Bytom, 14.06.2020r.

Laboratorium Programowania Komputerów 4

Temat: „Gra komputerowa - Tanks”

Autor: Michał Cholewa

Informatyka semestr 4 grupa 3 sekcja 2

Prowadzący: dr inż. Anna Gorawska

1. Temat

Celem programu jest stworzenie gry komputerowej w grafice 2D typu shooter. Gracz porusza się małym czołgiem i jego zadaniem jest zniszczyć jak największą ilość jednostek przeciwnika i zdobycie jak największej liczby punktów. Przeciwnicy będą to utrudniać graczowi poruszając się po mapie oraz strzelając. Na mapie będą również znajdować się różne elementy planszy w postaci bloków o różnych właściwościach np. bloczki ceglane możliwe do zniszczenia, bloczki metalowe niemożliwe do zniszczenia oraz bloczki bluszczu, pod którymi słabo widać poruszający się pojazd lub kulę. W grze będzie również dostępne Menu dzięki któremu, użytkownik będzie mógł wybrać poziom trudności gry oraz przypisać klawisze z klawiatury do sterowania graczem. W zależności od poziomu trudności, gracz będzie otrzymywać różną ilość punktów i przeciwnicy będą trudniejsi do pokonania. Wraz ze wzrostem poziomu trudności, prędkość poruszania się przeciwników i ich kul będzie proporcjonalnie rosnąć. Gra będzie kończyć się jeśli gracz zostanie zniszczony przez kulę przeciwnika. Plansza będzie odczytywana z pliku. W celu zaimplementowania gry i możliwości interakcji w czasie rzeczywistym została użyta biblioteka SFML.

2. Analiza, projektowanie

2.1. Algorytmy, struktury danych, ograniczenia specyfikacji

Algorytmy:

W programie zaimplementowano wiele algorytmów sprawdzających kolizję. Wiele z nich opiera się o prostą algebrę, np. na podstawie położenia jednostki można stwierdzić czy jest ona poza mapą czy też nie. Najistotniejszym algorytmem sprawdzania kolizji jest algorytm sprawdzania kolizji z bloczkami. Aby nie sprawdzać każdego bloczku z każdym graczem, plansza została podzielona na 26 bloczków w rzędach i kolumnach. Na podstawie kierunku gracza oraz jego położenia sprawdzane są tylko bloczki znajdujące się obok gracza tzn. program oblicza na których aktualnie bloczkach znajduje się gracz.

Struktury:

Program w głównej mierze korzysta z tablic wektorów do przechowywania wskaźników na obiekty jednostek oraz kul. Wykorzystywane są również mapy do przechowywania kodów klawiszy oraz tekstur wykorzystywanych w grze.

Ograniczenia specyfikacji:

Ograniczeniem jest format danych wejściowych. Program może przyjmować dane do odczytu mapy tylko w postaci pliku o formacie .csv lub .txt.

3. Specyfikacja zewnętrzna

3.1. Obsługa programu

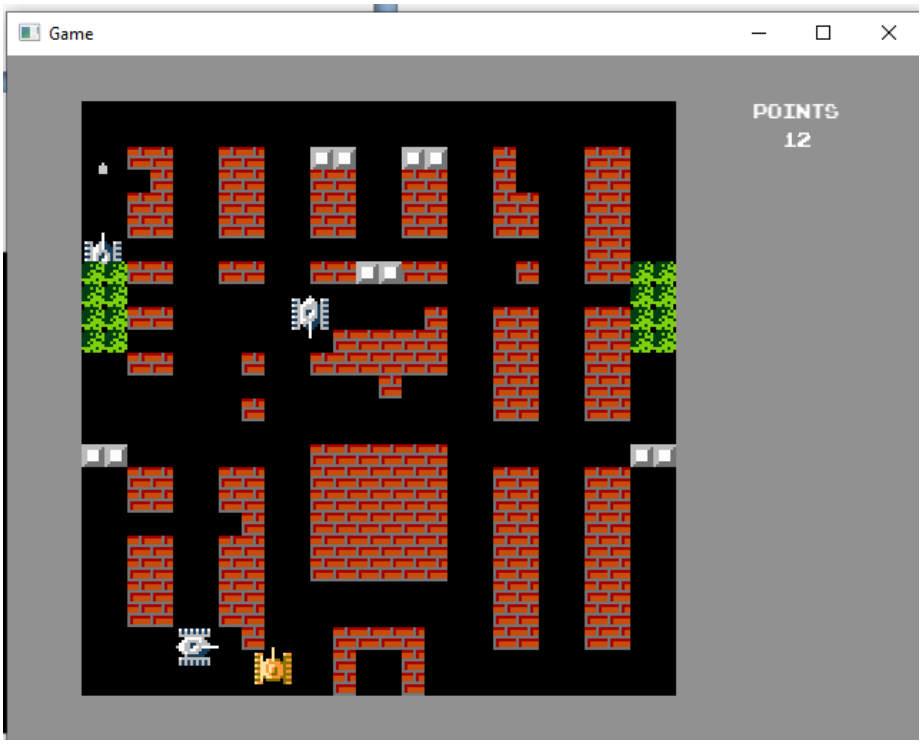
Program jest obsługiwany przy pomocy klawiatury. Po załączeniu programu załącza nam się główne menu z tytułem gry oraz prostym interfejsem po którym poruszamy się używając domyślnych klawiszy do sterowania gracza i menu – strzałek na klawiaturze, klawisza Enter oraz klawisza ESC. W głównym menu dostępne mamy trzy opcje: zagrać, zobaczyć opcje gry, wyjść z gry. Aktualnie wybraną pozycję wskazuje nam kursor w postaci tekstury czołgu gracza zwróconego w stronę aktualnie wybranej opcji.

Rysunek 1 Wygląd głównego menu

Po wejściu do opcji mamy do wyboru poziomy trudności oraz informację pod jakimi klawiszami aktualnie można sterować graczem. Po wciśnięciu klawisza Enter można te opcje zmieniać. W przypadku zamiany domyślnych klawiszy na inne pojawi się informacja z prośbą o wciśnięcie klawisza pod którym ma znajdować się dana interakcja.

Rysunek 2 Menu opcji wraz z wyświetloną prośbą o wciśnięcie klawisza

Do poprzedniego widoku po wybraniu ustawień możemy cofnąć się używając klawisza ESC. Po wyborze opcji pierwszej w głównym menu rozpoczyna się rozgrywka. W jej trakcie należy zdobyć jak największą ilość punktów niszcząc przeciwników, którzy pojawiają się w czasie nie krótszym niż 2s od ostatniego przeciwnika, który się pojawił na planszy.



Rysunek 3 Wygląd gry w trakcie rozgrywki

Jeżeli kula któregoś z przeciwników zniszczy gracza gra wyświetla komunikat jak na obrazku poniżej.

Rysunek 4 Komunikat po przegranej grze

3.2. Format danych wejściowych

- Ilość danych jest ograniczona pamięcią operacyjną.
- Danymi wejściowymi są pliki w formacie *.txt* lub *.csv* z informacją o rozmieszczeniu różnych rodzajów bloczków na planszy. Rodzaje bloczków są reprezentowane jako liczby od 0 - … w zależności od ilości różnych bloczków.
- Użytkownik może wprowadzić dowolną ilość informacji o planszy, ale zostanie odczytanych tylko tyle ile potrzeba, aby zapełnić planszę.

4. Specyfikacja wewnętrzna

4.1. Klasy i szablony

Całość projektu składa się z 9 klas. Każda z nich jest odpowiedzialna za różną funkcjonalność. Poniżej jest ich krótki opis. **Szczegółowy opis wszystkich klas, metod, typów i funkcji w załączniku.**

Podstawową klasą jest klasa „Game”. Jest to klasa przechowująca i zarządzająca wszystkimi obiektami w grze. Odpowiada ona za rysowanie, aktualizowanie, inicjowanie, usuwanie obiektów gry takich jak kule, przeciwnicy, plansza itp. Zawiera ona również metody sprawdzające kolizje kul oraz jednostek z krańcem planszy.

Kolejną klasą jest klasa „Object”. Jest to klasa abstrakcyjna reprezentująca obiekt gry np. jednostkę na mapie, kulę itp. Zawiera ona pola i metody pozwalające na określenie podstawowych cech obiektów w grze tj. położenia, tekstur, prędkości, zwrotu w którym się porusza itp.

Klasa Block jest pochodną klasy Object. Reprezentuje ona jednak statyczny bloczek planszy. Zawiera dodatkowo pole do określenia typu bloczka i skorelowanej z nim tekstury oraz zawiera metodę do ustawiania typu bloczka.

Klasa Bullet jest pochodną klasy Object i reprezentuję kulę wystrzeloną przez gracza lub przez przeciwnika. Zawiera dodatkowo metody do przesuwania kuli w odpowiednim kierunku oraz dodatkowe pola z informacją o rozmiarze i o jej przynależności.

Klasa Player również dziedziczy po klasie Object. Zawiera dodatkowe pole z informacją o rozmiarze gracza oraz metody przesuwające go i ustawiające mu kierunek. Oprócz tego zawiera ona metodę pozwalającą na obliczania na których bloczkach aktualnie znajduje się gracz, aby łatwiej można było sprawdzać kolizję.

Klasa Enemy jest ostatnią klasą dziedzicząca po klasie Object. Jest ona podobna do klasy Player z tym, że przeciwnik jest sterowany z użyciem silnika pseudolosowego. Klasa ta zawiera dwa dodatkowe pola liczące czas – „driving_timer” i „shooting_timer”. Służą one do określenia czy upłynęła określona chwila czasu od poprzedniego ustawienia czasu dla akcji przeciwnika, aby przeciwnicy płynniej się poruszali, w bardziej naturalny sposób. Klasa zawiera metody do wyboru akcji przeciwnika co określony czas oraz resetowaniu liczników i wyboru nowej akcji po upłynięciu wylosowanego wcześniej momentu czasu.

Klasa Level jest odpowiedzialna za planszę w grze. Odczytuje ją z pliku .csv lub .txt. Zawiera dwuwymiarową tablicę bloczków oraz jej rozmiary. Zawiera metodę do ustawiania typu bloczka w odpowiednim rzędzie i kolumnie planszy.

Klasa Sprites jest klasą zawierającą mapę tekstur i korzysta ona z wzorca singleton. Przy inicjowaniu klasy wczytuje ona dane do mapy z odpowiednim kluczem. Zawiera metodę do pobierania interesującego nas fragmentu tekstury.

Klasa Menu odpowiada za cały interfejs użytkownika w grze. Inicjuje odpowiednie napisy i pozycje tych napisów w grze. Dzięki niej użytkownik może również zmienić przypisanie klawiszy do pożądanej akcji w menu.

4.2. Realizacja tematów

Do napisania gry pomocne okazało się wykorzystanie kilku tematów omawianych na laboratorium. Wykorzystałem RTTI, Szablony, Kontenery STL oraz iteratory. RTTI zostało wykorzystane do określania czy przeciwnik ma zostać zniszczony oraz wywoływania akcji przeciwnika poprzez użycie dynamic_cast.

```
for (auto it = entities.begin() + 1; it < entities.end(); it++)
{
    dynamic_cast<Enemy*>(*it)->Choose_action();
}
```

Rysunek 5 Przykład użycia dynamic_cast do wywołania metody przeciwnika oraz iteratorów w metodzie Run() klasy Game.

```
bool Game::Check_if_bullet_destroys_entity(Object * bullet)
{
    for (auto it2 = entities.begin(); it2 < entities.end(); )
    {

        if (bullet->Get_position().x > (*it2)->Get_position().x &&
            bullet->Get_position().x < (*it2)->Get_position().x + 32 &&
            bullet->Get_position().y > (*it2)->Get_position().y &&
            bullet->Get_position().y < (*it2)->Get_position().y + 32) //sprawdz czy kula nachodzi za gracza
        {
            Object * ob = *it2;
            if (typeid(*ob) != typeid(Player) ) //RTTI do sprawdzenia czy gracz umarł jesli nie jest gracza
            {
                if (dynamic_cast<Bullet*>(bullet)->get_belongingness() == false) // jesli jest od przeciwnika
                {
                    return true;
                }
                else {
                    score += 10 * ratio;
                }
                --enemies_on_map;
            }
            else {
                Set_game_state(OVER);


            }

            delete *it2;
            it2 = entities.erase(it2);
            return true;
        }
        else {
            ++it2;
        }

    }

    return false;
}
```

Rysunek 6 Użycie RTTI do sprawdzenia czy kula należy do gracza. Dzięki RTTI przeciwnicy nie niszczą się wzajemnie

Kolejny temat – szablony – został użyty do napisania funkcji, które ułatwiają pozycjonowanie i ustawianie punktu zaczepienia różnych obiektów graficznych z biblioteki SFML np. tekstów, tekstur, sprite'ów itp.

```
template<class T>
void CenterOrigin(T & t)
{
    t.setOrigin(t.getLocalBounds().width / 2.0f, t.getLocalBounds().height/2.0f);
}

template<class T>
void CenterPosition_X(T & t, sf::RenderWindow *& window, float y)
{
    t.setPosition(window->getSize().x/2.0f, y);
}

template<class T>
void SetPositionfromCenter(T & t, sf::RenderWindow *& window, float x, float y)
{
    t.setPosition(window->getSize().x / 2.0f + x, window->getSize().y / 2.0f + y);
}
```

Rysunek 7 Szablony funkcji

Pierwszy z nich ustawia pozycję zaczepienia typu w jego środku. Dzięki temu pozycja obiektu x, y zawsze jest w środku tekstury lub tekstu. Druga funkcja pozwala ustawiać pozycję y zawsze zachowując x na środku ekranu. Trzecia funkcja pozwala pozycjonować obiekty różnych typów biorąc wartość (0,0) jako środek. Dzięki temu po oknie można poruszać się jak po układzie współrzędnych.

Iteratory są bardzo często stosowane do poruszania się po kontenerach. W związku z przechowywaniem w kontenerach obiektów na planszy gry, iteratory są wykorzystywane do sprawdzania kolizji dla każdego obiektu i wywołania dla niego odpowiedniej akcji.

```
void Game::Check_bullet_collisons(std::vector<Object*> & bullets)
{

    for (auto it = bullets.begin(); it < bullets.end();)
    {

        if (Check_if_bullet_is_not_on_map(*it) ||
            Check_if_bullet_destroys_entity(*it) ||
            Check_if_bullet_collides_with_block(*it))
        {
            delete *it;
            it = bullets.erase(it);
        }
        else {

            ++it;
        }
```

Rysunek 8 Przykład użycia iteratorów do sprawdzania kolizji dla każdej kuli

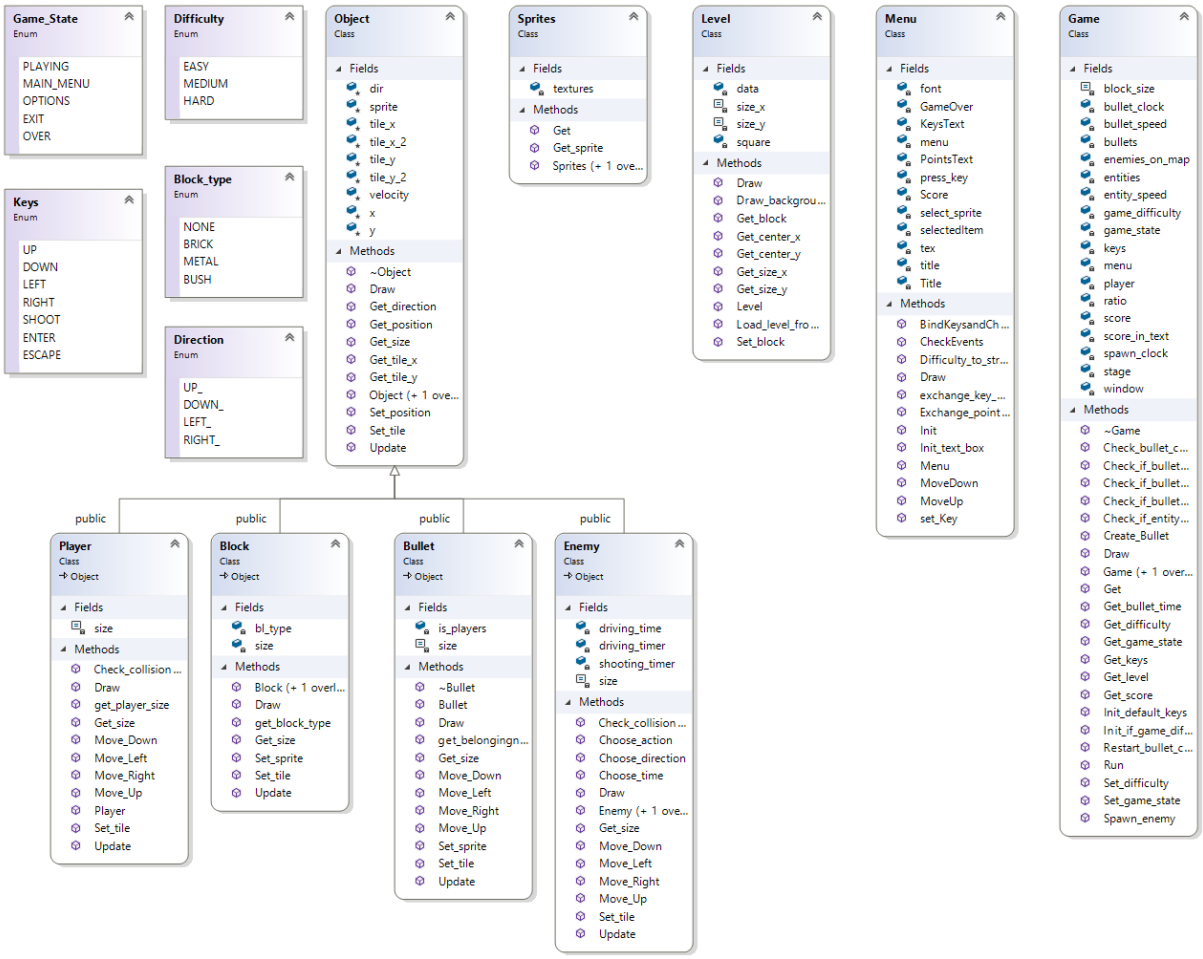
Ostatnim tematem wykorzystanym w projekcie jest użycie kontenerów STL. W projekcie użyte są do przechowywania głownie jednostek co widać na poprzednich zdjęciach. Wykorzystana jest również mapa do przechowywania tekstur gry, aby przy każdym inicjowaniu jednostki np. gracza albo przeciwnika nie trzeba było odczytywać plików.

```
///Klasa przechowująca tekstury (singleton)
class Sprites
{
    std::map<std::string, sf::Texture> textures; ///mapa tekstur

public:

    /** Metoda pozwala uzyskać odpowiedniego sprite'a z tekstury
    @param name klucz pod którym jest tekstura
    @param x który wycinek tekstury
    @return zwraca sprite'a
    */
    sf::Sprite Get_sprite(const std::string & name, sf::IntRect x);

    Sprites();
    Sprites(const Sprites & o) = delete;
    static Sprites & Get() { static Sprites sprites; return sprites; };


};
```

Rysunek 9 Deklaracja klasy przechowującej tekstury i umożliwiająca odczytanie konkretnego kawałka z tekstury

4.3. Diagram klas

W przypadku gdyby diagram klas był mało czytelny na githubie w folderze z projektem znajduje się plik .jpg.



5. Testowanie

Program został przetestowany pod kątem wycieków pamięci, odpowiedniego działania, niepożądanych działań. W trakcie testów nie wykryto żadnych niepowodzeń. Pod kątem działania gry była często testowana w trakcie pisania. Do sprawdzenia czy nie ma wycieków pamięci została użyta biblioteka vld.

6. Wnioski

Program umożliwił przećwiczenie i utrwalenie zagadnień związanych z realizacją laboratoriów. Dzięki projektowi nauczyłem się w jaki sposób stosować bibliotekę SFML. Zrobienie projektu pokazało, jak ważna jest architektura projektu oraz jasny i przejrzysty kod, aby się nie zgubić w trakcie pisania programu.

Załącznik

Tanks

Wygenerowano przez Doxygen 1.8.14

Spis treści

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

Rozdział 2

Indeks klas

2.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 3

Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Rozdział 4

Dokumentacja klas

4.1 Dokumentacja klasy Block

`#include <Block.h>`

Diagram dziedziczenia dla Block

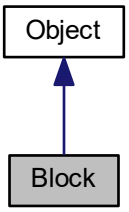
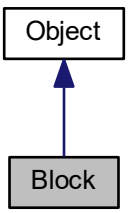


Diagram współpracy dla Block:



Metody publiczne

- Block ()
- Block (double x, double y)
- int Get_size ()
- void Set_tile ()
- void Draw (sf::RenderWindow *&window)
- void Set_sprite (Block_type type)
- void Update (sf::Event &ev, double dt)
- Block_type get_block_type ()

Dodatkowe Dziedziczone Składowe

4.1.1 Opis szczegółowy

Klasa reprezentujaca bloczek na planszy

4.1.2 Dokumentacja konstruktora i destruktora

4.1.2.1 Block() [1/2]

```
Block::Block ( )
```

4.1.2.2 Block() [2/2]

```
Block::Block (
            double x,
            double y )
```

Konstruktor wieloargumentowy, ustawia pozycje bloczka oraz domyslnie ustawia predkosc bloczka na 0 i kierunek w gore.

Parametry

x	pozycja x bloczka
y	pozycja y bloczka

4.1.3 Dokumentacja funkcji składowych

4.1.3.1 Draw()

```
void Block::Draw (
            sf::RenderWindow *& window )  [virtual]
```

Metoda rysuje bloczek w oknie

Parametry

window	wskaznik na obiekt okna

Implementuje Object.

4.1.3.2 get_block_type()

```
Block_type Block::get_block_type ( )  [inline]
```

Metoda zwraca rodzaj bloczka.

Zwraca

zwraca rodzaj bloczka

4.1.3.3 Get_size()

```
int Block::Get_size ( )  [virtual]
```

Getter do sprawdzania rozmiaru bloczka.

Zwraca

rozmiar bloczka

Implementuje Object.

4.1.3.4 Set_sprite()

```
void Block::Set_sprite (
            Block_type type )
```

Metoda na podstawie typu bloczka ustawia jego rodzaj i sprite'a

Parametry

type	rodzaj bloczka

4.1.3.5 Set_tile()

```
void Block::Set_tile ( )  [virtual]
```

Ustawia na jakim fragmencie planszy znajduje sie bloczek.

Implementuje Object.

4.1.3.6 Update()

```
void Block::Update (
            sf::Event & ev,
            double dt )  [virtual]
```

Funkcja przesuwa bloczek.

Parametry

ev	obiekt przechowujacy informacje o zdarzeniach
dt	czas miedzy klatkami (frametime)

Implementuje Object.

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Block.h
- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Block.cpp

4.2 Dokumentacja klasy Bullet

```
#include <Bullet.h>
```

Diagram dziedziczenia dla Bullet

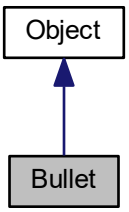
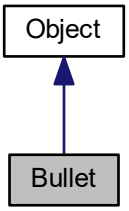


Diagram współpracy dla Bullet:



Metody publiczne

- ~Bullet ()=default
- Bullet (double x, double y, double speed, Direction dir, bool whose)
- int Get_size ()
- void Set_tile ()
- void Update (sf::Event &ev, double dt)
- void Set_sprite (Direction d)
- void Draw (sf::RenderWindow *&window)
- void Move_Up (double dt)
- void Move_Down (double dt)
- void Move_Left (double dt)
- void Move_Right (double dt)
- bool get_belongingness ()

Dodatkowe Dziedziczone Składowe

4.2.1 Opis szczegółowy

Klasa reprezentujaca obiekt kuli w grze

4.2.2 Dokumentacja konstruktora i destruktora

4.2.2.1 ~Bullet()

```
Bullet::~Bullet ( )  [default]
```

4.2.2.2 Bullet()

```
Bullet::Bullet (
            double x,
            double y,
            double speed,
            Direction dir,
            bool whose )
```

Konstruktor wieloargumentowy inicjujacy pozycje, predkosc, kierunek kuli oraz okresla czy jest przeciwnika czy gracza.

Parametry

x	pozycja x kuli
y	pozycja y kuli
speed	predkosc kuli
dir	zwrot kuli
whose	czyja jest kula

4.2.3 Dokumentacja funkcji składowych

4.2.3.1 Draw()

```
void Bullet::Draw (
            sf::RenderWindow *& window )  [virtual]
```

Metoda rysuje obiekt kuli w oknie programu.

Parametry

window	wskaznik na obiekt okna

Implementuje Object.

