

*Computer  
Programming  
Laboratory*

*“Cosmic” game*

Author: Nuradil Zhanurov

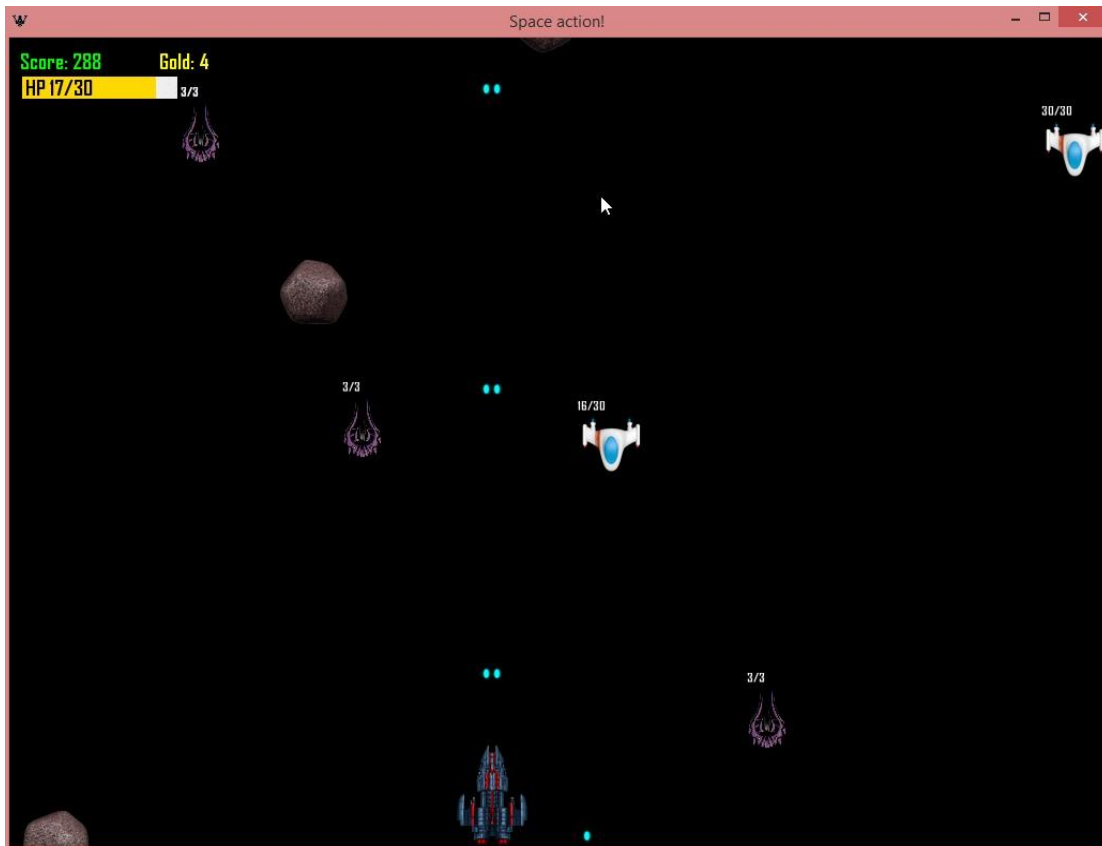
E-mail: [zhanurov97@gmail.com](mailto:zhanurov97@gmail.com)

Tutor: Michał Staniszewski

### Topic of the project:

The idea of the game is to pilot a spaceship, shoot various types of enemies, and get score, coins, lvl-up that opens new, modified attacks and increases difficulty of the game.

Screenshot of gameplay:



### Analysis:

In order to implement 2-d graphics there was SMFL library chosen for it being relatively simple to make a short game.

