Bournemouth University

Faculty of Science and Technology

2 0 17/ 2 0 18

Level 4

Object Oriented Games Programming

2D Video Game Program

Analysis, Design, and Implementation

Report Template

Student number: s4922675
Maciej Legas

1. Game Concept (Marks: 5%)

The game, named “Aliens From Outer Space”, is a classic 2D space shooter action game. The player, who is in control of a spaceship, is on a mission to repel the attack of Unidentified Flying Objects (also known as aliens). As the player’s goal is to kill enough aliens to make them retreat, he has to make his way avoiding the enemies’ shots and shooting as many as he can using his cannon.

The purpose of my game is to entertain, while also enhance the player’s reflexes. Therefore I will use a small enough game scene and generate enough enemies to keep him under stress. The usual gameplay time should also be short enough, to have the player restart the game if he wishes to play more and to make the goal more achievable, instead of having him become bored or irritated and quitting the game early.

To make the game more varied and less bland, the game will implement a few different enemy types, having different shooting mechanisms and speed values. A power-up mechanic will also be implemented to make up for the difficulty of the game, increasing the player’s spaceship statistics. The player will be able to pick up the power-ups from the wrecks of alien ships, basing on a percentage chance. The power-ups will affect the speed of the ship, how many bullets does it fire, the firing speed, the strength of its shots and allow the player to repair the ship, which will have a visual representation on the game’s interface. On the other hand, not to make the game too easy if the player maxes out the abilities on the ship, if the player’s spaceship gets shot the abilities should be downgraded a level so that the player is always aware that he can lose them, and so remain cautious in a way.

The game will also make use of an automatically generated background to create an illusion of a moving world.

The game should end when the player shoots enough aliens to repel the attack. This should be represented for clarity either by a total score or by a number of aliens shot down on the interface.

2. Analysis (Marks: 20%)

Having the game concept completed, we have the most important elements for the game loop. The game will have a galactic background, an interface, the player's spaceship and the alien spaceship. The player's spaceship and alien spaceships also have to fire bullets.

Let's expand on those main points for a brief description of the game loop.

Draw:

- The background of the game
- An interface, showing the score, power-up levels and the health of the spaceship
- The player's spaceship
- The alien spaceships
- Shot bullets

Basing on user input:

- Move the player's spaceship
- Shoot the cannon on player's spaceship

Check:

- If the player's spaceship has been shot
    - If yes, deduce an amount from the ship's health and destroy the bullet
- If any of the enemy spaceships has been shot
    - If yes, deduce an amount from the ship's health and destroy the bullet
- If the player's spaceship has been destroyed
    - If yes, remove the spaceship and display a game over screen
- If any of the enemy spaceships has been destroyed
    - If yes, remove the spaceship, play an explosion animation, add a value to the score and drop a power-up basing on a percentage chance

Update:

- Music and sounds
- The interface
- The player's spaceship statistics, basing on action:
    - Power-ups:
        - Health
        - Power
        - Speed
    - Taking damage (bullets)
        - Health
        - Power
        - Speed
- The game's window with drawn sprites

Create:

- Bullets
- Enemies

We can now create a simplistic game loop to develop pseudocode from it.

```
Clear the screen
Draw the background of the game
Draw an interface
Draw the player's spaceship

If there are any enemies created
        Draw enemies
If there are any bullets created
        Draw bullets
If the user pressed movement keys
        Change the position of the player's spaceship
If the user pressed a shooting key
        Create a bullet
If the player's spaceship collided with a bullet
        Deduct an amount from this ship's health
If the player's spaceship has been destroyed (health <= 0)
        Remove the spaceship and display a game over screen
If any of the enemy spaceships collided with a bullet
        Deduct an amount from this ship's health, and from his statistics
If any of the enemy spaceship has been destroyed (health <= 0)
        Remove the spaceship, play an explosion animation, add a value to the score and
drop a power-up basin on a percentage chance

Update the music and sounds
Update the interface
Update the player's spaceship statistics
Update the game's window

Create enemies with each time interval
        Make them shoot with each time interval
                Create bullets

If the player's score hits the goal
        Display a game over screen
```

What is already clear is that we will need loops to go through every created bullet and every created enemy to check for collisions, and to do that we need to store them, preferably by STL vectors. The bullets shot by the enemy and by the player should be stored in different containers to make sure that simultaneous creation of a bullet by the enemy and the player will not cause any problems. We also need to check if the bullets and enemies should be still drawn, as in if they are off-screen.

