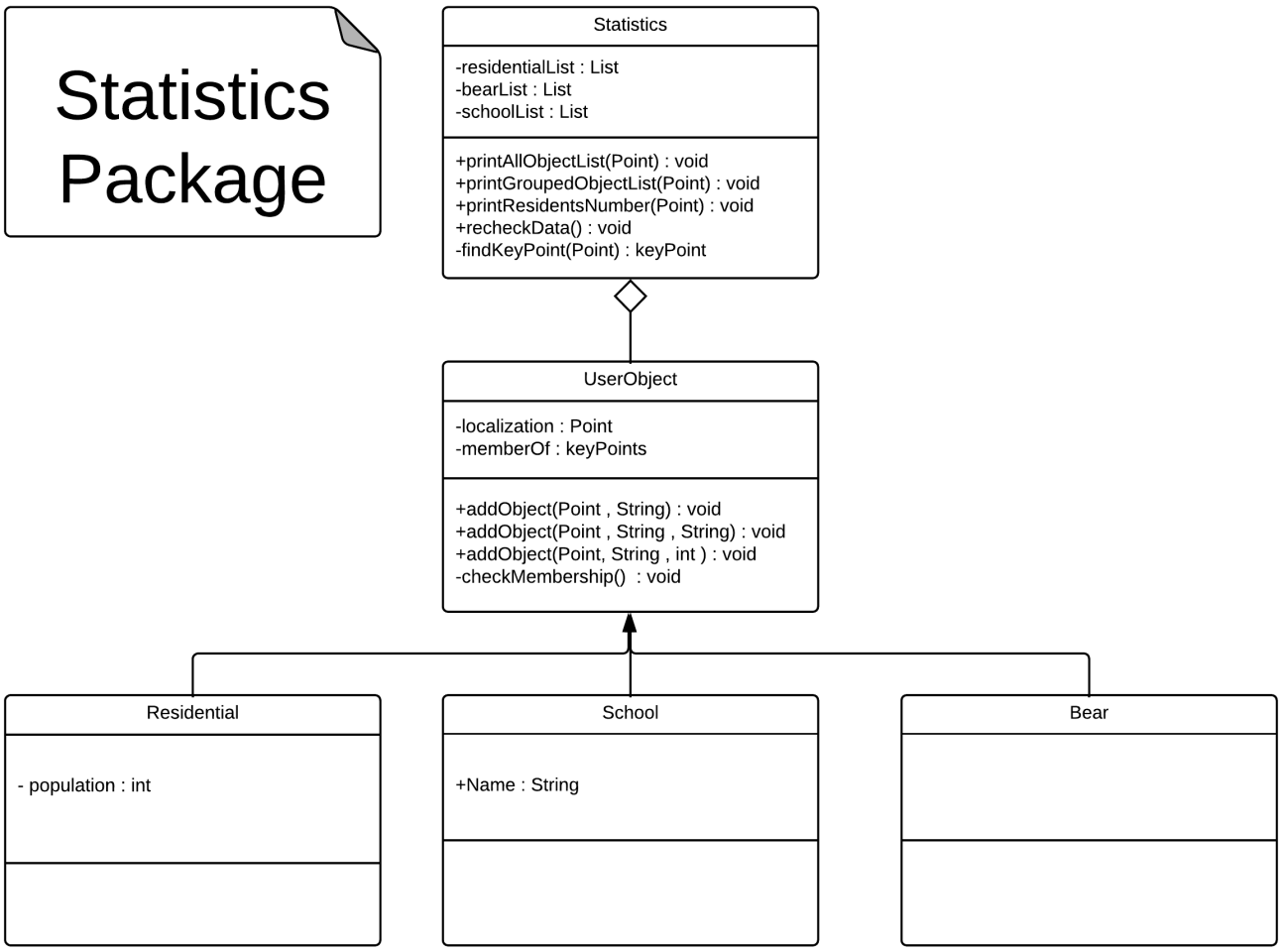
- Lista jednokierunkowa:

  Przechowywane w niej będą kompletne krawędzie diagramu Voronoi.

