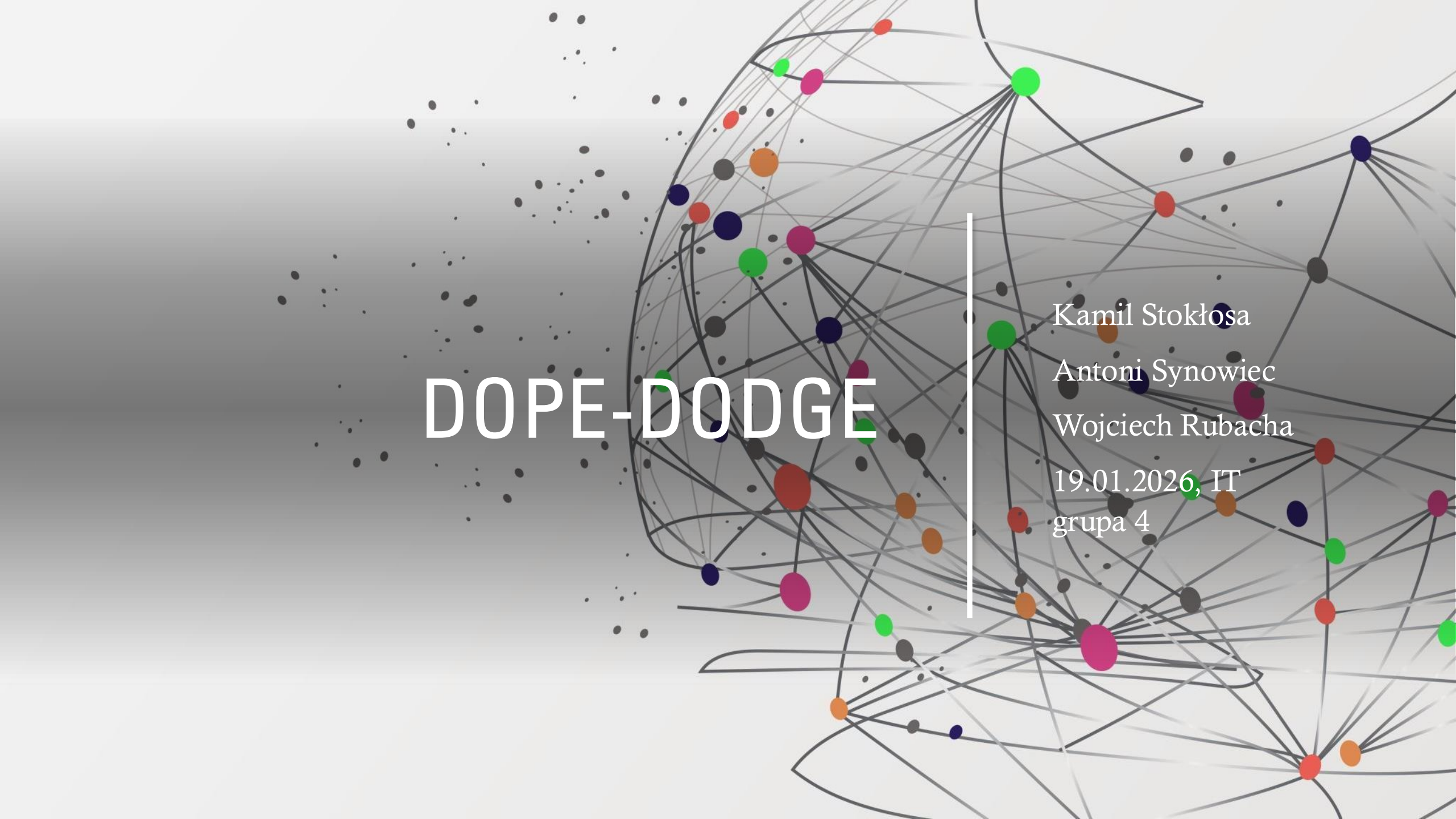


DOPE-DODGE

Kamil Stokłosa

Antoni Synowiec

Wojciech Rubacha

19.01.2026, IT

grupa 4

SILNIK FIZYCZNY

```
void Object::_physics_process(float delta) { // ta funcja dzieje się co klatkę i każda struktura może użyć tej funcji
    float length = std::sqrt(direction.x * direction.x + direction.y * direction.y);// która jest Object może mieć ruch
    if (length != 0) { // aby na ukos nie biegał szybciej
        direction.x /= length;
        direction.y /= length;
        last_direction = direction;
    }
    if (movable) {
        velocity.y = direction.y * speed; // tu liczy się prędkość
        velocity.x = direction.x * speed;
        sprite.move(velocity * delta);
        hard_accel = -5 * max_speed;
    }
    else {
        direction = { 0.f, 0.f }; // jeśli obiekt nie jest ruchomy to stoi w miejscu
        speed = 0;
    }
    if (!max_speed_on) {// tu jest zwalnianie normalne
        if (speed + used_accel * delta > 0.f) {
            speed += used_accel * delta;
        }
        else {
            speed = 0.f;
        }
    }
    if (dash_cooldown - delta >= 0.f) {  // tu obsługo cooldownu dasha
        dash_cooldown -= delta;
    }
    else {
        dash_cooldown = 0.f;
    }
    if (shot_cooldown - delta >= 0.f) {  // tu obsługo cooldownu dasha
        shot_cooldown -= delta;
    }
    else {
        shot_cooldown = 0.f;
    }
    if (speed > max_speed) { // tu to jest dla gracza na sztywno zrobione
        used_accel = hard_accel; // to sprawia że gracz szybko zwalnia do max prędkości
    }           // przez co jest lepszy feal gry i dasha
    else {
        used_accel = accel;
    }  // tu dla normalnego accel
```

OBSŁUGA GRACZA

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::C)) {  //strzał piłką
    if (gracz.shot_cooldown <= 0.f) {
        int ile_pociskow;
        if (gracz.multishot == true) {
            ile_pociskow = 2; //z bonusme strzał 2 razy
        }
        else {
            ile_pociskow = 1;
        }
        int wystrzelone = 0;
        for (int i = 0; i < max_bullets; i++) {  // tu przechodzi przez wszystkie dostępne bullety dostępne
            if (bullets[i].visible == false) {
                bullets[i].visible = true;  // tu zmienia parametry i ustawia dla tego bulleta
                if (!gracz.big_ball) {
                    bullets[i].sprite.setPosition(gracz.sprite.getPosition());
                }
                else {
                    bullets[i].sprite.setPosition(gracz.sprite.getPosition() - gracz.last_direction * 8.f);
