

Polecenie:

Stworzyć grę w statki opartą o programowanie obiektowe w języku c++

Analiza Problemu:

W grze możemy wyróżnić takie elementy jak: statek, plansza, gracz i strzał.

Obiektowość można wykorzystać w „budowie” gry. Np. Obiekt statek będzie się składał z mniejszych obiektów „część statku”. Lub obiekt „plansza” może się składać z obiektów „pole”. Oczywiście każdy z graczy będzie miał swoją „planszę” i swoje „statki”.

Polimorfizm można zastosować w określaniu czy dane pole jest trafieniem czy wodą itp. Ja jednak zdecydowałem się na zastosowanie polimorfizmu w sprawdzaniu warunku końca gry przez gracza i komputer.

Opis Interfejsu:

- Całość będzie aplikacją konsolową (rozważam jednak dodanie biblioteki graficznej)
- gra będzie możliwa tylko z „myślącym” botem
- na samym początku pojawi się menu z opcjami 1. Graj, 2. Zasady, 3. Leaderboard.
Przy wyborze opcji pierwszej trzeba będzie podać swój nick
- rozgrywka rozpocznie się od ustawienia statków na planszy. Aplikacja wskaże, które statek należy teraz rozmieścić. Użytkownik zostanie poproszony o podanie odpowiedniej ilości współrzędnych. Plansza będzie się odświeżała z każdym statkiem
- gracz będzie widział planszę ze swoimi statkami i strzałami przeciwnika oraz planszę z polami, w które sam strzelał. Obie będą aktualizowane z każdym strzałem
- każdy będzie dysponował: 1 statkiem 4 masztowym, 2 – 3, 3 – 2 i 4 – 1
- dodatkowo statki muszą mieć szerokość 1 kratki i nie mogą się „dotykać” (nawet na ukos)
- każde z pól będzie oznaczone innym znakiem (woda, trafienie, nasz statek, wrak, pudło/obrys zatopionego statku)
- na końcu zostanie pokazany wynik oparty na ilości potrzebnych strzałów.
- do pliku tekstowego zostanie zapisany początkowe ustawienie gracza, komputera, końcowe plansze obu w raz z zaznaczonymi strzałami przeciwnika, oraz informacja o każdym strzale (współrzędne, trafienie czy pudło, czy zatopienie).
- po każdej grze do osobnego pliku tekstowego będzie zapisywana tabela liderów.
(będzie ona również pobierana przy początku każdej gry.)

Testowanie:

Funkcje odpowiadające za zatapianie, obliczanie strzału przez bota czy umieszczanie statków testowałem ręcznie. Dlatego aby przyspieszyć ten proces napisałem funkcje pomocnicze. Np. gdy chciałem sprawdzić, czy Bot jest „inteligentny” i potrafi zatopić moją flotę w jak najkrótszym czasie, nie potrzebowałem za każdym razem ustawiać statków tak jak by to robił użytkownik przy starcie nowej gry. Dlatego napisałem funkcję. Która tworzy obiekty klasy Ship (statki) i już w konstruktorze podaje potrzebne informacje o umieszczeniu statku. Dzięki temu nie musiałem za każdym razem przechodzić przez proces ustawiania statków, który był trochę czasochłonny.

Wycieki pamięci sprawdziłem przy pomocy specjalnej funkcji, zwracające odpowiednie informacje przy debugowaniu. Po ostatecznych testach usunąłem jednak tę funkcję, aby nie zabierała niepotrzebnie pamięci i czasu. Testy nie wykryły żadnych wycieków.