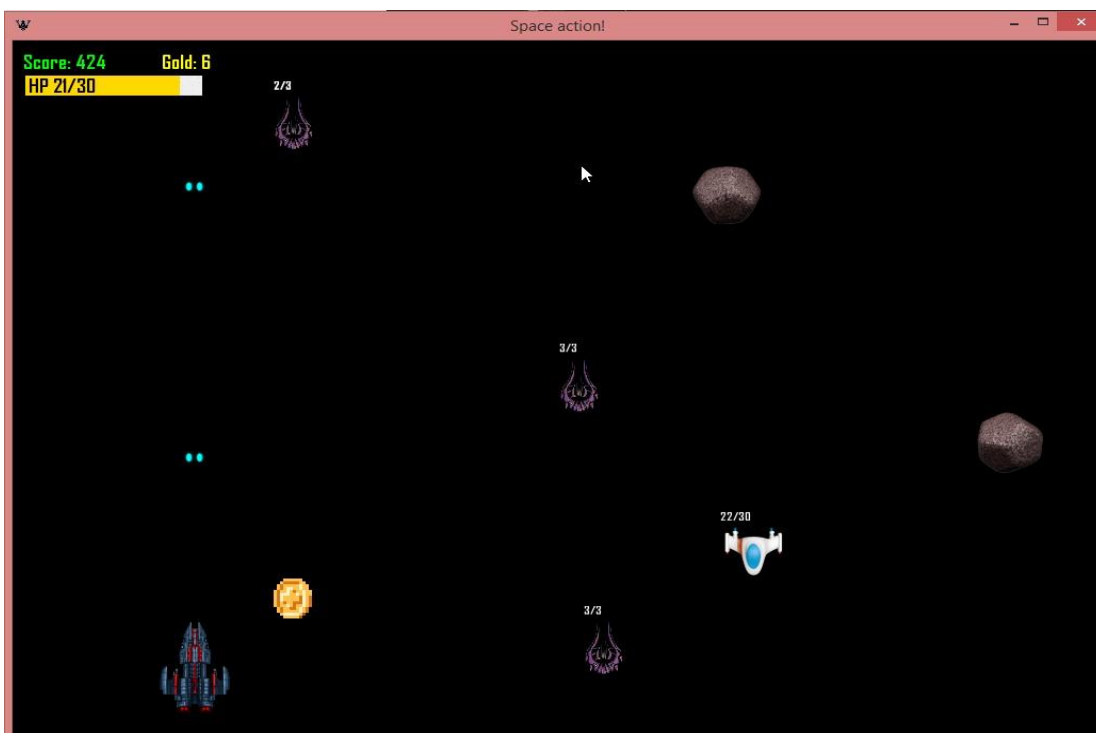
The idea of 2-d game is basically drawing images on the window that we create during a while loop that ends when the game is closed. The entire game object was created in for of a Sprite variable, which can be moved across the screen easily.

## External Specification

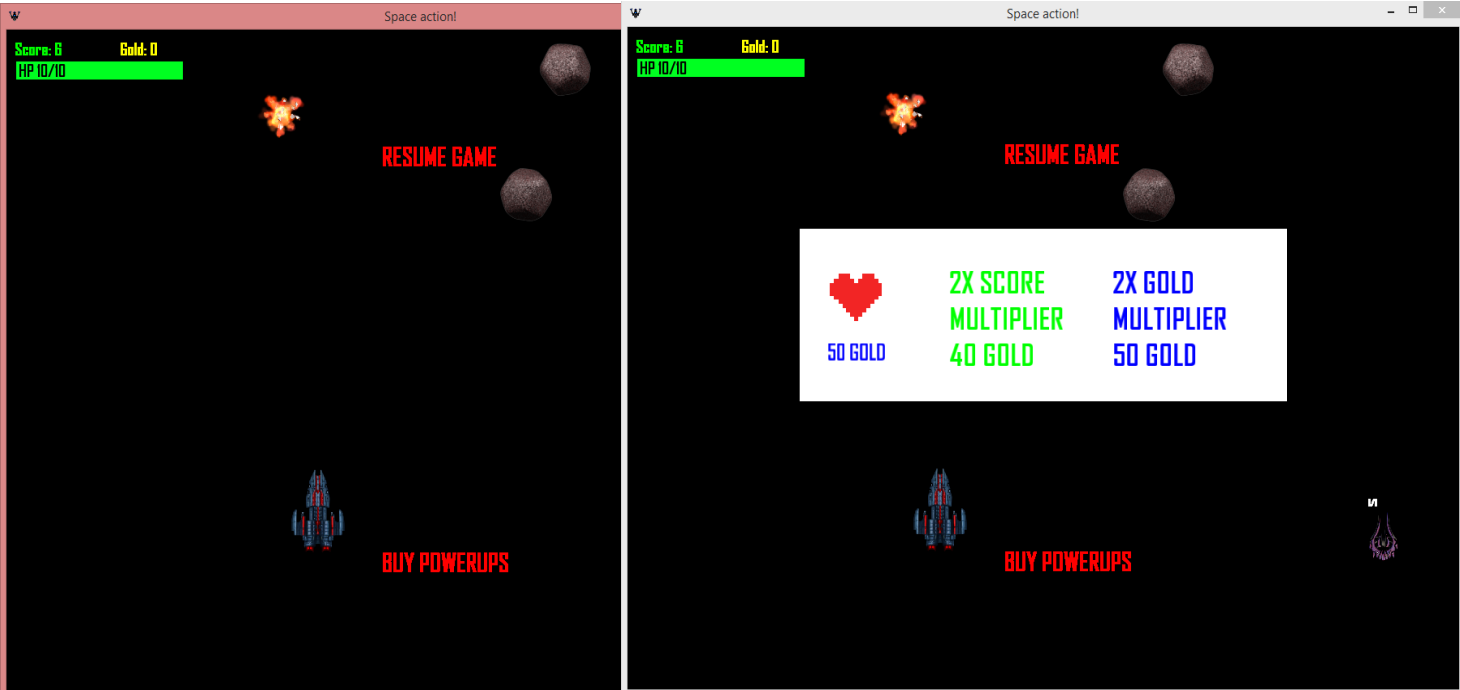
After running the game first menu screen appears, mouse is used to press buttons



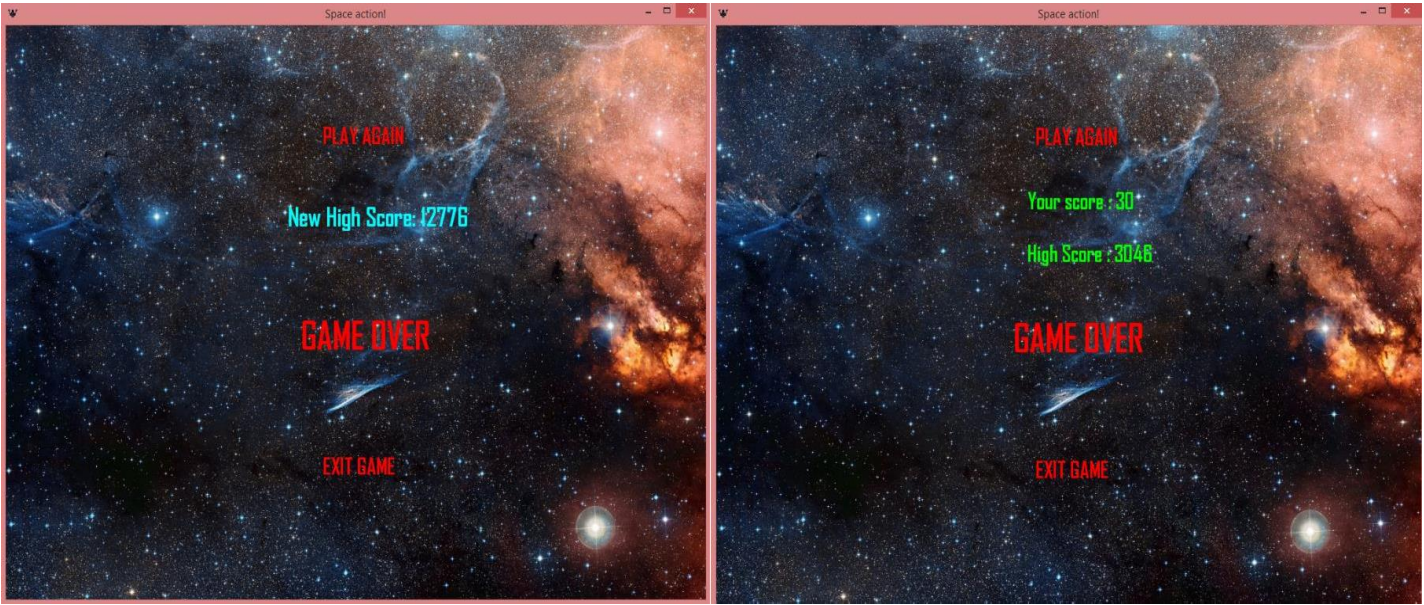
Player ship has health points and 3 different attacks which are updated after lvl-ups. Level increases with the higher score, after each level maximum health of the player is increased and player gets healed. Score is gained when enemy is taken down; score amount depends on type and level of an enemy. There are different types of enemies, such as shooting and non-shooting enemies moving across the screen right-to-left or asteroids. It is possible to get a coin dropped from asteroids with 25% chance.



