DOKUMENTACJA - POJAZD AUTONOMICZNY

URUCHOMIENIE PROGRAMU

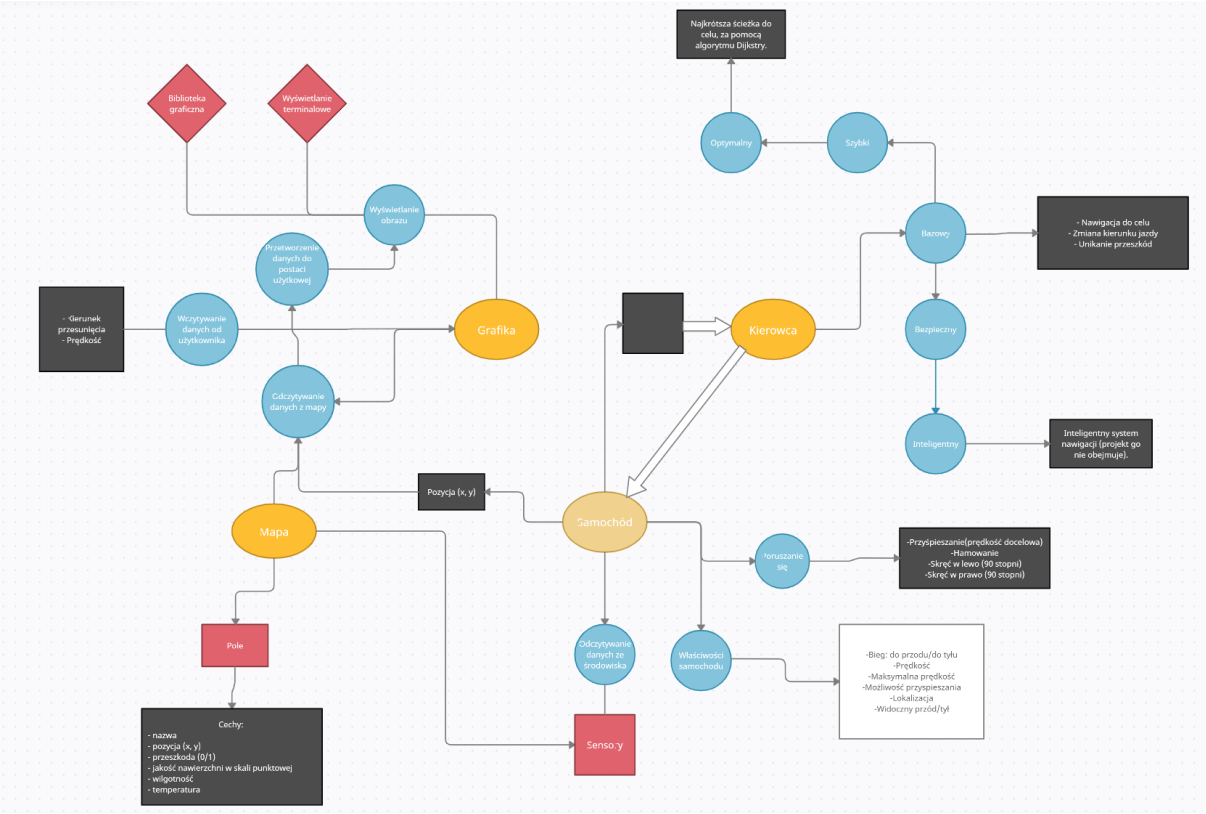
Należy uruchomić plik *main* i w terminalu wpisać odpowiednie parametry aby wybrać mapę oraz kierowcę.
Jak uruchomić program:

1. Należy odpalić aplikację *main*.
2. Wybieramy mapę, przez wpisanie numeru od 1 do 3.
3. Wybieramy kierowcę wpisując odpowiednio: 'I' - Inteligentny, 'S' - Głupi
4. Wybieramy punkty startowy i docelowy, punkty muszą leżeć w obrębie mapy i nie mogą być przeszkodami!

