Podręcznik

Wygenerowano przez Doxygen 1.7.3

Sun Jan 29 2012 02:05:08

Spis treści

Rozdział 1

RTTT - Risky Tic Tac Toe

1.1 Opis gry

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

1.2 Reguly panujace w kosmosie

1.2.1 Zdobywanie planet

Podstawowym elementem gry są posiadane planety. Aby podbić planete, należy umieścić na niej swoje jednoski. Wyslane jednostki po dotarciu do celu, walczą z stacjonujacymi tam statkami wroga. Po wygranej bitwie, planeta przechodzi w stan okupacji. Jeśli jest to planeta neutralna, należy ją okupować (posiadac tam conajmniej 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 tworzonego 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, nalezy zdobyć planety znajdujące się w jednej lini 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 Screen::mup(int x, int y, int key) /todo

--todo ?todo ∼∼todo∼∼ /todo/ to(-_-)do [todo]

--todo ?todo ∼∼todo∼∼ /todo/ to(-_-)do [todo]

Składowa Screen::updateArea(vector< pair< Vertex, Planet > > &items)

Rozdział 4

Struktura katalogów

4.1 Katalogi

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

Rozdział 5

Indeks przestrzeni nazw

5.1 Lista przestrzeni nazw

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

Rozdział 6

Indeks klas

6.1 Hierarchia klas

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

Rozdział 7

Indeks klas

7.1 Lista klas

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

Rozdział 8

Indeks plików

8.1 Lista plików

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

Rozdział 9

Dokumentacja katalogów

9.1 Dokumentacja katalogu include/

Pliki

- 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 **SocketSingleton.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 **SocketSingleton.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 oraz bufora obiektów.

- void setSurface (SDL_Surface ∗srf)

  Ustawia aktualną powierzchnię do rysowania. Nigdzie nie jest sprawdzane, czy nie jest NULLem.

- SDL_Surface ∗ getSurface ()

  Zwraca aktualną powierzchnię do rysowania.

- void setColor (unsigned int sc)

  Ustawia aktualny kolor, 0xAARRGGBB.

- 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 przeźroczystosci 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** (0.7071, 0.7071, 0)

10.1.1 Opis szczegółowy

Funkcje obsługujące rysowanie.

10.1.2 Dokumentacja funkcji

10.1.2.1 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 przesuń się o *dx, dy*

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

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

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

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

10.1.2.4 void Drawing::setObj (void * *obj*)

Ustawia aktualny obiekt wpisywany do bufora obiektów.

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

Parametry

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

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 funkcjii

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 sie powiodło
- MOVE_FIGHT - jeśli odbyła się walka

10.3 Dokumentacja przestrzeni nazw Screen

Chyba cala 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 handlery 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)
- void updateArea (vector< pair< Vertex, Planet > > &items)

  Aktualizacja pola gry.

- void addMessage (const string &msg)

  Wypisanie wiadomości msg.

Zmienne

- bool **lmb** = false
- bool **rmb** = false
- bool **mmb** = false
- bool **moved** = 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**
- Cube ∗ **src** = NULL
- Cube ∗ **dst** = NULL
- int **army** = 0
- Text **info** (0, 8, 8, 0, 0, 0, NULL,"", SCREENWIDTH-16, SCREENHEIGHT-16)
- Text **curr** (0, 8, SCREENHEIGHT-60, 0, 0, 0, NULL,"", SCREENWIDTH-16, 16)

- list< Text > **msgs**
- float **msgTimer** = 0
- Sprite * **bg**
- Vertex **tl**
- Vertex **scrtl**

10.3.1 Opis szczegółowy

Chyba cala logika okienka jest tutaj zawarta. Obsluga rysowania pola gry, obrotow, kliknieca na klocki~

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::mup (int *x*, int *y*, int *key*)

Wyslij informacje o wybranym celu i armii

Do zrobienia

/todo

Do zrobienia

--todo ?todo ~~todo~~ /todo/ to(-_-)do [todo]

Wyslij informacje o wybranym celu i armii

Do zrobienia

/todo

Do zrobienia

--todo ?todo ~~todo~~ /todo/ to(-_-)do [todo]

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

Aktualizacja pola gry.

Wywolywana po otrzymaniu zbiorczych informacji o aktualnym stanie pola gry

Do zrobienia

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 po puszczeniu klawisza na klawiaturze.

- bool addKeyPressedEventHandler (void(∗handle)(int))

  Rejestracja funkcji wywoływanej po po przytrzymaniu klawisza na klawiaturze.

- bool addMouseDownEventHandler (void(∗handle)(int, int, int))

  Rejestracja funkcji wywoływanej po wcisnieciu 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.

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 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 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 wcisnieciu 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 SDLem
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 metoda binduje handler do_close z metodą post socketu

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

11.3.1 Opis szczegółowy

Połączenie z serwerem. Klasa odpowiedzialna za obsługę połączenia z serwerem

Autor

Paweł Ściegienny

11.3.2 Dokumentacja konstruktora i destruktora

11.3.2.1 Client::Client (boost::asio::io_service & *io_service,* const char ∗ *host,* const char ∗ *port*)

Konstruktor połączenia Konstruktor. Ustawia handler połączenia.

Parametry

in	*host*	hostname
in	*ip*	adres ip

11.3.3 Dokumentacja funkcji składowych

11.3.3.1 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)
- void **reset** ()

Atrybuty publiczne

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

Statyczne atrybuty publiczne

- static const int **VERT_COUNT** = 24

Dokumentacja dla tej struktury została wygenerowana z pliku:

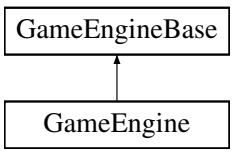
- src/screen.cpp

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 ()
  Konczy ture.
- void RemovePlayer (uint16 player)
  Usuwa gracza.
- RETURNS::MOVE Move (const Vertex &src, const Vertex &dst, uint16 num)
  Przenosi jednoski z jednej planety na drugą
- std::string **PlanetToString** (const Planet &planeta)

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 sześcianu 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::EndTurn ( )

Konczy ture.

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

Zwraca

Zwraca numer następnego gracza.

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

Przenosi jednoski 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

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

Zwraca

Zwraca ERRORS::MOVE

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

Usuwa gracza.

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

Możliwe do wykorzystania zarówno czy odłączeniu się gracza jak również czy pokananiu danego gracza

Parametry

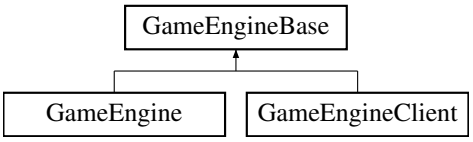
in	*player*	Numer gracza który ma zostać usunięty

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)
- uint16 ActPlayer () const

  Aktualny gracz.

- Planet & **GetPlanet** (const Vertex &src) const
- uint16 **GetSize** () const

Atrybuty chronione

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

11.6.1 Dokumentacja funkcji składowych

11.6.1.1 uint16 GameEngineBase::ActPlayer ( ) const

Aktualny gracz.

Zwraca numer aktualnego gracza.

Zwraca

Numer aktualnego gracza.

Dokumentacja dla tej klasy została wygenerowana z plików:

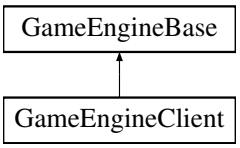
- include/gameenginebase.h
- src/gameenginebase.cpp

11.7 Dokumentacja klasy GameEngineClient

Klasa silnika gry dla klienta.

`#include <gameengineclient.h>`

Diagram dziedziczenia dla GameEngineClient



Metody publiczne

- **GameEngineClient** (uint16 size)
- Planet **StringToPlanet** (std::string msg)
- void **PlanetUpdate** (const Vertex &dst, const Planet &planet)

11.7.1 Opis szczegółowy

Klasa silnika gry dla klienta.

Autor

Marcin TORGiren Fabrykowski

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)
- void **operator=** (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 dłu;gość treści

- bool decode_header ()

  metoda odczytująca nagłówek

- void encode_header ()

  metoda zapisująca nagłówek

- void **source** (unsigned src)
- unsigned **source** () const
- std::string **getString** ()

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

11.8.3 Dokumentacja funkcji składowych

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

metoda zwracająca długość wiadomości

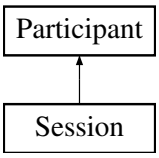
metoda zwracająca długość wiadomości WRAZ z długością nagłówka

Dokumentacja dla tej klasy została wygenerowana z plików:

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

11.9 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

11.10 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.

- RETURNS::ENDTURN EndTurn ()

  Kończy turę na danej planecie.

- RETURNS::MOVE **Zabierz** (uint16 ile)
- void **Dodaj** (uint16 ile)

Przyjaciele

- class GameEngine

11.10.1 Opis szczegółowy

Klasa planety. Opisuje właściwości planety - elementrarnej jednoski przestrzeni

Autor

Marcin TORGiren Fabrykowski

11.10.2 Dokumentacja konstruktora i destruktora

11.10.2.1 Planet::Planet ( )

Tworzy planete.

Konstruktor. Tworzy neutralna planete z losowa (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 planete Przeprowadza atak zadanej ilości jednostek na planete.

Parametry

ile	Liczba jednostek wroga, biąrąca 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 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

11.10.3.3 uint16 Planet::RetGracz ( ) const

Zwraca numer gracza-właściciela planety.

Zwraca numer gracza który jest aktualnie posiadaczem planety. Planeta może byc 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

11.10.3.4 uint16 Planet::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.

Zwraca

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

11.10.3.5 uint16 Planet::RetOkupant ( ) const

Zwraca numer gracza-okupanta planety.

Zwraca numer gracza który jest aktualnie okupantem planety

Zwraca

Numer gracza który okupuje planete, badź NULL jeśli takowego nie ma

11.10.3.6 uint16 Planet::RetPoziom ( ) const

Zwracam poziom zaawansowania okupacji.

Zwraca aktualny poziom okupacji. Wartość OCCUPY_MAX oznacza, ze planeta nie jest juz okupowana i jest w pełni przejęta

Zwraca

Poziom okupacji, bądź OCCUPY_MAX w przypadku gdy planeta nie jest okupowana i jest w pełni przejęta

11.10.3.7 void Planet::SetPlayer (uint16 *gracz*)

Ustawia nowego właściciela planety.

Metoda która ustawia nowego właściciela planety

Parametry

gracz	Numer gracza będącego nowym właścicielem

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

Metody publiczne

- void **join** (Participant_ptr participant)
- void **leave** (Participant_ptr participant)
- void deliver (const Message &msg)
- void **deliver** (unsigned who, const Message &msg)
- unsigned **search** (Participant *participant)
- Participant * search (unsigned ident)
- Message **todo** ()
- void **todo** (const Message msg)

11.12.1 Dokumentacja funkcji składowych

11.12.1.1 void Room::deliver (const Message & *msg*)

poczatek przykladu udupiania czesci pakietow

koniec przykladu udupiania czesci pakietow

11.12.1.2 Participant * Room::search (unsigned *ident*)

jak juz cos to zwrocmy pierwszego

Dokumentacja dla tej klasy została wygenerowana z plików:

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

11.13 Dokumentacja klasy Server

Metody publiczne

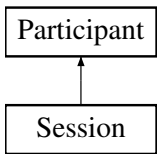
- **Server** (boost::asio::io_service &io_service, const tcp::endpoint &endpoint)
- void **handle_accept** (Session_ptr session, const boost::system::error_code &error)
- void **send** (const std::string &m)
- Message **receive** ()

Dokumentacja dla tej klasy została wygenerowana z plików:

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

11.14 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

11.15 Dokumentacja klasy SocketSingleton

Statyczne metody publiczne

- static boost::asio::io_service * **get** ()

Dokumentacja dla tej klasy została wygenerowana z plików:

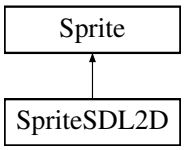
- include/SocketSingleton.hpp
- src/SocketSingleton.cpp

11.16 Dokumentacja klasy Sprite

Klasa zajmująca sie 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**

11.16.1 Opis szczegółowy

Klasa zajmująca sie 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

11.17 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**

11.17.1 Opis szczegółowy

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

Dokumentacja dla tej klasy została wygenerowana z plików:

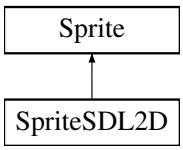
- include/sprite.h
- src/sprite.cpp

11.18 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.

11.18.1 Opis szczegółowy

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

11.18.2 Dokumentacja funkcji składowych

11.18.2.1 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.

11.18.2.2 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

11.19 Dokumentacja klasy Text

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)
- Text & **operator=** (string str)
- Text & **operator+=** (const char ∗str)
- Text & **operator+=** (string str)
- void **setPos** (float sx, float sy, float sz=0)
- void **setX** (float sx)
- void **setY** (float sy)
- void **setZ** (float sz)
- void **setPara** (float spx, float spy)
- void **setAlpha** (unsigned char sa)
- void **setFont** (Sprite ∗sSprite)
- void **setSprite** (Sprite ∗sSprite)
- void **setW** (unsigned int sw)
- void **setH** (unsigned int sh)
- void **setDim** (unsigned int sw, unsigned int sh)
- void **setStr** (const char ∗sStr)

- void **addStr** (const char *sStr)
- void **setAlign** (Align sa)
- void **setAlignLeft** ()
- void **setAlignCenter** ()
- void **setAlignRight** ()
- 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
- int **getSpSize** () const
- int **getTabSize** () const
- void **update** ()
- void **print** ()
- int **getWordLen** (const char *str)
- int **getLineLen** (const char *str)

Dokumentacja dla tej klasy została wygenerowana z plików:

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

11.20 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**

11.20.1 Opis szczegółowy

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

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** = 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 **RADTODEG** = 180.0f/3.141592653589793f
- const char **IMGEXT** [ ] = ".png"
- const char **ANIMEXT** [ ] = ".txt"
- const char **FONT** [ ] = "data/font_00"
- const char **BACKGROUND** [ ] = "data/bg_01"
- const float **MSG_HIDE_DELAY_FIRST** = 5.0f
- const float **MSG_HIDE_DELAY_NEXT** = 0.5f
- const unsigned int **MSG_MAX_COUNT** = 8
- const unsigned int PLAYER_COLORS [ ]

  Kolory graczy 1-8.

- const unsigned int PLANET_SRC_COLOR = 0x0058AF58

  Kolor wybranej planety zrodlowej.

- 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.

Skorowidz