

Podręcznik

Wygenerowano przez Doxygen 1.7.3

Tue Jan 24 2012 23:36:04



# Spis treści

<b>1</b>	<b>RTTT - Risky Tic Tac Toe</b>	<b>1</b>
1.1	Opis gry	1
1.2	Reguły panujące w kosmosie	1
1.2.1	Zdobywanie planet	1
1.2.2	Zdobywanie jednostek	1
1.2.3	Wygrana	2
<b>2</b>	<b>Algorytmy</b>	<b>3</b>
2.1	Algorytm walki	3
<b>3</b>	<b>Struktura katalogów</b>	<b>5</b>
3.1	Katalogi	5
<b>4</b>	<b>Indeks przestrzeni nazw</b>	<b>7</b>
4.1	Lista przestrzeni nazw	7
<b>5</b>	<b>Indeks klas</b>	<b>9</b>
5.1	Hierarchia klas	9
<b>6</b>	<b>Indeks klas</b>	<b>11</b>
6.1	Lista klas	11
<b>7</b>	<b>Indeks plików</b>	<b>13</b>
7.1	Lista plików	13
<b>8</b>	<b>Dokumentacja katalogów</b>	<b>15</b>
8.1	Dokumentacja katalogu include/	15
8.2	Dokumentacja katalogu src/	15
<b>9</b>	<b>Dokumentacja przestrzeni nazw</b>	<b>17</b>
9.1	Dokumentacja przestrzeni nazw Drawing	17
9.1.1	Opis szczegółowy	18
9.1.2	Dokumentacja funkcji	18
9.1.2.1	drawLine	18
9.1.2.2	drawQuad	19
9.1.2.3	putPix	19
9.1.2.4	setObj	19
9.2	Dokumentacja przestrzeni nazw RETURNS	19
9.2.1	Opis szczegółowy	19
9.2.2	Dokumentacja typów wyliczanych	19

9.2.2.1	MOVE	19
9.3	Dokumentacja przestrzeni nazw Screen	20
9.3.1	Opis szczegółowy	21
9.4	Dokumentacja przestrzeni nazw WindowEngine	21
9.4.1	Opis szczegółowy	23
9.4.2	Dokumentacja funkcji	23
9.4.2.1	addKeyDownEventHandler	23
9.4.2.2	addKeyPressedEventHandler	24
9.4.2.3	addKeyUpEventHandler	24
9.4.2.4	addMouseDownEventHandler	24
9.4.2.5	addMouseMotionEventHandler	24
9.4.2.6	addMouseUpEventHandler	24
9.4.2.7	init	24
<b>10</b>	<b>Dokumentacja klas</b>	<b>27</b>
10.1	Dokumentacja klasy Sprite::Anim	27
10.1.1	Opis szczegółowy	28
10.2	Dokumentacja klasy Sprite::Anim::AnimFrame	28
10.2.1	Opis szczegółowy	29
10.3	Dokumentacja klasy Client	29
10.3.1	Opis szczegółowy	29
10.3.2	Dokumentacja konstruktora i destruktor	29
10.3.2.1	Client	29
10.3.2.2	Client	30
10.3.3	Dokumentacja funkcji składowych	30
10.3.3.1	write	30
10.4	Dokumentacja struktury Cube	30
10.5	Dokumentacja klasy GameEngine	31
10.5.1	Opis szczegółowy	31
10.5.2	Dokumentacja konstruktora i destruktor	32
10.5.2.1	GameEngine	32
10.5.3	Dokumentacja funkcji składowych	32
10.5.3.1	ActPlayer	32
10.5.3.2	EndTurn	32
10.5.3.3	Move	32
10.5.3.4	RemovePlayer	33
10.6	Dokumentacja klasy GameEngineClient	33
10.6.1	Opis szczegółowy	33
10.7	Dokumentacja klasy Message	33
10.7.1	Opis szczegółowy	35
10.7.2	Dokumentacja konstruktora i destruktor	35
10.7.2.1	Message	35
10.7.3	Dokumentacja funkcji składowych	35
10.7.3.1	length	35
10.8	Dokumentacja klasy Participant	35
10.9	Dokumentacja klasy Planet	36
10.9.1	Opis szczegółowy	36
10.9.2	Dokumentacja konstruktora i destruktor	37
10.9.2.1	Planet	37
10.9.3	Dokumentacja funkcji składowych	37

10.9.3.1	Atak	37
10.9.3.2	EndTurn	37
10.9.3.3	RetGracz	37
10.9.3.4	RetJednostki	38
10.9.3.5	RetOkupant	38
10.9.3.6	RetPoziom	38
10.9.3.7	SetPlayer	38
10.10	Dokumentacja klasy Point	39
10.10.1	Opis szczegółowy	39
10.10.2	Dokumentacja atrybutów składowych	39
10.10.2.1	itsX	39
10.10.2.2	itsY	39
10.10.2.3	itsZ	39
10.11	Dokumentacja klasy Room	39
10.11.1	Dokumentacja funkcji składowych	40
10.11.1.1	deliver	40
10.12	Dokumentacja klasy Server	40
10.13	Dokumentacja klasy Session	40
10.14	Dokumentacja klasy Sprite	41
10.14.1	Opis szczegółowy	43
10.15	Dokumentacja klasy Sprite::SpritePtr	43
10.15.1	Opis szczegółowy	43
10.16	Dokumentacja klasy SpriteSDL2D	44
10.16.1	Opis szczegółowy	44
10.16.2	Dokumentacja funkcji składowych	44
10.16.2.1	flush	44
10.16.2.2	print	44
10.17	Dokumentacja struktury Vertex	45
10.17.1	Opis szczegółowy	46
<b>11</b>	<b>Dokumentacja plików</b>	<b>47</b>
11.1	Dokumentacja pliku include/consts.h	47
11.1.1	Opis szczegółowy	48
11.1.2	Dokumentacja definicji typów	48
11.1.2.1	FightResult	48
11.1.2.2	FightResultRow	48
11.1.2.3	uint	48
11.1.2.4	uint16	48
11.1.3	Dokumentacja zmiennych	48
11.1.3.1	OCCUPY_MAX	48



# Rozdział 1

## RTTT - Risky Tic Tac Toe

### 1.1 Opis gry

Gra strategiczna łącząca elementy gry Ryzyko z grą "Kółko i krzyżyk". Fabuła gry osadzona jest w przestrzeni kosmicznej. Twoim zadaniem, jak generała floty, jest odeprzeć inwazję kosmitów, oraz wyeliminować konkurencyjne frakcje

### 1.2 Reguły panujące w kosmosie

#### 1.2.1 Zdobywanie planet

Podstawowym elementem gry są posiadane planety. Aby podbić planetę, należy umieścić na niej swoje jednostki. Wysłane jednostki po dotarciu do celu, walczą z stacjonującymi tam statkami wroga. Po wygranej bitwie, planeta przechodzi w stan okupacji. Jeśli jest to planeta neutralna, należy ją okupować (posiadać tam co najmniej jedną jednostkę) przez 3 tury.

Jeśli natomiast jest to planeta przeciwnika trzeba odczekać 3 tury na obalenie tamtejszego rządu i kolejne 3 tury na utworzenie swojego.

Natomiast, jeśli podczas okupacji wróg najedzie na planetę która była okupowana przez 2 dni, pokona jednostki gracza i sam zacznie ją okupować, musi odczekać tylko 2 tury na obalenie tworzonoego tam rządu. Dokładnie tyle ile gracz poświęcił na jego utworzenie.

