Projekt: Konfiguracja koloru samochodu w środowisku VR

Kacper Łokietek, Jakub Kumer

---

1. Dokumentacja wstępna

Projekt polega na stworzeniu aplikacji VR umożliwiającej zmianę koloru samochodu za pomocą trzech suwaków RGB (red, green, blue). Po ustawieniu wybranego koloru i kliknięciu przycisku, pojazd zmienia swój kolor, co pozwala na wizualizację w czasie rzeczywistym.

Argumenty uzasadniające zastosowanie wybranej technologii:

- Unity wraz z XR Interaction Toolkit pozwala na szybkie tworzenie i testowanie aplikacji w środowisku VR, co jest kluczowe dla tego projektu.
- Technologia VR zapewnia wysoki poziom immersji i możliwość interakcji, co sprawia, że aplikacja jest atrakcyjna dla użytkowników.

Potencjalne cele stworzenia systemu:

- Umożliwienie personalizacji wyglądu samochodu w realistycznym środowisku VR.
- Wykorzystanie aplikacji w branży motoryzacyjnej jako narzędzia do prezentacji produktów klientom.

Potencjalni odbiorcy systemu:

- Klienci zainteresowani zakupem samochodu, którzy chcą zobaczyć pojazd w różnych konfiguracjach kolorystycznych.
- Firmy motoryzacyjne chcące zaprezentować swoje pojazdy w nowoczesny i interaktywny sposób.

Przykłady szczególnej wartości i użyteczności projektu:

- Intuicyjny interfejs użytkownika umożliwiający szybkie dostosowanie kolorystyki pojazdu.
- Realistyczne odwzorowanie kolorów i środowiska VR pozwala na dokładne ocenienie estetyki wybranego koloru.

---

2. Zarys specyfikacji

Wymagania funkcjonalne i kryteria akceptacji:

1. **Ustawianie wartości RGB za pomocą trzech suwaków.**
    - Kryterium akceptacji: Suwaki zmieniają wartości od 0 do 255 i aktualizują podgląd koloru.
2. **Wyświetlanie podglądu aktualnego koloru w interfejsie.**
    - Kryterium akceptacji: Kolor podglądu zmienia się dynamicznie po zmianie wartości na suwakach.

3. **Zmiana koloru samochodu po kliknięciu przycisku.**
    - Kryterium akceptacji: Kolor pojazdu zmienia się w ciągu 3 sekund do wybranego koloru.
4. **Płynna animacja zmiany koloru samochodu.**
    - Kryterium akceptacji: Zmiana koloru przebiega płynnie bez widocznych przeskoków.
5. **Interaktywność w środowisku VR (np. obsługa kontrolerów VR).**
    - Kryterium akceptacji: Aplikacja reaguje na wejścia użytkownika w środowisku VR.
6. **Zapis i wczytywanie koloru samochodu.**
    - Kryterium akceptacji: Kolor pojazdu jest automatycznie zapisywany podczas zamykania aplikacji i wczytywany przy jej ponownym uruchomieniu.

Wymagania niefunkcjonalne:

- Aplikacja powinna działać z płynnością minimum 60 FPS w środowisku VR.
- Maksymalny czas reakcji aplikacji na wejścia użytkownika: 0,5 sekundy.

Priorytetyzacja wymagań i wybór krytycznych:

- Krytyczne wymagania:
    1. **Zmiana koloru pojazdu.**
        - Wyjaśnienie: Jest to główna funkcjonalność aplikacji.
    2. **Intuicyjna obsługa interfejsu (suwaki RGB i przycisk).**
        - Wyjaśnienie: Bez tej funkcjonalności użytkownik nie będzie w stanie skorzystać z aplikacji.
    3. **Zapis i wczytywanie koloru.**
        - Wyjaśnienie: Funkcjonalność ta zapewnia ciągłość doświadczenia użytkownika.

---

3. **Zarys elementów projektu**

Decyzje projektowe zwiększające immersję Użytkownika:

1. **Wykorzystanie realistycznego modelu samochodu z Asset Store.**
    - Uzasadnienie: Zapewnia lepsze odwzorowanie rzeczywistości.
2. **Dodanie skyboxa symulującego realistyczne środowisko.**
    - Uzasadnienie: Zwiększa poczucie przestrzeni w środowisku VR.
3. **Płynne animacje zmiany koloru.**
    - Uzasadnienie: Umożliwia naturalną percepcję zmian wizualnych.

