

Podręcznik

Wygenerowano przez Doxygen 1.7.1

Mon Feb 6 2012 00:20:42

Spis treści

1	RTTT - Risky Tic Tac Toe	1
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
2	Algorytmy	3
2.1	Algorytm walki	3
3	Lista rzeczy do zrobienia	5
4	Struktura katalogów	7
4.1	Katalogi	7
5	Indeks przestrzeni nazw	9
5.1	Lista przestrzeni nazw	9
6	Indeks klas	11
6.1	Hierarchia klas	11
7	Indeks klas	13
7.1	Lista klas	13
8	Indeks plików	15
8.1	Lista plików	15
9	Dokumentacja katalogów	17
9.1	Dokumentacja katalogu include/	17

9.2 Dokumentacja katalogu src/	18
10 Dokumentacja przestrzeni nazw	19
10.1 Dokumentacja przestrzeni nazw Drawing	19
10.1.1 Opis szczegółowy	21
10.1.2 Dokumentacja funkcji	21
10.1.2.1 clearZBuff	21
10.1.2.2 drawLine	21
10.1.2.3 drawQuad	21
10.1.2.4 drawTriangle	21
10.1.2.5 putPix	22
10.1.2.6 SameSide	22
10.1.2.7 setColor	22
10.1.2.8 setObj	22
10.1.2.9 setSurface	22
10.1.3 Dokumentacja zmiennych	22
10.1.3.1 color	22
10.2 Dokumentacja przestrzeni nazw RETURNS	23
10.2.1 Opis szczegółowy	23
10.2.2 Dokumentacja typów wyliczanych	23
10.2.2.1 MOVE	23
10.3 Dokumentacja przestrzeni nazw Screen	24
10.3.1 Opis szczegółowy	27
10.3.2 Dokumentacja funkcji	27
10.3.2.1 addMessage	27
10.3.2.2 kup	27
10.3.2.3 mdown	28
10.3.2.4 mmove	28
10.3.2.5 mroll	28
10.3.2.6 mup	28
10.3.2.7 rotateArb	29
10.3.2.8 setCurrentPlayerID	29
10.3.2.9 setPlayerID	29
10.3.2.10 updateArea	29

10.3.3 Dokumentacja zmiennych	29
10.3.3.1 cid	29
10.3.3.2 curr	29
10.3.3.3 id	30
10.3.3.4 info	30
10.4 Dokumentacja przestrzeni nazw WindowEngine	30
10.4.1 Opis szczegółowy	32
10.4.2 Dokumentacja funkcji	32
10.4.2.1 addKeyDownEventHandler	32
10.4.2.2 addKeyPressedEventHandler	33
10.4.2.3 addKeyUpEventHandler	33
10.4.2.4 addMouseDownEventHandler	33
10.4.2.5 addMouseMotionEventHandler	33
10.4.2.6 addMouseUpEventHandler	33
10.4.2.7 init	34
11 Dokumentacja klas	35
11.1 Dokumentacja klasy Sprite::Anim	35
11.1.1 Opis szczegółowy	36
11.2 Dokumentacja klasy Sprite::Anim::AnimFrame	36
11.2.1 Opis szczegółowy	37
11.3 Dokumentacja klasy Client	37
11.3.1 Opis szczegółowy	37
11.3.2 Dokumentacja funkcji składowych	38
11.3.2.1 close	38
11.3.2.2 create	38
11.3.2.3 send	38
11.3.2.4 write	38
11.4 Dokumentacja struktury Cube	38
11.5 Dokumentacja klasy GameEngine	39
11.5.1 Opis szczegółowy	40
11.5.2 Dokumentacja konstruktora i destruktorów	40
11.5.2.1 GameEngine	40
11.5.3 Dokumentacja funkcji składowych	40

11.5.3.1	AddPlayer	40
11.5.3.2	CanDoAction	41
11.5.3.3	EndTurn	41
11.5.3.4	IsEndGame	41
11.5.3.5	Move	41
11.5.3.6	RemovePlayer	42
11.6	Dokumentacja klasy GameEngineBase	42
11.6.1	Dokumentacja konstruktora i destruktora	43
11.6.1.1	GameEngineBase	43
11.6.2	Dokumentacja funkcji składowych	43
11.6.2.1	ActPlayer	43
11.6.2.2	GetPlanet	43
11.6.2.3	GetSize	44
11.7	Dokumentacja klasy GameEngineClient	44
11.7.1	Opis szczegółowy	45
11.7.2	Dokumentacja funkcji składowych	45
11.7.2.1	Create	45
11.7.2.2	EndGame	45
11.7.2.3	MainLoop	46
11.7.2.4	PlanetUpdate	46
11.7.2.5	SendEndTurn	46
11.7.2.6	SendMove	46
11.8	Dokumentacja klasy Message	47
11.8.1	Opis szczegółowy	48
11.8.2	Dokumentacja konstruktora i destruktora	48
11.8.2.1	Message	48
11.8.2.2	Message	48
11.8.3	Dokumentacja funkcji składowych	48
11.8.3.1	length	48
11.8.3.2	operator=	49
11.8.3.3	source	49
11.9	Dokumentacja klasy Participant	49
11.9.1	Opis szczegółowy	49
11.10	Dokumentacja klasy Planet	50

11.10.1 Opis szczegółowy	51
11.10.2 Dokumentacja konstruktora i destruktora	51
11.10.2.1 Planet	51
11.10.3 Dokumentacja funkcji składowych	51
11.10.3.1 Atak	51
11.10.3.2 Dodaj	51
11.10.3.3 EndTurn	52
11.10.3.4 RetGracz	52
11.10.3.5 RetJednostki	52
11.10.3.6 RetOkupant	52
11.10.3.7 RetPoziom	53
11.10.3.8 SetPlayer	53
11.10.3.9 Zabierz	53
11.11Dokumentacja klasy Point	53
11.11.1 Opis szczegółowy	54
11.11.2 Dokumentacja atrybutów składowych	54
11.11.2.1 itsX	54
11.11.2.2 itsY	54
11.11.2.3 itsZ	54
11.12Dokumentacja klasy Room	54
11.12.1 Opis szczegółowy	55
11.12.2 Dokumentacja funkcji składowych	55
11.12.2.1 search	55
11.13Dokumentacja klasy Server	55
11.13.1 Dokumentacja konstruktora i destruktora	56
11.13.1.1 ~Server	56
11.13.2 Dokumentacja funkcji składowych	56
11.13.2.1 receive	56
11.13.2.2 send	56
11.13.2.3 send	57
11.14Dokumentacja klasy Session	57
11.14.1 Opis szczegółowy	58
11.15Dokumentacja klasy Sprite	58
11.15.1 Opis szczegółowy	60

11.15.2 Dokumentacja funkcji składowych	61
11.15.2.1 animate	61
11.15.2.2 flush	61
11.15.2.3 loadGfx	61
11.15.2.4 loadMask	61
11.15.2.5 print	62
11.15.2.6 reload	62
11.15.2.7 setSpritePtrs	62
11.16 Dokumentacja klasy Sprite::SpritePtr	63
11.16.1 Opis szczegółowy	63
11.17 Dokumentacja klasy SpriteSDL2D	63
11.17.1 Opis szczegółowy	64
11.17.2 Dokumentacja funkcji składowych	64
11.17.2.1 flush	64
11.17.2.2 print	64
11.18 Dokumentacja klasy Text	65
11.18.1 Opis szczegółowy	67
11.18.2 Dokumentacja składowych wyliczanych	68
11.18.2.1 Align	68
11.18.3 Dokumentacja funkcji składowych	68
11.18.3.1 addStr	68
11.18.3.2 getLineLen	68
11.18.3.3 getWordLen	68
11.18.3.4 operator+=	68
11.18.3.5 operator+=	69
11.18.3.6 operator=	69
11.18.3.7 operator=	69
11.18.3.8 setAlign	69
11.18.3.9 setDim	69
11.18.3.10 setStr	70
11.18.3.1 lupdate	70
11.19 Dokumentacja struktury Vertex	70
11.19.1 Opis szczegółowy	71
11.19.2 Dokumentacja funkcji składowych	71

11.19.2.1 eq2d	71
12 Dokumentacja plików	73
12.1 Dokumentacja pliku include/consts.h	73
12.1.1 Opis szczegółowy	75
12.1.2 Dokumentacja definicji typów	75
12.1.2.1 FightResult	75
12.1.2.2 FightResultRow	75
12.1.2.3 uint	75
12.1.2.4 uint16	75
12.1.3 Dokumentacja zmiennych	75
12.1.3.1 OCCUPY_MAX	75
12.1.3.2 PLAYER_COLORS	75

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

Lista rzeczy do zrobienia

Składowa `Sprite::print(float x, float y, float z, int anim, int frame, unsigned char alpha=255u, float px=1.0f, float py=1.0f)`