#### 1.2.2 Zdobywanie jednostek

Na każdej pobitej przez gracza planecie produkowane są statki kosmiczne. Tempo tworzenia statków wynisi jeden na turę i zawsze jest tworzony na koniec tury danego gracza. Tak więc po wykonaniu swoich manewrów, na każdej planecie tworzona jest jedna nowa jednostka. Na planetach okupowanych przez przeciwnika nie sa Tworzone jednostki.

### 1.2.3 Wygrana

Aby wygrać rozgrywkę, należy odeprzeć atak kosmitów. Można to zrobić poprzez eliminację wszystkich wrogich jednostek bądź wykorzystanie *Broni ostatecznej*. Aby móc z niej skorzystać, należy zdobyć planety znajdujące się w jednej linii na przestrzeni całego obszaru bitwy. Zostaje wtedy aktywowana *Bron ostateczna* i wszystkie wrogie jednostki zostają zniszczone.



## **Rozdział 2**

# **Algorytmy**

### **2.1 Algorytm walki**

W walce uczestniczy dwóch różnych graczy - atakujący i broniący się. Na każdą rundę walki wystawiana jest maksymalnie flota składająca się z 3 jednostek.



## Rozdział 3

# Struktura katalogów

### 3.1 Katalogi

Ta struktura katalogów jest posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

include . . . . .	15
src . . . . .	15



## Rozdział 4

# Indeks przestrzeni nazw

### 4.1 Lista przestrzeni nazw

Tutaj znajdują się wszystkie udokumentowane przestrzenie nazw wraz z ich krótkimi opisami:

<a href="#">Drawing</a> (Funkcje obsługujące rysowanie ) . . . . .	17
<a href="#">RETURNS</a> . . . . .	19
<a href="#">Screen</a> (Chyba cała logika okienka jest tutaj zawarta ) . . . . .	20
<a href="#">WindowEngine</a> (Tworzenie okienka, obsługa zdarzeń ) . . . . .	21



## Rozdział 5

# Indeks klas

### 5.1 Hierarchia klas

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

Sprite::Anim . . . . .	27
Sprite::Anim::AnimFrame . . . . .	28
Client . . . . .	29
Cube . . . . .	30
GameEngine . . . . .	31
GameEngineClient . . . . .	33
Message . . . . .	33
Participant . . . . .	35
Session . . . . .	40
Planet . . . . .	36
Point . . . . .	39
Room . . . . .	39
Server . . . . .	40
Sprite . . . . .	41
SpriteSDL2D . . . . .	44
Sprite::SpritePtr . . . . .	43
Vertex . . . . .	45





## Rozdział 6

# Indeks klas

### 6.1 Lista klas

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

<a href="#">Sprite::Anim</a> (Informacje o animacji ) . . . . .	27
<a href="#">Sprite::Anim::AnimFrame</a> (Klatka animacji ) . . . . .	28
<a href="#">Client</a> (Połączenie z serwerem ) . . . . .	29
<a href="#">Cube</a> . . . . .	30
<a href="#">GameEngine</a> (Główny silnik gry ) . . . . .	31
<a href="#">GameEngineClient</a> (Klasa silnika gry dla klienta ) . . . . .	33
<a href="#">Message</a> (Przesyłana wiadomość ) . . . . .	33
<a href="#">Participant</a> . . . . .	35
<a href="#">Planet</a> (Klasa planety ) . . . . .	36
<a href="#">Point</a> (Klasa położenia w przestrzeni ) . . . . .	39
<a href="#">Room</a> . . . . .	39
<a href="#">Server</a> . . . . .	40
<a href="#">Session</a> . . . . .	40
<a href="#">Sprite</a> (Klasa zajmująca się wczytaniem, wyświetlaniem i ogólnie obsługą obrazków ) . . . . .	41
<a href="#">Sprite::SpritePtr</a> (Smart Pointer na sprite. Zwalnia sprite jeśli nikt go nie uży- wa ) . . . . .	43
<a href="#">SpriteSDL2D</a> (Klasa sprite oparta na SDLu ) . . . . .	44
<a href="#">Vertex</a> (Prosty vertex/wektor 3D, zawiera podstawowe operacje ) . . . . .	45



## Rozdział 7

# Indeks plików

### 7.1 Lista plików

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

include/Client.hpp . . . . .	??
include/consts.h . . . . .	47
include/drawing.h . . . . .	??
include/gameengine.h . . . . .	??
include/gameengineclient.h . . . . .	??
include/main.creammy.h . . . . .	??
include/main.h . . . . .	??
include/Message.hpp . . . . .	??
include/Participant.hpp . . . . .	??
include/planet.h . . . . .	??
include/point.h . . . . .	??
include/Room.hpp . . . . .	??
include/screen.h . . . . .	??
include/Server.hpp . . . . .	??
include/Session.hpp . . . . .	??
include/sprite.h . . . . .	??
include/sprite_sdl_2d.h . . . . .	??
include/vertex.h . . . . .	??
include/windowengine.h . . . . .	??



## Rozdział 8

# Dokumentacja katalogów

### 8.1 Dokumentacja katalogu include/

#### Pliki

- plik **Client.hpp**
- plik [consts.h](#)
- plik **drawing.h**
- plik **gameengine.h**
- plik **gameengineclient.h**
- plik **main.creammy.h**
- plik **main.h**
- plik **Message.hpp**
- plik **Participant.hpp**
- plik **planet.h**
- plik **point.h**
- plik **Room.hpp**
- plik **screen.h**
- plik **Server.hpp**
- plik **Session.hpp**
- plik **sprite.h**
- plik **sprite\_sdl\_2d.h**
- plik **vertex.h**
- plik **windowengine.h**

### 8.2 Dokumentacja katalogu src/

#### Pliki

- plik **drawing.cpp**
- plik **gameengine.cpp**

- plik **gameengineclient.cpp**
- plik **main.creammy.cpp**
- plik **main.czaju.cpp**
- plik **main.torgiren.cpp**
- plik **planet.cpp**
- plik **Room.cpp**
- plik **screen.cpp**
- plik **Server.cpp**
- plik **Session.cpp**
- plik **sprite.cpp**
- plik **sprite\_sdl\_2d.cpp**
- plik **windowengine.cpp**

## Rozdział 9

# Dokumentacja przestrzeni nazw

### 9.1 Dokumentacja przestrzeni nazw Drawing

Funkcje obsługujące rysowanie.

#### Funkcje

- void `clearZBuff` ()  
*Czyszczenie zbuffera oraz bufora obiektów.*
- void `setSurface` (SDL\_Surface \*srf)  
*Ustawia aktualną powierzchnię do rysowania. Nigdzie nie jest sprawdzane, czy nie jest NULLe.*
- SDL\_Surface \* `getSurface` ()  
*Zwraca aktualną powierzchnię do rysowania.*
- void `setColor` (unsigned int sc)  
*Ustawia aktualny kolor, 0xAARRGGBB.*
- unsigned int `getColor` ()  
*Zwraca aktualny kolor.*
- void `setObj` (void \*obj)  
*Ustawia aktualny obiekt wpisywany do bufora obiektów.*
- void \* `getObj` (int x, int y)  
*Zwraca wskaźnik na obiekt znajdujący się na ekranie na pozycji x, y.*
- template<class T >  
void `swap` (T a, T b)

- void `putPix` (int x, int y, float z, float alpha)  
*Wstawia na pozycji x, y, z piksel o przezroczystosci równej alpha (od 0.0f do 1.0f).*
- void `drawLine` (const `Vertex` &a, const `Vertex` &b)  
*Rysuje linię łączącą punkty a i b.*
- bool `SameSide` (const `Vertex` &p1, const `Vertex` &p2, const `Vertex` &a, const `Vertex` &b)
- bool `PointInTriangle` (const `Vertex` &p, const `Vertex` &a, const `Vertex` &b, const `Vertex` &c)
- void `drawTriangle` (const `Vertex` &a, const `Vertex` &b, const `Vertex` &c)  
*Rysuje trójkąt łączący punkty a, b i c.*
- void `drawQuad` (const `Vertex` &a, const `Vertex` &b, const `Vertex` &c, const `Vertex` &d)  
*Rysuje czworokąt łączący punkty a, b, c i d.*

## Zmienne

- `SDL_Surface * srf` = NULL
- void \* `obj` = NULL
- float \* `zbuff` = NULL
- void \*\* `obuff` = NULL
- unsigned int `color` = 0xFFFFFFFF
- const `Vertex light` (0, 0.7071, -0.701)

### 9.1.1 Opis szczegółowy

Funkcje obsługujące rysowanie.

### 9.1.2 Dokumentacja funkcji

#### 9.1.2.1 void `Drawing::drawLine` ( const `Vertex` & a, const `Vertex` & b )

Rysuje linię łączącą punkty a i b.

Algorytm wygląda następująco:

1. Z twierdzenia Pitagorasa oblicz długość odcinka(*l*)
2. Oblicz odległość w poziomie (*dx*) i w pionie (*dy*) a następnie podziel je przez długość odcinka
3. Zapaczynając od jednego z punktów, odpal pętlę *l* razy
4. Dla każdej iteracji wypisz piksel w aktualnym punkcie i przesun się o *dx*, *dy*



**9.1.2.2 void Drawing::drawQuad ( const Vertex & a, const Vertex & b, const Vertex & c, const Vertex & d )**

Rysuje czworokąt łączący punkty *a*, *b*, *c* i *d*.

W rzeczywistości są to trójkąty *a*, *b*, *c* oraz *c*, *d*, *a*. Proponuję o tym pamiętać.

**9.1.2.3 void Drawing::putPix ( int x, int y, float z, float *alpha* ) [inline]**

Wstawia na pozycji *x*, *y*, *z* piksel o przezroczystości równej *alpha* (od 0.0f do 1.0f).

Sprawdzone jest położenie piksela, czy nie wystaje poza ekran. Współrzędna *z* używana jest tylko do zbuffera.

**9.1.2.4 void Drawing::setObj ( void \* *obj* )**

Ustawia aktualny obiekt wpisywany do bufora obiektów.

Bufor obiektów jest równy co do wielkości zbufferowi oraz powierzchni. Podczas wstawiania piksela, w tym samym miejscu zapisywana jest informacja o obiekcie tam znajdującym się.

#### Parametry

<i>in</i>	<i>obj</i>	Wskaźnik na obiekt. Musisz pamiętać, co podsyłasz, ponieważ bufor obiektów korzysta z wbudowanego w C++ dynamicznego rzutowania typów (void*)
-----------	------------	---

## 9.2 Dokumentacja przestrzeni nazw RETURNS

### Wyliczenia

- enum **MOVE** {  
TOO\_MUCH, OUT\_OF\_AREA, NOT\_ANY, MOVE\_OK,  
MOVE\_FIGHT }

#### 9.2.1 Opis szczegółowy

Zawiera komunikaty zwracane z funkcji

#### 9.2.2 Dokumentacja typów wyliczanych

##### 9.2.2.1 enum RETURNS::MOVE

Błędy zwracane przy operacjach przenoszenia jednostek

- **TOO\_MUCH** - jeśli wybrana ilość jednostek jest większa niż możliwa
- **OUT\_OF\_AREA** - jeśli wybrane źródło i/lub cel jest poza obszarem gry (normalnie nie występuje)
- **NOT\_ANY** - jeśli gracz nie posiada żadnych jednostek na danej planecie źródłowej
- **MOVE\_OK** - jeśli przenoszenie jednostek się powiodło
- **MOVE\_FIGHT** - jeśli odbyła się walka

### 9.3 Dokumentacja przestrzeni nazw Screen

Chyba cała logika okienka jest tutaj zawarta.

#### Funkcje

- void **drawCube** (**Cube** c)
- void **mdown** (int x, int y, int key)
- void **mup** (int x, int y, int key)
- void **mmove** (int x, int y, int key)
- void **mroll** (bool down)
- void **kpressed** (int k)
- void **init** ()  
*Inicjalizacja, ustawia handlersy klikniec i wielkosc poziomu na pewna z gory ustalona wartosc~.*
- void **update** ()  
*Ibumtralala.*
- void **draw** ()  
*Rysuje pole gry.*
- void **setSize** (int size)  
*Ustawia pole gry na zadana wielkosc.*
- void **rotateArb** (**Vertex** &v, const **Vertex** &s, const **Vertex** &a, float ang)

#### Zmienne

- bool **lmb** = false
- bool **rmb** = false
- bool **mmb** = false
- int **lx** = -1
- int **ly** = -1

- int **mx** = 0
- int **my** = 0
- float **rx** = 0.0f
- float **ry** = 0.0f
- float **rz** = 0.0f
- float **scale** = 0.0f
- const float **FRICTION** = 0.5f
- float **spdx** = 0.0f
- float **spdy** = 0.0f
- int **size** = 4
- vector< vector< vector< **Cube** > > > **area**

### 9.3.1 Opis szczegółowy

Chyba cała logika okienka jest tutaj zawarta. Obsługa rysowania pola gry, obrotów, kliknięcia na klocki~

## 9.4 Dokumentacja przestrzeni nazw WindowEngine

Tworzenie okienka, obsługa zdarzeń

### Wyliczenia

- enum **RenderType** { **SDL**, **OPENGL** }
- enum **WaitType** { **DELAY**, **DELTA** }

### Funkcje

- bool **initSDL** ()
- void **setFlags** (unsigned int flags)  
*Ustawia flagi okna (SDL). Nie tykac jeśli nie wiesz, co robisz.*
- void **setWaitType** (WaitType wt)  
*Ustawia sposób reagowania na koniec danej klatki.*
- RenderType **getRenderType** ()
- WaitType **getWaitType** ()
- float **getDelta** ()
- SDL\_Surface \* **getScreen** ()  
*Zwraca wskaźnik na ekran (SDL)*
- bool **init** (RenderType rt=SDL, WaitType wt=DELAY)  
*Inicjalizacja ekranu.*

- bool `quit` ()  
*Zamknięcie wszystkiego, co się da.*
- bool `update` ()  
*Obsługa zdarzeń*
- bool `print` ()  
*Wyświetlenie na ekran aktualnego stanu bufora.*
- bool `addKeyDownEventHandler` (void(\*handle)(int))  
*Rejestracja funkcji wywoływanej po wciśnięciu klawisza na klawiaturze.*
- bool `addKeyUpEventHandler` (void(\*handle)(int))  
*Rejestracja funkcji wywoływanej po puszczeniu klawisza na klawiaturze.*
- bool `addKeyPressedEventHandler` (void(\*handle)(int))  
*Rejestracja funkcji wywoływanej po przytrzymaniu klawisza na klawiaturze.*
- bool `addMouseDownEventHandler` (void(\*handle)(int, int, int))  
*Rejestracja funkcji wywoływanej po wciśnięciu przycisku myszy.*
- bool `addMouseUpEventHandler` (void(\*handle)(int, int, int))  
*Rejestracja funkcji wywoływanej po puszczeniu przycisku myszy.*
- bool `addMouseMoveEventHandler` (void(\*handle)(int, int, int))  
*Rejestracja funkcji wywoływanej po ruszeniu myszy.*
- void `delKeyDownEventHandler` (void(\*handle)(int))  
*Kasuje wskaźnik na funkcję handle.*
- void `delKeyUpEventHandler` (void(\*handle)(int))  
*Kasuje wskaźnik na funkcję handle.*
- void `delKeyPressedEventHandler` (void(\*handle)(int))  
*Kasuje wskaźnik na funkcję handle.*
- void `delMouseDownEventHandler` (void(\*handle)(int, int, int))  
*Kasuje wskaźnik na funkcję handle.*
- void `delMouseUpEventHandler` (void(\*handle)(int, int, int))  
*Kasuje wskaźnik na funkcję handle.*
- void `delMouseMoveEventHandler` (void(\*handle)(int, int, int))  
*Kasuje wskaźnik na funkcję handle.*
- void `clearEventHandlers` ()

*Kasuje wszystkie wskaźniki na funkcje.*

- bool `getKeyState` (int key)

*Zwraca true jeśli klawisz key jest wciśnięty.*

- bool `getMouseState` (int key)

*Zwraca true jeśli przycisk myszy key jest wciśnięty.*

## Zmienne

- bool **run** = true
- unsigned int **flags** = 0x0
- unsigned int **frameTime** = 0
- float **delta** = 0.0f
- set< void(\*) (int)> **keyDownHandles**
- set< void(\*) (int)> **keyUpHandles**
- set< void(\*) (int)> **keyPressedHandles**
- set< void(\*) (int, int, int)> **mouseDownHandles**
- set< void(\*) (int, int, int)> **mouseUpHandles**
- set< void(\*) (int, int, int)> **mouseMotionHandles**
- RenderType **rt**
- WaitType **wt**
- SDL\_Event **event**
- SDL\_Surface \* **screen** = NULL
- Uint8 \* **keys** = SDL\_GetKeyState(NULL)

### 9.4.1 Opis szczegółowy

Tworzenie okienka, obsługa zdarzeń Obsługuje dowolną ilość bibliotek, po uprzednim dopisaniu ich obsługi. Posiada dwa tryby działania: DELAY - stała przerwa między klatkami oraz DELTA - działa z maksymalną prędkością. DELTA zalecana jest dla OpenGLa, którego tutaj nie ma. Co by nie przeciążać procesora, zalecane jest używanie DELAY.

### 9.4.2 Dokumentacja funkcji

#### 9.4.2.1 bool WindowEngine::addKeyDownEventHandler ( void(\*) (int) handle )

Rejestracja funkcji wywoływanej po wciśnięciu klawisza na klawiaturze.

#### Parametry

<i>handle</i>	Wskaźnik na funkcję. Argument to kod klawisza
---------------	---

**9.4.2.2 bool WindowEngine::addKeyPressedEventHandler ( void(\*)(int) *handle* )**

Rejestracja funkcji wywoływanej po przytrzymaniu klawisza na klawiaturze.

**Parametry**

<i>handle</i>	Wskaźnik na funkcję. Argument to kod klawisza
---------------	---

**9.4.2.3 bool WindowEngine::addKeyUpEventHandler ( void(\*)(int) *handle* )**

Rejestracja funkcji wywoływanej po puszczeniu klawisza na klawiaturze.

**Parametry**

<i>handle</i>	Wskaźnik na funkcję. Argument to kod klawisza
---------------	---

**9.4.2.4 bool WindowEngine::addMouseDownEventHandler ( void(\*)(int, int, int) *handle* )**

Rejestracja funkcji wywoływanej po wciśnięciu przycisku myszy.

**Parametry**

<i>handle</i>	Wskaźnik na funkcję. Argumenty to kod klawisza i położenie myszy (x, y)
---------------	---

**9.4.2.5 bool WindowEngine::addMouseMotionEventHandler ( void(\*)(int, int, int) *handle* )**

Rejestracja funkcji wywoływanej po ruszeniu myszy.

**Parametry**

<i>handle</i>	Wskaźnik na funkcję. Argumenty to kod klawisza i położenie myszy (x, y)
---------------	---

**9.4.2.6 bool WindowEngine::addMouseUpEventHandler ( void(\*)(int, int, int) *handle* )**

Rejestracja funkcji wywoływanej po puszczeniu przycisku myszy.

**Parametry**

<i>handle</i>	Wskaźnik na funkcję. Argumenty to kod klawisza i położenie myszy (x, y)
---------------	---

**9.4.2.7 bool WindowEngine::init ( RenderType *rt* = SDL, WaitType *wt* = DELAY )**

Inicjalizacja ekranu.

**Parametry**

<i>in</i>	<i>rt</i>	Używana biblioteka graficzna. Nie ma nic poza SDLem
<i>in</i>	<i>wt</i>	Sposób reagowania na koniec danej klatki.





## Rozdział 10

# Dokumentacja klas

### 10.1 Dokumentacja klasy Sprite::Anim

Informacje o animacji.

```
#include <sprite.h>
```

#### Komponenty

- class [AnimFrame](#)  
*Klatka animacji.*

#### Metody publiczne

- **Anim** (float aspd, int fret)
- void [clear](#) ()  
*Czyści wszystkie animacje.*
- void [addFrame](#) (int x, int y, int w, int h, int spotx=0, int spoty=0, int actx=0, int acty=0, int boxx=0, int boxy=0, int boxw=0, int boxh=0)  
*Dodaje klatkę o podanych parametrach.*
- const [AnimFrame](#) & [getFrame](#) (unsigned int i)  
*Zwraca klatkę o podanym numerze.*
- void [setAspd](#) (float sa)  
*Ustawia szybkość animacji na podaną wartość*
- void [setFret](#) (int sa)  
*Ustawia klatkę powrotu na podaną*

- float `getAspd ()`  
*Zwraca aktualną predkość animacji.*
- int `getFret ()`  
*Zwraca aktualną klatkę powrotu.*
- int `getFrameCount ()`  
*Zwraca ilość klatek.*

### 10.1.1 Opis szczegółowy

Informacje o animacji.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- `include/sprite.h`

## 10.2 Dokumentacja klasy `Sprite::Anim::AnimFrame`

Klatka animacji.

```
#include <sprite.h>
```

### Metody publiczne

- **`AnimFrame`** (int x, int y, int w, int h, int spotx=0, int spoty=0, int actx=0, int acty=0, int boxx=0, int boxy=0, int boxw=0, int boxh=0)

### Atrybuty publiczne

- int **x**
- int **y**
- int **w**
- int **h**
- int **spotx**
- int **spoty**
- int **actx**
- int **acty**
- int **boxx**
- int **boxy**
- int **boxw**
- int **boxh**

### 10.2.1 Opis szczegółowy

Klatka animacji. Za dużo by pisać, zwykłego śmiertelnika raczej to nie powinno interesować. Czemu jest publiczne, pytasz? A czemu nie~?

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/sprite.h

## 10.3 Dokumentacja klasy Client

Połączenie z serwerem.

```
#include <Client.hpp>
```

### Metody publiczne

- **Client** (boost::asio::io\_service &io\_service, tcp::resolver::iterator endpoint\_iterator)

*Konstruktor połączenia Konstruktor: Ustawia handler połączenia.*

- **Client** (boost::asio::io\_service &io\_service, const char \*host, const char \*port)

*Konstruktor połączenia Konstruktor: Ustawia handler połączenia.*

- void **write** (const **Message** &msg)

*metoda wysyłająca wiadomość*

- void **close** ()

*metoda zamykająca połączenie metoda binduje handler do\_close z metodą post socketu*

- void **send** (const std::string &m)

### 10.3.1 Opis szczegółowy

Połączenie z serwerem. Klasa odpowiedzialna za obsługę połączenia z serwerem

#### Autor

Paweł Ściegienny

### 10.3.2 Dokumentacja konstruktora i destruktor

- 10.3.2.1 **Client::Client** ( boost::asio::io\_service & io\_service, tcp::resolver::iterator endpoint\_iterator ) [inline]

Konstruktor połączenia Konstruktor: Ustawia handler połączenia.

**Parametry**

in	<i>io_service</i>	referencja do obiektu boost::asio::io_service reprezentującego swego rodzaju socket
in	<i>endpoint_iterator</i>	tcp::resolver::iterator wskazujący na hostname i port

**10.3.2.2** `Client::Client ( boost::asio::io_service & io_service, const char * host, const char * port ) [inline]`

Konstruktor połączenia Konstruktor. Ustawia handler połączenia.

**Parametry**

in	<i>host</i>	hostname
in	<i>ip</i>	adres ip

**10.3.3 Dokumentacja funkcji składowych**

**10.3.3.1** `void Client::write ( const Message & msg ) [inline]`

metoda wysyłająca wiadomość

metoda bindująca handler do\_writer z metodą post socketu

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/Client.hpp

**10.4 Dokumentacja struktury Cube****Metody publiczne**

- **Cube** (int x=0, int y=0, int z=0, unsigned int col=0xFFFFFFFF)
- **Cube** (const [Cube](#) &c)
- void **reset** ()

**Atrybuty publiczne**

- int **x**
- int **y**
- int **z**
- unsigned int **col**
- [Vertex](#) **verts** [VERT\_COUNT]

### Statyczne atrybuty publiczne

- static const int **VERT\_COUNT** = 24

Dokumentacja dla tej struktury została wygenerowana z pliku:

- src/screen.cpp

## 10.5 Dokumentacja klasy GameEngine

Główny silnik gry.

```
#include <gameengine.h>
```

### Metody publiczne

- **GameEngine** (uint16 size, uint16 players)  
*Tworzy plansze.*
- **~GameEngine** ()  
*Destruktor zwalniający pamięć*
- **uint16 EndTurn** ()  
*Konczy ture.*
- **uint16 ActPlayer** () const  
*Aktualny gracz.*
- **RETURNS::MOVE Move** (const **Vertex** &src, const **Vertex** &dst, uint16 num)  
*Przenosi jednostki z jednej planety na drugą*
- void **RemovePlayer** (uint16 player)  
*Usuwa gracza.*
- **Planet & GetPlanet** (const **Vertex** &src)
- **uint16 GetSize** ()

### 10.5.1 Opis szczegółowy

Główny silnik gry. Klasa zajmuje się przeliczaniem rozgrywki, położeniem jednostek, systemem walki

#### Autor

Marcin TORGiren Fabrykowski

## 10.5.2 Dokumentacja konstruktora i destruktora

### 10.5.2.1 `GameEngine::GameEngine ( uint16 size, uint16 players )`

Tworzy plansze.

Konstruktor. Tworzy plansze o zadanym rozmiarze, oraz umieszcza na niej graczy. Plansza ma postać sześciangu o wymiarach:  $size * size * size$ . Gracze na planszy rozmieszczeni są w losowy sposób.

#### Parametry

<i>in</i>	<i>size</i>	Rozmiar planszy.
<i>in</i>	<i>players</i>	Liczba graczy biorących udział w rozgrywce

## 10.5.3 Dokumentacja funkcji składowych

### 10.5.3.1 `uint16 GameEngine::ActPlayer ( ) const`

Aktualny gracz.

Zwraca numer aktualnego gracza.

#### Zwraca

Numer aktualnego gracza.

### 10.5.3.2 `uint16 GameEngine::EndTurn ( )`

Konczy ture.

Metoda kończąca ture danego gracza. W tej chwili dodawane są jednostki dla "jeszcze" aktualnego gracza.

#### Zwraca

Zwraca numer następnego gracza.

### 10.5.3.3 `RETURNS::MOVE GameEngine::Move ( const Vertex & src, const Vertex & dst, uint16 num )`

Przenosi jednostki z jednej planety na drugą

Wykonuje operacje przeniesienia jednostek z planety źródłowej na docelową. Metoda sprawdza czy dana operacja jest możliwa (np: czy **num** <= liczba\_jednostek-1)

#### Parametry

<i>in</i>	<i>src</i>	Współrzędne planety źródłowej
<i>in</i>	<i>dst</i>	Współrzędne planety docelowej
<i>in</i>	<i>num</i>	Liczba jednostek do przeniesienia

**Zwraca**

Zwraca ERRORS::MOVE

**10.5.3.4 void GameEngine::RemovePlayer ( uint16 player )**

Usuwa gracza.

Metoda usuwająca gracza z rozgrywki. Wszystkie ewentualne jednostki należące do tego gracza stają się jednostkami neutralnymi. Posiadane planety również stają się neutralne.

Możliwe do wykorzystania zarówno przy odłączeniu się gracza jak również przy pokazaniu danego gracza

**Parametry**

in	<i>player</i>	Numer gracza który ma zostać usunięty
----	---------------	---------------------------------------

Dokumentacja dla tej klasy została wygenerowana z plików:

- include/gameengine.h
- src/gameengine.cpp

**10.6 Dokumentacja klasy GameEngineClient**

Klasa silnika gry dla klienta.

```
#include <gameengineclient.h>
```

**10.6.1 Opis szczegółowy**

Klasa silnika gry dla klienta.

**Autor**

Marcin TORGiren Fabrykowski

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/gameengineclient.h

**10.7 Dokumentacja klasy Message**

Przesyłana wiadomość.

```
#include <Message.hpp>
```

### Typy publiczne

- enum { **header\_length** = 4 }  
*maksymalna długość nagłówka*
- enum { **max\_body\_length** = 512 }  
*maksymalna długość wiadomości*

### Metody publiczne

- [Message](#) ()  
*Konstruktor.*
- **Message** (const [Message](#) &src)
- const char \* [data](#) () const  
*metoda zwracająca treść wiadomości razem z nagłówkiem*
- char \* [data](#) ()  
*metoda zwracająca treść wiadomości razem z nagłówkiem*
- size\_t [length](#) () const  
*metoda zwracająca długość wiadomości*
- const char \* [body](#) () const  
*metoda zwracająca treść wiadomości*
- char \* [body](#) ()  
*metoda zwracająca treść wiadomości*
- size\_t [body\\_length](#) () const  
*metoda zwracająca długość treści*
- void [body\\_length](#) (size\_t length)  
*metoda zwracająca długość treści*
- bool [decode\\_header](#) ()  
*metoda odczytująca nagłówki*
- void [encode\\_header](#) ()  
*metoda zapisująca nagłówki*
- void **source** (unsigned src)
- unsigned **source** () const



### 10.7.1 Opis szczegółowy

Przesyłana wiadomość. Klasa odpowiedzialna za poprawne informacje o wiadomości

#### Autor

Paweł Ściegienny

### 10.7.2 Dokumentacja konstruktora i destruktor

#### 10.7.2.1 `Message::Message ( )` [inline]

Konstruktor.

Konstruktor domyślny - inicjalizuje długość wiadomości

### 10.7.3 Dokumentacja funkcji składowych

#### 10.7.3.1 `size_t Message::length ( ) const` [inline]

metoda zwracająca długość wiadomości

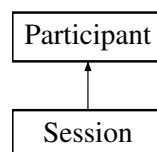
metoda zwracająca długość wiadomości WRAZ z długością nagłówka

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/Message.hpp

## 10.8 Dokumentacja klasy Participant

Diagram dziedziczenia dla Participant



### Metody publiczne

- virtual void **deliver** (const `Message` &msg)=0

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/Participant.hpp

## 10.9 Dokumentacja klasy Planet

Klasa planety.

```
#include <planet.h>
```

### Metody publiczne

- [Planet \(\)](#)  
*Tworzy planete.*
- [uint16 RetGracz \(\)](#) const  
*Zwraca numer gracza-właściciela planety.*
- [uint16 RetOkupant \(\)](#) const  
*Zwraca numer gracza-okupanta planety.*
- [uint16 RetPoziom \(\)](#) const  
*Zwracam poziom zaawansowania okupacji.*
- [uint16 RetJednostki \(\)](#) const  
*Zwraca ilosc jednostek na planecie Funkcja wraca liczbę floty znajdującej się na planecie. Jeśli planeta nie jest okupowana, jest to liczba jednostek gracza będącego właścicielem, natomiast w przypadku okupacji, jest to liczba jednostek okupanta.*
- [FightResult Atak \(uint16 ile, uint16 kogo\)](#)  
*Przeprowadza atak na planete Przeprowadza atak zadanej ilości jednostek na planete.*
- void [SetPlayer \(uint16 gracz\)](#)  
*Ustawia nowego właściciela planety.*
- void [EndTurn \(\)](#)  
*Kończy turę na danej planecie.*
- [RETURNS::MOVE Zabierz \(uint16 ile\)](#)
- void [Dodaj \(uint16 ile\)](#)

### Przyjaciele

- class [GameEngine](#)

#### 10.9.1 Opis szczegółowy

Klasa planety. Opisuje właściwości planety - elementarnej jednostki przestrzeni

**Autor**

Marcin TORGiren Fabrykowski

**10.9.2 Dokumentacja konstruktora i destruktor****10.9.2.1 Planet::Planet ( )**

Tworzy planete.

Konstruktor. Tworzy neutralna planete z losowa (od 0 do 9) liczbą jednostek

**10.9.3 Dokumentacja funkcji składowych****10.9.3.1 FightResult Planet::Atak ( uint16 ile, uint16 kogo )**

Przeprowadza atak na planete Przeprowadza atak zadanej ilości jednostek na planete.

**Parametry**

<i>ile</i>	Liczba jednostek wroga, biorąca udział w ataku
<i>kogo</i>	Numer gracza który przeprowadza atak

**Zwraca**

Zwraca wektor reprezentujący kolejne starcia, zawierający pary wektorów rzutów  
W przypadku mniejszej ilości jednostek po którejś ze stron, w miejsce rzutu wstawiana jest wartość 0

**10.9.3.2 void Planet::EndTurn ( )**

Kończy turę na danej planecie.

W przypadku okupowania planety następuje zdobywanie/zdejmowanie flagi.

W przypadku posiadanych planet, następuje tworzenie nowych jednostek

**10.9.3.3 uint16 Planet::RetGracz ( ) const**

Zwraca numer gracza-właściciela planety.

Zwraca numer gracza który jest aktualnie posiadaczem planety. Planeta może być okupowana przez innego gracza i wciąż być w posiadaniu starego właściciela

**Zwraca**

Zwraca numer gracza który jest właścicielem planety, bądź NULL jeśli takiego nie ma

#### 10.9.3.4 uint16 Planet::RetJednostki ( ) const

Zwraca ilość jednostek na planecie. Funkcja zwraca liczbę floty znajdującej się na planecie. Jeśli planeta nie jest okupowana, jest to liczba jednostek gracza będącego właścicielem, natomiast w przypadku okupacji, jest to liczba jednostek okupanta.

##### Zwraca

Liczba jednostek właściciela planety. W przypadku gdy planeta jest okupowana, to jest liczba jednostek okupanta.

#### 10.9.3.5 uint16 Planet::RetOkupant ( ) const

Zwraca numer gracza-okupanta planety.

Zwraca numer gracza który jest aktualnie okupantem planety.

##### Zwraca

Numer gracza który okupuje planete, bądź NULL jeśli takowego nie ma.

#### 10.9.3.6 uint16 Planet::RetPoziom ( ) const

Zwracam poziom zaawansowania okupacji.

Zwraca aktualny poziom okupacji. Wartość OCCUPY\_MAX oznacza, że planeta nie jest już okupowana i jest w pełni przejęta.

##### Zwraca

Poziom okupacji, bądź OCCUPY\_MAX w przypadku gdy planeta nie jest okupowana i jest w pełni przejęta.

#### 10.9.3.7 void Planet::SetPlayer ( uint16 gracz )

Ustawia nowego właściciela planety.

Metoda która ustawia nowego właściciela planety.

##### Parametry

<i>gracz</i>	Numer gracza będącego nowym właścicielem
--------------	--

Dokumentacja dla tej klasy została wygenerowana z plików:

- include/planet.h
- src/planet.cpp

## 10.10 Dokumentacja klasy Point

Klasa położenia w przestrzeni.

```
#include <point.h>
```

### Atrybuty publiczne

- [uint16 itsX](#)
- [uint16 itsY](#)
- [uint16 itsZ](#)

### 10.10.1 Opis szczegółowy

Klasa położenia w przestrzeni. Obrazuje położenie punktu w przestrzeni planszy

#### Autor

Marcin TORGiren Fabrykowski

### 10.10.2 Dokumentacja atrybutów składowych

#### 10.10.2.1 `uint16 Point::itsX`

Położenie na osi X

#### 10.10.2.2 `uint16 Point::itsY`

Położenie na osi Y

#### 10.10.2.3 `uint16 Point::itsZ`

Położenie na osi Z

Dokumentacja dla tej klasy została wygenerowana z pliku:

- `include/point.h`

## 10.11 Dokumentacja klasy Room

### Metody publiczne

- void **join** (Participant\_ptr participant)
- void **leave** (Participant\_ptr participant)
- void **deliver** (const [Message](#) &msg)

- unsigned **search** ([Participant](#) \*participant)
- [Message](#) **todo** ()
- void **todo** (const [Message](#) msg)

### 10.11.1 Dokumentacja funkcji składowych

#### 10.11.1.1 void Room::deliver ( const Message & msg )

początek przykładu udupiania części pakietów

koniec przykładu udupiania części pakietów

Dokumentacja dla tej klasy została wygenerowana z plików:

- include/Room.hpp
- src/Room.cpp

## 10.12 Dokumentacja klasy Server

### Metody publiczne

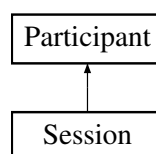
- **Server** (boost::asio::io\_service &io\_service, const tcp::endpoint &endpoint)
- void **handle\_accept** (Session\_ptr session, const boost::system::error\_code &error)
- void **send** (const std::string &m)
- [Message](#) **receive** ()

Dokumentacja dla tej klasy została wygenerowana z plików:

- include/Server.hpp
- src/Server.cpp

## 10.13 Dokumentacja klasy Session

Diagram dziedziczenia dla Session



### Metody publiczne

- **Session** (boost::asio::io\_service &io\_service, [Room](#) &room)
- tcp::socket & **socket** ()
- void **start** ()
- void **deliver** (const [Message](#) &msg)
- void **handle\_read\_header** (const boost::system::error\_code &error)
- void **handle\_read\_body** (const boost::system::error\_code &error)
- void **handle\_write** (const boost::system::error\_code &error)

Dokumentacja dla tej klasy została wygenerowana z plików:

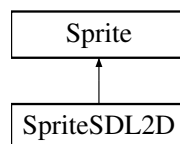
- include/Session.hpp
- src/Session.cpp

## 10.14 Dokumentacja klasy Sprite

Klasa zajmująca się wczytaniem, wyświetlaniem i ogólnie obsługą obrazków.

```
#include <sprite.h>
```

Diagram dziedziczenia dla Sprite



### Komponenty

- class [Anim](#)  
*Informacje o animacji.*
- class [SpritePtr](#)  
*Smart Pointer na sprite. Zwalnia sprite jeśli nikt go nie używa.*

### Metody publiczne

- **Sprite** (const std::string &name="", int w=0, int h=0)
- const std::string & **getName** ()
- void **getDim** (int &gw, int &gh)
- int **getW** ()
- int **getH** ()

- [Anim](#) & **getAnim** (unsigned int i)
- unsigned int **getAnimCount** ()
- virtual void **animate** (int anim, float &frame, float spd=-1.0f)
- virtual void **print** (float x, float y, float z, int anim, int frame, unsigned char alpha=255u, float px=1.0f, float py=1.0f, unsigned char r=255u, unsigned char g=255u, unsigned char b=255u)=0
- virtual void **flush** ()=0

### Statyczne metody publiczne

- static void **print** ()
- static void **clear** ()
- static void **reload** ()
- static [Sprite](#) \* **load** (const std::string &name, bool force=false)

*Wczytuje grafikę o podanej nazwie.*

### Metody chronione

- void **addSpritePtr** ([SpritePtr](#) \*s)
- void **delSpritePtr** ([SpritePtr](#) \*s)
- void **setSpritePtrs** ([Sprite](#) \*s)
- virtual bool **loadGfx** (const std::string &name)=0
- virtual bool **loadMask** (void \*pixs, int w, int h, int bpp)
- virtual bool **loadAnims** (const std::string &name)

### Atrybuty chronione

- std::set< [SpritePtr](#) \* > **spritePtrs**
- std::string **name**
- int **w**
- int **h**
- bool \* **mask**
- std::vector< [Anim](#) > **anims**
- std::map< std::string, [Anim](#) \* > **animNames**

### Statyczne atrybuty chronione

- static std::map< std::string, [Sprite](#) \* > **sprites**



### 10.14.1 Opis szczegółowy

Klasa zajmująca się wczytaniem, wyświetlaniem i ogólnie obsługą obrazków. Po niej powinny dziedziczyć wersje zajmujące się implementacją tych operacji w wybranej bibliotece graficznej. Aktualnie zrobione są dla SDL i OpenGL, jednak tutaj dostępny jest tylko SDL.

Dokumentacja dla tej klasy została wygenerowana z plików:

- `include/sprite.h`
- `src/sprite.cpp`

## 10.15 Dokumentacja klasy `Sprite::SpritePtr`

Smart Pointer na `sprite`. Zwalnia `sprite` jeśli nikt go nie używa.

```
#include <sprite.h>
```

### Metody publiczne

- `SpritePtr` (`Sprite *s`)
- `void operator=` (`Sprite *s`)
- `void setSprite` (`Sprite *s`)
- `void setAnim` (`int sa`)
- `void setSpd` (`float ss`)
- `void animate` ()
- `void print` (`float x`, `float y`, `float z`, `unsigned char alpha=255u`, `float px=1.0f`, `float py=1.0f`, `unsigned char r=255u`, `unsigned char g=255u`, `unsigned char b=255u`)

### Atrybuty publiczne

- `Sprite * sprite`

### 10.15.1 Opis szczegółowy

Smart Pointer na `sprite`. Zwalnia `sprite` jeśli nikt go nie używa. Dodatkowo posiada obsługę animacji i potrafi odpowiednio zareagować w przypadku ponownego wczytania `sprite` dla innej biblioteki graficznej.

Dokumentacja dla tej klasy została wygenerowana z plików:

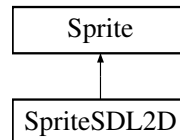
- `include/sprite.h`
- `src/sprite.cpp`

## 10.16 Dokumentacja klasy SpriteSDL2D

Klasa sprite oparta na SDLu.

```
#include <sprite_sdl_2d.h>
```

Diagram dziedziczenia dla SpriteSDL2D



### Metody publiczne

- **SpriteSDL2D** (const std::string &name="", int w=0, int h=0)
- void **print** (float x, float y, float z, int anim, int frame, unsigned char alpha=255u, float px=1.0f, float py=1.0f, unsigned char r=255u, unsigned char g=255u, unsigned char b=255u)
- void **flush** ()

*Dobre pytanie. Sam nie wiem.*

#### 10.16.1 Opis szczegółowy

Klasa sprite oparta na SDLu. Zajmuje się wyświetleniem i wczytaniem obrazka używając SDLa. 'Gdzieś' jest wersja robiąca to samo dla OpenGLa, ale tutaj nie ma dla niej miejsca.

#### 10.16.2 Dokumentacja funkcji składowych

##### 10.16.2.1 void SpriteSDL2D::flush ( ) [inline, virtual]

Dobre pytanie. Sam nie wiem.

Tak serio jest to zrobione pod kątem OpenGL'a (array buffer, vbo).

Implementuje [Sprite](#).

##### 10.16.2.2 void SpriteSDL2D::print ( float x, float y, float z, int anim, int frame, unsigned char alpha = 255u, float px = 1.0f, float py = 1.0f, unsigned char r = 255u, unsigned char g = 255u, unsigned char b = 255u ) [virtual]

#### Parametry

in	x	Współrzędna x
in	y	Współrzędna y

in	<i>z</i>	Współrzędna z
in	<i>anim</i>	Numer animacji
in	<i>frame</i>	Klatka animacji
in	<i>alpha</i>	Przeźroczystość, 0-255
in	<i>px</i>	Parallax scrolling, poziomy
in	<i>py</i>	Parallax scrolling, pionowy
in	<i>r</i>	Czerwony
in	<i>g</i>	Zielony
in	<i>b</i>	Niebieski

Implementuje [Sprite](#).