4.2.3.2 get_belongingness()

```
bool Bullet::get_belongingness ( )
```

Metoda zwraca przynaleznosc kuli.

Zwraca

Zwraca 0 kiedy kula jest przeciwnika, w przeciwnym przypadku zwraca 1.

4.2.3.3 Get_size()

```
int Bullet::Get_size ( )  [virtual]
```

Metoda zwraca rozmiar kuli.

Zwraca

zwraca rozmiar kuli

Implementuje Object.

4.2.3.4 Move_Down()

```
void Bullet::Move_Down (
            double dt )
```

Metoda przesuwa kule w dol.

Parametry

dt	czas miedzy klatkami (frametime)

4.2.3.5 Move_Left()

```
void Bullet::Move_Left (
            double dt )
```

Metoda przesuwa kule w lewo.

Parametry

dt	czas miedzy klatkami (frametime)

4.2.3.6 Move_Right()

```
void Bullet::Move_Right (
            double dt )
```

Metoda przesuwa kule w prawo.

Parametry

dt	czas miedzy klatkami (frametime)

4.2.3.7 Move_Up()

```
void Bullet::Move_Up (
            double dt )
```

Metoda przesuwa kule w gore.

Parametry

dt	czas miedzy klatkami (frametime)

4.2.3.8 Set_sprite()

```
void Bullet::Set_sprite (
            Direction d )
```

Metoda ustawia odpowiedni sprite dla kuli w zaleznosci od zwrotu w ktorym sie porusza.

Parametry

d	zwrot kuli

4.2.3.9 Set_tile()

```
void Bullet::Set_tile ( )  [virtual]
```

Metoda ustawia na jakim bloczku planszy znajduje sie kula.

Implementuje Object.

4.2.3.10 Update()

```
void Bullet::Update (
            sf::Event & ev,
            double dt )  [virtual]
```

Metoda przemieszcza kule w odpowiednim kierunku oraz ustawia na ktorym z bloczkow planszy aktualnie sie znajduje.

Parametry

ev	obiekt przechowujacy informacje o zdarzeniach
dt	czas miedzy klatkami (frametime)

Implementuje Object.

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Bullet.h
- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Bullet.cpp

4.3 Dokumentacja klasy Enemy

```
#include <Enemy.h>
```

Diagram dziedziczenia dla Enemy

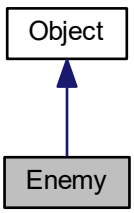
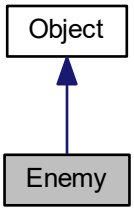


Diagram współpracy dla Enemy:



Metody publiczne

- Enemy ()=default
- Enemy (double x, double y, double speed, Direction dir)
- int Get_size ()
- void Set_tile ()
- void Draw (sf::RenderWindow ∗&window)
- void Update (sf::Event &ev, double dt)
- void Move_Up (const double &dt)
- void Move_Down (const double &dt)
- void Move_Left (const double &dt)
- void Move_Right (const double &dt)
- void Choose_direction ()
- float Choose_time (float x, float y)
- void Choose_action ()
- bool Check_collision_with_tiles (int Tile_x, int Tile_y) const

Dodatkowe Dziedziczone Składowe

4.3.1 Opis szczegółowy

Klasa Przeciwnik

4.3.2 Dokumentacja konstruktora i destruktora

4.3.2.1 Enemy() [1/2]

```
Enemy::Enemy ( )  [default]
```

4.3.2.2 Enemy() [2/2]

```
Enemy::Enemy (
            double x,
            double y,
            double speed,
            Direction dir )
```

Konstruktor wieloargumentowy inicjujacy pozycje, predkosc oraz kierunek przeciwnika

Parametry

x	pozycja x przeciwnika
y	pozycja y przeciwnika
speed	predkosc przeciwnika
dir	zwrot przeciwnika

4.3.3 Dokumentacja funkcji składowych

4.3.3.1 Check_collision_with_tiles()

```
bool Enemy::Check_collision_with_tiles (
            int Tile_x,
            int Tile_y ) const
```

Metoda sprawdza czy przeciwnik nie koliduje z odpowiednimi typami bloczkow

Parametry

Tile_x	pozycja bloczka w rzedzie
Tile_y	pozycja bloczka w kolumnie

Zwraca

w przypadku kiedy kolizja wystapila zwraca true, w przeciwnym wypadku false

4.3.3.2 Choose_action()

```
void Enemy::Choose_action ( )
```

Metoda co pewien czas ustawia kierunek w ktorym sie porusza przeciwnik oraz co pewnien czas aktywuje przeciwnikowi strzal

4.3.3.3 Choose_direction()

```
void Enemy::Choose_direction ( )
```

Metoda wybiera w ktora strone ma poruszac sie przeciwnik

4.3.3.4 Choose_time()

```
float Enemy::Choose_time (
            float x,
            float y )
```

Metoda losuje czas z danego przedzialu.

Parametry

x	lewa granica przedzialu
y	prawa granica przedzialu

Zwraca

zwraca wylosowany czas

4.3.3.5 Draw()

```
void Enemy::Draw (
            sf::RenderWindow *& window )  [virtual]
```

Funkcja rysuje sprite'a przeciwnika w oknie programu

Parametry

window	wskaznik na obiekt okna

Implementuje Object.

4.3.3.6 Get_size()

```
int Enemy::Get_size ( )  [virtual]
```

Metoda zwraca rozmiar przeciwnika.

Zwraca

zwraca rozmiar przeciwnika

Implementuje Object.

4.3.3.7 Move_Down()

```
void Enemy::Move_Down (
            const double & dt )
```

Metoda przemieszcza przeciwnika w dol, ustawia mu odpowiedni zwrot i skorelowany z nim sprite oraz sprawdza kolizje z bloczkami na dole.

Parametry

dt	czas miedzy klatkami (frametime)

4.3.3.8 Move_Left()

```
void Enemy::Move_Left (
            const double & dt )
```

Metoda przemieszcza przeciwnika w lewo, ustawia mu odpowiedni zwrot i skorelowany z nim sprite oraz sprawdza kolizje z bloczkami po lewej stronie.

Parametry

dt	czas miedzy klatkami (frametime)

4.3.3.9 Move_Right()

```
void Enemy::Move_Right (
            const double & dt )
```

Metoda przemieszcza przeciwnika w prawo, ustawia mu odpowiedni zwrot i skorelowany z nim sprite oraz sprawdza kolizje z bloczkami po prawej stronie.

Parametry

dt	czas miedzy klatkami (frametime)

4.3.3.10 Move_Up()

```
void Enemy::Move_Up (
            const double & dt )
```

Metoda przemieszcza przeciwnika w gore, ustawia mu odpowiedni zwrot i skorelowany z nim sprite oraz sprawdza kolizje z bloczkami u gory.

Parametry

dt	czas miedzy klatkami (frametime)

4.3.3.11 Set_tile()

```
void Enemy::Set_tile ( )  [virtual]
```

Metoda ustawia na jakich bloczkach przeciwnik sie znajduje na podstawie kierunku w ktorym sie porusza

Implementuje Object.

4.3.3.12 Update()

```
void Enemy::Update (
            sf::Event & ev,
            double dt )  [virtual]
```

Metoda przesuwa przeciwnika zgodnie z jego ustawionym kierunkiem i spradza czy nie znalazl sie poza mapa.

Implementuje Object.

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Enemy.h
- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Enemy.cpp

4.4 Dokumentacja klasy Game

```
#include <Game.h>
```

Metody publiczne

- Game ()
- ~Game ()
- Game (const Game &o)=delete
- void Set_game_state (Game_State state)
- Game_State Get_game_state ()
- void Set_difficulty (Difficulty diff)
- Difficulty Get_difficulty ()
- std::map< Keys, sf::Keyboard::Key > & Get_keys ()
- Level * Get_level ()
- int Get_score ()
- void Create_Bullet (Object *ob)
- void Run ()
- void Draw (sf::RenderWindow *window)
- void Init_default_keys ()
- void Check_bullet_collisons (std::vector< Object *> &bullets)
- bool Check_if_bullet_is_not_on_map (const Object *bullet) const
- bool Check_if_bullet_collides_with_block (Object *bullet)
- bool Check_if_bullet_destroys_entity (Object *bullet)
- float Get_bullet_time ()
- void Restart_bullet_clock ()
- void Init_if_game_diff_selected ()
- void Spawn_enemy ()
- bool Check_if_entity_is_not_on_map (Object *entity)

Statyczne metody publiczne

- static Game & Get ()

4.4.1 Opis szczegółowy

Klasa przechowujaca wszystkie obiekty w grze i zarzadzajaca nimi (singleton)

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 Game() [1/2]

```
Game::Game ( )
```

Konstruktor domyslny klasy. Inicjuje domyslnymi wartosciami podstawowe wartosci: predkosc jednostek, predkosc kul, rozmiar bloczka, stan gry, poziom trudnosci oraz wynik i ile jest przeciwnikow na mapie. Tworzy nowa plansze i inicjuje gracza oraz klawisze, ktorymi sie porusza.

4.4.2.2 ~Game()

```
Game::~Game ( )
```

Destruktor klasy Game. Usuwa wszystkie obiekty zaalokowane dynamiczne tj. kule, jednostki, plansze itp.

4.4.2.3 Game() [2/2]

```
Game::Game (
            const Game & o )  [delete]
```

4.4.3 Dokumentacja funkcji składowych

4.4.3.1 Check_bullet_collisons()

```
void Game::Check_bullet_collisons (
            std::vector< Object *> & bullets )
```

Metoda dla kazdej kuli w grze sprawdza czy nastapila kolizja z krancem mapy, z innym graczem lub z bloczkiem na planszy. W przypadku, gdy nastapi kolizja kuli z jednostka na mapie to w zaleznosci od przynaleznosci kuli niszczy kule i przeciwnika lub niszczy kule i gracza. Sprawdza rowniez kolizje z bloczkami.

Parametry

bullets	wektor kul znajdujacych sie w grze.

4.4.3.2 Check_if_bullet_collides_with_block()

```
bool Game::Check_if_bullet_collides_with_block (
            Object * bullet )
```

Sprawdza kolizje kuli z bloczkiem na ktorym znajduje sie aktualnie kula oraz w zaleznosci od typu bloczka ustawia mu teksture.

Zwraca

w przypadku kiedy kula nie niszczy bloczka i moze przez niego przeleciec zwraca false,

4.4.3.3 Check_if_bullet_destroys_entity()

```
bool Game::Check_if_bullet_destroys_entity (
            Object * bullet )
```

Metoda sprawdza czy kula niszczy jednostke na mapie np. gracza lub przeciwnika. Jezeli kula nachodzi na gracza to sprawdza czyja jest kula i na jaka jednostke nachodzi (gracza czy przeciwnika).Jesli kula nalezy do przeciwnika i nachodzi na przeciwnika to nie niszczy go. Jesli kula jest gracza i nachodzi na przeciwnika to niszczy go. Jesli natomiast kula jest przeciwnika i nachodzi na gracza to nastepuje koniec gry i gracz jest niszczony.

Zwraca

zwraca true jesli nastapila kolizja kuli z jakakolwiek jednostkom, a false jesli nie nastapila kolizja kuli z jednostka

4.4.3.4 Check_if_bullet_is_not_on_map()

```
bool Game::Check_if_bullet_is_not_on_map (
            const Object * bullet ) const
```

Metoda sprawdza czy kula nie znajduje sie na mapie.

Zwraca

zwraca true jesli kula jest poza mapa, w przeciwnym wypadku false

4.4.3.5 Check_if_entity_is_not_on_map()

```
bool Game::Check_if_entity_is_not_on_map (
            Object * entity )
```

Spradza czy jednostka nie jest na mapie.

Zwraca

zwraca true jesli jednostka jest poza mapa, w przeciwnym wypadku zwraca false.

4.4.3.6 Create_Bullet()

```
void Game::Create_Bullet (
            Object * ob )
```

Metoda tworzy nowa kule w grze na pozycji danej jednostki i zgodnie z jej kierunkiem oraz przydziela jej przynaleznosc w zaleznosci czy wystrzelil ja przeciwnik czy gracz.

Parametry

ob	wskaznik na obiekt, ktory tworzy kule

4.4.3.7 Draw()

```
void Game::Draw (
            sf::RenderWindow * window )
```

Metoda odpowiada za rysowanie obiektow w grze na ekran w zaleznosci od stanu gry. Odswieza ona ekran oraz rysuje i wyswietla elementy gry na ekranie.