- Alfa całego obrazka
- Parallax scrolling
- Barwienie obrazka

Rozdział 4

Struktura katalogów

4.1 Katalogi

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

include	17
src	18

Rozdział 5

Indeks przestrzeni nazw

5.1 Lista przestrzeni nazw

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

Drawing (Funkcje obsługujące rysowanie)	19
RETURNS	23
Screen (Chyba cała logika okienka jest tutaj zawarta)	24
WindowEngine (Tworzenie okienka, obsługa zdarzeń)	30

Rozdział 6

Indeks klas

6.1 Hierarchia klas

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

Sprite::Anim	35
Sprite::Anim::AnimFrame	36
Client	37
Cube	38
GameEngineBase	42
GameEngine	39
GameEngineClient	44
Message	47
Participant	49
Session	57
Planet	50
Point	53
Room	54
Server	55
Sprite	58
SpriteSDL2D	63
Sprite::SpritePtr	63
Text	65
Vertex	70

Rozdział 7

Indeks klas

7.1 Lista klas

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

Sprite::Anim (Informacje o animacji)	35
Sprite::Anim::AnimFrame (Klatka animacji)	36
Client (Połączenie z serwerem)	37
Cube	38
GameEngine (Główny silnik gry)	39
GameEngineBase	42
GameEngineClient (Klasa silnika gry dla klienta)	44
Message (Przesyłana wiadomość)	47
Participant (Interfejs pokoju)	49
Planet (Klasa planety)	50
Point (Klasa położenia w przestrzeni)	53
Room (Miejsce gdzie zbiegają się sockety)	54
Server	55
Session (Reprezentacja sesji)	57
Sprite (Klasa zajmująca się wczytaniem, wyświetlaniem i ogólnie obsługą obrazków)	58
Sprite::SpritePtr (Smart Pointer na sprite. Zwalnia sprite jeśli nikt go nie uży- wa)	63
SpriteSDL2D (Klasa sprite oparta na SDLu)	63
Text (Klasa wyświetlająca tekst)	65
Vertex (Prosty vertex/wektor 3D, zawiera podstawowe operacje)	70

Rozdział 8

Indeks plików

8.1 Lista plików

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

include/algorytmy.h	??
include/Client.hpp	??
include/consts.h	73
include/drawing.h	??
include/gameengine.h	??
include/gameenginebase.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/text.h	??
include/vertex.h	??
include/windowengine.h	??

Rozdział 9

Dokumentacja katalogów

9.1 Dokumentacja katalogu include/

Pliki

- plik **algorytmy.h**
- plik **Client.hpp**
- plik [consts.h](#)
- plik **drawing.h**
- plik **gameengine.h**
- plik **gameenginebase.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 **text.h**
- plik **vertex.h**
- plik **windowengine.h**

9.2 Dokumentacja katalogu src/

Pliki

- plik **Client.cpp**
- plik **drawing.cpp**
- plik **gameengine.cpp**
- plik **gameenginebase.cpp**
- plik **gameengineclient.cpp**
- plik **main.cpp**
- plik **main.creammy.cpp**
- plik **main.czaju.cpp**
- plik **main.torgiren.cpp**
- plik **Message.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 **text.cpp**
- plik **windowengine.cpp**

Rozdział 10

Dokumentacja przestrzeni nazw

10.1 Dokumentacja przestrzeni nazw Drawing

Funkcje obsługujące rysowanie.

Funkcje

- void `clearZBuff ()`
Czyszczenie zbuffera.
- void `setSurface (SDL_Surface *srf)`
Ustawia aktualną powierzchnię do rysowania.
- SDL_Surface * `getSurface ()`
Zwraca aktualną powierzchnię do rysowania.
- void `setColor (unsigned int sc)`
Ustawia aktualny kolor.
- unsigned int `getColor ()`
Zwraca aktualny kolor.
- unsigned int `getColorBlend (unsigned int c1, unsigned int c2, float alpha)`
Miesza kolor c1 z c2 w stosunku alpha (1.0 -> 100% c1).
- 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 przeźroczystości 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)`
Sprawdza czy punkty p1 i p2 leżą po tej samej stronie odcinka a, b.
- `bool PointInTriangle (const Vertex &p, const Vertex &a, const Vertex &b, const Vertex &c)`
Sprawdza, czy punkt p leży wewnątrz trójkąta a, b, 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`
Wskaźnik na ekran.
- `void * obj = NULL`
Wskaźnik wpisywany do bufora obiektów.
- `float * zbuff = NULL`
Bufor głębokości.
- `void ** obuff = NULL`
Bufor obiektów.
- `unsigned int color = 0xFFFFFFFF`
Aktualny kolor.
- `const Vertex light (0.7071, 0.7071, 0)`
Kierunek światła.

10.1.1 Opis szczegółowy

Funkcje obsługujące rysowanie.

Autor

crm

10.1.2 Dokumentacja funkcji

10.1.2.1 void Drawing::clearZBuff ()

Czyszczenie zbuffera.

Funkcja również czyści pozostałe bufory (dokładniej mówiąc to jeden bufor, obiektów)

10.1.2.2 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*

10.1.2.3 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 sa to trójkąty *a*, *b*, *c* oraz *c*, *d*, *a*. Proponuję o tym pamiętać.

10.1.2.4 void Drawing::drawTriangle (const Vertex & a, const Vertex & b, const Vertex & c)

Rysuje trójkąt łączący punkty *a*, *b* i *c*.

Algorytm wygląda następująco:

1. Znajdź skrajne punkty i utwórz z nich prostokąt zawierający w sobie cały trójkąt
2. Przejdź po wszystkich punktach wewnątrz prostokąta
3. Jeśli punkt jest wewnątrz trójkąta - wstaw piksel, w przeciwnym razie kontynuuj

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

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

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

10.1.2.6 bool Drawing::SameSide (const Vertex & *p1*, const Vertex & *p2*, const Vertex & *a*, const Vertex & *b*)

Sprawdza czy punkty *p1* i *p2* leżą po tej samej stronie odcinka *a*, *b*.

Thx, <http://www.blackpawn.com/texts/pointinpoly/default.html>

10.1.2.7 void Drawing::setColor (unsigned int *sc*)

Ustawia aktualny kolor.

Kolejność bajtów: 0xAARRGGBB, gdzie AA to alfa, RR to czerwony, GG zielony i BB niebieski

10.1.2.8 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

[in] *obj* Wskaźnik na dowolny obiekt. Musisz pamiętać, co podsyłasz, ponieważ bufor obiektów korzysta z wbudowanego w C++ dynamicznego rzutowania typów (void*)

10.1.2.9 void Drawing::setSurface (SDL_Surface * *srf*)

Ustawia aktualną powierzchnię do rysowania.

Nigdzie nie jest sprawdzane, czy nie jest NULlem.

10.1.3 Dokumentacja zmiennych

10.1.3.1 unsigned int Drawing::color = 0xFFFFFFFF

Aktualny kolor.

Kolejność bajtów: 0xAARRGGBB, gdzie AA to alfa, RR to czerwony, GG zielony i BB niebieski.

10.2 Dokumentacja przestrzeni nazw RETURNS

Definicje typów

- typedef `uint16` `ENDTURN`

Wyliczenia

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

Zmienne

- const `uint16` `NOTHING` = 1
- const `uint16` `NEW_UNIT` = 2
- const `uint16` `FLAG_DOWN` = 4
- const `uint16` `FLAG_UP` = 8
- const `uint16` `PLAYER_OUT` = 16
- const `uint16` `PLAYER_IN` = 32
- const `uint16` `FLAG_ERROR` = 64

10.2.1 Opis szczegółowy

Zawiera komunikaty zwracane z funkcji

10.2.2 Dokumentacja typów wyliczanych

10.2.2.1 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

10.3 Dokumentacja przestrzeni nazw Screen

Chyba cała logika okienka jest tutaj zawarta.

Funkcje

- void `drawCube` (`Cube &c`)
Rysuje kostkę (planetę) na ekran.
- void `mdown` (int x, int y, int key)
Obsługa kliknięcia.
- void `mup` (int x, int y, int key)
Obsługa puszczenia przycisku myszy.
- void `mmove` (int x, int y, int key)
Obsługa ruchu myszą.
- void `mroll` (bool down)
Obsługa kliknięcia rolką.
- void `kup` (int key)
Obsługa puszczenia przycisku na klawiaturze.
- void `init` ()
Inicjalizacja, ustawia handlersy kliknieć i wielkość poziomu na pewną z góry ustaloną wartość.
- void `update` ()
Ibumtralala.
- void `draw` ()
Rysuje pole gry.
- void `setSize` (int size)
Ustawia pole gry na zadana wielkość.
- void `rotateArb` (`Vertex &v`, const `Vertex &s`, const `Vertex &a`, float ang)
Obrót dowolnego wektora względem dowolny wektor zaczepionego w dowolnym punkcie o dowolny kąt.
- void `updateArea` (vector< pair< `Vertex`, `Planet` > > &items)
Aktualizacja pola gry.
- void `addMessage` (const string &msg)
Wypisanie wiadomości msg.