Dokumentacja dla tej klasy została wygenerowana z plików:

- include/sprite\_sdl\_2d.h
- src/sprite\_sdl\_2d.cpp

## 10.17 Dokumentacja struktury Vertex

Prosty vertex/wektor 3D, zawiera podstawowe operacje.

```
#include <vertex.h>
```

### Metody publiczne

- **Vertex** (float x, float y, float z)
- **Vertex & operator=** (const **Vertex** &v)
- **bool operator==** (const **Vertex** &v) const
- **bool eq2d** (const **Vertex** &v) const
- **Vertex operator+** (const **Vertex** &v) const
- **Vertex operator-** (const **Vertex** &v) const
- **Vertex operator\*** (float v) const
- **Vertex operator/** (float v) const
- **Vertex cross** (const **Vertex** &v) const  
*Iloczyn wektorowy. Z pewnych powodów pomija z. "Taki ficzer".*
- **Vertex crossz** (const **Vertex** &v) const  
*Iloczyn wektorowy.*
- **float dot** (const **Vertex** &v) const  
*Iloczyn skalarny.*
- **float len** () const  
*Długość wektora.*

**Atrybuty publiczne**

- float **x**
- float **y**
- float **z**

**10.17.1 Opis szczegółowy**

Prosty vertex/wektor 3D, zawiera podstawowe operacje. Funkcje rysujące przystosowane są do ułożenia wierzchołków przeciwnie do ruchu wskazówek zegara (CCW)

Dokumentacja dla tej struktury została wygenerowana z pliku:

- include/vertex.h

## Rozdział 11

# Dokumentacja plików

### 11.1 Dokumentacja pliku include/consts.h

```
#include <stdint.h>
#include <vector>
```

#### Przestrzenie nazw

- namespace `RETURNS`

#### Definicje typów

- typedef uint32\_t `uint`
- typedef uint16\_t `uint16`
- typedef std::pair< std::vector< `uint16` >, std::vector< `uint16` > > `FightResultRow`
- typedef std::vector< `FightResultRow` > `FightResult`

#### Wyliczenia

- enum `RETURNS::MOVE` {  
    `TOO_MUCH`, `OUT_OF_AREA`, `NOT_ANY`, `MOVE_OK`,  
    `MOVE_FIGHT` }

#### Zmienne

- const int `SCREENWIDTH` = 640
- const int `SCREENHEIGHT` = 480
- const int `BPP` = 32

- const char **GAMENAME** [] = "RTTTT - Risky Tic Tak Toe - Bitwa o Alfa Centauri 4000AD"
- const int **FPSDELAY** = 1000/50
- const float **DEGTORAD** = 3.141592653589793f/180.0f
- const float **RADTO DEG** = 180.0f/3.141592653589793f
- const int **OCCUPY\_MAX** = 5

### 11.1.1 Opis szczegółowy

### 11.1.2 Dokumentacja definicji typów

#### 11.1.2.1 typedef std::vector<FightResultRow> FightResult

Wektor wierszy logów z walki

#### 11.1.2.2 typedef std::pair<std::vector<uint16>,std::vector<uint16> > FightResultRow

Struktura wiersza logów z walki

#### 11.1.2.3 typedef uint32\_t uint

Liczba całkowita o rozmiarze 32bitów

#### 11.1.2.4 typedef uint16\_t uint16

Liczba całkowita o rozmiarze 16 bitów

### 11.1.3 Dokumentacja zmiennych

#### 11.1.3.1 const int OCCUPY\_MAX = 5

Maksymalny poziom okupowanej planety powodujący jej przejęcie

# Skorowidz

ActPlayer  
    GameEngine, [32](#)  
addKeyDownEventHandler  
    WindowEngine, [23](#)  
addKeyPressedEventHandler  
    WindowEngine, [23](#)  
addKeyUpEventHandler  
    WindowEngine, [24](#)  
addMouseDownEventHandler  
    WindowEngine, [24](#)  
addMouseMotionEventHandler  
    WindowEngine, [24](#)  
addMouseUpEventHandler  
    WindowEngine, [24](#)  
Atak  
    Planet, [37](#)  
  
Client, [29](#)  
    Client, [29](#), [30](#)  
    write, [30](#)  
consts.h  
    FightResult, [48](#)  
    FightResultRow, [48](#)  
    OCCUPY\_MAX, [48](#)  
    uint, [48](#)  
    uint16, [48](#)  
Cube, [30](#)  
  
deliver  
    Room, [40](#)  
Dokumentacja katalogu include/, [15](#)  
Dokumentacja katalogu src/, [15](#)  
Drawing, [17](#)  
    drawLine, [18](#)  
    drawQuad, [18](#)  
    putPix, [19](#)  
    setObj, [19](#)  
drawLine  
    Drawing, [18](#)  
drawQuad  
    Drawing, [18](#)  
  
EndTurn  
    GameEngine, [32](#)  
    Planet, [37](#)  
  
FightResult  
    consts.h, [48](#)  
FightResultRow  
    consts.h, [48](#)  
flush  
    SpriteSDL2D, [44](#)  
  
GameEngine, [31](#)  
    ActPlayer, [32](#)  
    EndTurn, [32](#)  
    GameEngine, [32](#)  
    Move, [32](#)  
    RemovePlayer, [33](#)  
GameEngineClient, [33](#)  
  
include/consts.h, [47](#)  
init  
    WindowEngine, [24](#)  
itsX  
    Point, [39](#)  
itsY  
    Point, [39](#)  
itsZ  
    Point, [39](#)  
  
length  
    Message, [35](#)  
  
Message, [33](#)  
    length, [35](#)  
    Message, [35](#)  
MOVE  
    RETURNS, [19](#)  
Move  
    GameEngine, [32](#)  
  
OCCUPY\_MAX  
    consts.h, [48](#)

Participant, [35](#)  
Planet, [36](#)  
    Atak, [37](#)  
    EndTurn, [37](#)  
    Planet, [37](#)  
    RetGracz, [37](#)  
    RetJednostki, [37](#)  
    RetOkupant, [38](#)  
    RetPoziom, [38](#)  
    SetPlayer, [38](#)  
Point, [39](#)  
    itsX, [39](#)  
    itsY, [39](#)  
    itsZ, [39](#)  
print  
    SpriteSDL2D, [44](#)  
putPix  
    Drawing, [19](#)  
  
RemovePlayer  
    GameEngine, [33](#)  
RetGracz  
    Planet, [37](#)  
RetJednostki  
    Planet, [37](#)  
RetOkupant  
    Planet, [38](#)  
RetPoziom  
    Planet, [38](#)  
RETURNS, [19](#)  
    MOVE, [19](#)  
Room, [39](#)  
    deliver, [40](#)  
  
Screen, [20](#)  
Server, [40](#)  
Session, [40](#)  
setObj  
    Drawing, [19](#)  
SetPlayer  
    Planet, [38](#)  
Sprite, [41](#)  
Sprite::Anim, [27](#)  
Sprite::Anim::AnimFrame, [28](#)  
Sprite::SpritePtr, [43](#)  
SpriteSDL2D, [44](#)  
    flush, [44](#)  
    print, [44](#)  
  
uint  
    consts.h, [48](#)  
uint16  
    consts.h, [48](#)  
  
Vertex, [45](#)  
  
WindowEngine, [21](#)  
    addKeyDownEventHandler, [23](#)  
    addKeyPressedEventHandler, [23](#)  
    addKeyUpEventHandler, [24](#)  
    addMouseDownEventHandler, [24](#)  
    addMouseMotionEventHandler, [24](#)  
    addMouseUpEventHandler, [24](#)  
    init, [24](#)  
write  
    Client, [30](#)