space_invaders_c++11

Generated by Doxygen 1.9.8

Chapter 1

Space Invaders - C++ Terminal game

Klasyczna gra zręcznościowa **Space Invaders** zaimplementowana w nowoczesnym C++ (standard C++11). Projekt działa w terminalu systemowym (Linux/macOS/Windows) i demonstruje wykorzystanie programowania obiektowego (OOP) oraz zarządzania pamięcią w środowisku tekstowym.

1.1 Zasady gry i Sterowanie

Celem gry jest obrona Ziemi przed inwazją obcych. Musisz zestrzelić wszystkich przeciwników, zanim dotrą do dolnej krawędzi ekranu.

1.2 Klawiszologia

Klawisz	Akcja
A	Ruch w lewo
—	—
D	Ruch w prawo
—	—
SPACJA	Strzał
—	—
Q	Wyjście z gry (Quit)
—	—

1.3 Warunki zwycięstwa i porażki

- **Zwycięstwo:** Zestrzelenie wszystkich obcych na planszy (gra resetuje formację obcych i gra toczy się dalej).
- **Porażka (game Over):** Jeśli jakikolwiek obcy dotknie dolnej krawędzi ekranu (linii obrony gracza).

1.4 Jak uruchomić projekt

Projekt można zbudować na dwa sposoby. Zalecaną metodą jest użycie **CMake**, ale możliwa jest również klasyczna kompilacja bezpośrednio przez **g++**.

1.4.1 Metoda 1: CMake (Zalecana)

Automatycznie zarządza zależnościami i procesem budowania.

1. Upewnij się, że jesteś w głównym katalogu projektu.
2. Utwórz katalog na pliki budowania:
   mkdir build
   cd build
3. Wygeneruj pliki Makefile:
   cmake ..
4. Skompiluj projekt:
   make
5. Uruchom grę:
   ./space_invaders

1.4.2 Metoda 2: Klasyczna kompilacja (g++)

Jeśli chcesz skompilować grę jedną komendą w terminalu:

1. Będąc w folderze z plikami źródłowymi (.cpp i .h), wpisz:
   g++ main.cpp **Game.cpp** (p. 43) **Entities.cpp** (p. 41) **ConsoleUtils.cpp** (p. 37) -o space_invaders -std=c++11 -pthread
   Uwaga: Flaga -pthread jest konieczna do obsługi funkcji usypiania wątku (std::this_thread::sleep_for)._
2. Uruchom grę:
   ./space_invaders

1.5 Jak to działa: Silnik Gry

1.5.1 Podstawowe założenie

Gra opiera się na **ciągłej pętli (Game (**p. **24) Loop)**, która wykonuje się wiele razy na sekundę (ok. 20 klatek/s). Aby uniknąć irytującego migania terminala (flickering), zastosowano technikę "buforowania" – najpierw cały świat jest rysowany do zmiennej w pamięci (bufor tekstowy), a dopiero potem gotowa klatka jest wypisywana na ekran po uprzednim wyczyszczeniu konsoli.

1.5.2 Cykl pojedynczej klatki

Pętla gry realizuje 5 kluczowych kroków w każdej iteracji:

1. **Input (Wejście):**
    - Sprawdzenie, czy klawisz został wciśnięty, w sposób **nieblokujący** (gra nie zatrzymuje się, czekając na reakcję gracza).
2. **Update (Aktualizacja):**
    - Przesunięcie jednostek (gracza, pocisków, obcych).
    - Wykrycie kolizji (czy pocisk trafił obcego?).
    - Logika formacji (zejście obcych w dół przy krawędzi).
    - Usunięcie martwych obiektów z pamięci.
3. **Render (Rysowanie do pamięci):**
    - Stworzenie czystej "kartki" (wektora stringów).
    - Naniesienie symboli wszystkich aktywnych obiektów na tę kartkę.
4. **Display (Wyświetlanie):**
    - Wyczyszczenie ekranu terminala.
    - Wypisanie gotowego bufora i wyniku.
5. **Sleep (Uśpienie):**
    - Krótka pauza (np. 50ms), aby ustabilizować prędkość gry, by nie działała zbyt szybko na nowoczesnych procesorach.

1.6 Architektura i Historia Projektu

Struktura projektu powstawała ewolucyjnie, zaczynając od fundamentów, a kończąc na punkcie wejścia. Poniżej przedstawiono "łańcuch zależności" – kolejność, w jakiej pliki były planowane i tworzone, co odzwierciedla architekturę systemu.

1.6.1 Kolejność warstw (od dołu do góry):

1. **Globals.h (**p. **44)**
    - *Fundament.* Tutaj zdefiniowano stałe (wymiary ekranu) oraz podstawową strukturę matematyczną **Vector2** (p. 34). Wszystkie inne pliki zależą od tego nagłówka.
2. **Entities.h (**p. **41)**
    - *Modele danych.* Definicja klas obiektów. Stworzono klasę bazową **Entity** (p. 19) oraz dziedziczące po niej: **Player** (p. 31), **Alien** (p. 11), **Bullet** (p. 15).
3. **Entities.cpp (**p. **41)**
    - *Logika obiektów.* Implementacja zachowań zdefiniowanych w pliku nagłówkowym (jak obiekty się poruszają, jak rysują się do bufora, jak wykrywają kolizje).
4. **ConsoleUtils.h (**p. **39)**

- *Interfejs systemowy.* Deklaracje funkcji specyficznych dla systemu operacyjnego (ukrywanie kursora, nieblokujące wejście).

5. **ConsoleUtils.cpp (**p. **37)**

   - *Implementacja systemowa.* "Brudna robota" niskopoziomowa. Tu znajduje się kod zależny od OS (Linux termios vs Windows conio.h), odizolowany od reszty gry.

6. **Game.h (**p. **43)**

   - *Plan silnika.* Definicja klasy zarządcy **Game** (p. 24), która przechowuje wskaźniki na obiekty (Player$*$, vector$<$Alien$*>$) i definicje metod sterujących pętlą.

7. **Game.cpp (**p. **43)**

   - *Implementacja silnika.* "Mózg" gry. Tutaj łączą się wszystkie wcześniejsze elementy: obsługa wejścia (ConsoleUtils), zarządzanie obiektami (Entities) i logika pętli głównej.

8. **main.cpp**

   - *Punkt wejścia.* Najwyższa warstwa. Plik jest minimalny – jego jedynym zadaniem jest utworzenie instancji **Game** (p. 24) i uruchomienie metody run().

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

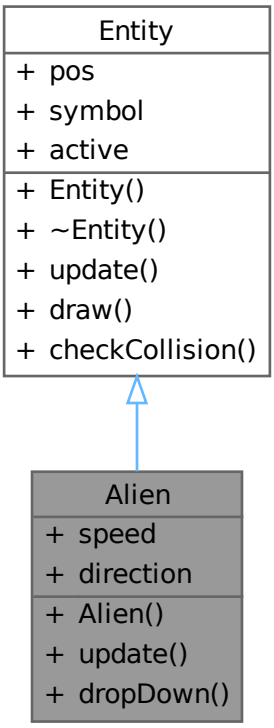
Class Documentation

5.1 Alien Class Reference

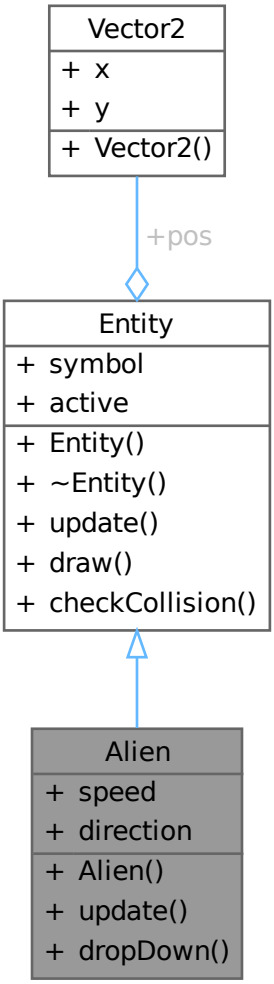
Klasa reprezentująca przeciwnika (Obcego).

`#include <Entities.h>`

Inheritance diagram for Alien:



Collaboration diagram for Alien:



Public Member Functions

- **Alien** (float x, float y)

  Konstruktor Obcego.
- void **update** () override

  Aktualizuje pozycję obcego.
- void **dropDown** ()

  Powoduje zejście obcego o jeden poziom w dół i zmianę kierunku ruchu.

Public Member Functions inherited from Entity

- **Entity** (float x, float y, char s)

  *Konstruktor klasy **Entity** (p. 19).*

- virtual ~**Entity** ()

  Wirtualny destruktor.
- void **draw** (std::vector< std::string > &buffer)

  Rysuje obiekt na wirtualnym buforze ekranu.
- bool **checkCollision** (const **Entity** &other)

  *Sprawdza kolizję z innym obiektem typu **Entity*** (p. *19*).

Public Attributes

- float **speed**

  Prędkość poruszania się w poziomie.
- int **direction**

  Kierunek ruchu poziomego.

Public Attributes inherited from Entity

- **Vector2 pos**

  Pozycja obiektu na ekranie (współrzędne x, y).
- char **symbol**

  Znak ASCII reprezentujący obiekt na konsoli (np. 'A', 'M', '|').
- bool **active**

  Flaga określająca, czy obiekt jest "żywy".

5.1.1 Detailed Description

Klasa reprezentująca przeciwnika (Obcego).

- Porusza się w formacji: poziomo, a po dotarciu do krawędzi schodzi w dół.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Alien()

```
Alien::Alien (
            float x,
            float y )
```

Konstruktor Obcego.

Konstruktor obcego.

Parameters

x	Pozycja startowa X.
y	Pozycja startowa Y.

- Ustawia symbol 'M', domyślną prędkość 0.1f i początkowy kierunek w prawo (1).

5.1.3 Member Function Documentation

5.1.3.1 dropDown()

```
void Alien::dropDown ( )
```

Powoduje zejście obcego o jeden poziom w dół i zmianę kierunku ruchu.

Manewr zejścia w dół.

- Wywoływana przez zarządcę gry (**Game** (p. 24)), gdy formacja dotrze do krawędzi ekranu.
- Wykonywany, gdy formacja dotrze do ściany.

1. Zwiększa Y o 1.0 (zejście niżej).
2. Odwraca kierunek (mnożenie przez -1), aby obcy zaczęli wracać w drugą stronę.

5.1.3.2 update()

```
void Alien::update ( )  [override], [virtual]
```

Aktualizuje pozycję obcego.

Logika automatycznego ruchu obcego.

- Przesuwa obiekt w poziomie zgodnie z aktualną prędkością i kierunkiem.
- Przesuwa obcego w poziomie: x = x + (prędkość $*$ kierunek).

Implements **Entity** (p. 23).

5.1.4 Member Data Documentation

5.1.4.1 direction

```
int Alien::direction
```

Kierunek ruchu poziomego.

- 1: Ruch w prawo.
- -1: Ruch w lewo.

5.1.4.2 speed

```
float Alien::speed
```

Prędkość poruszania się w poziomie.

The documentation for this class was generated from the following files:

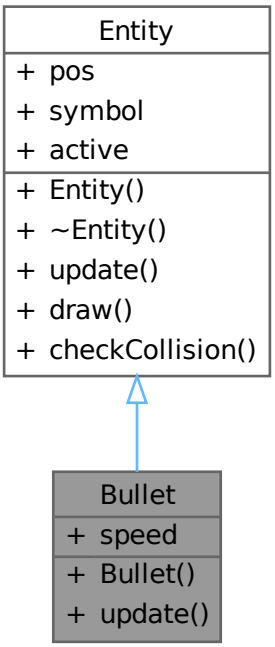
- space_invaders_cpp/ **Entities.h**
- space_invaders_cpp/ **Entities.cpp**

5.2 Bullet Class Reference

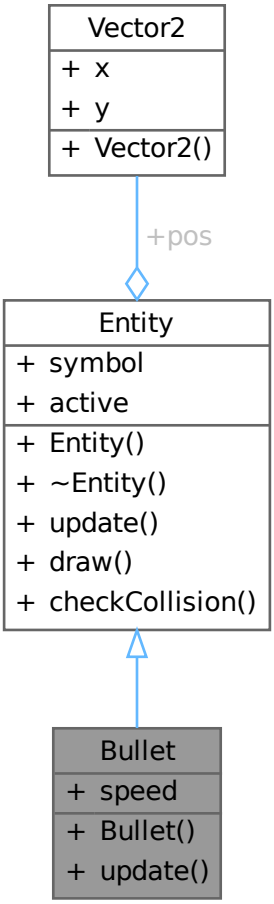
Klasa reprezentująca pocisk.

```
#include <Entities.h>
```

Inheritance diagram for Bullet:



Collaboration diagram for Bullet:



Public Member Functions

- **Bullet** (float x, float y, float s)

  Konstruktor pocisku.
- void **update** () override

  Aktualizuje pozycję pocisku.

Public Member Functions inherited from Entity

- **Entity** (float x, float y, char s)

  *Konstruktor klasy **Entity** (p. 19).*
- virtual ~**Entity** ()

  Wirtualny destruktor.
- void **draw** (std::vector< std::string > &buffer)

  Rysuje obiekt na wirtualnym buforze ekranu.
- bool **checkCollision** (const **Entity** &other)

  *Sprawdza kolizję z innym obiektem typu **Entity** (p. 19).*

Public Attributes

- float **speed**

  Prędkość pionowa pocisku.

Public Attributes inherited from Entity

- **Vector2 pos**

  Pozycja obiektu na ekranie (współrzędne x, y).
- char **symbol**

  Znak ASCII reprezentujący obiekt na konsoli (np. 'A', 'M', '|').
- bool **active**

  Flaga określająca, czy obiekt jest "żywy".

5.2.1 Detailed Description

Klasa reprezentująca pocisk.

- Dziedziczy po klasie **Entity** (p. 19). Może poruszać się w górę lub w dół (zależnie od prędkości).

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Bullet()

```
Bullet::Bullet (
            float x,
            float y,
            float s )
```

Konstruktor pocisku.

Parameters

x	Pozycja startowa X.
y	Pozycja startowa Y.
s	Prędkość (speed) - określa kierunek i szybkość lotu.

- Ustawia symbol na '|' (pionowa kreska) i przypisuje prędkość.

Parameters

x	Pozycja startowa X.
y	Pozycja startowa Y.
s	Prędkość (ujemna leci w górę, dodatnia w dół).

-

5.2.3 Member Function Documentation

5.2.3.1 update()

```
void Bullet::update ( )  [override], [virtual]
```

Aktualizuje pozycję pocisku.

Logika ruchu pocisku.

- Przesuwa pocisk w pionie. Jeśli wyleci poza ekran, ustawia flagę active na false.
- Przesuwa pocisk w pionie o wartość `speed`. Sprawdza, czy pocisk wyleciał poza ekran (góra/dół). Jeśli tak, dezaktywuje go (active = false), co pozwoli na usunięcie go z pamięci w **Game::update** (p. 29).

Implements **Entity** (p. 23).

5.2.4 Member Data Documentation

5.2.4.1 speed

```
float Bullet::speed
```

Prędkość pionowa pocisku.

- Wartość ujemna oznacza ruch w górę (strzał gracza).
- Wartość dodatnia oznacza ruch w dół (strzał obcego).

The documentation for this class was generated from the following files:

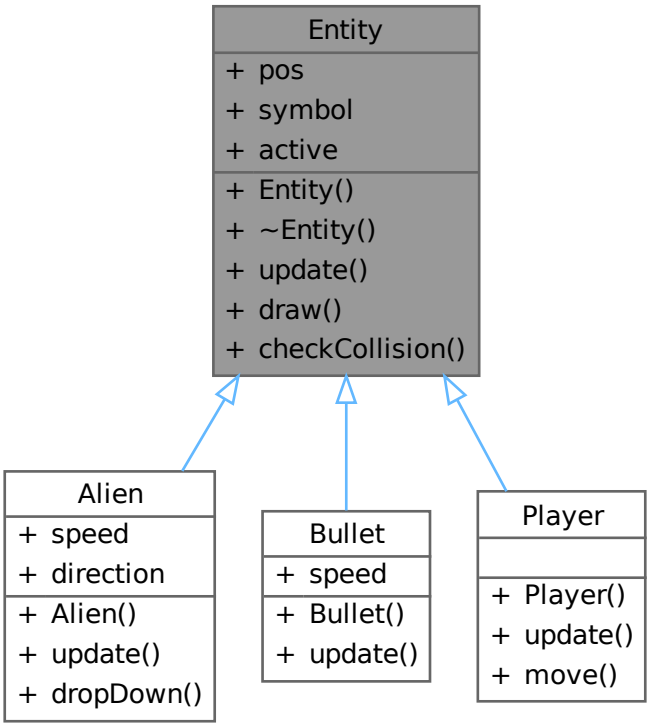
- space_invaders_cpp/ **Entities.h**
- space_invaders_cpp/ **Entities.cpp**

5.3 Entity Class Reference

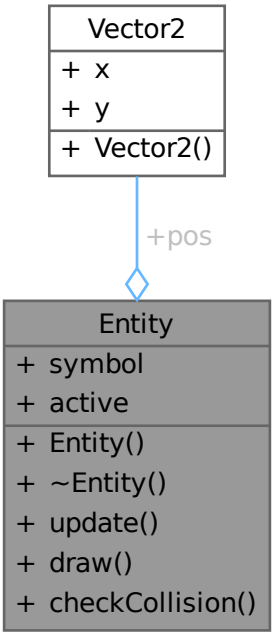
abstrakcyjna klasa bazowa reprezentująca każdy obiekt w grze.

`#include <Entities.h>`

Inheritance diagram for Entity:



Collaboration diagram for Entity:



Public Member Functions

- **Entity** (float x, float y, char s)

  *Konstruktor klasy **Entity** (p. 19).*
- virtual ~**Entity** ()

  Wirtualny destruktor.
- virtual void **update** ()=0

  Czysto wirtualna metoda aktualizacji stanu obiektu.
- void **draw** (std::vector< std::string > &buffer)

  Rysuje obiekt na wirtualnym buforze ekranu.
- bool **checkCollision** (const **Entity** &other)

  *Sprawdza kolizję z innym obiektem typu **Entity** (p. 19).*

Public Attributes

- **Vector2 pos**

  Pozycja obiektu na ekranie (współrzędne x, y).
- char **symbol**

  Znak ASCII reprezentujący obiekt na konsoli (np. 'A', 'M', '|').
- bool **active**

  Flaga określająca, czy obiekt jest "żywy".

5.3.1 Detailed Description

abstrakcyjna klasa bazowa reprezentująca każdy obiekt w grze.

Klasa **Entity** (p. 19) definiuje wspólny interfejs i właściwości dla wszystkich bytów występujących na planszy (Gracza, Obcych, Pocisków). Zawiera informacje o pozycji, wyglądzie (symbolu) oraz stanie aktywności.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Entity()

```
Entity::Entity (
            float x,
            float y,
            char s )
```

Konstruktor klasy **Entity** (p. 19).

Konstruktor klasy bazowej **Entity** (p. 19).

Parameters

x	Początkowa współrzędna X.
y	Początkowa współrzędna Y.
s	Znak (symbol) reprezentujący obiekt.

- Inicjalizuje podstawowe parametry wspólne dla wszystkich obiektów. Używa listy inicjalizacyjnej dla optymalizacji.

Parameters

x	Współrzędna początkowa X.
y	Współrzędna początkowa Y.
s	Znak (symbol), który będzie reprezentował obiekt.

-

5.3.2.2 ∼Entity()

```
Entity::∼Entity ( )  [virtual]
```

Wirtualny destruktor.

- Konieczny dla bezpiecznego usuwania obiektów przez wskaźnik na klasę bazową.
- Puste ciało, ale niezbędne dla poprawności polimorfizmu w C++.

5.3.3 Member Function Documentation

5.3.3.1 checkCollision()

```
bool Entity::checkCollision (
            const  Entity & other )
```

Sprawdza kolizję z innym obiektem typu **Entity** (p. 19).

Sprawdza kolizję metodą prostokątów (AABB) o wielkości 1x1.

- Wykorzystuje proste sprawdzanie odległości (AABB - Axis-Aligned Bounding Box) o wymiarach 1x1 jednostka.

Parameters

other	Referencja do drugiego obiektu, z którym sprawdzamy kolizję.

Returns

true Jeśli obiekty nachodzą na siebie.
false Jeśli brak kolizji lub któryś z obiektów jest nieaktywny.

- Oblicza różnicę odległości w osi X i Y po zrzutowaniu na liczby całkowite. Jeśli oba obiekty zajmują tę samą "kratkę" w terminalu, następuje kolizja.

Parameters

other	Obiekt, z którym sprawdzamy kolizję.

Returns

- true Jeśli odległość < 1 w obu osiach.

5.3.3.2 draw()

```
void Entity::draw (
            std::vector< std::string > & buffer )
```

Rysuje obiekt na wirtualnym buforze ekranu.

Wstawia symbol obiektu do bufora ekranu.

- Metoda sprawdza, czy obiekt znajduje się w granicach ekranu, a następnie wstawia jego symbol w odpowiednie miejsce w dwuwymiarowej tablicy znaków.

Parameters

buffer	Referencja do bufora ekranu (wektor stringów).

- Metoda sprawdza, czy obiekt znajduje się fizycznie w granicach ekranu (0 <= x < SCREEN_WIDTH oraz 0

<= y < SCREEN_HEIGHT). Jeśli tak, wpisuje znak `symbol` pod odpowiednie indeksy tablicy `buffer`.

Parameters

buffer	Referencja do wektora stringów reprezentującego klatkę obrazu.

-

5.3.3.3 update()

```
virtual void Entity::update ( )  [pure virtual]
```

Czysto wirtualna metoda aktualizacji stanu obiektu.

- Każda klasa pochodna musi zaimplementować własną logikę ruchu/zachowania.

Implemented in **Bullet** (p. 18), **Player** (p. 34), and **Alien** (p. 14).

5.3.4 Member Data Documentation

5.3.4.1 active

```
bool Entity::active
```

Flaga określająca, czy obiekt jest "żywy".

- Jeśli active == false, obiekt zostanie usunięty z pamięci przez silnik gry w najbliższej fazie czyszczenia.

5.3.4.2 pos

```
Vector2 Entity::pos
```

Pozycja obiektu na ekranie (współrzędne x, y).

5.3.4.3 symbol

```
char Entity::symbol
```

Znak ASCII reprezentujący obiekt na konsoli (np. 'A', 'M', '|').

The documentation for this class was generated from the following files:

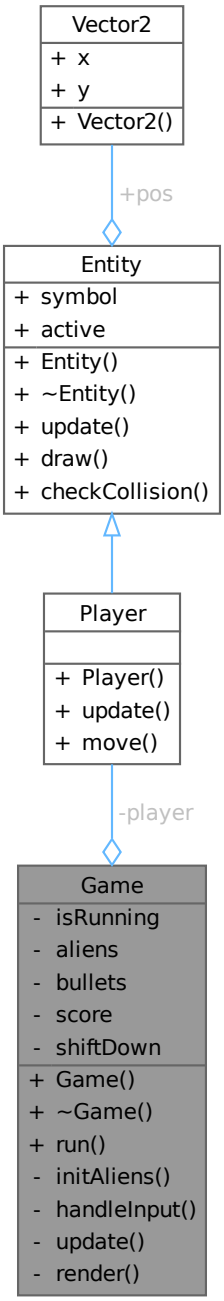
- space_invaders_cpp/ **Entities.h**
- space_invaders_cpp/ **Entities.cpp**

5.4 Game Class Reference

Główna klasa silnika gry "Space Invaders".

`#include <Game.h>`

Collaboration diagram for Game:



Public Member Functions

- **Game** ()

  Implementacja logiki silnika gry.
- ∼**Game** ()

  *Destruktor klasy **Game** (p. 24).*
- void **run** ()

  Uruchamia główną pętlę gry.

Private Member Functions

- void **initAliens** ()

  Inicjalizuje lub resetuje formację obcych na planszy.
- void **handleInput** ()

  Obsługuje wejście od użytkownika w sposób nieblokujący.
- void **update** ()

  Aktualizuje logikę gry (fizykę i zasady) dla jednej klatki.
- void **render** ()

  Rysuje aktualny stan gry na konsoli.

Private Attributes

- bool **isRunning**

  Flaga kontrolująca działanie głównej pętli gry.
- **Player** ∗ **player**

  Wskaźnik na obiekt gracza.
- std::vector< **Alien** ∗ > **aliens**

  Kontener przechowujący wskaźniki na wszystkich żywych obcych.
- std::vector< **Bullet** ∗ > **bullets**

  Kontener przechowujący wskaźniki na wszystkie aktywne pociski.
- int **score**

  Aktualny wynik punktowy gracza.
- bool **shiftDown**

  Flaga pomocnicza dla logiki poruszania się obcych.

5.4.1 Detailed Description

Główna klasa silnika gry "Space Invaders".

Klasa **Game** (p. 24) działa jako zarządca (manager) całej aplikacji. Jest odpowiedzialna za:

- Uruchomienie i utrzymanie głównej pętli gry.
- Przechowywanie stanu gry (punkty, czy gra działa).
- Zarządzanie cyklem życia wszystkich obiektów (Gracz, Obcy, Pociski).
- Koordynowanie logiki (aktualizacje, kolizje) oraz wyświetlania (renderowanie).

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Game()

`Game::Game ( )`

Implementacja logiki silnika gry.

Plik ten zawiera definicje metod klasy **Game** (p. 24), które odpowiadają za:

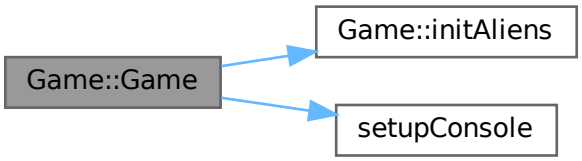
- Inicjalizację zasobów (konstruktor).
- Zarządzanie cyklem życia obiektów (destruktor).
- Obsługę wejścia (klawiatura).
- Logikę matematyczną i fizykę gry (update).
- Rysowanie stanu gry na ekranie (render).
- Główną pętlę sterującą czasem gry (run).

Konstruktor klasy **Game** (p. 24).

Inicjalizuje stan początkowy gry:

1. Ustawia flagę isRunning na true.
2. Zeruje wynik (score).
3. Konfiguruje konsolę (ukrywa kursor).
4. Tworzy obiekt gracza na stercie.
5. Inicjalizuje formację obcych.

Here is the call graph for this function:



5.4.2.2 ~Game()

`Game::~Game ( )`

Destruktor klasy **Game** (p. 24).

Sprząta po zakończeniu gry, zwalniając pamięć zaalokowaną dynamicznie. Usuwa gracza, wszystkich pozostałych obcych oraz wszystkie aktywne pociski. Na końcu przywraca widoczność kursora w terminalu.

5.4.3 Member Function Documentation

5.4.3.1 handleInput()

```
void Game::handleInput ( )  [private]
```

Obsługuje wejście od użytkownika w sposób nieblokujący.

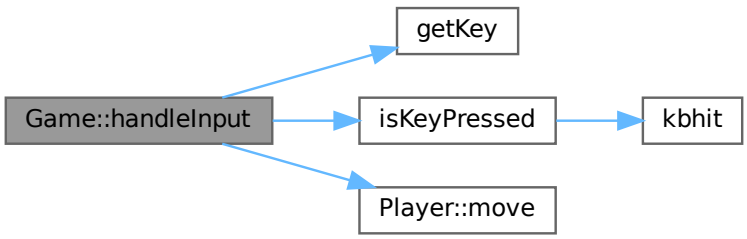
Obsługuje sterowanie grą.

- Sprawdza stan klawiatury i podejmuje akcje:
- 'a'/'d': Poruszanie graczem.
- ' ' (spacja): Wystrzelenie pocisku.
- 'q': Wyjście z gry.

Sprawdza w sposób nieblokujący, czy użytkownik wcisnął klawisz.

- 'q': Kończy grę (isRunning = false).
- 'a': Przesuwa gracza w lewo.
- 'd': Przesuwa gracza w prawo.
- ' ': Tworzy nowy pocisk na pozycji gracza, lecący w górę.

Here is the call graph for this function:



5.4.3.2 initAliens()

```
void Game::initAliens ( )  [private]
```

Inicjalizuje lub resetuje formację obcych na planszy.

Tworzy początkową formację obcych.

- Tworzy siatkę obiektów **Alien** (p. 11) i dodaje je do wektora aliens. Wywoływana w konstruktorze oraz po wyczyszczeniu poziomu.

Generuje siatkę przeciwników (4 rzędy po 8 kolumn). Obcy są rozmieszczani w stałych odstępach, zaczynając od pozycji (5, 2). Nowe obiekty są dodawane do wektora `aliens`.

5.4.3.3 render()

```
void Game::render ( )  [private]
```

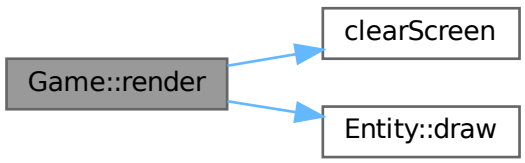
Rysuje aktualny stan gry na konsoli.

Rysuje klatkę gry.

- Tworzy bufor znakowy, wypełnia go symbolami obiektów, czyści ekran i wyświetla gotową klatkę wraz z wynikiem i ramką.

1. Tworzy pusty bufor znakowy (vector stringów).
2. Prosi wszystkie obiekty (gracza, obcych, pociski) o narysowanie się na buforze.
3. Czyści ekran konsoli.
4. Wyświetla interfejs (wynik, ramki) oraz zawartość bufora.

Here is the call graph for this function:



5.4.3.4 run()

```
void Game::run ( )
```

Uruchamia główną pętlę gry.

Główna pętla gry.

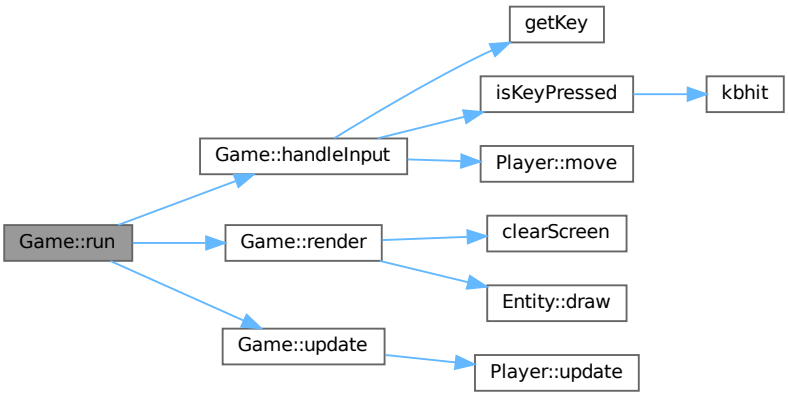
- Metoda ta sekwencyjnie wywołuje **handleInput()** (p. 27), **update()** (p. 29) i **render()** (p. 27) w nieskończonej pętli (dopóki isRunning jest true), kontrolując jednocześnie szybkość działania gry (sleep).

Utrzymuje działanie gry dopóki isRunning jest true. W każdej iteracji:

1. Pobiera wejście (handleInput).
2. Aktualizuje stan świata (update).
3. Rysuje świat (render).

4. Usypia wątek na 50ms, aby uzyskać stałe ∼20 klatek na sekundę (FPS).

Here is the call graph for this function:



5.4.3.5 update()

```
void Game::update ( )  [private]
```

Aktualizuje logikę gry (fizykę i zasady) dla jednej klatki.

Główna funkcja aktualizująca stan gry (fizykę).

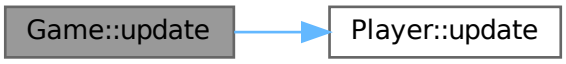
- Metoda ta wykonuje szereg zadań:

1. Aktualizuje pozycje gracza, pocisków i obcych.
2. Zarządza logiką "zejścia w dół" obcych przy krawędziach.
3. Wykrywa i obsługuje kolizje (pocisk vs obcy).
4. Usuwa nieaktywne obiekty z pamięci.
5. Sprawdza warunki końca gry (**Game** (p. 24) Over).

Wykonuje się w każdej klatce i realizuje następujące kroki:

1. Aktualizuje pozycję gracza i wszystkich pocisków.
2. Aktualizuje pozycję obcych i wykrywa, czy dotarli do krawędzi ekranu. Jeśli tak, ustawia flagę `shiftDown`.
3. Jeśli `shiftDown` jest true, obniża całą formację obcych.
4. Sprawdza kolizje pocisków z obcymi. Trafienie usuwa oba obiekty i dodaje punkty.
5. Usuwa nieaktywne (zniszczone lub wyleciałe poza ekran) obiekty z pamięci (std::remove_if + delete).
6. Sprawdza warunki końca gry (brak obcych -> respawn, obcy na dole -> **Game** (p. 24) Over).

Here is the call graph for this function:



5.4.4 Member Data Documentation

5.4.4.1 aliens

`std::vector< Alien*> Game::aliens [private]`

Kontener przechowujący wskaźniki na wszystkich żywych obcych.

- Użycie std::vector pozwala na dynamiczne usuwanie zestrzelonych przeciwników.

5.4.4.2 bullets

`std::vector< Bullet*> Game::bullets [private]`

Kontener przechowujący wskaźniki na wszystkie aktywne pociski.

- Zawiera zarówno pociski wystrzelone przez gracza, jak i (opcjonalnie) przez obcych.

5.4.4.3 isRunning

`bool Game::isRunning [private]`

Flaga kontrolująca działanie głównej pętli gry.

- Jeśli ustawiona na false (np. po wciśnięciu 'q' lub przegranej), metoda **run()** (p. 28) zakończy działanie.

5.4.4.4 player

`Player* Game::player [private]`

Wskaźnik na obiekt gracza.

- Obiekt jest tworzony dynamicznie w konstruktorze i usuwany w destruktorze.

5.4.4.5 score

```
int Game::score  [private]
```

Aktualny wynik punktowy gracza.

5.4.4.6 shiftDown

```
bool Game::shiftDown  [private]
```

Flaga pomocnicza dla logiki poruszania się obcych.

- Ustawiana na true w metodzie **update()** (p. 29), jeśli którykolwiek z obcych dotknie bocznej krawędzi ekranu. Powoduje to przesunięcie całej grupy w dół.

The documentation for this class was generated from the following files:

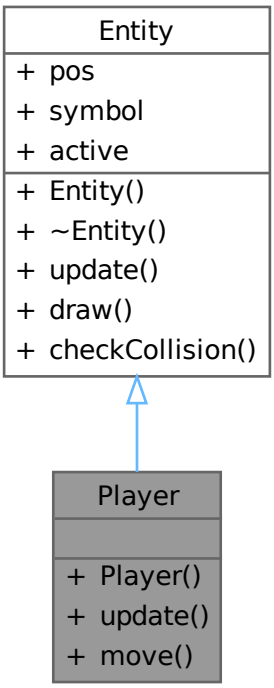
- space_invaders_cpp/ **Game.h**
- space_invaders_cpp/ **Game.cpp**

5.5 Player Class Reference

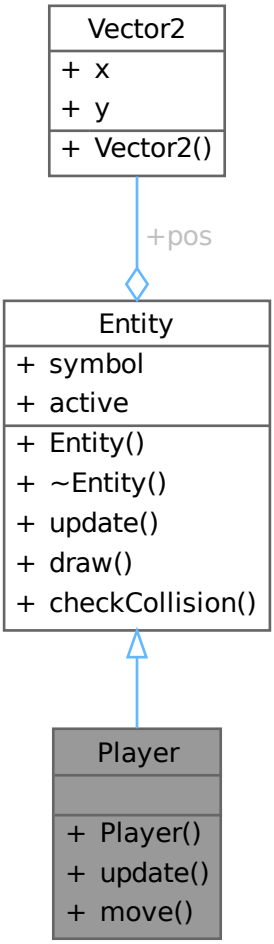
Klasa reprezentująca statek gracza.

```
#include <Entities.h>
```

Inheritance diagram for Player:



Collaboration diagram for Player:



Public Member Functions

- **Player** ()

  Konstruktor gracza.
- void **update** () override

  Aktualizuje stan gracza.
- void **move** (float dir)

  Przesuwa gracza o zadaną wartość.

Public Member Functions inherited from Entity

- **Entity** (float x, float y, char s)

  *Konstruktor klasy **Entity** (p. 19).*
- virtual ~**Entity** ()

Wirtualny destruktor.

- void **draw** (std::vector< std::string > &buffer)

  Rysuje obiekt na wirtualnym buforze ekranu.

- bool **checkCollision** (const **Entity** &other)

  *Sprawdza kolizję z innym obiektem typu **Entity** (p. 19).*

Additional Inherited Members

Public Attributes inherited from Entity

- **Vector2 pos**

  Pozycja obiektu na ekranie (współrzędne x, y).

- char **symbol**

  Znak ASCII reprezentujący obiekt na konsoli (np. 'A', 'M', '|').

- bool **active**

  Flaga określająca, czy obiekt jest "żywy".

5.5.1 Detailed Description

Klasa reprezentująca statek gracza.

- Sterowana przez użytkownika, porusza się tylko w poziomie na dole ekranu.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Player()

```
Player::Player ( )
```

Konstruktor gracza.

- Ustawia gracza w domyślnej pozycji startowej (dół ekranu, środek).
- Ustawia gracza na środku dolnej części ekranu z symbolem 'A'. SCREEN_WIDTH / 2 to środek osi X. SCREEN_HEIGHT - 2 to bezpieczna odległość od dolnej krawędzi.

5.5.3 Member Function Documentation

5.5.3.1 move()

```
void Player::move (
            float dir )
```

Przesuwa gracza o zadaną wartość.

Przesuwa gracza w poziomie.

Parameters

dir	Wartość przesunięcia (np. -2.0f dla lewo, 2.0f dla prawo).

Parameters

dir	Wartość przesunięcia (dodatnia w prawo, ujemna w lewo).

-

5.5.3.2 update()

```
void Player::update ( )  [override], [virtual]
```

Aktualizuje stan gracza.

Aktualizacja stanu gracza - ograniczenie ruchu (Clamping).

- Głównie odpowiada za pilnowanie granic ekranu (clamping), aby gracz nie wyjechał poza planszę.
- Gracz nie porusza się tutaj sam z siebie (ruch sterowany jest w handleInput), ale ta metoda upewnia się, że gracz nie wyjedzie poza lewą (x $<$ 1) lub prawą (x $>$ WIDTH - 2) krawędź ekranu.

Implements **Entity** (p. 23).

The documentation for this class was generated from the following files:

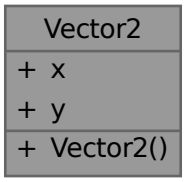
- space_invaders_cpp/ **Entities.h**
- space_invaders_cpp/ **Entities.cpp**

5.6 Vector2 Struct Reference

Prosta struktura reprezentująca wektor dwuwymiarowy lub punkt.

```
#include <Globals.h>
```

Collaboration diagram for Vector2:



Public Member Functions

- **Vector2** (float _x=0, float _y=0)

  *Konstruktor struktury **Vector2** (p. 34).*

Public Attributes

- float **x**

  Współrzędna pozioma (X).
- float **y**

  Współrzędna pionowa (Y).

5.6.1 Detailed Description

Prosta struktura reprezentująca wektor dwuwymiarowy lub punkt.

Używana do przechowywania pozycji obiektów (Gracza, Obcych, Pocisków) w przestrzeni gry. Używa typu float dla płynności ruchu, choć przy renderowaniu wartości są rzutowane na int.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Vector2()

```
Vector2::Vector2 (
            float _x = 0,
            float _y = 0 )  [inline]
```

Konstruktor struktury **Vector2** (p. 34).

Pozwala na szybką inicjalizację wektora.

Parameters

↩ _↩ *x*	Wartość początkowa dla współrzędnej X (domyślnie 0).
↩ _↩ *y*	Wartość początkowa dla współrzędnej Y (domyślnie 0).

5.6.3 Member Data Documentation

5.6.3.1 x

```
float Vector2::x
```

Współrzędna pozioma (X).

5.6.3.2 y

```
float Vector2::y
```

Współrzędna pionowa (Y).

The documentation for this struct was generated from the following file:

- space_invaders_cpp/ **Globals.h**

Chapter 6

File Documentation

6.1 space_invaders_cpp/ConsoleUtils.cpp File Reference

```
#include "ConsoleUtils.h"
#include <iostream>
#include <cstdio>
#include <termios.h>
#include <unistd.h>
#include <fcntl.h>
```

Functions

- void **setupConsole** ()

  Konfiguruje początkowe ustawienia konsoli.
- int **kbhit** (void)

  Sprawdza w sposób nieblokujący, czy naciśnięto klawisz klawiatury.
- bool **isKeyPressed** ()

  Sprawdza, czy został wciśnięty klawisz, nie zatrzymując programu.
- char **getKey** ()

  Pobiera ostatni wciśnięty znak z bufora klawiatury.
- void **clearScreen** ()

  Czyści zawartość ekranu konsoli.

6.1.1 Function Documentation

6.1.1.1 clearScreen()

```
void clearScreen ( )
```

Czyści zawartość ekranu konsoli.

Usuwa wszystkie znaki z widocznego obszaru terminala i ustawia kursor w lewym górnym rogu (pozycja 0,0). Używana w każdej klatce przed narysowaniem nowej sceny gry.

6.1.1.2 getKey()

```
char getKey ( )
```

Pobiera ostatni wciśnięty znak z bufora klawiatury.

Funkcja ta powinna być wywoływana zazwyczaj po upewnieniu się przez **isKeyPressed()** (p. 40), że znak jest dostępny.

Returns

char Znak odpowiadający wciśniętemu klawiszowi.

6.1.1.3 isKeyPressed()

```
bool isKeyPressed ( )
```

Sprawdza, czy został wciśnięty klawisz, nie zatrzymując programu.

Jest to funkcja typu "non-blocking". W przeciwieństwie do standardowego std::cin, ta funkcja natychmiast zwraca wynik.

Returns

true Jeśli w buforze klawiatury znajduje się oczekujący znak.

false Jeśli bufor jest pusty.

Here is the call graph for this function:

6.1.1.4 kbhit()

```
int kbhit (
            void  )
```

Sprawdza w sposób nieblokujący, czy naciśnięto klawisz klawiatury.

Funkcja ta emuluje zachowanie znanej z systemu Windows funkcji _kbhit() w środowiskach zgodnych z POSIX (Linux, macOS). Pozwala sprawdzić stan bufora wejścia bez zatrzymywania działania programu.

Działanie funkcji opiera się na tymczasowej modyfikacji ustawień terminala:

1. Pobiera obecne ustawienia terminala (struktura termios).
2. Wyłącza tryb kanoniczny (ICANON), aby znaki były dostępne natychmiast bez konieczności wciskania Enter.
3. Wyłącza echo (ECHO), aby testowany znak nie pojawiał się na ekranie.
4. Ustawia deskryptor wejścia (STDIN) w tryb nieblokujący (O_NONBLOCK), dzięki czemu próba odczytu nie zatrzyma programu, jeśli bufor jest pusty.
5. Próbuje pobrać jeden znak funkcją getchar().
6. Natychmiast przywraca oryginalne ustawienia terminala.
7. Jeśli znak został pobrany, jest zwracany do bufora (ungetc), aby kolejna funkcja (np. getKey) mogła go poprawnie odczytać.

Returns

int

- 1 (true): Jeśli w buforze wejścia znajduje się znak (klawisz został wciśnięty).
- 0 (false): Jeśli bufor jest pusty (brak wciśniętego klawisza).

Note

Funkcja jest przeznaczona dla systemów uniksowych (Linux, macOS). Wymaga bibliotek <termios.h>, <unistd.h> oraz <fcntl.h>.

6.1.1.5 setupConsole()

```
void setupConsole ( )
```

Konfiguruje początkowe ustawienia konsoli.

Funkcja ta powinna być wywołana na początku działania programu. Jej głównym zadaniem jest przygotowanie środowiska do wyświetlania gry, np. poprzez ukrycie migającego kursora, co zapobiega irytującym efektom wizualnym podczas częstego odświeżania ekranu.

6.2 space_invaders_cpp/ConsoleUtils.h File Reference

Deklaracje funkcji narzędziowych do obsługi konsoli systemowej.

Functions

- void **setupConsole** ()
  Konfiguruje początkowe ustawienia konsoli.
- bool **isKeyPressed** ()
  Sprawdza, czy został wciśnięty klawisz, nie zatrzymując programu.
- char **getKey** ()
  Pobiera ostatni wciśnięty znak z bufora klawiatury.
- void **clearScreen** ()
  Czyści zawartość ekranu konsoli.

6.2.1 Detailed Description

Deklaracje funkcji narzędziowych do obsługi konsoli systemowej.

Plik zawiera interfejs dla funkcji systemowych (zależnych od OS), które pozwalają na:

- Konfigurację konsoli (np. ukrywanie kursora).
- Nieblokującą obsługę wejścia klawiatury (kluczowe dla pętli gry).
- Czyszczenie ekranu. Implementacje tych funkcji znajdują się w pliku **ConsoleUtils.cpp** (p. 37) i różnią się w zależności od systemu operacyjnego (Windows vs Linux/macOS).

6.2.2 Function Documentation

6.2.2.1 clearScreen()

```
void clearScreen ( )
```

Czyści zawartość ekranu konsoli.

Usuwa wszystkie znaki z widocznego obszaru terminala i ustawia kursor w lewym górnym rogu (pozycja 0,0). Używana w każdej klatce przed narysowaniem nowej sceny gry.

6.2.2.2 getKey()

```
char getKey ( )
```

Pobiera ostatni wciśnięty znak z bufora klawiatury.

Funkcja ta powinna być wywoływana zazwyczaj po upewnieniu się przez **isKeyPressed()** (p. 40), że znak jest dostępny.

Returns

char Znak odpowiadający wciśniętemu klawiszowi.

6.2.2.3 isKeyPressed()

```
bool isKeyPressed ( )
```

Sprawdza, czy został wciśnięty klawisz, nie zatrzymując programu.

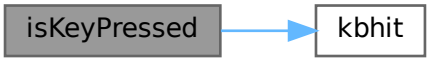
Jest to funkcja typu "non-blocking". W przeciwieństwie do standardowego std::cin, ta funkcja natychmiast zwraca wynik.

Returns

true Jeśli w buforze klawiatury znajduje się oczekujący znak.

false Jeśli bufor jest pusty.

Here is the call graph for this function:



6.2.2.4 setupConsole()

```
void setupConsole ( )
```

Konfiguruje początkowe ustawienia konsoli.

Funkcja ta powinna być wywołana na początku działania programu. Jej głównym zadaniem jest przygotowanie środowiska do wyświetlania gry, np. poprzez ukrycie migającego kursora, co zapobiega irytującym efektom wizualnym podczas częstego odświeżania ekranu.

6.3 ConsoleUtils.h

Go to the documentation of this file.

```
00001 #ifndef CONSOLE_UTILS_H
00002 #define CONSOLE_UTILS_H
00003
00028 void setupConsole();
00029
00039 bool isKeyPressed();
00040
00049 char getKey();
00050
00058 void clearScreen();
00059
00060 #endif // CONSOLE_UTILS_H
00061
00062
00063
```

6.4 space_invaders_cpp/Entities.cpp File Reference

Implementacja metod klas zdefiniowanych w **Entities.h** (p. 41).

```
#include "Entities.h"
```

6.4.1 Detailed Description

Implementacja metod klas zdefiniowanych w **Entities.h** (p. 41).

Plik ten zawiera kod wykonawczy dla logiki obiektów gry. Znajdują się tutaj implementacje metod odpowiedzialnych za:

- Rysowanie obiektów do bufora (**Entity::draw** (p. 22)).
- Wykrywanie kolizji (**Entity::checkCollision** (p. 22)).
- Aktualizację pozycji i stanu (update) dla Gracza, Obcych i Pocisków.

6.5 space_invaders_cpp/Entities.h File Reference

Definicje klas obiektów gry (Gracz, Obcy, Pociski).

```
#include "Globals.h"
#include <vector>
#include <string>
#include <cmath>
```

Classes

- class **Entity**

  abstrakcyjna klasa bazowa reprezentująca każdy obiekt w grze.
- class **Bullet**

  Klasa reprezentująca pocisk.
- class **Player**

  Klasa reprezentująca statek gracza.
- class **Alien**

  Klasa reprezentująca przeciwnika (Obcego).

6.5.1 Detailed Description

Definicje klas obiektów gry (Gracz, Obcy, Pociski).

6.6 Entities.h

Go to the documentation of this file.

```
00001 #ifndef ENTITIES_H
00002 #define ENTITIES_H
00003
00004 #include "Globals.h"
00005 #include <vector>
00006 #include <string>
00007 #include <cmath>
00008
00021 class Entity {
00022 public:
00026     Vector2 pos;
00027
00031     char symbol;
00032
00038     bool active;
00039
00046     Entity(float x, float y, char s);
00047
00052     virtual ~Entity();
00053
00058     virtual void update() = 0;
00059
00066     void draw(std::vector<std::string>& buffer);
00067
00076     bool checkCollision(const Entity& other);
00077 };
00078
00083 class Bullet : public Entity {
00084 public:
00090     float speed;
00091
00098     Bullet(float x, float y, float s);
00099
00104     void update() override;
00105 };
00106
00111 class Player : public Entity {
00112 public:
00117     Player();
00118
00123     void update() override;
00124
00129     void move(float dir);
00130 };
00131
00136 class Alien : public Entity {
00137 public:
00141     float speed;
00142
00148     int direction;
00149
00155     Alien(float x, float y);
```

```
00156
00161     void update() override;
00162
00167     void dropDown();
00168 };
00169
00170 #endif // ENTITIES_H
```

6.7 space_invaders_cpp/Game.cpp File Reference

```
#include "Game.h"
#include "ConsoleUtils.h"
#include <iostream>
#include <algorithm>
#include <thread>
#include <chrono>
```

6.8 space_invaders_cpp/Game.h File Reference

Definicja głównej klasy gry.

```
#include <vector>
#include "Entities.h"
```

Classes

- class **Game**

  Główna klasa silnika gry "Space Invaders".

6.8.1 Detailed Description

Definicja głównej klasy gry.

6.9 Game.h

Go to the documentation of this file.

```
00001 #ifndef GAME_H
00002 #define GAME_H
00003
00004 #include <vector>
00005 #include "Entities.h"
00006
00022 class Game
00023 {
00024 private:
00030     bool isRunning;
00031
00036     Player *player;
00037
00042     std::vector<Alien*> aliens;
00043
00048     std::vector<Bullet*> bullets;
00049
```

```
00053     int score;
00054
00060     bool shiftDown; // Flaga dla logiki ruchu obcych
00061
00067     void initAliens();
00068
00076     void handleInput();
00077
00087     void update();
00088
00094     void render();
00095
00096 public:
00097     Game();
00098     ~Game();
00099
00106     void run();
00107 };
00108
00109 #endif // GAME_H
```

6.10 space_invaders_cpp/Globals.h File Reference

Definicje stałych globalnych oraz podstawowych struktur danych.

Classes

- struct **Vector2**

  Prosta struktura reprezentująca wektor dwuwymiarowy lub punkt.

Variables

- const int **SCREEN_WIDTH** = 40

  Szerokość ekranu gry (w znakach konsoli).
- const int **SCREEN_HEIGHT** = 20

  Wysokość ekranu gry (w wierszach konsoli).

6.10.1 Detailed Description

Definicje stałych globalnych oraz podstawowych struktur danych.

Plik ten zawiera elementy współdzielone przez cały projekt, takie jak wymiary ekranu gry oraz strukturę wektora 2D używaną do określania pozycji obiektów

6.10.2 Variable Documentation

6.10.2.1 SCREEN_HEIGHT

```
const int SCREEN_HEIGHT = 20
```

Wysokość ekranu gry (w wierszach konsoli).

Określa liczbę wierszy w buforze i na ekranie.

6.10.2.2 SCREEN_WIDTH

```
const int SCREEN_WIDTH = 40
```

Szerokość ekranu gry (w znakach konsoli).

Określa liczbę kolumn w buforze i na ekranie.

6.11 Globals.h

Go to the documentation of this file.

```
00001 #ifndef GLOBALS_H
00002 #define GLOBALS_H
00003
00020 const int SCREEN_WIDTH = 40;
00021
00027 const int SCREEN_HEIGHT = 20;
00028
00036 struct Vector2 {
00040     float x;
00041
00045     float y;
00046
00055     Vector2(float _x = 0, float _y = 0) : x(_x), y(_y) {}
00056 };
00057
00058 #endif // GLOBALS_H
```

6.12 space_invaders_cpp/main1.cpp File Reference

```
#include "Game.h"
```

Functions

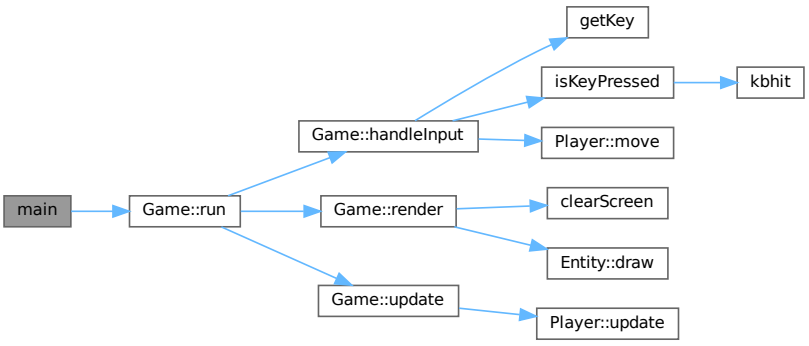
- int **main** ()

6.12.1 Function Documentation

6.12.1.1 main()

```
int main ( )
```

Here is the call graph for this function:



6.13 space_invaders_cpp/README.md File Reference

Index