Coins are used to buy “power-ups” such as healing heart, x2 score and gold multipliers. “Power-Ups” shop is available when game is paused by pressing ESC.



After dying the “game over” screen printed out, showing the score obtained and highest score.



## **Internal Specification**

The following classes have been created:

Base class for all game objects:

```
class Entity;
```

Classes derived from class Entity:

```
class Player;
```

```
class Enemy;
```

```
class Bullet;
```

```
class
```

```
Asteroid;
```

Other classes:

```
class
```

```
Explosion;
```

```
class Coin;
```

```
class Text;
```

```
class Button;
```

```
class Background;
```

```
///
```

class Entity has all the variables and functions common for all the classes derived from it, such as:

Sprite shape, `int` HP, `bool` canMove, `int` lvl, float speed and others.

Basic functions accessing all of the needed protected elements of the classes were created. Two other useful functions declared in the class:

`void` drawTo(sf::RenderWindow \*window) – drawing an object to the screen

`void` takeDamage(`int` lvl) – is used to deal damage to both player and enemies.

class Player is inherited from class entity and has only 1 other variable

`bool` isDead, and a couple of other functions:



`void updateMoves();` - that stands for movement of the player.

`void setLvl(std::vector<Enemy> enemies, int score, int &d1_, int &d2_,  
int &spawnDelay_, int &astrDelay, int &mltpl, int& shootDelay);`

This function is used to increment the level of the game and all the variables that stand for game difficulty such as enemy spawn delay or shooting delay.

The rest of the functions are simply used to reset and access some of protected variables.

One of the most important things is player's bullets vector:

`std::vector<Bullet> bullets;`

class Enemy is very similar to class player, it has its own bullets vector speed and horizontalSpeed variables, the only difference is void shoot() function that let's specific types of enemies shoot. Int shootDelay and int shootDelay\_ variables are used to control enemies' fire rate.