An another thing is that since the player's spaceship statistics are updated only when a power-up is picked up or when the player is hit, they should not be updated with each iteration of the game loop. Rather, we should only change the ship's characteristics when an event of such happens, and just let the interface update itself with each iteration, taking the values from the player's ship. This should be represented by a class (should be a struct if not for the necessary update function).

Therefore, we can also see that already a few of the used objects should be represented as classes in code, since they have their own identity, attributes and state.

These are:

- Player
- Interface
- Enemies
- Explosion
- Bullets
- Power-ups

The explosion animation could be implemented as a function, as it needs to update the frames, and having that implemented in the game loop would be unnecessary. But since we need to keep the explosion at the last position of the destroyed space ship, as well as the sprite of the explosion, and also since that the explosions are created dynamically just as the enemies or bullets, they should be implemented as classes.

Finally, as I will be using SFML, drawing sprites to the game window will just require the use of its functions, as well as creating functions to return sprites from game objects. That will remove the need for writing many rendering functions.

Let's now write down the game loop in simple pseudocode and mark the modules.

BEGIN
Clear the screen
Draw the background of the game
Draw an interface -> Interface class
Draw the player's spaceship

SFML

WHILE There are existing enemies -> Enemy class
    Draw enemies -> SFML

    IF Any of the enemy spaceships collided with a bullet -> Bullet class
        Deduct an amount from this ship's health -> Enemy.setHealth()
    END IF

    IF Any of the enemy spaceship has been destroyed (health <= 0) -> Enemy.getHealth()
        Remove the spaceship -> Vector enemies, play an explosion animation -> Explosion class, add a value to the score and drop a power-up basin on a percentage chance
    END IF

    IF Any of the enemy spaceship has gone under the lower boundary of the screen -> Enemy.getSprite().getPosition()
        Remove the spaceship -> Vector enemies
    END IF

    ELSE
        Move the spaceships -> Enemy.move()
    END ELSE
END WHILE

WHILE There are existing bullets -> Bullet class
    Draw bullets -> SFML
END WHILE
IF The user pressed movement keys -> SFML
    Change the position of the player's spaceship -> Player.move()
END IF

IF The user pressed a shooting key -> SFML
    Create a bullet -> Bullet class, Player.shoot()
END IF

IF The player's spaceship collided with a bullet
    Deduct an amount from the player's ship health, and from his statistics -> Player.isHit()
END IF

IF The player's spaceship has been destroyed (health <= 0) -> Player.getHealth()
    Remove the spaceship and display a game over screen -> GameOverScreen()
END IF

Update the music and sounds
Update the interface -> Interface class
Update the game's window

SFML

Create enemies with each time interval -> Enemy class, Vector
    IF The time interval is hit
        Make them shoot -> Enemy.shoot()
        Create bullets -> Bullet class
    END IF

IF The player's score -> Interface.getScore() hits the goal
    Display a game over screen -> GameOverScreen()
END IF

END

Therefore, as I said before, we should create six classes just from looking at the game loop.

What I thought of using for the background except of a static background, would be falling 2D filled circles which would imitate asteroids in a way. This would create an illusion of “action”, that the game is taking place at a high speed in space. The name that I thought of for them would be BackgroundSparkles.

Here is the simple pseudocode for it:

```
BEGIN

Draw 70 background sparkles

IF The position of the sparkle is below the game scene
        Move it above the screen with a random value added to or substracted from the X
        coordinate
END IF

END
```

