AKADEMIA TARNOWSKA

Tarnów, 07.06.2023r.

Grafika 3D i programowanie kart graficznych II

Prowadzący:
dr inż. Jędrzej Byrski

Temat projektu:
Kostka Rubika.

Imiona, nazwiska i numery indeksów członków grupy:
- Jakub Dojka, 35197
- Paweł Kamiński, 35678
- Kamil Dereń, 35196

Cel projektu:
Celem projektu jest stworzenie aplikacji w języku C++ z wykorzystaniem biblioteki OpenGL, która umożliwi użytkownikom układanie kostki Rubika w trójwymiarowej przestrzeni.

Wstęp

OpenGL, czyli Open Graphics Library, to uniwersalny standard, który jest powszechnie wykorzystywany do rozwijania interaktywnych aplikacji i gier komputerowych. Jest to cross-platformowa specyfikacja API (Application Programming Interface) służąca do tworzenia grafiki 2D i 3D.

Biblioteka OpenGL jest niezależna od systemu operacyjnego i pozwala na bezpośrednie zarządzanie sprzętowymi zasobami graficznymi. Wieloplatformowość i uniwersalność biblioteki zapewniają jej szerokie zastosowanie, od małych aplikacji mobilnych po duże produkcje gier i filmów.

OpenGL zapewnia efektywność i kontrolę na poziomie sprzętowym, umożliwiając programistom dostęp do funkcji graficznych karty, co przekłada się na lepsze wykorzystanie jej możliwości. Do najważniejszych funkcji OpenGL należą: renderowanie kształtów (prymitywów) 2D i 3D, manipulacja tekstur, efekty świetlne, cieniowanie i wiele innych.

Kostka Rubika to popularna układanka logiczna, której celem jest ułożenie wszystkich jej stron w jednolite jednokolorowe powierzchnie. Składa się z sześciu ścian, z których każda posiada 9 mniejszych kwadratowych kafelków. Kostka Rubika może mieć różne rozmiary, ale najbardziej powszechna wersja ma wymiary 3x3x3.

Każda ściana kostki Rubika może mieć inny kolor, a początkowo jest tak ustawiona, że każda strona ma swój unikalny kolor. Podstawowymi kolorami kostki Rubika są: biały, żółty, niebieski, zielony, czerwony i pomarańczowy. Celem układanki jest przestawienie kafelków na poszczególnych ścianach w taki sposób, aby na każdej z nich znalazł się tylko jeden kolor.

Kostka Rubika działa na zasadzie obracania poszczególnych warstw wokół osi, które przechodzą przez centrum kostki. Obracając odpowiednie warstwy, kafelki przemieszczają się wzdłuż tych osi, umożliwiając manipulację układanką.

Kostka Rubika jest nie tylko układanką, ale także przedstawia pewne wyzwania intelektualne. Rozwiązywanie kostki Rubika rozwija umiejętności koncentracji, myślenia przestrzennego, cierpliwości i zdolności analitycznych. Jest to również forma rozrywki i hobby dla wielu osób na całym świecie.

Opis problemu - cel projektu

Celem tego projektu jest stworzenie implementacji popularnej układanki logicznej - kostki Rubika, przy użyciu technologii OpenGL i języka programowania C++.

Podstawowe funkcjonalności projektu będą obejmować:

- Renderowanie kostki Rubika w trójwymiarze z wykorzystaniem biblioteki OpenGL.
- Obsługę interakcji użytkownika, takich jak obracanie poszczególnych warstw kostki za pomocą klawiatury, sterowanie kamerą.
- Zmiana koloru kostki dla osób z zaburzeniem wzroku.
- Wyświetlanie w konsoli informacji takich jak: ruchy na kostce, opcje aplikacji, najlepsze wyniki. Podczas układania także czas trwania układania i ilość ruchów.

Projekt ten ma na celu dostarczenie wirtualnej symulacji kostki Rubika, która pozwoli użytkownikowi na interaktywne rozwiązywanie tej popularnej łamigłówki.

Analiza problemu i teoretyczna propozycja jego rozwiązania

Problemem, który chcemy rozwiązać, jest stworzenie programu generującego wirtualną kostkę Rubika przy użyciu biblioteki OpenGL i języka programowania C++.

Aby osiągnąć ten cel, możemy zastosować różne algorytmy i techniki.

1. Renderowanie w trójwymiarze: Do wygenerowania wirtualnej kostki Rubika w OpenGL wykorzystamy techniki renderingu 3D. Kostka będzie składać się z sześciu płaskich ścian, z których każda będzie składana z 9 kwadratów. Wykorzystując techniki generowania geometrii, będziemy tworzyć odpowiednie wierzchołki i indeksy dla każdego kwadratu, a następnie używając tekstur do renderowania kolorów na poszczególnych ścianach kostki.

2. Obsługa interakcji użytkownika: Aby umożliwić interakcję użytkownika z kostką Rubika, będziemy musieli przechwycić dane wejściowe, takie jak ruchy myszką lub klawiaturą. Możemy użyć biblioteki OpenGL do rejestrowania i obsługi zdarzeń myszy i klawiatury. Ruch myszką pozwala na poruszanie wolnej kamery, a klawisze są używane do układanie kostki Rubika.

3. Symulacja wykonywania ruchów na kostce Rubika polega na zmianie położenia elementów kostki w odpowiedni sposób, aby odzwierciedlić ruchy wykonywane przez użytkownika. Aby to osiągnąć, możemy zastosować technikę podmiany tekstur na poszczególnych elementach kostki.

4. Pomieszanie kostki aby była możliwa do ułożenia opiera się na wykorzystaniu funkcji która wykonuje randomowe ruchy na kostce daną ilość razy którą określona poprzez stopień zaawansowania.

