Zmodyfikowana gra Mario

Wygenerowano przez Doxygen 1.8.20

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:

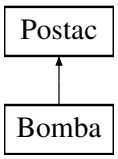
Rozdział 4

Dokumentacja klas

4.1 Dokumentacja klasy Bomba

`#include <Bomba.h>`

Diagram dziedziczenia dla Bomba



Metody publiczne

- Bomba (int x, int y, std::vector< std::vector< int >> &tabela)

  Konstruktor
- virtual void wykonaj_ruch (std::vector< std::vector< int >> &tabela, bool &koniec)

  Wykonany zostaje ruch odpowadający Bombie.
- virtual void smierc (std::vector< std::vector< int >> &tabela)

  Śmierć Bomby, usunięcie z planszy zajmowanych pól.

Dodatkowe Dziedziczone Składowe

4.1.1 Opis szczegółowy

Dziedziczy po klasie Postac. Jeśli zostanie stworzony obiekt klasy Bomba, to będą wykonywane zdefiniowane tutaj ruchy i śmierć.

4.1.2 Dokumentacja konstruktora i destruktora

4.1.2.1 Bomba()

```
Bomba::Bomba (
            int x,
            int y,
            std::vector< std::vector< int >> & tabela )
```

Konstruktor

Parametry

x	położenie danego przeciwnika na osi x
y	położenie danego przeciwnika na osi y
tabela	jako referencja przekazywana jest cała tabela, ponieważ zostanie w niej umieszczony przeciwnik

4.1.3 Dokumentacja funkcji składowych

4.1.3.1 smierc()

```
void Bomba::smierc (
            std::vector< std::vector< int >> & tabela )  [virtual]
```

Śmierć Bomby, usunięcie z planszy zajmowanych pól.

Parametry

tabela	jako referencja przekazywana jest cała tabela

Implementuje Postac.

4.1.3.2 wykonaj_ruch()

```
void Bomba::wykonaj_ruch (
            std::vector< std::vector< int >> & tabela,
            bool & koniec )  [virtual]
```

Wykonany zostaje ruch odpowadający Bombie.

Parametry

tabela	jako referencja przekazywana jest cała tabela
koniec	referencja na pole typu bool, jeśli zajdzie taka potrzeba (strzał dotknie gracza), to gra jest kończona

Implementuje Postac.

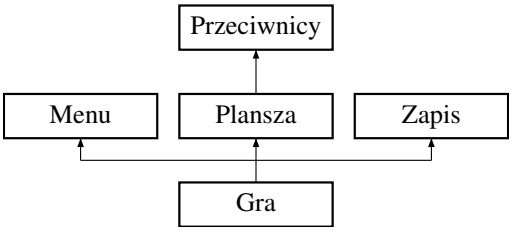
Dokumentacja dla tej klasy została wygenerowana z plików:

- Bomba.h
- Bomba.cpp

4.2 Dokumentacja klasy Gra

`#include <Gra.h>`

Diagram dziedziczenia dla Gra



Metody chronione

- Gra ()

  Konstruktor
- void przebieg (sf::RenderWindow &window)

  metoda realizująca przebieg całej gry

Przyjaciele

- class Mario

  Klasa jest wykorzystywana w klasie Mario, ma to na celu możliwość ponownej gry bez konieczności ponownego uruchamiania programu

Dodatkowe Dziedziczone Składowe

4.2.1 Opis szczegółowy

Klasa Gra odpowiada za złożenie trzech kluczowych fragmentów gry. Dziedziczy klasy Menu, Plansza i Zapis. Menu pozwala na wybranie ilości punktów do zdobycia. W klasie Plansza znajduje się przebieg gry. Natomiast Zapis odpowiada za zapis rezultatu gry.

4.2.2 Dokumentacja konstruktora i destruktora

4.2.2.1 Gra()

`Gra::Gra ( )  [protected]`

Konstruktor

4.2.3 Dokumentacja funkcji składowych

4.2.3.1 przebieg()

```
void Gra::przebieg (
            sf::RenderWindow & window )  [protected]
```

metoda realizująca przebieg całej gry

Parametry

window	przekazywane jest jako parametr aktualnie używane okno

4.2.4 Dokumentacja przyjaciół i funkcji związanych

4.2.4.1 Mario

```
friend class Mario  [friend]
```

Klasa jest wykorzystywana w klasie Mario, ma to na celu możliwość ponownej gry bez konieczności ponownego uruchamiania programu

Dokumentacja dla tej klasy została wygenerowana z plików:

- Gra.h
- Gra.cpp

4.3 Dokumentacja klasy Mario

```
#include <Mario.h>
```

Metody publiczne

- void przebieg ()

  tworzone zostaje okno. Następnie w pętli, dopóki okno jest otwarte tworzony zostaje obiekt klasy Gra i wywoływana jej metoda "przebieg".

4.3.1 Opis szczegółowy

Klasa umożliwiająca zagranie od początku bez potrzeby ponownego włączania programu.

4.3.2 Dokumentacja funkcji składowych

4.3.2.1 przebieg()

```
void Mario::przebieg ( )
```

tworzone zostaje okno. Następnie w pętli, dopóki okno jest otwarte tworzony zostaje obiekt klasy Gra i wywoływana jej metoda "przebieg".

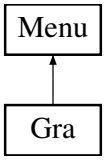
Dokumentacja dla tej klasy została wygenerowana z plików:

- Mario.h
- Mario.cpp

4.4 Dokumentacja klasy Menu

```
#include <Menu.h>
```

Diagram dziedziczenia dla Menu



Metody chronione

- Menu ()

  Konstruktor
- void wyswietl (sf::RenderWindow &window, int &iloscPunktow)

  zajmuje się wyświetleniem użytkownikowi początkowego interfejsu z wyborem ilości punktów do zdobycia

4.4.1 Opis szczegółowy

Klasa zajmująca się wyświetleniem użytkownikowi początkowego interfejsu z wyborem ilości punktów do zdobycia.

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 Menu()

```
Menu::Menu ( )  [protected]
```

Konstruktor

4.4.3 Dokumentacja funkcji składowych

4.4.3.1 wyswietl()

```
void Menu::wyswietl (
            sf::RenderWindow & window,
            int & iloscPunktow )  [protected]
```

zajmuje się wyświetleniem użytkownikowi początkowego interfejsu z wyborem ilości punktów do zdobycia

Parametry

window	przekazywane jest jako parametr przez referencję aktualnie używane okno
iloscPunktow	przez referencję przekazywana jest ilość punktów do zdobycia, która w tej metodzie zostaje przez użytkownika ustawiona

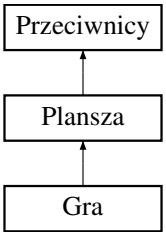
Dokumentacja dla tej klasy została wygenerowana z plików:

- Menu.h
- Menu.cpp

4.5 Dokumentacja klasy Plansza

`#include <Plansza.h>`

Diagram dziedziczenia dla Plansza



Metody chronione

- Plansza ()

  Konstruktor
- void wypelnienie_czesci_tablicy (std::mt19937 &gen, const int poczatekp, const int koniecp)

  metoda zajmująca się wypełnieniem części tablicy
- void wypelnienie_poczatkowych_pol (std::mt19937 &gen, const int poczatekp, const int koniecp)

  wypełnienie podłogami określonej liczby pól (bezpieczne_pola) by na samym początku na gracza nie czyhały żadne niebezpieczeństwa
- void LadowanieTekstur (sf::Texture &mario_tekstura_prawa, sf::Texture &mario_tekstura_lewa, sf::Texture &podl_tekstura, sf::Texture &moneta_tekstura, sf::Texture &tlo_tekstura, sf::Texture &strzal_poziomy_tekstura, sf::Texture &strzal_pionowy_tekstura, sf::Texture &strzelec_poziomy_tekstura, sf::Texture &strzelec_pionowy_tekstura, sf::Texture &bomba_tekstura)

  metoda zajmująca się załadowaniem wszystkich używanych tekstur
- void Rysowanie_czesci_planszy (const int start, sf::RenderWindow &window, sf::Texture &tlo_tekstura, sf::Texture &podl_tekstura, sf::Texture &mario_tekstura_prawa, sf::Texture &mario_tekstura_lewa, sf::Texture &moneta_tekstura, sf::Texture &strzal_poziomy_tekstura, sf::Texture &strzal_pionowy_tekstura, sf::Texture &strzelec_poziomy_tekstura, sf::Texture &strzelec_pionowy_tekstura, sf::Texture &bomba_tekstura)

  metoda zajmująca się wyświetleniem aktualnego stanu planszy
- void wyswietlenie (sf::RenderWindow &window)

  główna metoda klasy, zajmuje się sterowaniem i koordynowaniem całego przebiegu gry

Atrybuty chronione

- int iloscPunktow

  przechowuje całkowitą ilość punktów do zdobycia
- std::chrono::time_point< std::chrono::system_clock > start

  przechowuje czas startu gry
- std::chrono::time_point< std::chrono::system_clock > aktualny

  używany pomocniczo w celu odczytania czasu w konkretnym momencie trwania, aktualnie tylko na końcu, lecz w przyszłości najprawdopodobniej będzie miał również inne zastosowania
- int aktualna_ilosc_punktow

  przechowuje aktualną ilość punktów zebraną przez gracza

Metody prywatne

- void wypelnienie_pola (const double wartosc, const int pole)

  metoda zajmująca się wypełnieniem konkretnego pola tablicy
- bool wypelnienie_mindlwys (const int wysokosc, int pole)

  metoda sprawdzająca, czy wszystkie podłogi na poprzednim polu mają już wystarczającą długość, jeśli nie to wyspa zostaje przedłużona
- bool stworzenie_wyspy (const int i, const int pole)

  sprawdzanie jest, czy spełniony jest warunek wygenerowania wyspy, czyli po prostu czy na poprzednich polach była wystarczająca ilość podłóg
- bool maksymalny_odstep (const int pole)

  sprawdzenie, czy nie występuje zbyt duża ilość pustych pól bez podłóg
- void wykonanie_skoku (const int pierwsze_wyswietlane_pole)

  wykonanie skoku, przypisuje wartość polu skok
- bool sprawdzenie_pola (int i, int k)

  sprawdzenie pola na które będzie wchodził gracz
- void smierc (int i, int k)

  gracz poniósł klęskę
- void fizyka_postaci_pion (const int pierwsze_wyswietlane_pole)

  fizyka postaci (gracza) w pionie
- bool ruch_w_prawo (const int pierwsze_wyswietlane_pole)

  postać zostaje przesunięta w prawo (jeśli jest taka możliwość)
- bool czy_postac_jest_w_danej_kolumnie (const int kolumna)

  sprawdzenie, czy postać znajduje się w danej kolumnie
- bool ruch_w_lewo (const int pierwsze_wyswietlane_pole)

  postać zostaje przesunięta w lewo (jeśli jest taka możliwość)

Atrybuty prywatne

- std::vector< std::vector< int > > tabela

  tablica dwuwymiarowa integerów, to w niej przechowywany jest aktualny stan planszy zawierający potwory, gracza i strzały, podłogę, tło i monety.
- bool generowanie

  gdy gracz przejdzie na konkretne pola (ich położenie jest zależne od ustawionej długości planszy) to rozpoczyna się generowanie fragmentu (1/3) planszy, jednak gracz może stać na tym polu dłużej i żeby zapobiec uruchamianiu większej ilości wątków fakt odbywającej się generacji jest zawarty tutaj.
- std::barrier bariera

  bariera synchronizująca rozpoczęcie gry, ładowanie tekstur i wypełnianie pierwszych 2/3 planszy
- bool koniec

  przechowuje informacje o tym, czy etap faktycznej gry został zakończony
- int skok

  standardowo ustawiona na 0, gdy użytkownik naciśnie klawisz skoku zmieniona zostaje na ilość pól, na którą podskoczy postać z miejsca obecnego

4.5.1 Opis szczegółowy

Klasa odpowiadająca głównie za zarządzanie planszą i jej wyświetlanie, jak i umożliwienie sterowania graczowi.

4.5.2 Dokumentacja konstruktora i destruktora

4.5.2.1 Plansza()

```
Plansza::Plansza ( )  [protected]
```

Konstruktor

4.5.3 Dokumentacja funkcji składowych

4.5.3.1 czy_postac_jest_w_danej_kolumnie()

```
bool Plansza::czy_postac_jest_w_danej_kolumnie (
            const int kolumna )  [private]
```

sprawdzenie, czy postać znajduje się w danej kolumnie

Parametry

kolumna	współrzędna x

Zwraca

zwraca true jeśli gracz znajduje się w przekazanej jako argument kolumnie, w przeciwnym przypadku false

4.5.3.2 fizyka_postaci_pion()

```
void Plansza::fizyka_postaci_pion (
            const int pierwsze_wyswietlane_pole )  [private]
```

fizyka postaci (gracza) w pionie

Parametry

pierwsze_wyswietlane_pole	pierwsze wyswietlane pole, gracz zawsze znajduje się pomiędzy nim lub parę pól dalej (ich ilość znajduje się w pliku ConstantsAndIncludes.h i jest modyfikowalna)

4.5.3.3 LadowanieTekstur()

```
void Plansza::LadowanieTekstur (
            sf::Texture & mario_tekstura_prawa,
            sf::Texture & mario_tekstura_lewa,
            sf::Texture & podl_tekstura,
            sf::Texture & moneta_tekstura,
            sf::Texture & tlo_tekstura,
            sf::Texture & strzal_poziomy_tekstura,
            sf::Texture & strzal_pionowy_tekstura,
            sf::Texture & strzelec_poziomy_tekstura,
            sf::Texture & strzelec_pionowy_tekstura,
            sf::Texture & bomba_tekstura )  [protected]
```

metoda zajmująca się załadowaniem wszystkich używanych tekstur

4.5.3.4 maksymalny_odstep()

```
bool Plansza::maksymalny_odstep (
            const int pole )  [private]
```

sprawdzenie, czy nie występuje zbyt duża ilość pustych pól bez podłóg

Parametry

pole	aktualna współrzędna x

Zwraca

wartość true oznacza, że występowała podłoga na sprawdzonych polach. W przypadku false tworzona jest podługa na samym dole planszy (pozycja y: wysokosc_planszy - 1)

4.5.3.5 ruch_w_lewo()

```
bool Plansza::ruch_w_lewo (
            const int pierwsze_wyswietlane_pole )  [private]
```

postać zostaje przesunięta w lewo (jeśli jest taka możliwość)

Parametry

pierwsze_wyswietlane_pole	pierwsze wyswietlane pole, gracz zawsze znajduje się pomiędzy nim lub parę pól dalej (ich ilość znajduje się w pliku ConstantsAndIncludes.h i jest modyfikowalna)

Zwraca

zwraca wartość true jeśli udało się wykonać ruch, w przeciwnym przypadku false

4.5.3.6 ruch_w_prawo()

```
bool Plansza::ruch_w_prawo (
            const int pierwsze_wyswietlane_pole )  [private]
```

postać zostaje przesunięta w prawo (jeśli jest taka możliwość)

Parametry

pierwsze_wyswietlane_pole	pierwsze wyswietlane pole, gracz zawsze znajduje się pomiędzy nim lub parę pól dalej (ich ilość znajduje się w pliku ConstantsAndIncludes.h i jest modyfikowalna)

Zwraca

zwraca wartość true jeśli udało się wykonać ruch, w przeciwnym przypadku false

4.5.3.7 Rysowanie_czesci_planszy()

```
void Plansza::Rysowanie_czesci_planszy (
            const int start,
            sf::RenderWindow & window,
            sf::Texture & tlo_tekstura,
            sf::Texture & podl_tekstura,
            sf::Texture & mario_tekstura_prawa,
            sf::Texture & mario_tekstura_lewa,
            sf::Texture & moneta_tekstura,
            sf::Texture & strzal_poziomy_tekstura,
            sf::Texture & strzal_pionowy_tekstura,
            sf::Texture & strzelec_poziomy_tekstura,
            sf::Texture & strzelec_pionowy_tekstura,
            sf::Texture & bomba_tekstura )  [protected]
```

metoda zajmująca się wyświetleniem aktualnego stanu planszy

Parametry

start	współrzędna x, od niej zacznie się wyświetlanie planszy
window	przekazywane jest jako parametr aktualnie używane okno

4.5.3.8 smierc()

```
void Plansza::smierc (
            int i,
            int k )  [private]
```

gracz poniósł klęskę

Parametry

i	wysokość
k	aktualna współrzędna x

4.5.3.9 sprawdzenie_pola()

```
bool Plansza::sprawdzenie_pola (
            int i,
            int k )  [private]
```

sprawdzenie pola na które będzie wchodził gracz

Parametry

i	wysokość
k	aktualna współrzędna x

Zwraca

wartość true oznacza, że nie ma przeciwwskazań do wykonania ruchu

4.5.3.10 stworzenie_wyspy()

```
bool Plansza::stworzenie_wyspy (
            const int i,
            const int pole )  [private]
```

sprawdzanie jest, czy spełniony jest warunek wygenerowania wyspy, czyli po prostu czy na poprzednich polach była wystarczająca ilość podłóg

Parametry

i	wysokość
pole	aktualna współrzędna x

Zwraca

wartość true oznacza możliwość stworzenia wyspy

4.5.3.11 wykonanie_skoku()

```
void Plansza::wykonanie_skoku (
            const int pierwsze_wyswietlane_pole )  [private]
```

wykonanie skoku, przypisuje wartość polu skok

Parametry

pierwsze_wyswietlane_pole	pierwsze wyswietlane pole, gracz zawsze znajduje się pomiędzy nim lub parę pól dalej (ich ilość znajduje się w pliku ConstantsAndIncludes.h i jest modyfikowalna)

4.5.3.12 wypelnienie_czesci_tablicy()

```
void Plansza::wypelnienie_czesci_tablicy (
            std::mt19937 & gen,
            const int poczatekp,
            const int koniecp )  [protected]
```

metoda zajmująca się wypełnieniem części tablicy

Parametry

gen	przekazany przez referencję generator
poczatekp	współrzędna x, od niej zacznie się generowanie planszy
koniecp	współrzędna x, na niej zakończy się generowanie planszy

4.5.3.13 wypelnienie_mindlwys()

```
bool Plansza::wypelnienie_mindlwys (
            const int wysokosc,
            int pole )  [private]
```

metoda sprawdzająca, czy wszystkie podłogi na poprzednim polu mają już wystarczającą długość, jeśli nie to wyspa zostaje przedłużona

Parametry

wysokosc	sprawdzana wysokość
pole	aktualna współrzędna x

Zwraca

wartość true oznacza przedłużenie wyspy

4.5.3.14 wypelnienie_poczatkowych_pol()

```
void Plansza::wypelnienie_poczatkowych_pol (
            std::mt19937 & gen,
            const int poczatekp,
            const int koniecp )  [protected]
```

wypełnienie podłogami określonej liczby pól (bezpieczne_pola) by na samym początku na gracza nie czyhały żadne niebezpieczeństwa

Parametry

gen	przekazany przez referencję generator
poczatekp	współrzędna x, od niej zacznie się generowanie planszy
koniecp	współrzędna x, na niej zakończy się generowanie planszy

4.5.3.15 wypelnienie_pola()

```
void Plansza::wypelnienie_pola (
            const double wartosc,
            const int pole )  [private]
```

metoda zajmująca się wypełnieniem konkretnego pola tablicy

Parametry

wartosc	wartość "losowa" od której zależeć będzie generowany element
pole	współrzędna x planszy

4.5.3.16 wyswietlenie()

```
void Plansza::wyswietlenie (
            sf::RenderWindow & window )  [protected]
```

główna metoda klasy, zajmuje się sterowaniem i koordynowaniem całego przebiegu gry

Parametry

window	przekazywane jest jako parametr aktualnie używane okno

4.5.4 Dokumentacja atrybutów składowych

4.5.4.1 aktualna_ilosc_punktow

```
int Plansza::aktualna_ilosc_punktow  [protected]
```

przechowuje aktualną ilość punktów zebraną przez gracza

4.5.4.2 aktualny

```
std::chrono::time_point<std::chrono::system_clock> Plansza::aktualny  [protected]
```

używany pomocniczo w celu odczytania czasu w konkretnym momencie trwania, aktualnie tylko na końcu, lecz w przyszłości najprawdopodobniej będzie miał również inne zastosowania

4.5.4.3 bariera

```
std::barrier Plansza::bariera  [private]
```

bariera synchronizująca rozpoczęcie gry, ładowanie tekstur i wypełnianie pierwszych 2/3 planszy

4.5.4.4 generowanie

```
bool Plansza::generowanie  [private]
```

gdy gracz przejdzie na konkretne pola (ich położenie jest zależne od ustawionej długości planszy) to rozpoczyna się generowanie fragmentu (1/3) planszy, jednak gracz może stać na tym polu dłużej i żeby zapobiec uruchamianiu większej ilości wątków fakt odbywającej się generacji jest zawarty tutaj.

4.5.4.5 iloscPunktow

```
int Plansza::iloscPunktow  [protected]
```

przechowuje całkowitą ilość punktów do zdobycia

4.5.4.6 koniec

```
bool Plansza::koniec  [private]
```

przechowuje informacje o tym, czy etap faktycznej gry został zakończony

4.5.4.7 skok

```
int Plansza::skok  [private]
```

standardowo ustawiona na 0, gdy użytkownik naciśnie klawisz skoku zmieniona zostaje na ilość pól, na którą podskoczy postać z miejsca obecnego

4.5.4.8 start

```
std::chrono::time_point<std::chrono::system_clock> Plansza::start  [protected]
```

przechowuje czas startu gry

4.5.4.9 tabela

```
std::vector<std::vector<int> > Plansza::tabela  [private]
```

tablica dwuwymiarowa integerów, to w niej przechowywany jest aktualny stan planszy zawierający potwory, gracza i strzały, podłogę, tło i monety.

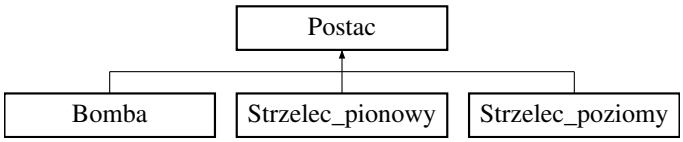
Dokumentacja dla tej klasy została wygenerowana z plików:

- Plansza.h
- Plansza.cpp

4.6 Dokumentacja klasy Postac

`#include <Ruchy.h>`

Diagram dziedziczenia dla Postac



Metody chronione

- bool sprawdzenie_pola (int wartosc_pola, bool &koniec)

  Sprawdza czy potencjalny strzał może zostać wykonany, w przypadku napotkania gracza przerwa również grę
- Postac (int x, int y)

  Konstruktor, przypisuje danemu przeciwnikowi współrzędne oraz czas stworzenia
- int zwroc_x () const

  Metoda zwracająca współrzędną x
- int zwroc_y () const

  Metoda zwracająca współrzędną y
- virtual void wykonaj_ruch (std::vector< std::vector< int >> &tabela, bool &koniec)=0

  Metoda czysto wirtualna, wykonywany będzie ruch zależny od tego, z jakim przeciwnikiem mamy do czynienia
- virtual void smierc (std::vector< std::vector< int >> &tabela)=0

  Metoda czysto wirtualna, wpływ śmierci przeciwnika na grę jest zależny od tego, z jakim przeciwnikiem mamy do czynienia

Atrybuty chronione

- std::chrono::time_point< std::chrono::high_resolution_clock > czas

  czas pojawienia się na mapie danego przeciwnika, będzie od niego zależeć czas strzału
- int x

  położenie danego przeciwnika na osi x
- int y

  położenie danego przeciwnika na osi y

Przyjaciele

- class Przeciwnicy

  W klasie Przeciwnicy znajduje się lista zawierająca wszystkich aktualnie znajdujących się na planszy przeciwników.

4.6.1 Opis szczegółowy

Zawiera dwie metody czysto wirtualne, jest dziedziczona przez wszystkie potwory umieszczone w grze. O każdym stworzonym potworze musimy przechowywać pewne istotne informacje, więc odpowiednie, służące temu pola znajdują się właśnie w tej klasie.

Parametry

czas	czas pojawienia się na mapie danego przeciwnika, będzie od niego zależeć czas strzału
x	położenie danego przeciwnika na osi x
y	położenie danego przeciwnika na osi y

4.6.2 Dokumentacja konstruktora i destruktora

4.6.2.1 Postac()

```
Postac::Postac (
            int x,
            int y )  [protected]
```

Konstruktor, przypisuje danemu przeciwnikowi współrzędne oraz czas stworzenia

Parametry

x	położenie danego przeciwnika na osi x
y	położenie danego przeciwnika na osi y

4.6.3 Dokumentacja funkcji składowych

4.6.3.1 smierc()

```
virtual void Postac::smierc (
            std::vector< std::vector< int >> & tabela )  [protected], [pure virtual]
```

Metoda czysto wirtualna, wpływ śmierci przeciwnika na grę jest zależny od tego, z jakim przeciwnikiem mamy do czynienia

Parametry

tabela	

Implementowany w Strzelec_poziomy, Strzelec_pionowy i Bomba.

4.6.3.2 sprawdzenie_pola()

```
bool Postac::sprawdzenie_pola (
```

```
            int wartosc_pola,
            bool & koniec )  [protected]
```

Sprawdza czy potencjalny strzał może zostać wykonany, w przypadku napotkania gracza przerwa również grę

Parametry

wartosc_pola	przekazywana jest wartość sprawdzanego pola planszy, nie ma potrzeby przekazywania jej całej, ponieważ i tak sprawdzany będzie pojedynczy element
koniec	referencja na pole typu bool, jeśli zajdzie taka potrzeba (strzał dotknie gracza), to gra jest kończona

Zwraca

Zwracana jest informacja, czy potencjalny strzał może zostać wykonany

4.6.3.3 wykonaj_ruch()

```
virtual void Postac::wykonaj_ruch (
            std::vector< std::vector< int >> & tabela,
            bool & koniec )  [protected], [pure virtual]
```

Metoda czysto wirtualna, wykonywany będzie ruch zależny od tego, z jakim przeciwnikiem mamy do czynienia

Parametry

tabela	jako referencja przekazywana jest cała tabela
koniec	referencja na pole typu bool, jeśli zajdzie taka potrzeba (strzał dotknie gracza), to gra jest kończona

Implementowany w Strzelec_poziomy, Strzelec_pionowy i Bomba.

4.6.3.4 zwroc_x()

```
int Postac::zwroc_x ( ) const  [protected]
```

Metoda zwracająca współrzędną x

Zwraca

x

4.6.3.5 zwroc_y()

```
int Postac::zwroc_y ( ) const  [protected]
```

Metoda zwracająca współrzędną y

Zwraca

y

4.6.4 Dokumentacja przyjaciół i funkcji związanych

4.6.4.1 Przeciwnicy

```
friend class Przeciwnicy  [friend]
```

W klasie Przeciwnicy znajduje się lista zawierająca wszystkich aktualnie znajdujących się na planszy przeciwników.

4.6.5 Dokumentacja atrybutów składowych

4.6.5.1 czas

```
std::chrono::time_point<std::chrono::high_resolution_clock> Postac::czas  [protected]
```

czas pojawienia się na mapie danego przeciwnika, będzie od niego zależeć czas strzału

4.6.5.2 x

```
int Postac::x  [protected]
```

położenie danego przeciwnika na osi x

4.6.5.3 y

```
int Postac::y  [protected]
```

położenie danego przeciwnika na osi y

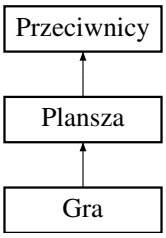
Dokumentacja dla tej klasy została wygenerowana z plików:

- Ruchy.h
- Ruchy.cpp

4.7 Dokumentacja klasy Przeciwnicy

`#include <Przeciwnicy.h>`

Diagram dziedziczenia dla Przeciwnicy



Metody chronione

- Przeciwnicy ()

  Konstruktor
- ~Przeciwnicy ()

  Destruktor zwalniający całą zajętą przez listę pamięć
- void usmierc_przeciwnika (std::vector< std::vector< int >> &tabela, int x, int y)

  po stanięciu na jakimś polu zajmowanym przez przeciwnika znika on z planszy wraz ze swoimi strzałami
- void usun_ewentualne_potwory (int x)

  usuwa z listy wszystkie potwory znajdujące się na danym polu x
- void wykonaj_ruchy_przeciwnikow (std::vector< std::vector< int >> &tabela, bool &koniec)

  wykonanie ruchów wszystkich przeciwników znajdujących się na liście
- void potencjalne_stworzenie_potworow (std::vector< std::vector< int >> &tabela, int x, std::mt19937 &gen)

  Na przekazanej jako argument pozycji x szuka podłóg, czyli potencjalnych miejsc na stworzenie przeciwników lub monety. Gdy podłoga zostanie znaleziona, to jest szansa na stworzenie tam przeciwnika lub postawienia monety, w zależności od "wylosowanej" wartości.

Atrybuty prywatne

- std::list< std::unique_ptr< Postac > > potwory

  potwory w liscie umieszczane są po kolei, w zależności od położenia x

4.7.1 Opis szczegółowy

Klasa umożliwiająca dodanie nowych przeciwników, w jej polu jest lista zawierająca wszystkich aktualnych przeciwników. Po uśmierceniu danego przeciwnika znajduje się on dalej w liście, zostaje usunięty dopiero w chwili, gdy pole przez niego zajmowane stanie się niewidoczne dla gracza.

4.7.2 Dokumentacja konstruktora i destruktora

4.7.2.1 Przeciwnicy()

```
Przeciwnicy::Przeciwnicy ( )  [protected]
```

Konstruktor

4.7.2.2 ∼Przeciwnicy()

```
Przeciwnicy::∼Przeciwnicy ( )  [protected]
```

Destruktor zwalniający całą zajętą przez listę pamięć

4.7.3 Dokumentacja funkcji składowych

4.7.3.1 potencjalne_stworzenie_potworow()

```
void Przeciwnicy::potencjalne_stworzenie_potworow (
            std::vector< std::vector< int >> & tabela,
            int x,
            std::mt19937 & gen )  [protected]
```

Na przekazanej jako argument pozycji x szuka podłóg, czyli potencjalnych miejsc na stworzenie przeciwników lub monety. Gdy podłoga zostanie znaleziona, to jest szansa na stworzenie tam przeciwnika lub postawienia monety, w zależności od "wylosowanej" wartości.

Parametry

tabela	jako referencja przekazywana jest cała tabela
x	współrzędna x
gen	przekazany przez referencję generator

stworzenie Strzelca pionowego

stworzenie Strzelca poziomego

stworzenie Bomby

umieszczenie monety

4.7.3.2 usmierc_przeciwnika()

```
void Przeciwnicy::usmierc_przeciwnika (
            std::vector< std::vector< int >> & tabela,
            int x,
            int y )  [protected]
```

po stanięciu na jakimś polu zajmowanym przez przeciwnika znika on z planszy wraz ze swoimi strzałami

Parametry

tabela	jako referencja przekazywana jest cała tabela
x	położenie danego przeciwnika na osi x
y	położenie danego przeciwnika na osi y

4.7.3.3 usun_ewentualne_potwory()

```
void Przeciwnicy::usun_ewentualne_potwory (
            int x )  [protected]
```

usuwa z listy wszystkie potwory znajdujące się na danym polu x

Parametry

x	współrzędna x

4.7.3.4 wykonaj_ruchy_przeciwnikow()

```
void Przeciwnicy::wykonaj_ruchy_przeciwnikow (
            std::vector< std::vector< int >> & tabela,
            bool & koniec )  [protected]
```

wykonanie ruchów wszystkich przeciwników znajdujących się na liście

Parametry

tabela	jako referencja przekazywana jest cała tabela
koniec	referencja na pole typu bool, jeśli zajdzie taka potrzeba (strzał dotknie gracza), to gra jest kończona

4.7.4 Dokumentacja atrybutów składowych

4.7.4.1 potwory

```
std::list<std::unique_ptr<Postac> > Przeciwnicy::potwory  [private]
```

potwory w liscie umieszczane są po kolei, w zależności od położenia x

Dokumentacja dla tej klasy została wygenerowana z plików:

- Przeciwnicy.h
- Przeciwnicy.cpp

4.8 Dokumentacja klasy Rekord

`#include <Rekord.h>`

Metody publiczne

- Rekord ()

  Konstruktor
- Rekord (std::string nazwa, long long czas, int zdobyte_punkty, int punkty_do_zdobycia)

  Konstruktor z deklaracją wszystkich elementów
- std::string zwroc_nazwe ()

  Metoda zwracająca nazwę
- long long zwroc_czas ()

  Metoda zwracająca czas gry w sekundach
- int zwroc_zdobyte_punkty ()

  metoda zwracająca ilość zdobytych przez gracza punktów
- int zwroc_punkty_do_zdobycia ()

  metoda zwracająca cel punktów ustawiony przez gracza na samym początku

Atrybuty prywatne

- std::string nazwa

  nazwa użytkownika, jej podanie jest opcjonalne
- long long czas

  całkowity czas gry w sekundach
- int zdobyte_punkty

  ilość zdobytych przez gracza punktów
- int punkty_do_zdobycia

  cel punktów ustawiony przez gracza na samym początku

Przyjaciele

- bool operator< (const Rekord &l, const Rekord &p)
- bool operator> (const Rekord &l, const Rekord &p)
- std::ostream & operator<< (std::ostream &str, const Rekord &rekord)
- std::istream & operator>> (std::istream &str, Rekord &rekord)

4.8.1 Opis szczegółowy

Klasa umożliwiająca zapisanie pojedynczego rekordu.

4.8.2 Dokumentacja konstruktora i destruktora

4.8.2.1 Rekord() [1/2]

```
Rekord::Rekord ( )
```

Konstruktor

4.8.2.2 Rekord() [2/2]

```
Rekord::Rekord (
            std::string nazwa,
            long long czas,
            int zdobyte_punkty,
            int punkty_do_zdobycia )
```

Konstruktor z deklaracją wszystkich elementów

Parametry

nazwa	nazwa użytkownika, jej podanie jest opcjonalne
czas	całkowity czas gry w sekundach
zdobyte_punkty	ilość zdobytych przez gracza punktów
punkty_do_zdobycia	cel punktów ustawiony przez gracza na samym początku

4.8.3 Dokumentacja funkcji składowych

4.8.3.1 zwroc_czas()

```
long long Rekord::zwroc_czas ( )
```

Metoda zwracająca czas gry w sekundach

Zwraca

czas

4.8.3.2 zwroc_nazwe()

```
std::string Rekord::zwroc_nazwe ( )
```

Metoda zwracająca nazwę

Zwraca

nazwa

4.8.3.3 zwroc_punkty_do_zdobycia()

```
int Rekord::zwroc_punkty_do_zdobycia ( )
```

metoda zwracająca cel punktów ustawiony przez gracza na samym początku

Zwraca

punkty_do_zdobycia

4.8.3.4 zwroc_zdobyte_punkty()

```
int Rekord::zwroc_zdobyte_punkty ( )
```

metoda zwracająca ilość zdobytych przez gracza punktów

Zwraca

zdobyte_punkty

4.8.4 Dokumentacja przyjaciół i funkcji związanych

4.8.4.1 operator<

```
bool operator< (
            const Rekord & l,
            const Rekord & p )  [friend]
```

4.8.4.2 operator<<

```
std::ostream& operator<< (
            std::ostream & str,
            const Rekord & rekord )  [friend]
```

4.8.4.3 operator>

```
bool operator> (
            const Rekord & l,
            const Rekord & p )  [friend]
```

4.8.4.4 operator>>

```
std::istream& operator>> (
             std::istream & str,
             Rekord & rekord )  [friend]
```

4.8.5 Dokumentacja atrybutów składowych

4.8.5.1 czas

```
long long Rekord::czas  [private]
```

całkowity czas gry w sekundach

4.8.5.2 nazwa

```
std::string Rekord::nazwa  [private]
```

nazwa użytkownika, jej podanie jest opcjonalne

4.8.5.3 punkty_do_zdobycia

```
int Rekord::punkty_do_zdobycia  [private]
```

cel punktów ustawiony przez gracza na samym początku

4.8.5.4 zdobyte_punkty

```
int Rekord::zdobyte_punkty  [private]
```

ilość zdobytych przez gracza punktów

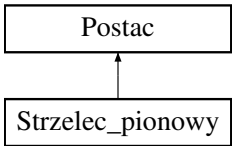
Dokumentacja dla tej klasy została wygenerowana z plików:

- Rekord.h
- Rekord.cpp

4.9 Dokumentacja klasy Strzelec_pionowy

`#include <Strzelec_pionowy.h>`

Diagram dziedziczenia dla Strzelec_pionowy



Metody publiczne

- Strzelec_pionowy (int x, int y, std::vector< std::vector< int >> &tabela)

  Konstruktor
- virtual void wykonaj_ruch (std::vector< std::vector< int >> &tabela, bool &koniec)

  Wykonany zostaje ruch odpowadający Strzelcowi pionowemu.
- virtual void smierc (std::vector< std::vector< int >> &tabela)

  Śmierć Strzelca pionowego, usunięcie z planszy zajmowanych pól oraz strzału znajdującego się w locie.

Dodatkowe Dziedziczone Składowe

4.9.1 Opis szczegółowy

Dziedziczy po klasie Postac. Jeśli zostanie stworzony obiekt klasy Strzelec_pionowy, to będą wykonywane zdefiniowane tutaj ruchy i śmierć.

4.9.2 Dokumentacja konstruktora i destruktora

4.9.2.1 Strzelec_pionowy()

```
Strzelec_pionowy::Strzelec_pionowy (
            int x,
            int y,
            std::vector< std::vector< int >> & tabela )
```

Konstruktor

Parametry

x	położenie danego przeciwnika na osi x
y	położenie danego przeciwnika na osi y
tabela	jako referencja przekazywana jest cała tabela, ponieważ zostanie w niej umieszczony przeciwnik

4.9.3 Dokumentacja funkcji składowych

4.9.3.1 smierc()

```
void Strzelec_pionowy::smierc (
            std::vector< std::vector< int >> & tabela )  [virtual]
```

Śmierć Strzelca pionowego, usunięcie z planszy zajmowanych pól oraz strzału znajdującego się w locie.

Parametry

tabela	jako referencja przekazywana jest cała tabela

Implementuje Postac.

4.9.3.2 wykonaj_ruch()

```
void Strzelec_pionowy::wykonaj_ruch (
            std::vector< std::vector< int >> & tabela,
            bool & koniec )  [virtual]
```

Wykonany zostaje ruch odpowadający Strzelcowi pionowemu.

Parametry

tabela	jako referencja przekazywana jest cała tabela
koniec	referencja na pole typu bool, jeśli zajdzie taka potrzeba (strzał dotknie gracza), to gra jest kończona

Implementuje Postac.

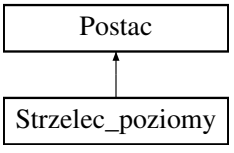
Dokumentacja dla tej klasy została wygenerowana z plików:

- Strzelec_pionowy.h
- Strzelec_pionowy.cpp

4.10 Dokumentacja klasy Strzelec_poziomy

```
#include <Strzelec_poziomy.h>
```

Diagram dziedziczenia dla Strzelec_poziomy



Metody publiczne

- Strzelec_poziomy (int x, int y, std::vector< std::vector< int >> &tabela)

  Konstruktor
- virtual void wykonaj_ruch (std::vector< std::vector< int >> &tabela, bool &koniec)

  Wykonany zostaje ruch odpowadający Strzelcowi poziomemu.
- virtual void smierc (std::vector< std::vector< int >> &tabela)

  Śmierć Strzelca poziomego, usunięcie z planszy zajmowanych pól oraz strzału znajdującego się w locie.

Dodatkowe Dziedziczone Składowe

4.10.1 Opis szczegółowy

Dziedziczy po klasie Postac. Jeśli zostanie stworzony obiekt klasy Strzelec_poziomy, to będą wykonywane zdefiniowane tutaj ruchy i śmierć.

4.10.2 Dokumentacja konstruktora i destruktora

4.10.2.1 Strzelec_poziomy()

```
Strzelec_poziomy::Strzelec_poziomy (
            int x,
            int y,
            std::vector< std::vector< int >> & tabela )
```

Konstruktor

Parametry

x	położenie danego przeciwnika na osi x
y	położenie danego przeciwnika na osi y
tabela	jako referencja przekazywana jest cała tabela, ponieważ zostanie w niej umieszczony przeciwnik

4.10.3 Dokumentacja funkcji składowych

4.10.3.1 smierc()

```
void Strzelec_poziomy::smierc (
            std::vector< std::vector< int >> & tabela )  [virtual]
```

Śmierć Strzelca poziomego, usunięcie z planszy zajmowanych pól oraz strzału znajdującego się w locie.

Parametry

tabela	jako referencja przekazywana jest cała tabela

Implementuje Postac.

4.10.3.2 wykonaj_ruch()

```
void Strzelec_poziomy::wykonaj_ruch (
            std::vector< std::vector< int >> & tabela,
            bool & koniec ) [virtual]
```

Wykonany zostaje ruch odpowadający Strzelcowi poziomemu.

Parametry

tabela	jako referencja przekazywana jest cała tabela
koniec	referencja na pole typu bool, jeśli zajdzie taka potrzeba (strzał dotknie gracza), to gra jest kończona

Implementuje Postac.

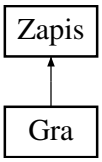
Dokumentacja dla tej klasy została wygenerowana z plików:

- Strzelec_poziomy.h
- Strzelec_poziomy.cpp

4.11 Dokumentacja klasy Zapis

```
#include <Zapis.h>
```

Diagram dziedziczenia dla Zapis



Metody chronione

- Zapis ()

  Konstruktor
- bool weryfikacja_nazwy (std::string &nazwa)

  Metoda sprawdzająca, czy wprowadzona nazwa gracza spełnia narzucone wymagania, czyli zawiera tylko litery i cyfry.
- void wyswietl (sf::RenderWindow &window, const int cel_punktow, const int uzyskane_punkty, std::chrono↩ ::time_point< std::chrono::system_clock > start, std::chrono::time_point< std::chrono::system_clock > koniec)

  Metoda odpowiadająca za umożliwienie użytkownikowi decydowania oraz wyświetlenie treści.

Atrybuty chronione

- bool koniec_zapisu

  Zawiera informację, czy faza wyświetlania rankingów została zakończona

4.11.1 Opis szczegółowy

Klasa zajmująca się głównie etapem zapisu wyniku do pliku i wyświetleniem rankingów.

4.11.2 Dokumentacja konstruktora i destruktora

4.11.2.1 Zapis()

```
Zapis::Zapis ( )  [protected]
```

Konstruktor

4.11.3 Dokumentacja funkcji składowych

4.11.3.1 weryfikacja_nazwy()

```
bool Zapis::weryfikacja_nazwy (
            std::string & nazwa )  [protected]
```

Metoda sprawdzająca, czy wprowadzona nazwa gracza spełnia narzucone wymagania, czyli zawiera tylko litery i cyfry.

Parametry

nazwa	przekazana jako referencja nazwa

Zwraca

zwracany zostaje wynik, jeśli nazwa spełnia wymagania to zwrócona jest prawda, w przeciwnym przypadku fałsz

4.11.3.2 wyswietl()

```
void Zapis::wyswietl (
            sf::RenderWindow & window,
```

```
const int cel_punktow,
const int uzyskane_punkty,
std::chrono::time_point< std::chrono::system_clock > start,
std::chrono::time_point< std::chrono::system_clock > koniec )  [protected]
```

Metoda odpowiadająca za umożliwienie użytkownikowi decydowania oraz wyświetlenie treści.

Parametry

window	przekazywane jest jako parametr przez referencję aktualnie używane okno
cel_punktow	przekazany zostaje ustawiony przez gracza cel punktów
uzyskane_punkty	przekazane zostaje ile punktów udało się zdobyć graczowi
start	czas rozpoczęcia gry
koniec	czas zakończenia gry

4.11.4 Dokumentacja atrybutów składowych

4.11.4.1 koniec_zapisu

```
bool Zapis::koniec_zapisu  [protected]
```

Zawiera informację, czy faza wyświetlania rankingów została zakończona

Dokumentacja dla tej klasy została wygenerowana z plików:

- Zapis.h
- Zapis.cpp

Rozdział 5

Dokumentacja plików

5.1 Dokumentacja pliku Bomba.cpp

```
#include "Bomba.h"
```

5.2 Dokumentacja pliku Bomba.h

```
#include "ConstantsAndIncludes.h"
#include "Ruchy.h"
```

Komponenty

- class Bomba

5.3 Dokumentacja pliku ConstantsAndIncludes.h

```
#include <vector>
#include <algorithm>
#include <barrier>
#include <random>
#include <thread>
#include <iostream>
#include <SFML/Graphics.hpp>
#include <chrono>
#include <list>
#include <windows.h>
#include <sstream>
#include <memory>
#include <regex>
#include <filesystem>
#include <ranges>
#include <fstream>
```

Definicje

- #define dlugosc_planszy 210
- #define wysokosc_planszy 16
- #define bezpieczne_pola 10
- #define maksymalny_odstep_const 4
- #define minimalna_dlugosc_wyspy 5
- #define roznica_wysokosci_pomiedzy_wyspami 3
- #define skok_const 4
- #define ilosc_wyswietlonych_pol 35
- #define odstep_gorny 30.0f
- #define odstep_boczny 30.0f
- #define minimalny_czas_przesuniecia_strzalu 0.3
- #define minimalny_czas_pomiędzy_ruchem_postaci 30
- #define zasieg_strzalu_pionowego 5
- #define zasieg_strzalu_poziomego 6
- #define maksymalne_cofniecie_gracza 5
- #define podloga 1
- #define gracz_p 10
- #define gracz_k 13
- #define pusta_przestrzen 0
- #define gora_gracza_p 10
- #define dol_gracza_p 11
- #define gora_gracza_l 12
- #define dol_gracza_l 13
- #define punkt 2
- #define strzelec_poziomy_g 20
- #define strzelec_poziomy_d 21
- #define strzal_poziomy 22
- #define strzelec_pionowy_g 23
- #define strzelec_pionowy_d 24
- #define strzal_pionowy 25
- #define bomba 26
- #define latwy_punkty 5

  ile punktów to poszczególny tryb trudności
- #define sredni_punkty 12

5.3.1 Dokumentacja definicji

5.3.1.1 bezpieczne_pola

```
#define bezpieczne_pola 10
```

5.3.1.2 bomba

```
#define bomba 26
```

5.3.1.3 dlugosc_planszy

```
#define dlugosc_planszy 210
```

5.3.1.4 dol_gracza_l

```
#define dol_gracza_l 13
```

5.3.1.5 dol_gracza_p

```
#define dol_gracza_p 11
```

5.3.1.6 gora_gracza_l

```
#define gora_gracza_l 12
```

5.3.1.7 gora_gracza_p

```
#define gora_gracza_p 10
```

5.3.1.8 gracz_k

```
#define gracz_k 13
```

5.3.1.9 gracz_p

```
#define gracz_p 10
```

5.3.1.10 ilosc_wyswietlonych_pol

```
#define ilosc_wyswietlonych_pol 35
```

5.3.1.11 latwy_punkty

```
#define latwy_punkty 5
```

ile punktów to poszczególny tryb trudności

5.3.1.12 maksymalne_cofniecie_gracza

```
#define maksymalne_cofniecie_gracza 5
```

5.3.1.13 maksymalny_odstep_const

```
#define maksymalny_odstep_const 4
```

5.3.1.14 minimalna_dlugosc_wyspy

```
#define minimalna_dlugosc_wyspy 5
```

5.3.1.15 minimalny_czas_pomiędzy_ruchem_postaci

```
#define minimalny_czas_pomiędzy_ruchem_postaci 30
```

5.3.1.16 minimalny_czas_przesuniecia_strzalu

```
#define minimalny_czas_przesuniecia_strzalu 0.3
```

5.3.1.17 odstep_boczny

```
#define odstep_boczny 30.0f
```

5.3.1.18 odstep_gorny

```
#define odstep_gorny 30.0f
```

5.3.1.19 podloga

```
#define podloga 1
```

5.3.1.20 punkt

```
#define punkt 2
```

5.3.1.21 pusta_przestrzen

```
#define pusta_przestrzen 0
```

5.3.1.22 roznica_wysokosci_pomiedzy_wyspami

```
#define roznica_wysokosci_pomiedzy_wyspami 3
```

5.3.1.23 skok_const

```
#define skok_const 4
```

5.3.1.24 sredni_punkty

```
#define sredni_punkty 12
```

5.3.1.25 strzal_pionowy

```
#define strzal_pionowy 25
```

5.3.1.26 strzal_poziomy

```
#define strzal_poziomy 22
```

5.3.1.27 strzelec_pionowy_d

```
#define strzelec_pionowy_d 24
```

5.3.1.28 strzelec_pionowy_g

```
#define strzelec_pionowy_g 23
```

5.3.1.29 strzelec_poziomy_d

```
#define strzelec_poziomy_d 21
```

5.3.1.30 strzelec_poziomy_g

```
#define strzelec_poziomy_g 20
```

5.3.1.31 wysokosc_planszy

```
#define wysokosc_planszy 16
```

5.3.1.32 zasieg_strzalu_pionowego

```
#define zasieg_strzalu_pionowego 5
```

5.3.1.33 zasieg_strzalu_poziomego

```
#define zasieg_strzalu_poziomego 6
```

5.4 Dokumentacja pliku Gra.cpp

```
#include "Gra.h"
```

5.5 Dokumentacja pliku Gra.h

```
#include "ConstantsAndIncludes.h"
#include "Menu.h"
#include "Plansza.h"
#include "Zapis.h"
#include "Mario.h"
```

Komponenty

- class Gra

5.6 Dokumentacja pliku Mario.cpp

```
#include "Mario.h"
```

5.7 Dokumentacja pliku Mario.h

```
#include "Gra.h"
```

Komponenty

- class Mario

5.8 Dokumentacja pliku MarioSFML.cpp

```
#include "Gra.h"
```

Funkcje

- int main ()

5.8.1 Dokumentacja funkcji

5.8.1.1 main()

```
int main ( )
```

5.9 Dokumentacja pliku Menu.cpp

```
#include "Menu.h"
```

5.10 Dokumentacja pliku Menu.h

```
#include "ConstantsAndIncludes.h"
```

Komponenty

- class Menu

5.11 Dokumentacja pliku Plansza.cpp

```
#include "Plansza.h"
```

5.12 Dokumentacja pliku Plansza.h

```
#include "ConstantsAndIncludes.h"
#include "StrukturyIFunkcje.h"
#include "Przeciwnicy.h"
#include "Menu.h"
```

Komponenty

- class Plansza

5.13 Dokumentacja pliku Przeciwnicy.cpp

```
#include "Przeciwnicy.h"
```

5.14 Dokumentacja pliku Przeciwnicy.h

```
#include "ConstantsAndIncludes.h"
#include "Bomba.h"
#include "Strzelec_pionowy.h"
#include "Strzelec_poziomy.h"
#include "Ruchy.h"
#include "StrukturyIFunkcje.h"
```

Komponenty

- class Przeciwnicy

5.15 Dokumentacja pliku Rekord.cpp

```
#include "Rekord.h"
```

Funkcje

- std::ostream & operator<< (std::ostream &str, const Rekord &rekord)
- std::istream & operator>> (std::istream &str, Rekord &rekord)
- bool operator< (const Rekord &l, const Rekord &p)
- bool operator> (const Rekord &l, const Rekord &p)

5.15.1 Dokumentacja funkcji

5.15.1.1 operator<()

```
bool operator< (
            const Rekord & l,
            const Rekord & p )
```

5.15.1.2 operator<<()

```
std::ostream& operator<< (
            std::ostream & str,
            const Rekord & rekord )
```

5.15.1.3 operator>()

```
bool operator> (
            const Rekord & l,
            const Rekord & p )
```

5.15.1.4 operator>>()

```
std::istream& operator>> (
            std::istream & str,
            Rekord & rekord )
```

5.16 Dokumentacja pliku Rekord.h

```
#include "ConstantsAndIncludes.h"
```

Komponenty

- class Rekord

5.17 Dokumentacja pliku Ruchy.cpp

```
#include "Ruchy.h"
```

5.18 Dokumentacja pliku Ruchy.h

```
#include "ConstantsAndIncludes.h"
```

Komponenty

- class Postac

5.19 Dokumentacja pliku StrukturyIFunkcje.cpp

```
#include "StrukturyIFunkcje.h"
```

Funkcje

- int przesuniecie (int pole, int przes)

5.19.1 Dokumentacja funkcji

5.19.1.1 przesuniecie()

```
int przesuniecie (
            int pole,
            int przes )
```

5.20 Dokumentacja pliku StrukturyIFunkcje.h

```
#include "ConstantsAndIncludes.h"
```

Funkcje

- int przesuniecie (int pole, int przes)

5.20.1 Dokumentacja funkcji

5.20.1.1 przesuniecie()

```
int przesuniecie (
            int pole,
            int przes )
```

5.21 Dokumentacja pliku Strzelec_pionowy.cpp

```
#include "Strzelec_pionowy.h"
#include "StrukturyIFunkcje.h"
```

5.22 Dokumentacja pliku Strzelec_pionowy.h

```
#include "ConstantsAndIncludes.h"
#include "Ruchy.h"
```

Komponenty

- class Strzelec_pionowy

5.23 Dokumentacja pliku Strzelec_poziomy.cpp

```
#include "Strzelec_poziomy.h"
#include "StrukturyIFunkcje.h"
```

5.24 Dokumentacja pliku Strzelec_poziomy.h

```
#include "ConstantsAndIncludes.h"
#include "Ruchy.h"
```

Komponenty

- class Strzelec_poziomy

5.25 Dokumentacja pliku Zapis.cpp

```
#include "Zapis.h"
```

5.26 Dokumentacja pliku Zapis.h

```
#include "ConstantsAndIncludes.h"
#include "Rekord.h"
#include "Menu.h"
#include "Plansza.h"
```

Komponenty

- class Zapis