BUDOWA PROJEKTU

Program składa się z 4 modułów - kierowcy, samochodu, mapy oraz grafiki. Każdy z tych modułów jest niezależny i może być modyfikowany a funkcjonalność programu zostanie zachowana.
Projekt rozpoczęliśmy od stworzenia pierwszego modelu w postaci mapy myśli:

https://app.creately.com/d/JovNBP3d3dS/edit



Podział pracy:

- Grafika - Karol Ziarek
- Kierowca - Wojciech Łapacz
- Samochód i Mapa - Szymon Jankowski
- Plik main i połączenie modułów w całość - Karol Ziarek, Wojciech Łapacz
- Poprawki w poszczególnych modułach podczas tworzenia aplikacji:
  Wojciech Łapacz, Karol Ziarek

MODUŁ 1. - KIEROWCA

Autor - *Wojciech Łapacz*

Dostępne są 4 typy kierowców: StupidDriver, Driver, SafeDriver oraz IntelligentDriver.

StupidDriver - klasa bazowa, steruje samochodem w sposób losowy - jeżeli natrafi na przeszkodę, to losowo zmieni kierunek jazdy. Nie dąży do żadnego konkretnego celu. W momencie wjechania w “ślepą uliczkę”, kierowca zatrzyma pojazd i zakończy swoje działanie.

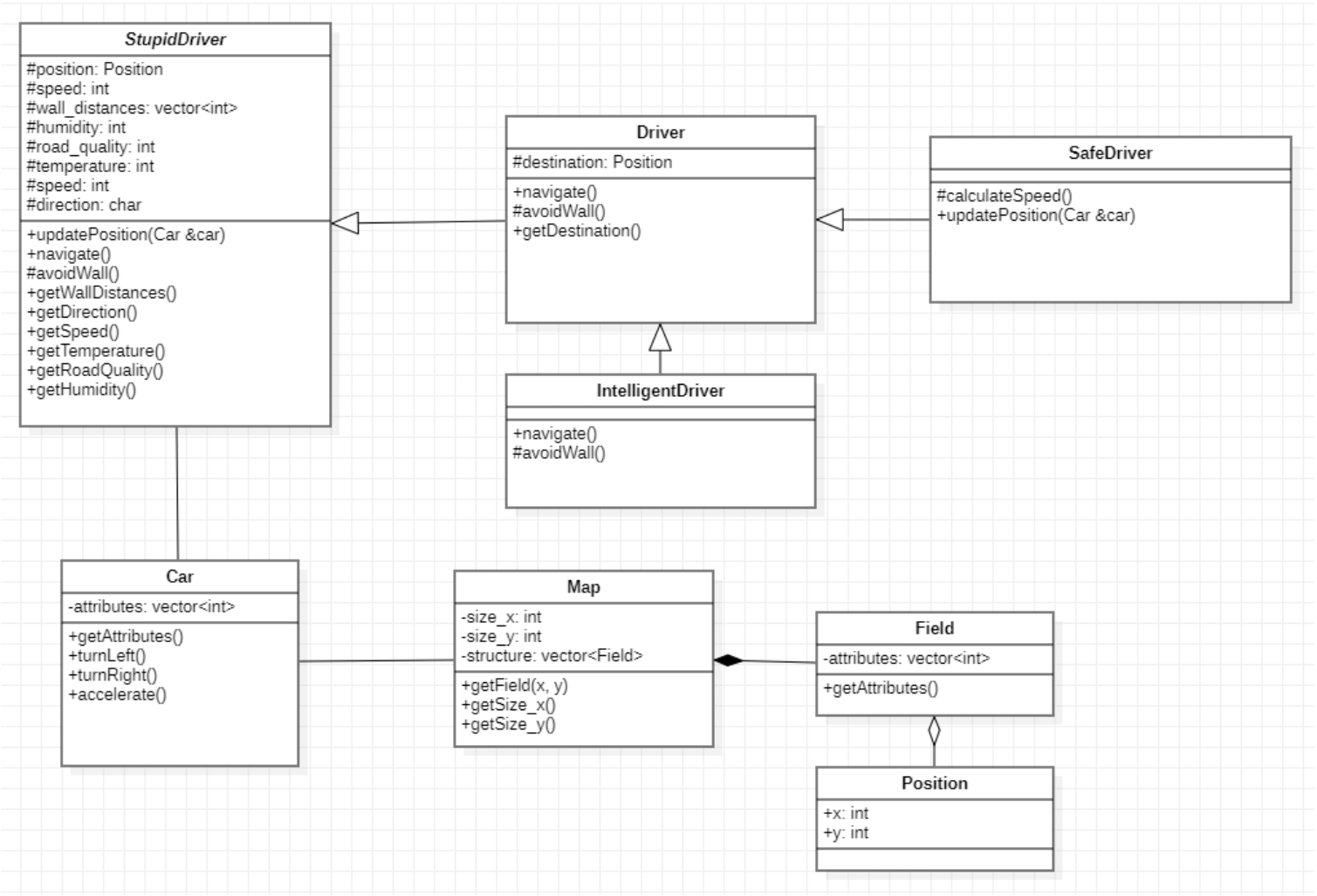
Driver - steruje samochodem, aby dotrzeć do wyznaczonego przez użytkownika celu (próbuje ustawić odpowiednią pozycję X samochodu a następnie Y). Potrafi również wyjechać ze “ślepej uliczki” w przeciwieństwie do kierowcy *StupidDriver*.

SafeDriver - dziedziczy po klasie *Driver* i ma wszystkie jego funkcje, ale dodatkowo oblicza bezpieczną prędkość do warunków na drodze. Wykorzystuje przy tym dane zebrane z czujników samochodu i oblicza bezpieczną prędkość z jaką może jechać pojazd. Steruje pojazdem w taki sam sposób jak kierowca *Driver*.

IntelligentDriver - dziedziczy po klasie *Driver*, ale zmieniona zostaje metoda *navigate()*, aby umożliwić wykorzystanie inteligentnego systemu do sterowania pojazdem (np. sieci neuronowej), nasz projekt nie obejmuje tego typu rozwiązań, więc zakładamy, że jest to miejsce, do którego możemy “podłączyć” system dostarczony przez producenta takiego oprogramowania.

Do każdego typu kierowcy wykonane zostały testy jednostkowe.

Dokładna struktura klas i dziedziczenie w module *Kierowca* przedstawiona w postaci schematu UML:
(klasy Car, Map oraz Field są jedynie poglądowe i nie odzwierciedlają rzeczywistej implementacji)



MODUŁ 2. - SAMOCHÓD oraz MAPA

Autor - *Szymon Jankowski*

Dane są 3 klasy, które składają się na samochód oraz 2 składające się na mapę. Samochód dodatkowo współpracuje z różnymi sensorami.

Steering - klasa stworzona w celu ułatwienia zmian sposoby sterowania pojazdem w przyszłości. Zawiera informacje takie jak kierunek, czy prędkość, które pozwalają przemieszczać się samochodowi. Sterowanie odbywa się na zasadzie zmiany prędkości oraz skrętom w lewo i prawo - obrotom o 90 stopni.

BaseCar - klasa bazowa samochodu. Dziedziczy po Steering. Ma informację o swoim położeniu oraz może się poruszać w kierunku, który jest wybrany. Nie potrafi za to odczytywać informacji z mapy.

Car - dziedziczy po BaseCar. Dzięki zbiorowi sensorów potrafi odczytać z mapy potrzebne informacje. Dostępne sensory:

- **TouchSensor** - wykrywa czy samochód jest 1 pole obok przeszkody(nie na ukos)
- **RadarSensor** - pobiera informację jak daleko od samochodu znajduje się przeszkoda - zwraca odległość z przodu, z lewej oraz z prawej od samochodu
- **TemperatureSensor, HumiditySensor, SurfaceSensor** - odczytują z pola informację po kolei o: temperaturze, wilgotności, jakości nawierzchni

Field - elementy tej klasy wypełniają całą mapę. Zawiera informacje o polu: lokalizację, czy jest barierą, wilgotność, jakość nawierzchni oraz temperaturę.

Map - klasa przetrzymująca wektor wektorów zawierających elementy klasy Field. Element klasy można stworzyć w kilka sposobów. Konstruktor domyślny tworzy obiekt klasy z pustym kontenerem. Można go natomiast wypełnić dowolnym wektorem wektorów zawierającym elementy klasy Field. Kontener można też wypełnić funkcją czytającą mapę z pliku tekstowego. Drugi konstruktor umożliwia utworzenie mapy od razu wczytując ją z pliku tekstowego. Trzeci konstruktor pozwala na stworzenie mapy o podanych wymiarach z polami o domyślnych parametrach. Dowolne pole na mapie można zmienić za pomocą funkcji setField.

MODUŁ 3. - GRAFIKA

Autor - *Karol Ziarek*

Moduł grafiki opiera się na 4 pomniejszych klasach-modułach, które razem tworzą funkcjonalną całość. Dodatkiem do modułu graficznego jest plik Menu_functions.h, który zawiera funkcje podstawowego menu używane w obecnej wersji pliku main.cpp

Display - klasa, która jest odpowiedzialna za wyświetlanie obrazu. Bazowa klasa Display posiada dwie niezbędne dla każdego Display'a metody - drawPoint, oraz getTexture. Aktualnie dostępne są 3 rodzaje funkcjonalnych wyświetlaczy:

- ***WindowsTerminalDisplay*** - konsolowy dla systemu Windows,
- ***LinuxTerminalDisplay*** - konsolowy dla systemów Unixowych,
- ***BmpDisplay*** - oraz zapisujący obraz do plików BMP, korzystający z zewnętrznej biblioteki graficznej EasyBMP

Object - klasa obiektów, odpowiedzialnych za wyświetlanie wybranych kształtów, wywoływanie odpowiednich zachowań w klasie Display. Bazowa klasa Object posiada funkcję wirtualną draw(), która dla każdego obiektu dziedziczącego wywołuje pożądane zachowanie. Aktualnie dostępnych jest 5 rodzajów obiektów:

- *DisplayTile* - wyświetlanie podanego znaku w zadanym punkcie
- *DisplayCar* - wyświetlanie symbolu samochodu w zadanym punkcie
- *DisplayLine* - linii znaków o podanej długości zaczynając od zadanego punktu
- *clearConsole* - czyszczenie obszaru o zadanej długości i szerokości (od punktu 0,0)
- *printBMP* - obiekt wymuszający zapisanie pliku BMP

Drawing - klasa kontenerowa przetrzymująca w vectorze kolejne obiekty klasy Object. Korzystając z konstrukcji unique_ptr możemy dodawać kolejne obiekty. Przy pomocy metody *draw()*, pożądane działania obiektów wywoływane są w kolejności dodawania do vectora. Po użyciu metody *draw()* vector jest opróżniany z obiektów o zrealizowanym działaniu.

Data Loader - klasa odpowiedzialna za komunikację z zewnętrznymi modułami. Realizuje wczytywanie i przetwarzanie danych z mapy i samochodu. Metoda *load_data()* kolejno odczytuje informacje o wszystkich polach mapy, oraz samochodzie i zamienia je na dane zrozumiałe dla klas Display'owych.

Menu_functions - plik przetrzymuje funkcje odpowiedzialne za wyświetlenie w konsoli prostego menu, mi.n. wyświetlenie informacji o możliwym wyborze mapy i kierowcy, wczytanie danych początkowych - mapa, kierowca, pozycja startowa i końcowa.