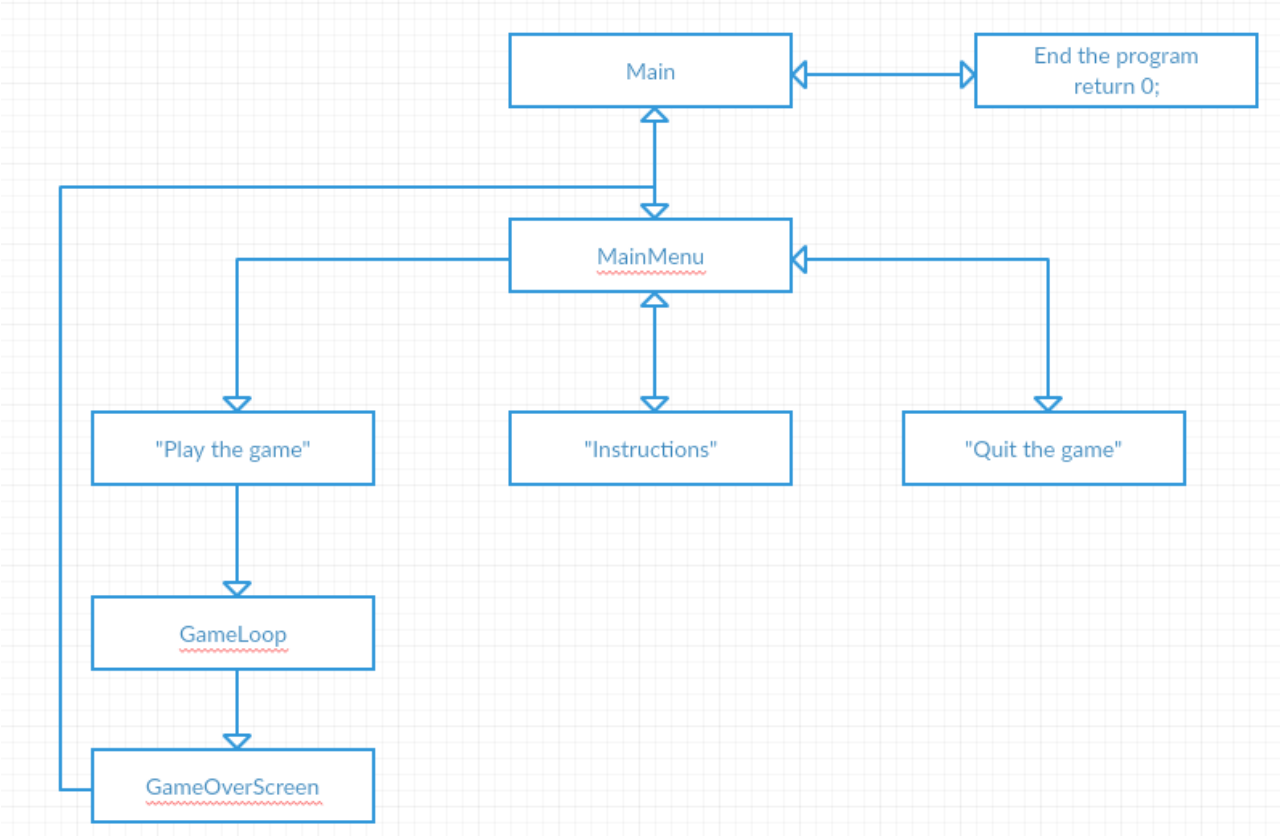
As each sparkle needs to hold its coordinate as an attribute, we need to set it as a class, and check it with each iteration of the game loop. That means we need to use a STL vector.

Now, to begin the game we should make use of a main menu, like so:

```
BEGIN

Draw a static background
Draw text
Draw buttons

IF Button no. 1 clicked
        CALL GameLoop()
END IF

IF Button no. 2 clicked
        CALL Instructions() (just a game screen explaining the game theme and controls)
END IF

IF Button no.3 clicked
        RETURN;
END IF

END
```

As buttons are not supported natively, we need to create a class that would store a sprite, texture and the action that a button would do, or a number representing the action.

Here is a module diagram that I will try to follow when designing the program.



3. Design (Marks: 25%)

As of now, we are sure that we need 8 classes:

- Background Sparkle
- Bullet
- Button
- Enemy
- Explosion
- Interface
- Power-up
- Player

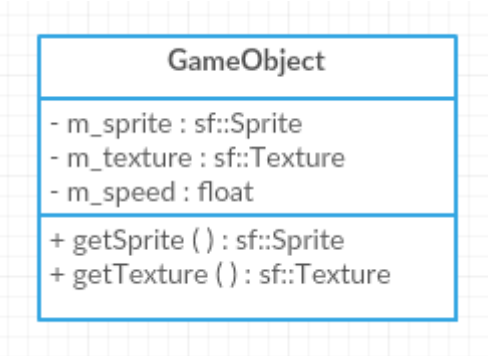
As well as four functions – MainMenu, GameLoop, Instructions and GameOverScreen.