For enemy there exist two types of constructors for two types of enemies:

```
Enemy::Enemy(sf::Texture *texture, sf::Vector2u windowSize, int lvl, bool canShoot, int type)
{
    this->canMove = true;
    this->canShoot = false;
    this->lvl = lvl;
    this->type = type;
    this->speed = 4.f;
    this->HPStart = 1;
    this->HP = HPStart*lvl;
    this->canMove = true;
    this->horizontalSpeed = 3.f;
    this->shape.setTexture(*texture);
    this->shape.setScale(0.3f, 0.3f);
    this->shape.setPosition( rand() % windowSize.x - (int)(this->shape.getGlobalBounds().width), 0 - this->shape.getGlobalBounds().height);
}
Enemy::Enemy(sf::Texture *texture, bool canShoot, bool canHeal, int lvl, sf::Vector2f(size), int type) {
    this->shape.setTexture(*texture);
    this->setPos(rand() % WINDOW_SIZE_X - (int)(this->shape.getGlobalBounds().width), 0 - getSize().y);
    this->shape.setScale(size);
    this->lvl = lvl;
    this->type = type;
    this->HPStart = 1;
    this->canMove = true;
    this->horizontalSpeed = 1.f;
    this->HP = HPStart*lvl;
    this->canShoot = canShoot;
    this->canHeal = canHeal;
    this->isDead = true;
    this->canMove = true;
    this->shootDelay = 0;
    this->shootDelay_ = 50;
    this->speed = .2;
}
```

and following 2 constructors for asteroids, one spawning at random Y position, and another for creating a row of asteroids.

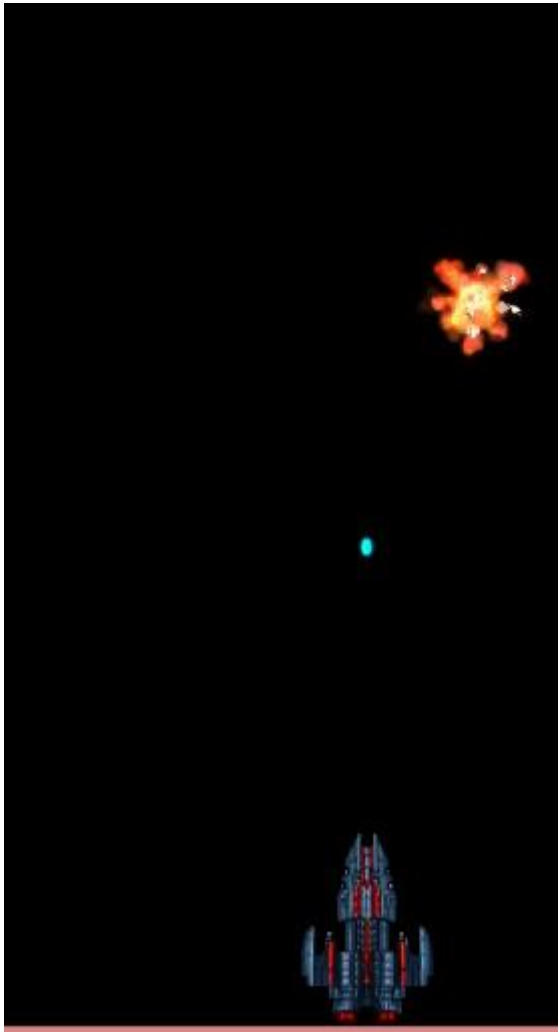
```
Asteroid::Asteroid(sf::Texture *texture, sf::Vector2u windowSize, int lvl) {
    this->lvl = lvl;
    this->speed = 3.f * lvl;
    this->HPStart = 1;
    this->HP = HPStart*lvl - 1;
    this->shape.setScale(1,1);
    this->shape.setTexture(*texture);
    this->shape.setTextureRect(sf::IntRect(rand() % 5 * 72, rand() % 4 * 72, 72, 72));
    this->shape.setPosition(rand() % windowSize.x - (int)(this->shape.getGlobalBounds().width), 0 - this->shape.getGlobalBounds().height);
}
Asteroid::Asteroid(sf::Texture *texture, sf::Vector2u windowSize, int lvl, int n, float y) {
    this->lvl = lvl;
    this->speed = 3.f * lvl;
    this->HPStart = 1;
    this->HP = HPStart*lvl - 1;
    this->shape.setScale(1, 1);
    this->shape.setTexture(*texture);
    this->shape.setTextureRect(sf::IntRect(rand() % 5 * 72, rand() % 4 * 72, 72, 72));
    this->shape.setPosition(0 + n*72, -72 + 72*y);
}
```

class Bullet; - bullet has only it's constructor:

```
Bullet::Bullet(sf::Texture *texture, sf::Vector2f pos, float speed)
{
    this->speed = 15.f;
    this->shape.setTexture(*texture);
    this->shape.setScale(1,1);
    this->shape.setPosition(pos);
}
```

class Explosion; - stands for animation of explosion when enemy is down.

Explosion constructor and function to change the image according to stage:



```
Explosion::Explosion(sf::Texture *texture,
sf::Vector2f(pos))
{
    source = {0, 0};
    sprite.setTexture(*texture);
    sprite.setTextureRect(sf::IntRect(source.
x * 400, source.y * 400, 400, 400));
    sprite.setPosition(pos
);
    sprite.setScale(.4, .4);
    stage = 1;
    explosionTimer = 0;
    explosionTimer_ = 3;
}
```

Explosion texture is loaded from a spritesheet consisting of 8 stages.

So after some timer delay each stage is drawn. sf::Vectro2f source = {0,0}

Is a position on a spritesheet.

Function getSource() returns source

according to the stage.

bool isOver() function returns true if explosion is over according to the stage.

```
void Explosion::change() {
    if (this->stage <= 9) {
        if (this->explosionTimer < this->explosionTimer_) {
            this->explosionTimer++;
        }
        else if (this->explosionTimer = this->explosionTimer_) {
            this->sprite.setTextureRect(sf::IntRect(getSource().x * 400,
getSource().y * 400, 400, 400));
            this->stage++;
            this->explosionTimer = 0;
        }
    }
}
```

Text, button and Background classes were created only for ease of creating multiple text labels, buttons and screen backgrounds with only 1 line of constructor instead of around 10-15 lines for each.

class Coin; - is pretty similar with explosion class as it implements the animation of coin spinning and moving. it works exactly same way like with explosion, the only thing is that the stage is resetting every time it reaches the last part of spritesheet as long as it is inside of the Screen. That lets the coin turn around all the time it exists and move down at the same time with the Coin::move() function.



### Main Function

First of all I created RenderWindow window; which is the window we going to draw everything to. After that I needed to create and load all the textures, sounds that are used later on. The next step was to initialize all the objects I needed:

Text objects that are drawn on the screen, such as – gold, playerHP, enemyHP, scoreboard, gameOver, newHighScore, highScore.

Then algorithm for adding health bar was written.

```
RectangleShape HPbar;  
//std::vector<Texture> Hp;  
HPbar.setPosition(10, 35);  
HPbar.setSize({210.f, 21.1f});  
Texture Hp[11];  
float HP_;  
float HPmax_;  
float HPbar_ = 0;  
for (int i = 0; i <= 10; i++) {  
    Hp[i].loadFromFile("textures/hp/" + std::to_string(i) + ".png");  
}  
HPbar.setTexture(&Hp[0]);
```



The textures loaded into an array are updated during the game with the player taking damage.

The following are all the variables and constants used in game loop, mostly to represent delays, timers, score, stage of the program and etc.

```
bool paused = false;           // if true paused screen is drawn
//VARIABLES FOR DELAYS AND ETC -----
int shootDelay_ = 20; //changed shoot delay int
shootDelay = shootDelay_;
int enemySpawnDelay = 0;
int enemySpawnDelay_ = 100;    // For basic enemies
int d1 = 0;
int d1_ = 2000;                // For 2nd kind of enemies
int d2 = 0;
int d2_ = 1000;                // For asteroid wall
int barDelay = 0;
const int barDelay_ = 15; int
asteroidSpawnDelay = 0;
int asteroidSpawnDelay_ = 50;
int score = 0;
int scoreMultipl = 1;
int gold_ = 0;
int goldMultipl = 1;
int stage = 0;
bool inShop = false;          // if true, shop screen is drawn
```

int stage stands for the stage of the game, for example:

stage == 0 means it's main menu stage.

stage == 1 means the game is going on

stage == 3 shows "game over" screen

---

The last step at this point was to create buttons and backgrounds.

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

is a main game loop, goes on as long as the window is open.

All the following part of the program is basically handling all the objects' changes, drawing and deleting them when needed. Some of the most important parts of the code were replaced by functions in function.cpp file in order to make main function shorter, make those parts of code easier to modify and avoid a lot of mess in code. Here are some of the most important of them:

`void handleMenu(...);` function check for a button pressed in main menu, and sets stage to 1 if PLAY button has been pressed, and `window->close();` when EXIT, as well as draws the menu screen.

`void handlePausedMenu(...);` draws paused screen if `bool paused == true`; and checks for button pressed. If "RESUME" button is pressed it sets `paused = true`; or

bool inShop = true; which allows the next `void handleShop();` function to do the same thing for shop screen.

`void isCollidingWindow(Player *ship, RenderWindow *window);`

This function doesn't let the ship go outside of the window boundaries.

`void createBullets(...);` adds bullets to ship.bullets vector according to the level of the player. It checks if either LKM or SPACE have been pressed, and if the shoot delay allows to it add bullet to the vector.

`void handleEnemies(...);` is the part where I add enemies, first or the enemy spawn timers are checked and updated. Then if the timer allows, enemies are added to vectors. As well as this, the enemies movements and collision are considered in this function. Nothing special was used there to be mentioned.

`void reset(everything to reset);` resets all the necessary variables to default, clears all the vectors. Executed when "PLAY AGAIN" is pressed on "game over" screen.

---

`int checkHighScore();`

`void updateHighScore(int score);`

`void updateGold(int& gold);`

These functions are used to manipulate with the files that store high score and total gold. (No use of total gold was implemented at this point).

If score reached is higher than high score saved in the file, the file is being updated by `updateHighScore(score);` After every game the gold adds to the total gold value stored in the file, and saved back to the file.

So everything happening in the main game loop is checking for events, such as shooting, moving. Enemies are created regardless to all the events through the entire loop.

## **Testing**

While creating the game I didn't come across to many troubles.

First problem met was that the game started glitching and slowing down after some time player. As a result of debugging I found out that I forgot to clear the enemies' bullets vector and it kept adding bullets over and over, making updating of the bullets very time consuming.

One of the most notable one was the problem with vectors, for some reason game was crashing at some points. As I figured out it was when I was trying to erase from vector inside of for loop. The solution appeared to be pretty simple – just break; all the time after calling erase() function.

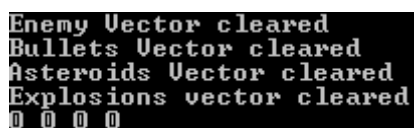
Another problem is that sometimes when enemy is shooting the bullet texture is not loading and it shows white rectangle instead of it. The reason was not figured out, so the problem remained unsolved.

No other notable problems were met, as well as no errors occurs during game process, everything seems to be working smoothly.

The game have been tested for hours, and no notable problems were found.

It successfully goes into another game after dying clearing all the vectors and resetting all the variables.

vectors cleared, sizes of vectors == 0



```
Enemy Vector cleared  
Bullets Vector cleared  
Asteroids Vector cleared  
Explosions vector cleared  
0 0 0 0
```

Scores and gold is indeed being updated and saved into files after each game.

The level system works too, ship get new attacks (double, and triple firing). And difficulty of the game is increasing as all the spawn delays are decreasing and speed of enemies increasing as well as their HP.