- void `setPlayerID` (int `id`)
Ustawia ID gracza na podane.
- void `setCurrentPlayerID` (int `id`)
Ustawia ID gracza aktualnie wykonującego ruch na podane.
- void `setGameEngineClient` (`GameEngineClient` *`e`)

Zmienne

- bool `lmb` = false
Wciśnięty lewy przycisk myszy.
- bool `rmb` = false
Wciśnięty prawy przycisk myszy.
- bool `mmb` = false
Wciśnięta rolka.
- bool `moved` = false
Myszka ruszyła się.
- int `lx` = -1
Ostatni x myszy.
- int `ly` = -1
Ostatni y myszy.
- float `mx` = 0
Ostatni ruch w x.
- float `my` = 0
Ostatni ruch w y.
- float `rx` = 0.0f
Aktualny obrót w x.
- float `ry` = 0.0f
Aktualny obrót w y.
- float `rz` = 0.0f
Aktualny obrót w z.
- float `tx` = 0.0f
Aktualne przesunięcie w x.

- float **ty** = 0.0f
Aktualne przesunięcie w y.
- float **scale** = 1.0f
Aktualna skala.
- const float **FRICTION** = 0.1f
Tarcie, zwalnia obrót.
- float **spdx** = 0.0f
Szybkość obrotu w x.
- float **spdy** = 0.0f
Szybkość obrotu w y.
- float **minz**
- float **maxz**
- float **tminz**
- float **tmaxz**
- int **size** = 4
Wielkość pola gry.
- vector< vector< vector< **Cube** > > > **area**
Tablica trójwymiarowa pola gry.
- int **id** = 0
ID gracza.
- int **cid** = 0
ID gracza wykonującego ruch.
- **Cube** * **src** = NULL
Wskaźnik na kostkę (planetę) źródłową.
- **Cube** * **dst** = NULL
Wskaźnik na kostkę (planetę) docelową.
- int **army** = 0
Ilość jednostek do wystania.
- **Text** **info** (0, 8, 8, 0, 0, 0, NULL, "", **SCREENWIDTH**-16, **SCREENHEIGHT**-16)
Górny tekst.
- **Text** **curr** (0, 8, **SCREENHEIGHT**-60, 0, 0, 0, NULL, "", **SCREENWIDTH**-16, 16)

Dolny tekst.

- list< [Text](#) > [msgs](#)

Lista wiadomości.

- float [msgTimer](#) = 0

Odliczanie do zniknięcia kolejnej wiadomości.

- float [rotTimer](#) = 0

Odliczanie do obracania.

- [Sprite](#) * [bg](#)

Wskaźnik na obrazek tła.

- [GameEngineClient](#) * [engine](#)

- [Vertex](#) [tl](#)

- [Vertex](#) [scrtl](#)

10.3.1 Opis szczegółowy

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

Autor

crm

10.3.2 Dokumentacja funkcji

10.3.2.1 void Screen::addMessage (const string & *msg*)

Wypisanie wiadomości *msg*.

Parametry

msg Wiadomość do wypisania

Wiadomości wyskakują od góry, starsze przeskakują w dół. Pierwsza/nowa znika po *MSG_HIDE_DELAY_FIRST* sekundacg, kolejne po *MSG_HIDE_DELAY_NEXT* sekundach. Maksymalna ilość wynosi *MSG_MAX_COUNT*.

10.3.2.2 void Screen::kup (int *key*)

Obsługa puszczenia przycisku na klawiaturze.

Parametry

key Kod puszczonego klawisza

10.3.2.3 void Screen::mousedown (int *x*, int *y*, int *key*)

Obsługa kliknięcia.

Parametry

- x* Współrzędna x myszy
- y* Współrzędna y myszy
- key* Kod wciśniętego klawisza

10.3.2.4 void Screen::mmove (int *x*, int *y*, int *key*)

Obsługa ruchu myszą.

Parametry

- x* Współrzędna x myszy
- y* Współrzędna y myszy
- key* Kod wciśniętego klawisza

10.3.2.5 void Screen::mroll (bool *down*)

Obsługa kliknięcia rolką.

Parametry

- down* Jest *prawdą* jeśli rolka została kliknięta, jeśli została puszczona jest *fałszem*

10.3.2.6 void Screen::mup (int *x*, int *y*, int *key*)

Obsługa puszczenia przycisku myszy.

Parametry

- x* Współrzędna x myszy
- y* Współrzędna y myszy
- key* Kod wciśniętego klawisza

10.3.2.7 void Screen::rotateArb (Vertex & *v*, const Vertex & *s*, const Vertex & *a*, float *ang*)

Obrót dowolnego wektora względem dowolny wektor zaczepionego w dowolnym punkcie o dowolny kąt.

Parametry

v Wektor do obrócenia

s Punkt początkowy

a Oś obrotu

ang kąt

10.3.2.8 void Screen::setCurrentPlayerID (int *id*)

Ustawia ID gracza aktualnie wykonującego ruch na podane.

Zależnie od ID gracza będzie rysowany trójkąt w odpowiednim kolorze

10.3.2.9 void Screen::setPlayerID (int *id*)

Ustawia ID gracza na podane.

Zależnie od ID gracza będzie rysowana ramka innego koloru

10.3.2.10 void Screen::updateArea (vector< pair< Vertex, Planet > > & *items*)

Aktualizacja pola gry.

Wywoływana po otrzymaniu zbiorczych informacji o aktualnym stanie pola gry

10.3.3 Dokumentacja zmiennych**10.3.3.1 int Screen::cid = 0**

ID gracza wykonującego ruch.

Używane do rysowania kolorowego trójkąta

10.3.3.2 Text Screen::curr(0, 8, SCREENHEIGHT-60, 0, 0, 0, NULL,"", SCREENWIDTH-16, 16)

Dolny tekst.

Informacje o planecie znajdującej się pod kursorem

10.3.3.3 int Screen::id = 0

ID gracza.

Używane do rysowania kolorowej ramki wokół poziomu

10.3.3.4 Text Screen::info(0, 8, 8, 0, 0, 0, NULL, "", SCREENWIDTH-16, SCREENHEIGHT-16)

Górny tekst.

Informacje o planecie źródłowej, docelowej i ilości jednostek do wysłania

10.4 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 `addMouseMotionEventHandler (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 `delMouseMotionEventHandler (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)

10.4.1 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.

Autor

crm

10.4.2 Dokumentacja funkcji

10.4.2.1 bool WindowEngine::addKeyDownEventHandler (void(*) (int) handle)

Rejestracja funkcji wywoływanej po wciśnięciu klawisza na klawiaturze.

Parametry

handle Wskaźnik na funkcję. Argument to kod klawisza

10.4.2.2 bool WindowEngine::addKeyPressedEventHandler (void(*) (int) *handle*)

Rejestracja funkcji wywoływanej po przytrzymaniu klawisza na klawiaturze.

Parametry

handle Wskaźnik na funkcję. Argument to kod klawisza

10.4.2.3 bool WindowEngine::addKeyUpEventHandler (void(*) (int) *handle*)

Rejestracja funkcji wywoływanej po puszczeniu klawisza na klawiaturze.

Parametry

handle Wskaźnik na funkcję. Argument to kod klawisza

10.4.2.4 bool WindowEngine::addMouseDownEventHandler (void(*) (int, int, int) *handle*)

Rejestracja funkcji wywoływanej po wcisnięciu przycisku myszy.

Parametry

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

10.4.2.5 bool WindowEngine::addMouseMotionEventHandler (void(*) (int, int, int) *handle*)

Rejestracja funkcji wywoływanej po ruszeniu myszy.

Parametry

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

10.4.2.6 bool WindowEngine::addMouseUpEventHandler (void(*) (int, int, int) *handle*)

Rejestracja funkcji wywoływanej po puszczeniu przycisku myszy.

Parametry

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

10.4.2.7 `bool WindowEngine::init (RenderType rt = SDL, WaitType wt = DELAY)`

Inicjalizacja ekranu.

Parametry

- [in] *rt* Używana biblioteka graficzna. Nie ma nic poza *SDL*em
- [in] *wt* Sposób reagowania na koniec danej klatki.

Rozdział 11

Dokumentacja klas

11.1 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.

11.1.1 Opis szczegółowy

Informacje o animacji.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- `include/sprite.h`

11.2 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**

11.2.1 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

11.3 Dokumentacja klasy Client

Połączenie z serwerem.

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

Metody publiczne

- void `close` ()
metoda zamykająca połączenie
- void `send` (const std::string &m)
metoda wysyłająca wiadomość do serwera
- `~Client` ()
destruktor
- std::string `receive` ()
metoda zwracająca wiadomość od serwera
- void `write` (const `Message` &msg)
metoda wysyłająca wiadomość

Statyczne metody publiczne

- static `Client` * `create` (const std::string host, const std::string port)
Nazwany konstruktor.