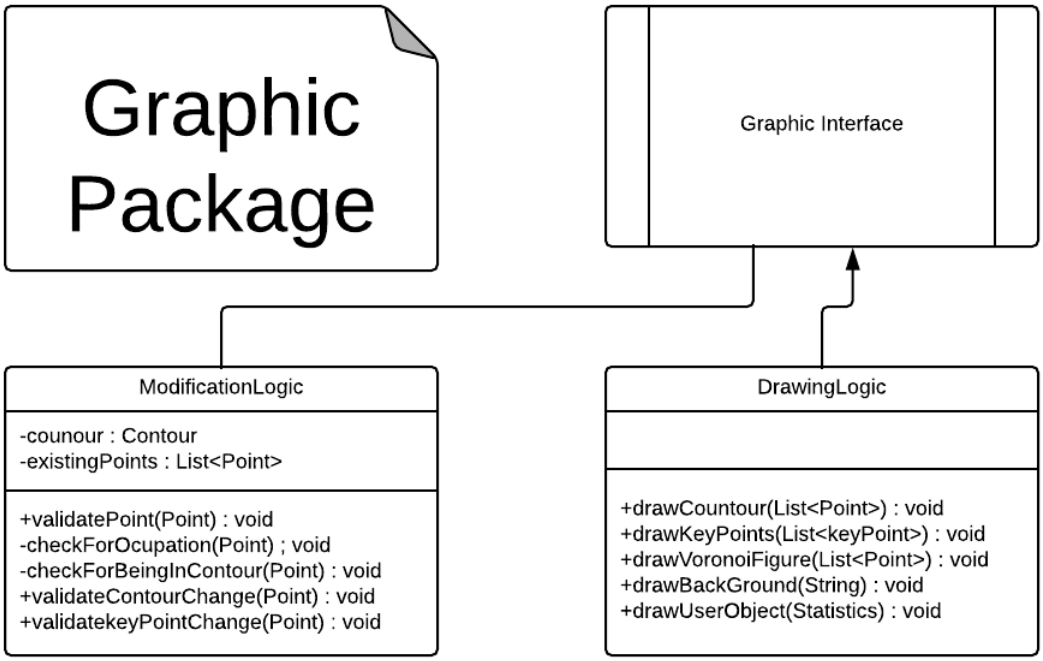
Opisany powyżej algorytm, może ulec zmianie w trakcie implementacji.

