Wrocław 18.06.2018 r

Raport:

Space Invaders

Wykonał:

Jakub Wylandowski 230190

Termin zajęć:

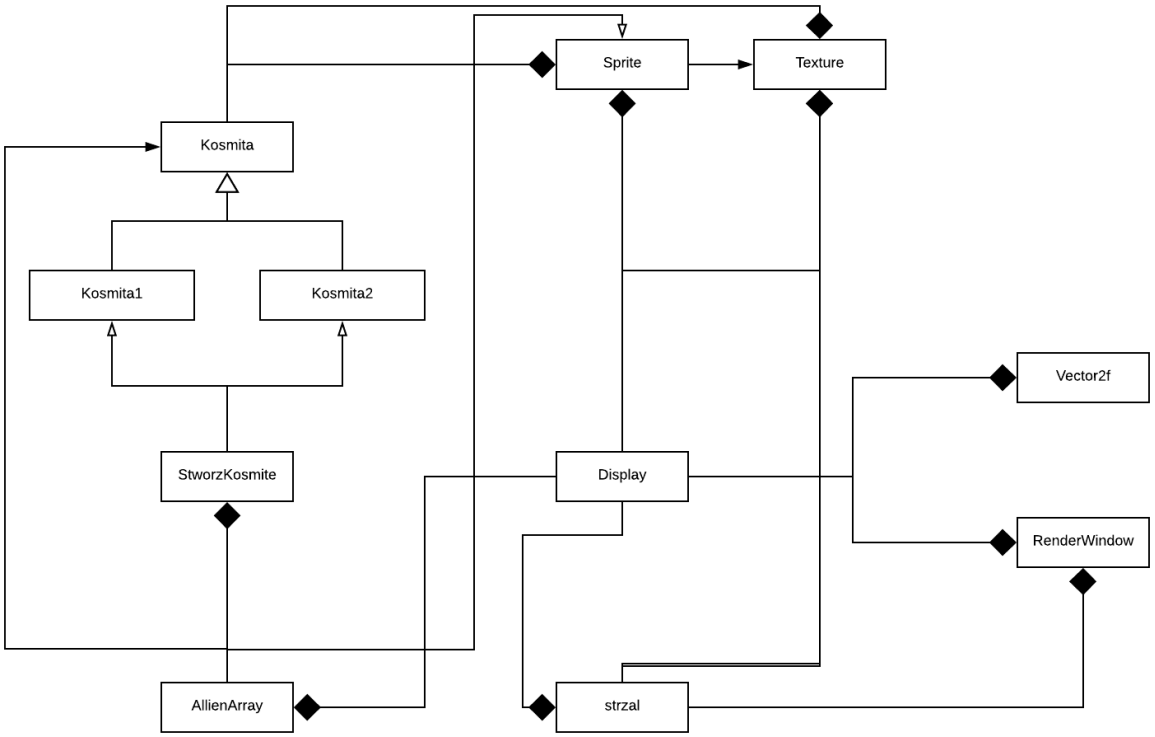
Poniedziałek 15:15 TN

1. Założenia

Założeniem było wykonanie gry na wzór klasycznego Space Invaders znanego z automatów oraz starszych konsol. Gra została napisana w języku C++ w IDE Microsoft Visual Studio 2017, przy użyciu bibliotek graficznych SFML w wersji 2.5.0. Użyte wzorce projektowe to Singleton (klasa Display) oraz metodę wytwórczą (StworzKosmite).

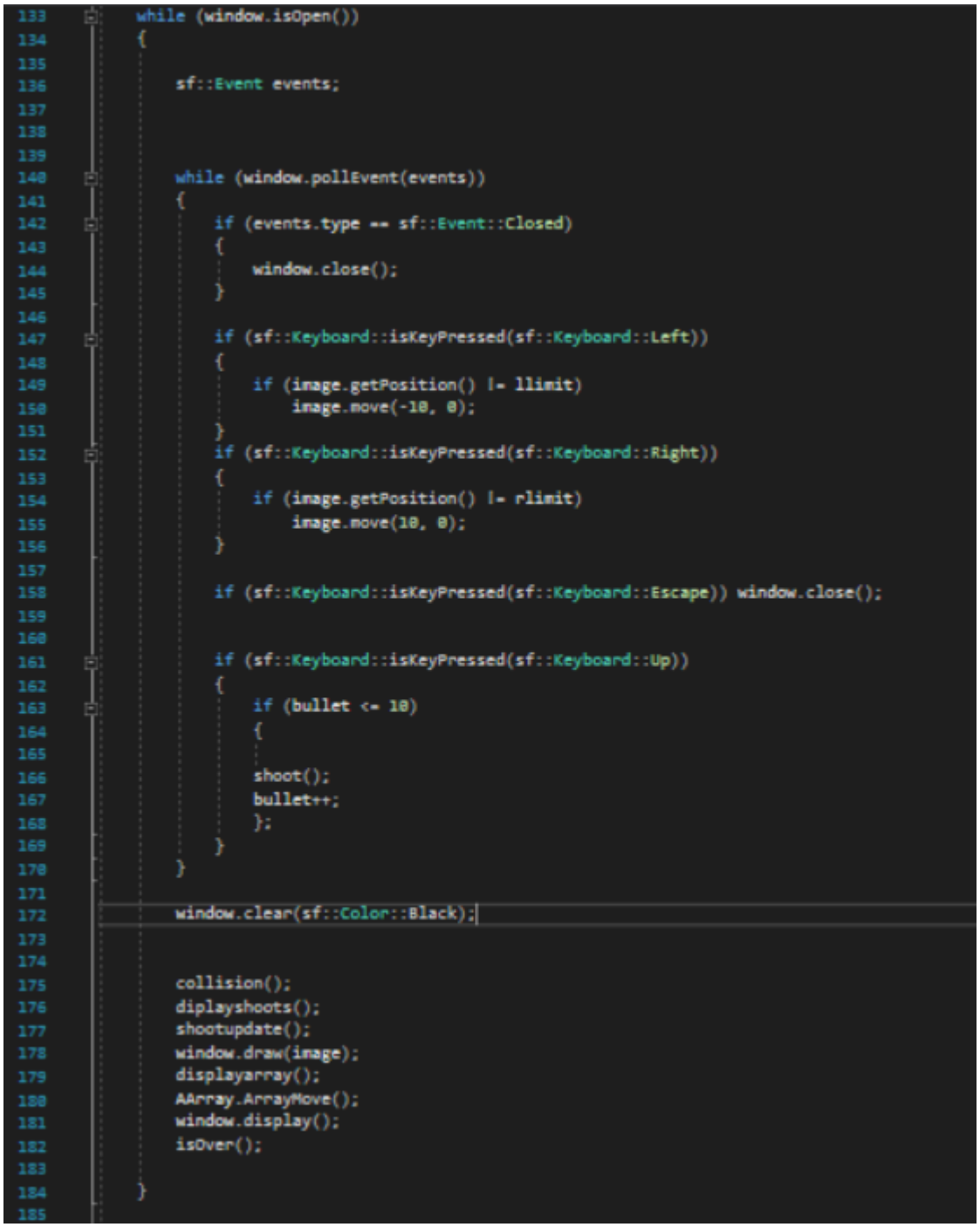
2. Wykonanie

Na wstępie załączę diagram UML klas dla projektu.



Klasy Sprite, Texture, RenderWindow oraz Vector2f są dostarczane z biblioteki SFML i umożliwiają otwarcie okna oraz rysowanie w nim elementów.

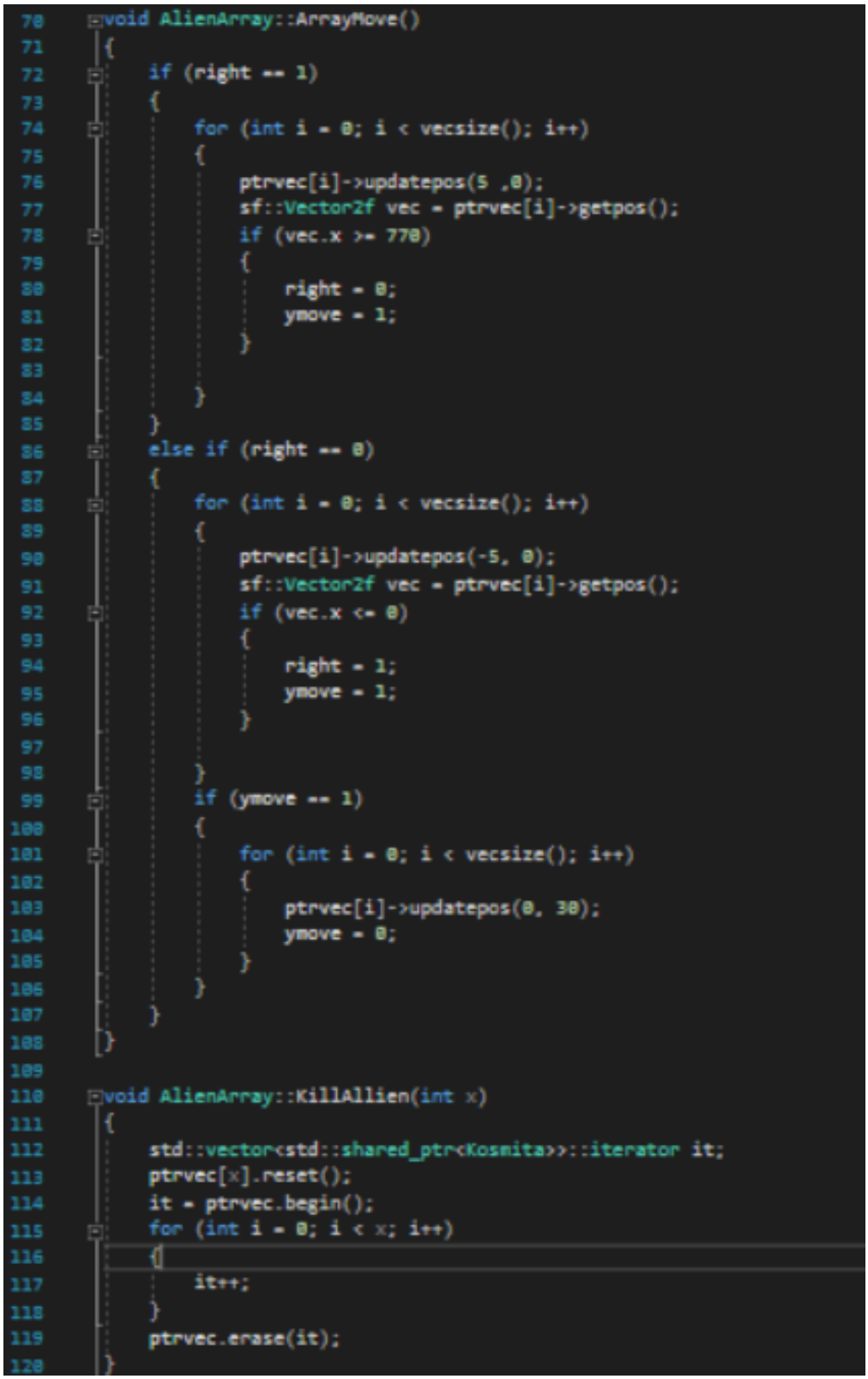
Klasa Display jest główną klasą, otwierającą okienko gry oraz zawierającą pętlę rysującą i detekcje kolizji. Kontroluje również ona powstawanie strzałów oraz ich usuwanie.



```
while (window.isOpen())
{

    sf::Event events;



    while (window.pollEvent(events))
    {
        if (events.type == sf::Event::Closed)
        {
            window.close();
        }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
        {
            if (image.getPosition() != llimit)
                image.move(-10, 0);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
        {
            if (image.getPosition() != rlimit)
                image.move(10, 0);
        }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) window.close();


        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
        {
            if (bullet <= 10)
            {

            shoot();
            bullet++;
            };
        }
    }

    window.clear(sf::Color::Black);


    collision();
    displayshoots();
    shootupdate();
    window.draw(image);
    displayarray();
    AArray.ArrayMove();
    window.display();
    isOver();

}
```

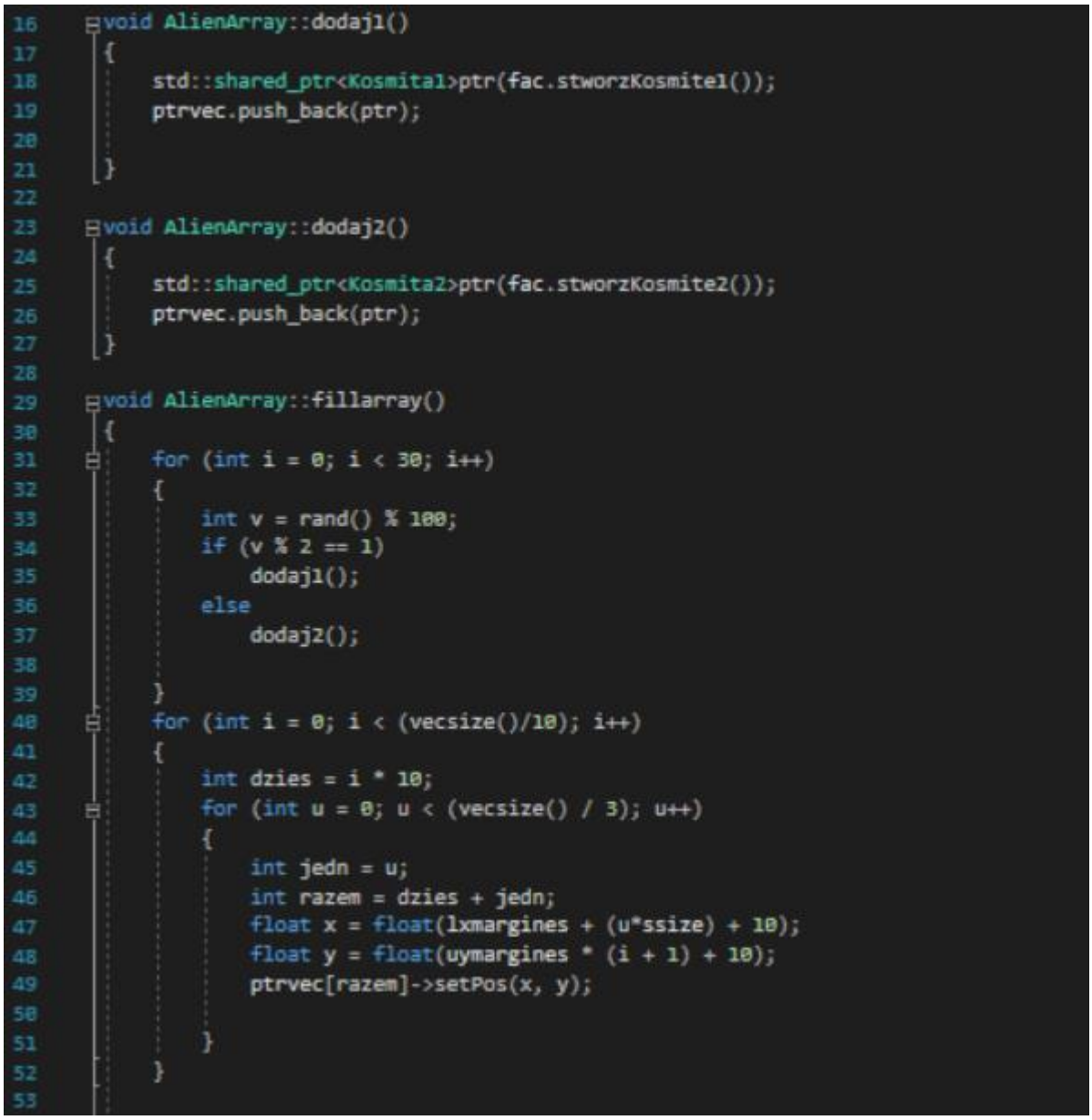
Pętla rysująca.

Klasa AllienArray, wchodząca w skład klasy Display i odpowiada za powstanie szeregów kosmitów, kontroluje ona ich ruch i usuwa trafionych kosmitów. Kosmici są tu zawarci jako wektor wskaźników na konkretne egzemplarze kosmitów, które to są kasowane po ewentualnym trafieniu.



```
70  void AlienArray::ArrayMove()
71  {
72      if (right == 1)
73      {
74          for (int i = 0; i < vecsize(); i++)
75          {
76              ptrvec[i]->updatepos(5 ,0);
77              sf::Vector2f vec = ptrvec[i]->getpos();
78              if (vec.x >= 770)
79              {
80                  right = 0;
81                  ymove = 1;
82              }
83
84          }
85      }
86      else if (right == 0)
87      {
88          for (int i = 0; i < vecsize(); i++)
89          {
90              ptrvec[i]->updatepos(-5, 0);
91              sf::Vector2f vec = ptrvec[i]->getpos();
92              if (vec.x <= 0)
93              {
94                  right = 1;
95                  ymove = 1;
96              }
97
98          }
99          if (ymove == 1)
100         {
101             for (int i = 0; i < vecsize(); i++)
102             {
103                 ptrvec[i]->updatepos(0, 30);
104                 ymove = 0;
105             }
106         }
107     }
108 }
109
110 void AlienArray::KillAllien(int x)
111 {
112     std::vector<std::shared_ptr<Kosmita>>::iterator it;
113     ptrvec[x].reset();
114     it = ptrvec.begin();
115     for (int i = 0; i < x; i++)
116     {
117         it++;
118     }
119     ptrvec.erase(it);
120 }
```

Kod odpowiedzialny za poruszanie się kosmitów, oraz ich kasowanie po trafieniu.



```
16  void AlienArray::dodaj1()
17  {
18      std::shared_ptr<Kosmita1>ptr(fac.stworzKosmite1());
19      ptrvec.push_back(ptr);
20
21  }
22
23  void AlienArray::dodaj2()
24  {
25      std::shared_ptr<Kosmita2>ptr(fac.stworzKosmite2());
26      ptrvec.push_back(ptr);
27  }
28
29  void AlienArray::fillarray()
30  {
31      for (int i = 0; i < 30; i++)
32      {
33          int v = rand() % 100;
34          if (v % 2 == 1)
35              dodaj1();
36          else
37              dodaj2();
38
39      }
40      for (int i = 0; i < (vecsize()/10); i++)
41      {
42          int dzies = i * 10;
43          for (int u = 0; u < (vecsize() / 3); u++)
44          {
45              int jedn = u;
46              int razem = dzies + jedn;
47              float x = float(lxmargines + (u*ssize) + 10);
48              float y = float(uymargines * (i + 1) + 10);
49              ptrvec[razem]->setPos(x, y);
50
51          }
52      }
53
```

Zapełnianie szeregów kosmitów.

Kolejną klasą jest Kosmita, będący interfejsem dla Kosmita1 i Kosmita2. Klasa StworzKosmite natomiast jest klasą fabrykującą kolejne egzemplarze kosmitów, w zależności od potrzeb Kosmite1 lub Kosmite2 (w przypadku gry wybór co do proporcji Kosmitów1 i Kosmitów2 jest losowy).

```
class Kosmita
{
private:
public:
    virtual void setPos(float,float) = 0;
    virtual sf::Sprite& ref() = 0;
    virtual ~Kosmita() {};
    virtual void updatepos(float, float)=0;
    virtual const sf::Vector2f& getpos()=0;
};

class Kosmita1 :public Kosmita
{
private:
    int hp;
    sf::Sprite sprt;
    sf::Texture txt;
public:
    Kosmita1();
    virtual void setPos(float, float);
    virtual sf::Sprite& ref()override;
    void updatepos(float , float )override;
    const sf::Vector2f& getpos();

};

class Kosmita2 :public Kosmita
{
private:
    int hp;
    sf::Sprite sprt;
    sf::Texture txt;
public:
    Kosmita2();
    virtual void setPos(float, float);
    virtual sf::Sprite& ref()override;
    void updatepos(float, float)override;
    const sf::Vector2f& getpos();
```

Kosmita jako interface.

```
class stworzKosmite
{
private:
    std::vector<Kosmita*>regvec;
    void reg(Kosmita*);
public:
    Kosmita1* stworzKosmite1();
    Kosmita2* stworzKosmite2();

};
```

Tworzenie Kosmitów.

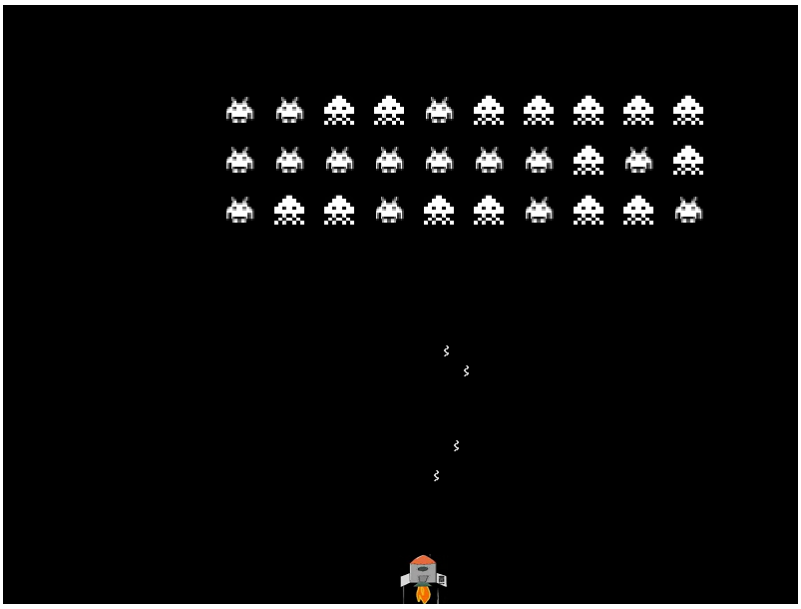
Na koniec została klasa strzal, odpowiadająca za pojedyncze pociski

```
9
10  class strzal
11  {
12  private:
13      sf::Vector2f currpos;
14      sf::Sprite sprt;
15      sf::Texture tex;
16      static int cnt;
17
18
19  public:
20      sf::Sprite& getref();
21      strzal(const sf::Vector2f& x) ;
22      void update();
23      float getposx();
24      float getposy();
25      ~strzal() { std::cout << "usuniety pocisk" << std::endl; cnt--; };
26
27  };
28
29
```

Plik nagłówkowy klasy strzal.

5. Podsumowanie

Ogólnie gra działa i spełnia założenia. Głównymi obiektami odpowiedzialnymi za rysowanie są klasy dostarczone z biblioteki SFML, a klasy pisane przeze mnie głównie służą jako ich interfejsy. Tekstury użyte w grze zostały w większości zapożyczone z internetu, a w mniejszości wykonane przeze mnie, przy pomocy profesjonalnych narzędzi „Paint”. Gra posiada dynamiczny poziom trudności – im mniej kosmitów tym szybciej się oni poruszają, co spowodowane jest redukcją ilości obliczeń w każdej kolejnej iteracji pętli w przypadku redukcji ilości kosmitów.



Screen z gotowej gry.