4. **Użycie tekstury podłoża dla lepszego odczucia przestrzeni.**
    - Uzasadnienie: Zmniejsza wrażenie pustki w aplikacji VR.
5. **Zapis koloru samochodu i jego automatyczne wczytywanie przy uruchomieniu aplikacji.**
    - Uzasadnienie: Ułatwia użytkownikowi kontynuowanie pracy.

Prototyp aplikacji:

- **Funkcjonalności:**
    1. Suwaki RGB do ustawiania koloru.
    2. Przycisk zmieniający kolor samochodu.
    3. Zapis i wczytywanie ustawionego koloru.

---

4. **Ogólny zarys elementów architektury**

Opis ogólnej architektury aplikacji:

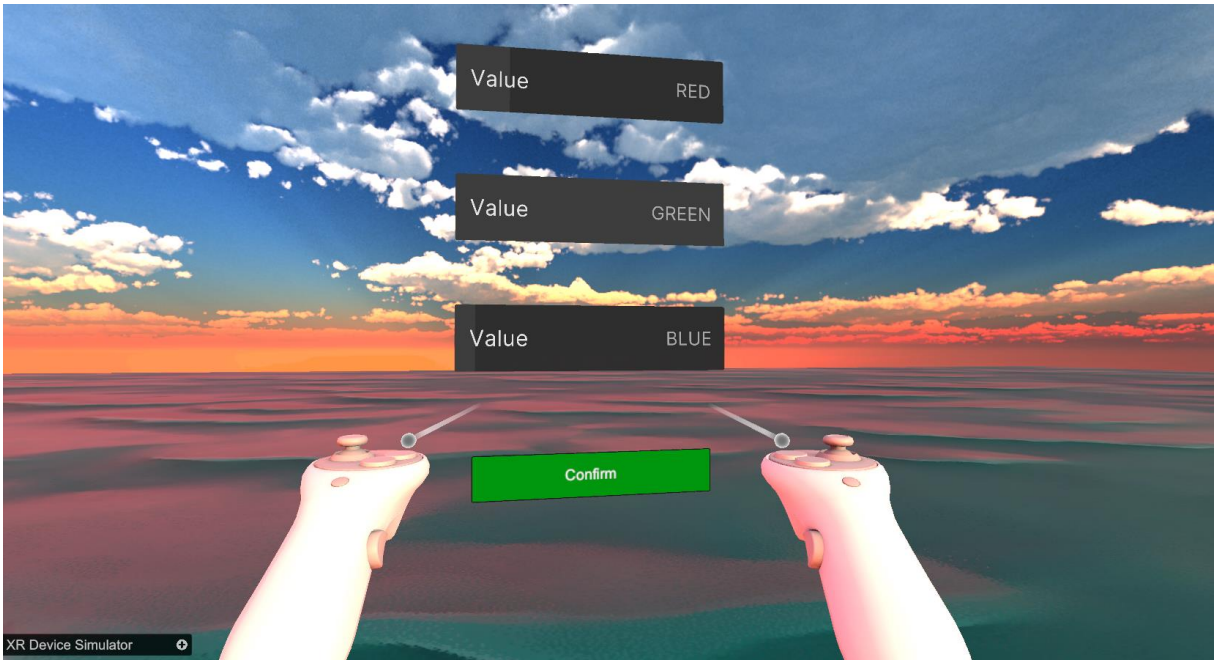
- **Komponenty:**
    - Interfejs użytkownika: Suwaki, przycisk zmiany koloru, podgląd koloru.
    - Logika aplikacji: Skrypt ColorChanger do zarządzania zmianą koloru.
    - Zapis stanu: Skrypt ColorSaver do zapisu i wczytywania koloru samochodu.
    - Model samochodu: Obiekt 3D pobrany z Asset Store.