Wnioski:

Programowanie obiektowe potrafi naprawdę ułatwić i przyspieszyć proces tworzenia gier czy baz danych. Niestety w moim programie nie ma za wiele polimorfizmu. Wydaje mi się on zbędny i tylko negatywnie by wpłynął na optymalizację rozwiązania. Postanowiłem jednak zastosować go w warunku sprawdzania końca gry. Stworzyłem wektor wskaźników na abstrakcyjną klasę rodzica. Następnie dodałem do niego wskaźniki na dzieci. Przy wywoływaniu wirtualnej funkcji sprawdzałem po prostu w pętli: `v[i] -> f()`; Przy większych projektach lub przy bazach danych takie rozwiązanie bardzo upraszcza i przyspiesza proces twórczy.

Jako że był to mój pierwszy projekt związany z programowaniem obiektowym, natrafiłem na kilka problemów. Jednak po czasie uważam, że jest to naprawdę świetne rozwiązanie i pozwala na proste i szybkie rozwiązania wielu problemów.

My Project

Wygenerowano przez Doxygen 1.9.1

1 Indeks hierarchiczny	1
1.1 Hierarchia klas	1
2 Indeks klas	3
2.1 Lista klas	3
3 Dokumentacja klas	5
3.1 Dokumentacja klasy Bot	5
3.2 Dokumentacja klasy Bullet	6
3.3 Dokumentacja klasy Leader	6
3.4 Dokumentacja klasy Leaderboard	7
3.5 Dokumentacja klasy Player	7
3.5.1 Dokumentacja funkcji składowych	8
3.5.1.1 setNick()	8
3.6 Dokumentacja klasy Ship	9
3.7 Dokumentacja klasy ShipPart	9
3.8 Dokumentacja klasy User	10
Indeks	11

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

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

Bullet	6
Leader	6
Leaderboard	7
Ship	9
ShipPart	9
User	10
Bot	5
Player	7

Rozdział 2

Indeks klas

2.1 Lista klas

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

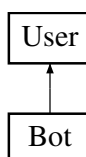
Bot	5
Bullet	6
Leader	6
Leaderboard	7
Player	7
Ship	9
ShipPart	9
User	10

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja klasy Bot

Diagram dziedziczenia dla Bot



Metody publiczne

- void **where** (int size, **Bot** &b)
"Zycie" Komputera.
- void **coords** (**Player** &p)
Funkcja ustawiająca statek komputera.
- bool **shot** (**Player** &p)
Funkcja obliczająca współrzędne kolejnego strzału.
- bool **stop** ()
Funkcja strzelania, zwracająca informacje o tym czy było trafienie czy pudło.
- void **dmg** ()
*Funkcja sprawdzająca warunek końca gry (zwyciestwo Gracza) - funkcja wirtualna, rozniaca sie od funkcji **stop()****Player** wypisywana informacja.*
- int **getHp** ()
Funkcja uszkodzająca statek Bota. (zmniejszająca jego "zycie").
- void **display** (**Player** &p, **Bot** &b)
Funkcja zwracająca zycie Bota.
- void **load1** ()
Funkcja wypisująca w konsoli plansze Bota z zamaskowanymi nietrafionymi statkami.
- void **load2** (**Player** &p)
Funkcja zapisująca do pliku początkowe ustawienie statków Komputera.

Dodatkowe Dziedziczone Składowe

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Piotr/Desktop/PK/Ships/Bot.h
- C:/Users/Piotr/Desktop/PK/Ships/Bot.cpp

3.2 Dokumentacja klasy Bullet

Metody publiczne

- int [getX](#) ()
Zmienna statyczna zliczająca ilość strzałów.
- int [getY](#) ()
Funkcja zwracająca współrzędną X danego strzału.
- void [allHit](#) ()
Funkcja zwracająca współrzędną Y danego strzału.
- int [getHit](#) ()
Funkcja ustawiająca wartość zmiennej `m_hit` na "trafienie zatapiające".
- [Bullet](#) (int x, int y, int hit)
Funkcja zwracająca zmienna `m_hit`.

Statyczne atrybuty publiczne

- static int [ilosc](#) = 0
Zmienna informująca o tym czy dany strzał był trafieniem, pudłem czy strzałem zatapiającym.

Przyjaciele

- ostream & [operator<<](#) (ostream &out, const [Bullet](#) &bullet)
Konstruktor wieloargumentowy obiektu [Bullet](#).

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Piotr/Desktop/PK/Ships/Bullet.h
- C:/Users/Piotr/Desktop/PK/Ships/Bullet.cpp

3.3 Dokumentacja klasy Leader

Metody publiczne

- void [setNick](#) (string s)
Zmienna przechowująca punktację gracza w tabeli.
- void [setScore](#) (int a)
Funkcja odpowiadająca za ustawienie nicku.
- string [getNick](#) ()
Funkcja odpowiadająca za ustawienie wyniku.
- int [getScore](#) ()
Funkcja zwracająca nick gracza.

Przyjaciele

- ostream & `operator<<` (ostream &out, `Leader` &leader)
Funkcja zwracająca punktację gracza.
- bool `operator>` (const `Leader` &l1, const `Leader` &l2)
Przeciążanie operatora wypisu.

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Piotr/Desktop/PK/Ships/Leader.h
- C:/Users/Piotr/Desktop/PK/Ships/Leader.cpp

3.4 Dokumentacja klasy Leaderboard

Metody publiczne

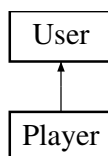
- void `getLeaders` ()
Wektor wskaźników na obiekty klasy lider, będące najlepszymi graczami.
- void `check` (`Player` &p)
Funkcja zwracająca Tabele liderów z pliku.
- void `show` ()
Funkcja sprawdzająca czy gracz wpisuje się do tabeli.
- void `load` ()
Funkcja wypisująca liderów.
- void `sort` ()
Funkcja ładująca tabele liderów do pliku tekstowego.
- void `swap` (`Leader` &l1, `Leader` &l2)
Funkcja sortująca Tabele Liderów (używa przeciązanego operatora porównania)

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Piotr/Desktop/PK/Ships/Leaderboard.h
- C:/Users/Piotr/Desktop/PK/Ships/Leaderboard.cpp

3.5 Dokumentacja klasy Player

Diagram dziedziczenia dla Player



Metody publiczne

- void `shot_coords` ()
Zmienna przechowująca nick gracza.
- bool `shot` (Bot &b)
Funkcja pytająca i pobierająca współrzędne kolejnego strzału.
- void `display` (Bot &bot)
Funkcja strzelająca, zwraca wartość zależną od tego czy było trafienie czy nie.
- void `where` (int size, Bot &b)
Funkcja wypisująca aktualną planszę gracza.
- bool `stop` ()
Funkcja ustawiająca dany statek gracza.
- void `dmg` ()
Funkcja wirtualna sprawdzająca warunek końca gry. Informuje o porażce gracza.
- int `getHp` ()
Funkcja uszkadzająca statek gracza. Wywołowana przy każdym trafieniu przez Bota.
- void `load2` (Bot &b)
Funkcja zwracająca życie gracza.
- void `load1` ()
Funkcja zapisująca do pliku końcową planszę Gracza z naniesionymi strzałami komputera.
- int `score` (Bot &b)
Funkcja zapisująca do pliku początkowe ustawienie statków gracza.
- void `setNick` ()
Funkcja obliczająca wynik gracza.
- int `getScore` ()
- string `getNick` ()
Funkcja zwracająca wynik gracza.

Dodatkowe Dziedziczone Składowe

3.5.1 Dokumentacja funkcji składowych

3.5.1.1 setNick()

```
void Player::setNick ( ) [inline]
```

Funkcja obliczająca wynik gracza.

Funkcja ustawiająca nick gracza.

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Piotr/Desktop/PK/Ships/Player.h
- C:/Users/Piotr/Desktop/PK/Ships/Player.cpp

3.6 Dokumentacja klasy Ship

Metody publiczne

- `vector< shared_ptr< ShipPart > > getParts ()`
Wektor wskaźników na obiekty klasy ShipPart będącymi elementami statku np. Statek 4 masztowy "składa się" z 4 obiektów klasy ShipPart.
- `Ship (int x, int y, int size, bool vertical)`
Funkcja zwracająca wektor części statku.
- `bool sink ()`
Konstruktor obiektu Klasy Ship.
- `int getSize ()`
Funkcja sprawdzająca czy statek zatonał.
- `bool getOrientation ()`
Funkcja zwracająca długość statku.
- `int getX ()`
Funkcja zwracająca orientację statku.
- `int getY ()`
Funkcja zwracająca początkową współrzędną X statku.

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Piotr/Desktop/PK/Ships/Ship.h
- C:/Users/Piotr/Desktop/PK/Ships/Ship.cpp

3.7 Dokumentacja klasy ShipPart

Metody publiczne

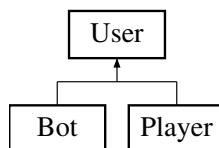
- `ShipPart (int x, int y, bool alive)`
Informacja czy w dany element oddano już strzał.
- `int getX ()`
Konstruktor wieloargumentowy.
- `int getY ()`
Funkcja zwracająca współrzędną X danego elementu.
- `void damage ()`
Funkcja zwracająca współrzędną Y danego elementu.
- `bool isAlive ()`
Funkcja zmieniająca wartość m_alive danego elementu na false.

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Piotr/Desktop/PK/Ships/ShipPart.h
- C:/Users/Piotr/Desktop/PK/Ships/ShipPart.cpp

3.8 Dokumentacja klasy User

Diagram dziedziczenia dla User



Metody publiczne

- void `addBullet` (int x, int y, int hit)
Wektor wskaznikow na obiekty klasy `Bullet` (tylko oddane strzaly).
- void `addShot` (int x, int y, int hit)
Funkcja dodajaca element do wektora `m_bullets` i wywołująca konstruktor `Bullet`.
- vector< `Bullet` > `getShots` ()
Funkcja dodajaca element do wektora `m_shots` i wywołująca konstruktor `Bullet`.
- vector< shared_ptr< `Bullet` > > `getBullets` ()
Funkcja zwracajaca wektor `m_shots`.
- void `addShip` (int x, int y, int size, bool horizontal)
Funkcja zwracajaca wektor `m_bullets`.
- vector< shared_ptr< `Ship` > > `getShips` ()
Funkcja umieszczajaca statek na planszy.
- void `alive` (int i)
Funkcja zwracajaca wektor wskaznikow na obiekty klasy `Ship` (statki).
- bool `collision` (int x, int y, int size, bool orientation)
Funkcja usuwajaca statek z wektora.
- virtual bool `stop` ()=0
Funkcja walidacyjna, pilnujaca aby statki przy ich ustawieniu sie nie stykaly.
- void `shots` ()
Metoda czysto wirtualna odpowiadajaca za sprawdzanie warunku konca gry.

Atrybuty chronione

- int `m_x`
- int `m_y`
Wspolrzedna x strzalu.
- vector< shared_ptr< `Ship` > > `m_ships`
Wsporzledna y strzalu.
- vector< shared_ptr< `Bullet` > > `m_bullets`
Wektor wskaznikow na obiekty klasy statek (statki).
- vector< `Bullet` > `m_shots`
Wektor wskaznikow na obiekty klasy `Bullet` (oddane strzaly + obrys).

Dokumentacja dla tej klasy została wygenerowana z plików:

- C:/Users/Piotr/Desktop/PK/Ships/User.h
- C:/Users/Piotr/Desktop/PK/Ships/User.cpp

Indeks

Bot, [5](#)

Bullet, [6](#)

Leader, [6](#)

Leaderboard, [7](#)

Player, [7](#)

 setNick, [8](#)

setNick

 Player, [8](#)

Ship, [9](#)

ShipPart, [9](#)

User, [10](#)