Parametry

window	wskaznik na obiekt okna

4.4.3.8 Get()

```
static Game& Game::Get ( )  [inline], [static]
```

Metoda zwraca instancje obiektu gry.

Zwraca

obiekt gry.

4.4.3.9 Get_bullet_time()

```
float Game::Get_bullet_time ( )
```

Metoda zwraca jak dlugo porusza sie ostatnia wystrzelona kula przez gracza.

Zwraca

czas od ostatniej stworzonej kuli przez gracza.

4.4.3.10 Get_difficulty()

```
Difficulty Game::Get_difficulty ( )  [inline]
```

Metoda zwraca aktualnie ustawiony poziom trudnosci w grze.

Zwraca

poziom trudnosci

4.4.3.11 Get_game_state()

```
Game_State Game::Get_game_state ( )  [inline]
```

Metoda zwraca aktualny stan gry np. czy jestesmy w Menu czy toczy sie rozgrywka.

Zwraca

zwraca stan gry

4.4.3.12 Get_keys()

```
std::map<Keys, sf::Keyboard::Key>& Game::Get_keys ( )  [inline]
```

Metoda zwraca mape z klawiszami, ktorych uzytkownik uzywa do poruszania sie.

Zwraca

mapa z kodami klawiszy

4.4.3.13 Get_level()

```
Level* Game::Get_level ( )  [inline]
```

Metoda zwraca aktualny wybrany poziom w grze.

Zwraca

zwraca wskaznik na poziom gry.

4.4.3.14 Get_score()

```
int Game::Get_score ( )  [inline]
```

Metoda zwraca aktualny wynik gracza.

Zwraca

aktualny wynik uzyskany przez gracza.

4.4.3.15 Init_default_keys()

```
void Game::Init_default_keys ( )
```

Metoda inicjuje domyslne klawisze do obslugi wejscia w grze.

4.4.3.16 Init_if_game_diff_selected()

```
void Game::Init_if_game_diff_selected ( )
```

Metoda inicjuje szybkosc poruszania sie kul oraz mnoznik w zaleznosci od wybranego poziomu trudnosci.

4.4.3.17 Restart_bullet_clock()

```
void Game::Restart_bullet_clock ( )
```

Metoda resetuje zegar ostatnio wystrzelonej kuli.

4.4.3.18 Run()

```
void Game::Run ( )
```

Metoda "uruchamia" gre. Odpowiada ona za rysowania wszystkich obiektow na ekran w zaleznosci od stanu gry, obliczanie czasu miedzy klatkami oraz sprawdzaniu wydarzen i aktualizowaniu pozycji wszystkich obiektow na planszy.

4.4.3.19 Set_difficulty()

```
void Game::Set_difficulty (
            Difficulty diff )  [inline]
```

Metoda ustawia poziom trudnosci w zaleznosci od parametru.

Parametry

diff	poziom trudnosci jaki ma zostac ustawiony

4.4.3.20 Set_game_state()

```
void Game::Set_game_state (
            Game_State state )  [inline]
```

Metoda do ustawiania stanu gry.

Parametry

state	na jaki stan ma zostac zmieniona aktualnie gra

4.4.3.21 Spawn_enemy()

```
void Game::Spawn_enemy ( )
```

Tworzy gracza na planszy, jesli ich jest mniej niz 3 i w okreslonych odstepach czasu.

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Game.h
- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Game.cpp

4.5 Dokumentacja klasy Level

```
#include <Level.h>
```

Metody publiczne

- Level (sf::RenderWindow *&window)
- void Load_level_from_file (const std::string &filename)
- void Draw (sf::RenderWindow *&window)
- void Draw_background (sf::RenderWindow *&window)
- float Get_center_x ()
- float Get_center_y ()
- const float Get_size_x () const
- const float Get_size_y () const
- Block Get_block (int x, int y)
- void Set_block (int x, int y, Block_type type)

4.5.1 Opis szczegółowy

Klasa odpowiadajaca za plansze w grze

4.5.2 Dokumentacja konstruktora i destruktora

4.5.2.1 Level()

```
Level::Level (
            sf::RenderWindow *& window )
```

Konstruktor klasy Level. Tworzy tlo planszy oraz pozycjonuje ja na ekranie

Parametry

window	wskaznik na obiekt okna

4.5.3 Dokumentacja funkcji składowych

4.5.3.1 Draw()

```
void Level::Draw (
            sf::RenderWindow *& window )
```

Metoda rysujaca wszystkie bloczki planszy na ekranie

Parametry

window	wskaznik na obiekt okna

4.5.3.2 Draw_background()

```
void Level::Draw_background (
            sf::RenderWindow *& window )
```

Metoda rysujaca tlo planszy na ekran.

Parametry

window	wskaznik na obiekt okna

4.5.3.3 Get_block()

```
Block Level::Get_block (
            int x,
            int y )
```

Metoda zwracajaca bloczek na danym fragmencie planszy

Parametry

x	numer wiersza kratki na planszy
y	numer kolumny kratki na planszy

Zwraca

Obiekt bloczka

4.5.3.4 Get_center_x()

```
float Level::Get_center_x ( )
```

Metoda zwracajaca pozycje x srodka planszy

Zwraca

pozycja x srodka planszy

4.5.3.5 Get_center_y()

```
float Level::Get_center_y ( )
```

Metoda zwracajaca pozycje y srodka planszy

Zwraca

pozycja y srodka planszy

4.5.3.6 Get_size_x()

```
const float Level::Get_size_x ( ) const
```

Metoda zwracajaca szerokosc (x) planszy

Zwraca

szerokosc planszy w bloczkach

4.5.3.7 Get_size_y()

```
const float Level::Get_size_y ( ) const
```

Metoda zwracajaca wysokosc (y) planszy

Zwraca

wysokosc planszy w bloczkach

4.5.3.8 Load_level_from_file()

```
void Level::Load_level_from_file (
            const std::string & filename )
```

Metoda wczytujaca plansze z pliku .csv

Parametry

filename	nazwa pliku

4.5.3.9 Set_block()

```
void Level::Set_block (
            int x,
            int y,
            Block_type type )
```

Metoda ustawia odpowiedni sprite i rodzaj bloczka w danej kolumnie i wierszy planszy

Parametry

x	pozycja bloczka w rzedzie
y	pozycja bloczka w kolumnie
type	typ na jaki zostanie ustawiony bloczek

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Level.h
- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Level.cpp

4.6 Dokumentacja klasy Menu

`#include <Menu.h>`

Metody publiczne

- Menu (sf::RenderWindow ∗&window)
- void Init_text_box (sf::Text &text_box)
- void MoveUp ()
- void MoveDown ()
- void CheckEvents (sf::Event ev, sf::RenderWindow ∗&window, std::map< Keys, sf::Keyboard::Key > &keys)
- void Draw (sf::RenderWindow ∗&window)
- void Init ()
- int set_Key (std::map< Keys, sf::Keyboard::Key > &keys, sf::RenderWindow ∗&window, enum Keys id)
- std::string exchange_key_code_to_string (int KeyCode)
- void BindKeysandChangeDifficulty (std::map< Keys, sf::Keyboard::Key > &keys, sf::RenderWindow ∗&window)
- std::string Difficulty_to_string (Difficulty diff)
- void Exchange_points_to_text ()

4.6.1 Opis szczegółowy

Klasa Menu odpowiadajaca za wyswietlanie informacji i tworzenie interfejsu wizualnego z uzytkownikiem

4.6.2 Dokumentacja konstruktora i destruktora

4.6.2.1 Menu()

```
Menu::Menu (
            sf::RenderWindow *& window )
```

Konstruktor klasy Menu. Inicjuje wszystkie text boxy, sprite'y i ich pozycje w Menu

Parametry

window	wskaznik na obiekt okna

4.6.3 Dokumentacja funkcji składowych

4.6.3.1 BindKeysandChangeDifficulty()

```
void Menu::BindKeysandChangeDifficulty (
            std::map< Keys, sf::Keyboard::Key > & keys,
            sf::RenderWindow *& window )
```

Funkcja ustawia poziom trudnosci oraz pozwala na ustawianie klawiszy w opcjach gry

Parametry

keys	mapa przyciskow
window	wskaznik na okno aplikacji

4.6.3.2 CheckEvents()

```
void Menu::CheckEvents (
            sf::Event ev,
            sf::RenderWindow *& window,
            std::map< Keys, sf::Keyboard::Key > & keys )
```

Funkcja sprawdza czy gracz wcisnal klawisz i wywoluje odpowiednia akcje w Menu gry

Parametry

ev	obiekt przechowujacy informacje o zdarzeniach
window	wskaznik na okno aplikacji
keys	mapa przyciskow

4.6.3.3 Difficulty_to_string()

```
std::string Menu::Difficulty_to_string (
            Difficulty diff )
```

Funkcja zameinia poziom trudnosci jako klucz na napis

Parametry

diff	poziom trudnosci

Zwraca

Zwraca obiekt std::string z napisem poziomu trudnosci

4.6.3.4 Draw()

```
void Menu::Draw (
            sf::RenderWindow *& window )
```

Funkcja rysuje elementy Menu w zaleznosci od stanu w oknie

Parametry

window	wskaznik na obiekt okna

4.6.3.5 exchange_key_code_to_string()

```
std::string Menu::exchange_key_code_to_string (
            int KeyCode )
```

Funkcja zamienia kod klaiwsza na odpowiednia litere

Parametry

KeyCode	kod klawisza

Zwraca

Zwraca obiekt std::string z nazwa klawisza z klawiatury

4.6.3.6 Exchange_points_to_text()

```
void Menu::Exchange_points_to_text ( )
```

Zamienia punkty na tekst, aby mozna bylo je wyswietlic na ekran.

4.6.3.7 Init()

```
void Menu::Init ( )
```

Funkcja odczytuje odpowiednie zasoby z plikow i inicjuje sprite'y widoczne w Menu

4.6.3.8 Init_text_box()

```
void Menu::Init_text_box (
            sf::Text & text_box )
```

Metoda inicjuje ramke z tekstem

Parametry

text_box	ramka tekstu

4.6.3.9 MoveDown()

```
void Menu::MoveDown ( )
```

Funkcja przesuwa pozycje wyboru w dol

4.6.3.10 MoveUp()

```
void Menu::MoveUp ( )
```

Funkcja przesuwa pozycje wyboru w menu gore

4.6.3.11 set_Key()

```
int Menu::set_Key (
            std::map< Keys, sf::Keyboard::Key > & keys,
            sf::RenderWindow *& window,
            enum Keys id )
```

Funkcja przypisuje kod wcisnietego klawisza do mapy klawiszy. Czeka ona na wcisniecie klawisza przez gracza

Parametry

keys	mapa przyciskow
window	wskaznik na okno aplikacji
id	klucz okreslajacy jaki klawisz ustawiamy

Zwraca

Zwraca kod wcisnietego klawisza

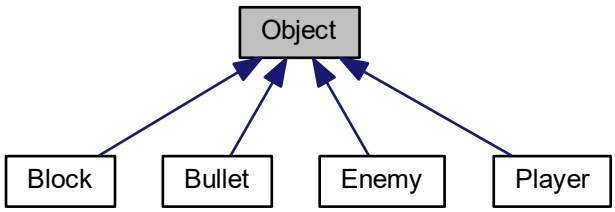
Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Menu.h
- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Menu.cpp

4.7 Dokumentacja klasy Object

`#include <Object.h>`

Diagram dziedziczenia dla Object



Metody publiczne

- Object ()
- virtual ~Object ()=default
- Object (double x, double y, double speed, Direction d)
- virtual int Get_size ()=0
- virtual void Set_tile ()=0
- virtual void Draw (sf::RenderWindow *&window)=0
- virtual void Update (sf::Event &ev, double dt)=0
- Direction Get_direction ()
- sf::Vector2f Get_position () const
- void Set_position (double X, double Y)
- int Get_tile_x ()
- int Get_tile_y ()

Atrybuty chronione

- sf::Sprite sprite
- Direction dir
- double x
- double y
- double velocity
- int tile_x
- int tile_y
- int tile_x_2
- int tile_y_2

4.7.1 Opis szczegółowy

Klasa reprezentuje obiekt gry

4.7.2 Dokumentacja konstruktora i destruktora

4.7.2.1 Object() [1/2]

```
Object::Object ( )
```

4.7.2.2 ~Object()

```
virtual Object::~Object ( )  [virtual], [default]
```

4.7.2.3 Object() [2/2]

```
Object::Object (
            double x,
            double y,
            double speed,
            Direction d )
```

Konstruktor wieloargumentowy inicjujacy pozycje, predkosc oraz kierunek obiektu

Parametry

x	pozycja x obiektu
y	pozycja y obiektu
speed	predkosc obiektu
d	zwrot obiektu

4.7.3 Dokumentacja funkcji składowych

4.7.3.1 Draw()

```
virtual void Object::Draw (
            sf::RenderWindow *& window )  [pure virtual]
```

Metoda rysuje obiekt w oknie

Parametry

window	wskaznik na obiekt okna

Implementowany w Bullet, Enemy, Block i Player.