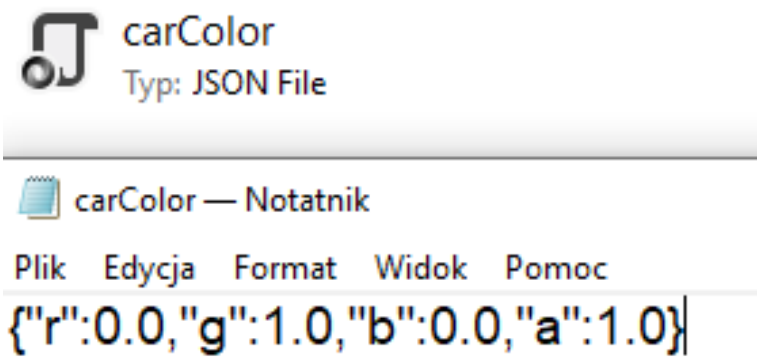
Sugerowany stos technologiczny:

- Unity 3D z XR Interaction Toolkit.
- VR headset (np. Oculus Quest 2).
- Assety: model samochodu, tekstura podłoża, skybox.
- Biblioteka System.IO do obsługi plików JSON dla zapisu i odczytu ustawionych kolorów.

---

5. **Demonstracja specyfikacji i kodu źródłowego**





Odczyt zapisu w pliku carColor.JSON

Skrypt Color Changer.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UI;

public class ColorChanger : MonoBehaviour
{
   [SerializeField] Slider sliderRed;
   [SerializeField] Slider sliderGreen;
   [SerializeField] Slider sliderBlue;
   [SerializeField] Color color;
   [SerializeField] Material material;
   [SerializeField] Image colorImage;
   [SerializeField] float TimeToChange = 3f;
   [SerializeField] Button ChangeColorButton;
   void Start()
   {
      color.r = sliderRed.value;
      color.g = sliderGreen.value;
      color.b = sliderBlue.value;
      ChangeMaterialColor();
      sliderRed.onValueChanged.AddListener(UpdateRed);
      sliderGreen.onValueChanged.AddListener(UpdateGreen);
      sliderBlue.onValueChanged.AddListener(UpdateBlue);
      ChangeColorButton.onClick.AddListener(Change);
   }
   private void Change()
   {
      StartCoroutine(ChangeMaterialColor());
```

```
}
private void UpdateImage()
{
   colorImage.color = color;
}
private void UpdateRed(float val)
{
    color.r = val;
   UpdateImage();
}
private void UpdateGreen(float val)
{
    color.g = val;
   UpdateImage();
}
private void UpdateBlue(float val)
{
    color.b = val;
   UpdateImage();
}
IEnumerator ChangeMaterialColor()
{
   Color current = material.color;
   float timePassed = 0;
   while (timePassed<TimeToChange)
   {
      current= Color.Lerp(material.color, color, timePassed/TimeToChange );
      material.color= current;
      yield return new WaitForSeconds(0.05f);
      timePassed += 0.05f;
   }
```

```
    }
}
```

Skrypt ColorSaver.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;

public class ColorSaver : MonoBehaviour
{
    // Ścieżka do pliku z zapisanym kolorem
    private string filePath;

    // Referencja do obiektu, którego kolor zmieniamy
    public Renderer carRenderer;

    void Start()
    {
        // Ustawienie ścieżki zapisu w katalogu domowym użytkownika
        filePath = Path.Combine(Application.persistentDataPath, "carColor.json");

        // Wczytanie ostatniego koloru przy starcie aplikacji
        LoadColor();
    }

    public void SaveColor(Color color)
    {
        // Konwersja koloru do formatu JSON
        ColorData colorData = new ColorData(color);
        string json = JsonUtility.ToJson(colorData);
```

```
    // Zapis do pliku
    File.WriteAllText(filePath, json);

    Debug.Log("Kolor zapisany: " + json);
}

public Color LoadColor()
{
    if (File.Exists(filePath))
    {
        string json = File.ReadAllText(filePath);
        ColorData colorData = JsonUtility.FromJson<ColorData>(json);
        return colorData.ToColor();
    }
    else
    {
        Debug.Log("Brak zapisanego koloru.");
        return Color.white; // Domyślny kolor, jeśli plik nie istnieje
    }
}

[System.Serializable]
public class ColorData
{
    public float r;
    public float g;
    public float b;
    public float a;

    public ColorData(Color color)
    {
```

```
            r = color.r;
            g = color.g;
            b = color.b;
            a = color.a;
        }

        public Color ToColor()
        {
            return new Color(r, g, b, a);
        }
    }
}
```