Implementacja

Repozytorium projektu:
https://github.com/DwiN3/Rubik_Cube_Project

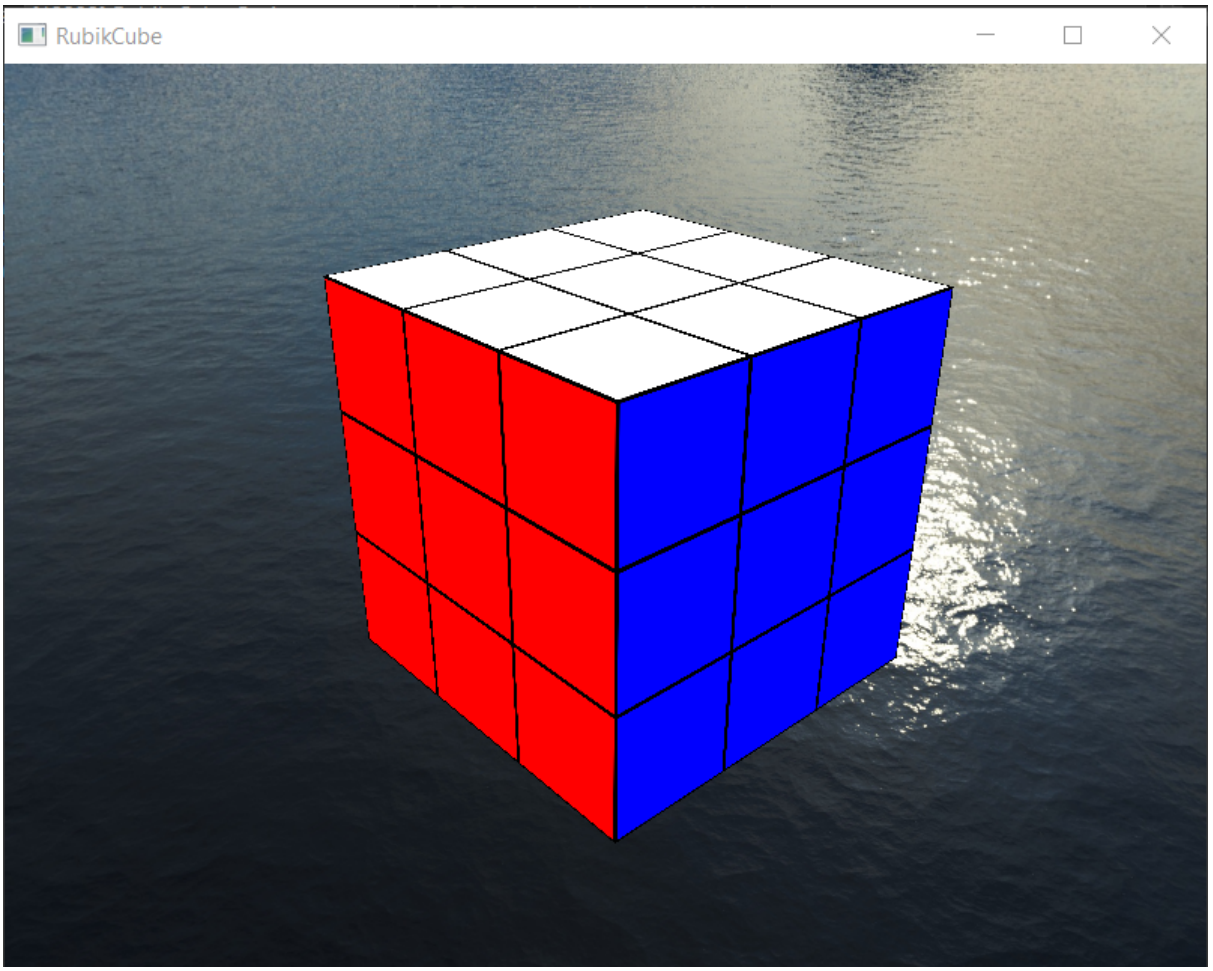
Wybór środowiska:
Pracowaliśmy na oprogramowaniu Visual Studio posługując się biblioteką programistyczną OpenGL do języka C++.

Kod który realizuje symulacje ruchu kostki od góry do dołu:

```
void turn_cube_up_to_down(int which, int each, bool game) {
    if (game) count_moves += 1;
    int a, b, c;
    a = sideCube[which][5];
    b = sideCube[which + 9][5];
    c = sideCube[which + 2 * 9][5];
    sideCube[which][5] = sideCube[which + 2 * 9][0];
    sideCube[which + 9][5] = sideCube[which + 2 * 9 + each][0];
    sideCube[which + 2 * 9][5] = sideCube[which + 2 * 9 + 2 * each][0];

    sideCube[which + 2 * 9][0] = sideCube[which + 2 * each + 2 * 9][4];
    sideCube[which + 2 * 9 + each][0] = sideCube[which + 2 * each + 9][4];
    sideCube[which + 2 * 9 + 2 * each][0] = sideCube[which + 2 * each][4];

    sideCube[which + 2 * each][4] = sideCube[which][1];
    sideCube[which + 2 * each + 9][4] = sideCube[which + each][1];
    sideCube[which + 2 * each + 2 * 9][4] = sideCube[which + 2 * each][1];

    sideCube[which][1] = c;
    sideCube[which + each][1] = b;
    sideCube[which + 2 * each][1] = a;

    for (int i = 2; i < 4; i++){
        a = sideCube[which][i];
        b = sideCube[which + 9][i];
        c = sideCube[which + 2 * 9][i];

        sideCube[which][i] = sideCube[which + 2 * 9][i];
        sideCube[which + 9][i] = sideCube[which + 2 * 9 + each][i];
        sideCube[which + 2 * 9][i] = sideCube[which + 2 * 9 + 2 * each][i];

        sideCube[which + 2 * 9][i] = sideCube[which + 2 * each + 2 * 9][i];
        sideCube[which + 2 * 9 + each][i] = sideCube[which + 2 * each + 9][i];
        sideCube[which + 2 * 9 + 2 * each][i] = sideCube[which + 2 * each][i];

        sideCube[which + 2 * each + 2 * 9][i] = sideCube[which + 2 * each][i];
        sideCube[which + 2 * each + 9][i] = sideCube[which + each][i];
        sideCube[which + 2 * each][i] = sideCube[which][i];

        sideCube[which][i] = c;
        sideCube[which + each][i] = b;
        sideCube[which + 2 * each][i] = a;
    }
    if (is_cube_solved() == true) cube_arranged(false);
}
```