3 Diagramy klas

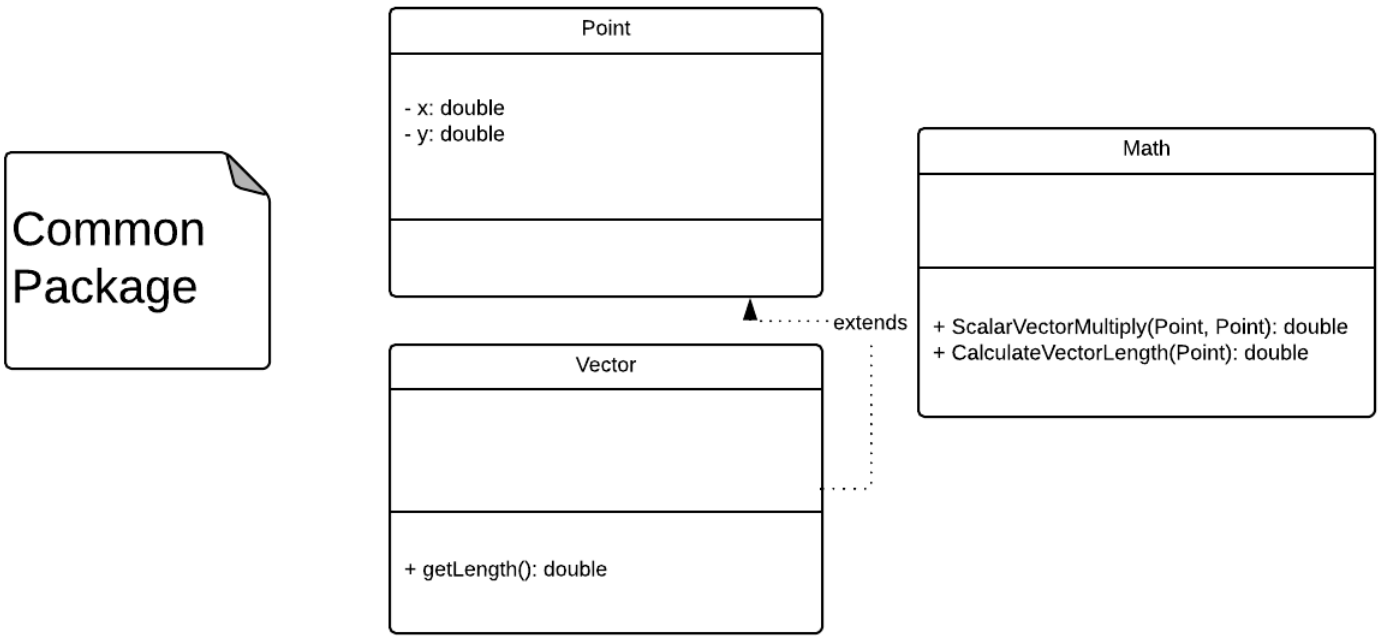
3.1 Statistics



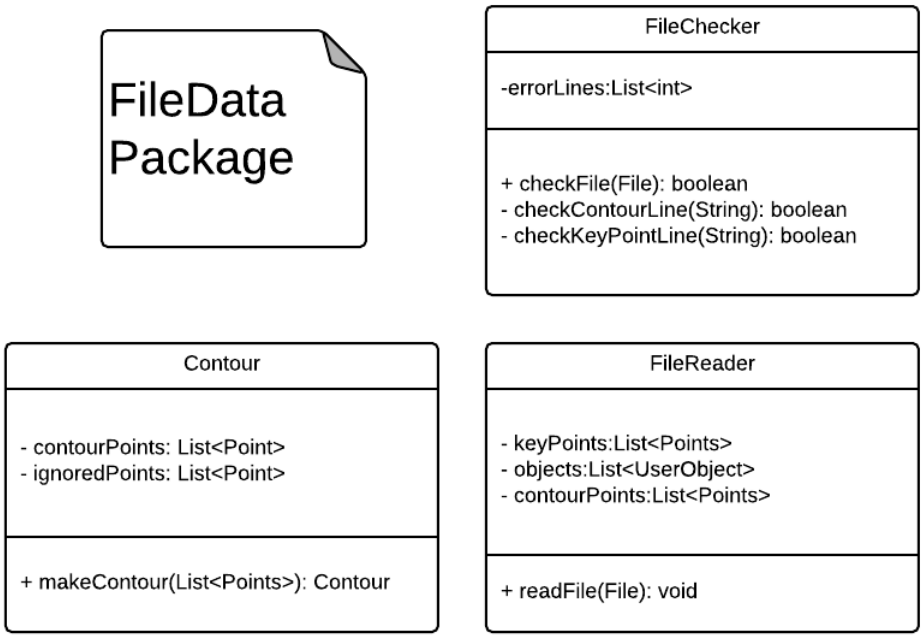
3.2 Graphic Package



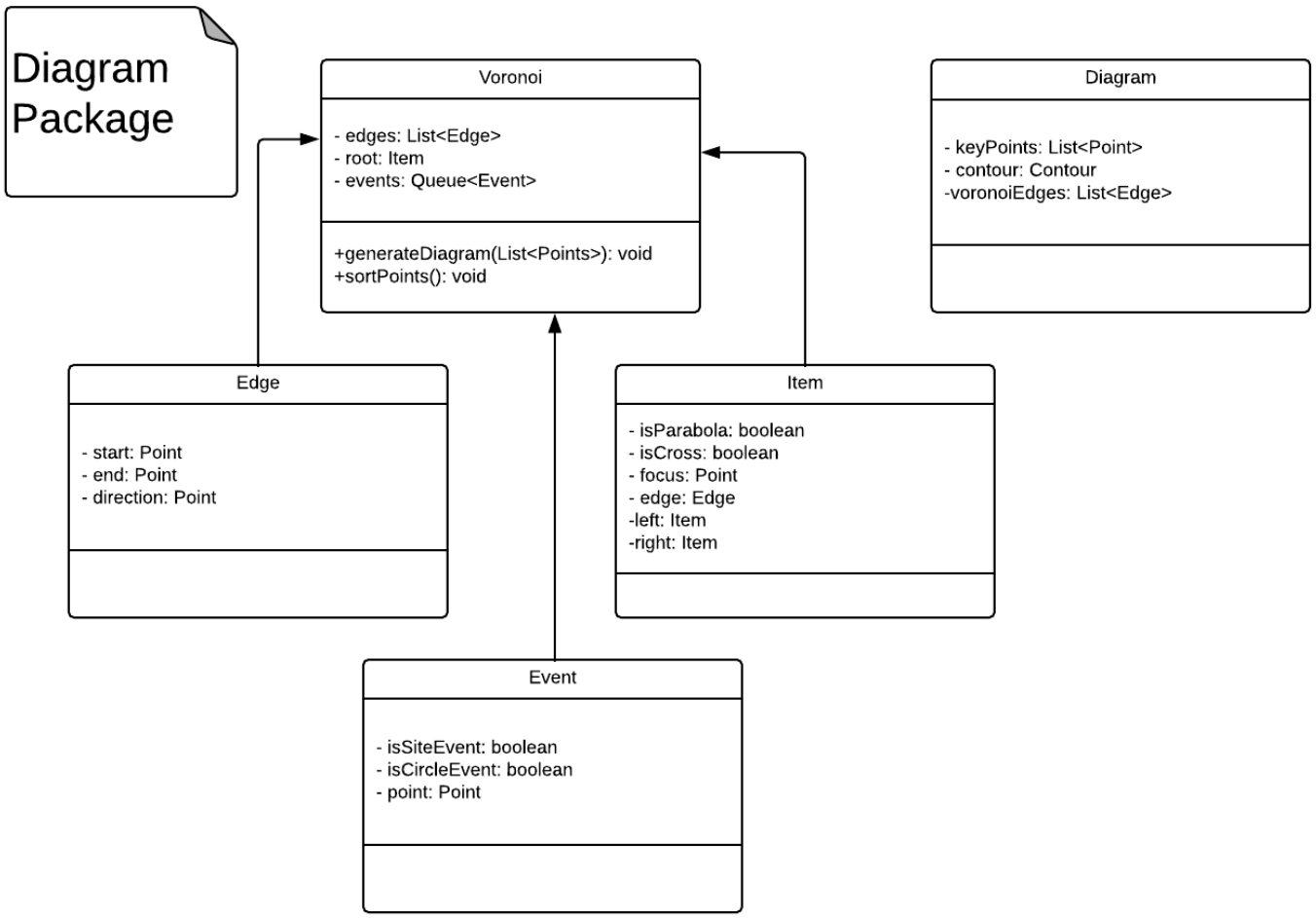
3.3 Common



3.4 FileData


FileData Package
FileChecker
-errorLines:List<int>
+ checkFile(File): boolean
- checkContourLine(String): boolean
- checkKeyPointLine(String): boolean
Contour
- contourPoints: List<Point>
- ignoredPoints: List<Point>
+ makeContour(List<Points>): Contour
FileReader
- keyPoints:List<Points>
- objects:List<UserObject>
- contourPoints:List<Points>
+ readFile(File): void

3.5 Diagram



4 Opis ważniejszych metod

4.1 Pakiet Statistics

- Klasa **UserObject** stanowi bazową klasę opisującą dowolny obiekt jakie może zostać naniesione na obszar konturu. Nie będzie ona udostępniać publicznych konstruktorów, a jedynie metody dodania, w celu hermetyzacji sposobu tworzenia nowych obiektów. Przy tworzeniu obiekt zostanie wpisany w listę statystyki, zostanie też, przy użyciu metody **checkMembership**, określone do obszaru którego punktu kluczowego należy obiekt. Taki sposób organizacji pozwoli na łatwą rozbudowę bazy obiektów o nowe typy.

