REV	DATA	ZMIANY
0.1	4.01.2024	*Wojciech Kamiński (wojkam@student.agh.edu.pl)*

GRA KARCIANA ”OCZKO”

Autor: Wojciech Kamiński

Akademia Górniczo-Hutnicza
Elektronika i Telekomunikacja

Kraków (C) 2024

Spis treści

Lista oznaczeń

SFML	Simple and Fast Multimedia Library
MT	Mersenne Twister

1. Wstęp

Niniejszy raport techniczny dotyczy projektu gry karcianej "Oczko", który został zrealizowany w dwóch wariantach: konsolowym oraz z interfejsem graficznym przy użyciu biblioteki SFML. Celem projektu było stworzenie aplikacji umożliwiającej rozgrywkę w popularną grę karcianą "Oczko" pomiędzy graczem a komputerowym przeciwnikiem.

Gra "Oczko" to klasyczna gra karciana, w której celem gracza jest zdobycie jak najbliżej 21 punktów, unikając jednocześnie przekroczenia tej wartości.

W projekcie skoncentrowano się na stworzeniu funkcjonalnej implementacji reguł gry oraz dostarczeniu dwóch wersji - jednej w formie konsolowej, a drugiej z interfejsem graficznym.

Wersja konsolowa gry została napisana w języku C++, wykorzystując podstawowe struktury danych, pętle i instrukcje warunkowe. Z kolei wersja z interfejsem graficznym wykorzystuje bibliotekę SFML do stworzenia przyjaznego dla użytkownika środowiska graficznego, umożliwiającego interaktywną rozgrywkę.

Poniżej przedstawiono fragmenty kodu z obu wersji gry, prezentujące implementację kluczowych elementów. W kolejnych sekcjach raportu zostaną dokładniej omówione poszczególne aspekty techniczne projektu.

Link do wersji konsolowej: https://github.com/wojkam23/JPO_projekt_wersja_konsolowa

Link do wersji z SFML: https://github.com/wojkam23/JPO_projekt_SFML_wersja

Link do GTest: https://github.com/wojkam23/JPO_projekt_GTest

1. Wymagania systemowe

Podstawowe założenia projektu:

1. **Język Programowania:**
    - Projekt został napisany w języku C++ z wykorzystaniem standardu C++17.
2. **Platforma:**
    - Wersja konsolowa działa na systemach takich jak Windows, Linux, macOS.
    - Wersja z interfejsem graficznym wymaga obsługi SFML na tych samych systemach.
3. **Biblioteki:**
    - Wersja konsolowa nie wymaga dodatkowych bibliotek.
    - Wersja z interfejsem graficznym wymaga zainstalowanego i skonfigurowanego SFML.
4. **Obsługa Wejścia/Wyjścia:**
    - Wersja konsolowa korzysta z klawiatury i wyświetla informacje w konsoli.
    - Wersja z interfejsem graficznym pozwala na interakcję za pomocą myszy i klawiatury, w tym:
        - Użycie klawiszów na ekranie SFML (D – dobierz, S – stój)
        - Po zakończeniu rozgrywki, decyzję o rozpoczęciu kolejnej partii podejmuje się w konsoli poprzez wpisanie T (tak) lub N (nie)

2. Funkcjonalność

1. **Rozgrywka:**
    - **Wersja konsolowa:**
        - Gra rozpoczyna się automatycznie po uruchomieniu programu.
        - Gracz podejmuje decyzje o dobieraniu kart poprzez wpisanie 'D' (dobierz) lub 'S' (stój).
        - Komunikaty w konsoli informują gracza o aktualnym stanie gry, wartości jego kart, oraz ruchach przeciwnika.
        - Po zakończeniu partii, gracz decyduje o kontynuacji, wpisując 'T' (tak) lub 'N' (nie) w konsoli.
    - **Wersja z interfejsem graficznym:**
        - Gra rozpoczyna się automatycznie po uruchomieniu programu.
        - Gracz podejmuje decyzje o dobieraniu kart poprzez kliknięcia na przyciski 'D' (dobierz) lub 'S' (stój).
        - Informacje o stanie gry są prezentowane w oknie SFML oraz w konsoli.
        - Po zakończeniu partii, gracz decyduje o kontynuacji, wpisując 'T' (tak) lub 'N' (nie) w konsoli.
2. **Zasady Gry:**
    - Oczko to prosta gra karciana, gdzie celem jest uzyskanie jak najbliżej, ale nie przekraczając 21 punktów.
    - As ma wartość 1 lub 11, karty Walet, Dama, Król mają wartość 10, pozostałe karty mają wartość zgodną z ich numerem.
    - Gracz dobiera karty decydując, czy chce uzyskać więcej punktów, czy też zatrzymać się na danej wartości.
    - Komputer dobiera karty według ustalonych reguł (do wartości 17).
3. **Warunki Wygranej i Przegranej:**
    - Gracz wygrywa, jeśli:
        - Jego wynik jest bliższy 21 niż wynik komputera.
        - Komputer przekroczył 21 punktów.
    - Gracz przegrywa, jeśli:
        - Przekroczył 21 punktów.
        - Wynik komputera jest bliższy 21 niż jego wynik.

3. Analiza problemu

1. Przyporządkowania kolorów do kart

Opis Problemu:

W talii każda karta ma przypisany swój kolor (pik, kier, karo, trefl).

Rozwiązanie problemu:

Aby skutecznie rozwiązać ten problem, zaimplementowano klasę Karty. W konstruktorze klasy zastosowano zagnieżdżone pętle for, aby dynamicznie przypisać odpowiedni kolor każdej karcie w talii. Wartości karty oraz kolor są przechowywane w wektorach, a funkcja make_pair została użyta do stworzenia pary wartość-kolor dla każdej karty.

```cpp
// Klasa reprezentująca talię kart
class Karty {
public:
    // Wektor przechowujący wartości kart
    vector<string> karta = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Walet", "Dama", "Krol", "As"};
    // Wektor przechowujący kolory kart
    vector<string> kolor = {"pik", "kier", "karo", "trefl"};
    // Wektor przechowujący pary wartość-kolor dla każdej karty w talii
    vector<pair<string, string>> talia;

    // Konstruktor inicjalizujący talie kart
    Karty() {
        for (int i = 0; i < kolor.size(); ++i) {
            for (int j = 0; j < karta.size(); ++j) {
                // Tworzenie pary wartość-kolor i dodawanie jej do talii
                talia.push_back(make_pair( &: karta[j],  &: kolor[i]));
            }
        }
    }
};
```

2. Tasowanie kart

Opis problemu:

Tasowanie kart, przy użyciu funkcji rand(), mogłoby doprowadzić do generowania sekwencji, które nie są wystarczając losowe
Rozwiązanie problemu:

Funkcja tasowania używa zaawansowanego generatora liczb pseudolosowych MT. Zastosowano funkcję shuffle do losowego przemieszczania kart w talii.

```
void tasowanie() {
    random_device rd;     // Inicjalizacja generatora liczb losowych
    mt19937 g( sd: rd()); // Utworzenie obiektu generatora MT
    shuffle( first: taliaKart.begin(),  last: taliaKart.end(),  &: g); // Tasowanie talii kart
}
```

3. Przypisanie wartości punktowych kartom

Opis problemu:

W grach karcianych, takich jak Oczko, istnieje potrzeba przypisania wartości punktowej dla poszczególnych kart w talii. Problemem jest zróżnicowanie wartości kart, zwłaszcza w przypadku kart specjalnych, takich jak Walet, Dama, Król i As, które posiadają niestandardowe wartości liczbowe. Dodatkowo, karta As może przyjąć dwie różne wartości:
1 lub 11, w zależności od sytuacji w grze.

Rozwiązanie problemu:

W celu rozwiązania tego problemu, zaimplementowano funkcję wartoscReki, która dokładnie określa wartość punktową dla danej ręki w grze. Wartości liczbowe kart numerycznych są przypisywane za pomocą funkcji ‘stoi’ do zamiany wartości typu string na int. Dla kart specjalnych (Walet, Dama, Król), przypisuje się stałą wartość 10. Natomiast dla kart As, zastosowano elastyczne podejście, gdzie ich wartość to 11, chyba że przekroczyłaby ona 21 punktów, wtedy As przyjmuje wartość 1, co pozwala uniknąć przekroczenia limitu punktów.

```
int wartoscReki(const vector<pair<string, string>>& reka) {
    int wartosc = 0;
    int liczbaAsow = 0;

    for (int i = 0; i < reka.size(); ++i) {
        if (reka[i].first == "As") {
            wartosc += 11;
            liczbaAsow++;
        } else if (reka[i].first == "Walet" || reka[i].first == "Dama" || reka[i].first == "Krol") {
            wartosc += 10;
        } else {
            wartosc += stoi( str: reka[i].first); //funkcja 'stoi' konwertuje wartości string na int
        }
    }
    //dostosowanie wartości Asa w zależności od sytuacji
    while (wartosc > 21 && liczbaAsow > 0) {
        wartosc -= 10;
        liczbaAsow--;
    }

    return wartosc;
}
```

4. Dobieranie kart dla gracza i komputera

Opis problemu:

Problem polega na skonstruowaniu efektywnego mechanizmu zarządzania kartami, uwzględniającego zasady gry, takie jak limit punktów (21) oraz decyzje podjęte przez gracza.

Rozwiązanie problemu:

W celu rozwiązania problemu, wprowadzono kilka funkcji w klasie Oczko, które zarządzają procesem dobierania kart. Szczegółowe rozwiązanie obejmuje:

- funkcja ‘dobierzKarte’

```
void dobierzKarte(vector<pair<string, string>>& reka) {
    reka.push_back(taliaKart.back());
    taliaKart.pop_back();
}
```

Odpowiada za dodanie nowej karty do ręki gracza lub komputera. Wykorzystuje ona talię kart, pobierając kartę z jej końca i dodając ją do ręki.

- funkcja ‘rozdaniePoczatkowe’

```
void rozdaniePoczatkowe() {
    dobierzKarte( &: rekaGracza);
    dobierzKarte( &: rekaKomputera);
    dobierzKarte( &: rekaGracza);
```

Realizuje początkowe rozdanie, dobierając dwie karty dla gracza i jedną dla komputera.

- funkcja ‘graj’

```
void graj() {
    char wybor;
    do {
        cout << "Czy chcesz dobrac karte? (D - dobierz, S - stoj): ";
        cin >> wybor;

        if (wybor == 'D' || wybor == 'd') {
            dobierzKarte(rekaGracza);
            // Dodatkowa logika związana z dobieraniem dla gracza...
        } else if (wybor == 'S' || wybor == 's') {
            break;
        } else {
            cout << "Niepoprawny wybor. Wybierz ponownie." << endl;
        }
    } while (true);

    // Dodatkowa logika związana z dobieraniem dla komputera...
}
```

Odpowiada za logikę gry, w tym interakcję z graczem w kontekście dobierania kart. Gracz ma możliwość decydowania, czy chce kontynuować dobieranie.