4.7.3.2 Get_direction()

```
Direction Object::Get_direction ( )
```

Getter umozliwiajacy sprawdzenie zwrotu obiektu gry

Zwraca

zwraca kierunek obiektu gry

4.7.3.3 Get_position()

```
sf::Vector2f Object::Get_position ( ) const
```

Metoda zwraca wektor z pozycja obiektu

Zwraca

zwraca dwuelementowy wektor z pozycja

4.7.3.4 Get_size()

```
virtual int Object::Get_size ( )  [pure virtual]
```

Interfejs do sprawdzania rozmiaru obiektu

Implementowany w Bullet, Enemy, Block i Player.

4.7.3.5 Get_tile_x()

```
int Object::Get_tile_x ( )
```

Getter do sprawdzania na ktorym bloczku aktualnie znajduje sie obiekt

Zwraca

zwraca w ktorym rzedzie znajduje sie obiekt

4.7.3.6 Get_tile_y()

```
int Object::Get_tile_y ( )
```

Getter do sprawdzania na ktorym bloczku aktualnie znajduje sie obiekt

Zwraca

zwraca w ktorej kolumnie znajduje sie obiekt

4.7.3.7 Set_position()

```
void Object::Set_position (
            double X,
            double Y )
```

Metoda ustawia pozycje gracza

Parametry

X	pozycja x
Y	pozycja Y

4.7.3.8 Set_tile()

```
virtual void Object::Set_tile ( )  [pure virtual]
```

Interfejs do ustawiania na jakim bloczku (bloczkach) znajduje sie obiekt gry

Implementowany w Bullet, Enemy, Block i Player.

4.7.3.9 Update()

```
virtual void Object::Update (
            sf::Event & ev,
            double dt )  [pure virtual]
```

Metoda aktualizuje pozycje obiektu gry

Parametry

ev	obiekt przechowujacy informacje o zdarzeniach
dt	czas miedzy klatkami (frametime)

Implementowany w Block, Enemy, Player i Bullet.

4.7.4 Dokumentacja atrybutów składowych

4.7.4.1 dir

```
Direction Object::dir  [protected]
```

zwrot obiektu gry

4.7.4.2 sprite

```
sf::Sprite Object::sprite  [protected]
```

sprite obiektu gry

4.7.4.3 tile_x

```
int Object::tile_x  [protected]
```

Fragment planszy na ktorym aktualnie znajduje sie obiekt gry

4.7.4.4 tile_x_2

```
int Object::tile_x_2  [protected]
```

4.7.4.5 tile_y

```
int Object::tile_y  [protected]
```

4.7.4.6 tile_y_2

```
int Object::tile_y_2  [protected]
```

4.7.4.7 velocity

```
double Object::velocity  [protected]
```

predkosc obiektu gry

4.7.4.8 x

```
double Object::x  [protected]
```

pozycja gracza

4.7.4.9 y

```
double Object::y  [protected]
```

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Object.h
- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Object.cpp

4.8 Dokumentacja klasy Player

```
#include <Player.h>
```

Diagram dziedziczenia dla Player

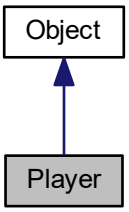
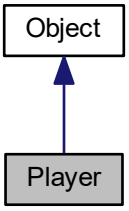


Diagram współpracy dla Player:



Metody publiczne

- Player (double x, double y, double speed, Direction dir)
- int Get_size ()
- void Set_tile ()
- void Draw (sf::RenderWindow ∗&window)
- void Update (sf::Event &ev, double dt)
- void Move_Up (const double &dt)
- void Move_Down (const double &dt)
- void Move_Left (const double &dt)
- void Move_Right (const double &dt)
- const int get_player_size () const
- bool Check_collision_on_tiles (int Tile_x, int Tile_y)

Dodatkowe Dziedziczone Składowe

4.8.1 Opis szczegółowy

Klasa reprezentujaca gracza

4.8.2 Dokumentacja konstruktora i destruktora

4.8.2.1 Player()

```
Player::Player (
            double x,
            double y,
            double speed,
            Direction dir )
```

Konstruktor wieloargumentowy inicjujacy pozycje, predkosc oraz kierunek gracza.

Parametry

x	pozycja x gracza
y	pozycja y gracza
speed	predkosc gracza
dir	zwrot gracza

4.8.3 Dokumentacja funkcji składowych

4.8.3.1 Check_collision_on_tiles()

```
bool Player::Check_collision_on_tiles (
            int Tile_x,
            int Tile_y )
```

Metoda sprawdza czy gracz nie koliduje z odpowiednimi typami bloczkow

Parametry

Tile_x	pozycja bloczka w rzedzie
Tile_y	pozycja bloczka w kolumnie

Zwraca

w przypadku kiedy kolizja wystapila zwraca true, w przeciwnym wypadku false

4.8.3.2 Draw()

```
void Player::Draw (
            sf::RenderWindow *& window )  [virtual]
```

Funkcja rysuje sprite'a gracza w oknie programu.

Parametry

window	wskaznik na obiekt okna

Implementuje Object.

4.8.3.3 get_player_size()

```
const int Player::get_player_size ( ) const
```

Metoda zwraca rozmiar gracza

Zwraca

rozmiar gracza

4.8.3.4 Get_size()

```
int Player::Get_size ( )  [virtual]
```

Metoda zwraca rozmiar gracza.

Zwraca

zwraca rozmiar gracza

Implementuje Object.

4.8.3.5 Move_Down()

```
void Player::Move_Down (
            const double & dt )
```

Metoda przemieszcza gracza w dol, ustawia mu odpowiedni zwrot i skorelowany z nim sprite oraz sprawdza kolizje z bloczkami na dole.

Parametry

dt	czas miedzy klatkami (frametime)

4.8.3.6 Move_Left()

```
void Player::Move_Left (
            const double & dt )
```

Metoda przemieszcza gracza w lewo, ustawia mu odpowiedni zwrot i skorelowany z nim sprite oraz sprawdza kolizje z bloczkami po lewej stronie.

Parametry

dt	czas miedzy klatkami (frametime)

4.8.3.7 Move_Right()

```
void Player::Move_Right (
            const double & dt )
```

Metoda przemieszcza gracza w prawo, ustawia mu odpowiedni zwrot i skorelowany z nim sprite oraz sprawdza kolizje z bloczkami po prawej stronie.

Parametry

dt	czas miedzy klatkami (frametime)

4.8.3.8 Move_Up()

```
void Player::Move_Up (
            const double & dt )
```

Metoda przemieszcza gracza w gore, ustawia mu odpowiedni zwrot i skorelowany z nim sprite oraz sprawdza kolizje z bloczkami u gory.

Parametry

dt	czas miedzy klatkami (frametime)

4.8.3.9 Set_tile()

```
void Player::Set_tile ( )  [virtual]
```

Metoda ustawia na jakich bloczkach planszy znajduje sie gracz na podstawie kierunku w ktorym sie porusza.

Implementuje Object.

4.8.3.10 Update()

```
void Player::Update (
            sf::Event & ev,
            double dt )  [virtual]
```

Metoda sprawdza wcisniete klawisze z klawiatury i wywoluje odpowiednie akcje dla gracza np. przemieszcza go lub tworzy kule. Ustawia rowniez na jakim fragmencie planszy znajduje sie gracz.

Parametry

ev	obiekt przechowujacy informacje o zdarzeniach
dt	czas miedzy klatkami (frametime)

Implementuje Object.

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Player.h
- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Player.cpp

4.9 Dokumentacja klasy Sprites

`#include <Sprites.h>`

Metody publiczne

- sf::Sprite Get_sprite (const std::string &name, sf::IntRect x)
- Sprites ()
- Sprites (const Sprites &o)=delete

Statyczne metody publiczne

- static Sprites & Get ()

4.9.1 Opis szczegółowy

Klasa przechowujaca tekstury (singleton)

4.9.2 Dokumentacja konstruktora i destruktora

4.9.2.1 Sprites() [1/2]

```
Sprites::Sprites ( )
```

4.9.2.2 Sprites() [2/2]

```
Sprites::Sprites (
            const Sprites & o )  [delete]
```

4.9.3 Dokumentacja funkcji składowych

4.9.3.1 Get()

```
static Sprites& Sprites::Get ( )  [inline], [static]
```

4.9.3.2 Get_sprite()

```
sf::Sprite Sprites::Get_sprite (
            const std::string & name,
            sf::IntRect x )
```

Metoda pozwala uzyskac odpowiedniego sprite'a z tekstury

Parametry

name	klucz pod ktorym jest tekstura
x	ktory wycinek tekstury

Zwraca

zwraca sprite'a

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Sprites.h
- C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Sprites.cpp

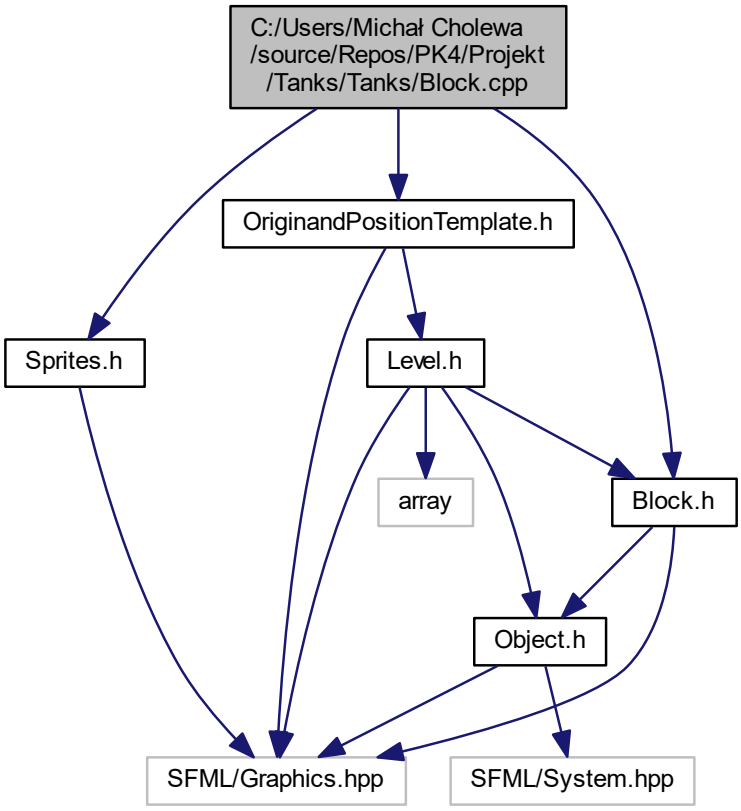
Rozdział 5

Dokumentacja plików

5.1 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/↩ Tanks/Block.cpp

```
#include "Block.h"
#include "Sprites.h"
#include "OriginandPositionTemplate.h"
```

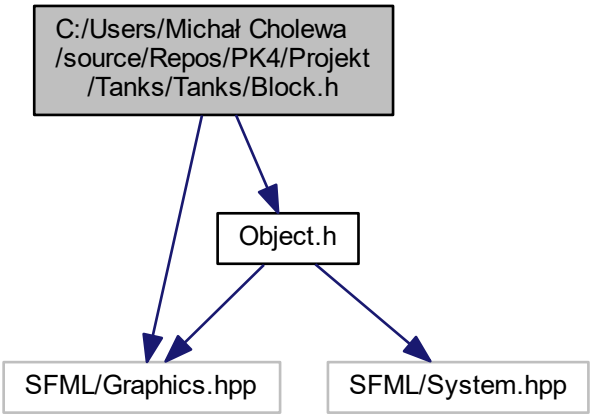
Wykres zależności załączania dla Block.cpp:



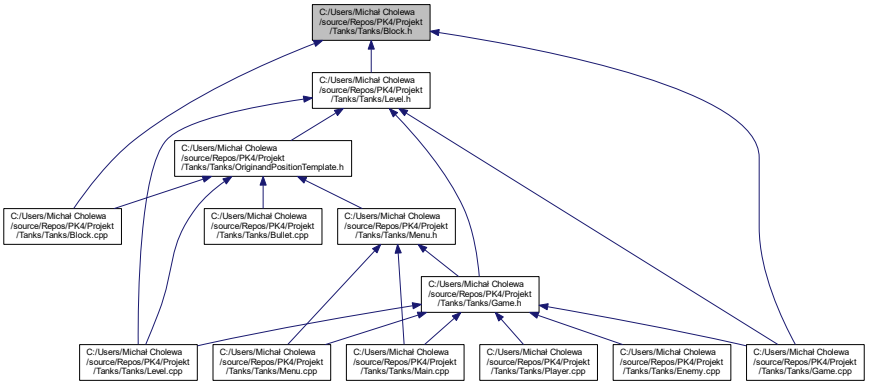
5.2 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Block.h

```
#include "Object.h"
#include "SFML/Graphics.hpp"
```

Wykres zależności załączania dla Block.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Block

Wyliczenia

- enum Block_type { NONE, BRICK, METAL, BUSH }

5.2.1 Dokumentacja typów wyliczanych

5.2.1.1 Block_type

```
enum Block_type
```

typ wyliczeniowy do okreslenia typu bloczka

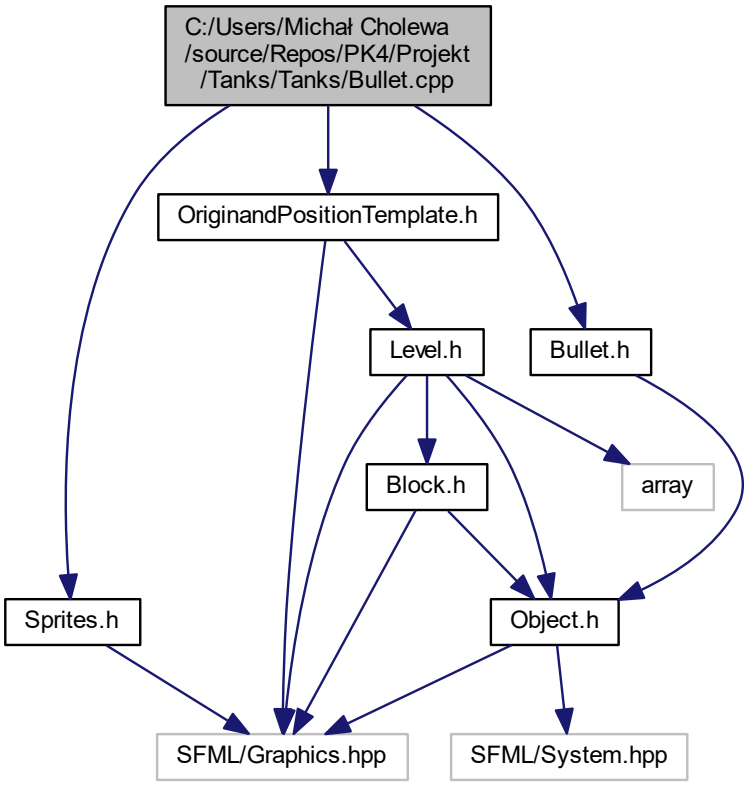
Wartości wyliczeń

NONE	
BRICK	
METAL	
BUSH	

5.3 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/↩ Tanks/Bullet.cpp

```
#include "Bullet.h"
#include "Sprites.h"
#include "OriginandPositionTemplate.h"
```

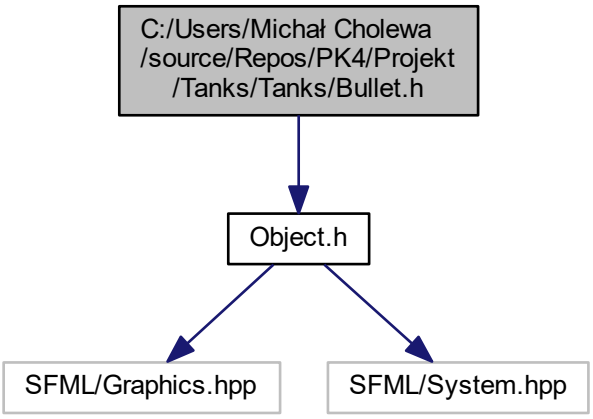
Wykres zależności załączania dla Bullet.cpp:



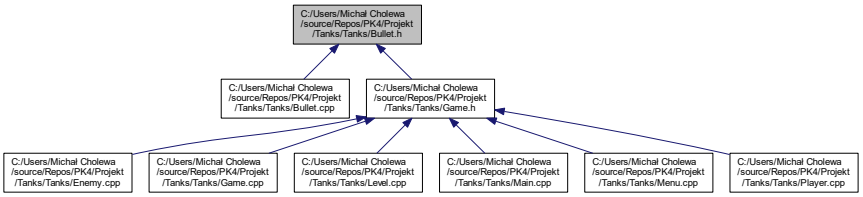
5.4 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Bullet.h

```
#include "Object.h"
```

Wykres zależności załączania dla Bullet.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

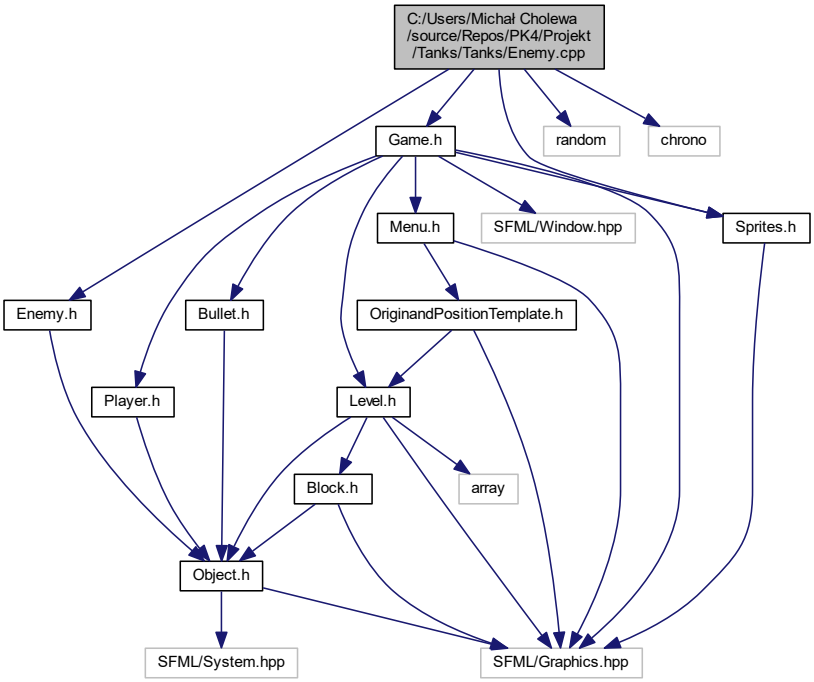
- class Bullet

5.5 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Enemy.cpp

```
#include "Enemy.h"
#include "Sprites.h"
#include "Game.h"
#include <random>
```

```
#include <chrono>
```

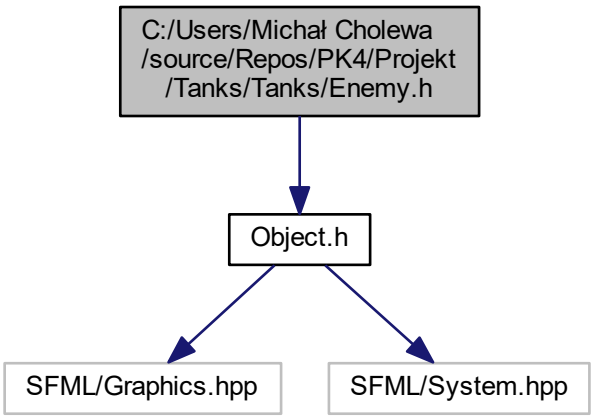
Wykres zależności załączania dla Enemy.cpp:



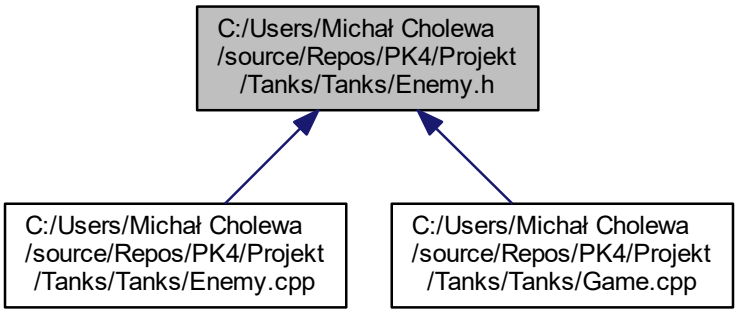
5.6 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Enemy.h

```
#include "Object.h"
```

Wykres zależności załączania dla Enemy.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



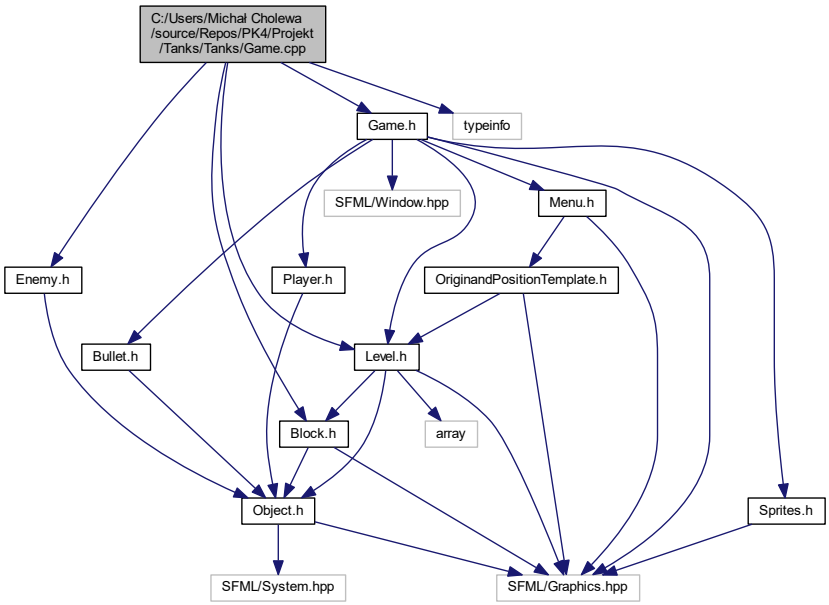
Komponenty

- class Enemy

5.7 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Game.cpp

```
#include "Game.h"
#include "Level.h"
#include "Block.h"
#include "Enemy.h"
#include <typeinfo>
```

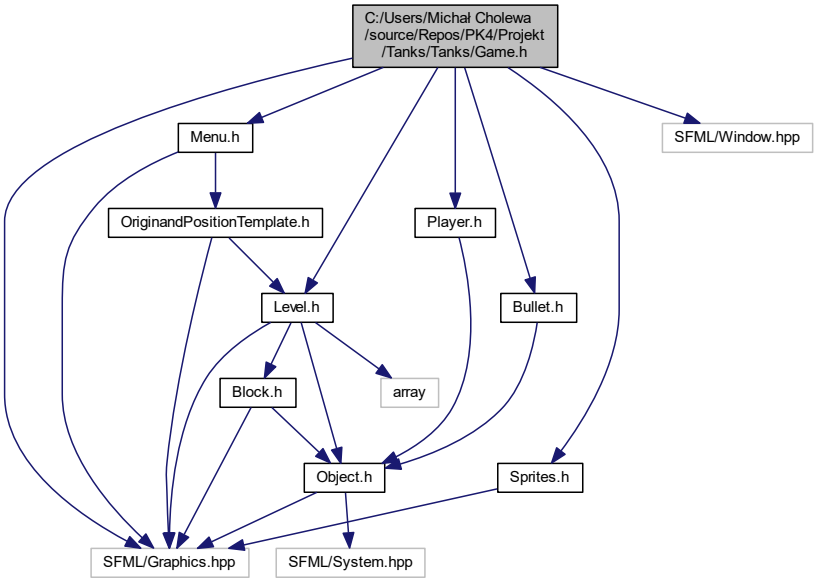
Wykres zależności załączania dla Game.cpp:



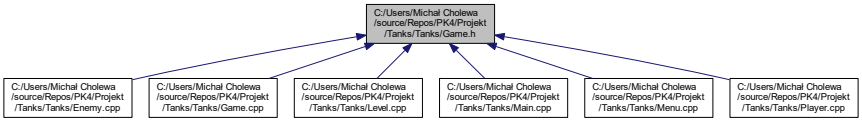
5.8 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/↩ Tanks/Game.h

```
#include "SFML/Graphics.hpp"
#include "SFML/Window.hpp"
#include "Menu.h"
#include "Level.h"
#include "Sprites.h"
#include "Player.h"
#include "Bullet.h"
```

Wykres zależności załączania dla Game.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

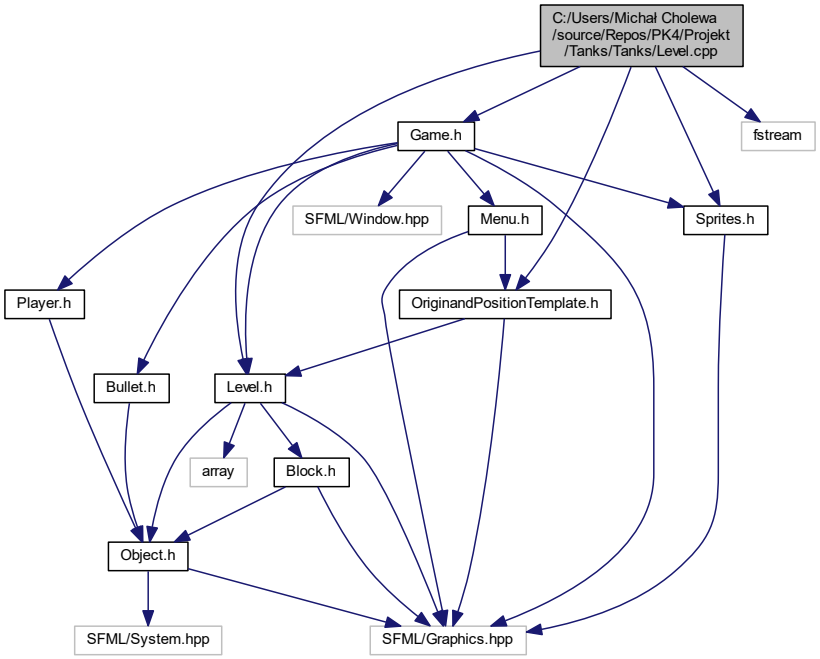
- class Game

5.9 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Level.cpp

```
#include "Level.h"
#include "OriginandPositionTemplate.h"
#include "Game.h"
#include "Sprites.h"
```

```
#include <fstream>
```

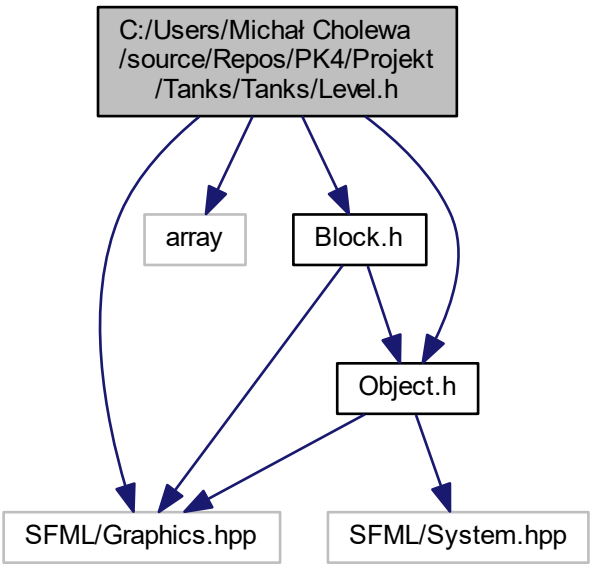
Wykres zależności załączania dla Level.cpp:



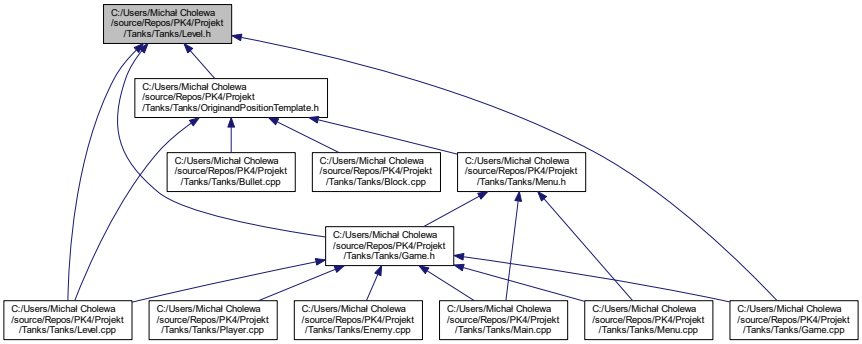
5.10 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/↩ Tanks/Level.h

```
#include "SFML/Graphics.hpp"
#include <array>
#include "Object.h"
#include "Block.h"
```

Wykres zależności załączania dla Level.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

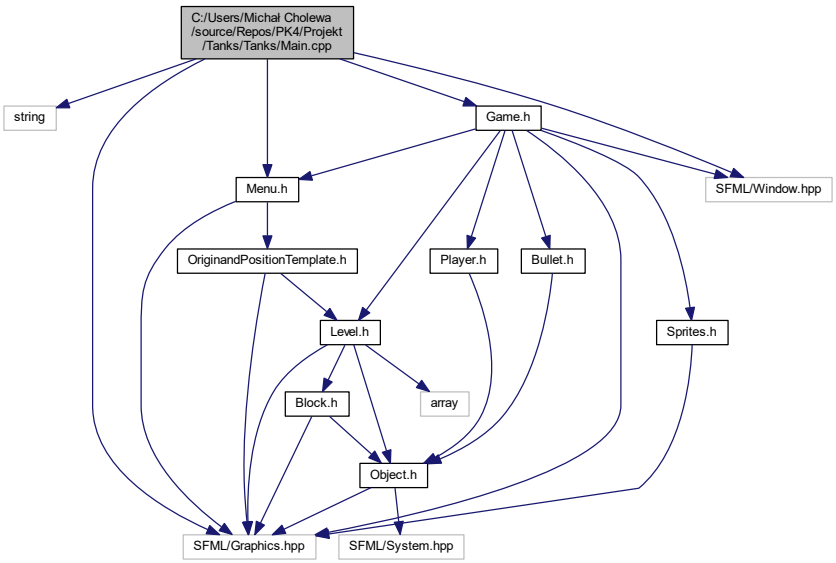
- class Level

5.11 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Main.cpp

```
#include <string>
#include "SFML/Graphics.hpp"
```

```
#include "SFML/Window.hpp"
#include "Menu.h"
#include "Game.h"
```

Wykres zależności załączania dla Main.cpp:



Funkcje

- int main ()

5.11.1 Dokumentacja funkcji

5.11.1.1 main()

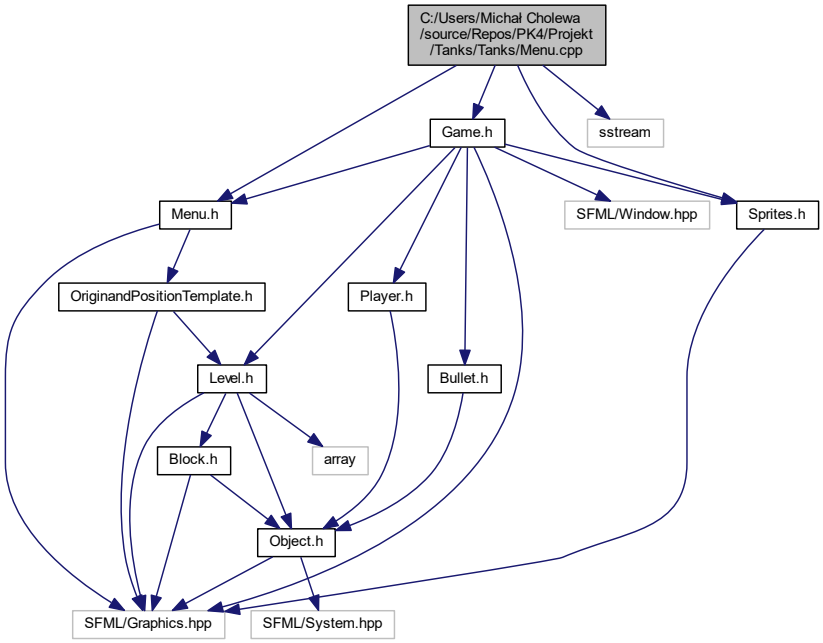
```
int main ( )
```

5.12 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Menu.cpp

```
#include "Menu.h"
#include "Game.h"
#include "Sprites.h"
```

```
#include <sstream>
```

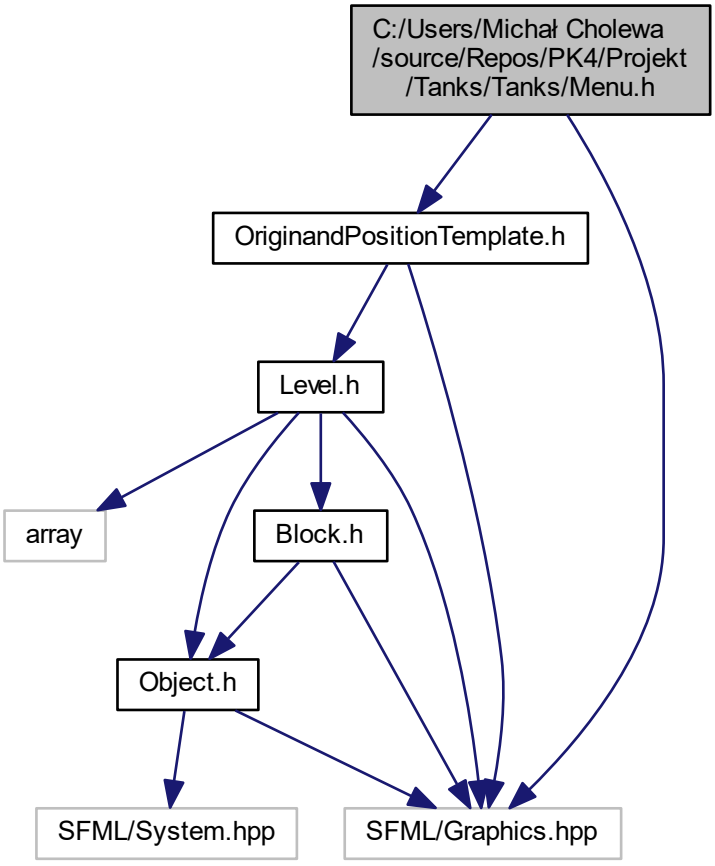
Wykres zależności załączania dla Menu.cpp:



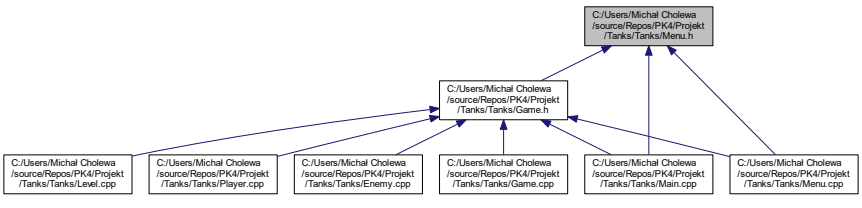
5.13 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/↩ Tanks/Menu.h

```
#include "OriginandPositionTemplate.h"
#include "SFML/Graphics.hpp"
```

Wykres zależności załączania dla Menu.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Menu

Wyliczenia

- enum Difficulty { EASY, MEDIUM, HARD }
- enum Keys {
  UP, DOWN, LEFT, RIGHT,
  SHOOT, ENTER, ESCAPE }
- enum Game_State {
  PLAYING, MAIN_MENU, OPTIONS, EXIT,
  OVER }

5.13.1 Dokumentacja typów wyliczanych

5.13.1.1 Difficulty

```
enum Difficulty
```

Typ wyliczeniowy do zapamietywania poziomu trudnosci wybranej przez gracza

Wartości wyliczeń

EASY	
MEDIUM	
HARD	

5.13.1.2 Game_State

```
enum Game_State
```

Typ wyliczeniowy do rozpoznawania i sprawdzania stanu gry

Wartości wyliczeń

PLAYING	
MAIN_MENU	
OPTIONS	
EXIT	
OVER	

5.13.1.3 Keys

```
enum Keys
```

Klucze odpowiadajace wcisnieciu odpowiednich przyciskow i zwiazanej z nimi reakcji

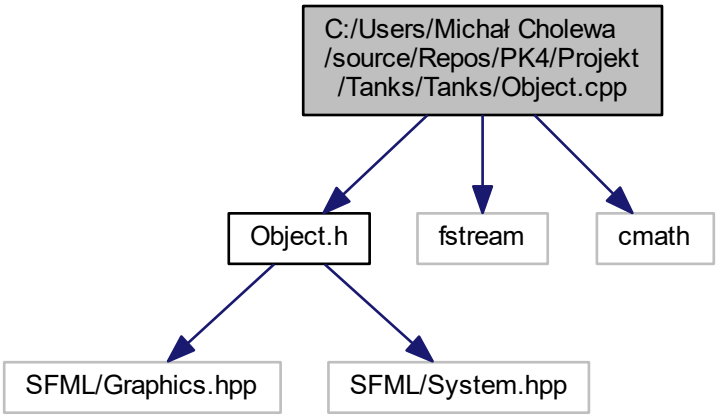
Wartości wyliczeń

UP	
DOWN	
LEFT	
RIGHT	
SHOOT	
ENTER	
ESCAPE	

5.14 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Object.cpp

```
#include "Object.h"
#include <fstream>
#include <cmath>
```

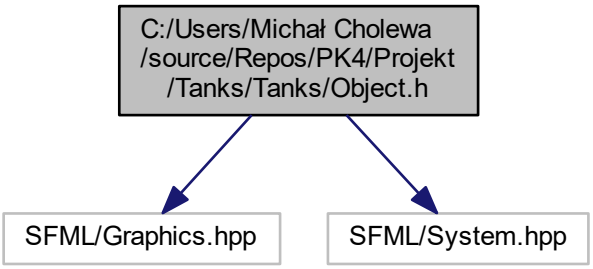
Wykres zależności załączania dla Object.cpp:



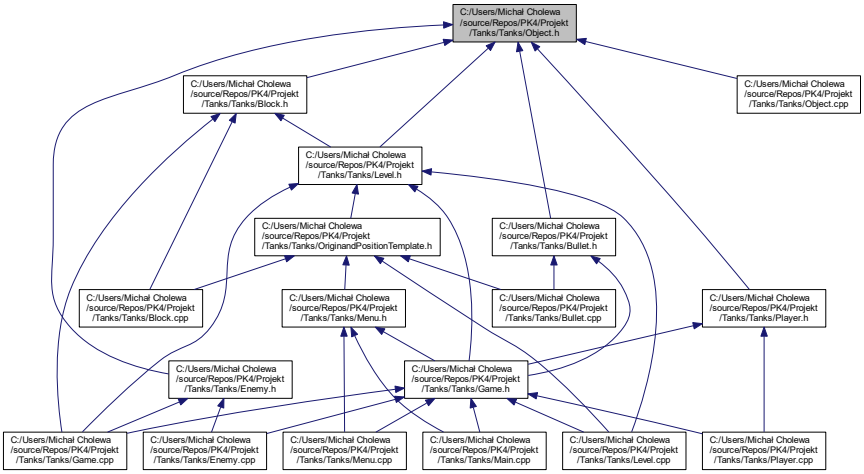
5.15 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Object.h

```
#include "SFML/Graphics.hpp"
#include "SFML/System.hpp"
```

Wykres zależności załączania dla Object.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Object

Wyliczenia

- enum Direction { UP_, DOWN_, LEFT_, RIGHT_ }

5.15.1 Dokumentacja typów wyliczanych

5.15.1.1 Direction

```
enum Direction
```

Typ wyliczeniowy do okreslenia zwrotu obiektu

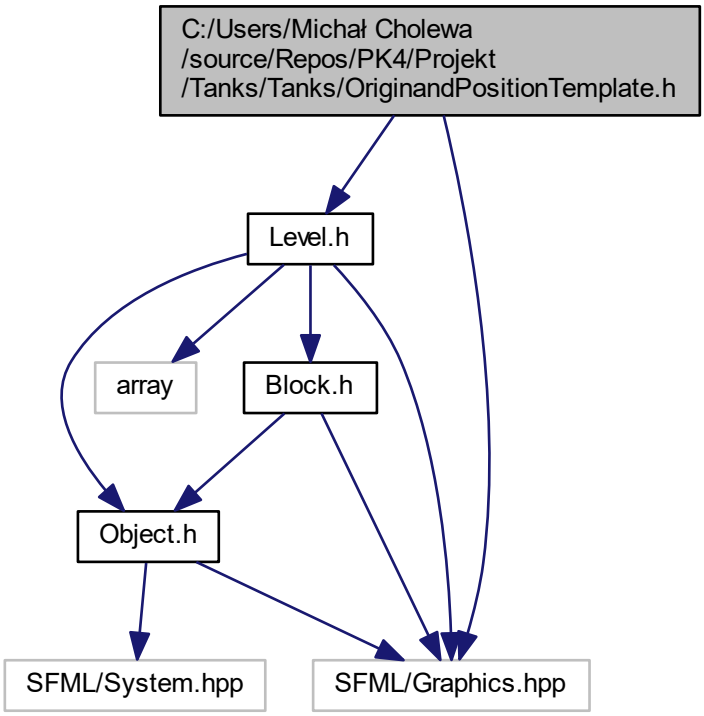
Wartości wyliczeń

UP_	
DOWN_	
LEFT_	
RIGHT_	

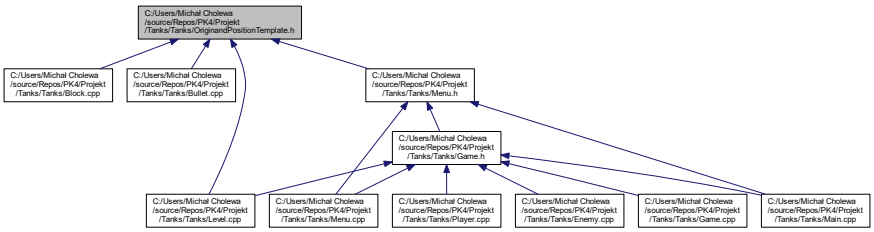
5.16 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/OriginandPositionTemplate.h

```
#include "SFML/Graphics.hpp"
#include "Level.h"
```

Wykres zależności załączania dla OriginandPositionTemplate.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Funkcje

- template<class T >
  void CenterOrigin (T &t)
- template<class T >
  void CenterPosition_X (T &t, sf::RenderWindow *&window, float y)
- template<class T >
  void SetPositionfromCenter (T &t, sf::RenderWindow *&window, float x, float y)

5.16.1 Dokumentacja funkcji

5.16.1.1 CenterOrigin()

```
template<class T >
void CenterOrigin (
            T & t )
```

5.16.1.2 CenterPosition_X()

```
template<class T >
void CenterPosition_X (
            T & t,
            sf::RenderWindow *& window,
            float y )
```

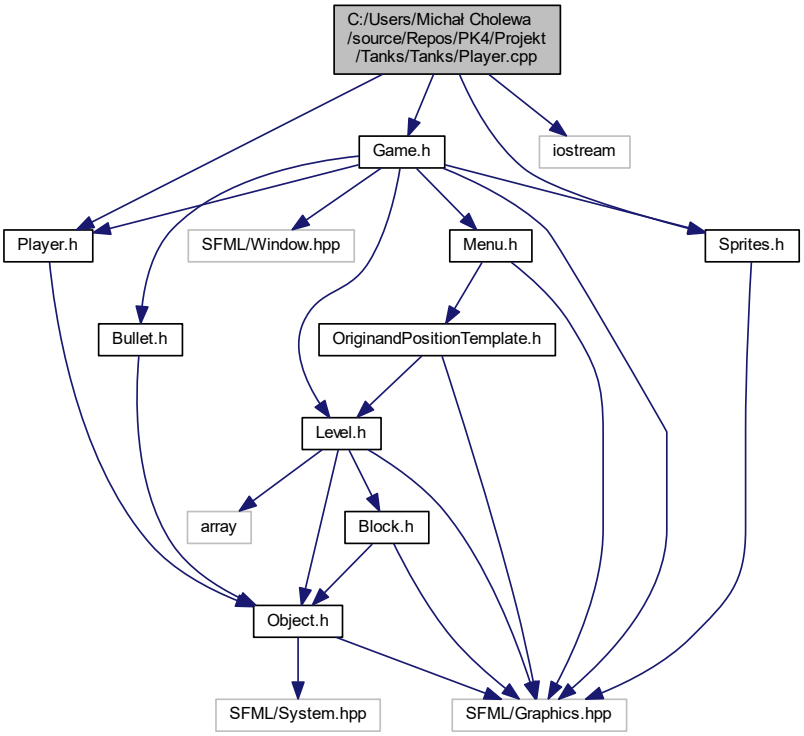
5.16.1.3 SetPositionfromCenter()

```
template<class T >
void SetPositionfromCenter (
            T & t,
            sf::RenderWindow *& window,
            float x,
            float y )
```

5.17 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Player.cpp

```
#include "Player.h"
#include "Game.h"
#include "Sprites.h"
#include <iostream>
```

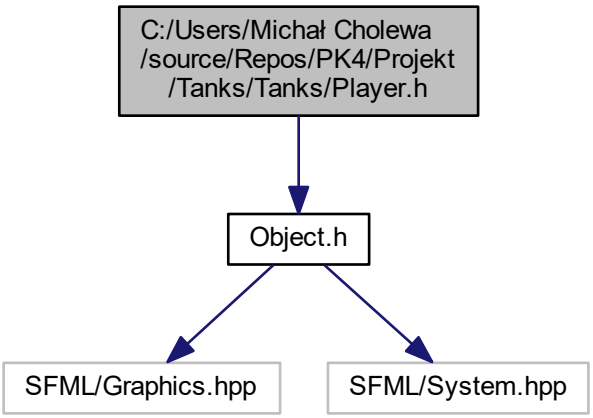
Wykres zależności załączania dla Player.cpp:



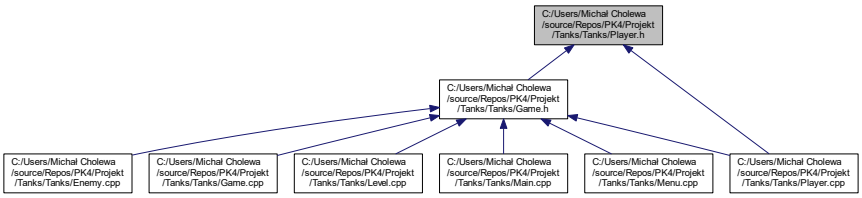
5.18 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Player.h

```
#include "Object.h"
```

Wykres zależności załączania dla Player.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

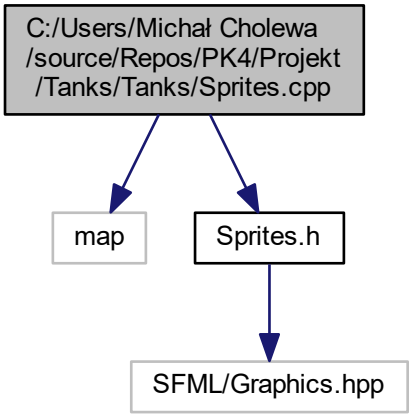
- class Player

5.19 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/resource.h

5.20 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Sprites.cpp

```
#include <map>
#include "Sprites.h"
```

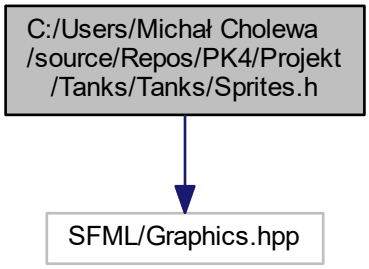
Wykres zależności załączania dla Sprites.cpp:



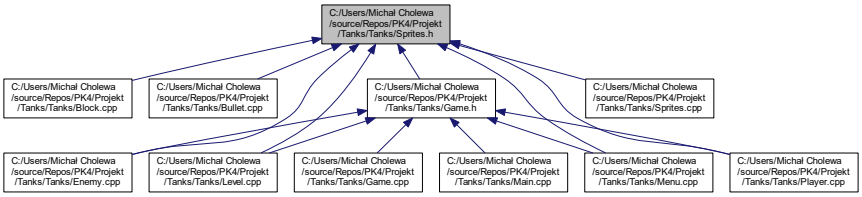
5.21 Dokumentacja pliku C:/Users/Michał Cholewa/source/Repos/PK4/Projekt/Tanks/Tanks/Sprites.h

```
#include <SFML/Graphics.hpp>
```

Wykres zależności załączania dla Sprites.h:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Sprites

Skorowidz