Programowanie komputerów 4

Projekt „Bullethell”

Krzysztof Para
Informatyka, sem 4, gr 4

Prow. Zajęcia:
Jolanta Kawulok

1. Temat

Napisać Grę Typu „Bullethell” Używając 4 zagadnień przedstawionych na zajęciach

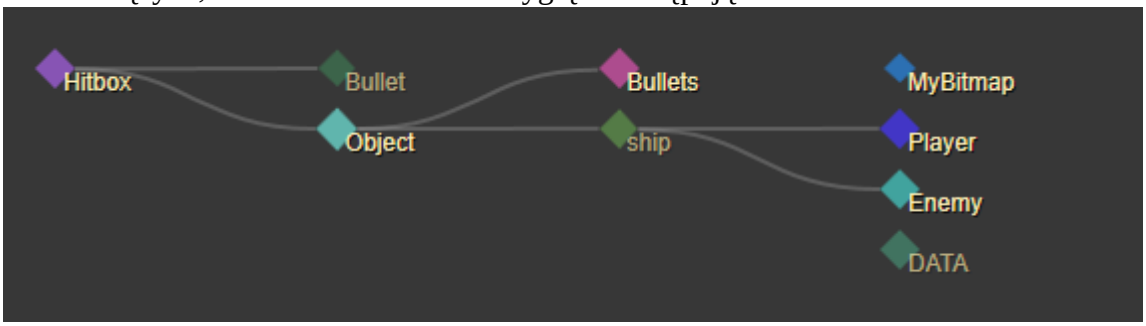
2. Analiza, Projektowanie

1. Algorytmy, struktury danych, ograniczenia specyfikacji

Wszystkie dynamiczne struktury danych oparte zostały na wektorach ze względu na łatwość użycia oraz były one wymagane jako jeden z tematów omawianych na zajęciach.

2. Analiza obiektowa

Jednym z wymogów projektu było utworzyć przynajmniej 6 klas wzajemnie dziedziczących, drzewo dziedziczenia wygląda następująco:



Klasa *Hitbox* odpowiada za przechowywanie informacji o „fizyzcznym” umiejscowieniu obiektu oraz przestrzeni jaką zajmuje.

Klasa *Bullet* zawiera podstawowe informacje o pocisku takie jak obrót, oraz prędkość

Klasa *Object* odpowiada za rysowanie obiektów na ekranie oraz przechowywanie informacji o bitmapach.

Klasa *Bullets* odpowiada za przechowywanie wektora pocisków, oraz zawiera jedną bitmapę która jest grafiką wszystkich pocisków w wektorze, odpowiada również za obliczanie kolizji dla ich wszystkich z jednym innym obiektem (np.: graczem) oraz rysuje wszystkie pociski w wektorze.

Klasa *Ship* przechowuje informacje wspólne dla klas *Player* oraz *Enemy* takie jak ilość życia i prędkość.

Klasa *Player* odpowiada za wszystko co może zrobić gracz czyli, poruszanie się oraz strzelanie.

Klasa Enemy przechowuje funkcję po której porusza się przeciwnik, okres co jaki strzela, miejsca startowe na których przeciwnik się pojawił przydatne do liczenia przemieszczenia z funkcji ruchu

Klasa MyBitmap obudowuje Bitmapy z allegro dla prostszego użytkowania.

Klasa DATA przechowuje wszystkie informacje używane przez wątki.

3. Specyfikacja zewnętrzna

1. Obsługa programu

Gracz jest w stanie poruszać swoim statkiem poprzez naciśnięcie i przytrzymanie klawiszy W A S D oraz strzelać poprzez przytrzymanie spacji.

Gdy użytkownik wyjedzie myszką z okna programu wyświetli się przycisk pauzy i gra się zatrzyma, tak samo w przypadku zmienienia „focusu” okna (alt+tab).

2. Komunikaty

Program wyświetla komunikaty tylko w przypadku błędu typu „Failed to initialize Allegro”, albo gdy ścieżka do pliku nie zostanie znaleziona.

4. Specyfikacja wewnętrzna

1. TemplateVector.h

TemplateVector jest to **Template** obudowywujący **std::vector<T>**, i dodający funkcję optymalnego usuwania ze środka vektora.

2. Hitbox.h

1. Zmienne

radius	Zmienna typu double, przechowje Promień hitboxa kołowego
type	Zmienna typu HitboxType, przechowuje typ hitboxu (punkt, koło, prostokąt)
posX	Zmienna typu unique_ptr<double>, przechowuje współrzędną X środka Hitboxu
posY	Zmienna typu unique_ptr<double>, przechowuje współrzędną Y środka Hitboxu
boundryX	Zmienna typu double, przechowuje wysokość prostokątnego Hitboxu
boundryY	Zmienna typu double, przechowuje szerokośc prostokątnego Hitboxu

2. Funkcje

```
Hitbox(double x, double y);
```

Konstruktor tworzący Hitbox typu punkt.

```
Hitbox(double x, double y, double radius);
```

Konstruktor tworzący Hitbox typu koło

```
Hitbox(double x, double y, double boundryX, double boundryY);
```

Konstruktor tworzący Hitbox typu prostokąt

```
Double GetX();
```

Funkcja zwraca współżędną X na której znajduje się Hitbox

```
Double GetY();
```

Funkcja zwraca współżędną Y na której znajduje się Hitbox

```
Bool CalculateCollision(Hitbox *second);
```

Funkcja zwraca **1** gdy wykryje kolizje z podanym obiektem typu Hitbox

3. Bullet.h

1. Zmienne

rotation	Zmienna typu **double**, przechowuje ilosc radianów o którą pocisk powinien zostać obrócony
speedX	Zmienna typu **double**, przechowuje prędkosc w osi X
speedY	Zmienna typu **double**, przechowuje prędkosc w osi Y
startY	Zmienna typu **double**, przechowuje współrzędną Y na której obiekt został „zrespiony”

2. Funkcje

```
Bullet(double rotation, double posX, double posY, double speedX, double speedY);
```

Konstruktor tworzący obiekt klasy **Bullet** o hitboxie punktowym

```
Bullet(double rotation, double posX, double posY, double speedX,
double speedY, double sizeX);
```

Konstruktor tworzący obiekt klasy **Bullet** o hitboxie kołowym

```
Bullet(double rotation, double posX, double posY, double speedX,
double speedY, double sizeX, double sizeY);
```

Konstruktor tworzący obiekt klasy **Bullet** o hitboxie prostokątnym

```
Double GetRotation();
```

Funkcja zwraca wartośc zmiennej prywatnej **rotation**

```
Int calculate();
```

Funkcja liczy przemieszczenie pocisku na podstawie prędkości w osiach X oraz Y, zwraca 1 gdy pocisk wyleci poza dolną granicę ekranu.

```
Int calculate(double f(double x));
```

Funkcja liczy przemieszczenie pocisku na podstawie funkcji ruchu oraz prędkości w osi X, zwraca 1 gdy pocisk wyleci poza dolną granicę ekranu.

4. Object.h

1. Zmienne

sizeX	Zmienna typu **int**, przechowująca szerokość bitmapy przypisanej do obiektu
sizeY	Zmienna typu **int**, przechowująca wysokość bitmapy przypisanej do obiektu
bitmapa	Wskaźnik na obiekt typu **MyBitmap**, przechowujący grafikę używaną przez obiekt

2. Funkcje

```
Object(const char *filename);
```

Konstruktor tworzący obiekt klasy **Object** bez hitboxa

```
Object(const char *filename, int posX, int posY, int x);
```

Konstruktor tworzący obiekt klasy **Object** o hitboxie kołowym o promieniu **x** na koortynatach **posX**, **posY**

```
Object(const char *filename, int posX, int posY, int x, int y);
```

Konstruktor tworzący obiekt klasy **Object** o hitboxie prostokątnmy o szerokości **x**, i wysokości **y** na koortynatach **posX**, **posY**

```
Void ChangeBitmap(const char *filename);
```

Funkcja zmieniająca załadowaną do pamięci bitmapę

```
ALLEGRO_BITMAP *GetBitmap();
```

Funkcja zwracająca bitmapę przechowywaną w zmiennej **bitmapa**

5. Bullets.h

1. Zmienne

vector	Zmienna typu **TemplateVector<Bullet*>**, przechowywująca wektor pocisków danego typu.
damage	Zmienna typu **int**, przechowywująca obrażenia zadawane gdy pocisk trafi w obiekt klasy **ship**.
f	Wskaźnik na funkcję opisującą ruch danego typu pocisku.

2. Funkcje

```
Bullets(const char *filename);
```

Konstruktor tworzący obiekt klasy Bullets o przypisanej bitmapie **filename.**

```
Bullets(const char *filename, double (*f)(double));
```

Konstruktor tworzący obiekt klasy Bullets o przypisanej bitmapie **filename** oraz funkcji ruchu f.

```
Void AddBullet(double rotation, double posX, double posY, double speedX, double speedY);
```

Funkcja dodająca pocisk o hitboxie punktowym do wektora pocisków.

```
Void AddBullet(double rotation, double posX, double posY, double speedX, double speedY, double sizeX);
```

Funkcja dodająca pocisk o hitboxie kołowym do wektora pocisków.

```
Void AddBullet(double rotation, double posX, double posY, double speedX, double speedY, double sizeX, double sizeY);
```

Funkcja dodająca pocisk o hitboxie prostokątnym do wektora pocisków.

```
Void setDmg(int value);
```

Funkcja ustawiająca wartość obrażeń pocisków.

```
Void CalculateBullets();
```

Funkcja licząca przemieszczenie pocisków bez funkcji ruchu(tylko na podstawie prędkości po osi X oraz Y.

```
Void CalculateBulletsWithFunction();
```

Funkcja licząca przemieszczenie pocisków używając przechowanej funkcji ruchu.

```
Void CalculateBulletsWithFunction(double f2(double x));
```

Funkcja licząca przemieszczenie pocisków używając podanej funkcji ruchu.

```
Int CalcuclateBulletsCollision(Hitbox *hitbox);
```

Funkcja oblicza kolizje pocisków względem zadanego hitboxu a do tego zwraca ich ilość

```
std::vector<Bullet*>*GetBullets();
```

Funkcja zwracająca wektor pocisków przechowywanych w klasie

6. Ship.h

1. Zmienne

rotation	Zmienna typu **double**, przechowuje ilośc radianów o którą pocisk powinien zostać obrócony
speed	Zmienna typu **double**, przechowuje prędkośc z jaką porusza się statek
health	Zmienna typu **int**, przechowuje ilośc punktów życia statku

2. Funkcje

```
ship(const char *filename, int x, int y, int radius, int health);
```

Konstrutor klasy ship ustawiający koordynaty, bitmapę, promień oraz życie z prędkością ustawioną na 5

```
ship(const char*filename, int x, int y, int radius, double s, int health);
```

Konstrutor klasy ship ustawiający koordynaty, bitmapę, promień, życie oraz prędkością

```
Bool CalculateCollisions(Bullets *bullets);
```

Funkcja obliczająca kolizję z pociskami przy kolizji zwraca 1

```
Int GetHealth();
```

Funkcja zwracająca życie statku

```
Double GetX();
```

Funkcja zwracająca pozycję na osi X statku

```
Double GetY();
```

Funkcja zwracająca pozycję na osi Y statku

```
Double GetSpeed();
```

Funkcja zwracająca prędkość statku

7. Player.h

1. Zmienne

up	Zmienna typu **bool**, przechowująca czy przycisk odpowiadający za ruch w górę jest naciśnięty
down	Zmienna typu **bool**, przechowująca czy przycisk odpowiadający za ruch w dół jest naciśnięty
left	Zmienna typu **bool**, przechowująca czy przycisk odpowiadający za ruch w lewo jest naciśnięty
right	Zmienna typu **bool**, przechowująca czy przycisk odpowiadający za ruch w prawo jest naciśnięty
shooting	Zmienna typu **bool**, przechowująca czy przycisk odpowiadający za strzelanie jest naciśnięty

2. Funkcje

```
Player(constchar*filename,intradius,inthealth);
```

Funkcja zwracająca prędkość statku

```
Player(constchar*filename,intradius,ints,inthealth);
```

Funkcja zwracająca prędkość statku

```
voidcalculatePosition();
```

Funkcja zwracająca prędkość statku

```
voidDrawPlayer(intx,inty,introtation=0,intscale=1);
```

Funkcja zwracająca prędkość statku

```
boolCalculateCollisionsForPlayer(Bullets*bullets);
```

Funkcja zwracająca prędkość statku

```
voidPlayerShot(Bullets*bullets,intcounter);
```

