Podręcznik

Wygenerowano przez Doxygen 1.7.3

Mon Feb 6 2012 15:16:53

Spis treści

Rozdział 1

RTTT -Risky Tick Tac Toe

1.1 Opis gry

1.1.1 Ogólne

Risky Tic Tac Toe jest połączeniem gry "Kółko i Krzyżyk" oraz elementów z gry "Ryzyko". Projekt został wykonany przez trzech studentów Informatyki Stosowanej na Wydziale Fizyki i Informatyki Stosowanej Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie. Skład zespołu przedstawia się następująco:

- Dawid Barnaś – specjalista od biblioteki SDL, główna osoba odpowiedzialna za poprawne wyświetlanie i obsługę zdarzeń
- Marcin Fabrykowski – główny programista, implementacja zasad gry, a także abstrakcyjnego poziomu przesyłanych komunikatów. Integracja modułu wyświetlania i modułu sieciowego z silnikiem gry.
- Paweł Ściegienny – odpowiedzialny za komunikację przez TCP/IP, buforowanie wiadomości pomiędzy serwerem a instancjami klienckimi, dokumentacja.

1.1.2 Opis gry

Prymitywna gra w kółko i krzyżyk nawet w 3 wymiarowej przestrzeni, stąd zasady RTTT zostały zmodyfikowane.

zasady Zasady poczatek Początek Na początku, gracz dostaje 3 jednostki, które może dowolnie rozplanować na planszy rozgrywka Rozgrywkka Gracz wybiera swoją prowincję na której ma >1 jednosktę z której chce wysłać jednoski, oraz docelową, którą chce przejąć.

Atakujący i obroniący się "rzucają kostką" K6 w liczbie odpowiadających jednostek walczących w danej potyczce. Porównywane są odpowiednio największe wyniki, np:

nieudany Nieudany

Atakujący wyrzuci: 3 4 6 6, natomiast:

Broniący wyrzuci: 5 5 5. porównujemy odpowiednio:

6>5 wygrywa atakujący

6>5 wygrywa atakujący

4<5 przegrywa atakujący

wynik takiej potyczki jest taki, że prowincja broniąca obroniła się, i pozostał jej jeden wojownik, natomiast atakujący stracił 1 wojownika. Wojska które przeżyły atak, wracają na prowincję z której wyruszyły.

udany Udany Atakujący wyrzuci 3 4 6 6, natomiast:

Broniący wyrzuci: 5 5 3, porównujemy odpowiednio:

6>5 wygrywa atakujący

6>5 wygrywa atakujący

4>3 wygrywa atakujący

broniący sie nie ma już jednostek, dlatego atakujący podbija tą prowincję i przenosi tam swoje jednostki. Liczba przeniesionych jednostek nie może być mniejsza niż pozostałych przy życiu po walce ani większa niż suma jednostek pozostałych na prowincji oraz pozostałych przy życiu minus 1. Jedna jednostka musi zostać na prowincji.

przejmowanie Przejmowanie prowincji Po udanym ataku na prowincje, dana prowincja staje się prowincją okupowaną przez danego gracz. Aby ją przejąc, gracz musi zawiesić na niej swoją flagę. Proces trwa 5 tur.

Jeśli gracz podbije prowincje która już należy do innego gracza (już ma jego flagę) gracz ten musi poczekać 5 tur na zdjęcie flagi przeciwnika oraz kolejne 5 tur na zawieszenie swojej.

W przypadku jeżeli jeden gracz wiesza swoją flage i w tym czasie prowincja zostanie przejęta przez innego gracza, podbijający musi poczekać tyle tur na zdjęcie flagi przeciwnika, ile ten poświęcił na jej wzniesienie. np:

Gracz X okupuje prowincje i ma ją przez 2 tury, jego flaga jest na "drugim poziomie". Następnie gracz Y podbija tą prowincje i musi poczekać 2 tury aż zdejmie flage przeciwnika a następnie 5 tur aby zawiesić swoją.

Analogicznie, jeśli w powyższym przypadku gracz Y podbija prowincje i ma ją przez jedną turę, to flaga przeciwnika spada z drugiego poziomu na pierwszy. Jeśli w tym czasie gracz X ponownie podbije prowincję musi czekać tylko 4 tury na oznaczenie tej prowincji jako jego, gdyż prowincja miałą już jego flagę na poziomie pierwszym.

Jeżeli gracz X podbije prowincje należącą do gracza Y, prowincja należy nadal do gracza Y do czasu gdy gracz X nie zdejmie jego flagi. Gracz Y jednak nie ma prawa wstawiać tam nowych jednostek.

koniec Koniec Koniec gry następuje gdy jeden z graczy przejmie (postawi w pełni flagę) na 5 prowincjach ułożonych w lini prostej, bądź gdy jeden z graczy się podda.

