**METRO 2k19**

Sprawozdanie ogólne

Nasz projekt skupia się na symulacji przepływu ludności poruszającej się metrem. Przyjęliśmy założenia, że ludzie przepływujący między stacjami są reprezentowane głownie poprzez ich ilość. Dodatkowo, ilość ta jest w rozłożona wg swojej lokalizacji, gęstość ludności nie jest stała, co w przybliżeniu odpowiada faktycznemu rozkładowi ludności w miejscowościach. A mianowicie: w kierunku centrum miejscowości gęstość ludności wzrasta.

Przydzielenie ludności do stacji odbywa się na podstawie odległości do najbliższej. Ludność z punktów „ granicznych ”, jest przydzielana jednostajnie do stacji, do których odległość jest równa. W naszym wypadku odległość od stacji jest całkowita (obliczana oczywiscie z twierdzenia Pitagorasa).

Ludzie wykorzystowują maximum przepustowości każdego połączenia w jednostce czasu. Ponadto, o pożądaniach ludności co do kierunku podróży decyduje współczynnik

*ki= Li / L \* 100 %*

, gdzie *Li* – liczba ludności, która ma najbliżej do i – tej stacji, *L* – całkowita liczba ludności. Zgodnie z tym, z j – tej stacji do i – tej przepłynie

*Pj = kj \* Li*

, Pj  - ilość ludności przepływującej.

Program bazuje na grafie (klasa *Graph*), w którym znajdują się informacje o stacjach (klasa *Station*) i odcinkach je łączących (klasa *Stretch*). W klasie *Graph* odbywa się również symulacja całego procesu przemieszczania się pasażerów między stacjami. Udało nam się to osiągnąć, implementując odpowiedni algorytm, o którym jest mowa w kolejnym akapicie. Lokalizacje stacji, czyli punkty na planszy, przechowuje klasa Point.

Algorytm bazuje na algorytmie Dijkstry i DFS. Pierwszy z nich oblicza najkrótszą odległość danej stacji od innych i zapisuje je w tablicy. Następnie z danej stacji startuje algorytm DFS, którego głębokość jest ograniczona przez odległości z algorytmu Dijkstry. Algorytm DFS nie zejdzie głębiej, jeśli jego aktualna odległość będzie różna od optymalnej. Powtórzone dla każdego wierzchołka daje to złożoność O( n3 log(n) ), gdzie n to ilość stacji. Nie jest to najoptymalniejsze rozwiązanie, ale ponieważ n jest małe ( n<=100 ), to jest to optymalność w pełni zadowalająca.

Mapa rozmieszczenia ludności, na podstawie której obliczana jest ilość pasażerów na danych stacjach, przechowywana jest w klasie *Area*. Zawiera ona również metody obsługujących generowanie ludności.

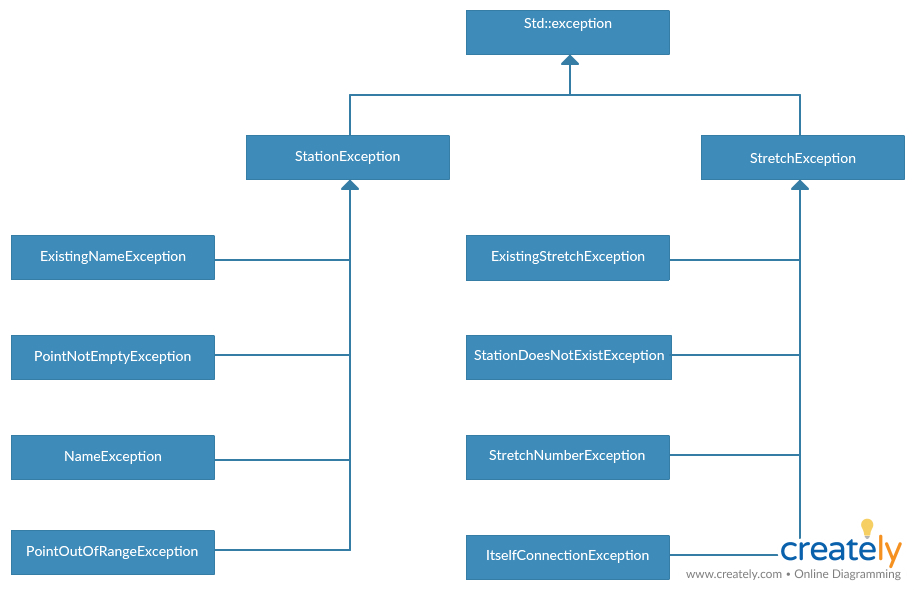
Za wyświetlanie projektu w formie graficznej odpowiada klasa *Graphics*, a w formie tekstowej - klasa *Display*. Obie te klasy dziedziczą po abstrakcyjnej klasie *Present*. Pliki graficzne generowane w formacie svg dzięki prostej bibliotece simple\_svg.hpp dodanej do projektu. Pozwala ona tworzyć podstawowe figury geometryczne. Na grafice wyświetlane są lokalizacje stacji, obecne na nich ilości pasażerów i przepustowości poszczególnych odcinków.

Nasz projekt obsługuje również odczyt i zapis obecnego stanu metra i miasta do pliku tekstowego. Odpowiedzialna jest za to klasa *SaveAndLoad*. Na początku działania programu użytkownik jest pytany o to, czy chce załadować wcześniej zapisane dane.

Obsługa menu i interakcja z użytkownikiem to zadanie klasy *Menu*. Ta sama klasa wywołuje metody pozostałych klas.

Klasa *Parser* odpowiada za poprawność wprowadzonych danych. Sprawdza między innymi, czy dany punkt nie jest już zajęty, albo czy nazwa stacji nie została już użyta. W przeciwnym wypadku generuje się jeden z wyjątków zawartych w folderze Exceptions.

Wyjątki StationException i StretchException dziedziczą po standardowym wyjątku std::exception. Są one klasami bazowymi do szerszej ilości wyjątków, które się odpowiednio odnoszą do stacji i do połączenia. Relacje pomiędzy wyjątkami są zilustrowane na niżej.



Wyjątki ExistingNameException, PointNotEmptyException, PointOutOfRangeException odnoszą się do wprowadzenia danych charakteryzujących stacje.

ExistingNameException – sprawdza, czy przypadkiem imię stacji, którą wprowadziliśmy, nie istnieje już na liście stacji.

PointNotEmptyException – czy w danym pukcie już nie istnieje stacja.

PointOutOfRangeException – czy punkt, w którym chcemy umieścić stację należy do planszy.

NameException – uniemożliwia występowanie cyfr w nazwie stacji.

Wyjątki ExistingStretchException, StationDoesNotExistException, StretchNumberException, ItselfConnectionException odnoszą się do obsługi prawidłowego funkcjonowania mechanizmu dodawania połączeń.

ExistingStretchException – sprawdza, czy połączenie już istnieje (nieważny kierunek)

StationDoesNotExistException – sprawdza, czy stacja, do której chcemy sie podłączyć istnieje.

StretchNumberException – sprawdza, czy połączeń jest *n \* (n – 1) / 2* , gdzie *n* – liczba stacji.

ItselfConnectionException – sprawdza, czy stacja nie jest połączona sama z sobą, porównywane na podstawie położenia. (Jeszcze jedno z założeń mówi o tym, że nie można w tym samym punkcie przechowywać 2 – ch lub większej ilości stacji).

Rozszerzeniem projektu może być dynamiczne dodawanie stacji i połączeń do symulacji, zarówno przy tworzeniu, jak i przy modyfikacji istniejącej wersji.

Kolejnym punktem może być dodanie Graficznego interfejsu użytkownika.

Usunięcie założenia o niemożliwości istnienia stacji w jednym punkcie, pozwoli rozszerzyć projekt do „ 3 – wymiarowej ” postaci.