- Klasa **Statistics** zawiera trzy publiczne metody które odpowiadają na wymagania odnośnie prowadzenia i wyświetlania statystyk. Po zarejestrowaniu żądania użytkownika wywoływana jest odpowiednia metoda dla przykładu omówmy metodę **printAllObjectList:**

  1. Określany jest obszar którego obiekty mają zostać wyświetlone, w tym celu wywoływana jest metoda **findKeyPoints:**
     (a) metoda **findKeyPoints** określa odległość punktu od wszystkich punktów kluczowych i zwraca jako wartość najmniej odległy.
  2. Metoda iteruje po listach obiektów zawartych w klasie **UserObject**.
  3. Obiekty dla których wartość pola memberOf jest wyszukanym wcześniej punktem kluczowym zostają wypisane do przewijanego pola tekstowego zawartego w interfejsie. .

Analogiczne działania należy wykonać w pozostałych metodach, grupując obiekty zgodnie z typami w metodzie **printGroupedObjectList**, lub zliczając mieszkańców w metodzie **printResidentsNumber**. Metoda **recheckData** wywoływana jest za każdym razem gdy nastąpi modyfikacja punktów kluczowych lub granic konturu i sprawdza do jakiego punktu kluczowego należą obiekty oraz czy nie znalazły się poza granicami konturu.

4.2 Graphic

- Klasa koncepcyjna **GraphicInterface** stanowi bazową klasę narzędzie JavaFX. Zawierać będzie obiekty interfejsu przedstawionego w specyfikacji funkcjonalnej. Oraz będzie klasą główną sterującą programem po wykonaniu inicjacji z bazowej klasy **Main**.

- Klasa **ModificationLogic** będzie wykorzystana do obsługi i sprawdzania prawidłowości danych wprowadzanych przez użytkownika w czasie działania programu. Istotne jest to, że zawiera ona bazę zajętych punktów, i właśnie to jest sprawdzane najczęściej, zgodnie z założeniem, że na danych współrzędne może znajdować się tylko jeden obiekt/punkt kluczowy. Dla wywołania metody **validatePoint** sprawdzamy czy wybrany nowy punkt leży w konturze a następnie czy punkt na którym chcemy umieścić nowy obiekt nie jest zajęty. Istotne jest aktualizowanie bazy zajętych punktów oraz współrzędnych konturu, dlatego też utworzona zostanie tylko jeden obiekt **ModificationLogic**, oraz cały program zyska do niego chroniony dostęp (użyty zostanie wzorzec Singleton). Modyfikacja konturu lub punktu kluczowego powodować będzie konieczność obliczenia na nowy podziału obszaru konturu na figury Voronoja. To też klasa będzie wywoływać obliczanie tych podziałów od nowa. A następnie wywoływać ich rysowanie.

- Klasa **DrawingLogic** stanowi podstawową klasę komunikacji miedzy programem a interfejsem. Rysowanie konturów i obszarów wymaga podanie współrzędnych punktów w określonej kolejności. Rysowanie obiektów użytkownika będzie wykonywane na podstawie obiektu klasy Statistics. W kompetencjach tej klasy nie leży weryfikacji poprawności, wszystkie przyjmowane argumenty przyjmuje się za zgodne z ustalonymi założeniami.

4.3 Common

- Klasa **Point** jest kluczowa dla wszelkich operacji na podanych przez użytkownika punktach, ponieważ jest reprezentacją punktu.

- Klasa **Vector** rozszerza klasę **Point** i reprezentuje wektor, a metoda **getLength** pozwoli na łatwy dostęp do jego długości.

- Klasa **Math**, będzie zawierała metody do wszelkich potrzebnych operacji matematycznych, które zostaną uznane za zaawansowane. Poprawi to w znacznym stopniu czytelność kodu, ponieważ przy obliczaniu diagramu Voronoi, czy przy sprawdzaniu wypukłości konturu, pojawi się dużo operacji matematycznych.

4.4 FileData

- Klasa **FileChecker**, będzie odpowiadać za sprawdzenie poprawności pliku wejściowego. Metoda **checkFile** prześledzi plik wywołując pozostałe metody z tej klasy dedykowane pod linie pliku zawierające konkretne dane. W przypadku błędu, w którejś linii doda jej numer do kolekcji.

- Klasa **FileReader** odpowiedzialna będzie za odczytanie danych z pliku. Metoda **readFile** przeczyta cały plik wywołując dla każdej linii odpowiednią metodę dedykowaną pod dane zawarte w tej linii. Punkty kluczowe, obiekty i punkty konturu trafią do odpowiednich kolekcji w tej klasie.

- Klasa **Contour** zajmie się przechowywaniem konturu oraz jego tworzenie. Metoda **makeContour**, na podstawie algorytmu Jarvisa, z otrzymanych punktów stworzy otoczkę wypukłą, a punkty, które nie wejdą w jej skład doda do kolekcji ignorowanych. Pozostałe punkty trafią do **contourPoints** w kolejności łączenia.

4.5 Diagram

- Klasa **Edge** będzie reprezentacją krawędzi, które będą powstawać w trakcie tworzenia diagramu Voronoja. Zawiera ona pole **direction**, ponieważ w myśl założeń algorytmu Fortune'a, kierunek będzie pełnił istotną rolę przy konstrukcji krawędzi.
- Klasa *Event* jest reprezentacją zdarzeń, które są elementem działania algorytmu Fortune'a. Może reprezentować zarówno zdarzenie punktowe jak i zdarzenia okręgu. Zawiera także informacje o punkcie, który dane zdarzenie wywołuje.
- Klasa **Item** jest reprezentacją elementów, które będą wchodziły w skład drzewa binarnego, które przechowywać będzie informacje o linii brzegowej oraz punktach diagramu Voronoi. Dwoista natura tej klasy polega na tym, że może reprezentować zarówno punkt przecięcia paraboli linii brzegowej jak i poszczególną parabolę wchodzącą w skład linii brzegowej. Wynika to ze struktury drzewa binarnego, która jest przewidziana przez algorytm Fortune'a. Dzięki temu operacje na tym drzewie staną się sprawniejsze.
- Klasa **Diagram** będzie przechowywać dane o konturze, punktach kluczowych oraz krawędziach diagramu Voronoi.
- Klasa **Voronoi** skupi w sobie wszystkie działania związane z wyznaczeniem diagramu Voronoi. Jej celem będzie uzyskanie kolekcji krawędzi diagramu Voronoi odpowiedniego dla podanej kolekcji punktów kluczowych. Metoda **generateDiagram** będzie głównym punktem realizacji algorytmu Fortune'a. Punkty, które otrzyma zostaną posortowane w odpowiedniej kolejności (**rozdział 2**). Metoda **generateDiagram** wywoływać, będzie pozostałem metody, które wejdą w skład klasy **Voronoi**. To ona zbuduje drzewo binarne modyfikując linie brzegową i ostatecznie dostarczy kolekcje krawędzi diagramu Voronoi.

5 Testy

5.1 Statistics Package

W celu przetestowania tego pakietu założymy istnienie tylko jednego punktu kluczowego do którego należą wszystkie obiekty, nie sprawdzamy czy obiekt tworzony jest w prawidłowym miejscu obszaru, ponieważ wykona to klasa ModificationLogic:

- dodanie obiektów różnych typów;
- dodanie obiektu podając referencje do zmiennej null;
- dodanie dwóch obiektów o tych samych nazwach;
- sprawdzenie czy obiekt dodano do odpowiedniej struktury;
- odczyt obiektów pogrupowanych;
- odczyt obiektów wylistowanych;
- zliczenie liczby mieszkańców obiektów, w tym kiedy wszystkie obiekty mieszkalne są puste, lub nie istnieje żaden.