1.1.3 Wymagania sprzetowe

Procesor przynajmniej 1.2GHz

Przynajmniej 16MB pamięci RAM

System Windows/Linux

bibliteki Boost (Boost System, FileSystem, Thread) >=1.42, SDL1.2, SDL-Image1.2

Mysz/Touchpad

Monitor

Karta sieciowa

Gracz

Wygodne siedzisko

Przeciwnik (co najmniej 1)

1.2 Zalozenia wstepne przyjete w realizacji projektu.

Nasza unikatowa gra ma być unikatowa dzięki kilku elementom:

- gra ma pozwalać na grę zarówno na zdalnej maszynie, jak i na maszynie lokalnej
- to gracz decyduje o rozmiarze planszy
- sześciany będą oddalone od siebie w celu umożliwienia wysłania jednostek na niebrzegowe pola
- intuicyjna obsługa za pomocą myszy (naciśnij, przeciągnij, kliknij)
- nielimitowany obrót wokół każdej osi
- pola każdego z użytkowników wyróżniają się unikatowym kolorem

1.3 Analiza projektu

1.3.1 Specyfikacja interfejsu uzytkownika

Wybór planety źródłowej - Lewy przycisk myszy Wybór planety docelowej - Prawy przycisk myszy Wybór ilości jednostek - Rolka (góra - zwiększenie, dół - zmniejszenie) Wysłanie jednostek - Rolka (wciśnięcie) Obrót pola gry - Wciśnięty lewy przycisk myszy + ruch, klawisze strzałek Skalowanie pola gry - Wciśnięty prawy przycisk myszy + ruch Przesunięcie pola gry - Wciśnięta rolka + ruch

1.3.2 Wyodrebnienie i zdefiniowanie zadań

- komunikacja z użytkownikiem – mysz, klawiatura
- wyświetlenie okna, generowanie planszy – SDL
- komunikacja okna z silnikiem gry – Boost Asio, TCP/IP
- logika gry kółko i krzyżyk z elementami gry w ryzyko

1.3.3 Decyzja o wyborze narzedzi

Projekt został wykonany przy użyciu następujących bibliotek:

- Boost 1.46.1 – ze względu na znakomite możliwości Boost Asio do połączeń TCP/IP
- SDL 1.2 – generowanie okna
- Vim, Gedit, Eclipse
- repozytorium GIT
- Gentoo Linux, Debian, Ubuntu, MS Windows

analiza_czasowa Podział pracy i analiza czasowa Ze względu na to, iż nasz projekt dzieli się w naturalny sposób na trzy moduły, barbarzyństwem byłoby nie wykorzystać tego przy podziale obowiązków.

Po ustaleniu wspólnych interfejsów i po wielu owocnych dyskusjach (ok 10% czasu projektu), przystąpiliśmy do pracy.

W ten sposób szacujemy że poniższe zadania w sumie zajęły 90%

- Komunikacja sieciowa – 15% [Paweł Ściegienny]
- Stworzenie okna gry, implementacja algorytmu rysowania, obsługa zdarzeń – 20% [Dawid Barnaś]
- Implementacja logiki gry – 15%[Marcin Fabrykowski]
- Weryfikacja, ustalenie nowego SRS – 5% [razem]
- Refactoring komunikacji – 5% [Paweł Ściegienny]
- Optymalizacja wyświetlania – 10% [Dawid Barnaś]
- Integracja -15% [Marcin Fabrykowski]
- Uzupłenienie dokumentacji – 5% [Paweł Ściegienny]

1.4 Tutaj opis algorytmów graficznych...

1.5 Dokumentacja techniczna

Rozdział 2

Algorytmy

2.1 Algorytmy rysowania

2.1.1 Rysowanie linii

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. Zaczynając od jednego z punktów, odpal pętlę l razy
4. Dla każdej iteracji wypisz piksel w aktualnym punkcie i przesuń się o dx, dy

2.1.2 Rysowanie trojkata:

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

2.1.3 Wyszukiwanie obiektow:

1. Ustaw wskaźnik na obiekt
2. Używając ZBuffera sprawdź, czy można wstawić piksel
3. Jeśli tak, wstaw piksel, zaktualizuj ZBuffer i wstaw wskaźnik na obiekt do bufora obiektów

Rozdział 3

- Risky Tic Tac Toe old

Bitwa

3.1 Opis gry

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

3.2 Reguly panujace w kosmosie

3.2.1 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 tworzonego tam rządu. Dokładnie tyle ile gracz poświęcił na jego utworzenie.

3.2.2 Zdobywanie jednostek

Na każdej pobitej przez gracza planecie produkowane są statki kosmiczne. Tempo tworzenia statków wynosi 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 są Tworzone jednostki.

3.2.3 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 4

Protokoły

4.1 Komunikacji sieciowej

4.1.1 Inicjacja

1. Client: Hello
2. Server: Witam
3. Server: player <num> //Numer gracza jaki został mu przypisany
4. Server: size <num> //Rozmiar planszy na które prowadzona jest bitwa
5. Server: planet <num1> <num2> <num3> <num4> <num5> <num6> <num7> <num8> //Wysyła stan planet. Num1-3 pozycja planety, Num4-8 dane planety Planet::ToString()
6. Powyższe dla wszystkich planet na planszy
7. Server: act <num> //Numer aktualnego gracza

4.1.2 Rozgrywka

1. Client: move <num1> <num2> <num3> <num4> <num5> <num6> <num7> //Żądanie przeniesienia <num7> jednostek z planety o wsp. num1-3 na planete num4-6
2. Server: planet <num1> <num2> <num3> <num4> <num5> <num6> <num7> <num8> //Wysyła stan planet. Num1-3 pozycja planety, Num4-8 dane planety Planet::ToString()
3. Powyższe dla wszystkich planet na planszy
4. Client ponownie "move", bądź:
5. Client: end //Kończy turę gracza<i/li>

6. Server: planet <num1> <num2> <num3> <num4> <num5> <num6> <num7> <num8> //Wysyła stan planet. Num1-3 pozycja planety, Num4-8 dane planety Planet::ToString()
7. Powyższe dla wszystkich planet na planszy
8. Server: act <num> //Numer aktualnego gracza

Rozdział 5

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

Alfa całego obrazka

Parallax scrolling

Barwienie obrazka

Rozdział 6

Struktura katalogów

6.1 Katalogi

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

Rozdział 7

Indeks przestrzeni nazw

7.1 Lista przestrzeni nazw

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

Rozdział 8

Indeks klas

8.1 Hierarchia klas

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

Rozdział 9

Indeks klas

9.1 Lista klas

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

Rozdział 10

Indeks plików

10.1 Lista plików

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

Rozdział 11

Dokumentacja katalogów

11.1 Dokumentacja katalogu include/

Pliki

- plik **algorytmy.h**
- plik **Client.hpp**
- plik consts.h
- plik **doc.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 **protokoly.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**

11.2 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 12

Dokumentacja przestrzeni nazw

12.1 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 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)

  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.

12.1.1 Opis szczegółowy

Funkcje obsługujące rysowanie.

Autor

crm

12.1.2 Dokumentacja funkcji

12.1.2.1 void Drawing::clearZBuff ( )

Czyszczenie zbuffera.

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

12.1.2.2 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 przesuń się o *dx, dy*

12.1.2.3 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.

12.1.2.4 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

12.1.2.5 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.

12.1.2.6 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

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

Ustawia aktualny kolor.

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

12.1.2.8 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 dowolny obiekt. Musisz pamiętać, co podsyłasz, ponieważ bufor obiektów korzysta z wbudowanego w C++ dynamicznego rzutowania typów (void*)

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

Ustawia aktualną powierzchnię do rysowania.

Nigdzie nie jest sprawdzane, czy nie jest NULLem.

12.1.3 Dokumentacja zmiennych

12.1.3.1 unsigned int Drawing::color = 0xFFFFFFFF

Aktualny kolor.

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

12.2 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

12.2.1 Opis szczegółowy

Zawiera komunikaty zwracane z funkcjii

12.2.2 Dokumentacja typów wyliczanych

12.2.2.1 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

12.3 Dokumentacja przestrzeni nazw Screen

Chyba cala 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 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)

  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)

  Ustawia wskaźnik na GameEngineClient.

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

  Minimalne z kostki.

- float maxz

  Maksymalne z kostki.

- float tminz

  Tymczasowe minimalne z kostki.

- float tmaxz

  Tymczasowe maksymalne z kostki.

- 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 wysłania.

- 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

  Wskaźnik na GameEngineClient.

- Vertex tl

  Przesunięcie kostek do (0, 0, 0)

- Vertex scrtl

  Przesunięcie kostek do środka ekranu.

12.3.1 Opis szczegółowy

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

Autor

crm

12.3.2 Dokumentacja funkcji

12.3.2.1 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*.

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

Obsługa puszczenia przycisku na klawiaturze.

Parametry

key	Kod puszczonego klawisza

12.3.2.3 void Screen::mdown (int *x*, int *y*, int *key*)

Obsługa kliknięcia.

Parametry

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

12.3.2.4 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

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

Obsługa kliknięcia rolką

Parametry

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

12.3.2.6 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

12.3.2.7 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

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

Ustawia ID gracza aktualnie wykonującego ruch na podane.

Zależnie od ID gracza będzie rysowany trójkąt w odpowiednim kolorze

12.3.2.9 void Screen::setGameEngineClient (GameEngineClient * *e*)

Ustawia wskaźnik na *GameEngineClient*.

Parametry

e	Wskaźnik na *GameEngineClient*

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

Ustawia ID gracza na podane.

Zależnie od ID gracza będzie rysowana ramka innego koloru

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

Aktualizacja pola gry.

Wywolywana po otrzymaniu zbiorczych informacji o aktualnym stanie pola gry

12.3.3 Dokumentacja zmiennych

12.3.3.1 int Screen::cid = 0

ID gracza wykonującego ruch.

Używane do rysowania kolorowego trójkąta

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

Dolny tekst.

Informacje o planecie zjandującej się pod kursorem

12.3.3.3 int Screen::id = 0

ID gracza.

Używane do rysowania kolorowej ramki wokół poziomu

12.3.3.4 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

12.3.3.5 float Screen::maxz

Maksymalne z kostki.

Używane do cieniowania. Wartość otrzymana w poprzedniej iteracji.

12.3.3.6 float Screen::minz

Minimalne z kostki.

Używane do cieniowania. Wartość otrzymana w poprzedniej iteracji.

12.3.3.7 Vertex Screen::tl

Przesunięcie kostek do (0, 0, 0)

Używane przy obracaniu kostek

12.4 Dokumentacja przestrzeni nazw WindowEngine

Tworzenie okienka, obsługa zdarzeń

Wyliczenia

- enum RenderType { **SDL**, **OPENGL** }
  Typ rendera.
- enum WaitType { **DELAY**, **DELTA** }
  Typ przerwy między klatkami.

Funkcje

- bool initSDL ()
  Inicjalizacja SLD'a.
- 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 ()
  Zwraca typ rendera.
- WaitType getWaitType ()
  Zwraca typ przerwy.
- float getDelta ()
  Zwraca deltę
- 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

  Informacja o działaniu.

- unsigned int flags = 0x0

  Flagi inicjalizacyjne.