11.3.1 Opis szczegółowy

Połączenie z serwerem. Klasa odpowiedzialna za obsługę połączenia z serwerem

Autor

Paweł Ściegienny

11.3.2 Dokumentacja funkcji składowych

11.3.2.1 void Client::close ()

metoda zamykająca połączenie

metoda binduje handler do_close z metodą post socketu

11.3.2.2 Client * Client::create (const std::string *host*, const std::string *port*) [static]

Nazwany konstruktor.

Jedyny legalny sposób tworzenia instancji klienckich

Parametry

- [in] *host* hostname
- [in] *port* numer portu

11.3.2.3 void Client::send (const std::string & *m*)

metoda wysyłająca wiadomość do serwera

metoda konwertuje stringa do [Message](#), a następnie wysyła do serwera

Parametry

- [in] *m* referencja do stringa który ma zostać wysłany

11.3.2.4 void Client::write (const Message & *msg*)

metoda wysyłająca wiadomość

metoda bindująca handler do_writer z metodą post socketu

Dokumentacja dla tej klasy została wygenerowana z plików:

- include/Client.hpp
- src/Client.cpp

11.4 Dokumentacja struktury Cube

Metody publiczne

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

Atrybuty publiczne

- int **x**
- int **y**
- int **z**
- unsigned int **col**
- int **army**
- float **pct**
- float **roll**
- [Vertex](#) **verts** [VERT_COUNT]

Statyczne atrybuty publiczne

- static const int **VERT_COUNT** = 24

Dokumentacja dla tej struktury została wygenerowana z pliku:

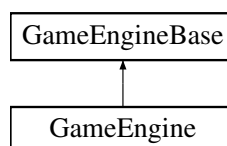
- src/screen.cpp

11.5 Dokumentacja klasy GameEngine

Główny silnik gry.

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

Diagram dziedziczenia dla GameEngine



Metody publiczne

- [GameEngine](#) (uint16 size, uint16 players)
Tworzy plansze.
- [uint16 EndTurn](#) ()
Kończy turę.
- void [RemovePlayer](#) (uint16 player)
Usuwa gracza.
- [RETURNS::MOVE Move](#) (const [Vertex](#) &src, const [Vertex](#) &dst, uint16 num)

Przenosi jednostki z jednej planety na drugą.

- `uint16 AddPlayer (uint16 socket_id)`

Dodaje nowego gracza do bitwy.

- `bool CanDoAction (uint16 socket_id)`

Sprawdza czy gracz może wykonać jakąkolwiek operację.

- `bool IsEndGame () const`

Sprawdza czy to już koniec gry.

11.5.1 Opis szczegółowy

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

Autor

Marcin TORGiren Fabrykowski

11.5.2 Dokumentacja konstruktora i destruktora

11.5.2.1 `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

[in] *size* Rozmiar planszy.

[in] *players* Liczba graczy biorących udział w rozgrywce

11.5.3 Dokumentacja funkcji składowych

11.5.3.1 `uint16 GameEngine::AddPlayer (uint16 socket_id)`

Dodaje nowego gracza do bitwy.

Dodaje nowego gracza do bitwy i przyporządkowuje mu id socketa na którym ten klient nadaje

Parametry

socket_id Id socketa na którym nadaje gracz

Zwraca

Zwraca Numer Gracza jaki dostał nowy gracz

11.5.3.2 bool GameEngine::CanDoAction (uint16 socket_id)

Sprawdza czy gracz może wykonać jakąkolwiek operację.

Sprawdza czy numer gracza nadającego z socketa o zadanym id, może wykonywać ruch w tej turze.

Parametry

socket_id Id socketa z którego przyszło żądanie akcji

Zwraca

TRUE jeśli to tura tego gracza, FALSE w przeciwnym wypadku

11.5.3.3 uint16 GameEngine::EndTurn ()

Kończy turę.

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

Zwraca

Zwraca numer następnego gracza.

11.5.3.4 bool GameEngine::IsEndGame () const

Sprawdza czy to już koniec gry.

Sprawdza czy ustawiona jest już flaga zakończenia gry

Zwraca

TRUE jeśli to już koniec gry, FALSE w przeciwnym wypadku

11.5.3.5 RETURNS::MOVE GameEngine::Move (const Vertex & src, const Vertex & dst, uint16 num)

Przenosi jednostki z jednej planety na drugą.

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

Parametry

- [in] *src* Współrzędne planety źródłowej
- [in] *dst* Współrzędne planety docelowej
- [in] *num* Liczba jednostek do przeniesienia

Zwraca

Zwraca ERRORS::MOVE

11.5.3.6 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 czy odłączeniu się gracza jak również czy pokonaniu danego gracza

Parametry

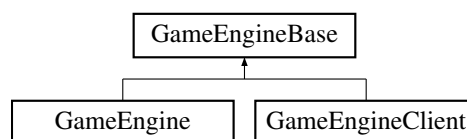
- [in] *player* Numer gracza który ma zostać usunięty

Dokumentacja dla tej klasy została wygenerowana z plików:

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

11.6 Dokumentacja klasy GameEngineBase

Diagram dziedziczenia dla GameEngineBase

**Metody publiczne**

- [GameEngineBase \(uint16 size\)](#)
Konstruktor tworzący pole bitwy.
- [uint16 ActPlayer \(\)](#) const

Aktualny gracz.

- **Planet & GetPlanet** (const **Vertex** &src) const
Zwraca planetę o zadanym położeniu.
- **uint16 GetSize** () const
Zwraca rozmiar pola bitwy.

Atrybuty chronione

- std::set< **uint16** > **itsPlayers**
- std::set< **uint16** >::iterator **itsActPlayer**
- **Planet *** itsPlanety**
- **uint16 itsSize**

11.6.1 Dokumentacja konstruktora i destruktor

11.6.1.1 GameEngineBase::GameEngineBase (**uint16 size**)

Konstruktor tworzący pole bitwy.

Konstruktor klasy bazowej dla Silnika gry i silnika klienta. TWorzy on pole bitwy o zadanym rozmiarze. Pole ma postać sześciiany o rozmiarze size

Parametry

size Rozmiar boku sześcianu pola bitwy liczony w ilości planet

11.6.2 Dokumentacja funkcji składowych

11.6.2.1 **uint16 GameEngineBase::ActPlayer ()** const

Aktualny gracz.

Zwraca numer aktualnego gracza.

Zwraca

Numer aktualnego gracza.

11.6.2.2 **Planet & GameEngineBase::GetPlanet (const **Vertex** & *src*)** const

Zwraca planetę o zadanym położeniu.

Zwraca referencję do planety znajdującej się w położeniu Vertexu podanego jako argument

Parametry

src [Vertex](#) wskazujący na położenie planety która ma być zwrócona

Zwraca

Referencja do planety z zadanego położenia

11.6.2.3 uint16 GameEngineBase::GetSize () const

Zwraca rozmiar pola bitwy.

Zwraca rozmiar pola bitwy

Zwraca

Rozmair pola bitwy

Dokumentacja dla tej klasy została wygenerowana z plików:

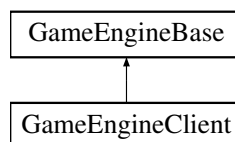
- include/gameenginebase.h
- src/gameenginebase.cpp

11.7 Dokumentacja klasy GameEngineClient

Klasa silnika gry dla klienta.

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

Diagram dziedziczenia dla GameEngineClient

**Metody publiczne**

- void [MainLoop](#) ()
Główna pętla gry.
- void [PlanetUpdate](#) (const [Vertex](#) &dst, const [Planet](#) &planet)
Uaktualnia dane o planecie.
- void [EndGame](#) ()
Ustawia flagę końca gry.

- void `SendMove` (`Vertex` src, `Vertex` dst, `uint16` num)

Wysyła żądanie przesunięcia jednostek.

- void `SendEndTurn` ()

Wysyła żądanie końca tury.

Statyczne metody publiczne

- static `GameEngineClient` * `Create` (std::string ip)

Tworzy instancje klienckiego silnika gry.

11.7.1 Opis szczegółowy

Klasa silnika gry dla klienta. Klasa zajmująca się obsługą zachowań gracza po stronie klienta. Rozbudować system przeliczania rozgrywki, aby odciążyć łącze

Autor

Marcin TORGiren Fabrykowski

11.7.2 Dokumentacja funkcji składowych

11.7.2.1 static GameEngineClient* GameEngineClient::Create (std::string ip) [inline, static]

Tworzy instancje klienckiego silnika gry.

Statyczna funkcja, przyjmująca adres serwera do którego będzie się łączył kliencki silnik gry. Tworzy ona połączenie, pobiera stan rozgrywki (rozmiar planszy, swój numer gracza, parametry planet), a następnie na podstawie tych danych tworzy instancje klienckiego silnika gry

Parametry

ip Łańcuch znaków zawierający adres ip serwera gry

Zwraca

Wskaźnik na instancję klasy klienckiego silnika gry

11.7.2.2 void GameEngineClient::EndGame ()

Ustawia flagę końca gry.

Ustawia flagę zakończonej gry

11.7.2.3 void GameEngineClient::MainLoop ()

Główna pętla gry.

Główna pętla gry, wykonująca się do czasu otrzymania sygnału o zakończeniu rozgrywki. Zajmuje się ona odbieraniem komunikatów od serwera i odpowiedniego reagowania na nie

11.7.2.4 void GameEngineClient::PlanetUpdate (const Vertex & *dst*, const Planet & *planet*)

Uaktualnia dane o planecie.

Ustawia nowe parametry planety znajdującej się pod wskazaniem Vertexa na parametry takie jak zadanej planety

Parametry

dst Wskazanie planety która będzie aktualizowana

planet Planeta wzorcowa - po aktualizacji planeta znajdująca się pod *dst* będzie taka sama jak zadana w parametrze

11.7.2.5 void GameEngineClient::SendEndTurn ()

Wysyła żądanie końca tury.

Wysyła do serwera sygnalizację zakończenia tury przez danego gracza

11.7.2.6 void GameEngineClient::SendMove (Vertex *src*, Vertex *dst*, uint16 *num*)

Wysyła żądanie przesunięcia jednostek.

Wysyła do serwera żądanie gracza o przeniesienie jednostek z planety pod Vertexem *src* do planety pod Vertexem *dst* w liczbie *num*. W przypadku planet należących do różnych graczy nastąpi walka o tą planetę.

Parametry

src Vertex planety źródłowej

dst Vertex planety docelowej

num Liczba jednostek do przeniesienia

Dokumentacja dla tej klasy została wygenerowana z plików:

- include/gameengineclient.h
- src/gameengineclient.cpp

11.8 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)
Konstruktor.
- void [operator=](#) (const [Message](#) &src)
operator przypisania
- 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 nagłówek
- void `encode_header ()`
metoda zapisująca nagłówek
- void `source (unsigned src)`
metoda dopisująca do wiadomości id klienta
- unsigned `source () const`
metoda zwracająca id klienta z wiadomości
- std::string `getString ()`
metoda konwertująca wiadomość do stringa [depracated]

11.8.1 Opis szczegółowy

Przesyłana wiadomość. Klasa odpowiedzialna za poprawne informacje o wiadomości

Autor

Paweł Ściegienny

11.8.2 Dokumentacja konstruktora i destruktora

11.8.2.1 `Message::Message ()`

Konstruktor.

Konstruktor domyślny - inicjalizuje długość wiadomości

11.8.2.2 `Message::Message (const Message & src)`

Konstruktor.

Konstruktor kopiujący - w pewnym momencie bardzo istotny element programu

11.8.3 Dokumentacja funkcji składowych

11.8.3.1 `size_t Message::length () const [inline]`

metoda zwracająca długość wiadomości

metoda zwracająca długość wiadomości WRAZ z długością nagłówka

11.8.3.2 void Message::operator= (const Message & src)

operator przypisania

operator przypisania [depracated]

11.8.3.3 void Message::source (unsigned src) [inline]

metoda dopisująca do wiadomości id klienta

Parametry

[in] *src* id klienta

Dokumentacja dla tej klasy została wygenerowana z plików:

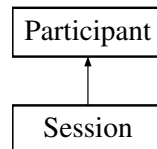
- include/Message.hpp
- src/Message.cpp

11.9 Dokumentacja klasy Participant

Interfejs pokoju.

```
#include <Participant.hpp>
```

Diagram dziedziczenia dla Participant



Metody publiczne

- virtual [~Participant](#) ()
wirtualny destruktor
- virtual void [deliver](#) (const [Message](#) &msg)=0
metoda czysto wirtualna

11.9.1 Opis szczegółowy

Interfejs pokoju. Klasa abstrakcyjna reprezentująca połączenie socketów od klientów

Autor

Paweł Ściegienny

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/Participant.hpp

11.10 Dokumentacja klasy Planet

Klasa planety.

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

Metody publiczne

- [Planet \(\)](#)
Tworzy planetę.
- [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 ilość jednostek na planecie.
- [FightResult Atak \(uint16 ile, uint16 kogo\)](#)
Przeprowadza atak na planetę Przeprowadza atak zadanej ilości jednostek na planetę.
- void [SetPlayer \(uint16 gracz\)](#)
Ustawia nowego właściciela planety.
- RETURNS::ENDTURN [EndTurn \(\)](#)
Kończy turę na danej planecie.
- RETURNS::MOVE [Zabierz \(uint16 ile\)](#)
Zabiera z planety zadaną liczbę jednostek.
- void [Dodaj \(uint16 ile\)](#)
Dodaje jednostki do planety.

- `std::string ToString ()`
- `operator std::string ()`

Statyczne metody publiczne

- `static Planet ToPlanet (std::string str)`

11.10.1 Opis szczegółowy

Klasa planety. Opisuje właściwości planety - elementarnej jednostki przestrzeni

Autor

Marcin TORGiren Fabrykowski

11.10.2 Dokumentacja konstruktora i destruktor

11.10.2.1 Planet::Planet ()

Tworzy planetę.

Konstruktor. Tworzy neutralną planetę z losową (od 0 do 9) liczbą jednostek

11.10.3 Dokumentacja funkcji składowych

11.10.3.1 FightResult Planet::Atak (uint16 ile, uint16 kogo)

Przeprowadza atak na planetę. Przeprowadza atak zadanej ilości jednostek na planetę.

Parametry

ile Liczba jednostek wroga, biorąca udział w ataku

kogo 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

11.10.3.2 void Planet::Dodaj (uint16 ile)

Dodaje jednostki do planety.

Zwiększa liczbę jednostek na planecie o zadaną ilość

Parametry

ile Liczba jednostek które zostaną dodane do garnizonu planety

11.10.3.3 RETURNS::ENDTURN Planet::EndTurn ()

Kończy turę na danej planecie.

W przypadku okupowania planety następuje zdobywanie/zdejmowanie flagi.

W przypadku posiadanych planet, następuje tworzenie nowych jednostek

11.10.3.4 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

11.10.3.5 uint16 Planet::RetJednostki () const

Zwraca ilość 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

Zwraca

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

11.10.3.6 uint16 Planet::RetOkupant () const

Zwraca numer gracza-okupanta planety.

Zwraca numer gracza który jest aktualnie okupantem planety

Zwraca

Numer gracza który okupuje planetę, bądź NULL jeśli takowego nie ma

11.10.3.7 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

11.10.3.8 void Planet::SetPlayer (uint16 gracz)

Ustawia nowego właściciela planety.

Metoda która ustawia nowego właściciela planety

Parametry

gracz Numer gracza będącego nowym właścicielem

11.10.3.9 RETURNS::MOVE Planet::Zabierz (uint16 ile)

Zabiera z planety zadaną liczbę jednostek.

Zmniejsza liczbę jednostek na danej planecie o zadaną zawartość. Sprawdza tylko czy zadana wartość jest mniejsza bądź równa ilości jednostek na planecie

Parametry

ile Zadana ilość jednostek do zabrania

Zwraca

Zwraca status operacji

Dokumentacja dla tej klasy została wygenerowana z plików:

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

11.11 Dokumentacja klasy Point

Klasa położenia w przestrzeni.

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

Atrybuty publiczne

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

11.11.1 Opis szczegółowy

Klasa położenia w przestrzeni. Obrazuje położenie punktu w przestrzeni planszy

Autor

Marcin TORGiren Fabrykowski

11.11.2 Dokumentacja atrybutów składowych

11.11.2.1 `uint16 Point::itsX`

Położenie na osi X

11.11.2.2 `uint16 Point::itsY`

Położenie na osi Y

11.11.2.3 `uint16 Point::itsZ`

Położenie na osi Z

Dokumentacja dla tej klasy została wygenerowana z pliku:

- `include/point.h`

11.12 Dokumentacja klasy Room

miejsce gdzie zbiegają się sockety.

```
#include <Room.hpp>
```

Metody publiczne

- void [join](#) (Participant_ptr participant)
metoda dodająca uczestnika
- void [leave](#) (Participant_ptr participant)
metoda usuwająca uczestnika

- void `deliver` (const `Message` &msg)
dostarczenie wiadomości do wszystkich klientów
- void `deliver` (unsigned who, const `Message` &msg)
metoda dostarczająca wiadomość do konkretnego klienta
- unsigned `search` (`Participant` *participant)
metoda pozwalająca zidentyfikować uczestnika na podstawie socketu
- `Participant` * `search` (unsigned ident)
metoda pozwalająca znaleźć socket na podstawie ID klienta
- `Message` `todo` ()
odczyt wiadomości
- void `todo` (const `Message` msg)
dodanie wiadomości do bufora

11.12.1 Opis szczegółowy

miejsce gdzie zbiegają się sockety. implementacja interfejsu `Participant`. Konkretnie rozwiązania

Autor

Paweł Ściegienny

11.12.2 Dokumentacja funkcji składowych

11.12.2.1 `Participant` * `Room::search` (unsigned *ident*)

metoda pozwalająca znaleźć socket na podstawie ID klienta

jak już coś to zwróćmy pierwszego

Dokumentacja dla tej klasy została wygenerowana z plików:

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

11.13 Dokumentacja klasy Server

Metody publiczne

- `~Server` ()

Destruktor.

- void `send` (const std::string &m)
Metoda wysyłająca wiadomość do wszystkich.
- void `send` (unsigned who, std::string m)
Metoda wysyłająca wiadomość do konkretnego użytkownika.
- `Message receive` ()
Metoda odbierająca.

Statyczne metody publiczne

- static `Server * create` (std::string port)
Nazwany konstruktor.

11.13.1 Dokumentacja konstruktora i destruktora

11.13.1.1 `Server::~Server () [inline]`

Destruktor.

Zamyka połączenie, sprząta

11.13.2 Dokumentacja funkcji składowych

11.13.2.1 `Message Server::receive () [inline]`

Metoda odbierająca.

Metoda pozwalająca sprawdzić co serwer ma do roboty

Zwraca

`Message` wiadomość zawierającą id i treść

11.13.2.2 `void Server::send (unsigned who, std::string m) [inline]`

Metoda wysyłająca wiadomość do konkretnego użytkownika.

metoda pozwalająca na wysłanie wiadomości do konkretnego użytkownika

Parametry

[in] *who* id użytkownika

[in] *m* string z treścią wiadomości

11.13.2.3 void Server::send (const std::string & m) [inline]

Metoda wysyłająca wiadomość do wszystkich.

metoda pozwalająca rozesłać do wszystkich użytkowników wiadomość

Parametry

[in] *m* string z treścią wiadomości

Dokumentacja dla tej klasy została wygenerowana z plików:

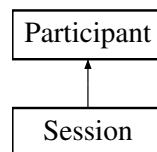
- include/Server.hpp
- src/Server.cpp

11.14 Dokumentacja klasy Session

reprezentacja sesji

```
#include <Session.hpp>
```

Diagram dziedziczenia dla Session



Metody publiczne

- `Session` (boost::asio::io_service &io_service, `Room` &room)
konstruktor tworzący z listą inicjalizacyjną
- `tcp::socket & socket ()`
jaki to socket
- `void start ()`
starter
- `void deliver` (const `Message` &msg)
dostarczac wiadomosci
- `void handle_read_header` (const boost::system::error_code &error)
odczyt nagłówka
- `void handle_read_body` (const boost::system::error_code &error)

odczyt ciała wiadomości

- void `handle_write` (const boost::system::error_code &error)
handler wysyłający

11.14.1 Opis szczegółowy

reprezentacja sesji klasa utrzymująca szerokorozumianą sesję

Autor

Paweł Ściegienny

Dokumentacja dla tej klasy została wygenerowana z plików:

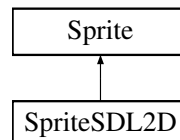
- include/Session.hpp
- src/Session.cpp

11.15 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)
Animuje animację anim z prędkością spd. Do frame wpisuje nową klatkę animacji.
- 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
*Wyświetla **Sprite** z animacją anim i klatką frame na współrzędnych x, y, z.*
- virtual void **flush** ()=0
Wyrzucenie bufora.

Statyczne metody publiczne

- static void **print** ()
*Wywołuje flush na wszystkich wczytanych **Sprite**.*
- static void **clear** ()
Kasuje wszystkie sprite.
- static void **reload** ()
*Ponowne wczytanie **Sprite**.*
- static **Sprite** * **load** (const std::string &name, bool force=false)
Wczytuje grafikę o podanej nazwie.

Metody chronione

- void **addSpritePtr** (**SpritePtr** *s)
Dodaje wskaźnik na smart pointera do listy.
- void **delSpritePtr** (**SpritePtr** *s)
Kasuje wskaźnik na smart pointera z listy.
- void **setSpritePtrs** (**Sprite** *s)
*Przestawia smart pointery z danego **Sprite** na inny.*
- virtual bool **loadGfx** (const std::string &name)=0
Wczytywanie grafiki.

- virtual bool `loadMask` (void *pixs, int `w`, int h, int bpp)

Generowanie maski kolizji.

- virtual bool `loadAnims` (const std::string &`name`)

Wczytywanie animacji.

Atrybuty chronione

- std::set< `SpritePtr` * > `spritePtrs`

Lista smart pointerów.

- std::string `name`

Nazwa `Sprite`.

- int `w`

Wymiary.

- int `h`

- bool * `mask`

Maska kolizji.

- std::vector< `Anim` > `anims`

Animacje.

- std::map< std::string, `Anim` * > `animNames`

Nazwy animacji.

Statyczne atrybuty chronione

- static std::map< std::string, `Sprite` * > `sprites`

Statyczna lista wszystkich wczytanych `Sprite`ów.

11.15.1 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.

Autor

crm

11.15.2 Dokumentacja funkcji składowych

11.15.2.1 void Sprite::animate (int *anim*, float & *frame*, float *spd* = -1.0f)
[virtual]

Animuje animację *anim* z prędkością *spd*. Do *frame* wpisuje nową klatkę animacji.

Jeśli *spd* jest mniejsze od 0 to używa standardowej szybkości animacji

Parametry

[in] *anim* Animacja

[in, out] *frame* Klatka początkowa, zmieniane na kolejną

[in] *spd* Szybkość animacji

11.15.2.2 virtual void Sprite::flush () [pure virtual]

Wyrzucenie bufora.

Docelowo przeznaczone do OpenGL'a i tablicy wierzchołków. Tutaj nieużywane.

Implementowany w [SpriteSDL2D](#).

11.15.2.3 virtual bool Sprite::loadGfx (const std::string & *name*)
[protected, pure virtual]

Wczytywanie grafiki.

Do zdefiniowania w klasach poniżej

Parametry

name Nazwa

11.15.2.4 bool Sprite::loadMask (void * *pixs*, int *w*, int *h*, int *bpp*)
[protected, virtual]

Generowanie maski kolizji.

Tutaj wyłączone celem zaoszczędzenia pamięci

Parametry

pixs Piksele

w Szerokość

h Wysokość

bpp Głębina koloru

11.15.2.5 `virtual void Sprite::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) [pure virtual]`

Wyświetla [Sprite](#) z animacją *anim* i klatką *frame* na współrzędnych x, y, z.

Pozostałe parametry są opcjonalne i - obecnie - nieużywane.

Do zrobienia

Alfa całego obrazka

Parallax scrolling

Barwienie obrazka

Parametry

[in] *x* Współrzędna x
[in] *y* Współrzędna y
[in] *z* Współrzędna z
[in] *anim* Numer animacji
[in] *frame* Klatka animacji
[in] *alpha* Przeźroczystość, 0-255
[in] *px* Parallax scrolling, poziomy
[in] *py* Parallax scrolling, pionowy
[in] *r* Czerwony
[in] *g* Zielony
[in] *b* Niebieski

Implementowany w [SpriteSDL2D](#).

11.15.2.6 `void Sprite::reload () [static]`

Ponowne wczytanie [Sprite](#).

Używać po przestawieniu trybu wyświetlania

11.15.2.7 `void Sprite::setSpritePtrs (Sprite * s) [protected]`

Przestawia smart pointery z danego [Sprite](#) na inny.

Parametry

s Nowy sprite

Dokumentacja dla tej klasy została wygenerowana z plików:

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

11.16 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`

11.16.1 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:

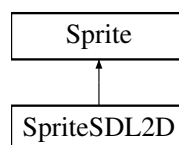
- include/sprite.h
- src/sprite.cpp

11.17 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.

11.17.1 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.

Autor

crm

11.17.2 Dokumentacja funkcji składowych

11.17.2.1 void SpriteSDL2D::flush () [inline, virtual]

Dobre pytanie. Sam nie wiem.

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

Implementuje **Sprite**.

11.17.2.2 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] **z** Współrzędna z
- [in] **anim** Numer animacji
- [in] **frame** Klatka animacji
- [in] **alpha** Przeźroczystość, 0-255
- [in] **px** Parallax scrolling, poziomy
- [in] **py** Parallax scrolling, pionowy
- [in] **r** Czerwony

[in] **g** Zielony
[in] **b** Niebieski

Implementuje [Sprite](#).

Dokumentacja dla tej klasy została wygenerowana z plików:

- include/sprite_sdl_2d.h
- src/sprite_sdl_2d.cpp

11.18 Dokumentacja klasy Text

Klasa wyświetlająca tekst.

```
#include <text.h>
```

Typy publiczne

- enum [Align](#) { **LEFT**, **CENTER**, **RIGHT** }

Metody publiczne

- **Text** (unsigned int id, float x, float y, float z, float px, float py, [Sprite](#) *sSprite, const char *sText, unsigned int w, unsigned int h, int nlSize=16, int spSize=12, int tabSize=32)
- **Text** (const [Text](#) &txt)
- [Text](#) & **operator=** (const char *str)
Przypisanie tekstu str.
- [Text](#) & **operator=** (string str)
Przypisanie tekstu str.
- [Text](#) & **operator+=** (const char *str)
Dopisanie tekstu str.
- [Text](#) & **operator+=** (string str)
Dopisanie tekstu str.
- void **setPos** (float sx, float sy, float sz=0)
Ustawienie nowej pozycji.
- void **setX** (float sx)
Ustawienie nowej pozycji.
- void **setY** (float sy)

Ustawienie nowej pozycji.

- void **setZ** (float sz)
Ustawienie nowej pozycji.
- void **setPara** (float spx, float spy)
Ustawienie parametrów parallax scrollingu.
- void **setAlpha** (unsigned char sa)
Ustawienie przezroczystości tekstu.
- void **setFont** (Sprite *sSprite)
Ustawienie nowej czcionki.
- void **setSprite** (Sprite *sSprite)
Ustawienie nowej czcionki.
- void **setW** (unsigned int sw)
Ustawienie maksymalnej szerokości tekstu.
- void **setH** (unsigned int sh)
Ustawienie wysokości tekstu. Nie używane do niczego.
- void **setDim** (unsigned int sw, unsigned int sh)
Ustawienie wymiarów tekstu.
- void **setStr** (const char *sStr)
Przypisanie tekstu sStr.
- void **addStr** (const char *sStr)
Dopisanie tekstu sStr.
- void **setAlign** (Align sa)
Ustawienie wyrównania tekstu.
- void **setAlignLeft** ()
Ustawienie wyrównania tekstu do lewej.
- void **setAlignCenter** ()
Ustawienie wyrównania tekstu do środka.
- void **setAlignRight** ()
Ustawienie wyrównania tekstu do prawej.
- void **getPos** (float &gx, float &gy, float &gz) const
- float **getX** () const

- float **getY** () const
- float **getZ** () const
- void **getPara** (float &gpx, float &gpy) const
- const **Sprite** * **getSprite** () const
- unsigned int **getW** () const
- unsigned int **getH** () const
- void **getDim** (unsigned int &gw, unsigned int &gh) const
- const char * **getText** () const
- const char * **getStr** () const
- int **getAlign** () const
- int **getNlSize** () const

Zwraca wielkość nowej linii (wysokość linii tekstu).

- int **getSpSize** () const

Zwraca wielkość spacji (ilość pikseli odstępu między znakami).

- int **getTabSize** () const

Zwraca wielkość tabulatora.

- void **update** ()

Aktualizacja tekstu.

- void **print** ()

Wypisanie tekstu.

- int **getWordLen** (const char *str)

Zwraca długość podanego tekstu (do białego znaku) używając aktualnej czcionki.

- int **getLineLen** (const char *str)

Zwraca długość podanej linii tekstu używając aktualnej czcionki.

11.18.1 Opis szczegółowy

Klasa wyświetlająca tekst.

Autor

crm

Obsługuje:

- Wyrównywanie tekstu do lewej, prawej i środka
- Zawijanie
- Różne czcionki
- Dowolną długość spacji, tabulatora i wysokość linii

11.18.2 Dokumentacja składowych wyliczanych

11.18.2.1 `enum Text::Align`

LEFT - Wyrównanie do lewej *CENTER* - Centrowanie tekstu *RIGHT* - Wyrównanie do prawej

11.18.3 Dokumentacja funkcji składowych

11.18.3.1 `void Text::addStr (const char * sStr)`

Dopisanie tekstu *sStr*.

Parametry

sStr Tekst

11.18.3.2 `int Text::getLineLen (const char * str)`

Zwraca długość podanej linii tekstu używając aktualnej czcionki.

Parametry

str Tekst

11.18.3.3 `int Text::getWordLen (const char * str)`

Zwraca długość podanego tekstu (do białego znaku) używając aktualnej czcionki.

Parametry

str Tekst

11.18.3.4 `Text& Text::operator+= (string str) [inline]`

Dopisanie tekstu *str*.

Parametry

str Tekst

11.18.3.5 Text& Text::operator+=(const char * *str*) [inline]

Dopisanie tekstu *str*.

Parametry

str Tekst

11.18.3.6 Text& Text::operator=(string *str*) [inline]

Przypisanie tekstu *str*.

Parametry

str Tekst

11.18.3.7 Text& Text::operator=(const char * *str*) [inline]

Przypisanie tekstu *str*.

Parametry

str Tekst

11.18.3.8 void Text::setAlign (Align *sa*) [inline]

Ustawienie wyrównania tekstu.

Parametry

str Typ wyrównania

11.18.3.9 void Text::setDim (unsigned int *sw*, unsigned int *sh*) [inline]

Ustawienie wymiarów tekstu.

Parametry

sw Szerokość

sh Wysokość

11.18.3.10 void Text::setStr (const char * sStr)

Przypisanie tekstu *sStr*.

Parametry

sStr Tekst

11.18.3.11 void Text::update ()

Aktualizacja tekstu.

Tutaj nieużywane.

Dokumentacja dla tej klasy została wygenerowana z plików:

- include/text.h
- src/text.cpp

11.19 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

Porównanie dwóch wektorów z pominięciem współrzędnej z.

- **Vertex operator+** (const **Vertex** &v) const
- **Vertex operator-** (const **Vertex** &v) const
- **Vertex operator*** (float v) const
- **Vertex operator/** (float v) const
- **operator std::string** ()
- **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`

11.19.1 Opis szczegółowy

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

Autor

crm

11.19.2 Dokumentacja funkcji składowych

11.19.2.1 `bool Vertex::eq2d (const Vertex & v) const [inline]`

Porównanie dwóch wektorów z pominięciem współrzędnej `z`.

Współrzędne są rzutowane na liczbę całkowitą

Dokumentacja dla tej struktury została wygenerowana z pliku:

- `include/vertex.h`

Rozdział 12

Dokumentacja plików

12.1 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`
- typedef `uint16` `RETURNS::ENDTURN`

Wyliczenia

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

Zmienne

- const int `SCREENWIDTH` = 800
 Szerokość okienka.

- const int **SCREENHEIGHT** = 600
Wysokość okienka.
- const int **BPP** = 32
Głębina koloru.
- const char **GAMENAME** [] = "RTTT - Risky Tic Tak Toe - Bitwa o Alfa Centauri 4000AD"
Tytuł okienka z grą.
- const int **FPSDELAY** = 1000/50
Przerwa między klatkami (tylko dla DELAY).
- const float **DEGTORAD** = 3.141592653589793f/180.0f
Zmienna zamieniająca stopnie na radiany.
- const float **RADTODEG** = 180.0f/3.141592653589793f
Zmienna zamieniająca radiany na stopnie.
- const char **IMGEXT** [] = ".png"
Rozszerzenie obrazka.
- const char **ANIMEXT** [] = ".txt"
Rozszerzenie pliku z animacjami.
- const char **FONT** [] = "data/font_00"
Ścieżka do pliku z czcionką.
- const char **BACKGROUND** [] = "data/bg_01"
Ścieżka do pliku z tłem.
- const float **MSG_HIDE_DELAY_FIRST** = 5.0f
Czas do schowania pierwszej wiadomości, w sekundach.
- const float **MSG_HIDE_DELAY_NEXT** = 0.5f
Czas do schowania kolejnych wiadomości, w sekundach.
- const unsigned int **MSG_MAX_COUNT** = 8
Maksymalna ilość wiadomości.
- const unsigned int **PLAYER_COLORS** []
Kolory graczy 1-8.
- const unsigned int **PLANET_SRC_COLOR** = 0x0058AF58
Kolor wybranej planety źródłowej.

- const unsigned int `PLANET_DST_COLOR` = 0x00C04B4B

Kolor wybranej planety docelowej.

- const int `OCCUPY_MAX` = 5
- const uint16 `RETURNS::NOTHING` = 1
- const uint16 `RETURNS::NEW_UNIT` = 2
- const uint16 `RETURNS::FLAG_DOWN` = 4
- const uint16 `RETURNS::FLAG_UP` = 8
- const uint16 `RETURNS::PLAYER_OUT` = 16
- const uint16 `RETURNS::PLAYER_IN` = 32
- const uint16 `RETURNS::FLAG_ERROR` = 64

12.1.1 Opis szczegółowy

12.1.2 Dokumentacja definicji typów

12.1.2.1 typedef std::vector<FightResultRow> FightResult

Wektor wierszy logów z walki

12.1.2.2 typedef std::pair<std::vector<uint16>,std::vector<uint16> > FightResultRow

Struktura wiersza logów z walki

12.1.2.3 typedef uint32_t uint

Liczba całkowita o rozmiarze 32bitów

12.1.2.4 typedef uint16_t uint16

Liczba całkowita o rozmiarze 16 bitów

12.1.3 Dokumentacja zmiennych

12.1.3.1 const int OCCUPY_MAX = 5

Maksymalny poziom okupowanej planety powodujący jej przejęcie

12.1.3.2 const unsigned int PLAYER_COLORS[]

Wartość początkowa:

```
{  
0x00C00000,  
0x00FEA100,  
0x00FBFE00,  
0x003FDE00,  
0x0017EECD,  
0x00228FFF,  
0x005E1FFF,  
0x00CF13EB  
}
```

Kolory graczy 1-8.