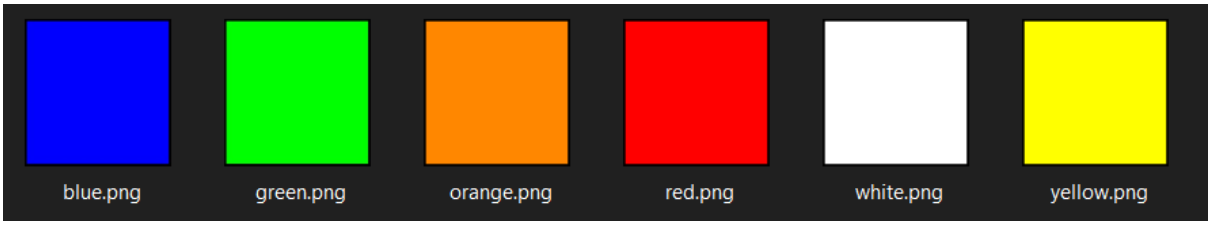
Powyższa funkcja symuluję obracanie kostki rubika z góry w dół.
Jako parametry, funkcja dostaje dwie główne zmienne **which** i **each**.
Obie te funkcję są związane z koordynatami bloków kostki, natomiast różnice są takie:
Which - zmienna ta wyznacza od którego punktu zaczynamy.
Each - zmienna ta wyznacza co ile bloków przeskakujemy.
Główna zasada działania tej funkcji to podmiana bloków w taki sposób aby bloki przemieściły się zgodnie z obracaniem kostki Rubika. Pierwsza część funkcji określa zamianę bloków w pionie a reszta część kodu wraz z pętlą zamienia bloki z lewej i prawej strony.

Kostka Rubika w naszym programie w trybie domyślnym:

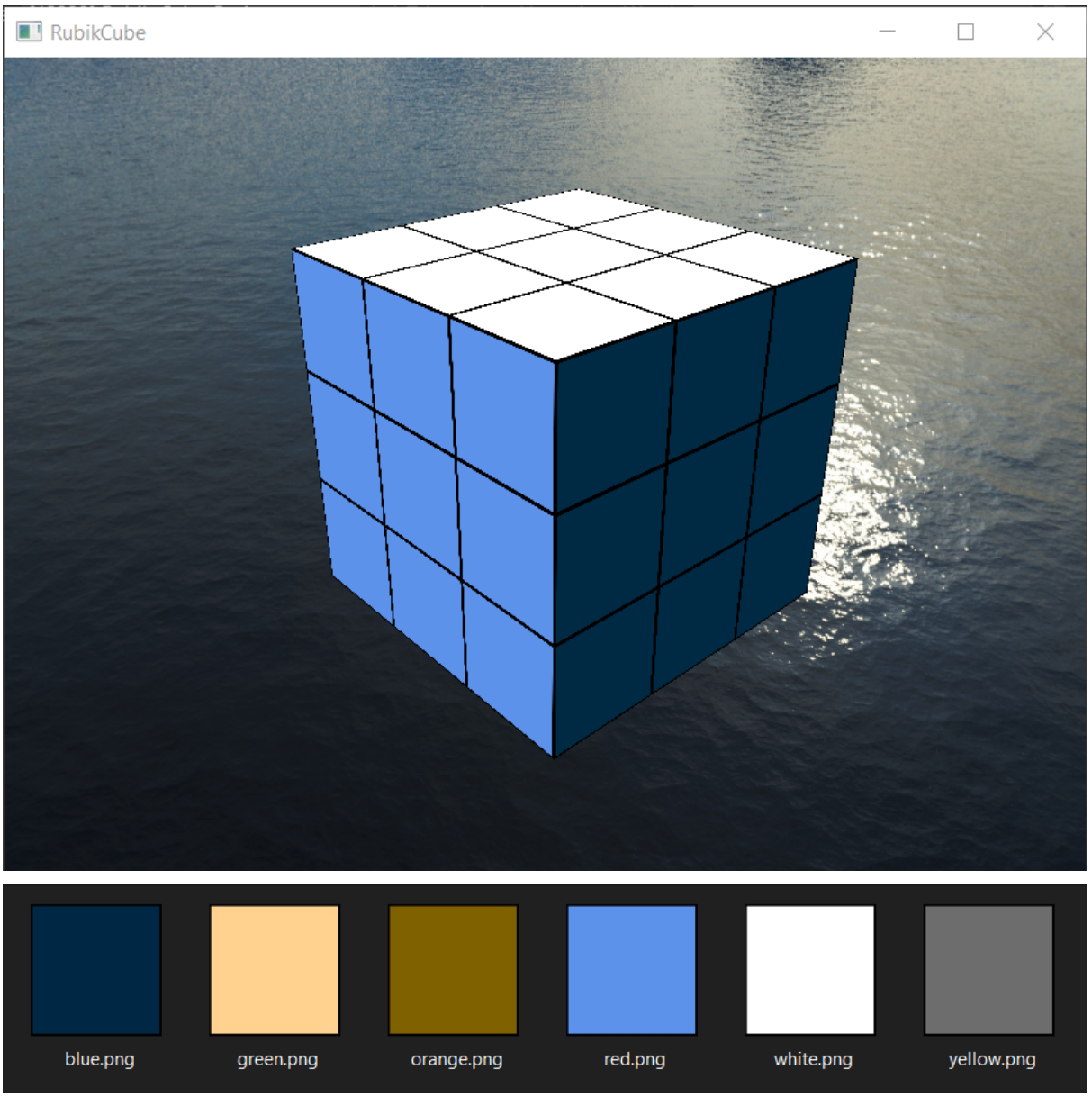


Korzystamy z trzech palet kolorów odpowiedniej dla osób bez zaburzeń wzroku. Pierwsza paleta przedstawia standardowe klasyczne kolory używane w kostce rubika. Druga paleta służy dla osób z problemem w rozróżnianiu barwy zielonej (deuteranopia). Dla osób z problemami w rozpoznawaniu barw żółtej i niebieskiej (tritanopia) istnieje osobna paleta kolorów. Kolory dla osób z zaburzeniem wzroku nie są w pełni odzwierciedleniem poprawnych kolorów ze względu na brak możliwości rozpoznania ich. Kolory zostały ustawione w taki sposób, by osoby z zaburzeniem nie miały problemu w ich odczytaniu.

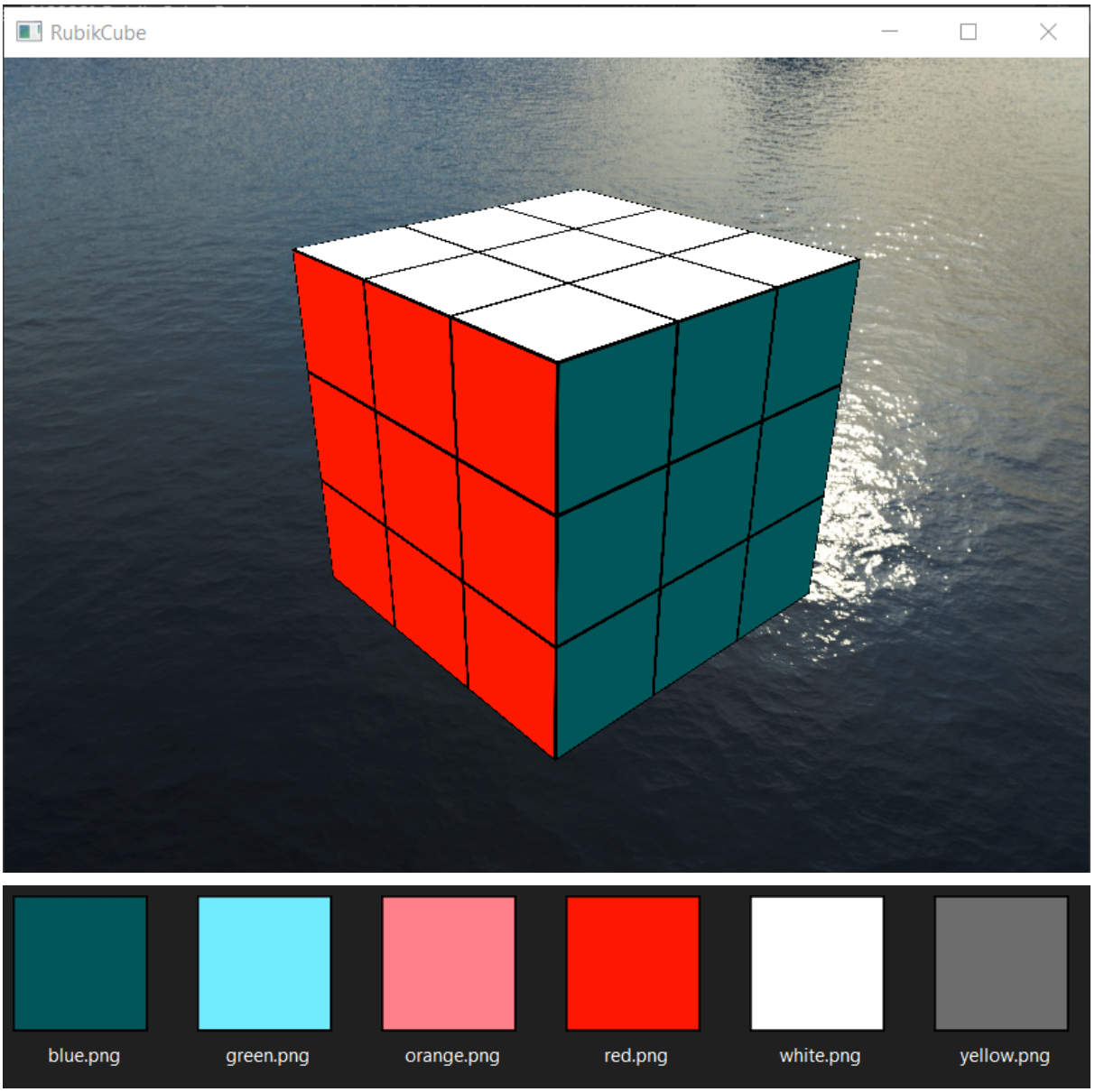
klasyczny:



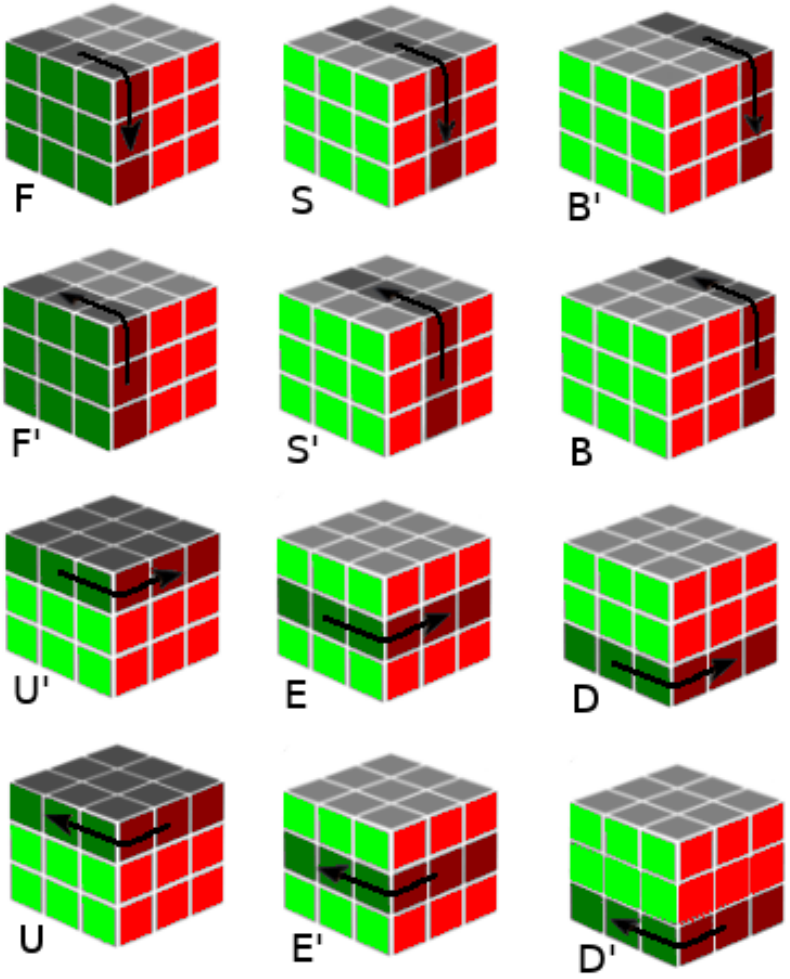
Kostka rubika w trybie dla daltonistów (deuteranopia):



Kostka rubika w trybie dla daltonistów (tritanopia):

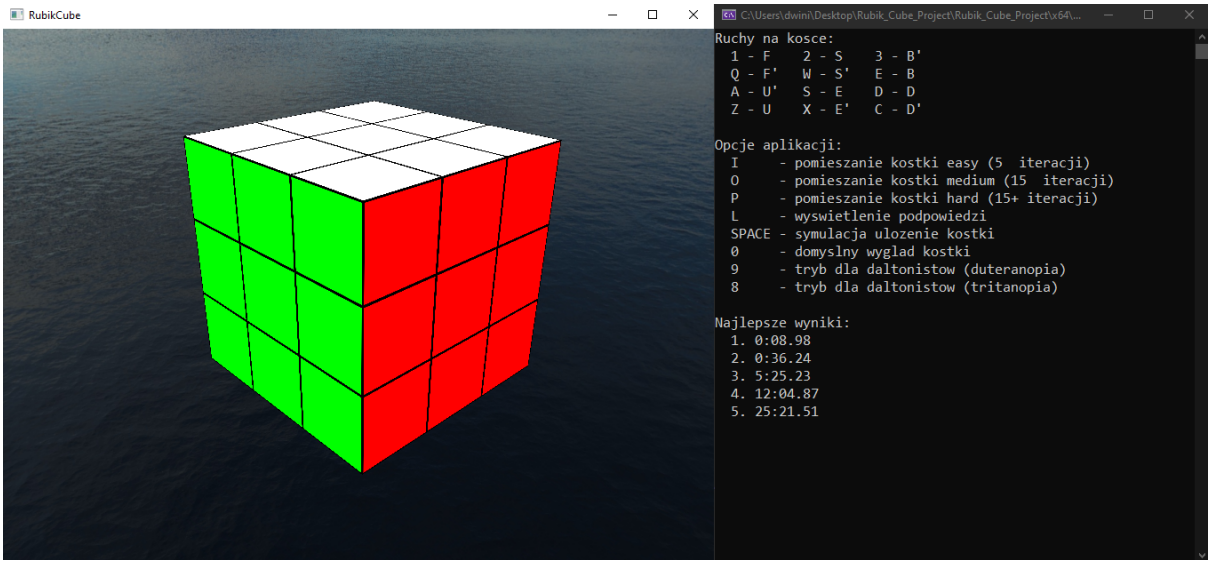


Sekwencja ruchów na naszej kostce:

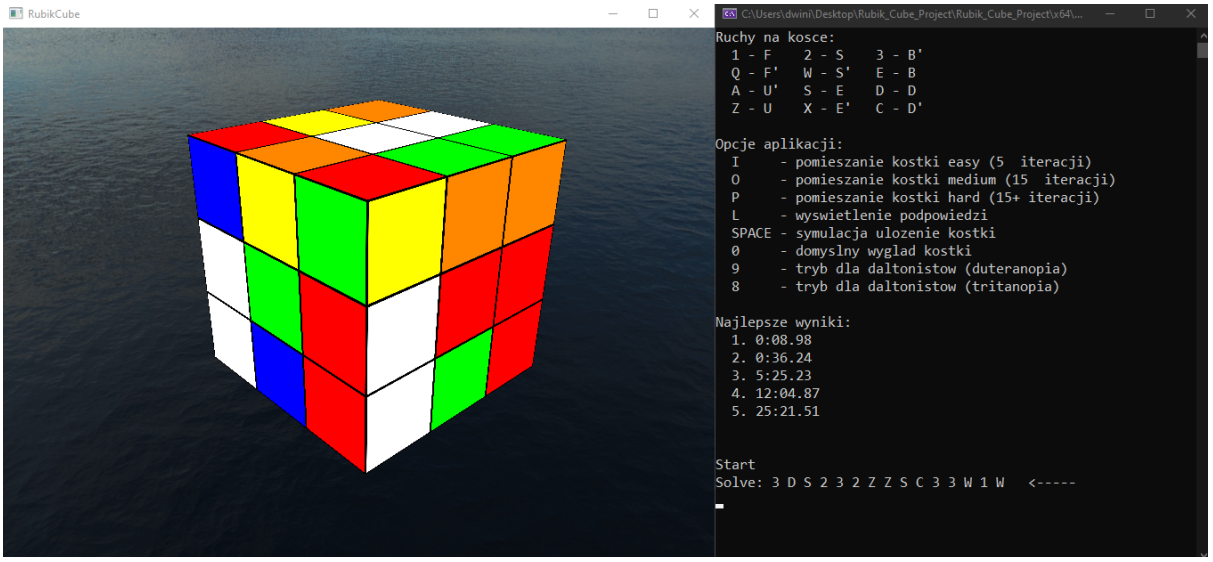


W terminalu są umieszczane najważniejsze elementy, które wytłumaczają użytkownikowi jak posługiwać się programem. Są wyświetlane:

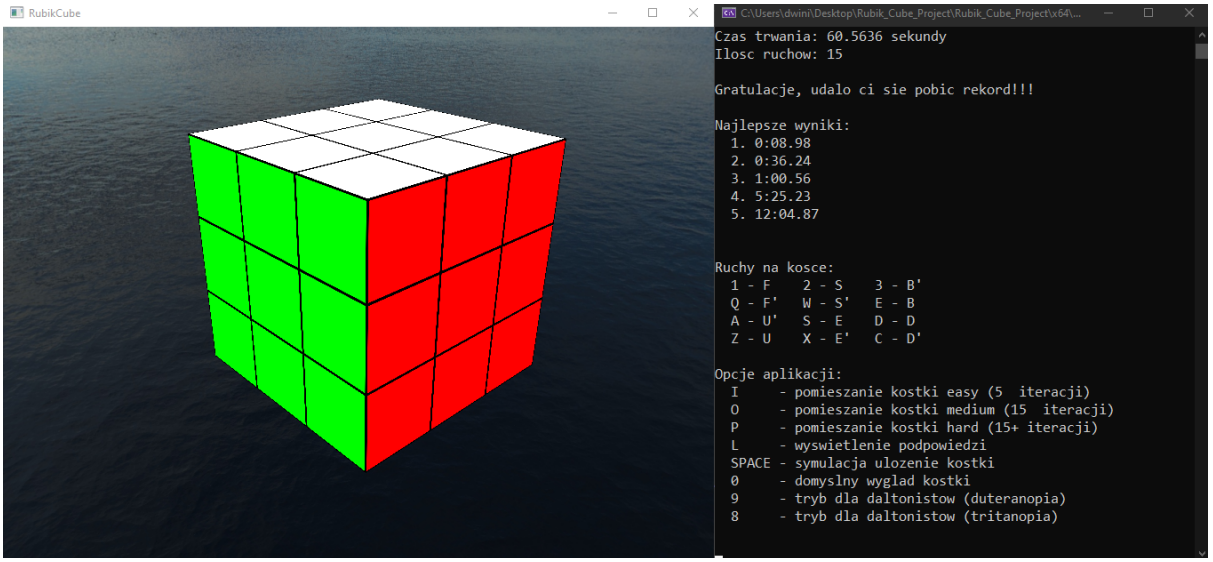
- możliwe dostępne ruchy
- opcje aplikacji
- najlepsze wyniki