- unsigned int frameTime = 0

  Czas poświęcony na klatkę

- float delta = 0.0f

  Delta.

- set< void(∗)(int)> keyDownHandles

  Wskaźniki na funkcje obsługujące wciśnięcie klawisza.

- set< void(∗)(int)> keyUpHandles

  Wskaźniki na funkcje obsługujące puszczenie klawisza.

- set< void(∗)(int)> keyPressedHandles

  Wskaźniki na funkcje obsługujące przytrzymanie klawisza.

- set< void(∗)(int, int, int)> mouseDownHandles

  Wskaźniki na funkcje obsługujące wciśnięcie przycisku myszy.

- set< void(∗)(int, int, int)> mouseUpHandles

  Wskaźniki na funkcje obsługujące puszczenie przycisku myszy.

- set< void(∗)(int, int, int)> mouseMotionHandles

  Wskaźniki na funkcje obsługujące ruch myszy.

- RenderType rt

  Typ rendera.

- WaitType wt

  Typ przerwy.

- SDL_Event event

  Zdarzenie SDL'a.

- SDL_Surface ∗ screen = NULL

  Ekran.

- Uint8 ∗ keys = SDL_GetKeyState(NULL)

  Stan klawiszy.

12.4.1 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

12.4.2 Dokumentacja typów wyliczanych

12.4.2.1 enum WindowEngine::RenderType

Typ rendera.

- SDL - Używa biblioteki SDL
- OPENGL - Używa biblioteki OpenGL

12.4.2.2 enum WindowEngine::WaitType

Typ przerwy między klatkami.

- DELAY - Zwykłe uśpienie procesu na *x* milisekund
- DELTA - Działanie z maksymalną prędkością, obliczana jest *delta*, zmienna informująca o czasie poświęconym na ostatnią klatkę.

12.4.3 Dokumentacja funkcji

12.4.3.1 bool WindowEngine::addKeyDownEventHandler (void(∗)(int) *handle*)

Rejestracja funkcji wywoływanej po wciśnięciu klawisza na klawiaturze.

Parametry

handle	Wskaźnik na funkcję. Argument to kod klawisza

12.4.3.2 bool WindowEngine::addKeyPressedEventHandler (void(∗)(int) *handle*)

Rejestracja funkcji wywoływanej po po przytrzymaniu klawisza na klawiaturze.

Parametry

handle	Wskaźnik na funkcję. Argument to kod klawisza

12.4.3.3 bool WindowEngine::addKeyUpEventHandler (void(∗)(int) *handle*)

Rejestracja funkcji wywoływanej po po puszczeniu klawisza na klawiaturze.

Parametry

handle	Wskaźnik na funkcję. Argument to kod klawisza

12.4.3.4 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)

12.4.3.5 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)

12.4.3.6 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)

12.4.3.7 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 13

Dokumentacja klas

13.1 Dokumentacja klasy Sprite::Anim

Informacje o animacji.

`#include <sprite.h>`

Komponenty

- class AnimFrame

  Klatka animacji.

Metody publiczne

- Anim (float aspd, int fret)

  Konstruktor.

- ~Anim ()

  Destruktor.

- 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.

13.1.1 Opis szczegółowy

Informacje o animacji.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/sprite.h

13.2 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)

  Konstruktor.

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**

13.2.1 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

13.3 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 wiadomosc 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.

13.3.1 Opis szczegółowy

Połączenie z serwerem. Klasa odpowiedzialna za obsługę połączenia z serwerem

Autor

Paweł Ściegienny

13.3.2 Dokumentacja funkcji składowych

13.3.2.1 void Client::close ( )

metoda zamykająca połączenie

metoda binduje handler do_close z metodą post socketu

13.3.2.2 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

13.3.2.3 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 strina który ma zostać wysłany

13.3.2.4 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

13.4 Dokumentacja struktury Cube

Kostka widoczna na ekranie.

Metody publiczne

- Cube (int x=0, int y=0, int z=0, unsigned int col=0xFFFFFFFF)
  Konstruktor.
- Cube (const Cube &c)
  Konstruktor kopiujący.
- operator Vertex ()
  Rzutowanie na Vertex.
- void reset ()
  Wyzerowanie współrzędnych Vertexów.

Atrybuty publiczne

- int **x**
- int **y**
- int **z**
- unsigned int col
  Kolor.
- int army
  Ilość jednostek na 'planecie'.
- float pct
  Stopień przejęcia 'planety'.
- float roll
  Unused.
- Vertex verts [VERT_COUNT]
  Lista Vertexów.

Statyczne atrybuty publiczne

- static const int VERT_COUNT = 24
  Ilość Vertexów dla sześcianu (nie zmieniać bez powodu)

13.4.1 Opis szczegółowy

Kostka widoczna na ekranie. Posiada współrzędne, listę vertexów, kolor, wielkość armii i stopień przejęcia

Dokumentacja dla tej struktury została wygenerowana z pliku:

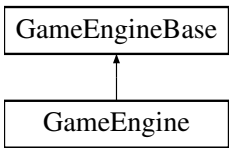
- src/screen.cpp

13.5 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 operacje.

- bool IsEndGame () const

  Sprawdza czy to już koniec gry.

13.5.1 Opis szczegółowy

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

Autor

Marcin TORGiren Fabrykowski

13.5.2 Dokumentacja konstruktora i destruktora

13.5.2.1 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

13.5.3 Dokumentacja funkcji składowych

13.5.3.1 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

13.5.3.2 bool GameEngine::CanDoAction (uint16 *socket_id*)

Sprawdza czy gracz może wykonać jakąkolwiek operacje.

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

13.5.3.3 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.

13.5.3.4 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

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

Przenosi jednostki z jednej planety na drugą

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

Parametry

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

Zwraca

Zwraca ERRORS::MOVE

13.5.3.6 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:

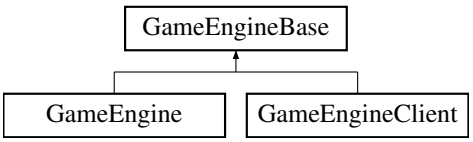
- include/gameengine.h
- src/gameengine.cpp

13.6 Dokumentacja klasy GameEngineBase

Klasa bazowa dla silników gry.

`#include <gameenginebase.h>`

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

Lista graczy.

- std::set< uint16 >::iterator itsActPlayer

  Aktualny gracz.

- Planet *** itsPlanety

  Planety na planszy.

- uint16 itsSize

  Rozmiar pola bitwy.

13.6.1 Opis szczegółowy

Klasa bazowa dla silników gry. Klasa bazowa dla klas silnika gry i klienckiego silnika gry

Autor

Marcin TORGiren Fabrykowski

13.6.2 Dokumentacja konstruktora i destruktora

13.6.2.1 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 sześciany o rozmiarze size

Parametry

size	Rozmiar boku sześcianu pola bitwy liczony w ilości planet

13.6.3 Dokumentacja funkcji składowych

13.6.3.1 uint16 GameEngineBase::ActPlayer ( ) const

Aktualny gracz.

Zwraca numer aktualnego gracza.

Zwraca

Numer aktualnego gracza.

13.6.3.2 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

13.6.3.3 uint16 GameEngineBase::GetSize ( ) const

Zwraca rozmiar pola bitwy.

Zwraca rozmiar pola bitwy

Zwraca

Rozmair pola bitwy

13.6.4 Dokumentacja atrybutów składowych

13.6.4.1 std::set<uint16>::iterator GameEngineBase::itsActPlayer [protected]

Aktualny gracz.

Iterator wskazujący na aktualnego gracza

13.6.4.2 Planet*** GameEngineBase::itsPlanety [protected]

Planety na planszy.

Tablica trzy wymiarowa zawierająca planety pola bitwy

13.6.4.3 std::set<uint16> GameEngineBase::itsPlayers [protected]

Lista graczy.

Zawiera zbiór numerów graczy biorących udział w rozgrywce. Gracze wyeliminowani są z tej listy usuwani

13.6.4.4 uint16 GameEngineBase::itsSize [protected]

Rozmiar pola bitwy.

Długość boku sześciennego pola bitwy

Dokumentacja dla tej klasy została wygenerowana z plików:

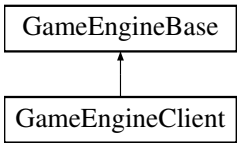
- include/gameenginebase.h
- src/gameenginebase.cpp

13.7 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.

13.7.1 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

13.7.2 Dokumentacja funkcji składowych

13.7.2.1 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

13.7.2.2 void GameEngineClient::EndGame ( )

Ustawia flagę końca gry.

Ustawia flagę zakończonej gry

13.7.2.3 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

13.7.2.4 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

13.7.2.5 void GameEngineClient::SendEndTurn ( )

Wysyła żądanie końca tury.

Wysyła do serwera sygnalizację zakończenia tury przez danego gracza

13.7.2.6 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

13.8 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 dłu;gość 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 wiadmości
- std::string getString ()
  metoda konwertująca wiadomość do stringa [depracated]

13.8.1 Opis szczegółowy

Przesyłana wiadomość. Klasa odpowiedzialna za poprawne informacje o wiadomości

Autor

Paweł Ściegienny

13.8.2 Dokumentacja konstruktora i destruktora

13.8.2.1 Message::Message ( )

Konstruktor.

Konstruktor domyślny - inicjalizuje długość wiadomości

13.8.2.2 Message::Message (const Message & *src*)

Konstruktor.

Konstruktor kopiujący - w pewnym momencie bardzo istotny element programu

13.8.3 Dokumentacja funkcji składowych

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

metoda zwracająca długość wiadomości

metoda zwracająca długość wiadomości WRAZ z długością nagłówka

13.8.3.2 void Message::operator= (const Message & *src*)

operator przypisania

operator przypisania [depracated]

13.8.3.3 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:

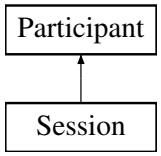
- include/Message.hpp
- src/Message.cpp

13.9 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

13.9.1 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

13.10 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
- 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 ()
  Konwertuje planetę do postaci stringa.
- operator std::string ()
  Konwertuje planetę do postaci stringa.

Statyczne metody publiczne

- static Planet ToPlanet (std::string str)
  Tworzy planetę na podstawie stringa.

13.10.1 Opis szczegółowy

Klasa planety. Opisuje właściwości planety - elementarnej jednostki przestrzeni

Autor

Marcin TORGiren Fabrykowski

13.10.2 Dokumentacja konstruktora i destruktora

13.10.2.1 Planet::Planet ( )

Tworzy planetę

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

13.10.3 Dokumentacja funkcji składowych

13.10.3.1 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

13.10.3.2 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

13.10.3.3 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

13.10.3.4 Planet::operator std::string ( )

Konwertuje planetę do postaci stringa.

To samo to ToString();

Zobacz również

ToPlanet(std::string str)

13.10.3.5 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

13.10.3.6 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

13.10.3.7 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

13.10.3.8 uint16 Planet::RetPoziom ( ) const

Zwracam poziom zaawansowania okupacji.

Zwraca aktualny poziom okupacji. Wartość OCCUPY_MAX oznacza, ze 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

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

Ustawia nowego właściciela planety.

Metoda która ustawia nowego właściciela planety

Parametry

gracz	Numer gracza będącego nowym właścicielem

13.10.3.10 Planet Planet::ToPlanet (std::string *str*) [static]

Tworzy planetę na podstawie stringa.

Tworzy planetę na podstawie stringa o formacie: Nr_gracza Poziom_flagi Nr_Gracza_-Posiadacza Liczba_Jednostek Nr_Gracza_Okupanta

Zwraca

Klasa planety powstała po interpretacji stringa

Zobacz również

ToString()

13.10.3.11 std::string Planet::ToString ( )

Konwertuje planetę do postaci stringa.

Konwertuje obiekt klasy Planeta do postaci stringa. Format to:

Nr_gracza Poziom_flagi Nr_Gracza_Posiadacza Liczba_Jednostek Nr_Gracza_Okupanta

Zwraca

String reprezentujący tą planete

Zobacz również

ToPlanet(std::string str)

13.10.3.12 RETURNS::MOVE Planet::Zabierz (uint16 *ile*)

Zabiera z planety zadaną liczbę jednostek.

Zmniejsza liczbę jednostek na danej planecie o zadaną zwartość. 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

13.11 Dokumentacja klasy Point

Klasa położenia w przestrzeni.

`#include <point.h>`

Atrybuty publiczne

- uint16 itsX
- uint16 itsY
- uint16 itsZ

13.11.1 Opis szczegółowy

Klasa położenia w przestrzeni. Obrazuje położenie punktu w przestrzeni planszy

Autor

Marcin TORGiren Fabrykowski

13.11.2 Dokumentacja atrybutów składowych

13.11.2.1 uint16 Point::itsX

Położenie na osi X

13.11.2.2 uint16 Point::itsY

Położenie na osi Y

13.11.2.3 uint16 Point::itsZ

Położenie na osi Z

Dokumentacja dla tej klasy została wygenerowana z pliku:

- include/point.h

13.12 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 uczesnika 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

13.12.1 Opis szczegółowy

miejsce gdzie zbiegają się sockety. implementacja interfejsu Participant. Konkretne rozwiązania

Autor

Paweł Ściegienny

13.12.2 Dokumentacja funkcji składowych

13.12.2.1 Participant ∗ Room::search (unsigned *ident*)

metoda pozwalająca znaleźć socket na podstawie ID klienta

jak juz cos to zwrocmy pierwszego

Dokumentacja dla tej klasy została wygenerowana z plików:

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

13.13 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.

13.13.1 Dokumentacja konstruktora i destruktora

13.13.1.1 Server::~Server ( ) [inline]

Destruktor.

Zamyka połączenie, sprząta

13.13.2 Dokumentacja funkcji składowych

13.13.2.1 Message Server::receive ( ) [inline]

Metoda odbierająca.

Metoda pozwalająca sprawdzić co serwer ma do roboty

Zwraca

Message wiadomość zawierającą id i treść

13.13.2.2 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

13.13.2.3 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:

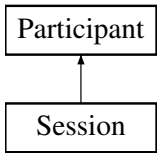
- include/Server.hpp
- src/Server.cpp

13.14 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)
  dostarczacz 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

13.14.1 Opis szczegółowy

reprezentacja sesji klasa utrzymująca szerokorozumianą sesję

Autor

Paweł Ściegienny

Dokumentacja dla tej klasy została wygenerowana z plików:

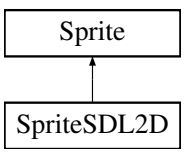
- include/Session.hpp
- src/Session.cpp

13.15 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)

  Konstruktor.

- virtual ~Sprite ()

  Destruktor.

- const std::string & getName ()

  Zwraca nazwę grafiki.

- void getDim (int &gw, int &gh)

  Zwraca wymiary obrazka.

- int getW ()

  Zwraca Szerokość

- int getH ()

  Zwraca Wysokość

- Anim & getAnim (unsigned int i)

  Zwraca animację o numerze i.

- unsigned int getAnimCount ()

  Zwraca ilość animacji.

- 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.

13.15.1 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.

Autor

crm

13.15.2 Dokumentacja konstruktora i destruktora

13.15.2.1 Sprite::Sprite (const std::string & *name* = " ", int *w* = 0, int *h* = 0)

Konstruktor.

Parametry

name	Nazwa grafiki
w	Szerokość obrazka
h	Wysokość obrazka

13.15.3 Dokumentacja funkcji składowych

13.15.3.1 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

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

Wyrzucenie bufora.

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

Implementowany w SpriteSDL2D.

13.15.3.3 Anim& Sprite::getAnim (unsigned int *i*) [inline]

Zwraca animację o numerze *i*.

Parametry

i	Numer animacji

13.15.3.4 void Sprite::getDim (int & *gw*, int & *gh*) [inline]

Zwraca wymiary obrazka.

Parametry

out	*gw*	Szerokość
out	*gh*	Wysokość

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

Wczytywanie grafiki.

Do zdefiniowania w klasach poniżej

Parametry

name	Nazwa

13.15.3.6 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	Głębia koloru

13.15.3.7 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.

13.15.3.8 void Sprite::reload ( ) [static]

Ponowne wczytanie Sprite.

Używać po przestawieniu trybu wyświetlania

13.15.3.9 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

13.16 Dokumentacja klasy Sprite::SpritePtr

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

`#include <sprite.h>`

Metody publiczne

- SpritePtr ()
  Konstruktor domyślny.
- SpritePtr (Sprite *s)
  Konstruktor.
- ~SpritePtr ()
  Destruktor.
- void operator= (Sprite *s)
  Przypisanie.
- void setSprite (Sprite *s)
  Przypisanie.
- void setAnim (int sa)
  Zmiana animacji.
- void setSpd (float ss)

Zmiana szybkości animacji.

- void animate ()

  Animowanie.

- 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)

  Wyświetlenie grafiki.

Atrybuty publiczne

- Sprite ∗ sprite

  Wskaźnik na grafikę

13.16.1 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.

13.16.2 Dokumentacja konstruktora i destruktora

13.16.2.1 Sprite::SpritePtr::SpritePtr (Sprite ∗ *s*) [inline]

Konstruktor.

Parametry

s	Wskaźnik na grafikę

13.16.3 Dokumentacja funkcji składowych

13.16.3.1 void Sprite::SpritePtr::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)

Wyświetlenie grafiki.

Parametry

in	*x*	Współrzędna x
in	*y*	Współrzędna y
in	*z*	Współrzędna z
in	*alpha*	Przeźroczystość, 0-255

in	*px*	Parallax scrolling, poziomy
in	*py*	Parallax scrolling, pionowy
in	*r*	Czerwony
in	*g*	Zielony
in	*b*	Niebieski

Dokumentacja dla tej klasy została wygenerowana z plików:

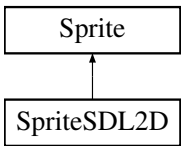
- include/sprite.h
- src/sprite.cpp

13.17 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)
- virtual ~SpriteSDL2D ()
- 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.

13.17.1 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.

Autor

crm

13.17.2 Dokumentacja konstruktora i destruktora

13.17.2.1 SpriteSDL2D::SpriteSDL2D (const std::string & *name* = " ", int *w* = 0, int *h* = 0) [inline]

Parametry

Konstruktor	
name	Nazwa grafiki
w	Szerokość
h	Wysokość

13.17.2.2 virtual SpriteSDL2D::~SpriteSDL2D ( ) [inline, virtual]

Parametry

Destruktor	

13.17.3 Dokumentacja funkcji składowych

13.17.3.1 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.

13.17.3.2 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

13.18 Dokumentacja klasy Text

Klasa wyświetlająca tekst.

`#include <text.h>`

Typy publiczne

- enum Align { **LEFT**, **CENTER**, **RIGHT** }

Metody publiczne

- Text ()

  Konstruktor domyślny.

- 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)

  Konstruktor.

- Text (const Text &txt)

  Konstruktor kopiujący.

- ∼Text ()

  Destruktor.

- 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 przeźroczystości 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
  Zwraca pozycję tekstu.
- float getX () const
  Zwraca pozycję x.
- float getY () const
  Zwraca pozycję y.
- float getZ () const
  Zwraca pozycję z.
- void getPara (float &gpx, float &gpy) const
  Zwraca parametry Parallax scrollingu.
- const Sprite * getSprite () const
  Zwraca czcionkę
- unsigned int getW () const
  Zwraca szerokość
- unsigned int getH () const
  Zwraca wysokość
- void getDim (unsigned int &gw, unsigned int &gh) const
  Zwraca wymiary tekstu.
- const char * getText () const
  Zwraca tekst.
- const char * getStr () const
  Zwraca tekst.
- int getAlign () const
  Zwraca wyrównanie tekstu.
- 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.

13.18.1 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

13.18.2 Dokumentacja składowych wyliczanych

13.18.2.1 enum Text::Align

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

13.18.3 Dokumentacja konstruktora i destruktora

13.18.3.1 Text::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)

Konstruktor.

Parametry

id	ID tekstu
x	Współrzędna x
y	Współrzędna x
z	Współrzędna x
px	Parallax scrolling, poziomy
py	Parallax scrolling, pionowy
sSprite	Czcionka
sText	Tekst
w	Szerokość
h	Wysokość (zmienne)
nlSize	Wysokość linii
spSize	Długość spacji
tabSize	Długość tabulatora

13.18.3.2 Text::Text (const Text & *txt*)

Konstruktor kopiujący.

Parametry

txt	Tekst do skopiowania

13.18.4 Dokumentacja funkcji składowych

13.18.4.1 void Text::addStr (const char * *sStr*)

Dopisanie tekstu *sStr*.

Parametry

sStr	Tekst

13.18.4.2 void Text::getDim (unsigned int & *gw*, unsigned int & *gh*) const [inline]

Zwraca wymiary tekstu.

Parametry

out	*gw*	Szerokość
out	*gh*	Wysokość

13.18.4.3 int Text::getLineLen (const char * *str*)

Zwraca długość podanej linii tekstu używając aktualnej czcionki.

Parametry

str	Tekst

13.18.4.4 void Text::getPara (float & *gpx*, float & *gpy*) const [inline]

Zwraca parametry Parallax scrollingu.

Parametry

out	*gpx*	Poziomy
out	*gpy*	Pionowy

13.18.4.5 void Text::getPos (float & *gx*, float & *gy*, float & *gz*) const [inline]

Zwraca pozycję tekstu.

Parametry

out	*gx*	Współrzędna x
out	*gy*	Współrzędna y
out	*gz*	Współrzędna z

13.18.4.6 int Text::getWordLen (const char * *str*)

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

Parametry

str	Tekst

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

Dopisanie tekstu *str*.

Parametry

str	Tekst

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

Dopisanie tekstu *str*.

Parametry

str	Tekst

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

Przypisanie tekstu *str*.

Parametry

str	Tekst

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

Przypisanie tekstu *str*.

Parametry

str	Tekst

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

Ustawienie wyrównania tekstu.

Parametry

sa	Typ wyrównania

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

Ustawienie wymiarów tekstu.

Parametry

sw	Szerokość
sh	Wysokość

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

Przypisanie tekstu *sStr*.

Parametry

sStr	Tekst

13.18.4.14 void Text::update ( )

Aktualizacja tekstu.

Tutaj nieużywane.

Dokumentacja dla tej klasy została wygenerowana z plików:

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

13.19 Dokumentacja struktury Vertex

Prosty vertex/wektor 3D, zawiera podstawowe operacje.

`#include <vertex.h>`

Metody publiczne

- Vertex (float x, float y, float z)
  Konstruktor.
- Vertex ()
  Domyślny konstruktor, zeruje wszystkie zmienne.
- Vertex & operator= (const Vertex &v)
  Przypisanie.
- bool operator== (const Vertex &v) const
  Porównanie dwóch wektorów.
- bool eq2d (const Vertex &v) const
  Porównanie dwóch wektorów z pominięciem współrzędnej z.
- Vertex operator+ (const Vertex &v) const
  Dodanie dwóch wektorów.
- Vertex operator- (const Vertex &v) const
  Odjęcie dwóch wektorów.
- Vertex operator* (float v) const
  Mnożenie wektora przez liczbę
- Vertex operator/ (float v) const
  Dzielenie wektora przez liczbę
- operator std::string ()

Wypisanie wektora.

- 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 dwóch wektorów.

- float len () const

  Długosć wektora.

Atrybuty publiczne

- float x

  Współrzędna x.

- float y

  Współrzędna y.

- float z

  Współrzędna z.

13.19.1 Opis szczegółowy

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

Autor

crm

13.19.2 Dokumentacja konstruktora i destruktora

13.19.2.1 Vertex::Vertex (float *x*, float *y*, float *z*) [inline]

Konstruktor.

Parametry

x	Współrzędna x
y	Współrzędna y
z	Współrzędna z

13.19.3 Dokumentacja funkcji składowych

13.19.3.1 Vertex Vertex::cross (const Vertex & *v*) const [inline]

Iloczyn wektorowy. Z pewnych powodów pomija z. "Taki ficzer".

Parametry

v	Wektor

13.19.3.2 Vertex Vertex::crossz (const Vertex & *v*) const [inline]

Iloczyn wektorowy.

Parametry

v	Wektor

13.19.3.3 float Vertex::dot (const Vertex & *v*) const [inline]

Iloczyn skalarny dwóch wektorów.

Parametry

v	Wektor

13.19.3.4 bool Vertex::eq2d (const Vertex & *v*) const [inline]

Porównanie dwóch wektorów z pominięciem współrzędnej *z*.

Współrzędne są rzutowane na liczbę całkowitą

Parametry

v	Wektor z którym jest porównywany

13.19.3.5 Vertex Vertex::operator∗ (float *v*) const [inline]

Mnożenie wektora przez liczbę

Parametry

v	Liczba

13.19.3.6 Vertex Vertex::operator+ (const Vertex & *v*) const [inline]

Dodanie dwóch wektorów.

Parametry

v	Wektor który jest dodawany

13.19.3.7 Vertex Vertex::operator- (const Vertex & *v*) const [inline]

Odjęcie dwóch wektorów.

Parametry

v	Wektor który jest odejmowany

13.19.3.8 Vertex Vertex::operator/ (float *v*) const [inline]

Dzielenie wektora przez liczbę

Parametry

v	Liczba

13.19.3.9 Vertex& Vertex::operator= (const Vertex & *v*) [inline]

Przypisanie.

Parametry

v	Wektor który jest przypisywany

13.19.3.10 bool Vertex::operator== (const Vertex & *v*) const [inline]

Porównanie dwóch wektorów.

Parametry

v	Wektor z którym jest porównywany

Dokumentacja dla tej struktury została wygenerowana z pliku:

- include/vertex.h

Rozdział 14

Dokumentacja plików

14.1 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

  Głębia 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 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

14.1.1 Opis szczegółowy

14.1.2 Dokumentacja definicji typów

14.1.2.1 typedef std::vector<FightResultRow> FightResult

Wektor wierszy logów z walki

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

Struktura wiersza logów z walki

14.1.2.3 typedef uint32_t uint

Liczba całkowita o rozmiarze 32bitów

14.1.2.4 typedef uint16_t uint16

Liczba całkowita o rozmiarze 16 bitów

14.1.3 Dokumentacja zmiennych

14.1.3.1 const int OCCUPY_MAX = 5

Maksymalny poziom okupowanej planety powodujący jej przejęcie

14.1.3.2 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