- 0x00C00000 - Czerwony
- 0x00FEA100 - Pomarańczowy
- 0x00FBFE00 - Żółty
- 0x003FDE00 - Zielony
- 0x0017EECD - Cyan
- 0x00228FFF - Niebieski
- 0x005E1FFF - Fioletowy
- 0x00CF13EB - Różowy

Skorowidz

- ~Server
 - Server, [56](#)
- ActPlayer
 - GameEngineBase, [43](#)
- addKeyDownEventHandler
 - WindowEngine, [32](#)
- addKeyPressedEventHandler
 - WindowEngine, [32](#)
- addKeyUpEventHandler
 - WindowEngine, [33](#)
- addMessage
 - Screen, [27](#)
- addMouseDownEventHandler
 - WindowEngine, [33](#)
- addMouseMotionEventHandler
 - WindowEngine, [33](#)
- addMouseUpEventHandler
 - WindowEngine, [33](#)
- AddPlayer
 - GameEngine, [40](#)
- addStr
 - Text, [68](#)
- Align
 - Text, [68](#)
- animate
 - Sprite, [61](#)
- Atak
 - Planet, [51](#)
- CanDoAction
 - GameEngine, [41](#)
- cid
 - Screen, [29](#)
- clearZBuff
 - Drawing, [21](#)
- Client, [37](#)
 - close, [38](#)
 - create, [38](#)
 - send, [38](#)
 - write, [38](#)
- close
 - Client, [38](#)
- color
 - Drawing, [22](#)
- consts.h
 - FightResult, [75](#)
 - FightResultRow, [75](#)
 - OCCUPY_MAX, [75](#)
 - PLAYER_COLORS, [75](#)
 - uint, [75](#)
 - uint16, [75](#)
- Create
 - GameEngineClient, [45](#)
- create
 - Client, [38](#)
- Cube, [38](#)
- curr
 - Screen, [29](#)
- Dodaj
 - Planet, [51](#)
- Dokumentacja katalogu include/, [17](#)
- Dokumentacja katalogu src/, [18](#)
- Drawing, [19](#)
 - clearZBuff, [21](#)
 - color, [22](#)
 - drawLine, [21](#)
 - drawQuad, [21](#)
 - drawTriangle, [21](#)
 - putPix, [21](#)
 - SameSide, [22](#)
 - setColor, [22](#)
 - setObj, [22](#)
 - setSurface, [22](#)
- drawLine
 - Drawing, [21](#)
- drawQuad
 - Drawing, [21](#)
- drawTriangle
 - Drawing, [21](#)

EndGame
 GameEngineClient, 45
 EndTurn
 GameEngine, 41
 Planet, 52
 eq2d
 Vertex, 71
 FightResult
 consts.h, 75
 FightResultRow
 consts.h, 75
 flush
 Sprite, 61
 SpriteSDL2D, 64
 GameEngine, 39
 AddPlayer, 40
 CanDoAction, 41
 EndTurn, 41
 GameEngine, 40
 IsEndGame, 41
 Move, 41
 RemovePlayer, 42
 GameEngineBase, 42
 ActPlayer, 43
 GameEngineBase, 43
 GetPlanet, 43
 GetSize, 44
 GameEngineClient, 44
 Create, 45
 EndGame, 45
 MainLoop, 45
 PlanetUpdate, 46
 SendEndTurn, 46
 SendMove, 46
 getLineLen
 Text, 68
 GetPlanet
 GameEngineBase, 43
 GetSize
 GameEngineBase, 44
 getWordLen
 Text, 68
 id
 Screen, 29
 include/consts.h, 73
 info
 Screen, 30
 init
 WindowEngine, 33
 IsEndGame
 GameEngine, 41
 itsX
 Point, 54
 itsY
 Point, 54
 itsZ
 Point, 54
 kup
 Screen, 27
 length
 Message, 48
 loadGfx
 Sprite, 61
 loadMask
 Sprite, 61
 MainLoop
 GameEngineClient, 45
 mdown
 Screen, 27
 Message, 47
 length, 48
 Message, 48
 operator=, 48
 source, 49
 mmove
 Screen, 28
 MOVE
 RETURNS, 23
 Move
 GameEngine, 41
 mroll
 Screen, 28
 mup
 Screen, 28
 OCCUPY_MAX
 consts.h, 75
 operator+=
 Text, 68
 operator=
 Message, 48
 Text, 69
 Participant, 49
 Planet, 50

- Atak, [51](#)
- Dodaj, [51](#)
- EndTurn, [52](#)
- Planet, [51](#)
- RetGracz, [52](#)
- RetJednostki, [52](#)
- RetOkupant, [52](#)
- RetPoziom, [52](#)
- SetPlayer, [53](#)
- Zabierz, [53](#)
- PlanetUpdate
 - GameEngineClient, [46](#)
- PLAYER_COLORS
 - consts.h, [75](#)
- Point, [53](#)
 - itsX, [54](#)
 - itsY, [54](#)
 - itsZ, [54](#)
- print
 - Sprite, [61](#)
 - SpriteSDL2D, [64](#)
- putPix
 - Drawing, [21](#)
- receive
 - Server, [56](#)
- reload
 - Sprite, [62](#)
- RemovePlayer
 - GameEngine, [42](#)
- RetGracz
 - Planet, [52](#)
- RetJednostki
 - Planet, [52](#)
- RetOkupant
 - Planet, [52](#)
- RetPoziom
 - Planet, [52](#)
- RETURNS, [23](#)
 - MOVE, [23](#)
- Room, [54](#)
 - search, [55](#)
- rotateArb
 - Screen, [28](#)
- SameSide
 - Drawing, [22](#)
- Screen, [24](#)
 - addMessage, [27](#)
 - cid, [29](#)
 - curr, [29](#)
 - id, [29](#)
 - info, [30](#)
 - kup, [27](#)
 - mdown, [27](#)
 - mmove, [28](#)
 - mroll, [28](#)
 - mup, [28](#)
 - rotateArb, [28](#)
 - setCurrentPlayerID, [29](#)
 - setPlayerID, [29](#)
 - updateArea, [29](#)
- search
 - Room, [55](#)
- send
 - Client, [38](#)
 - Server, [56](#)
- SendEndTurn
 - GameEngineClient, [46](#)
- SendMove
 - GameEngineClient, [46](#)
- Server, [55](#)
 - ~Server, [56](#)
 - receive, [56](#)
 - send, [56](#)
- Session, [57](#)
- setAlign
 - Text, [69](#)
- setColor
 - Drawing, [22](#)
- setCurrentPlayerID
 - Screen, [29](#)
- setDim
 - Text, [69](#)
- setObj
 - Drawing, [22](#)
- SetPlayer
 - Planet, [53](#)
- setPlayerID
 - Screen, [29](#)
- setSpritePtrs
 - Sprite, [62](#)
- setStr
 - Text, [69](#)
- setSurface
 - Drawing, [22](#)
- source
 - Message, [49](#)
- Sprite, [58](#)
 - animate, [61](#)

- flush, [61](#)
- loadGfx, [61](#)
- loadMask, [61](#)
- print, [61](#)
- reload, [62](#)
- setSpritePtrs, [62](#)
- Sprite::Anim, [35](#)
- Sprite::Anim::AnimFrame, [36](#)
- Sprite::SpritePtr, [63](#)
- SpriteSDL2D, [63](#)
 - flush, [64](#)
 - print, [64](#)
- Text, [65](#)
 - addStr, [68](#)
 - Align, [68](#)
 - getLineLen, [68](#)
 - getWordLen, [68](#)
 - operator+=, [68](#)
 - operator=, [69](#)
 - setAlign, [69](#)
 - setDim, [69](#)
 - setStr, [69](#)
 - update, [70](#)
- uint
 - consts.h, [75](#)
- uint16
 - consts.h, [75](#)
- update
 - Text, [70](#)
- updateArea
 - Screen, [29](#)
- Vertex, [70](#)
 - eq2d, [71](#)
- WindowEngine, [30](#)
 - addKeyDownEventHandler, [32](#)
 - addKeyPressedEventHandler, [32](#)
 - addKeyUpEventHandler, [33](#)
 - addMouseDownEventHandler, [33](#)
 - addMouseMotionEventHandler, [33](#)
 - addMouseUpEventHandler, [33](#)
 - init, [33](#)
- write
 - Client, [38](#)
- Zabierz
 - Planet, [53](#)