Klasy w C++ Pierwsza gra

Na ostatnich zajęciach zajmowaliśmy się operację dziedziczenia w klasach. Zadanie polegało na stworzeniu obiektów graficznych na scenie openGL oraz manipulowanie nimi. Podczas działań wykorzystaliśmy polimorfizm w klasach. Polegało to na odnoszeniu się do każdej z figur, z poziomu klasy bazowej „*CFigura*". Na dzisiejszych utrwalimy i wykorzystamy zdobytą do tej pory wiedzę.

Celem zajęć jest stworzenie części gry (kolejna cześć zostanie dopisana na kolejnych zajęciach), w której możliwe jest stoczenie rozgrywki pomiędzy przeciwnikami. Każdy z przeciwników dysponuje armatą której umiejscowienie na planszy 2D jest losowe (pseudolosowe). W momencie startu rozgrywki gracz ustawia siłę uderzenie oraz kąt pod jakim ma być ustawiona armata. Jeśli gracz trafi w przeciwnika wówczas wygrywa, jeśli nie, wtedy kolejka przechodzi na przeciwnika.

Pracę zaczniemy od przygotowania naszego programu. Pierwszą rzeczą jest usunięcie linijki kodu, która po naciśnięciu przycisku „ok" ukrywała okno główne programu:

```
this->ShowWindow(SW_HIDE);// ukrywamy glowne okno
```

Okno bowiem będzie nam teraz potrzebne w celu zadania parametrów strzału. Chcemy też aby nasza scena była większa i żeby była ustawiona w odpowiednim miejscu naszego obszaru pracy, dodajemy zatem:

```
glutInitWindowSize (800, 600);// ustawiamy początkową wielkosc okna
glutInitWindowPosition (200, 100);// poczatkowa pozycja okna (lewy górny róg)
```

Uwaga! Powyższy kod dodajemy przed funkcją: *glutCreateWindow(…)*

Następnie zmieniamy nieco nasz widok przestawiając pozycję kamery (w metodzie *void openGLview:: resize(int width, int height)*):

```
gluLookAt( // ustawienie początkowe kamery
0.0, 0.0, 250.0,0.0, 0.0, 0.0,0.0, 1.0, 0.0);
```

Teraz musimy zastanowić się jak powinny wyglądać obiekty na naszej scenie. Po pierwsze potrzebna jest ruchoma lufa, którą można przybliżyć prostokątem, korpus armaty czyli np. koło, zwieszenie działa – stanowiące przegub np. trójkąt oraz podłoże – najlepiej prostokąt. Poza stanowiskiem armaty potrzebne są nam kule- pociski. Stworzymy zatem nowe dwie klasy: klasę *CArmata* oraz *CPocisk*

Dodajemy zatem klasę armaty, która to będzie agregować obiekty z klasy *CKwadrat, CTrojkat* oraz *Ckolo*:

```
class CArmata
{
public:
        CArmata(void);
        CArmata(int x, int y, int z, double r, double g, double b); //przeładowany konstruktor
        ~CArmata(void);

public:
        void Rysuj(void);//metoda rysujaca armatę
private:
        CKolo * korpus; //korpus armaty
```

```
        CKwadrat * lufa; //lufa czolgu
        CTrojkat * przegub;//zawieszenie lufy
        CKwadrat * podloze; ;//armata musi na czymś leżeć
public:
        double kat; //kat ustawienia lufy
        int pozycja[3]; //pozycja armaty na ekranie
};
```

Oczywiście pamiętamy aby dodać stosowane pliki nagłówkowe.

Mając klasę, możemy dopisać definicję metod (w pliku Armata.cpp):

```
CArmata::CArmata(int x, int y, int z, double r, double g, double b) {


        lufa       = new ...//ustawiamy w pozycji (0, 0) o wymiarze 10x2 i kolorze (r, g, b)
        podloze = new …//ustawiamy w pozycji (0, 0) o wymiarze 30x2 i kolorze (0, 1, 0)
        korpus  = new …//ustawiamy w pozycji (0, 0) o wymiarze 30x2 i kolorze (r, g, b)
        przegub = new …//ustawiamy wierzchołki w pozycji (0, 0), (10,0), oraz (5,4) o kolorze (0, 0, 1)


//ustawiamy pozycje x
//ustawiamy pozycje y
//ustawiamy pozycje z
//oraz ustawiamy kat poczatkowy na 90stopni
}

CArmata::~CArmata(void)
{
        // zwalniamy pamięć = usuwamy nasze obiekty graficzne

}


void CArmata::Rysuj(void) {//metoda rusyjaca armatę

        // wykonujemy rotacje lufy o (0, 0, kat-90) stopni:
        lufa->rotacja[0]=
        ……
        // wykonujemy przesunięcie lufy o (pozycja[0]+15-cos(…) , ... , 0) :
        lufa->translacja[0]=…


        // wykonujemy przesunięcie podłoża o (pozycja[0] , pozycja[1] , pozycja[2]) :
        podloze->translacja[0]=…

        // wykonujemy przesunięcie korpusu  o (pozycja[0]+15 , pozycja[1]+7 , pozycja[2]) :

        // przesuwamy przegub  o (pozycja[0]+10 , pozycja[1]+2 , pozycja[2]) :


        // ostatecznie rysujemy wszystkie obiekty:
        lufa->Rysuj();
        …
}
```

Mając już gotową klasę armaty możemy przetestować jej działanie. Wystarczy dodać obiekt w klasie naszej sceny:

```
#include <gl\glut.h>
#include "figura.h"
#include "Armata.h"

#pragma once

class openGLview
{
public:
//  CFigura * figury[3]; // tablica figur
//  CKwadrat * kwadrat; // kwadrat nr 1
//  CKolo * kolo;
//  CTrojkat * trojkat;
    CArmata * armata;
```

Oraz powołać do życia, jak i usunąć, odpowiednio w konstruktorze i destruktorze:

```
openGLview::openGLview(void)
{

    numer = 0;
        j = 0.0;
        k = 0.1;

 armata= new CArmata(-100,-100,0,1,0,1);

}

openGLview::~openGLview(void)
{

delete armata;
```

Armata musi być wyświetlona więc dodajemy wywołanie metody:

armata->Rysuj();

w metodzie rysuj_figury().

Warto też (dla testu) dodać obsługę nachylenia lufy :

```
void openGLview::keyboard(unsigned char key, int x, int y)
{
 if (key == 27) { exit(0); }// escape key is ascii 27 }
 if (key == 97) {  armata->kat+=10; } //key a
 if (key == 115) { armata->kat-=10; } //key s
//if (key == 100) { figury[numer]->Przesun(0,0,10); } //key d
 //if (key == 122) { figury[numer]->Przesun(-10,0,0); } //key z
```

Teraz kompilujemy i uruchamiamy nasz program. Na ekranie powinna pojawić się armata:

Pierwsza faza pracy zakończona. Teraz zajmiemy się tworzeniem klasy Pocisk. Dodajemy klasę, a w jej ciało wpisujemy metody, jak poniżej:

```
#pragma once
#include "Kolo.h"

class CPocisk
{
public:
	CPocisk(void);
	CPocisk(int x, int y, int z , int radius, double r, double g, double b);//radius - promien pocisku
	~CPocisk(void);

	void Rysuj(); // rysuje pocisk na ekranie
	void trajektoria(int wiatr); // wyznacza pozycję pocisku uwzgledniając wiatr

	CKolo * obudowa; //ksztalt pocisku

	double v0; //predkosc poczatkowa
	double vx; //predkosc w osi x;
	double vy; //predkosc w osi y;
	double kat0; // poczatkowy kat pod jakim porusza sie pocisk
	double px; // pozycja pocisku w osi x
	double py; // pozycja pocisku w osi y
};
```

Do klasy dopisujemy również definicje metod (wszystko w pliku Pocisk.cpp):

```
CPocisk::CPocisk(void)
{
}

CPocisk::~CPocisk(void)
{
		//zwalniamy pamięć-dopisać
}

CPocisk::CPocisk(int x, int y, int z , int radius, double r, double g, double b) {//radius - promien pocisku
//dopisać:
// tworzymy ksztalt pocisku na podstawie klasy CKolo
// ustawia początkową wartość zmiennej x na podstawie parametrow przekazanych do konstruktora
// ustawia poczatkowa wartosc zmiennej y na podstawie parametrow przekazanych do konstruktora

}
void CPocisk::Rysuj() { //metoda rysuje pocisk na ekranie
// rysujemy pocisk – dopisać:
}
void CPocisk::trajektoria(int wiatr){
double dt=1;//krok symulacji lotu pocisku w sekundach
px=px+vx*dt;//obliczamy nowa pozycjê pocisku w osi x - poprzez całkowanie prędkości
py=py+vy*dt;//obliczamy nowa pozycje pocisku w osi y - poprzez całkowanie prędkości

//przepisujemy wspólrzedną x do kola
//przepisujemy wspólrzedną y do kola -dopisać!
```

```
vx+=wiatr/10; //zmienimy prędkość w osi x na podstawie siły wiatru
vy-=0.1;//prędkość w osi y maleje w każdej jednostce czasu o stałą wartość
}
```

Mając już uzupełnione wszystkie metody możemy dodać na naszej scenie dwie armaty (docelowo) oraz dwa pociski:

```
class openGLview
{
public:

    CArmata * armata1;
    CArmata * armata2;

    CPocisk * pocisk1;
    CPocisk * pocisk2;
```

Dla naszych celów, w klasie openGLview, dodajemy też zmienną odpowiedzialną za wiatr:

```
int wiatr; //siła i kierunek wiatru
```

Dodajemy również metodę, która losuje nam wartość wiatru:

```
losujWiatr(); // losowo dobieramy sile wiatru
```

Teraz przechodzimy do pliku .cpp gdzie musimy powołać nasze obiekty w konstruktorze oraz zwolnić pamięć w destruktorze:

```
openGLview::openGLview(void)
{

    numer = 0;
        j = 0.0;
        k = 0.1;

 armata1 = new CArmata(-100,-100,0,1,0,1); //armata 1
 armata2 = new CArmata( 80,-100,0,0,1,1);  //armata 2
 pocisk1 = new CPocisk(-120,-90,0,2,1,0,1);//pocisk 1
 pocisk2 = new CPocisk(120,-90,0,2,0,1,1); //pocisk 2
}

openGLview::~openGLview(void)
{

delete armata1;
delete armata2;
delete pocisk1;
delete pocisk2;

}
```

Dodajemy też definicję metody losującej wartość prędkości wiatru:

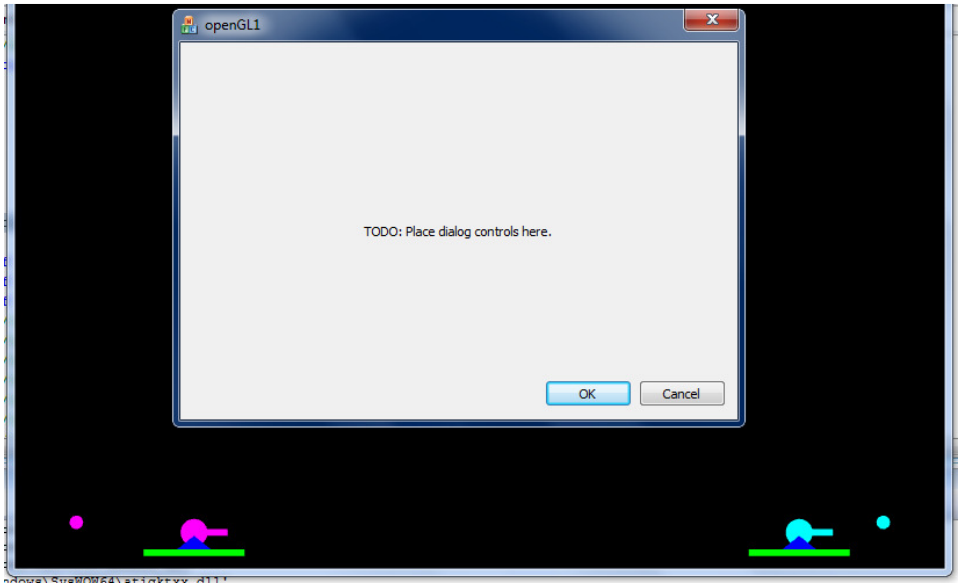
```
void openGLview::losujWiatr() {
int min=-10;
int max=10;
wiatr = min + rand() % (max - min);
}
```

Ostatecznie musimy narysować wszystko na scenie:

armata1->Rysuj();
armata2->Rysuj();
pocisk1->Rysuj();
pocisk2->Rysuj();

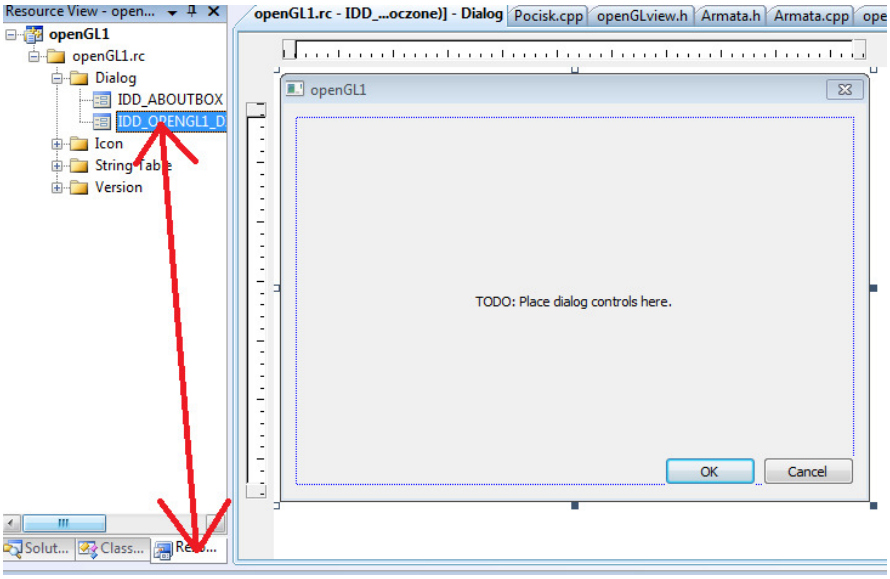
Oczywiście chcąc zmienić kąt pod jaki jest ustawiona armata (przy użyciu klawiatury) musimy odpowiednio zaingerować w metodzie: *void openGLview::keyboard(…)*

Teraz możemy skompilować nasz program a efekt uzyskany na scenie powinien być zbliżony do tego, który jest przedstawiony na rysunku:

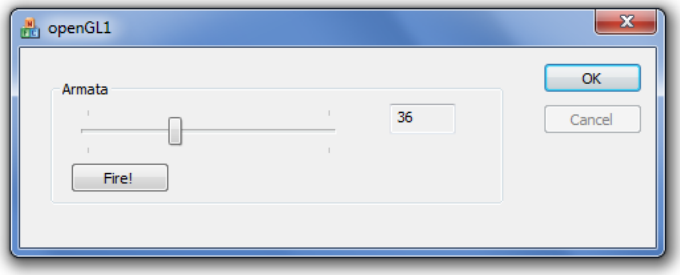


Mamy już dwie armaty, dwa pociski oraz okno dialogowe, które musimy teraz odpowiednio skonfigurować.

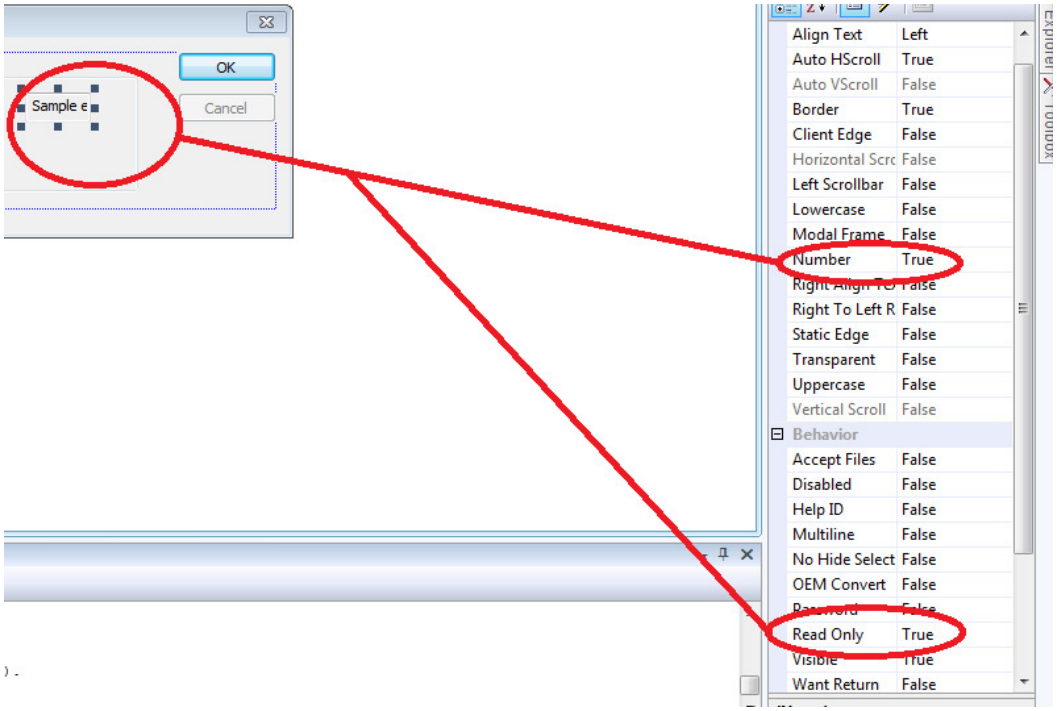
Klikamy na zakładkę resources i wybieramy nasze okno główne:



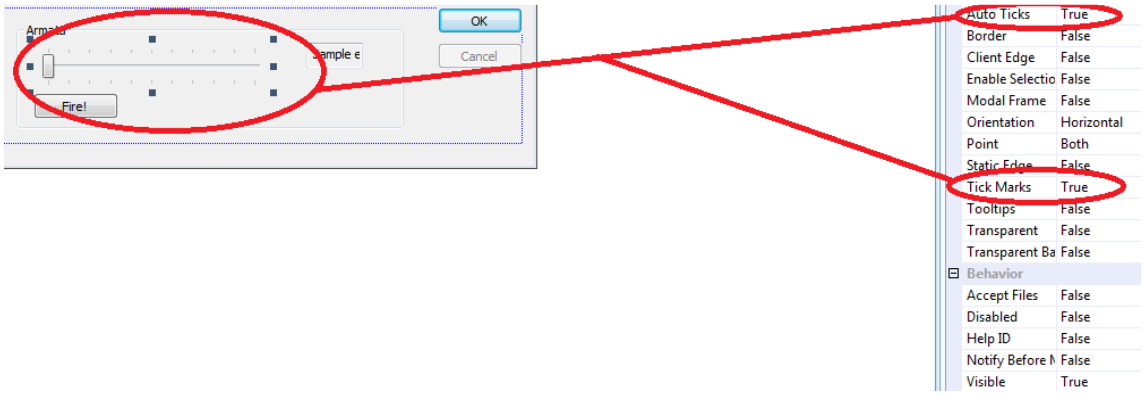
W naszym oknie dodajemy odpowiednie elementy, aby wyglądało ono tak jak poniżej:



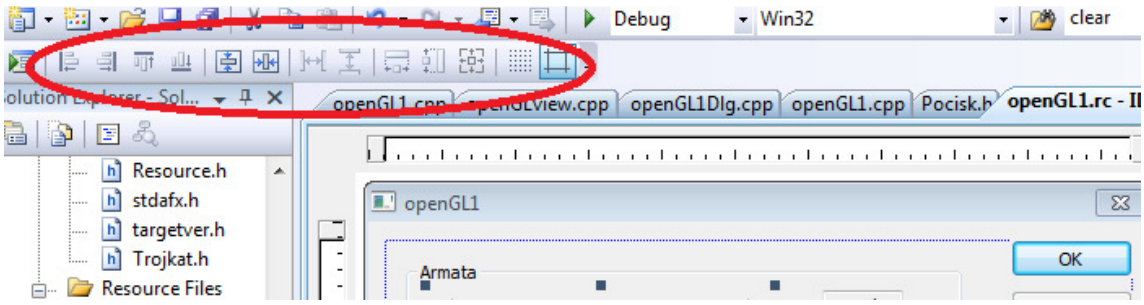
Mając już „slider” oraz Edit box warto zmienić ich parametry. Po pierwsze zmienimy tryb dostępu do naszego Edit box wybierając „true” w zakładce „Read Only” oraz zmieniamy typ wielkości jaką przyjmuje nasz Edit:



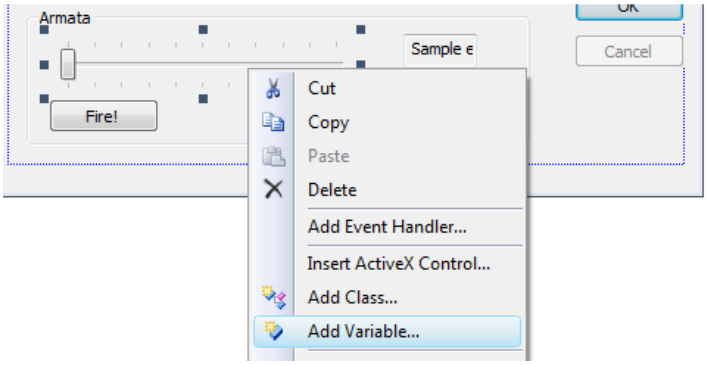
Zmian dokonujemy też w „Sliderze”:



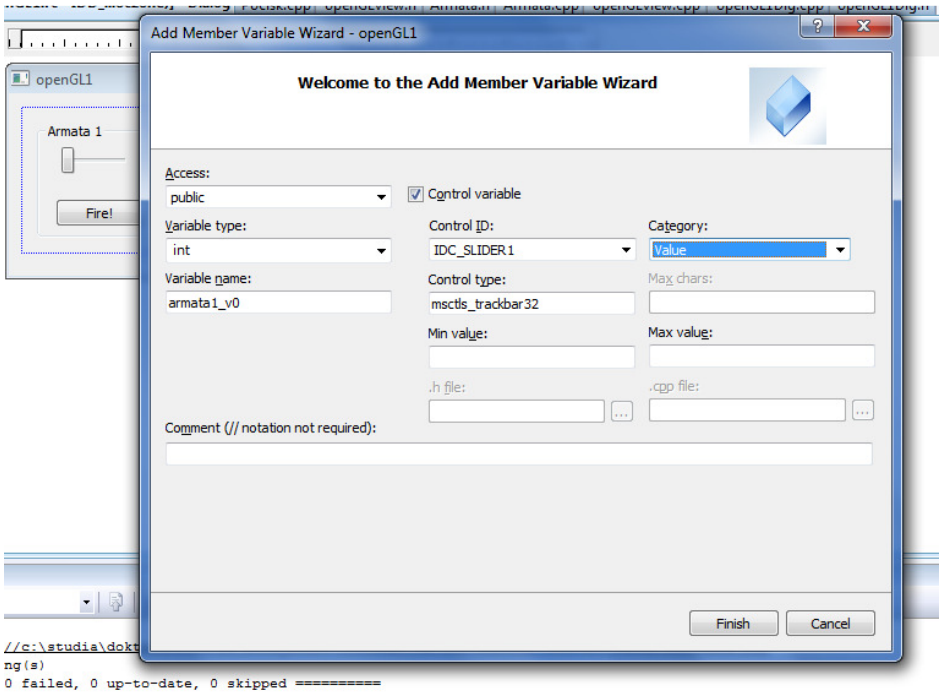
Uwaga! W celu właściwego ułożenia elementów okna dialogowego warto korzystać z funkcji, znajdujących się na pasku narzędzi:



Skończyliśmy prace stylistyczne, teraz przejdziemy do obsługi zmiennych. Najpierw skojarzymy je z oknami. Klikamy prawym przyciskiem na „slider” , a następnie wybieramy „add variable”



Ostatecznie nadajemy jej nazwę oraz typ:



Podobnie postępujemy w przypadku pola „Edit” z tą jednak różnicą, że nazwą zmiennej będzie: „`edit1_v0`”. Teraz klikamy dwukrotnie na przycisk „Fire” i dopisujemy obsługę przycisku:

```
void CopenGL1Dlg::OnBnClickedButton1()
{
        UpdateData(true);        //konieczne przed odczytywaniem wartości
        gl.pocisk1->v0=armata1_v0; // ustawiamy początkową wartość prędkości pocisku
        // TODO: Add your control notification handler code here
}
```

Następnie klikamy dwukrotnie na slider i dopisujemy obsługę Edit box:

```
void CopenGL1Dlg::OnNMCustomdrawSlider1(NMHDR *pNMHDR, LRESULT *pResult)
{
    LPNMCUSTOMDRAW pNMCD = reinterpret_cast<LPNMCUSTOMDRAW>(pNMHDR);

    UpdateData(true);
    edit1_v0=armata1_v0;
    UpdateData(false);

    // TODO: Add your control notification handler code here
    *pResult = 0;
}
```

Mamy już wszystko. Teraz wystarczy skompilować i uruchomić. Pojawi się okno dialogowe, po wybraniu „ok” wyświetlona zostanie cena, na której za tydzień stoczymy bitwę.

Sprawozdanie:

1. Proszę zamieścić kod konstruktora przeładowanego klasy CArmata
2. Proszę zamieścić kod konstruktora przeładowanego klasy CPocisk
3. Dlaczego aktualizacja zmiennej *edit1_v0* odbywa się w metodzie *OnNMCustomdrawSlider1(…)* ?
4. W jakim celu dodajemy UpdataData(true) i UpdateData(false) w metodzie *OnNMCustomdrawSlider1(…)* ? Sprecyzować wpływ powyższych funkcji.
5. Podaj relacje występujące pomiędzy klasami w programie (najlepiej skorzystać z diagramu UML). Proszę nie uwzględniać klasy okna dialogowego.