Uniwersytet Rzeszowski

Podstawy C++

DOKUMENTACJA PROJEKTU

Arkadiusz Kornafel | 101392
Piotr Bobusia | 101272

Spis treści:

1. Tematyka i założenia aplikacji

Projekt ma na celu stworzenie gry typu Space Shooter w bibliotece SFML. Główne założenia projektu:

- Projekt wykonany w SFML
- Gra będzie zawierać co najmniej 3 typy przeciwników
- Gra będzie wyświetlać pasek zdrowia gracza
- Gra będzie wyświetlać zdobyte punkty gracza

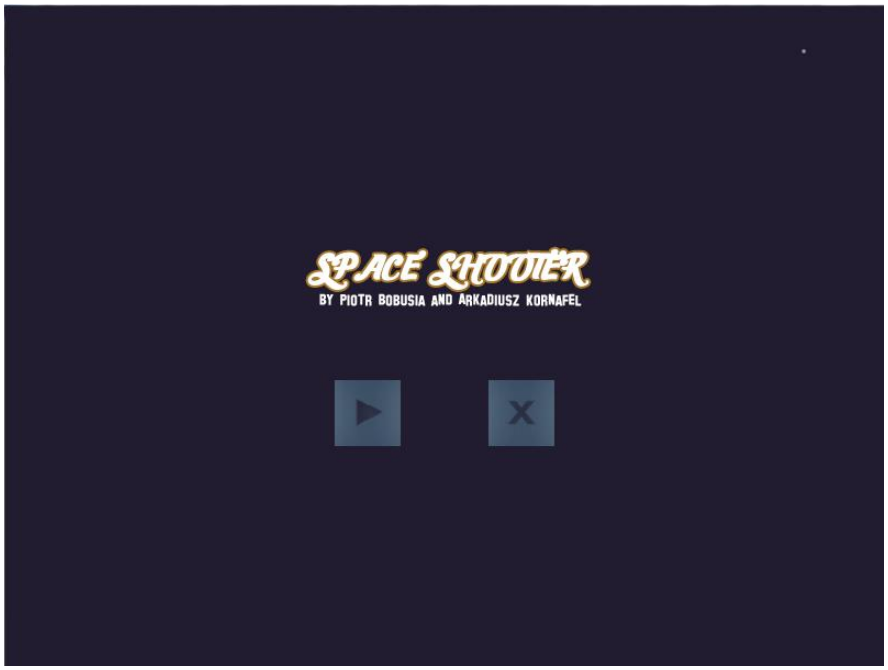
2. Opis gry

Menu

Menu gry składa się z logo projektu oraz dwóch przycisków.

Pierwszy przycisk od lewej strony to przycisk rozpoczęcia rozgrywki.

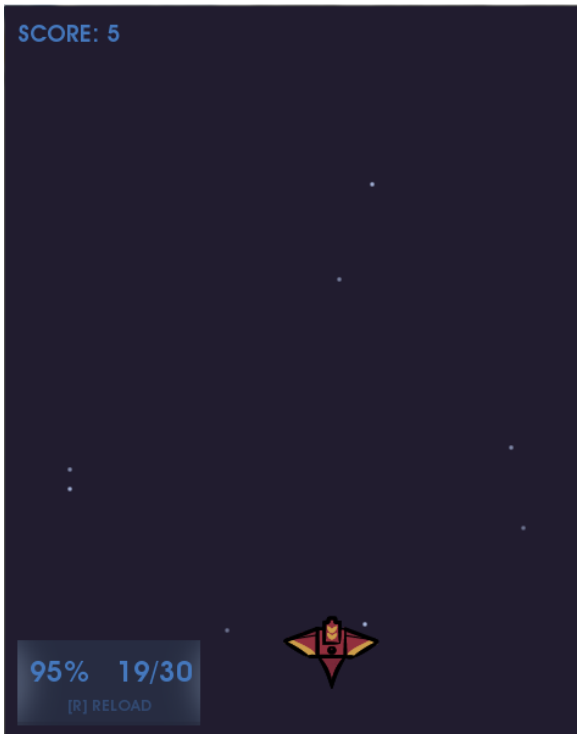
Drugim przyciskiem jest przycisk, który pozwala zakończyć działania naszej aplikacji.



W tle menu działa również skrypt odpowiedzialny za generowanie gwiazd w przestrzeni. Jedną z nich widać na powyższym screenie w prawym górnym rogu ekranu.

Interfejs

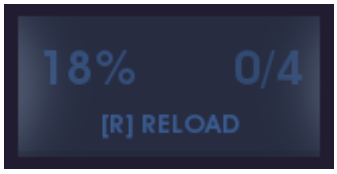
Interfejs gry składa się z części przedstawiającej nam stan naszego statku kosmicznego (lewy dolny róg ekranu) oraz z licznika naszych punktów, które zdobywamy niszcząc wrogie statki (lewy górny róg ekranu).



Procentowy wskaźnik po lewej stronie oznacza poziom naszego pancerza i ogólnego stanu naszego statku. Gdy wskaźnik pancerza ma mniej niż 20% zaczyna mrugać.

Po prawej stronie wyświetlany jest stan magazynku, który zostaje przyciemniony, gdy magazynek jest pusty.

Na dole jest informacja na temat przeładowania magazynku. Gdy posiadamy amunicje jest on wygaszony, a gdy amunicja się skończy, informacja mruga informacyjnie.



Sterowanie

Myszka – Ruch po osi X

[Spacja] – Strzał

[R] – Przeładowanie magazynka

Przeciwnicy

W grze występują 4 rodzaje przeciwników o 5 różnych rodzajach AI.

Przeciwnik 1: Jest to początkowy przeciwnik, który z góry ma ustaloną pozycję, którą ma przyjmować na mapie.

Przeciwnik 2: Jest to drugi przeciwnik, który ma za zadanie podlatywać do naszego statku i za nim podążać.

Przeciwnik 3: Ma za zadanie wykonywać „podlot” jak przeciwnik nr 2, jednak nie podąża on za naszym statkiem. Posiada on nieco więcej punktów pancerza od innych przeciwników. Wycofuje się, gdy posiada mniej pancerza.

Przeciwnik 4: Posiada on 2 typy AI. Pierwszy z nich wycofuje się w górną część mapy i podąża za naszym statkiem. Drugi typ zachowuje się tak samo jednak, gdy posiada mało punktów pancerza, zaczyna on uciekać.

3. Opis działania

Tworzenie okna aplikacji

Konfiguracją okna SFML zajmuje się funkcja configureWindow().

```
void configureWindow(){ ///Funkcja do konfiguracji okna SFML
    W.create(VideoMode(800,600,32),"Projekt SFML");
    W.setActive(true);
    W.setKeyRepeatEnabled(true);
    W.setFramerateLimit(60);
    //W.setVerticalSyncEnabled(true);
    W.setPosition(Vector2i(10,10));
}
```

Klasy

```
///Klasy
class Airplane;
class HeroAirplane;
class EnemyAirplane;
class Bullet;
class EnemyBullet;
class HeroBullet;
class Interface;
class Level;
void collisionDetector(HeroBullet**, int, EnemyAirplane**, int);
void collisionDetectorHero(EnemyBullet**, int, HeroAirplane*, int);

class SkyBg;
class Star;
```

Airplane – klasa, z której dziedziczą klasy HeroAirplane oraz EnemyAirplane. Zawiera niezbędne elementy dla każdego tworzonego statku.

```
class Airplane{      ///Klasa statków do dziedziczenia
public:
    Texture texture;     //tekstura statku
    Sprite sprite;       //sprite statku
    int posX, posY;      //koordynaty aktualnie
    int docX, docY;      //koordynaty docelowe
    int posXBulletAdd;

    ///METODY
    Airplane( int , int );
    Sprite getSprite();
    int getX();
    int getY();
    int getX_end();
    int getY_end();
};
```

HeroAirplane – klasa naszego statku. Zawiera wszystkie niezbędne elementy do sterowania, obliczania i rysowania obiektu.

```
class HeroAirplane : public Airplane {      ///Klasa statku którym sterujemy
private:
    int hp;      //punkty zycia
    int ammo, maxAmmo;    //ilocs amunicji
    int bulletTimer , weaponSpeed;  //Ogranicznik strzalów, ¿eby nie strzelac za szybko
    bool readyToShoot;  //Czy jest gotowy do strzalu
public:
    bool active;
    int score; //zdobyte punkty

    HeroAirplane(int, int);
    void tick();
    void keyboardEvent(Event);      //obs³uga klawiatury i myszki

    HeroBullet* bulletArray[100];    //Tablica pocisków
    void drawBullets();      //Funkcja rysowania posicków na ekranie
    bool getHit(int);

    void checkWeaponTick(); //tick do sprawdzania mo¿liwœci strzalu

    void setMaxAmmo(int);

    void updateMaxAmmo(int);

    int getHp();
    int getAmmo();
    int getMaxAmmo();

    void resetBullets();
    void resetStats();
};
```

bulletArray to tablica pocisków, które możemy wystrzelić naszym statkiem.

EnemyAirplane – klasa wrogich statków. Zawiera wszystkie niezbędne elementy do sterowania, obliczania i rysowania obiektów.

```
class EnemyAirplane : public Airplane {     ///Klasa statków przeciwnika
private:
    int hp, maxHp; //punkty zdrowia
    int speed;     //predkosc poruszania sie
    int aiLevel; //poziom ai (1-mieso armatnie ("ujadacz")  |  2-podlot_zwykly  |  3-mało HP to w nogi)
    int bulletTimer , weaponSpeed;  //Ogranicznik strzalów, ¿eby nie strzelac za szybko
    bool goToPosition;
    bool readyToShoot;
public:
    bool active;
    bool dead;
    EnemyAirplane(int, int, int, int); //(level, score, docX, docY)
    void aiTick(HeroAirplane);  //tick AI wroga
    void tick();    //reszta obliczen i ustalen np. sprite.setPosition
    void spawn(int, int);   //zmiania BOOL ACTIVE -> TRUE  +  wskazuje koordynaty do spawnu
    bool getHit(int); //zwraca true jesli konczy sie to smiercia
    int getHp();
    int score; //iule punktow wypada z przeciwnika
    EnemyBullet* bulletArray[100];
```

```
    void checkWeaponTick(); //tick do sprawdzania mo¿liwoœci strzalu
    void drawBullets();     //Funkcja rysowania posicków na ekranie
private:    ///Zmienne dla AI
    bool podlot; //qktywne gdy przeciwnik podlatuje
    bool podlot_zwykly; //podlatuje w dol
    bool podlot_atakujacy; //podlatuje naprzeciw gracza
    bool powrot;    //aktywne gdy wraca na pozycje docX docY
    bool wycofanie; //cofa sie do gory (mozliwie za swoich kolegow)
    bool unikanie;  //przeciwnik rusza sie na boki
    bool unikanie_prawo; //unika w prawo
    bool unikanie_lewo; //unika w lewo
    bool kotwiczenie; //kotwica na graczu - statek porusza sie za statkiem gracza na osi X
    int podlot_Y, podlot_rand;
    int wycofanie_Y;
    bool strzelaj;
};
```

bulletArray to tablica pocisków, które mogą zostać wystrzelone przez enemyAirplane.

Bullet – klasa pocisku służąca do dziedziczenia dla heroBullet oraz enemyBullet.

```
class Bullet {      ///Klasa pocisku
public:
    bool active;    //Do sprawdzania czy pocisk jest aktywnym uczestnikiem rozgrywk
    int power;      //ile zadaje obrazen

    Texture texture;
    Sprite sprite;

    int posX, posY; //koordynaty
    int speed;  //Szybkosc lotu pocisku
    Bullet(int, int, int);

    Sprite getSprite();
    int getX();
    int getY();
    int bulletWidth, bulletHeight;

    void setBullet(int, int, int);  //Funkcja do ustawiania i aktywacji pocisku
};

class EnemyBullet : public Bullet { ///Klasa pocisków wystrzelonych przez wrogów
public:
    void setTexture(String);    //Funkcja do ustawiania tekstury (ró¿ni wrogowie to ró¿ne posicki)
    void tick();
    EnemyBullet(int, int, int, String);
};

class HeroBullet : public Bullet {  ///Klasa pocisków wystrzelonych przez nasza postac
public:
    HeroBullet(int, int, int);
    void tick();
};
```

Interface – klasa interfejsu, umożliwiająca jego obliczanie i wyświetlanie.

```
class Interface {    ///Klasa interfejsu
private:
    bool podjasnij, przyciemnij;
    bool podjasnijHp, przyciemnijHp;
    int reloadAlpha, ammoAlpha, hpAlpha;
    Texture texture;
    Sprite sprite;

public:
    Interface();
    Font font1, font2, font3;
    void showInterface(HeroAirplane*);
};
```

SkyBg/Star – SkyBg to klasa, która służy do generowania tła kosmosu i gwiazd na nim. Zawiera tablice z obiektami typu Star. Każda gwiazda posiada własny tick() służący do obliczania jej ruchu.

```
class SkyBg {    ///Klasa tła kosmosu
private:
    int timeToStar, timer;
    void throwStar();
public:
    SkyBg();
    Star* starArray[50];
    void showStars();
    void tick();
};
```

```
class Star {
private:
    Texture texture;
    Sprite sprite;

    int posX, posY;
    int speedTimer, timer;
public:
    Star(int);
    bool active;
    void spawn(int,int);
    Sprite getSprite();
    void tick();
    int getSpeedTimer(){return speedTimer;};
};
```

Funkcje

Oto spis funkcji menu:

```
///Niezbedne rzeczy do menu
Texture playTexture, exitTexture, logoTexture;    //Przyciski play i exit
Sprite playSprite, exitSprite, logoSprite;
Texture playTextureActive, exitTextureActive;    //Przyciski play i exit
Sprite playSpriteActive, exitSpriteActive;
bool activeMenu = true; //Czy menu jest aktywne??
void showMenu();    //Funkcja sluzy do wyswietlania menu
void loadMenu();    //Funkcja laduje textury i sprite do menu
int yBtn = 340; int yLogo = 200;
int xPlayBtn = 300; int xLogo = 250;
int xExitBtn = 440;
void menuKeyboardEvent(Event);  //funkcja do obslugi myszy/przyciskow w menu
bool activeBtnPlay = false; //czy myszka najechala na przycisk Play
bool activeBtnExit = false; //czy myszka najechala na przycisk Exit

void gameReset();    //Reset stanu gry
```

Oto spis funkcji:

```
///Funkcje
void configureWindow();
void loadTexturesSprites();
void loadEnemy();
void enemySpawner(); int nextEnemyTime = 120, enemyTimer = 0, maxEnemyInWave  = 4; ///nextEnemyTime - ilosc cykli do spawny przeciwnika
int numberOfEnemy = 0; ///ilosc zespawnowanych wrogow                              ///enemyTimer - zwykly timer odliczajacy do spawnu
int actualEnemyIndex = -1;                                                                                    ///maxEnemyInWave - ile wrogow moze byc

bool firstBlood = false;        ///do sprawdzania czy pierwszy enemy zostal pokonany
bool tutEnemy1 = true;      ///BOOLe fal tutorialowych
bool tutEnemy2 = false;
bool tutEnemy3 = false;
```

Pętla gry

Pętla gry składa się z sekcji ticków klas oraz z sekcji rysowania:

```
    /**=======================*
         SEKCJA TICKÓW GRY
      *=======================*/
    if(!activeMenu){
         enemySpawner();

         heroAirplane->tick();
         heroAirplane->checkWeaponTick();

         for(int i = 0; i<100; i++)
         if(airplaneEnemyArray[i] -> active){ //dzialaj jesli wrog jest aktywny
             airplaneEnemyArray[i] -> aiTick(*heroAirplane);
             airplaneEnemyArray[i] -> tick();
             airplaneEnemyArray[i] -> checkWeaponTick();
             collisionDetectorHero(airplaneEnemyArray[i]->bulletArray, 100, heroAirplane, 1);
         }


         collisionDetector(heroAirplane->bulletArray, 100, airplaneEnemyArray, 100);


    }
    skyBg.tick();


/**=======================*
      SEKCJA EVENTOWA
  *=======================*/

while(W.pollEvent(e)){
    if(e.type==Event::Closed || (Keyboard::isKeyPressed(Keyboard::Escape) )) W.close();

    if(activeMenu) menuKeyboardEvent(e);
    else {
        heroAirplane->keyboardEvent(e);
    }
}



/**====================*
     SEKCJA RYSOWANIA
  *====================*/
W.clear(Color(33,28,47));
for(int i = 0; i<50; i++) if(skyBg.starArray[i]->active) W.draw(skyBg.starArray[i]->getSprite());

if(!activeMenu){
    W.draw(heroAirplane->getSprite());
    heroAirplane->drawBullets();

    for(int i = 0; i<100; i++)if(airplaneEnemyArray[i] -> active) W.draw(airplaneEnemyArray[i] -> getSprite());
    for(int i = 0; i<100; i++) airplaneEnemyArray[i] -> drawBullets();
    //W.draw(sp2); ///TEST
    //W.draw(sp3); ///TEST
    //W.draw(sp4); ///TEST
    //W.draw(sp5); ///TEST
    interface->showInterface(heroAirplane);
} else showMenu();
W.display();
```