Klikając ‘I’, ’O’ lub ’P’ kostka zostaje zmieszana w sposób możliwy do ułożenia. Po kliknięciu tego przycisku w tle jest uruchamiany stoper, który liczy czas od kliknięcia przycisku do momentu jej poprawnego ułożenia. Pod klawiszem ’L’ można wyświetlić podpowiedzi do ułożenia kostki. Program oferuje możliwość symulacji ułożenia kostki, który ułoży ją po naciśnięciu ’SPACE’ przywracając kostkę do domyślnego stanu.



Gdy układanie kostki Rubika zakończy się sukcesem na ekranie terminala zostaną wyświetlone informacje dotyczące naszego czasu układania. Program poinformuje użytkownika o zakończeniu układania kostki i poda czas ułożenia. Wyświetla również ruchy po których nastąpiło ułożenie. Jeżeli wynik będzie większy niż poprzedni to zostanie zapisany do tablicy wyników, która wyświetla pięciu najlepszych graczy.



Testowanie - poprawność, wydajność

W projekcie zostały użyte różne metody, które pozwoliły nam zabezpieczyć program przed niechcianym, lub nieprzewidzianym działaniem użytkownika. Przykłady zabezpieczeń w naszym projekcie:

- Przechowywanie stanu układania kostki - jak jest false to nie trwa, jak jest true, to znaczy że kostka jest dalej układana.

```
bool arranging = false;
```

- Sprawdzenie czy podpowiedź została wyświetlona. Podpowiedź wyświetli się ona tylko raz.

```
bool soSolved = true;
```

- Funkcja sprawdzająca, czy kostka jest ułożona

```
bool is_cube_solved()
{
    for (int i = 0; i < 27; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            if (sideCube[i][j] != j + 1) return false;
        }
    }
    return true;
}
```

- Przykładowe wykorzystanie funkcji is_cube_solved()

```
if ((glfwGetKey(window, GLFW_KEY_I) == GLFW_PRESS) && is_cube_solved() == true)
    mix_the_cube(1);
```

- Deklaracja funkcji po ułożeniu kostki

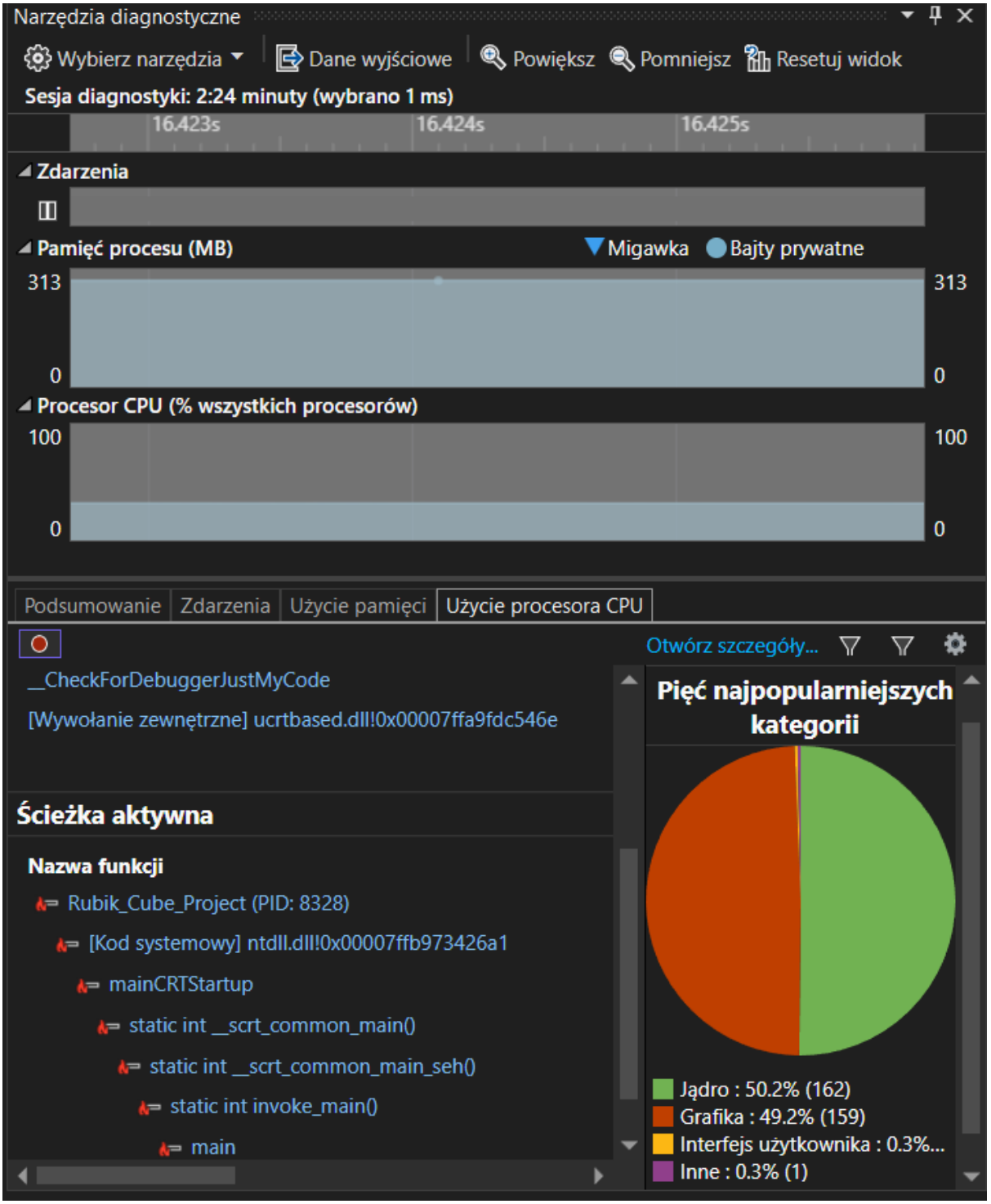
```
void cube_arranged(bool skip)
```

- Wykorzystywanie parametru skip w celu nie wypisywaniu rekordu czasu ułożenia kostki

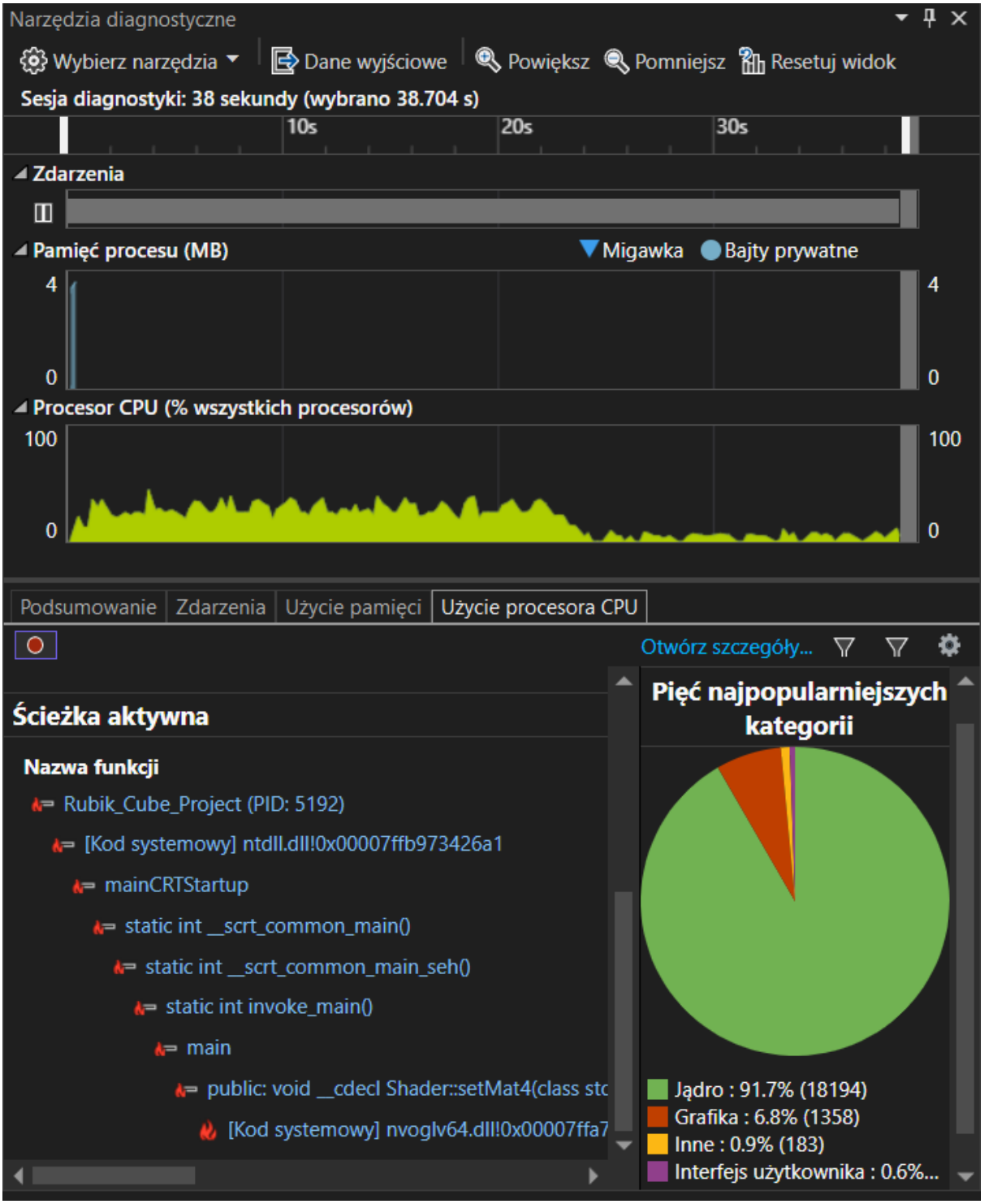
```
if (duration.count() < top_scores[TOP_SCORES_COUNT - 1] && skip == false) {
            cout << "Gratulacje, udalo ci sie pobic rekord!!!\n" << endl;

            top_scores[TOP_SCORES_COUNT - 1] = duration.count();
            sort(top_scores, top_scores + TOP_SCORES_COUNT);

            ofstream file("best_score.txt");
            if (file.good()) {
                for (int i = 0; i < TOP_SCORES_COUNT; i++) {
                    file << fixed << setprecision(2) << top_scores[i] << endl;
                }
                file.close();
            }
            else {
                cout << "Nie mozna zapisac do pliku" << endl;
            }
        }
```

Testowanie wydajności

Testowanie wydajności funkcji turn_cube_up_to_down(). Podczas sesji diagnostycznej zostały wyświetlone zużycie CPU oraz pamięci. W prawym dolnym rogu zostały wyświetlone procesy w procentach. Jak widać, jądro i grafika są zrównoważone podczas używania funkcji.



Testowanie wydajności funkcji main(). Po testach wydajności widać, że użycie CPU wynosi około 30%. Jądro wykonuje ponad 91% pracy, podczas gdy grafika to prawie 7%.



Wydajność aplikacji jest zależna od wykonywanych prac.

Instrukcja obsługi

Ruchy na kostce
Aby poruszać konkretnymi segmentami na kostce Rubika należy kliknąć odpowiedni przycisk na klawiaturze.
Klawisze odpowiedzialne za ruch kostką, to:

```
1  2  3
Q  W  E
A  S  D
Z  X  C
```

Klawisze te odpowiadają kolejno ruchom na realnej kostce, które nazywają się:

```
F  S  B'
F' S' B
U' E  D
U  E' D'
```

Ruch kamery
Aby umożliwić użytkownikowi poruszanie wokół kostki oraz w przejrzysty sposób zaprezentować działanie programu została dodana opcja poruszania się. Do poruszania się w przestrzeni należy użyć “strzałek” dostępnych na klawiaturze.

Strzałka w lewo:

```
←-
```

Powoduje przesunięcie się w lewą stronę ekranu.

Strzałka w prawo:

```
-→
```

Powoduje przesunięcie się w lewą stronę ekranu.

Strzałka w górę:

```
↑
```

Powoduje przybliżenie się do obiektu.

Strzałka w dół:

```
↓
```

Powoduje oddalenie się od obiektu
Do rozglądania się wokół należy użyć myszy komputerowej lub touchpada. Sposób w jaki zostało to zaimplementowane jest bardzo intuicyjny.

Opcje aplikacji
W opcjach aplikacji znajdują się specjalne interakcje, które użytkownik może zastosować.
Przyciski:

```
I O P
```

Powodują uruchomienie trybów, które zamieszają kostkę rubika według kombinacji, które są dostępne w aplikacji, dzięki czemu kostka jest zawsze możliwa do ułożenia np. wykonując ruchy które spowodowałe zmieszanie kostki.

Przycisk:

```
I
```

Powoduje pomieszanie kostki na 5 iteracji. Jest to najprostszy tryb w programie.

Przycisk:

```
O
```

Powoduje pomieszanie kostki na 15 iteracji. Jest to tryb średnio zaawansowany.

Przycisk:

```
P
```

Powoduje pomieszanie kostki na wartość losową w zakresie od 15 do 50 iteracji. Tryb przeznaczony dla zaawansowanych użytkowników.

Dodatkowa opcja w programie jest to wyświetlenie podpowiedzi dla każdego z trybu dla pomieszania kostki.
Przycisk służący do tego celu, to:

```
L
```

Tryb dla osób z zaburzeniem wzroku
W programie zostały zaimplementowane trzy palety kolorów.
Pierwsza paleta jest uruchomiona domyślnie, jednak można ją też uruchomić ponownie klikając przycisk:

```
0
```

Druga paleta kolorów jest skierowana dla osób z wadą wzroku o nazwie deuteranopia.
Zmiana jest dostępna pod przyciskiem:

```
9
```

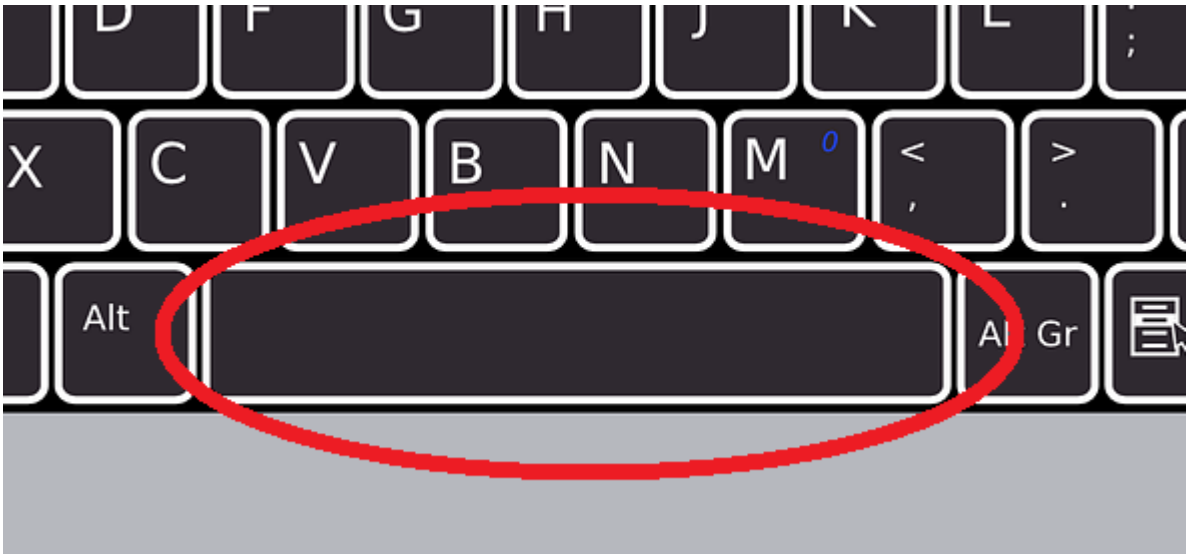
Trzecia paleta kolorów jest skierowana dla osób z wadą wzroku o nazwie tritanopia. Zmiana jest dostępna pod przyciskiem:

```
8
```

Powrót do domyślnego ułożenia

Przycisk spacji symuluje ułożenie kostki rubika. Sprawia, że kostka rubika powraca do swojego pierwotnego stanu przed zmianami.

Przycisk znajduje się na klawiaturze pod przyciskiem spacebar:



fot: https://pixabay.com/pl/vectors/klawiatura-czarny-kompaktowy-34176/

Proponowane dalsze możliwości rozbudowy

- animacje poruszania się elementów
- wprowadzenie algorytmów do układania kostki rubika
- bardziej zaawansowany interfejs użytkownika
- pokazywanie rankingu najmniejszej ilości ruchów przy ułożeniu kostki

Instrukcja kompilacji

1. Skopiować folder LIB w ścieżce: “C:\LIB”
2. Należy przyłączyć wersje na x64 oraz ustawić Debug
3. Włączyć właściwości Rubik_Cube_Project
4. W zakładce Katalogi VC++ dodać w zakładkach:
   Katalogi plików nagłówkowych i dodać:
   - C:\LIB\glad\include
   - C:\LIB\glfw\include

   Katalogi bibliotek:
   - C:\LIB\glfw\lib-vc2022
5. W zakładce Dane Wejściowe dodać:
   Dodatkowe Zależności:
   - glfw3.lib

Zdjęcia poprawnego dodania ścieżek: