Imię i nazwisko Michał Skrzyczek Filip Sikora Mikołaj Samek	Kierunek ITE	Rok studiów i grupa 1 rok 4 grupa
Data zajęć 19.01.2026	Numer i temat sprawozdania Sprawozdanie nr 10 Projekt	

NAZWA: Flanki the Game

CEL I OPIS PROJEKTU:

Celem projektu było stworzenie gry typu physic-based w języku C++. Postawiliśmy na naszą adaptacje studenckiej gry plenerowej „flanki". Dwóch graczy rywalizuje ze sobą, o wygraną, rzucając lotką do celu (puszki) i jak najszybszym wypiciem swojego napoju.

Postaraliśmy się żeby odwzorować realne zasady „flanek":
- Po celnym trafieniu, gracz który trafił „pije" wirtualny napój, podczas gdy przeciwnik musi pobiec ustawić puszkę i wrócić na miejsce.

- Wygrywa gracz, który jako pierwszy opróżni swój napój.

- Gra oferuje tryb dla 2 graczy na jednym komputerze oraz gry ze sztuczną inteligencją.

ZASTOSOWANE TECHNIKI PROGRAMISTYCZNE

Programowanie obiektowe: Kod podzieliliśmy na klasy odpowiedzialne za konkretne aspekty gry np. Gra(logika główna), Bohater (animacja i stan gracza), Pocisk (fizyka lotki), Menu (obsługa interfejsu).
Funkcje i metody: Logikę programu podzieliliśmy na mniejsze bloki realizujące konkretne zadania co ułatwia zarządzanie kodem i pozwala uniknąć pisania kilka razy tej samej funkcjonalności
Pętle:
- Pętla główna gry (while): Odpowiada za ciągłe odświeżanie stanu gry i renderowanie klatek animacji, dopóki okno aplikacji jest otwarte.

- Pętle iteracyjne (for): Wykorzystane m.in. przy tworzeniu postaci do wczytywania sekwencji plików z klatkami animacji.
Instrukcje warunkowe: Rozbudowane systemy decyzyjne sterują logiką gry.
Tablice i kontenery danych: Wykorzystaliśmy kontener do przechowywania klatek animacji oraz użyliśmy tablic wierzchołków do rysowania linii przy celowaniu.
Operacje na plikach: Wykorzystaliśmy operacje odczytu plików zapisanych na dysku w celu załadowania tekstur do gry.
Główna pętla gry: Zastosowano podział na obsługę wejścia, aktualizację stanu gry i

rysowanie klatek.
Maszyna stanów: Obsługa stanów aplikacji: Menu główne -> Rozgrywka oraz obsługa stanów animacji postaci: Rzut -> Bieg -> Spoczynek.
Wektory i matematyka: Wykorzystanie wektorów do obliczenia trajektorii lotu, wykrywanie kolizji oraz wektorów prędkości postaci i lotki.
Zdarzenia: Obsługa wejścia oparta na systemie zdarzeń biblioteki SFML.

WYKORZYSTANE BIBLIOTEKI

1. **SFML (3.0) –** Główna biblioteka wykorzystana do obsługi grafiki 2D, urządzeń wejścia, okna aplikacji oraz wczytywania tekstur.
2. **CMATH:** Do obliczeń potrzebnych w fizyce gry.

WYBRANE FUNKCJONALNOŚCI

KOD

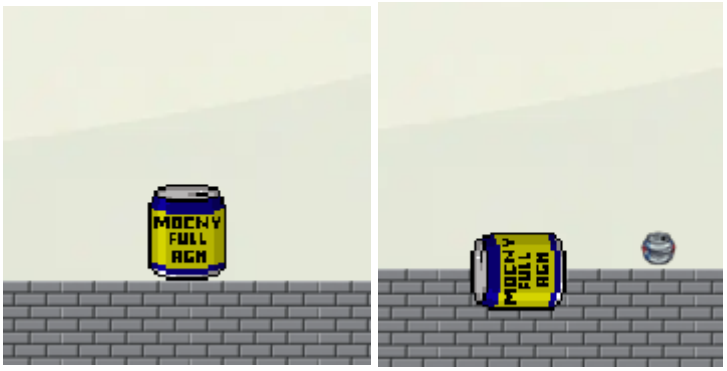
```
void pocisk::rysujCelowanie(sf::RenderWindow& okno)
{
    if (czyCeluje)
    {
        sf::Vertex line[] = {
            sf::Vertex(pozycjaStartowa, sf::Color::White),
            sf::Vertex(pozycja, sf::Color::Magenta)
        };
        okno.draw(line, 2, sf::PrimitiveType::Lines);
    }
}
```

WYNIK

KOD

```
void przeszkoda::przewroc(float predkoscPociskuX)
{
    if (wTrakcieUpadku || std::abs(aktualnyKat) >= 89.0f) return;

    wTrakcieUpadku = true;
    czyTrafiona = true;

    float silaUderzenia = std::abs(predkoscPociskuX);
    float bazowaPredkoscObrotu = silaUderzenia * 0.6f;

    if (bazowaPredkoscObrotu < 2.0f) bazowaPredkoscObrotu = 2.0f;
    if (bazowaPredkoscObrotu > 15.0f) bazowaPredkoscObrotu = 15.0f;

    if (predkoscPociskuX > 0) predkoscObrotu = bazowaPredkoscObrotu;
    else predkoscObrotu = -bazowaPredkoscObrotu;
}
```

WYNIK

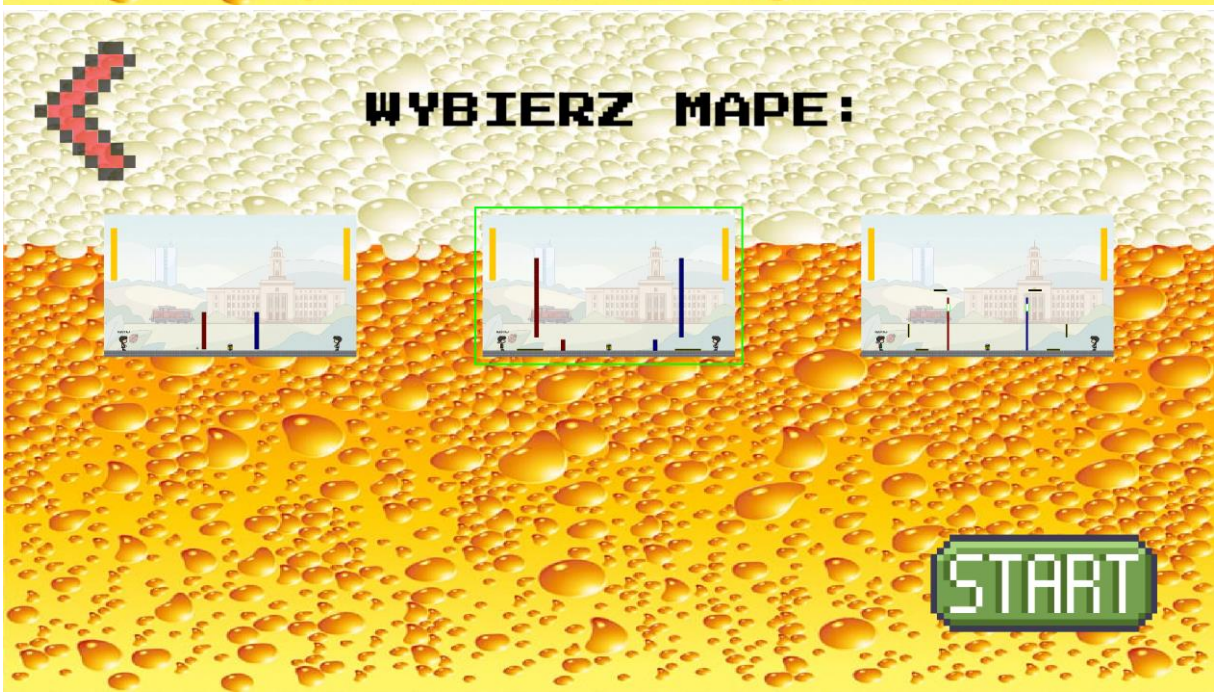


KOD

```
void Menu::rysuj(sf::RenderWindow& okno)
{
    // Prosta pętla renderowania: Clear -> Draw -> Display
    // Nie potrzebujemy tu skomplikowanego potoku graficznego (brak shaderów,
świateł).
    okno.clear(sf::Color::Black);
    okno.draw(spriteTlo);

    if (!showMapSelection) {
        // Ekran głównego menu
        okno.draw(tekstMenu);
        okno.draw(spritePvP);
        okno.draw(spritePvE);
    }
    else {
        // Ekran wyboru mapy
        okno.draw(tekstMapy);
        okno.draw(spriteMapa1);
        okno.draw(spriteMapa2);
        okno.draw(spriteMapa3);
```

```
        okno.draw(spriteBack);
        okno.draw(spriteStart);

        // Podświetlenie wybranej mapy
        if (selectedMap > 0) {
            sf::Sprite* selectedSprite = nullptr;
            switch (selectedMap) {
            case 1: selectedSprite = &spriteMapa1; break;
            case 2: selectedSprite = &spriteMapa2; break;
            case 3: selectedSprite = &spriteMapa3; break;
            }

            if (selectedSprite) {
                sf::RectangleShape highlight(
                    sf::Vector2f(selectedSprite->getGlobalBounds().size.x + 20,
                        selectedSprite->getGlobalBounds().size.y + 20)
                );
                highlight.setPosition(
                    sf::Vector2f(
                        selectedSprite->getPosition().x - 10,
                        selectedSprite->getPosition().y - 10));
                highlight.setFillColor(sf::Color::Transparent);
                highlight.setOutlineColor(sf::Color::Green);
                highlight.setOutlineThickness(3.0f);
                okno.draw(highlight);
            }
        }
    }

    okno.display();
}
```

WYNIK

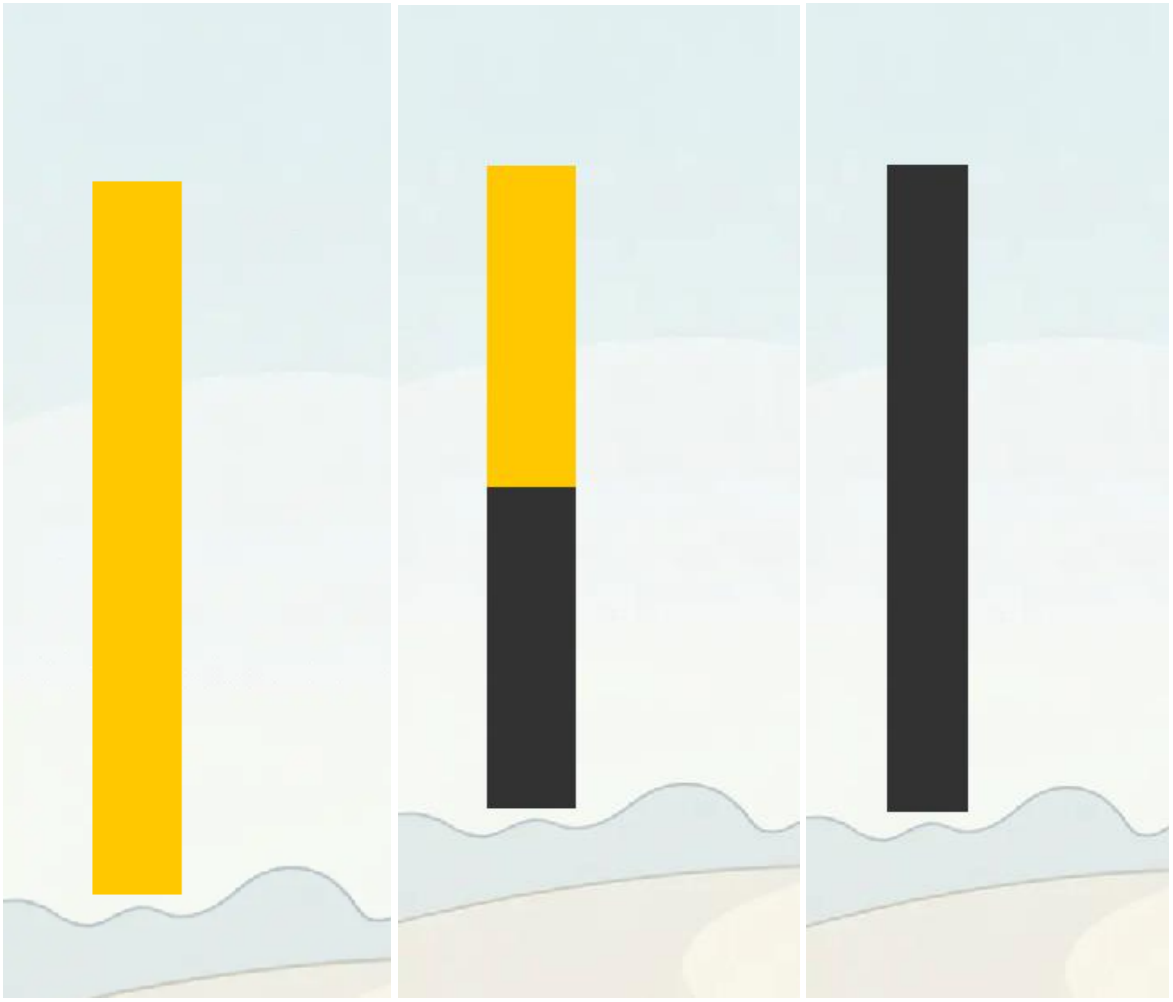


KOD

```
void Gra::obsluzWejscie(sf::RenderWindow& okno) {
    while (const std::optional event = okno.pollEvent()) {
        if (event->is<sf::Event::Closed>()) okno.close();
        if (event->is<sf::Event::KeyPressed>() && event-
>getIf<sf::Event::KeyPressed>()->code == sf::Keyboard::Key::Escape) okno.close();
        if (czyKoniecGry) continue;
```

```
        if (fazaBiegania) {
            if (event->is<sf::Event::KeyPressed>()) {
                const auto& keyEvent = event->getIf<sf::Event::KeyPressed>();
                if (keyEvent->code == sf::Keyboard::Key::Space) {
                    if (turaLewego) obsluzPicie(); else obsluzBieganie();
                }
                if (!graZBotem && keyEvent->code == sf::Keyboard::Key::Enter) {
                    if (turaLewego) obsluzBieganie(); else obsluzPicie();
                }
            }
        }
        else if (!strzalWTok) {
            if (turaLewego) graczLewy.obsluzWejscie(*event, okno);
            else if (!graZBotem) graczPrawy.obsluzWejscie(*event, okno);
        }
    }
}

void Gra::obsluzPicie() {
    if (czyKoniecGry) return;
    if (turaLewego) {
        if (poziomLewego > 0) {
            poziomLewego -= LYK_PIWA;
            if (poziomLewego <= 0.1f) { poziomLewego = 0.0f; sprawdzWygrana(); }
            piwoLewe.setSize(sf::Vector2f(50.0f, (poziomLewego / 100.0f) *
400.0f));
        }
    }
    else {
        if (poziomPrawego > 0) {
            poziomPrawego -= LYK_PIWA;
            if (poziomPrawego <= 0.1f) { poziomPrawego = 0.0f; sprawdzWygrana();
}
            piwoPrawe.setSize(sf::Vector2f(50.0f, (poziomPrawego / 100.0f) *
400.0f));
        }
    }
}
```

WYNIK



LINK DO GITHUBA: https://github.com/Skrzyqu/Flanki