PROBLEMY I ROZWIĄZANIA DLA PROGRAMU Z SFML

1 Wczytanie grafiki kart

Rozwiązanie problemu:

Dodano funkcję ‘nazwaKartyNaPlik’ do obsługi wczytywania grafiki.

```
string nazwaKartyNaPlik(const pair<string, string>& karta) {
    return "obrazy_kart/" + karta.first + "_" + karta.second + ".png";
}
```

2 Obsługa klawiszy

Rozwiązanie problemu:

Wykorzystano pętlę zdarzeń SFML do śledzenia wciśnięć klawiszy (sf::Event::KeyPressed) i wywołania odpowiednich funkcji w przypadku naciśnięcia klawisza.

```
while (window.isOpen() && !graczPrzekroczyl21) {
    sf::Event event;
    while (window.pollEvent(event)) {
        if (event.type == sf::Event::Closed) {
            window.close();
        }
        else if (event.type == sf::Event::KeyPressed) {
            if (event.key.code == sf::Keyboard::D) {
                oczko.dobierzKarte(oczko.getRekaGracza());
                cout << "Twoje karty: ";
                oczko.pokazReke(oczko.getRekaGracza());
                cout << "Wartość Twoich kart: " << oczko.wartoscReki(oczko.getRekaGracza()) << endl;

                window.clear();
                rysujAktualneKarty(window, oczko);
                window.draw(instructions);  // Dodanie napisu na górze okna
                window.display();

                if (oczko.wartoscReki(oczko.getRekaGracza()) > 21) {
                    cout << "Przekroczyłeś 21! Przegrałeś." << endl;
                    graczPrzekroczyl21 = true;
                }
            }
            else if (event.key.code == sf::Keyboard::S) {
                graczPrzekroczyl21 = true;
            }
        }
    }
```

3 Rysowanie kart na ekranie

Rozwiązanie problemu:

Dodano funkcje ‘rysujKarte’ (do obsługi wczytywania obrazów i rysowania kart na ekranie) oraz ‘rysujAktualneKarty’(do rysowania kart gracza i komputera na ekranie po każdym ruchu).

```
void rysujAktualneKarty(sf::RenderWindow& window, Oczko& oczko) {
    // Rysowanie kart gracza
    float x = 50.0f;
    float y = 300.0f;
    vector<pair<string, string>>& rekaGracza = oczko.getRekaGracza();
    for (int i = 0; i < rekaGracza.size();i++) {
        rysujKarte(window, rekaGracza[i], x, y);
        x += 182.0f;
    }

    // Rysowanie kart komputera
    x = 50.0f;
    y = 100.0f;
    vector<pair<string, string>>& rekaKomputera = oczko.getRekaKomputera();
    for (int i = 0; i < rekaKomputera.size(); i++) {
        rysujKarte(window, rekaKomputera[i], x, y);
        x += 182.0f;
    }
}
```

5. Projekt techniczny

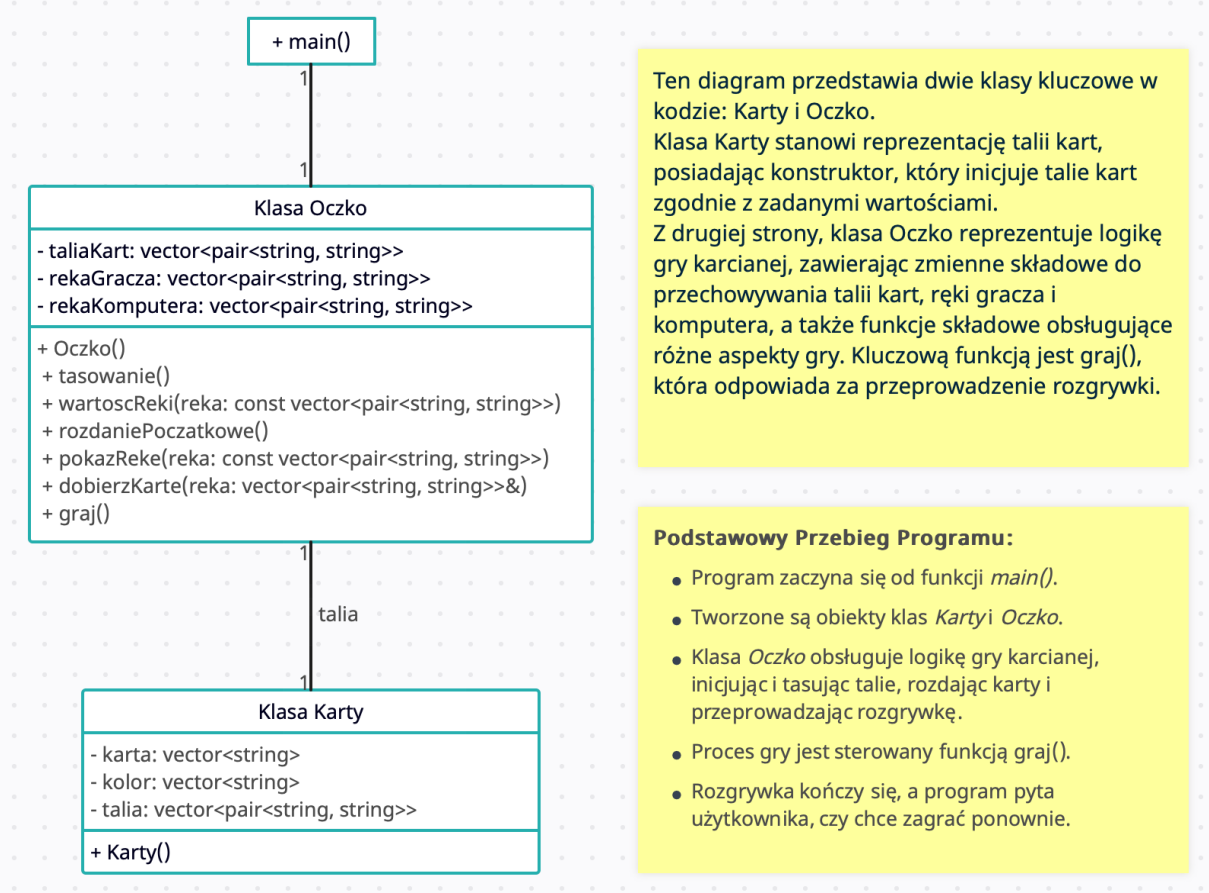


diagram stworzony na stronie: app.creately.com

5.1 Opis funkcji oraz klas

Opisy funkcji:

```
//////////////////////////////////////////////////////////////
// Ta funkcja oblicza wartość ręki w grze
//////////////////////////////////////////////////////////////
//
// INPUT:
//      reka - wektor par reprezentujący rękę gracza lub komputera
// OUTPUT:
//      Całkowita wartość ręki
// REMARKS:
//      Obsługuje specjalny przypadek Asa, który może przyjąć wartość 1 lub 11
//
//
int wartoscReki(const vector<pair<string, string>>& reka)
```

```
//////////////////////////////////////////////////////////////
// Ta funkcja rozdaje początkowe karty graczowi i komputerowi
//////////////////////////////////////////////////////////////
//
// INPUT:
//      Brak
// OUTPUT:
//      Brak
// REMARKS:
//      Karty są dodawane do do ręki gracza i komputera. Wywołuje funkcję dobierzKarte()
//
//
void rozdaniePoczatkowe()
```

```
//////////////////////////////////////////////////////////////
// Ta funkcja wyświetla karty w ręce
//////////////////////////////////////////////////////////////
//
// INPUT:
//      reka - wektor par reprezentujący karty w ręce
// OUTPUT:
//      Brak
// REMARKS:
//      Brak
//
//
void pokazReke(const vector<pair<string, string>>& reka)
```

```
//////////////////////////////////////////////////////////////
// Ta funkcja dobiera kartę do ręki gracza lub komputera
//////////////////////////////////////////////////////////////
//
// INPUT:
//      reka - wektor par reprezentujący karty w ręce
// OUTPUT:
//      Brak
// REMARKS:
//      Wylosowana karta jest usuwana z talii
//
//
void dobierzKarte(const vector<pair<string, string>>& reka)
```

```
////////////////////////////////////////////////////////////////////////////////////////////
// Ta funkcja zarządza rozgrywką, pozwalając graczowi dobierać karty aż zdecyduje się spasować
// lub przekroczy 21. Następnie komputer gra, a zwycięzca jest określany.
////////////////////////////////////////////////////////////////////////////////////////////
//
// INPUT:
//      Brak
// OUTPUT:
//      Brak
// REMARKS:
//
//
void graj()
```

```
////////////////////////////////////////////////////////////
// Ta funkcja zwraca referencję do ręki gracza
////////////////////////////////////////////////////////////
//
// INPUT:
//      Brak
// OUTPUT:
//      Referencja do wektora reprezentującego rękę gracza
//
void getRekaGracza()
```

```
////////////////////////////////////////////////////////////
// Ta funkcja zwraca referencję do ręki komputera
////////////////////////////////////////////////////////////
//
// INPUT:
//      Brak
// OUTPUT:
//      Referencja do wektora reprezentującego rękę komputera
//
void getRekaKomputera()
```

```
////////////////////////////////////////////////////////////////////////////////
// Ta funkcja tworzy nazwę pliku dla grafiki karty na podstawie jej wartości
////////////////////////////////////////////////////////////////////////////////
//
// INPUT:
//      karta - para reprezentująca wartość i kolor karty
// OUTPUT:
//      Nazwa pliku dla grafiki karty
//
void nazwaKartyNaPlik(const pair<string, string>& karta)
```

```
////////////////////////////////////////////////////////////////////////
// Ta funkcja rysuje kartę na oknie SFML na określonej pozycji
////////////////////////////////////////////////////////////////////////
//
// INPUT:
//      window - okno SFML
//      karta - para reprezentująca wartość i kolor karty
//      x - współrzędna x dla pozycji karty
//      y - współrzędna y dla pozycji karty
// OUTPUT:
//      Brak
// REMARKS:
//      Używa SFML do wczytania i narysowania grafiki karty
//
//
void rysujKarte(sf::RenderWindow& window, const pair<string, string>& karta, float x, float y)
```

```
////////////////////////////////////////////////////////////////////////
// Ta funkcja rysuje bieżące karty gracza i komputera na oknie SFML
////////////////////////////////////////////////////////////////////////
//
// INPUT:
//      window - okno SFML
//      oczko - instancja klasy Oczko reprezentująca stan gry
// OUTPUT:
//      Brak
// REMARKS:
//      Wywołuje rysujKarte() dla każdej karty w ręce gracza i komputera
//
//
void rysujAktualneKarty(sf::RenderWindow& window, Oczko& oczko)
```

Opisy klas:

a) klasa Karty:

```
////////////////////////////////////////////////////////////////////////////////
// Klasa reprezentująca talie kart do gry w oczko. Zawiera dwa wektory reprezentujące
// wartości kart oraz ich kolory. Inicjalizuje talię tworząc pary wartość-kolor.
////////////////////////////////////////////////////////////////////////////////
class Karty {
public:
    // Konstruktor inicjalizujący talie kart.
    Karty() {
        // Inicjalizacja talii kart.
        // ...
    }
    // ... (inne metody i pola klasy)
};
```

b) klasa Oczko:

```
////////////////////////////////////////////////////////////////////////////////
// Klasa reprezentująca logikę gry w oczko. Zawiera wektory dla talii kart, reki gracza
// i reki komputera. Posiada funkcje do tasowania talii, obliczania wartości ręki,
//rozdawania kart, pokazywania rąk i prowadzenia gry.
////////////////////////////////////////////////////////////////////////////////

//  Ta klasa reprezentuje grę "Oczko".
class Oczko {
private:
    // Wektory kart

public:
    // Konstruktor inicjalizujący nową grę oczko.
    Oczko() {
        // Inicjalizacja nowej gry oczko.
        // ...
    }

    // Tasowanie talii kart.
    void tasowanie() {
        // Algorytm tasowania talii kart.
        // ...
    }

    // Obliczenie wartość ręki gracza lub komputera.
    int wartoscReki(const vector<pair<string, string>>& reka) {
        // Algorytm obliczania wartości ręki.
        // ...
    }

    // Rozdanie początkowe kart gracza i komputera.
    void rozdaniePoczatkowe() {
        // Logika rozdania początkowego.
        // ...
    }

    // Pokazanie karty w ręce.
    void pokazReke(const vector<pair<string, string>>& reka) {
        // Logika pokazywania kart w ręce.
        // ...
    }

    // Dobieranie karty do ręki gracza lub komputera.
    void dobierzKarte(vector<pair<string, string>>& reka) {
        // Logika dobierania karty do ręki.
        // ...
    }

    // Logika przeprowadzania gry oczko.
    void graj() {
        // Główna logika gry.
        // ...
    }
};
```

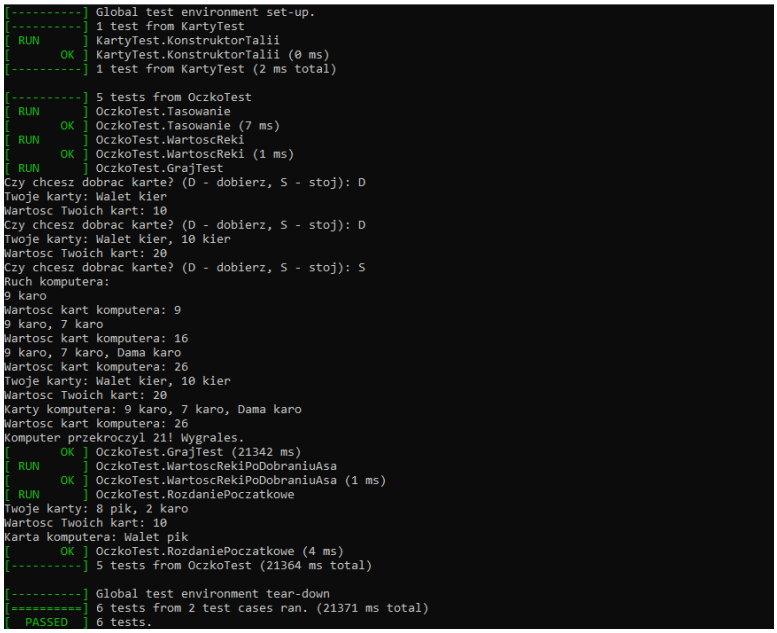
6. Opis realizacji (*implementation report*)

W trakcie implementacji projektu korzystano głównie z platformy Visual Studio 2022, która pełniła rolę środowiska programistycznego. Projekt został opracowany na maszynie z systemem operacyjnym Windows. Podczas procesu kompilacji i budowy projektu wykorzystano wbudowane narzędzia i kompilatory dostępne w środowisku Visual Studio 2022.

Również skonfigurowano bibliotekę SFML, która została użyta do implementacji elementów graficznych w projekcie. Użyte grafiki talii w programie zostały pobrane ze strony pixabay.com. Użyto wcześniej pobranej czcionki ‘Arial’ do tekstów w wersji programu z interfejsem graficznym.

7. Google Test

Poniższy zrzut ekranu przedstawia wyniki przeprowadzonych testów, w tym szczegółowe informacje dotyczące każdego testu, czasu ich wykonania oraz ogólnej statystyki.
Wszystkie testy zakończyły się pomyślnie, co wskazuje na poprawność działania programu.

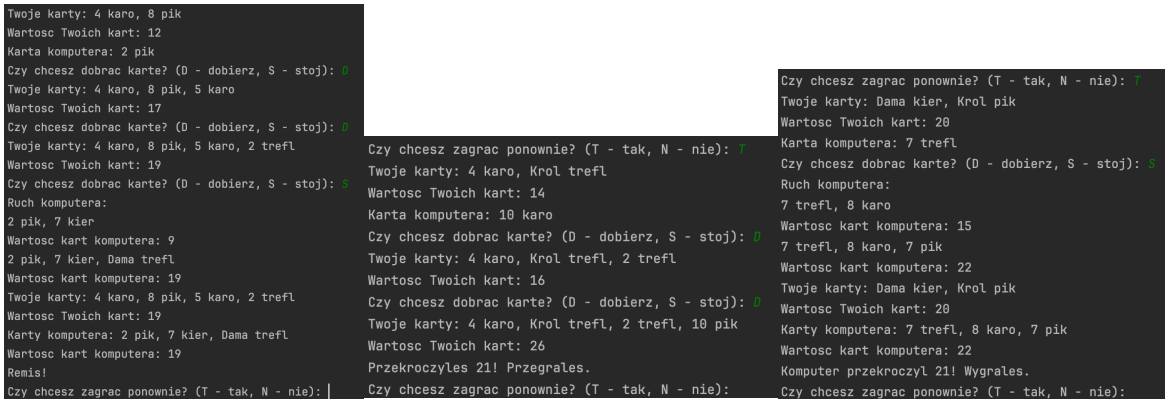


Przeprowadzone testy:

1. KartyTest.KonstruktorTalii
   - rezultat: OK
   - opis: Test sprawdza, czy konstruktor klasy Karty poprawnie inicjalizuje talię, oczekując 52 kart
2. OczkoTest.Tasowanie
   - rezultat: OK
   - opis: Testuje, czy funkcja tasowania klasy Oczko skutkuje różnym porządkiem kart przed i po tasowaniu
3. OczkoTest.WartoscReki
   - rezultat: OK
   - opis: Testuje funkcję obliczającą wartość ręki, porównując ją z oczekiwanymi wynikami dla różnych kombinacji kart
4. OczkoTest.GrajTest
   - rezultat: OK
   - opis: Przeprowadza symulację gry, sprawdzając poprawność interakcji i zakończenia gry
5. OczkoTest.WartoscRekiPoDobraniuAsa
   - rezultat: OK
   - opis: Sprawdza, czy wartość ręki uwzględnia poprawne zliczanie wartości karty As
6. OczkoTest.RozdaniePoczatkowe
   - rezultat: OK
   - opis: Testuje rozdanie początkowe gry, upewniając się, że każdy gracz otrzymuje dwie karty, a talia pomniejsza się o odpowiednią liczbę kart.

8. Podręcznik użytkownika (*user's manual*)

Zrzuty ekranu prezentujące rozgrywkę gracza z komputerem w wersji konsolowej programu:



Zrzut ekranu prezentujące rozgrywkę gracza z komputerem w wersji z interfejsem graficznym SFML:

W wersji programu z interfejsem graficznym gracz decyduje o dobieraniu kart poprzez naciśnięcie klawisza D lub S w oknie SFML

W konsoli wyświetlają się informacje o rozgrywce oraz gracz decyduje, czy chce zagrać kolejną partię.

Bibliografia

[1] Cyganek B.: Programowanie w języku C++. Wprowadzenie dla inżynierów. PWN, 2023.

[2] https://www.learncpp.com/cpp-tutorial/generating-random-numbers-using-mersenne-twister/?utm_content=cmp-trueOppenheim_A.V.,_Schafer_R.W.:_Discrete-Time_Signal Processing, Prentice Hall, 1989.

[3] https://www.scaler.com/topics/vector-pair-in-cpp/

[4] https://cpp0x.pl/kursy/Kurs-SFML-2-x-C++/460