Funkcja zwracająca prędkość statku

`Void SetUp(bool var);`	Funkcje odpowiadające za zmianę odpowiednich zmiennych prywatnych w klasie.
`Void SetDown(bool var);`	
`Void SetLeft(bool var);`	
`Void SetRight(bool var);`	
`Void SetShooting(bool var);`	

`Double GetX();`
Funkcja zwracająca pozycję na osi X statku

`Double GetY();`
Funkcja zwracająca pozycję na osi Y statku

8. Enemy.h

1. Zmienne

rotation	Zmienna typu **double**, przechowuje ilosc radianów o którą pocisk powinien zostać obrócony
period	Zmienna typu **int**, przechowuje co ile tickow przeciwnik ma strzelać.
startX, startY	Zmienne typu **double**, przechowują na jakich koordynatach przeciwnik się zespawnował.
f	Wskaźnik na funkcję zwracającą **double** oraz przyjmującą parametr typu **double,** odpowiada za liczenie przemieszczenia przeciwnika

2. Funkcje

```
Enemy(const char *filename, int x, int y, double speed, int period);
```

Konstruktor przypisujący grafikę, pozycję, prędkość, oraz okres strzału przeciwnika

```
Enemy(const char *filename, int x, int y, double speed, double (*f2)
(double x), int period);
```

Konstruktor przypisujący grafikę, pozycję, prędkość, funkcję ruchu oraz okres strzału przeciwnika

```
Void calculatePosition();
```

Funkcja zwracająca pozycję na osi Y statku

```
Void DrawEntity(int x, int y, double rotation = 0, int scale = 1);
```

Funkcja wywołuje funkcję rysującą obiekt.

```
Void EntityShot(Bullets *bullets, int timer);
```

Funkcja dodająca pocisk do **bullets** wg **timer** oraz **period**

```
Double GetHealth();
```

Funkcja zwracająca życie statku

```
Double GetHealthAndBoundarys();
```

Funkcja zwracająca życie statku, jeżeli statek jest poza dolną granicą ekranu to zwraca 0 aby usunąć obiekt klasy.

`Double GetX();`
Funkcja zwracająca pozycję na osi X statku

`Double GetY();`
Funkcja zwracająca pozycję na osi Y statku

9. MyBitmal.h

1. Zmienne

image	Wskaźnik typu ALLEGRO_BITMAP, przechowuje grafikę rastrową

2. Funkcje

```
MyBitmap(const char *filename);
```

Konstruktor ładujący grafikę z danej ścieżki do **image**.

```
Void ChangeBitmap(constchar *filename);
```

Funkcja zmieniająca wczytaną bitmapę

10. DATA.h

1. Zmienne

Playerbullets enemyBullets	Wskaźniki na obiekty typu **bullets**, przechowują wektory pocisków gracza oraz przeciwników.
player	Wskaźnik na obiekty klasy **Player,**
counter	Zmienna typu **int**, przechowuje ilość ticków % 1800 (60 ticków na sekundę)
enemies	Wskaźnik na obiekt typu **TemplateVctor**, przechowuje wszystkich aktywnych przeciwników.
ready	Zmienna typu **bool**, przechowuje informacje czy dane są gotowe (najprostszy semafor)

2. funkcje

```
DATA(Bullets *Playerbullets, Bullets *enemyBullets, Player *player);
```

Konstruktor przypisujący wskaźniki na pociski oraz gracza

```
Void AddEnemy(const char *filename, int x, int y, double speed,
intperiod);
```

Funkcja dodająca przeciwnika o podanej grafice, koordynatach, prędkości oraz częstości strzelania

```
Void AddEnemy(const char *filename, int x, int y, double speed,
double(*f2)(double x), int period);
```

Funkcja dodająca przeciwnika o podanej grafice, koordynatach, prędkości, funkcji ruchu oraz częstości strzelania

`Void DrawEnemies();`
Funkcja rysująca wszystkich przeciwników

`Void calculateEnemies(int timer);`
Funkcja licząca pozycje, kolizje oraz strzelanie przeciwników według timera

`Void ClearDeadEnemies();`
Funkcja czyszcząca martwych bądź zbyt oddalonych przeciwników

5. Testowanie

Projekt został przetestowany pod względem przecieków pamięci używając `_CRTDBG_MAP_ALLOC`