WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

Projekt

Program "Samochód"

Prowadzący:
Dr inż. Barbara Fryc

Student:
Łukasz Kuczera, 64130

Kierunek:
Informatyka I stopień, niestacjonarne

Przedmiot:
Programowanie Obiektowe

Rzeszów, 2023

Spis treści

1 Założenia Projektu **2**

2 Cele Projektu **2**

3 Wymagania Funkcjonalne **3**

4 Wymagania Niefunkcjonalne **4**

5 Struktura Programu **5**
5.1 Interfejs 6
5.2 Graficzny Interfejs Użytkownika 8
5.3 Gameplay Objects 10
5.4 Inne klasy 13

6 Opis techniczny projektu **15**
6.1 Wykorzystany język 15
6.2 Wykorzystane narzędzia 15
6.3 Minimalne wymagania sprzętowe 15
6.4 Zarządzanie danymi 16

7 Diagram Gantta **17**

8 Repozytorium i system kontroli wersji **19**

9 Prezentacja warstwy użytkowej aplikacji **20**
9.1 Menu główne 20
9.2 O programie 21
9.3 System weryfikacji danych kierowcy 22
9.4 Menu wyboru pojazdu 24
9.5 Etap rozgrywki 25

10 Podsumowanie **27**
10.1 Zrealizowane prace 27
10.2 Dalsze plany dotyczące projektu 27

11 Literatura **28**

Wstęp

W dzisiejszych czasach samochody są powszechną i niezbędną częścią życia i prawidłowego funkcjonowania gospodarki. Pojazdy silnikowe różnią się wyglądem, wagą, a także źródłem siły wprawiającej je w ruch. Oprócz najpopularniejszych samochodów z silnikami spalinowymi spalającymi benzyne lub olej napędowy od niedawna po ulicach poruszają się także samochody z silnikami elektrycznymi, a bliskiej przyszłości na ulicach mogą zagościć samochody, których silniki zasilane są wodorem.

Pomimo różnic w budowie tych pojazdów wszystkie mają kilka cech wspólnych i kluczowych dla realizacji projektu. Projekt jednak skupi się na samochodach spalinowych z skrzynią biegów. Kluczowymi parametrami, które zostaną uwzględnione w symulacji to między innymi: Moc silnika, przełożenie skrzyni biegów i masa pojazdu.

1 Założenia Projektu

Projekt zakłada stworzenie programu symulującego działanie samochodu oraz umożliwienie użytkownikowi sterowania samochodem. Program zostanie wyposażony w interfejs graficzny ułatwiający użytkownikowi obserwowanie zmian parametrów samochodu takich jak: Prędkość, Obroty silnika. Dodatkowo zaimplementowana zostanie prosta symulacja fizyki działania pojazdu samochodowego.

2 Cele Projektu

Celem projektu jest stworzenie programu symulującego działanie prawdziwego pojazdu silnikowego z uwzględnieniem prostej fizyki silnika spalinowego. Dodatkowym celem projektu jest rozwinięcie umiejętności samodzielnej pracy oraz poznanie aspektów tworzenia programów komputerowych z szerszej perspektywy.

3 Wymagania Funkcjonalne

Większość użytkowników programów komputerowych preferuje korzystanie z interfejsu graficznego niż z aplikacji komunikujących się z użytkownikiem przez konsolę. Zapewnia to nie tylko lepszy wygląd programu ale też większą czytelność informacji. Dodatkowo twórca nie jest ograniczony przez liczbę znaków możliwych do wyświetlenia w konsoli, a sam użytkownik nie jest narażony na atak epileptyczny spowodowany ciągłym odświeżaniem okienka konsoli w przypadku programów z obsługą w czasie rzeczywistym.

Kolejnym z założonych wymagań jest obsługa zdarzeń w czasie rzeczywistym, w taki sposób aby program w ciągły sposób reagował na akcje użytkownika, a nawet i bez niej. Nie jest dopuszczalna sytuacja, w której wszelkie akcje czy to związane ze stroną fizyczną czy też graficzną zostają zatrzymane w oczekiwaniu na wciśnięcie przez gracza przycisku.

W przypadku aplikacji symulującej działanie pojazdu samochodowego użytkownik wymaga aby wydarzenia dziejące się na ekranie były jak najbardziej zbliżone do rzeczywistości lub przynajmniej trzymały się ustalonych ram. Wymaga to od twórcy zaimplementowanie systemu, który będzie w jak najdokładniejszy sposób odwzorowywał wydarzenia związane z fizyką przedstawionych rzeczy. Implementacja dokładnej fizyki jest trudna nawet dla doświadczonego i dużego zespołu, dlatego nawet jeśli zostanie zaimplementowany uproszczony system fizyki to powinien on jak najdokładniej naśladować zdarzenia, które miały by miejsce w świecie rzeczywistym. Lub przynajmniej starać się. Sytuacja w tym przypadku jest o tyle prosta, że zakładamy iż interfejs graficzny programu będzie 2D, a liczba elementów do opisania takich jak fizyka działającego silnika czy też działanie skrzyni biegów jest nie wielka.

- Implementacja interfejsu graficznego - Stworzenie funkcjonalnej deski rozdzielczej z działającymi zegarami prezentującymi parametry pojazdu.
- Symulacja działania silnika spalinowego na podstawie parametrów: Pojemność silnika, Stopień otwarcia przepustnicy.
- Symulacja działania skrzyni biegów - Stosunek przełożenia obrotów silnika na koła pojazdu.
- Symulacja podstawowej fizyki pojazdu - Cieższy samochód będzie wymagał silnika o większej mocy.
- Umożliwienie użytkownikowi wyboru pojazdu.
- Implementacja systemu usterek samochodu takich jak: Zatarty silnik, Brak paliwa, Uszkodzona świeca zapłonowa - Spadek mocy pojazdu.
- Umożliwienie użytkownikowi sterowania pojazdem: Włączenie/Wyłączenie silnika, Zmiana biegów, Regulacja przepustnicy, Hamowanie.
- Obsługa zdarzeń w czasie rzeczywistym: Aplikacja powinna w czasie rzeczywistym reagować na akcje użytkownika lub jej brak oraz prezentować informacje zwrotną.

4 Wymagania Niefunkcjonalne

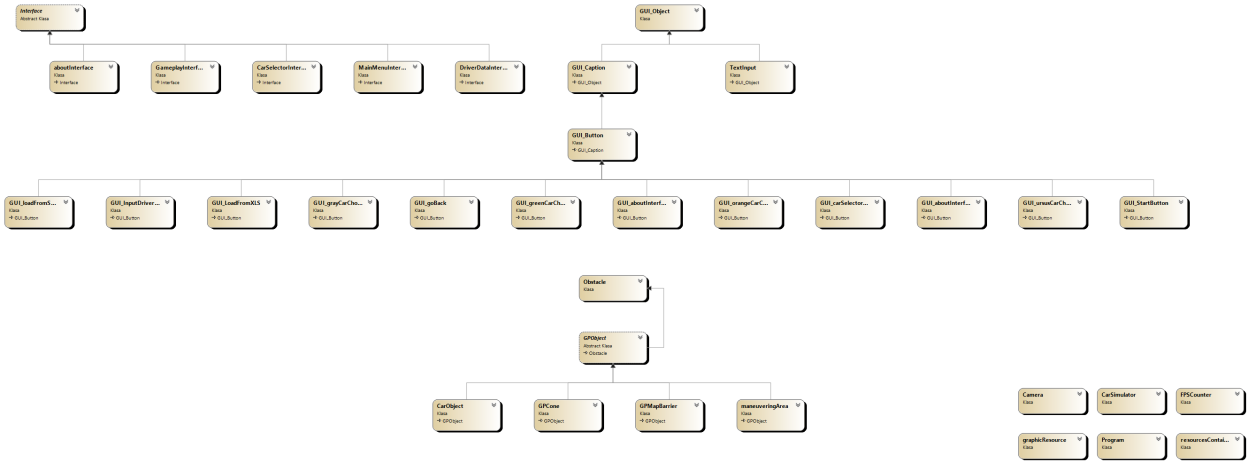
Od programu komputerowego użytkownik wymaga aby jego interfejs graficzny był przejrzysty i czytelny w taki sposób aby użytkownik nie miał problemu z odczytaniem i zrozumieniem prezentowanych wartości. Kolejną ważną kwestią jest aby program efektywnie korzystał z zasobów komputera, niedopuszczalna jest sytuacja, w której aplikacja zużywa zbyt dużo mocy procesora, pamięci operacyjnej lub zajmowała zbyt dużo miejsca na dysku. Moc procesora i miejsce na dysku wykorzystane przes program jest głównie kwestią optymalizacyjną i wymaga od twórcy mądrego zarządzania zasobami, lecz w przypadku pamięci RAM może pojawić się kilka błędów, które utrudnią lub uniemożliwią korzystanie z aplikacji. Jednym z takich błędów jest wyciek pamięci, jest to sytuacja, w której nieużywane i zbędne fragmenty pamięci nie są na bieżąco zwalniane doprowadzając do zajmowania coraz większych jednostek tego zasobu. Wyciek pamięci może mieć charakter powolny lecz konsekwentny lub natychmiastowy skutkując natychmiastowym wyczerpaniem się tego zasobu. Kolejnym problemem związanym z niewłasciwym zarządzaniem pamięcią RAM jest sytuacja, w której program próbuje uzyskać dostęp do nie swojej sekcji pamięci. Skutkuje to błędami przy różnego rodzaju obliczeniach lub nawet zatrzymaniem programu, co również jest niedopuszczalne.

Ostatnim z najważniejszych wymagań niefunkcjonalnych jest zabezpieczenie przed wszelkiego rodzaju błędami. Najrozsądniej jest o to zadbać już na etapie projektowania programu, tak aby dało się przewidzieć jego przebieg i zaimplementować zabezpieczenia. Mówiąc o błędach chodzi między innymi o sytuacje, w których program nie obsługuje wyjątków lub utyka w pętli. W przypadku tych pierwszych program najpewniej zostanie zatrzymany informując użytkownika o błędzie, a przy utknięciu w pętli lub przypadkowym stworzeniu pętli nieskończonej program nagle przestanie odpowiadać na akcje użytkownika by dopiero po chwili zostać zatrzymanym przez system lub samego użytkownika. Podsumowywując, tworząc aplikacje należy zadbać o:

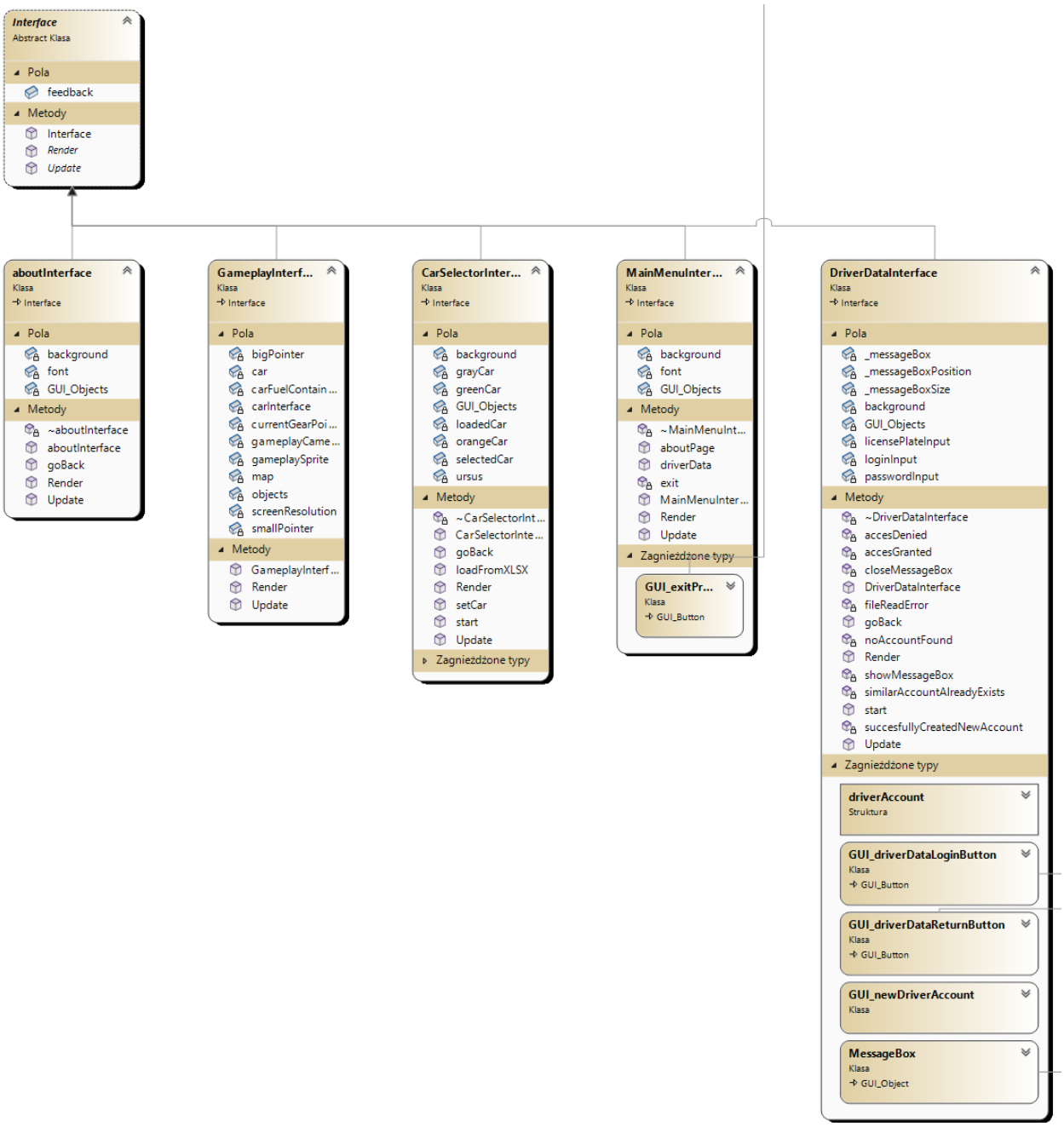
- Intuicyjny interfejs graficzny: Użytkownik nie powinien mieć problemu z odczytaniem reprezentowanych informacji.
- Efektywne wykorzystanie zasobów: Program nie może zużywać wiecej pamięci komputera lub mocy procesora niż jest to wymagane.
- Zabezpieczenie przed błędami: Program powinien obsługiwać wyjątki oraz nie dopuścić do sytuacji wycieku pamięci lub utknięcia w pętli.

5 Struktura Programu

Przedstawiona poniżej grafika prezentuje ogólną strukturę programu. Na grafice można wyróżnić cztery grupy klas. Klasy związane z interfejsem, graficznym interfejsem użytkownika, obiektami rozgrywki oraz klasy nie związane z dziedziczeniem. Wszystkie grupy klas zostaną poniżej szczegółowo omówione.



5.1 Interfejs



Klasa Interface oraz wszystkie dziedziczące po niej klasy są bezpośrednio związane z obecnie obsługiwanym interfejsem. Klasa Interface jest klasą abstrakcyjną, co znaczy, że obiekt klasy Interface nie został ani razu utworzony ze względu na abstrakcyjne metody Render i Update.

```
using SFML.Graphics;
using SFML.Window;

//Klasa bazowa interfejsów
Odwołania: 12
public abstract class Interface
{
    public CarSimulator feedback;
    Odwołania: 5
    public Interface(CarSimulator feedback)
    {
        this.feedback = feedback;
    }
    Odwołania: 6
    public abstract void Render(RenderTexture drawTexture);
    Odwołania: 6
    public abstract void Update(List<Keyboard.Key> pressedKeys);
}
```

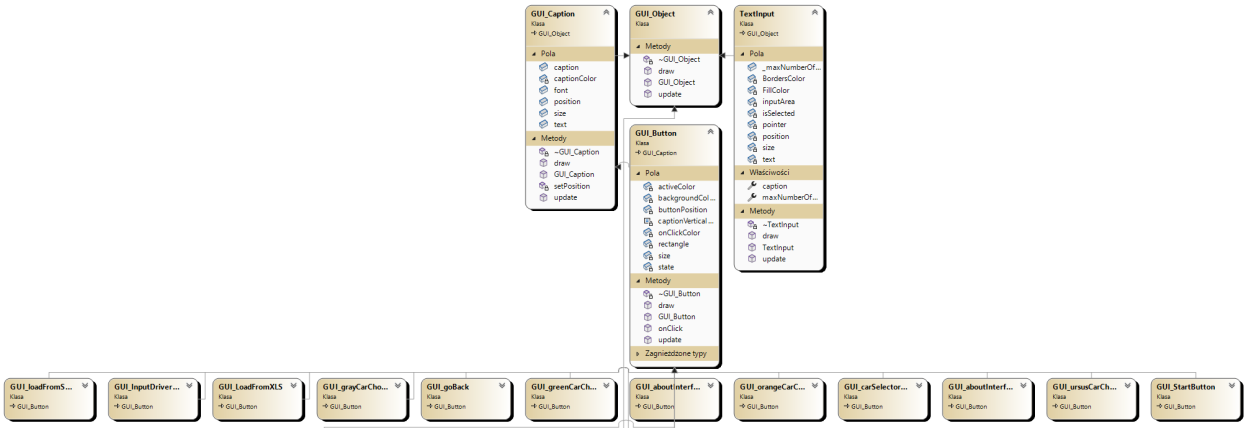
Metody Render i Update są zastępowane w klasach dziedziczących, które deklarują własną definicje metody ze względu na zróżnicowanie interfejsów. Metoda Render w nagłówku deklaruje zmienną typu RenderTexture, na której klasy dziedziczące rysują elementy interfejsu graficznego, natomiast metoda Update przyjmuje listę wciśniętych klawiszy do obsłużenia. Pole feedback typu CarSimulator zawiera informacje o klasie tworzącej obiekt danego interfejsu, dzięki czemu interfejs może wysłać informacje zwrotną do głównej klasy programu o np. potrzebie zmiany obecnie wyświetlanego interfejsu.

Klasy dziedziczące po klasie Interface to:

- **aboutInterface** - Interfejs wyświetlający krótki opis programu. Pola to background, czyli tło oraz GUI_Objects będące listą obiektów interfejsu graficznego. Metoda gBack jest wywoływana przez przycisk powrotu i wywołuje metodę obiektu feedback, która zmienia wyświetlany interfejs.
- **GameplayInterface** - Interfejs głównej części programu, czyli rozgrywki. W polach zawierają się między innymi: Lista obiektów rozgrywki, elementy interfejsu graficznego i informacje o obecnie kierowanym pojeździe.
- **CarSelectorInterface** - Interfejs wyboru jednego z pośród czterech dostępnych pojazdow. Parametry wybranego pojazdu musza być załadowane z pliku .xlsx, wtedy zostaje odblokowany przycisk umożliwiający przejście do GameplayInterface.
- **MainMenuInterface** -Interfejs Menu Głównego, pozwala na wybór miedzy DriverDataInterface, AboutInterface oraz zamknięciem programu.

- **DriverDataInterface** - Interfrejs danych kierowcy. Prowizoryczny system tworzenia oraz logowani do konta kierowcy. Stworzony specjalnie na potrzeby wymagań projektowych, zapewnia podstawową walidacje danych.

5.2 Graficzny Interfejs Użytkownika



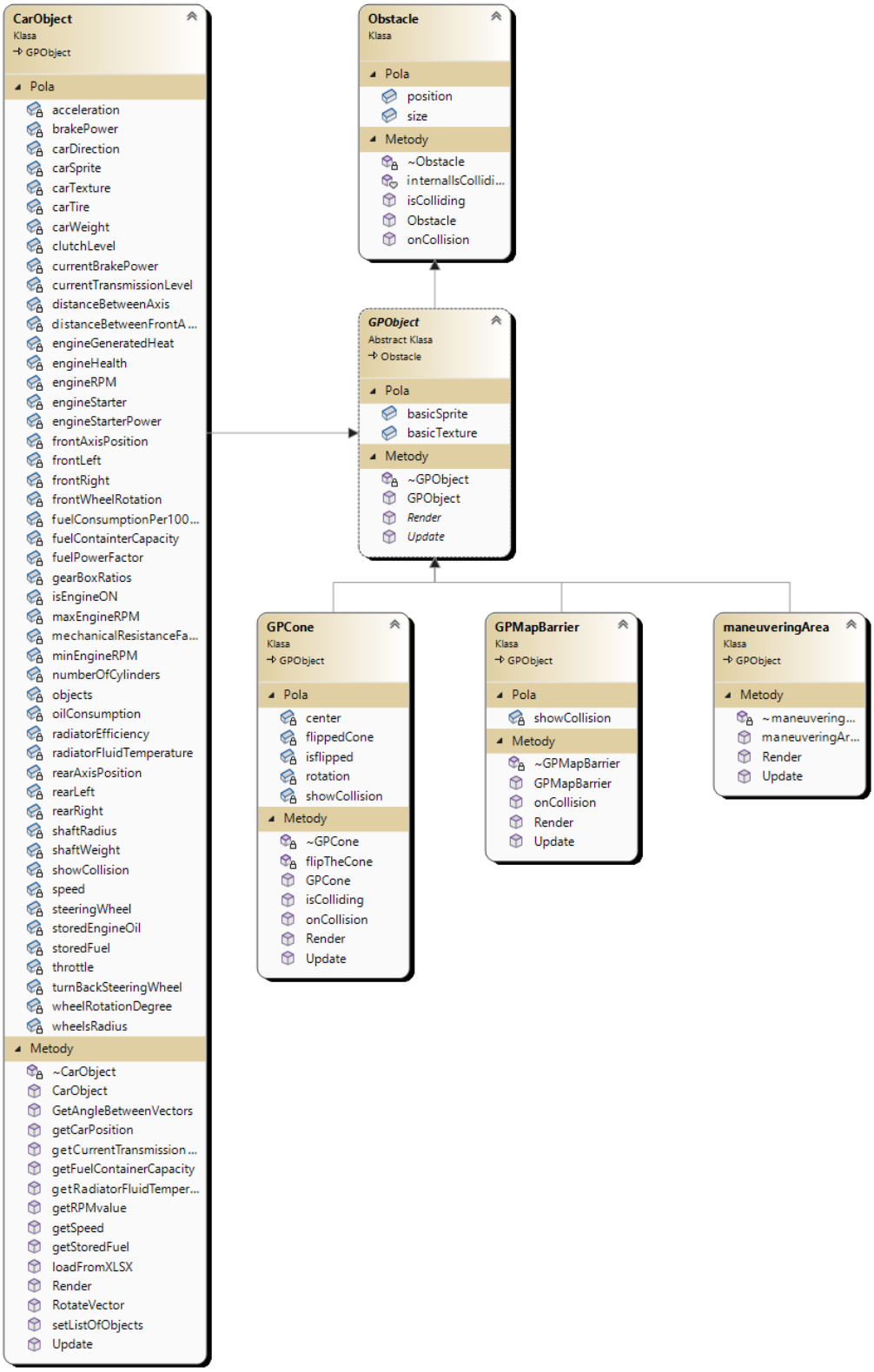
Podstawową klasą interfejsu graficznego jest klasa GUI_Object. Klasa GUI_Object podobnie jak bazowa klasa interfejsów deklaruje metody Render i Update. Metoda render przyjmuje argument typu RenderTexture, na którym dziedziczące elementy interfejsu graficznego rysują, a metoda Update przyjmuje wektor, ktory jest pozycją myszy, argument typu bool informujący czy lewy przycisk myszy jest wcisnięty oraz listę wciśniętych klawiszy klawiatury.

```
using SFML.Graphics;
using SFML.System;
using SFML.Window;

//Klasa bazowa obiektów interfejsu
Odwołania: 22
public class GUI_Object
{
    1 odwołanie
    public GUI_Object()
    {

    }

    Odwołania: 0
    ~GUI_Object()
    {

    }
    Odwołania: 11
    public virtual void draw(RenderTexture target)
    {

    }

    Odwołania: 10
    public virtual void update(Vector2i mousePosition, bool isLMBPressed, List<Keyboard.Key> pressedKeys)
    {

    }
}
```

Z klasy GUI_Object dziedziczą bezpośrednio klasy GUI_Caption raz TextInput. Z klasy GUI_Caption dziedziczy klasa GUI_button, z której dziedziczą wszystkie przyciski w projekcie takie jak "GUI_LoadFromXLSX", "GUI_StartButton" czy też przyciski wyboru pojazdu.

- **GUI_Caption** - Klasa odpowiedzialna za przechowywanie oraz rysowanie zadanego napisu na obiekcie klasy RenderTexture.
- **TextInput** - Klasa odpowiedzialna za wyświetlanie i przechowywanie tekstu wprowadzonego przez użytkownika. Klasa TextInput deklaruje w polach obiekt klasy GUI_Caption, mimo iż mogła by dziedziczyć bezpośrednio z klasy GUI_Caption zamiast z klasy GUI_object. Jest to drobne niedopatrzenie dostrzeżone po czasie.
- **GUI_Button** - Klasa odpowiedzialna za rysowanie przycisku wraz z napisem oraz reagowanie na akcje użytkownika takie jak najechanie kursorem czy naciśnięcie przycisku. Klasa rozwiązuje problem różnorodności akcji po wciśnięciu przycisku w następujący sposób: Wszystkie klasy dziedziczące zachowują metody draw i update klasy bazowej, natomiast nadpisują metode "onClick", w której jest deklarowana akcja danego przycisku. We wszystkich przypadkach wygląda to tak, że konstruktor danego przycisku przyjmuje zmienną typu tworzącego ją interfejsu, a następnie przy wykryciu naciśnięcia wywołuje odpowiednią metode klasy tworzącej. Wywołanie metody onClick odbywa się w metodzie update klasy bazowej.

5.3 Gameplay Objects



Klasa "Obstacle" jest klasą bazową dla wszystkich obiektów rozgrywki, reprezentuje wzajemną kolizje obiektów. Z klasy "Obstacle" dziedziczy klasa GPObject, z której dziedziczą wszystkie obiekty występujące w rozgrywce. Nie widzę przeszkód by klasa "Obstacle" zamiast dziedziczenia była implementowana wewnątrz klasy GPObject. Jest to kolejne drobne niedopatrzenie zauważone podczas pisania pracy.

```
using SFML.Graphics;
using SFML.System;
using SFML.Window;

//Klasa obiektu rozgrywki (GamePlay Object)
//Z tej klasy dziedziczą wszystkie inne klasy obiektów rozgrywki
Odwołania: 17
public abstract class GPObject : Obstacle
{
    public Texture basicTexture;
    public Sprite basicSprite;
    Odwołania: 4
    public GPObject(Vector2f size, Vector2f position) : base(position, size)
    {

    }
    Odwołania: 0
    ~GPObject()
    {
        basicTexture = null;
        basicSprite = null;
        GC.Collect();
    }
    Odwołania: 7
    public abstract void Render(RenderTexture drawTexture, Camera targetScene);
    Odwołania: 6
    public abstract void Update(List<Keyboard.Key> pressedKeys);
}
```

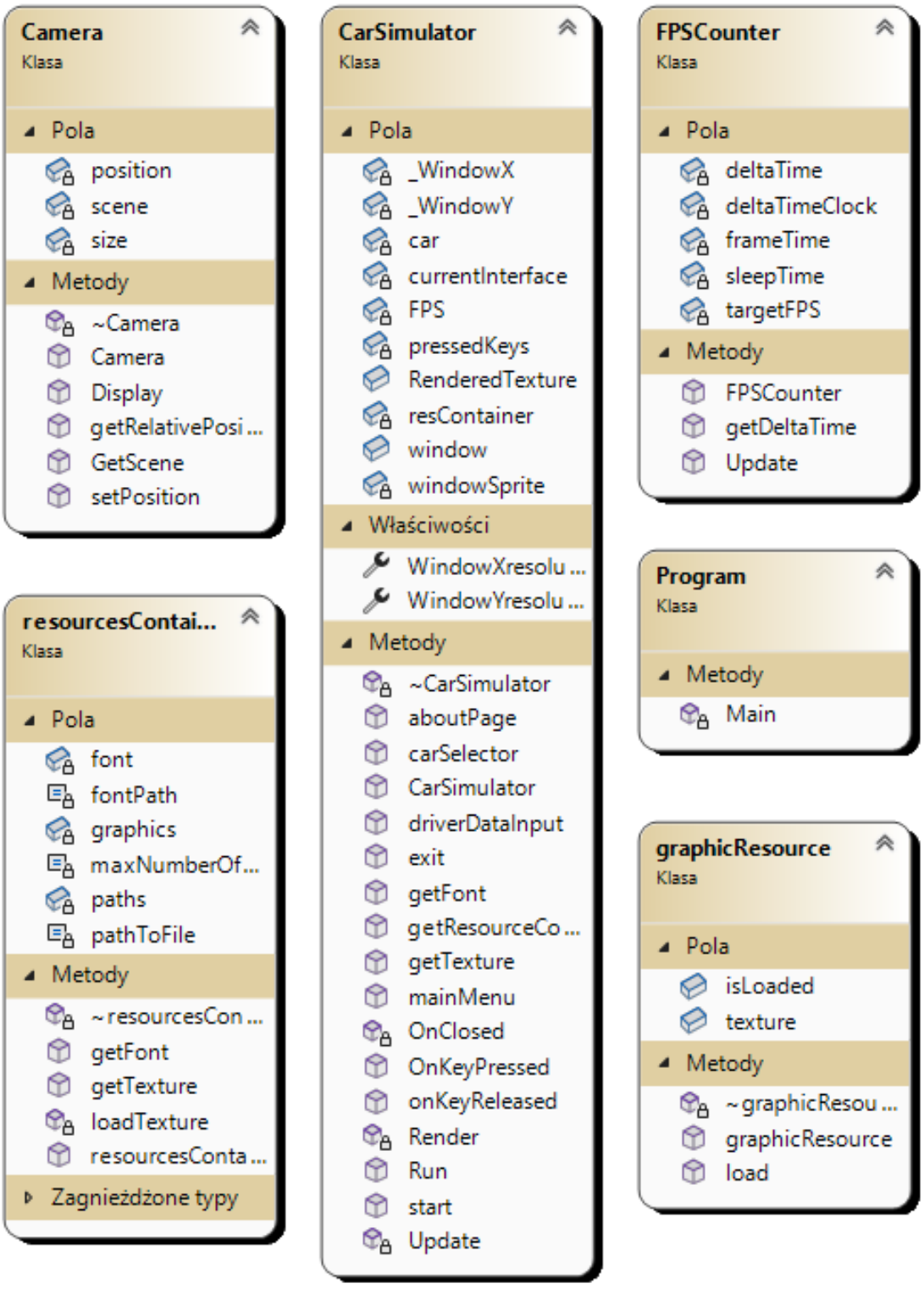
- **Obstacle** - Zawiera informacje na temat pozycji oraz rozmiaru obiektu oraz metody do obliczania kolizji oraz obsługi zdarzeń przy wykryciu kolizji.
- **GPObject** - Klasa abstrakcyjna, z której dziedziczą wszystkie obiekty elementu rozgrywki. Zawiera metody takie jak "Render" oraz "Update", które jak w przypadku podobnych wystąpień w innych klasach przyjmują obiekty typu "RenderTexture" oraz listę wciśniętych klawiszy. Dodatkowo metoda "Render" przyjmuje obiekt typu Camera, który zajmuje się obliczaniem relatywnej pozycji obiektów względem kamery.
- **maneuveringArea** - Mapa, po której porusza się pojazd, a dokładniej jest to tło wyświetlane pod pojazdem.
- **GPMapBarrier** - Obiekty w kształcie prostokąta stanowiące granice mapy. Po kolizji pojazd jest unieruchamiany w kierunku bariery. Pojazd może kontynuować jazdę w

przeciwnym kierunku co w dużym stopniu eliminuje problem "utknięcia" w krawędzi mapy.

- **GPCone** - Klasa reprezentująca pachołek. Niestety nie została do końca dopracowana przez co pachołek po kolizji przewraca się tylko w jednym kierunku, a sama metoda obliczania kolizji nie zawsze poprawnie działa.

- **CarObject** - Najbardziej rozbudowana klasa w całym programie. Klasa reprezentuje pojazd, którym kieruje uzytkownik. Pola klasy przechowują informacje na temat między innymi: pozycji tylnej i przedniej ośi pojazdu, przyśpieszenia, prędkości, obrotów silnika, stanu paliwa, stanu przepustnicy i tym podobne, a z metod można wyróżnić grupę zwracającą dane parametry pojazdu, metodę odpowiedzialną za rysowanie i metodę odpowiedzialną za obliczanie fizyki pojazdu oraz inne pomniejsze metody.

5.4 Inne klasy



- **Program** - Podstawowa klasa programu.

- **CarSimulator** - Główna klasa programu. Zawiera metody "Render" i "Update", w ktorych są rysowane i aktualizowane interfejsy. Metody "Render" i "Update" umieszczone są w nieskończonej pętli, która jest przerywana w przypadku zamknięcia okna lub użycia metody "exit". W polach klasy zawarte są informacje na temat rozdzielczości okna, pojazdu, obecnie wyświetlanego interfejsu, licznik klatek na sekunde, listy wciśniętych przycisków, zmienna typu RenderTexture, na której rysowane są interfejsy oraz klasy przechowującej zasoby graficzne.

- **ResoucesContainer** i **graphicResource** - Są to klasy odpowiedzialne za ładowanie, przechowywanie i udostępnianie w programie zasobów graficznych. Klasa "graphicResource" jest to pojedyńczy zasób, zawiera pole typu bool informujące czy tekstura jest załadowana oraz pole typu Texture, które przechowuje załadowaną teksture. Klasa "ResourcesContainer" posiada pole będące listą obiektów "graphicResource", pole będące listą ścieżek do plików graficznych i pole przechowujace czcionke. Dodatkowo w klasie zadeklarowany jest typ enum, który tłumaczy nazwy tekstur na typ int. W przypadku gdy jakiś obiekt chce otrzymać daną teksture to przy użyciu metody "getTexture" z odpowiednią wartością wspomnianego powyżej typu enum sprawdzane jest czy tekstura w liście na zadanej pozycji jest załadowana, jeżeli nie metoda podejmuje próbę jej załadowania. Jeżeli próba przebiegła pomyślnie, tekstura jest zwracana w miejsce wywołania.

- **Camera** - Głównym zadaniem klasy "Camera" jest obliczanie relatywnej pozycji obiektów rozgrywki oraz wyodrębnianie tych, które powinny zostać narysowane ponieważ znajdują się w obszarze kamery oraz te, które nie powinny być rysowane. Klasa "Camera" zawiera pole typu RenderTexture, na ktorym rysowane są obiekty.

- **FPSCounter** - Licznik klatek na sekunde. Ogranicza ilość rysowanych klatek, a co za tym idzie ilość aktualizacji zawartości programu do zadanej liczby. Oblicza też czas jaki upłynął pomiędzy dwiema klatkami.

6 Opis techniczny projektu

6.1 Wykorzystany język

Do stworzenia programu został wykorzystany język *C#*. *C#* jest wieloparadygmatowym językiem programowania obiektowego. Powstał w latach 1998-2001 nad językiem pracował zespół kierowany przez Andersa Hejlsberga. Język *C#* jest językiem wieloplatformowym, podobnym do języka *C++*. Nazwa języka jest nawiązaniem do nazwy *C++* gdzie cztery symbole plus tworzą kratke.
Program napisany w języku *C#* kompilowany jest do języka Common Intermediate Language, który jest specjalnym kodem pośredniego wykonywania w środowisku uruchomieniowym takim jak .NET.

6.2 Wykorzystane narzędzia

Program opiera się na platformie .NET, a do jego stworzenia zostało wykorzystane zintegrowane środowisko programistyczne "Visual Studio". Interfejs graficzny programu został napisany na potrzeby projektu i oprócz funkcji tworzenia okna oraz rysowania jest on w pełni autorski. Do stworzenia plików graficznych został wykorzystany program GIMP na licencji GNU. Poniżej przedstawione są główne biblioteki programistyczne wykorzystane w celu stworzenia projektu.

- **SFML 2.4** - Jest głównym narzędziem wykorzystanym przy tworzeniu programu. SFML jest multimedialną biblioteką klas umożliwiających tworzenie okien oraz rysowanie Sprite'ów. Oprócz tego biblioteka SFML obsługuje klawiature i mysz, funkcje audio, oraz funkcje związane z łączem internetowym. SFML jest darmową biblioteką open source napisaną w języku C++ przez Laurent Gomila wraz z zespołem programistów.

- **EPPlus** - Jest biblioteką dla platformy .NET umożliwiającą ładowanie, edytowanie i zapisywanie plików pakietu excel. Biblioteka EPPlus jest darmowa dla niekomercyjnych zastosowań.

- **Klasy wbudowane** - Oprócz wyżej wymienionych bibliotek w projekcie zostały wykorzystane klasy wbudowane w język *C#* takie jak klasa "List" umożliwiająca tworzenie list typów generycznych.

6.3 Minimalne wymagania sprzętowe

Program został stworzony na systemie operacyjnym Windows 10 na środowisku .NET w wersji 7.0, więc, wymienione parametry są zalecaną konfiguracją środowiska, na którym będzie program uruchamiany. Program może działać na innych wersjach systemu windows, innych platformach OS lub na innej wersji środowiska .NET jedak jego zachowanie może stać się nieprzewidywalne co w konsekwencji może prowadzić do błędów. Po przeprowadzeniu kilku sesji testowych ustalono, że program nie przekracza zużycia pamięci ram powyżej 135 MB. Jeżeli chodzi o procesor to przy sześciordzeniowym Intel Core i5-10400F 2.90GHz zużycie

nie przekroczyło 2% mocy procesora. Uznając, że zużycie procesora jest równomiernie rozłożone na wszystkie rdzenie z obliczeń $2.9GHz * 6 * 2\%$ otrzymujemy wynik $0.348GHz$. Do powyższych wymagań należy dodać wymagania sprzętowe dla wykorzystywanego systemu operacyjnego, w tym wypadku jest to Windows 10. Wykorzystanie procsora graficznego o mocy 1.3GHz na karcie graficznej NVIDIA GeForce RTX 2060 osiąga wartość 18%. Wartość wykorzystania procesora graficznego i pamięci procesora graficznego jest niska więc program bez problemu będzie działał z słabszymi kartami graficznymi. Należy jednak uznać, że NVIDIA GeForce RTX 2060 jest docelową kartą graficzną. Jeżeli chodzi o wymagania dotyczące obsługi audio lub połączenia internetowego to brak takowych wymagań gdyż program nie obsługuje dzwięku ani nie korzysta z połączenia internetowego. Finalne wymagania sprzętowe wyglądają następująco:

- **System Operacyjny:** Windows 10 lub nowszy z środowiskiem .NET w wersji conajmniej 7.0
- **Procesor CPU:** 1.35 GHz lub szybszy
- **Pamięć RAM:** Conajmniej 1280 MB (Zaokrąglone do najbliższej najwyższej wartości)
- **Karta graficzna:** NVIDIA GeForce RTX 2060 lub podobna
- **Miejsce na dysku dla programu:** Conajmniej 16MB
- **Karta sieciowa / dźwiękowa:** Brak wymagań

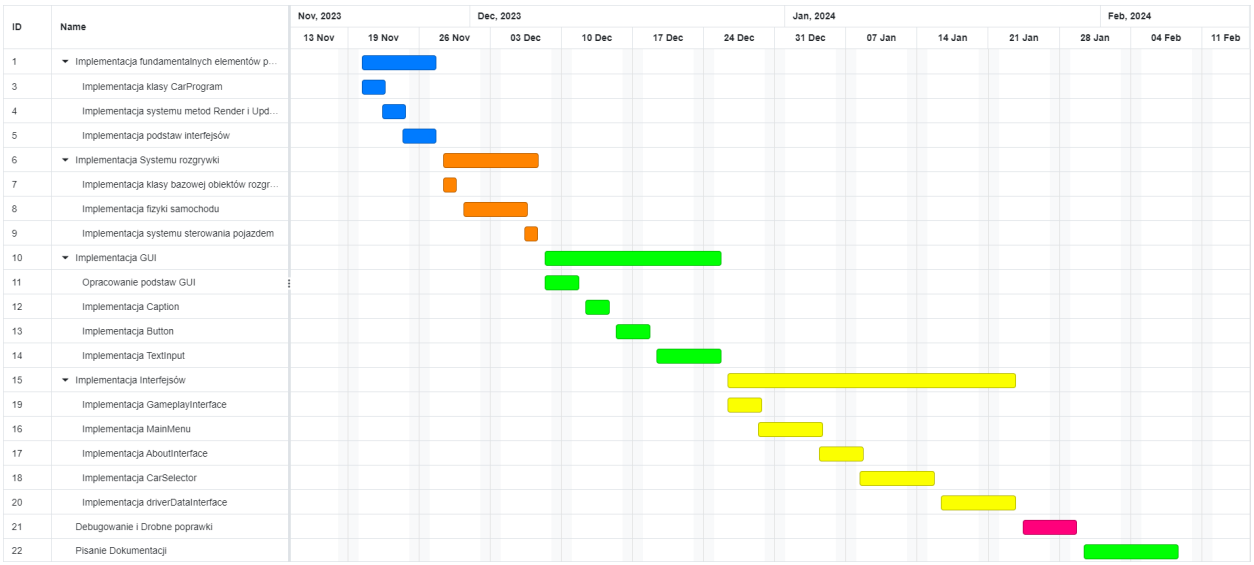
6.4 Zarządzanie danymi

Program operuje na 4 plikach z danymi:

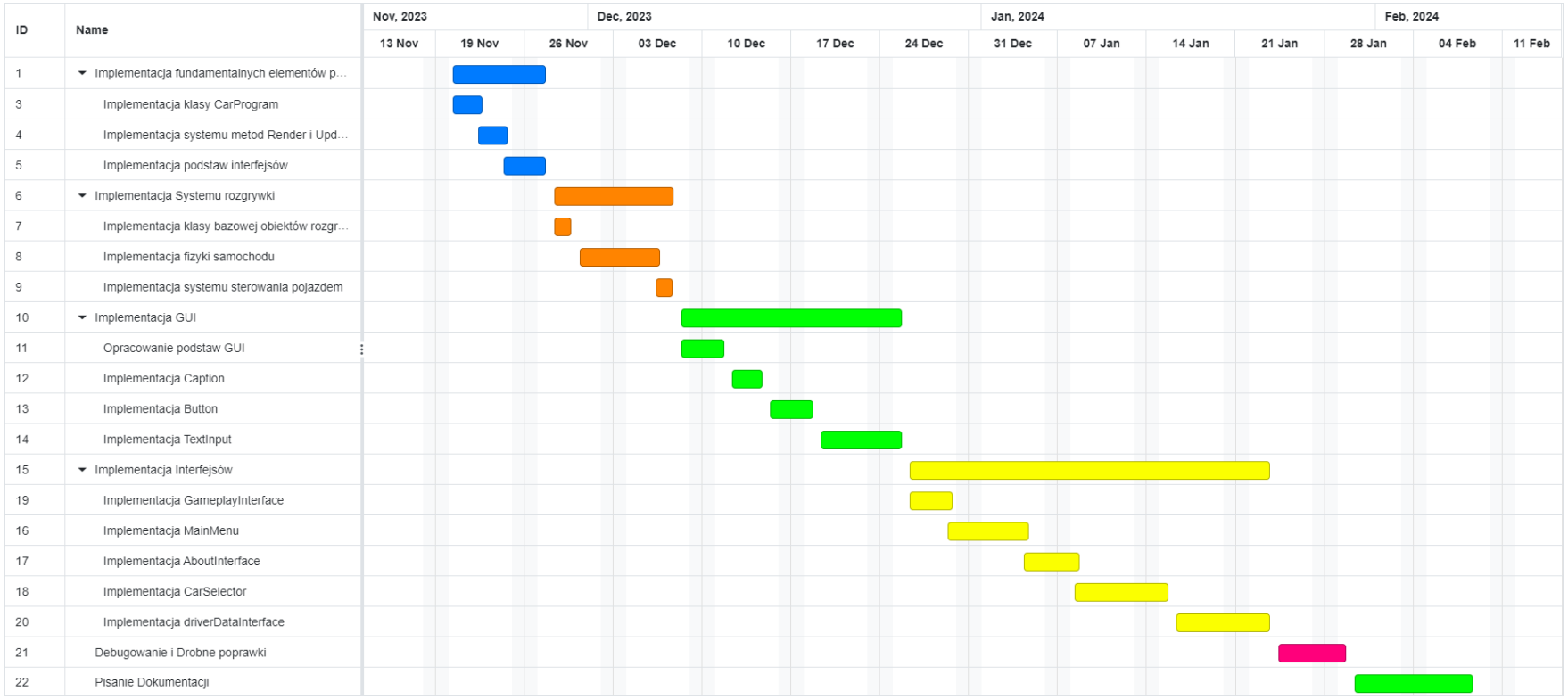
- **aboutProgram.txt** z którego wczytuje najpierw liczbe linii, a następnie wspomniane linie danych z pliku.
- **graphicsPaths.txt** z którego wczytuje końcówki scieżek do określonych plików graficznych. Każda linia odpowiada jednej końcówce scieżki pliku graficznego o z góry określonej pozycji.
- **carParameters.xlsx** z którego program wczytuje parametry dla wybranego pojazdu. Każda kolumna to inny pojazd natomiast każdy wiersz to inny parametr. Liczba wierszy i kolumn jest z góry ustalona w programie.
- **driversData.xlsx** z którego program wczytuje informacje dotyczące kont użytkowników. W pliku są 4 kolumny: login, password, licensePlate, i drivenDistance, a na każde konto użytkownika przypada jeden wiersz. W przypadku próby dodania nowego konta - wiersza wyszukiwany jest pierwszy pusty wiersz, w którego miejsce wstawiane są dane nowego użytkownika.

Wszystkie próby odczytu lub zapisu do pliku zabezpieczone są obsługą wyjątków try-catch. W przypadku błędu odczytu wykonywanu jest wyjątek, a program może działać dalej.

7 Diagram Gantta



Powyżej został przedstawiony diagram Gantta dla opisywanego projektu. Są w nim wymienione kluczowe etapy produkcji programu wraz z przedziałami czasowymi. Diagram Gantta został wykonany przy użyciu strony: onlinegantt.com


ID
Name
Nov, 2023
Dec, 2023
Jan, 2024
Feb, 2024
13 Nov
19 Nov
26 Nov
03 Dec
10 Dec
17 Dec
24 Dec
31 Dec
07 Jan
14 Jan
21 Jan
28 Jan
04 Feb
11 Feb
1 Implementacja fundamentalnych elementów p…
3 Implementacja klasy CarProgram
4 Implementacja systemu metod Render i Upd…
5 Implementacja podstaw interfejsów
6 Implementacja Systemu rozgrywki
7 Implementacja klasy bazowej obiektów rozgr…
8 Implementacja fizyki samochodu
9 Implementacja systemu sterowania pojazdem
10 Implementacja GUI
11 Opracowanie podstaw GUI
12 Implementacja Caption
13 Implementacja Button
14 Implementacja TextInput
15 Implementacja Interfejsów
19 Implementacja GameplayInterface
16 Implementacja MainMenu
17 Implementacja AboutInterface
18 Implementacja CarSelector
20 Implementacja driverDataInterface
21 Debugowanie i Drobne poprawki
22 Pisanie Dokumentacji

8 Repozytorium i system kontroli wersji

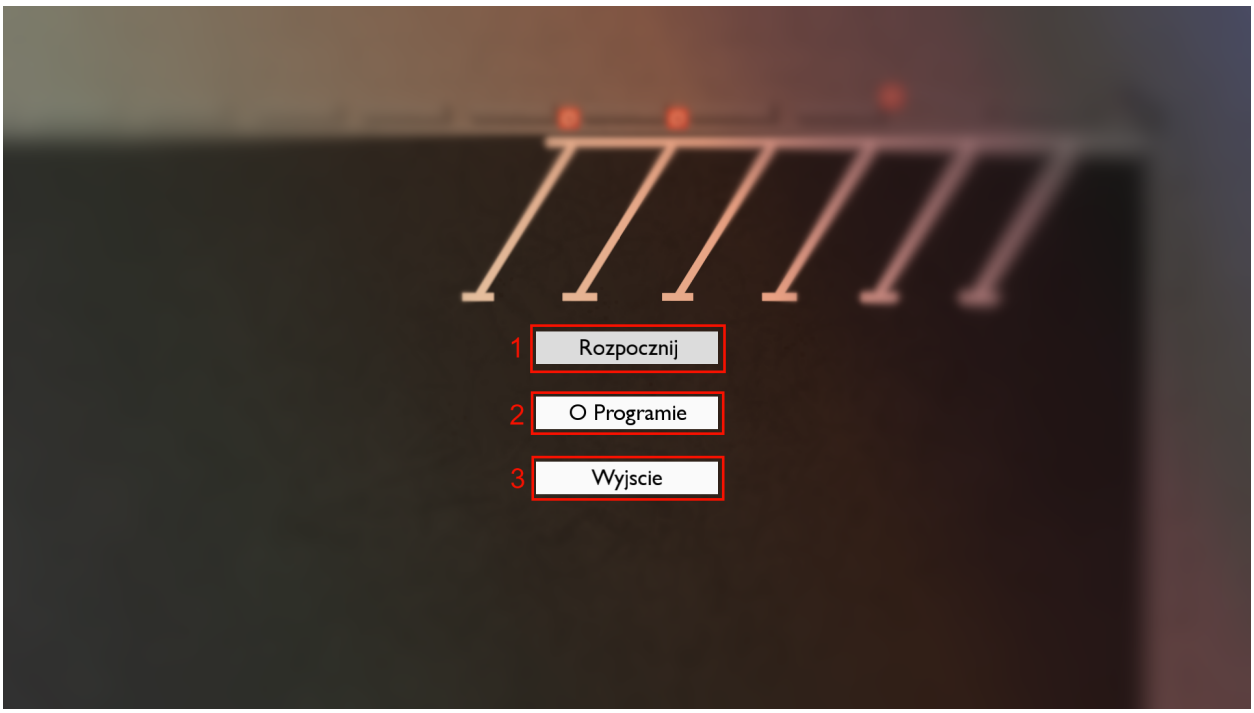
Repozytorium oraz systemem kontroli wersjii zajmuje się portal github.com, gdzie w moim Repozytorium znajdują się wszystkie pliki klas, wykonywalne oraz zasoby wykorzystywane przez program. Obecna wersja programu została otagowana jako v1.0 - Wersja Końcowa Projektu.

Repozytorium

9 Prezentacja warstwy użytkowej aplikacji

W tym dziale zostanie zaprezentowana warstwa użytkowa aplikacji. Warstwę użytkową można podzielić na pięć części: Menu główne, O programie, System weryfikacji danych kierowcy, Menu wyboru pojazdu, Etap rozgrywki.

9.1 Menu główne

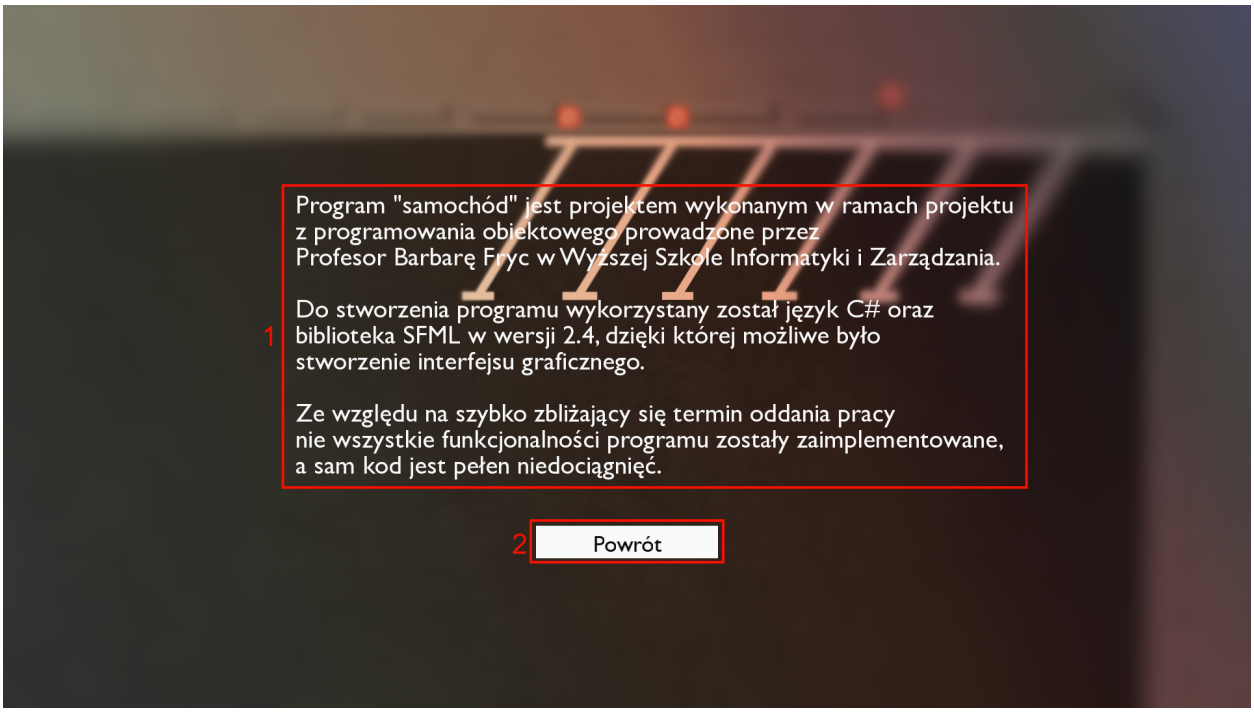


Rysunek 1: Menu główne programu

Powyżej przedstawione jest menu główne programu. Jest to pierwszy interfejs, który można ujrzeć po uruchomieniu programu. Można na nim wyszczególnić trzy elementy:

- **1** - Przycisk "Rozpocznij" przekieruje użytkownika do interfejsu logowania do konta kierowcy. Na przycisku znajduje się kursor myszy co jest komunikowane szarym kolorem tła przycisku.
- **2** - Przycisk "O Programie" przekieruje użytkownika od interfejsu wyświetlającego krótki opis programu.
- **3** - Przycisk "Wyjście" kończy działanie programu

9.2 O programie

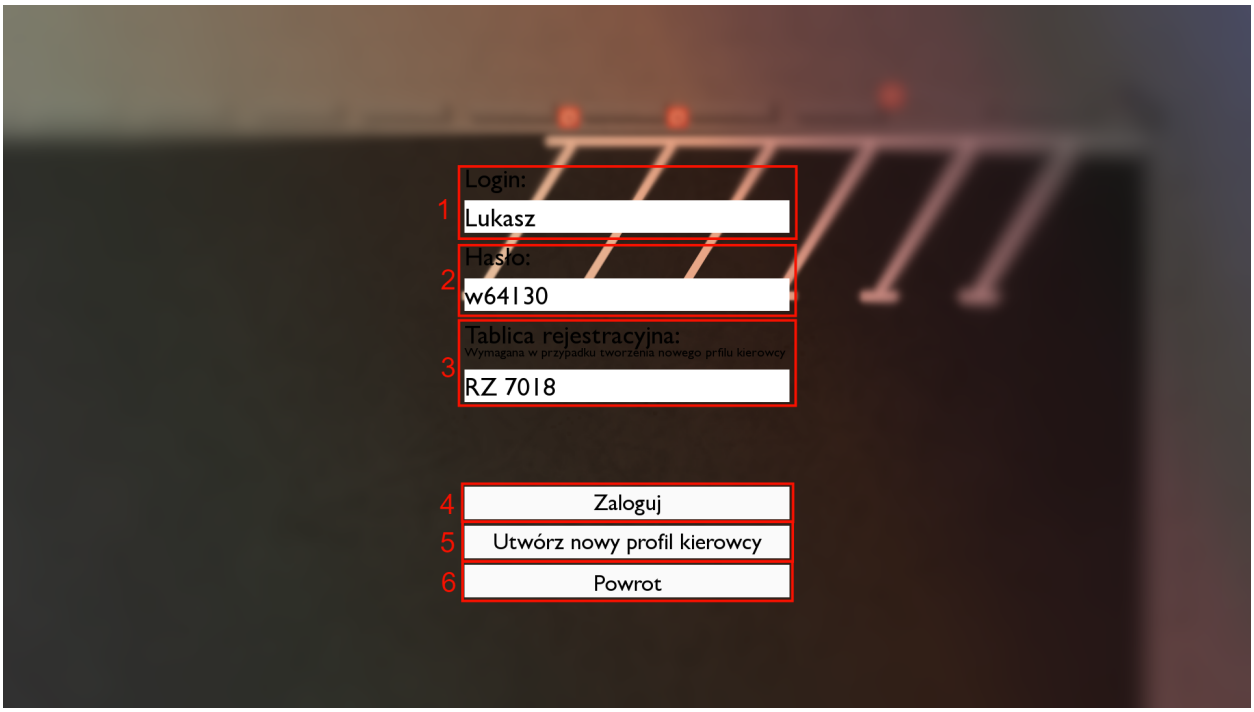


Rysunek 2: O programie

Powyżej przedstawiony jest interfejs "O programie". Interfejs wyświetla krótki opis programu, który jest wczytywany z pliku tekstowego.

- **1** - Krótki opis programu wczytywany bezpośrednio z pliku .txt
- **2** - Przycisk "Powrót" przekierowuje użytkownika spowrotem do menu głównego

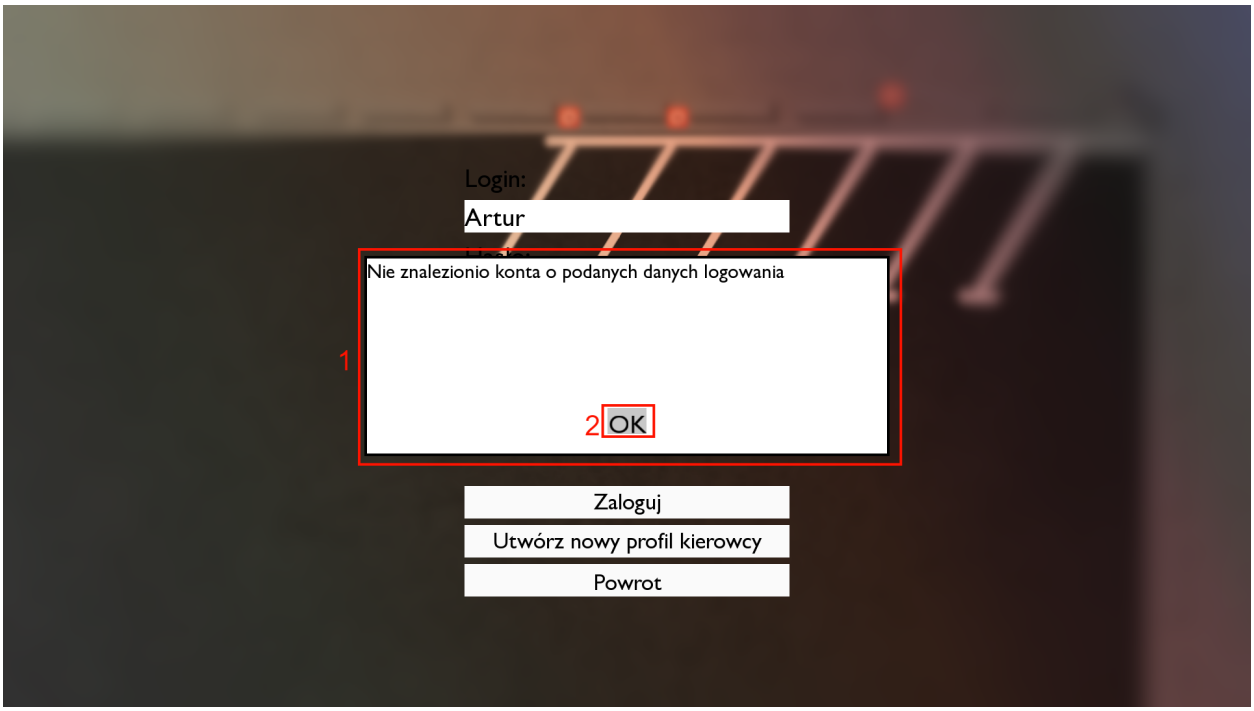
9.3 System weryfikacji danych kierowcy



Rysunek 3: Interfejs logowania kierowcy

Powyżej przedstawiony jest interfejs wprowadzania danych logowania kierowcy. Użytkownik może utworzyć nowe konto, zalogować się na już istniejące lub powrócić do menu głównego. Zalogowanie się do konta jest wymagane aby móc przejść do menu wyboru pojazdu. Pola wprowadzania tekstu są stworzone od podstaw i nie posiadają zaimplementowanej obsługi znaków specjalnych czy klawiatury numerycznej.

- **1** - Pole wprowadzania loginu konta
- **2** - Pole wprowadzania hasła konta
- **3** - Pole wprwoadzania numeru tablicy rejestracyjnej konta
- **4** - Przycisk "Zaloguj" powoduje powrównanie wprowadzonego loginu i hasła z istniejącymi kontami, jeżeli znaleziono konto o pasujących danych użytkownik jest przekierowywany do menu wyboru pojazdu
- **5** - Przycisk "Utwórz nowy profil kierowcy" powoduje porównanie podanych danych z kontami przechowywanymi w pliku .xlsx, jeżeli profil o takich samych danych nie został znaleziony to nowy profil zostanie utworzony
- **6** - Przycisk "Powrót" przekierowuje użytkownika do menu głównego

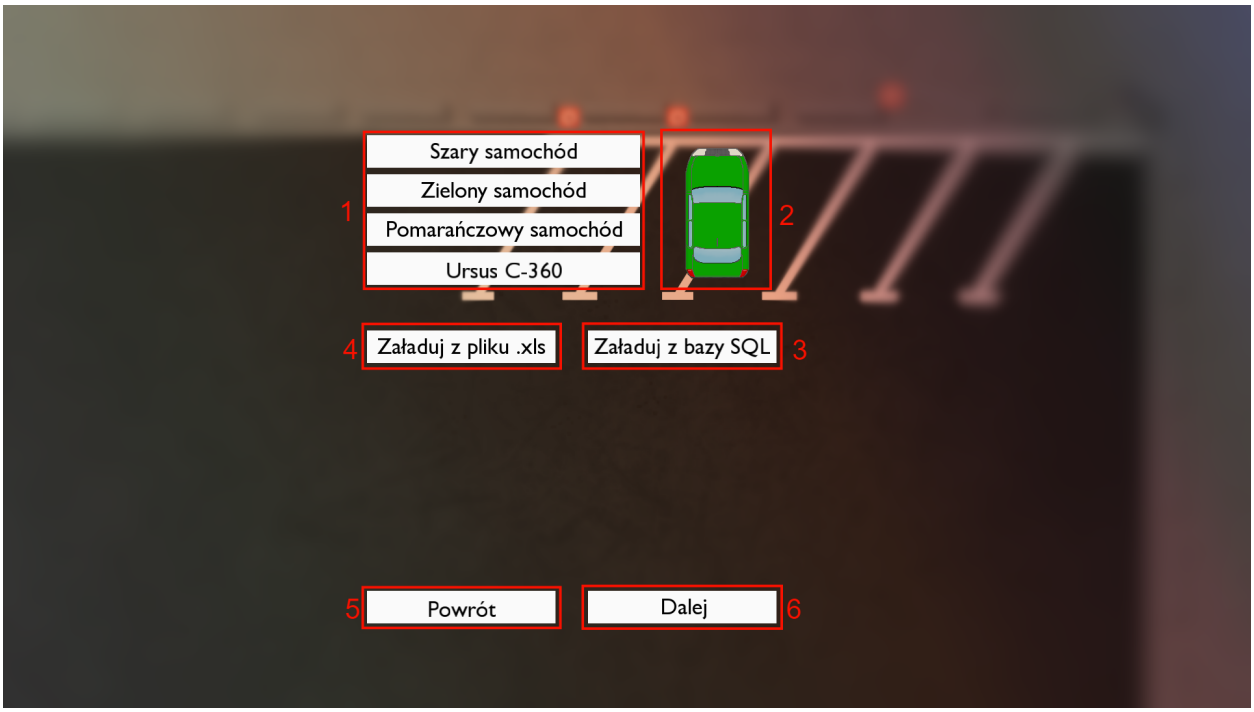


Rysunek 4: Okienko z informacją dla użytkownika

W przypadku wprowadzenia błędnych danych, próbie utworzenia już istniejącego konta lub błędu odczytu pliku użytkownik zostanie poinformowany stosownym komunikatem w okienku.

- **1** - "Okienko z komunikatem o wprowadzeniu błędnych danych"
- **2** - Przycisk "Ok" zamykający okienko

9.4 Menu wyboru pojazdu

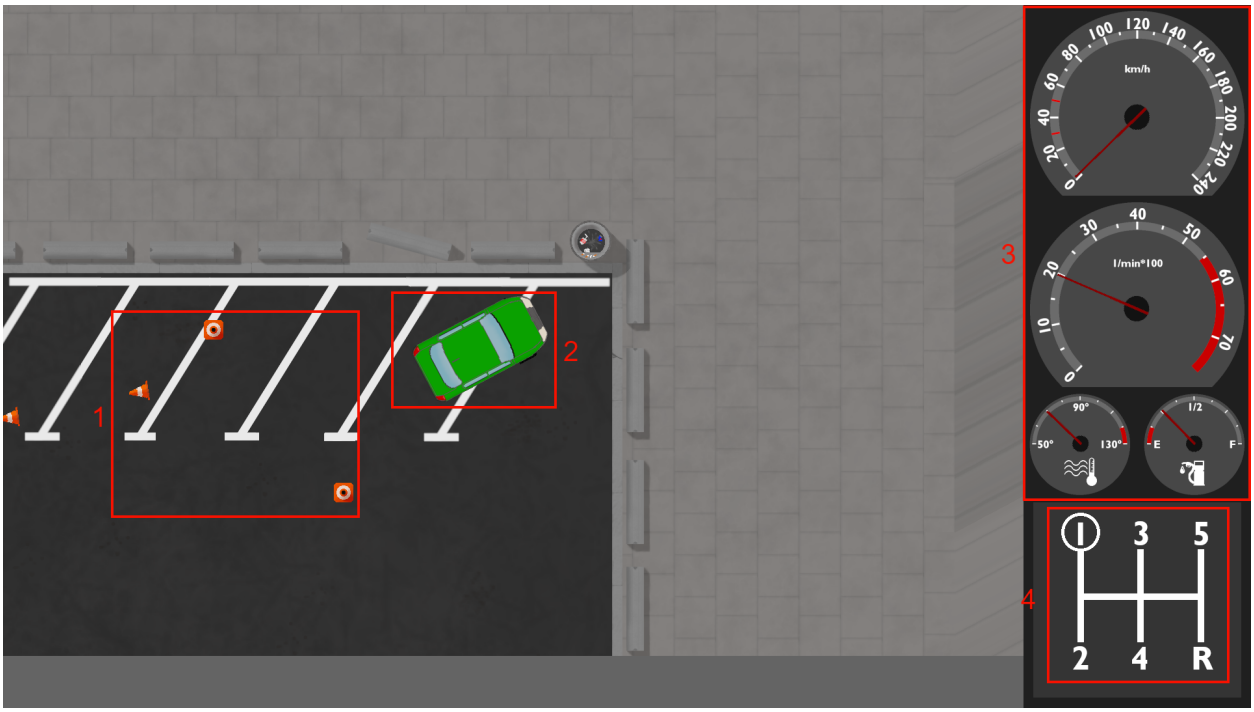


Rysunek 5: Interfejs wyboru pojazdu

Powyżej przedstawiony jest interfejs wyboru pojazdu. Użytkownik ma możliwość wyboru pojazdu z czterech dostepnych: Szarego, pomarańczowego, zielonego samochodu i Ursusa C-360. Każdy pojazd różni się parametrami takimi jak promień skrętu, pojemność silnika, zużycie paliwa i tym podobne. Aby móc zacząć rozgrywke użytkownik musi załadować parametry pojazdu. Niestety w obecnej wersji niestety można załadować dane tylko z pliku .xlsx, a przycisk "Załaduj z bazy SQL" nie posiada zaimplementowanej funkcjonalności. Dopiero pomyślne załadowanie parametrów pojazdu skutkuje pojawieniem się przycisku "Dalej".

- **1** - Przyciski wyboru pojazdu
- **2** - Obok przyciskow wyświetlany jest becnie wybrany pojazd
- **3** - Przycisk "Załaduj z bazy SQL", obecnie nie posiada funkcjonalności
- **4** - Przycisk "Załaduj z pliku .xls" powoduje podjęcie próby załadowania parametrów pojazdu z pliku .xlsx
- **5** - Przycisk "Powrót" powoduje zmiane interfejsu na interfejs logowania
- **6** - Przycisk "Dalej", dostępny jedynie po pomyślnym załadowaniu parametrów programu z pliku .xlsx. Powoduje przejście do interfejsu prowadzenia pojazdu

9.5 Etap rozgrywki



Rysunek 6: Interfejs etapu rozgrywki

Powyższa grafika przedstawia interfejs rozgrywki. Na obrazku można zobaczyć samochód, którym kieruje użytkownik, samochód znajduje się na placu manewrowym otoczonym barierkami (Samochód w normalnych warunkach nie jest w stanie opuścić placu manewrowego), grupę pachołków oraz interfejs samochodu po prawej. Samochód przemieszcza się według rozkazów użytkownika, a kamera cały czas porusza się za samochodem. Skręt samochodu oprócz obrotu samochodu jest również sygnalizowany obrotem przednich kół w odpowiednim kierunku. Pachołki są osobnymi obiektami rozgrywki, z którymi gracz może wchodzić w interakcje taranując je. Wjechanie na pachołek powoduje wywrócenie. Metoda odpowiedzialna za obliczanie kolizji pachołków z samochodem jest obarczona kilkoma błędami przez co pachołki nie zawsze reagują na kolizje tak jak powinny. Po prawej stronie znajduje się interfejs graficzny z parametrami samochodu. Od góry: Wskaźnik prędkości samochodu, wskaźnik obrotów silnika i po lewej wskaźnik temperatury płynu chlodniczego, a po prawej wskaźnik poziomu paliwa. Na samym dole znajduje się grafika rozkładu biegów, która informuje użytkownika o obecnie wbitym biegu białym okręgiem w odpowiedniej pozycji. Niefortunnie jedynym wyjściem z tego interfejsu jest zamknięcie całego programu.

- **1** - grupa pachołków, z którą gracz może wchodzić w interakcje
- **2** - Samochód kierowany przez gracza
- **3** - Zegary wskazujące parametry samochodu

- **4** - "Skrzynia biegów" przedstawiająca obednie wbity bieg.

Sterowanie samochodem wygląda nastepująco:

- **Strzałka w lewo**, **Strzałka w prawo** - Obrót kierownicą w wybranym kierunku
- **Strzałka w górę** - Otwarcie przepustnicy / Dodanie gazu
- **Strzałka w dół** - Zamknięcie przepustnicy / Zmniejszenie gazu
- **E** - Aktywacja rozrusznika / Włączenie silnika
- **˜**, **N** - Wrzucenie biegu neutralnego
- **1** - Wrzucenie pierwszego biegu
- **2** - Wrzucenie drugiego biegu
- **3** - Wrzucenie trzeciego biegu
- **4** - Wrzucenie czwartego biegu
- **5** - Wrzucenie piątego biegu
- **R** - Wrzucenie biegu wstecznego

Elementy symulacji samochodu:

- **Prędkość samochodu obliczana jest na postawie parametrów takich jak obroty silnika, obecnie wbity bieg czy waga pojazdu**
- **Obroty silnika obliczane są na podstawie stopnia otwarcia przepustnicy i pojemności skokowej**
- **Silnik ze zbyt małymi obrotami samoczynnie zgaśnie**
- **Silnik ze zbyt dużymi obrotami będzie samoczynnie dławiony**
- **Zaimplementowana została symulacja nagrzewania sie płynu chłodniczego, a w przypadku awarii płyn może osiągnąć temperature powyżej 90°**
- **Zużycie paliwa obliczane jest na podstawie stopnia otwarcia przepustnicy, a gdy skończy się paliwo to silnik samoczynnie zgaśnie**

10 Podsumowanie

10.1 Zrealizowane prace

Głównym założeniem projektu był stworzenie aplikacji symulującej działanie pojazdu samochodowego. Głównymi aspektami symuliacji miały być: Symulacja działania silnika spalinowego, symulacja jazdy / kierowania samochodem, system symulacji uszkodzeń pojazdu takich jak zatarty silnik lub przegrzany płyn chłodniczy. Z pośród wymienionych powyżeń założeń znaczną część udało się zrealizować implementując fizyke silnika spalinoweg i pojazdu samochodowego oraz system symulacji nagrzewania się płynu chłodniczego i zużywania paliwa.
Znaczna częśc czasu pracy nad projektem została poświęcona na implementacje fizyki pojazdu samochodowego. Okazało się to być trudniejszym zadaniem niż zakładałem. Dużo nakładu pracy zostało też zainwestowane w stworzenie od podstaw interfejsu graficznego. Uznałem, że ciężko będzie stworzyć program symulujący prowadzenie pojazdu samochodowego bez graficznego interfejsu użytkownika, jednak stworzenie całego interfejsu graficznego od podstaw okazało się być zajęciem zbyt czasochłonnym aby w pełni skupić się na postawionych założeniach, a gdy uświadomiłem to sobie było już za późno na zmianę decyzji ze względu na ilość pracy jaką zainwestowałem w projekt. Ostatecznie założenia projektowe w znacznym stopniu zostały zrealizowane, a całość aplikacji udało się doprowadzić do stanu, w którym może zostać zaprezentowana.

10.2 Dalsze plany dotyczące projektu

W trakcie realizacji projektu moja wiedza na temat programowania, a zwłaszcza języka C# znacznie się pogłębiła i muszę przyznać, że pisanie aplikacji i obserwowanie jak program staje się coraz bardziej interaktywny sprawiło mi sporo przyjemności, lecz obecnie nie planuje dalszych prac nad omawianą aplikacją.

11 Literatura

- Jacek Matulewski: lekcje programowania : praktyczna nauka programowania dla platform .NET i .NET Core
- A. Stellman, J. Greene, C#. Rusz głową!, Helion, Gliwice, 2022
- SFML - Tutorials