5.2 Graphic Package

Większość testów zostanie wykonana dla klasy ModificationLogic. Specyfika tej klasy jest taka że dla danych prawidłowych przekaże wywołania dalej natomiast dla błędnych zgłosi wyjątek, także w znacznej części testów spodziewać się będziemy wyjątków.

- walidacja punktu(kluczowego) prawidłowego;
- walidacja punktu(kluczowego) leżącego poza konturem;

- walidacja punktu(kluczowego) który jest już zajęty;
- próba dodania punktu konturu który zaburzy wypukłość;
- dodanie prawidłowego punktu konturu.

Poprawność metody DrawingLogic zostanie przetestowana manualnie.

5.3 Common Package

Większość testów tego pakietu zostanie wykonana dla klasy Math i operacji matematycznych, które zostaną w niej zawarte, a przypadki na jakie będziemy zwracać uwagę przy testowaniu to:

- dzielenie przez zero;
- podanie wartości null;
- podanie ujemnej wartości w niepożądanym miejscu;
- poprawne dane i oczekiwany wynik.

5.4 FileData

Testy tego pakietu będą przeprowadzone dla metod klas **FileReader** oraz **FileChecker**, a także dla metody **makeContour**. Przypadki jakie będziemy testować to:

- nieprawidłowa ilość sekcji w pliku wejściowym;
- za mała ilość argumentów w linii w pliku wejściowym;
- za duża ilość argumentów w linii w pliku wejściowym;
- kodowanie pliku różne od UTF-8;
- prosty prawidłowy kontur i kilka punktów kluczowych wewnątrz tego konturu;
- kontur, którego krawędzie przecinają się;
- punkty, których współrzędne są takie same jak innego punktu lub obiektu;
- obiekty, których współrzędne są takie same jak innego punktu lub obiektu;
- nieprawidłowa definicja typu;
- plik bez punktów kluczowych;
- plik bez konturu;
- obiekty poza granicami konturu;
- punkt kluczowy poza granicami konturu;
- kontur, który nie jest figurą wypukłą;
- kontur, który jest figurą wypukłą.
- jeden punkt konturu;
- dwa punkty konturu.

5.5 Diagram

Testy wykonane dla tego pakietu będą ściśle związane z zagadnieniem algorytmu Fortune'a. Przypadki na jakie będziemy zwracać uwagę to:

- dwa punkty kluczowe o tej samej współrzędnej y;
- dwa punkty kluczowe o tej samej współrzędnej x;
- dodanie paraboli do linii brzegowej o jednej paraboli;
- dodanie paraboli do linii brzegowej o trzech parabolach;
- dodanie paraboli do linii brzegowej pomiędzy dwoma parabolami;
- dodanie paraboli do linii brzegowej skrajnie z lewej strony;
- dodanie paraboli do linii brzegowej skrajnie z prawej strony;
- zdarzenie okręgu po lewej stronie paraboli;
- zdarzenie okręgu po prawej stronie paraboli;
- zdarzenie punktowe o współrzędnej x powodującej natychmiastowe zdarzenie okręgu.

W trakcie implementacji wszystkie powyższe testy mogą ulec zmianie, a także mogą zostać rozpatrzone przypadki nie zawarte w powyższych opisach.

6 Informacje o sprzęcie i oprogramowaniu

Program będzie pisany w języku Java wersji 9.0.4 ze wsparciem biblioteki JavaFX oraz środowiska Intellij IDEA.

Zostanie przetestowany na komputerach:

1. Lenovo G510 o procesorze Intel Core i5 2.5GHz, pamięci RAM 6GB, karcie graficznej AMD Radeon HD 8570M i systemie operacyjnym Windows 10,
2. Asus X7500J o procesorze Intel Core i7 2.4GHz, pamięci RAM 8GB, karcie graficznej NVIDIA GeForce GT 740M i systemie operacyjnym Windows 10.