                }
                bullets[i].direction = gracz.last_direction;
                bullets[i].speed = 500.f;
                bullets[i].normal_speed = 500.f;
                bullets[i].life_time = 10.f;
                bullets[i].team = gracz.team;
                //ustalnienie kierunku
                if (wystrzelone == 0) {
                    bullets[i].direction = gracz.last_direction;
                }
                else {
                    bullets[i].direction = -gracz.last_direction;
                }
                //Powiększnie piłki
                if (gracz.big_ball) {
                    bullets[i].sprite.setScale({ 2.0f,2.0f });
                }
                else {
                    bullets[i].sprite.setScale({ 1.0f, 1.0f });
                }
                wystrzelone = wystrzelone + 1;
                if (wystrzelone >= ile_pociskow) {
                    break;
                }

            }
        }
        gracz.shot_cooldown = 2;
```

OBSŁUGA GRACZA CZ.2

```
void Object::rotations() {
    if (abs(direction.x) > 0.9f) {
        if (direction.x > 0) {
            sprite.setRotation(sf::degrees(0.f));
        }
        if (direction.x < 0) {
            sprite.setRotation(sf::degrees(180.f));
        }
    }
    else if (abs(direction.y) > 0.9f) {
        if (direction.y > 0) {
            sprite.setRotation(sf::degrees(90.f));
        }
        if (direction.y < 0) {
            sprite.setRotation(sf::degrees(270.f));
        }
    }
    else if (abs(direction.y) > 0.3 and abs(direction.x) > 0.3) {
        if (direction.y > 0) {
            if (direction.x > 0) {
                sprite.setRotation(sf::degrees(45.f));
            }
            if (direction.x < 0) {
                sprite.setRotation(sf::degrees(135.f));
            }
        }
        if (direction.y < 0) {
            if (direction.x > 0) {
                sprite.setRotation(sf::degrees(315.f));
            }
            if (direction.x < 0) {
                sprite.setRotation(sf::degrees(225.f));
            }
        }
    }
}
```

KOLIZJE

Kluczowe, użyte funkcje Biblioteki SFML:

- GetGlobalBounds() - definicja przestrzeni, pobiera aktualne położenie, rozmiar i skalę obiektu oraz tworzy niewidzialny prostokąt (hitbox), który podąża za grafiką danego obiektu.
- FindIntersection() - wyznacza część wspólną (prostokąt kolizji), co pozwala określić głębokość i oś zderzenia.

ODBIJANIE POCISKÓW OD ŚCIAN

```
auto pole_kolizji = bullets[i].sprite.getGlobalBounds().findIntersection(walls[j].sprite.getGlobalBounds());
//zmienna przechowuje mały prostokąt, czyli wspolny obszar piłki i ściany gdy zachodzi kolizja

if (pole_kolizji) { //doszło do zderzenia
    if (pole_kolizji->size.x < pole_kolizji->size.y) { //uderzenie z boku (węższe pole kolizji)
        bullets[i].direction.x *= -1.0f; // odwracanie kierunku poziomego
        //wypchnięcie pocisku poza ścianę o szerokość kolizji
        if (bullets[i].sprite.getPosition().x > walls[j].sprite.getPosition().x) { //uderzenie w prawą stronę ściany
            bullets[i].sprite.move({ pole_kolizji->size.x, 0.0f }); // przesuwanie piłki poza ścianę
        }
        else { //uderzenie w lewą stronę ściany
            bullets[i].sprite.move({ -pole_kolizji->size.x, 0.0f });
        }
    }
    else { //uderzenie z góry lub dołu
        bullets[i].direction.y *= -1.0f;
        //wypchnięcie w pionie o wysokość kolizji
        if (bullets[i].sprite.getPosition().y > walls[j].sprite.getPosition().y) {
            bullets[i].sprite.move({ 0.0f , pole_kolizji->size.y }); // przesuwanie piłki poza ścianę
        }
        else {
            bullets[i].sprite.move({ 0.0f , -pole_kolizji->size.y });
        }
    }
    bullets[i].sprite.move(bullets[i].direction * 2.0f);
}
```

KOLIZJA GRACZY

```
//kolizja między 1 a 2 graczem
//sprawdzanie czy gracze na siebie nachodzą, jeśli tak to tworzy się ich część wspólna
auto pole_zderzenia = gracz.sprite.getGlobalBounds().findIntersection(gracz2.sprite.getGlobalBounds());
if (pole_zderzenia) {//gracze się zderzyli
    if (pole_zderzenia->size.x < pole_zderzenia->size.y) {//jeśli prostokąt kolizji jest węższy to zderzyli się poziomo
        //kolizja pozioma (zderzenie bokami)
        float kierunek;
        if (gracz.sprite.getPosition().x < gracz2.sprite.getPosition().x) {
            kierunek = -1.0f; //gracz 1 jest po lewej, więc pchamy go w lewo
        }
        else {
            kierunek = 1.0f; //gracz 1 jest z prawej więc pchamy go w prawo
        }
        //Każdego gracza przesuwamy o połowę głębokości zderzenia w przeciwnych kierunkach
        gracz.sprite.move({ (pole_zderzenia->size.x / 2.0f) * kierunek,0.0f });
        gracz2.sprite.move({ (pole_zderzenia->size.x / 2.0f) * -kierunek,0.0f });

        gracz.velocity.x = 0;//zerowanie prędkości w osi poziomej, aby gracze przestali na siebie napierać
        gracz2.velocity.x = 0;
    }
    else {
        //kolizja pionowa
        float kierunek;
        if (gracz.sprite.getPosition().y < gracz2.sprite.getPosition().y) {
            kierunek = -1.0f; //gracz 1 jest wyżej, więc wypychamy w górę
        }
        else {
            kierunek = 1.0f; //gracz 1 jest niżej, więc wypychamy w dół
        }
        //Rozsunięcie graczy w pionie o połowę głębokości kolizji
        gracz.sprite.move({ 0.0f, (pole_zderzenia->size.y / 2.0f) * kierunek });
        gracz2.sprite.move({ 0.0f, (pole_zderzenia->size.y / 2.0f) * -kierunek });
        //zerowanie prędkości w osi pionowej
        gracz.velocity.y = 0;
        gracz2.velocity.y = 0;
    }
}
```

ŚLIZGANIE GRACZY PO ŚCIANACH

```
//kolizje dla 1 gracza
auto pole_g1 = gracz.sprite.getGlobalBounds().findIntersection(walls[j].sprite.getGlobalBounds());
if (pole_g1) {
    if (pole_g1->size.x < pole_g1->size.y) {//kolizja pozioma(boczna)
        float kierunek;
        if (gracz.sprite.getPosition().x < walls[j].sprite.getPosition().x) {
            kierunek = -1.0f; // wypchany w lewo, gdyż gracz jest po lewej stronie ściany
        }
        else {
            kierunek = 1.0f;
        }
        //wypchnięcie gracza poza ścianę w osi X
        gracz.sprite.move({ pole_g1->size.x * kierunek, 0.0f });
        gracz.velocity.x = 0; //blokada ruchu w głąb śćiany(X)
        gracz.velocity.y *= 0.8f; //tarcie (spowolnienie)
    }
    else {//kolizja pionowa
        float kierunek;
        if (gracz.sprite.getPosition().y < walls[j].sprite.getPosition().y) {
            kierunek = -1.0f; // wypchany w górę, gdyż gracz jest nad ścianą
        }
        else {
            kierunek = 1.0f;
        }
        //wypchnięcie gracza poza ścianę w osy Y
        gracz.sprite.move({ 0.0f, pole_g1->size.y * kierunek });
        gracz.velocity.y = 0; //blokada ruchu w głąb ściany (Y)
        gracz.velocity.x *= 0.8f; //tarcie (spowolnienie)
    }
}
```

KOLIZJA GRACZ Z POCISKIEM

```
// Trafienie gracza pociskiem

if (bullets[i].sprawdzKolizje(gracz) && bullets[i].team != gracz.team) {
    gracz.health -= 1; //odjęcie punktu życia
    bullets[i].visible = false; //pocisk znika po trafieniu
}
if (bullets[i].sprawdzKolizje(gracz2) && bullets[i].team != gracz2.team) {
    gracz2.health -= 1;
    bullets[i].visible = false;
}
```

POWERUPY

```
void Object::apply_buffs(int rodzaj) {
    if (!buffable) {
        return;
    }
    switch (rodzaj) {
    case 0:
        speed_buff_time = 3.f;
        max_speed += 100;
        speedy = true;
        break;
    case 1:
        slow_buff_time = 3.f;
        max_speed -= 100;
        slowy = true;
        break;
    case 2:
        big_ball_time = 5.f;
        big_ball = true;
        break;
    case 3:
        multishot_time = 10.f;
        multishot = true;
        break;
    }
}
```

- **Speedup** - zwiększenie prędkości, czas trwania: 3 sekundy
- **Slowdown** – spowolnienie przeciwnika, czas trwania: 3 sekundy
- **Big Ball** - zwiększenie rozmiaru pocisku, czas trwania: 5 sekund
- **Multishot** - podwójne strzały w dwóch kierunkach, czas trwania: 10 sekund

ODRADZANIE POWERUPÓW

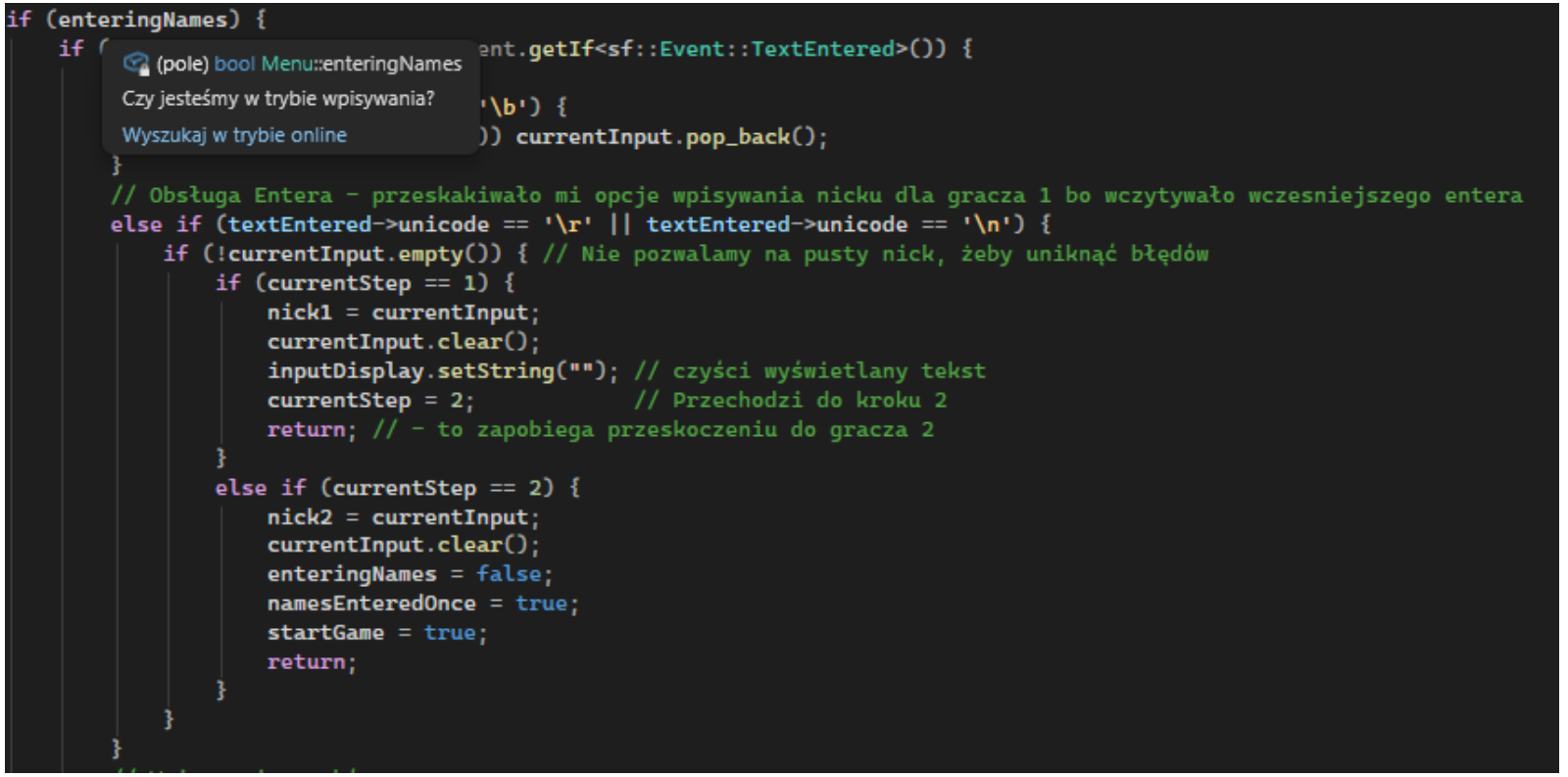
```
//sprawdzanie każdego bonusu
//Speedup
if (speedup.visible == false) {//jeśli bonus został zebrany i jest niewidoczny
    timer_speedup += delta; //zwiększenie liczniku czasu
    if (timer_speedup >= respawn_time_limit) {
        bool kolizja = true;//flaga kontrolująca pętle losowania
        while (kolizja) {//dopóki nie znajdzie miejsca, które nie koliduje ze ścianą
            speedup.sprite.setPosition({ (float)(rand() % 600 + 100), (float)(rand() % 600 + 100) });
            kolizja = false; //zakładam że pozycja jest poprawna
            speedup.visible = true;
            //sprawdzanie czy dotyka ściany
            for (int i = 0; i < walls.size(); i++) {
                if (speedup.sprawdzKolizje(walls[i])) {//wykryto kolizję
                    kolizja = true; //zmianiamy dlagę na true, co wymusza ponowne losowanie
                    break; //wiemy że to miejsce jest złe więc przerywamy pętle for
                }
            }
        }
        timer_speedup = 0.f;//po poprawnym odrodzeniu zerujemy licznik czas
    }
}
```

MASZYNA STANOW I SEPARACJA STANOW GRY

```cpp
enum class GameState //enum zarzadza stanami gry w tym samym oknie
{
    MENU,
    GAME,
    END_SCREEN,
    NAME_INPUT
};
```

```cpp
while (window.isOpen()) {   // to spawia że gra działa dopóki okno jest otwarte
    //wywołaie menu
    if (state == GameState::MENU || state == GameState::END_SCREEN) {
        while (const std::optional event = window.pollEvent()) {
            if (event->is<sf::Event::Closed>()) window.close();
            menu.handleEvent(*event);

            //Dynamiczna aktualizacja głośności muzyki podczas przesuwania paska w menu
            music.setVolume(menu.getVolume());
        }

        if (menu.shouldStartGame()) {
            // resertuje stan gry przed nowym startem
            //Jeśli menu mówi, że gra ma wystartować, upewniamy się, że stan to GAME
                resetujGre(gracz, gracz2, bullets, zycia, walls, seed, speedup, slowdown, big_ball, pdouble);
            menu.resetFlags();
            state = GameState::GAME;
            nick1.setString(menu.getNick1());
            nick1.setOrigin({ nick1.getLocalBounds().size.x / 2, 0 });
            nick2.setString(menu.getNick2());
            nick2.setOrigin({ nick2.getLocalBounds().size.x / 2, 0 });
            clock.restart();
        }

        if (menu.shouldExit()) window.close();

        window.clear();
        menu.draw();
        window.display();

        continue;
    }
```

OBSLUGA WPISYWANIA NICKOW




```
if (enteringNames) {
    if (                                ent.getIf<sf::Event::TextEntered>()) {
                                        '\b') {
                                    )) currentInput.pop_back();
        }
        // Obsługa Entera - przeskakiwało mi opcje wpisywania nicku dla gracza 1 bo wczytywało wczesniejszego entera
        else if (textEntered->unicode == '\r' || textEntered->unicode == '\n') {
            if (!currentInput.empty()) { // Nie pozwalamy na pusty nick, żeby uniknąć błędów
                if (currentStep == 1) {
                    nick1 = currentInput;
                    currentInput.clear();
                    inputDisplay.setString(""); // czyści wyświetlany tekst
                    currentStep = 2;            // Przechodzi do kroku 2
                    return; // - to zapobiega przeskoczeniu do gracza 2
                }
                else if (currentStep == 2) {
                    nick2 = currentInput;
                    currentInput.clear();
                    enteringNames = false;
                    namesEnteredOnce = true;
                    startGame = true;
                    return;
                }
            }
        }
```

REINICJALIZACJA SWIATA

```cpp
void resetujGre(Object& g1, Object& g2, std::vector<Bullet>& b, std::vector<Zycie>& z, std::vector<Object>& walls, int seed, Object& speedup, Object& slowdown, Object& big_ball, Object& pdouble) {
    // Przywracanie zdrowia i pozycji gracza 1
    g1.health = 3;
    g1.velocity = { 0.f, 0.f };
    g1.sprite.setPosition({ 100.f, 100.f });
    g1.visible = true;
    g1.big_ball_time = 0.f;
    g1.multishot_time = 0.f;
    g1.speed_buff_time = 0.f;
    g1.slow_buff_time = 0.f;

    // Przywracanie zdrowia i pozycji gracza 2
    g2.health = 3;
    g2.velocity = { 0.f, 0.f };
    g2.sprite.setPosition({ 700.f, 700.f });
    g2.visible = true;
    g2.big_ball_time = 0.f;
    g2.multishot_time = 0.f;
    g2.speed_buff_time = 0.f;
    g2.slow_buff_time = 0.f;

    // Ukrywanie wszystkich pocisków na ekranie
    for (auto& bullet : b) {
        bullet.visible = false;
    }

    // Przywracanie widoczności serduszek
    for (auto& serce : z) {
        serce.visible = true;
    }
```

PODZIAŁ OBOWIĄZKÓW

- **Kamil Stokłosa: Silnik fizyczny, Obsługa gracza, Debuging, Grafika**
- **Antoni Synowiec: Kolizje, Powerupy, Organizacja kodu**
- **Wojciech Rubacha: Graficzny interfejs użytkownika (GUI)**

PREZENTACJA DZIAŁANIA PROGRAMU: