PROJEKT

Wykonał Michał Kuckowicz

2 gr. ITE

OPIS GRY

Gra polega na skutecznym ścinaniu drzew (otrzymuje się za to punkty), unikając przy tym różne przeszkody.

Zawiera w sobie:

- Animacje postaci: skakanie, kucanie, chód oraz zamach siekierą
- Czasomierz oraz licznik punktów
- Intuicyjne menu
- Zwiększający się progresywnie poziom trudności (statycznie ustawione co 5 sekund)
- Dopasowane hitboxy gracza oraz przeszkód (dokładne kolizje)
- Poruszające się w stronę gracza przeszkody (przy czym wygląda to jakby postać szła w ich kierunku)
- Losowo generujące się przeszkody
- Sterowanie postaci za pomocą: W (skok), S (kucanie), Spacja (zamach)

FRAGMENTY KODU

1. ANIMACJA POSTACI

```
void Game::initCharacter(unsigned char i)
{
    if (!this->TextureOfCharacter[0].loadFromFile("./graphics/timberman1test.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    if (!this->TextureOfCharacter[1].loadFromFile("./graphics/timberman2test.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    if (!this->TextureOfCharacter[2].loadFromFile("./graphics/timbermantest.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    if (!this->TextureOfCharacter[3].loadFromFile("./graphics/timbermaninaction.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    if (!this->TextureOfCharacter[4].loadFromFile("./graphics/timbermancrouch.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    this->Character.setTexture(TextureOfCharacter[i]);
}
```

CHÓD

```
if (!this->action) {
    if (this->change > 35.0f) {
        this->initCharacter(this->zero_one);
        if (this->zero_one >= 1) {
            this->zero_one = 0;
        }
        else {
            this->zero_one += 1;
        }
        this->change = 0.0f;
    }
    else {
        this->change += 1.0f;
    }
    this->Character.setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfCharacter[0].getSize().y);
    this->Hitboxes[0].setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfHitboxes[0].getSize().y);
    this->Hitboxes[1].setPosition(50 + 54, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfHitboxes[1].getSize().y);
}
```

ZAMACH

```
case sf::Event::KeyReleased:
    if (this->event.key.code == sf::Keyboard::Space) {
        this->action = 1;
        this->initCharacter(3);
        /*this->action = 0;
        this->initCharacter(1);*/
        this->Character.setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfCharacter[3].getSize().y);
    }
```

```
if (this->Character.getTexture() == &this->TextureOfCharacter[3]) {
    this->Character.setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfCharacter[3].getSize().y);
    if (this->atimer >= this->atimermax) {
        this->Character.setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfCharacter[0].getSize().y);
        this->initCharacter(1);
        this->action = 0;
        this->atimer = 0.0f;
    }
    this->atimer += 1.0f;
}
```

SKOK

```
if (this->event.key.code == sf::Keyboard::W || this->event.key.code == sf::Keyboard::Up) {
    //this->action = 1;
    //this->initCharacter(3);
    this->action = 1;
    this->initCharacter(2);
}
```

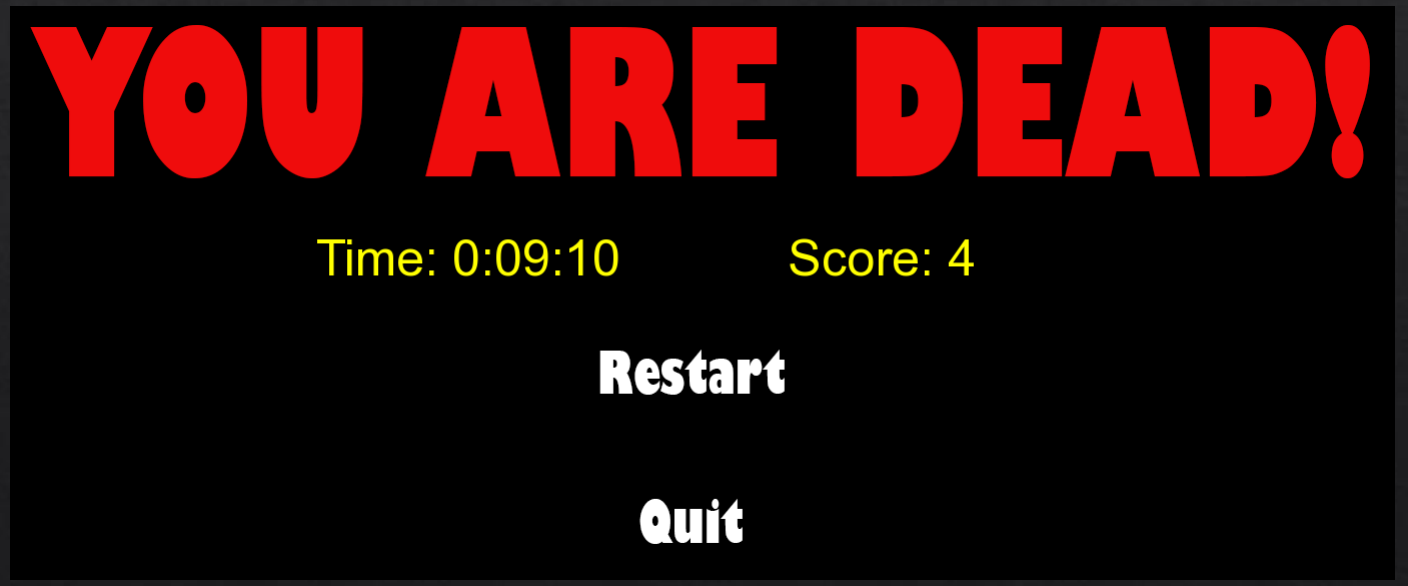
```
else if (this->Character.getTexture() == &this->TextureOfCharacter[2]) {
    if (this->Character.getPosition().y <= this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfCharacter[2].getSize().y) {
        this->Character.move(0, this->jumpmove);
        this->Hitboxes[0].move(0, this->jumpmove);
        this->Hitboxes[1].move(0, this->jumpmove);
        if (this->Character.getPosition().y <= this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfCharacter[2].getSize().y - this->jumpmax) {
            this->jumpmove = -this->jumpmove;
        }
    }
    else {
        this->action = 0;
        this->jumpmove = -this->jumpmove;
    }
}
```

KUCANIE

```
if (this->event.key.code == sf::Keyboard::S || this->event.key.code == sf::Keyboard::Down) {
    //this->action = 1;
    //this->initCharacter(3);
    this->action = 1;
    this->initCharacter(4);
}
```

```
else if (this->Character.getTexture()== &this->TextureOfCharacter[4]) {
    this->Character.setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfCharacter[4].getSize().y);
    this->Hitboxes[0].setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfHitboxes[0].getSize().y);
    this->Hitboxes[1].setPosition(50 + 54, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfHitboxes[1].getSize().y);
}
```

2. CZASOMIERZ, LICZNIK PUNKTÓW I KOLIZJE



CZASOMIERZ

```
std::string Game::updateTimer()
{
    std::string t = "";
    this->time = clock.getElapsedTime();
    if (static_cast<int>(this->time.asSeconds()) >= 60) {
        this->minutes += 1;
        this->time = clock.restart();
        this->count = 1;
    }
    if (this->minutes <= 9) {
        t += std::to_string(this->minutes)+":";
    }
    else {
        t += std::to_string(this->minutes) + ":";
    }
    if (static_cast<int>(this->time.asSeconds()) <= 9) {
        t += "0"+std::to_string(static_cast<int>(this->time.asSeconds())) + ":";
    }
    else {
        t += std::to_string(static_cast<int>(this->time.asSeconds())) + ":";
    }
    if ((this->time.asMilliseconds() % 1000 / 10) <= 9) {
        t += "0" + std::to_string(this->time.asMilliseconds() % 1000 / 10);
    }
    else {
        t += std::to_string(this->time.asMilliseconds() % 1000 / 10);
    }
    return t;
}
```

PUNKTY I KOLIZJE

```
if (!this->action) {
    if (this->Entities[i].getGlobalBounds().intersects(this->Hitboxes[0].getGlobalBounds()) || this->Entities[i].getGlobalBounds().intersects(this->Hitboxes[1].getGlobalBounds())) {
        if (this->Entities[i].getTexture() == &this->TextureOfEntity[0]) {
            if (this->Hitboxes[1].getGlobalBounds().contains(this->Entities[i].getGlobalBounds().getPosition().x + 189, this->Entities[i].getGlobalBounds().getPosition().y + 525)) {
                //this->Entities.erase(this->Entities.begin() + i);
                Death();
            }
        }
        else {
            //this->Entities.erase(this->Entities.begin() + i);
            Death();
        }
    }
}
else {
    if (this->Character.getTexture() == &this->TextureOfCharacter[3]) {
        if (this->Entities[i].getGlobalBounds().intersects(this->Hitboxes[2].getGlobalBounds())) {
            if (this->Entities[i].getTexture() == &this->TextureOfEntity[0]) {
                if (this->Hitboxes[2].getGlobalBounds().contains(this->Entities[i].getGlobalBounds().getPosition().x + 189, this->Entities[i].getGlobalBounds().getPosition().y + 525)) {
                    this->Entities.erase(this->Entities.begin() + i);
                    this->points += 1;
                }
            }
            else {
                //this->Entities.erase(this->Entities.begin() + i);
                Death();
            }
        }
    }
    else if(this->Character.getTexture() ==&this->TextureOfCharacter[2]){
        if (this->Entities[i].getGlobalBounds().intersects(this->Hitboxes[1].getGlobalBounds()) || this->Entities[i].getGlobalBounds().intersects(this->Hitboxes[0].getGlobalBounds())) {
            if (this->Entities[i].getTexture() == &this->TextureOfEntity[0]) {
                if (this->Hitboxes[1].getGlobalBounds().contains(this->Entities[i].getGlobalBounds().getPosition().x + 189, this->Entities[i].getGlobalBounds().getPosition().y + 500) || this->Hitboxes[0].getGlobalBounds().contains(this->Entitie
                    Death();
                }
            }
            else {
                //this->Entities.erase(this->Entities.begin() + i);
                Death();
            }
        }
    }
    else {
        if (this->Entities[i].getGlobalBounds().intersects(this->Hitboxes[3].getGlobalBounds())) {
            if (this->Entities[i].getTexture() == &this->TextureOfEntity[0]) {
                if (this->Hitboxes[3].getGlobalBounds().contains(this->Entities[i].getGlobalBounds().getPosition().x + 189, this->Entities[i].getGlobalBounds().getPosition().y + 525)) {
                    Death();
                }
            }
            else {
                //this->Entities.erase(this->Entities.begin() + i);
                Death();
            }
        }
    }
}
```

3. HITBOXY POSTACI I PRZESZKODY

```
void Game::initCharacter(unsigned char i)
{
    if (!this->TextureOfCharacter[0].loadFromFile("./graphics/timberman1test.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    if (!this->TextureOfCharacter[1].loadFromFile("./graphics/timberman2test.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    if (!this->TextureOfCharacter[2].loadFromFile("./graphics/timbermantest.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    if (!this->TextureOfCharacter[3].loadFromFile("./graphics/timbermaninaction.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    if (!this->TextureOfCharacter[4].loadFromFile("./graphics/timbermancrouch.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drwala";
    }
    this->Character.setTexture(TextureOfCharacter[i]);
}
```

HITBOXY

```
void Game::initHitboxes()
{
    if (!this->TextureOfHitboxes[0].loadFromFile("./graphics/hitbox1.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury hitboxu drwala";
    }
    if (!this->TextureOfHitboxes[1].loadFromFile("./graphics/hitbox2.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury hitboxu drwala";
    }
    if (!this->TextureOfHitboxes[2].loadFromFile("./graphics/timbermaninaction.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury hitboxu drwala";
    }
    if (!this->TextureOfHitboxes[3].loadFromFile("./graphics/timbermancrouch.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury hitboxu drwala";
    }
    this->Hitboxes[0].setTexture(TextureOfHitboxes[0]);
    this->Hitboxes[0].setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfHitboxes[0].getSize().y);
    this->Hitboxes[1].setTexture(TextureOfHitboxes[1]);
    this->Hitboxes[1].setPosition(50+54, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfHitboxes[1].getSize().y);
    this->Hitboxes[2].setTexture(TextureOfHitboxes[2]);
    this->Hitboxes[2].setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfHitboxes[2].getSize().y);
    this->Hitboxes[3].setTexture(TextureOfHitboxes[3]);
    this->Hitboxes[3].setPosition(50, this->resolution.height - this->TextureOfBackground[3].getSize().y - this->TextureOfHitboxes[3].getSize().y);
}
```

PRZESZKODY

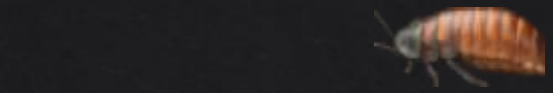
```
void Game::initEntities()
{
    if (!this->TextureOfEntity[0].loadFromFile("./graphics/tree.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury drzewa";
    }
    if (!this->TextureOfEntity[1].loadFromFile("./graphics/bugcut.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury robaka";
    }
    if (!this->TextureOfEntity[2].loadFromFile("./graphics/bird.png")) {
        std::cout << "ERROR: nie mozna zaladowac tekstury ptaka";
    }
    this->Entity[0].setTexture(this->TextureOfEntity[0]);
    this->Entity[1].setTexture(this->TextureOfEntity[1]);
    this->Entity[2].setTexture(this->TextureOfEntity[2]);
}
```

```
this->ent = rand() % 3;
/*if (this->Entities.size() < this->maxentities) {*/
    if (this->etimer >= this->etimermax) {
        this->SpawnEntities(ent);
        this->etimer = 0.f+rand()%25;
    }
    else {
        this->etimer+= 1.f;
    }
```

```
void Game::SpawnEntities(unsigned char i)
{
    this->pos = rand() % 2 + 1;
    switch (pos) {
        case 1:
            pos = 250;
            break;
        case 2:
            pos = 200;
            break;
        case 3:
            pos = 125;
            break;
    }
    this->Entity[0].setPosition(this->resolution.width, this->resolution.height - (this->TextureOfBackground[3].getSize().y + this->TextureOfEntity[0].getSize().y ));
    this->Entity[1].setPosition(this->resolution.width, this->resolution.height - (this->TextureOfBackground[3].getSize().y + this->TextureOfEntity[1].getSize().y ));
    this->Entity[2].setPosition(this->resolution.width, this->resolution.height - (this->TextureOfBackground[3].getSize().y + this->TextureOfEntity[2].getSize().y + pos));

    this->Entities.push_back(this->Entity[i]);
}
```

4. POZIOM TRUDNOŚCI, GENEROWANIE PRZESZKÓD ORAZ PORUSZANIE NIMI



ZMIANA POZIOMU TRUDNOŚCI

```cpp
void Game::updateDifficulty()
{
    if (static_cast<int>(this->time.asSeconds())/this->count>=5) {
        this->space -= 0.5f;
        this->etimermax -= 2.5f;
        this->count++;
        std::cout << "Speed changes: " << this->space<<" per tick\n";
        std::cout << "Max time changes: " << this->etimermax << "\n";
        //std::cout << this->Character.getGlobalBounds().getPosition()
    }
}
```

LOSOWE GENEROWANIE PRZESZKÓD

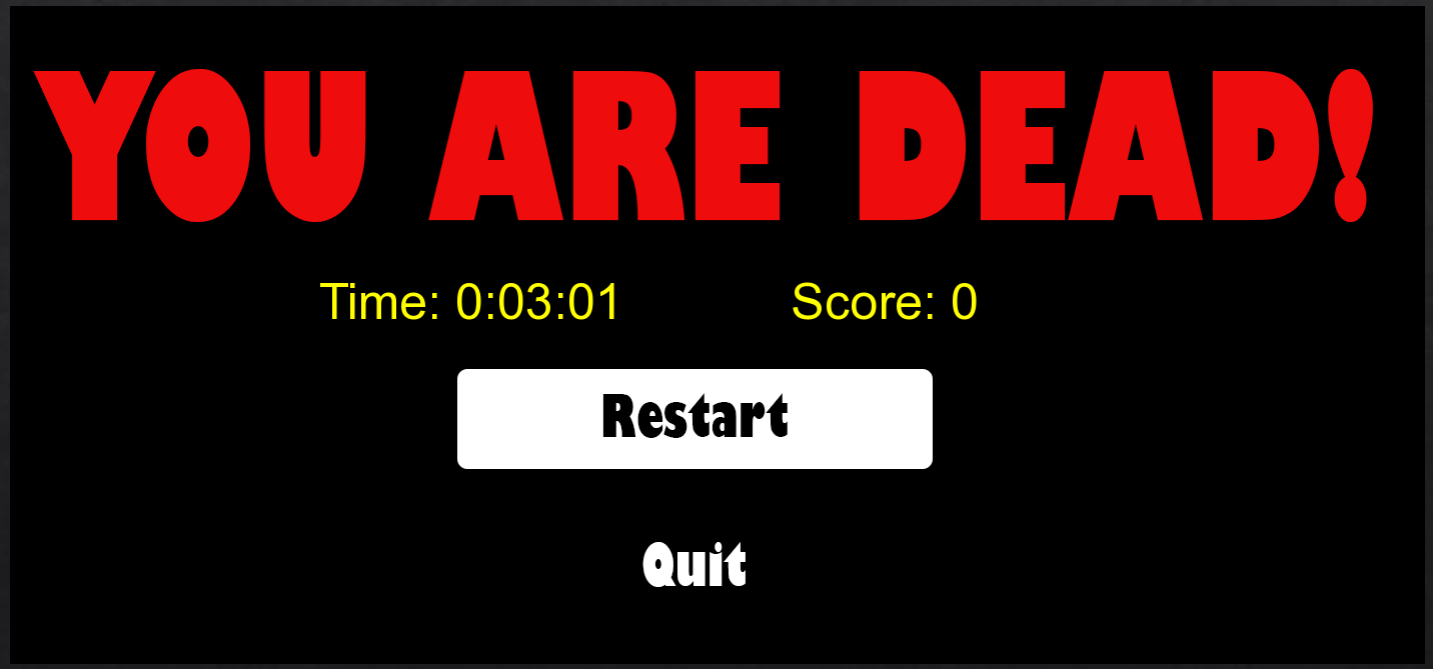
```
this->ent = rand() % 3;
/*if (this->Entities.size() < this->maxentities) {*/
    if (this->etimer >= this->etimermax) {
        this->SpawnEntities(ent);
        this->etimer = 0.f+rand()%25;
    }
    else {
        this->etimer+= 1.f;
    }
```

PORUSZANIE PRZESZKODAMI

```
//obiekty w grze
std::vector<sf::Sprite> Entities;
```

```
for (int i = 0; i < this->Entities.size(); i++) {
    this->Entities[i].move(this->space, 0);
    if (this->Entities[i].getPosition().x+this->Entities[i].getGlobalBounds().getSize().x <= 0) {
        this->Entities.erase(this->Entities.begin() + i);
    }
```

5. MENU



START MENU

```
if (!this->startGame) {
    //Start button
    if (sf::Mouse::isButtonPressed(sf::Mouse::Left) == true) {
        if (this->buttons[0].getGlobalBounds().contains(this->mousePosView)) {
            Start();
        }
    }
    if (this->buttons[0].getGlobalBounds().contains(this->mousePosView)) {
        this->buttons[0].setTexture(this->TextureOfButtons[1]);
    }
    else {
        this->buttons[0].setTexture(this->TextureOfButtons[0]);
    }

    //Quit Button
    if (sf::Mouse::isButtonPressed(sf::Mouse::Left) == true) {
        if (this->buttons[1].getGlobalBounds().contains(this->mousePosView)) {
            this->window->close();
        }
    }
    if (this->buttons[1].getGlobalBounds().contains(this->mousePosView)) {
        this->buttons[1].setTexture(this->TextureOfButtons[3]);
    }
    else {
        this->buttons[1].setTexture(this->TextureOfButtons[2]);
    }
}
```

GAMEOVER MENU

```
else {
    //Restart button
    if (sf::Mouse::isButtonPressed(sf::Mouse::Left) == true) {
        if (this->buttons[2].getGlobalBounds().contains(this->mousePosView)) {
            Start();
        }
    }
    if (this->buttons[2].getGlobalBounds().contains(this->mousePosView)) {
        this->buttons[2].setTexture(this->TextureOfButtons[5]);
    }
    else {
        this->buttons[2].setTexture(this->TextureOfButtons[4]);
    }

    //Quit Button
    if (sf::Mouse::isButtonPressed(sf::Mouse::Left) == true) {
        if (this->buttons[3].getGlobalBounds().contains(this->mousePosView)) {
            this->window->close();
        }
    }
    if (this->buttons[3].getGlobalBounds().contains(this->mousePosView)) {
        this->buttons[3].setTexture(this->TextureOfButtons[3]);
    }
    else {
        this->buttons[3].setTexture(this->TextureOfButtons[2]);
    }
}
```

PREZENTACJA GRY: