

SpaceShips

Wygenerowano przez Doxygen 1.8.15



<b>1 Strona główna</b>	<b>1</b>
<b>2 Indeks hierarchiczny</b>	<b>3</b>
2.1 Hierarchia klas . . . . .	3
<b>3 Indeks klas</b>	<b>5</b>
3.1 Lista klas . . . . .	5
<b>4 Indeks plików</b>	<b>7</b>
4.1 Lista plików . . . . .	7
<b>5 Dokumentacja klas</b>	<b>9</b>
5.1 Dokumentacja klasy MainWindow . . . . .	9
5.1.1 Opis szczegółowy . . . . .	10
5.1.2 Dokumentacja funkcji składowych . . . . .	10
5.1.2.1 getGame() . . . . .	10
5.1.2.2 mousePressEvent() . . . . .	10
5.1.2.3 newGame() . . . . .	10
5.1.2.4 shotGame() . . . . .	11
5.1.2.5 showStatus() . . . . .	11
5.2 Dokumentacja klasy MyLabel . . . . .	12
5.2.1 Opis szczegółowy . . . . .	12
5.2.2 Dokumentacja konstruktora i destruktor . . . . .	12
5.2.2.1 MyLabel() . . . . .	12
5.3 Dokumentacja klasy Status . . . . .	13
5.3.1 Opis szczegółowy . . . . .	13
5.3.2 Dokumentacja konstruktora i destruktor . . . . .	13
5.3.2.1 Status() . . . . .	14
5.3.3 Dokumentacja funkcji składowych . . . . .	14
5.3.3.1 getBoard() . . . . .	14
5.3.3.2 getCode() . . . . .	14
5.3.3.3 getShipName() . . . . .	15
5.3.3.4 getSteps() . . . . .	15
5.3.3.5 getType() . . . . .	15
5.3.3.6 setBoard() . . . . .	15
5.3.3.7 setCode() . . . . .	16
5.3.3.8 setShipName() . . . . .	16
5.3.3.9 setSteps() . . . . .	16
5.3.3.10 setType() . . . . .	16
5.3.3.11 toString() . . . . .	17
<b>6 Dokumentacja plików</b>	<b>19</b>
6.1 Dokumentacja pliku mainwindow.h . . . . .	19
6.1.1 Opis szczegółowy . . . . .	19

6.2 Dokumentacja pliku mylabel.h . . . . .	19
6.2.1 Opis szczegółowy . . . . .	20
6.3 Dokumentacja pliku status.h . . . . .	20
6.3.1 Opis szczegółowy . . . . .	20
<b>Indeks</b>	<b>21</b>

# Rozdział 1

## Strona główna

Autorzy: Ernest Bieś, Konrad Czechowski, Dawid Kwaśny

### Podstawowe informacje

"Statki kosmiczne" to gra logiczna będąca połączeniem dwóch popularnych gier "Statki" oraz "Saper".

### Zasady gry

Po rozpoczęciu gry na planszy o wymiarach 9x9 utworzonej z kwadratów są losowo rozmieszczone statki kosmiczne o różnych wielkościach: Dwa *Transportowce* o wymiarze 4 Trzy *Samoloty kosmiczne* o wymiarze 3 Trzy *Wahadłowce* o wymiarze 2 Dwa *Szturmowce* o wymiarze 1

Statki nie są widoczne. Zadaniem gracza jest odnalezienie wszystkich statków w jak najmniejszej liczbie kroków. W przypadku odkrycia pola, które nie zawiera żadnego statku widoczna jest liczba, która wskazuje na ilu polach sąsiadujących z odkrytym polem znajdują się statki. Wobec powyższego gracz musi podejmować swoją decyzję dotyczącą pola, które w następnej kolejności odkryje na podstawie logicznego myślenia, gdyż tym samym zwiększa swoje szanse na odnalezienie statków w pobliżu.

### Opis klienta

#### Interfejs graficzny klienta

W głównym oknie programu po lewej stronie widoczna jest plansza o rozmiarze 9x9 kwadratów na której wyświetlane są ikony zestrzelonych statków oraz pola, które zostały odkryte przez gracza, a nie zawierają żadnych statków. Po uruchomieniu gry po lewej stronie wyświetlone zostają podstawowe informacje dotyczące zasad gry oraz statków jakie musimy zestrzelić. W górnym lewym rogu znajduje się przycisk za pomocą którego możemy ukryć zasady gry. W lewym dolnym rogu okna znajdują się pola służące do wprowadzenia nazwy użytkownika i hasła oraz przyciski za pomocą których możemy utworzyć nową grę na serwerze lub wczytać już istniejącą. Na dole w środku okna znajduje się konsola na której wyświetlane są informacje dotyczące akcji podejmowanych przez gracza. Z prawej strony w dolnej części okna znajduje się licznik z aktualnie wykonanymi krokami przez gracza.

W trakcie gry użytkownik po wskazaniu myszką wybranego pola i naciśnięciu lewego przycisku myszki odsłania poszczególne pola na planszy. W przypadku trafienia statku jego ikona zostaje wyświetlona w odpowiednim miejscu na planszy, natomiast w przypadku "pudła" we wskazanym polu pojawia się znak X oraz liczba wskazująca na ilu polach sąsiadujących ze wskazanym znajdują się statki. Równocześnie po wskazaniu pola odtwarzany jest odpowiedni dźwięk.

## Instrukcja obsługi

Po uruchomieniu klienta użytkownik wprowadza nazwę użytkownika oraz hasło, a następnie naciska przycisk "↩ Nowa gra". Zostaje utworzona nowa gra dla użytkownika, następnie korzystając z myszki naciska lewy przycisk na wybranym polu na planszy. Jeżeli pod odkrytym polem znajduje się statek, jego ikona zostanie wyświetlona w odpowiednim kwadracie oraz program zasignalizuje trafienie odpowiednim dźwiękiem. Jeżeli natomiast wskazane pole nie zawiera żadnego statku to zostanie na nim wyświetlony znak X oraz liczba wskazująca na ilu polach sąsiadujących znajdują się statki. Jednocześnie za każdym wskazaniem pola zostaje zwiększony licznik znajdujący się w prawym, dolnym rogu ekranu. Jednocześnie na konsoli będą pojawiały się komunikaty dotyczące trafienia danego statku, jego nazwy lub też, że oddany strzał to niestety "pudło". Po zestrzeleniu wszystkich statków kosmicznych wyświetlany jest komunikat z zapytaniem czy użytkownik chce zakończyć działanie programu. W przypadku udzielenia negatywnej odpowiedzi gra jest zapisywana na serwerze, natomiast klient zeruje wszelkie dane i ustawienia celem umożliwienia użytkownikowi rozpoczęcia nowej gry.

## Rozdział 2

# Indeks hierarchiczny

### 2.1 Hierarchia klas

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

QLabel	
MyLabel . . . . .	12
QMainWindow	
MainWindow . . . . .	9
Status . . . . .	13





## Rozdział 3

# Indeks klas

### 3.1 Lista klas

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

#### MainWindow

Klasa [MainWindow](#) definiuje metody tworzące i obsługujące sesje gry na serwerze, odpowiada za całościowy wygląd klienta gry . . . . . 9

#### MyLabel

Klasa [MyLabel](#) opisuje obiekt QLabel wyświetlający planszę gry . . . . . 12

#### Status

Klasa [Status](#) zapewnia metody służące do komunikacji klienta z serwerem. Przechowuje ilość oddanych strzałów, nazwę trafionego statku, otrzymaną planszę z serwera i inne . . . . . 13



## Rozdział 4

# Indeks plików

### 4.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

<a href="#">mainwindow.h</a>	
Plik nagłówkowy <a href="#">mainwindow.h</a> zawierający klasę <a href="#">MainWindow</a> . . . . .	19
<a href="#">mylabel.h</a>	
Plik nagłówkowy <a href="#">mylabel.h</a> zawierający klasę <a href="#">MyLabel</a> . . . . .	19
<a href="#">status.h</a>	
Plik nagłówkowy <a href="#">status.h</a> zawierający klasę <a href="#">Status</a> . . . . .	20



## Rozdział 5

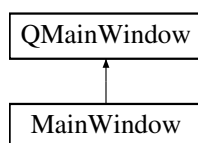
# Dokumentacja klas

### 5.1 Dokumentacja klasy MainWindow

Klasa [MainWindow](#) definiuje metody tworzące i obsługujące sesje gry na serwerze, odpowiada za całokształt wyglądu klienta gry.

```
#include <mainwindow.h>
```

Diagram dziedziczenia dla MainWindow



#### Metody publiczne

- **MainWindow** (QWidget \*parent=nullptr)
- [Status newGame](#) (QString user, QString server)  
*Metoda wywoływana podczas tworzenia nowej gry. Wysyła żądanie do serwera w celu utworzenia nowej sesji gry. Odtwarza dźwięk.*
- [Status shotGame](#) (QString user, QString shot, QString server)  
*Metoda wywoływana podczas kliknięcia na jedno z pól na planszy. Wysyła żądanie do serwera w celu sprawdzenia i zapisania rezultatu oddanego strzału.*
- [Status getGame](#) (QString user, QString server)  
*Metoda wywoływana podczas wczytywania gry. Wysyła żądanie do serwera w celu otrzymania danych o istniejącej sesji gry.*
- void [showStatus](#) ([Status](#) \*status)  
*Wypisuje informacje do konsoli. Odtwarza odpowiednie dźwięki. Wyświetla zapytania do użytkownika.*

#### Metody chronione

- void [mousePressEvent](#) (QMouseEvent \*event)  
*Rysuje znak X lub ikonę statku na klikniętym polu.*
- void [paintEvent](#) (QPaintEvent \*)  
*Przeciążenie rysowania zapewnia skalowanie tła.*
- void [resizeEvent](#) (QResizeEvent \*event)  
*Ustala położenie wszystkich elementów w skalowalnym oknie.*

### 5.1.1 Opis szczegółowy

Klasa [MainWindow](#) definiuje metody tworzące i obsługujące sesje gry na serwerze, odpowiada za całościowy wygląd klienta gry.

### 5.1.2 Dokumentacja funkcji składowych

#### 5.1.2.1 `getGame()`

```
Status MainWindow::getGame (
    QString user,
    QString server )
```

Metoda wywoływana podczas wczytywania gry. Wysyła żądanie do serwera w celu otrzymania danych o istniejącej sesji gry.

##### Parametry

<i>user</i>	nazwa użytkownika
<i>server</i>	adres serwera gry

##### Zwraca

#### 5.1.2.2 `mousePressEvent()`

```
void MainWindow::mousePressEvent (
    QMouseEvent * event ) [protected]
```

Rysuje znak X lub ikonę statku na klikniętym polu.

##### Parametry

<i>event</i>	
--------------	--

#### 5.1.2.3 `newGame()`

```
Status MainWindow::newGame (
    QString user,
    QString server )
```

Metoda wywoływana podczas tworzenia nowej gry. Wysyła żądanie do serwera w celu utworzenia nowej sesji gry. Odtwarza dźwięk.

**Parametry**

<i>user</i>	nazwa użytkownika
<i>server</i>	adres serwera gry

**Zwraca**

Status

**5.1.2.4 shotGame()**

```
Status MainWindow::shotGame (
    QString user,
    QString shot,
    QString server )
```

Metoda wywoływana podczas kliknięcia na jedno z pól na planszy. Wysyła żądanie do serwera w celu sprawdzenia i zapisania rezultatu oddanego strzału.

**Parametry**

<i>user</i>	nazwa użytkownika
<i>shot</i>	kod współrzędnych pola
<i>server</i>	adres serwera gry

**Zwraca**

Obiekt Status

**5.1.2.5 showStatus()**

```
void MainWindow::showStatus (
    Status * status )
```

Wypisuje informacje do konsoli. Odtwarza odpowiednie dźwięki. Wyświetla zapytania do użytkownika.

**Parametry**

<i>status</i>	
---------------	--

Dokumentacja dla tej klasy została wygenerowana z plików:

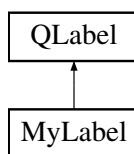
- [mainwindow.h](#)
- [mainwindow.cpp](#)

## 5.2 Dokumentacja klasy MyLabel

Klasa [MyLabel](#) opisuje obiekt QLabel wyświetlający planszę gry.

```
#include <mylabel.h>
```

Diagram dziedziczenia dla MyLabel



### Metody publiczne

- [MyLabel](#) (QWidget \*parent)  
*Konstruktor dziedziczący po QLabel, zapewniający takie metody jak paintEvent.*

### Metody chronione

- void [paintEvent](#) (QPaintEvent \*)  
*rysuje pole gry, siatkę, tło*

#### 5.2.1 Opis szczegółowy

Klasa [MyLabel](#) opisuje obiekt QLabel wyświetlający planszę gry.

#### 5.2.2 Dokumentacja konstruktora i destruktora

##### 5.2.2.1 MyLabel()

```
MyLabel::MyLabel (
    QWidget * parent ) [explicit]
```

Konstruktor dziedziczący po QLabel, zapewniający takie metody jak paintEvent.

##### Parametry

<i>parent</i>	
---------------	--



Dokumentacja dla tej klasy została wygenerowana z plików:

- [mylabel.h](#)
- [mylabel.cpp](#)

## 5.3 Dokumentacja klasy Status

Klasa [Status](#) zapewnia metody służące do komunikacji klienta z serwerem. Przechowuje ilość oddanych strzałów, nazwę trafionego statku, otrzymaną planszę z serwera i inne.

```
#include <status.h>
```

### Metody publiczne

- [Status](#) (QString code, QString shipName, int type, int steps, QString board)  
*Konstruktor tworzący nowy obiekt [Status](#) zgodnie z podanymi parametrami.*
- int [getSteps](#) ()  
*Zwraca liczbę kroków zapisanych w obiekcie [Status](#).*
- QString [getCode](#) ()  
*getCode*
- QString [getShipName](#) ()  
*Zwraca nazwę trafionego statku.*
- int [getType](#) ()  
*Zwraca typ trafionego statku.*
- QString [getBoard](#) ()  
*Zwraca planszę gry w formie tekstowej.*
- QString [toString](#) ()  
*Parsuje obiekt [Status](#) do ciągu tekstowego w formacie JSON do wysłania do serwera.*
- void [setSteps](#) (int steps)  
*Ustawia ilość kroków.*
- void [setCode](#) (QString code)  
*Ustawia kod trafionego pola.*
- void [setShipName](#) (QString shipName)  
*Ustawia nazwę trafionego statku.*
- void [setType](#) (int type)  
*Ustawia typ trafionego statku.*
- void [setBoard](#) (QString board)  
*Ustawia stan planszy gry w formie zapisu tekstowego.*

### 5.3.1 Opis szczegółowy

Klasa [Status](#) zapewnia metody służące do komunikacji klienta z serwerem. Przechowuje ilość oddanych strzałów, nazwę trafionego statku, otrzymaną planszę z serwera i inne.

### 5.3.2 Dokumentacja konstruktora i destruktor

### 5.3.2.1 Status()

```
Status::Status (
    QString code,
    QString shipName,
    int type,
    int steps,
    QString board )
```

Konstruktor tworzący nowy obiekt [Status](#) zgodnie z podanymi parametrami.

#### Parametry

<i>kod</i>	
<i>nazwa</i>	statku
<i>typ</i>	
<i>ilość</i>	kroków
<i>plansza</i>	

## 5.3.3 Dokumentacja funkcji składowych

### 5.3.3.1 getBoard()

```
QString Status::getBoard ( )
```

Zwraca planszę gry w formie tekstowej.

#### Zwraca

plansza

### 5.3.3.2 getCode()

```
QString Status::getCode ( )
```

getCode

#### Zwraca

kod odpowiedzi np. SHOTDOWN, ENDGAME

#### 5.3.3.3 getShipName()

```
QString Status::getShipName ( )
```

Zwraca nazwę trafionego statku.

**Zwraca**

nazwa trafionego statku

#### 5.3.3.4 getSteps()

```
int Status::getSteps ( )
```

Zwraca liczbę kroków zapisanych w obiekcie [Status](#).

**Zwraca**

liczba kroków

#### 5.3.3.5 getType()

```
int Status::getType ( )
```

Zwraca typ trafionego statku.

**Zwraca**

typ trafionego statku

#### 5.3.3.6 setBoard()

```
void Status::setBoard (
    QString board )
```

Ustawia stan planszy gry w formie zapisu tekstowego.

**Parametry**

<i>plansza</i>	
----------------	--

#### 5.3.3.7 `setCode()`

```
void Status::setCode (
    QString code )
```

Ustawia kod trafionego pola.

##### Parametry

<i>kod,np</i>	"ENDGAME"
---------------	-----------

#### 5.3.3.8 `setShipName()`

```
void Status::setShipName (
    QString shipName )
```

Ustawia nazwę trafionego statku.

##### Parametry

<i>nazwa</i>	statku
--------------	--------

#### 5.3.3.9 `setSteps()`

```
void Status::setSteps (
    int steps )
```

Ustawia ilość kroków.

##### Parametry

<i>ilość</i>	kroków
--------------	--------

#### 5.3.3.10 `setType()`

```
void Status::setType (
    int type )
```

Ustawia typ trafionego statku.

## Parametry

<i>typ</i>	trafionego statku
------------	-------------------

## 5.3.3.11 toString()

```
QString Status::toString ( )
```

Parsuje obiekt [Status](#) do ciągu tekstowego w formacie JSON do wysłania do serwera.

## Zwraca

obiekt [Status](#) w formie JSON

Dokumentacja dla tej klasy została wygenerowana z plików:

- [status.h](#)
- [status.cpp](#)



## Rozdział 6

# Dokumentacja plików

### 6.1 Dokumentacja pliku mainwindow.h

Plik nagłówkowy `mainwindow.h` zawierający klasę `MainWindow`.

```
#include <QMainWindow>
#include <QtNetwork/QtNetworkReply>
#include <QMouseEvent>
#include "status.h"
#include "mylabel.h"
```

#### Komponenty

- class `MainWindow`

*Klasa `MainWindow` definiuje metody tworzące i obsługujące sesje gry na serwerze, odpowiada za całościowy wygląd klienta gry.*

#### 6.1.1 Opis szczegółowy

Plik nagłówkowy `mainwindow.h` zawierający klasę `MainWindow`.

### 6.2 Dokumentacja pliku mylabel.h

Plik nagłówkowy `mylabel.h` zawierający klasę `MyLabel`.

```
#include <QWidget>
#include <QLabel>
#include <QPainter>
```

## Komponenty

- class [MyLabel](#)

*Klasa [MyLabel](#) opisuje obiekt `QLabel` wyświetlający planszę gry.*

## Zmienne

- QString [boardGame](#)

*Zmienna `boardGame` zawierająca aktualną planszę w formie tekstowej.*

### 6.2.1 Opis szczegółowy

Plik nagłówkowy [mylabel.h](#) zawierający klasę [MyLabel](#).

## 6.3 Dokumentacja pliku status.h

Plik nagłówkowy [status.h](#) zawierający klasę [Status](#).

```
#include <QString>
```

## Komponenty

- class [Status](#)

*Klasa [Status](#) zapewnia metody służące do komunikacji klienta z serwerem. Przechowuje ilość oddanych strzałów, nazwę trafionego statku, otrzymaną planszę z serwera i inne.*

### 6.3.1 Opis szczegółowy

Plik nagłówkowy [status.h](#) zawierający klasę [Status](#).



# Indeks

- getBoard
  - Status, [14](#)
- getCode
  - Status, [14](#)
- getGame
  - MainWindow, [10](#)
- getShipName
  - Status, [14](#)
- getSteps
  - Status, [15](#)
- getType
  - Status, [15](#)
- MainWindow, [9](#)
  - getGame, [10](#)
  - mousePressEvent, [10](#)
  - newGame, [10](#)
  - shotGame, [11](#)
  - showStatus, [11](#)
- mainwindow.h, [19](#)
- mousePressEvent
  - MainWindow, [10](#)
- MyLabel, [12](#)
  - MyLabel, [12](#)
- mylabel.h, [19](#)
- newGame
  - MainWindow, [10](#)
- setBoard
  - Status, [15](#)
- setCode
  - Status, [16](#)
- setShipName
  - Status, [16](#)
- setSteps
  - Status, [16](#)
- setType
  - Status, [16](#)
- shotGame
  - MainWindow, [11](#)
- showStatus
  - MainWindow, [11](#)
- Status, [13](#)
  - getBoard, [14](#)
  - getCode, [14](#)
  - getShipName, [14](#)
  - getSteps, [15](#)
  - getType, [15](#)
  - setBoard, [15](#)
  - setCode, [16](#)
  - setShipName, [16](#)
  - setSteps, [16](#)
  - setType, [16](#)
  - Status, [13](#)
  - toString, [17](#)
- status.h, [20](#)
- toString
  - Status, [17](#)