As I found early on, SFML doesn't have a native way of setting the origin to center instead of the upper left corner. Therefore, I also will need a function called SetOriginToCenter. This will be used for buttons and most of the sprites.

Now, can we find a common characteristic between the classes? Of course we can. Most of them contain a sprite and a texture, as well as speed value for moving. Therefore we can create a base class called GameObject, which will be a base class for these classes:

- Background Sparkle
- Bullet
- Button
- Enemy
- Explosion
- Interface
- Power-up
- Player

Almost all of the derived classes also have to move except for the Explosion class, so we could create also a move() function which would move the sprites of the derived classes. Unfortunately, most of them differ in the way they move, as in – taking different parameters than the base class. Having to define an empty virtual function just so that the compiler does not complain for most of the classes would be a bad use of polymorphism, therefore I set a separate move() function for all of the classes.



The getSprite() and getTexture() functions do exactly what the name says – return the m_sprite and m_texture.

The header file of GameObject also contains an enum Direction:

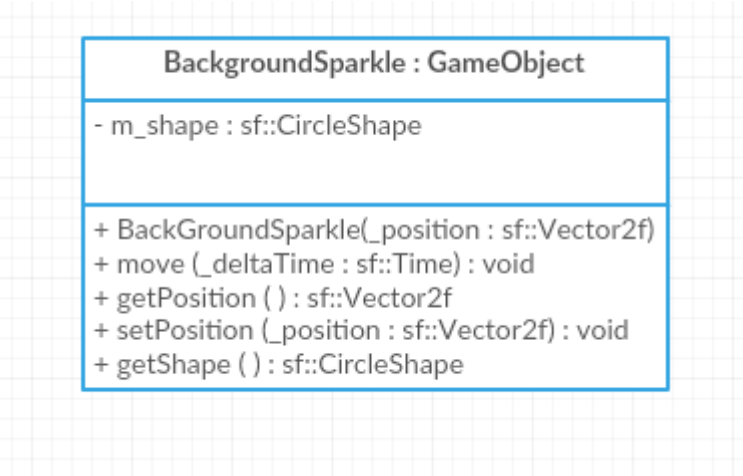
BEGIN

ENUM Direction
	m_left, m_right, m_up, m_down
END ENUM

END

This enum is used for specifying the movement of the player, as well as setting the direction of bullets.

Let's move on and describe each of the classes.



BackgroundSparkle is used for creating the animated background, as it imitates asteroids and other star objects. The constructor of it creates a sf::CircleShape at the given location, like so:

BEGIN

Set the position of the m_shape to _position
Set the radius of the m_shape to 5
Set the fill color to white, with transparency
Set the m_speed to 150 (speed of movement)

END

This is later used when constructing the initial background when the game is firstly run.

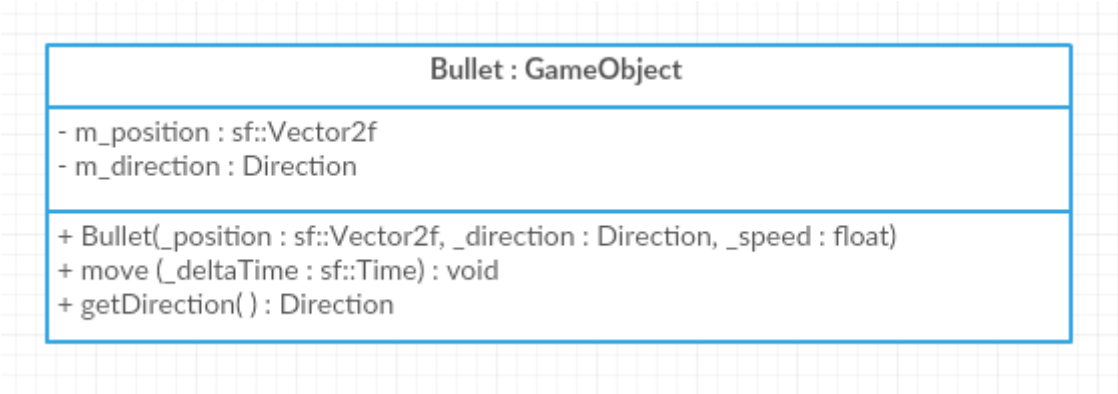
The move() function is:

BEGIN

Move the shape's Y coordinate by m_speed times _deltaTime

END

The _deltaTime parameter is used to fix the movement with the time difference, as in the running speed of the game.



Bullet is used for creating bullets, simple as that.

The constructor's pseudocode:

BEGIN

Set m_speed to _speed
Set m_direction to _direction

IF _direction == m_up
    Load "bullet.png" for the m_texture
ELSE IF _direction == m_down
    Load "enemyBullet.png" for the m_texture
END IF

Set the m_texture for sf::Sprite temp
Set temp's origin to center
Set temp's position to _position

Set m_sprite as temp

END

The check of the direction of the bullet is to see if the bullet should be shown as a bullet of the enemy team or of the player, for the visual aid.

The move() function:

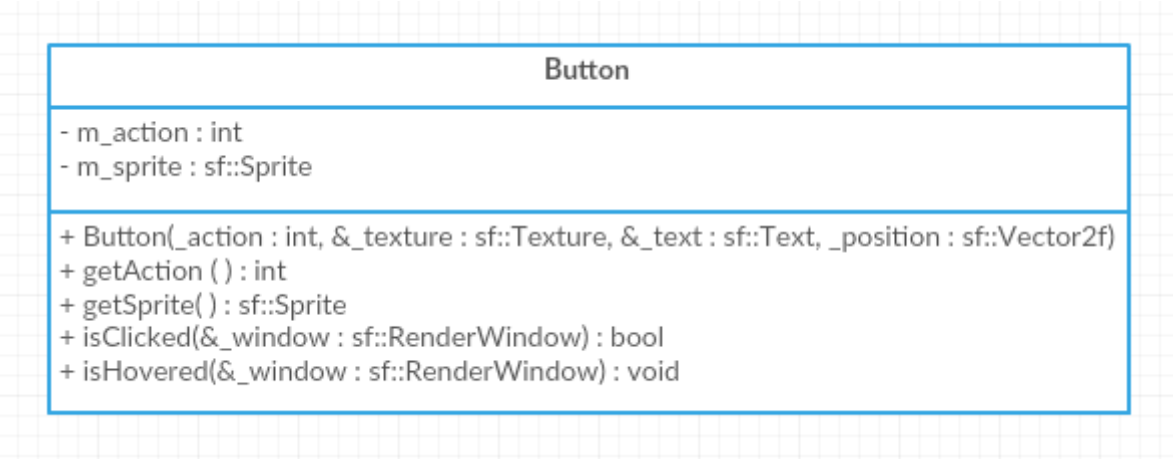
BEGIN

IF m_direction == m_up
    Move the shape's Y coordinate by -m_speed times _deltaTime
ELSE IF m_direction == m_down
    Move the shape's Y coordinate by m_speed times _deltaTime
ELSE
    OUTPUT An error message
END IF

END

The m_direction variable, set via the constructor, is used to determine which way should the bullet travel – upwards or downwards.

The getDirection function simply returns the direction of the bullet.



The button class has been implemented as there is no native support for clickable sprites in SFML.

It creates a button combining a text and a texture, setting the text at the center of the button.

The constructor:

BEGIN

Set m_action to _action
Set the origin of _text to its center

INITIALIZE sf::Sprite button with the _texture
Set the origin of button to its center

Set the position of button to _position
Set the position of _text to _position

Set m_sprite to button

END

As seen, the constructor simply creates a button using the passed _text and _texture. The reference symbols are used to ensure the texture and text is not lost in memory when they are passed.

The getAction() function simply returns an m_action. This is later on used in MainMenu to decide what should the function do based on the click of the chosen button.

Since Button is not really a game object, the getSprite() had to be specified here. This is used to draw the sprite of the button.

The isClicked() function:

BEGIN

IF Mouse position is in the bounds of m_sprite

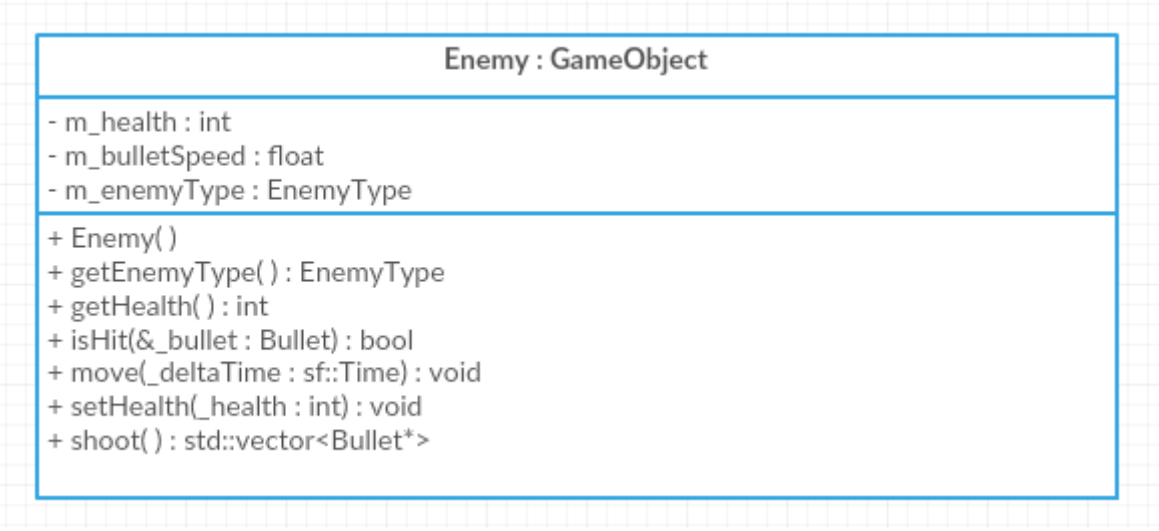
```
        RETURN true
ELSE
        RETURN false
END IF

END
```

The return of this function is later combined with SFML's function to detect mouse input. If there has been a left mouse click, this function will check if the given button has been clicked.

The isHovered() function:

```
BEGIN

IF Mouse position is in the bounds of m_sprite
        Set the m_sprite to be half-transparent
ELSE
        Set the m_sprite to be just a bit transparent
END IF

END
```

This function creates a nice effect of a responsive mouse system for the menu.



The Enemy class. The main antagonist, villain.

The header file also contains an enum for different types of the enemy:

```
BEGIN

ENUM EnemyType

m_weak, m_normal, m_strong

END ENUM

END
```

The names tell for themselves. The stronger the alien is, the higher score value the player gains for killing it.

The constructor:

BEGIN

Set the seed of rand() to a time generated value

INITIALIZE integer temp

Randomize a value between 1 and 10 inclusive, set it as temp

```
IF temp <= 4
        m_enemyType = m_weak
ELSE IF temp > 4 and temp <= 7
        m_enemyType = m_normal
ELSE IF temp > 7
        m_enemyType = m_strong
END IF

IF m_enemyType == m_weak
        Load “alien1.png” to m_texture
        Set m_health to 3
        Set m_speed to 225
        Set m_bulletSpeed to 400
ELSE IF m_enemyType == m_normal
        Load “alien2.png” to m_texture
        Set m_health to 6
        Set m_speed to 175
        Set m_bulletSpeed to 400
ELSE IF m_enemyType == m_strong
        Load “alien3.png” to m_texture
        Set m_health to 10
        Set m_speed to 100
        Set m_bulletSpeed to 400
END IF
```

Set m_sprite’s texture to m_texture

Randomize a value between 300 and 725 inclusive, set it as temp

Set the X coordinate of m_sprite to temp
Set the Y coordinate of m_sprite to -200

Set the origin of m_sprite to its center

END

This function sets the type of the generated enemy to one of the three types, with a randomly generated chance deciding which alien will spawn. 40% for a weak one, 30% for a normal one, and 30% for a strong one.

The getEnemyType function returns the m_enemyType. This will be used to check what score award should be given for killing the alien.

The getHealth function returns the health of the alien. This will be used to check if the alien is dead.

The isHit() function:

BEGIN

IF Bullet's sprite intersects m_sprite and the direction of the bullet is not down (check if the bullet is from the enemy team)
	Subtract 1 from m_health
	RETURN true
ELSE
	RETURN false

END

This function is later used with each bullet that the player shot. Even though player's bullets and the AI's bullets use two different vectors to differentiate between them, I check the bullet anyway to ensure that no friendly fire is taking place, even if somehow the vectors got switched.

The shoot() function:

BEGIN

INITIALIZE A STL vector<Bullet*> createdBullets

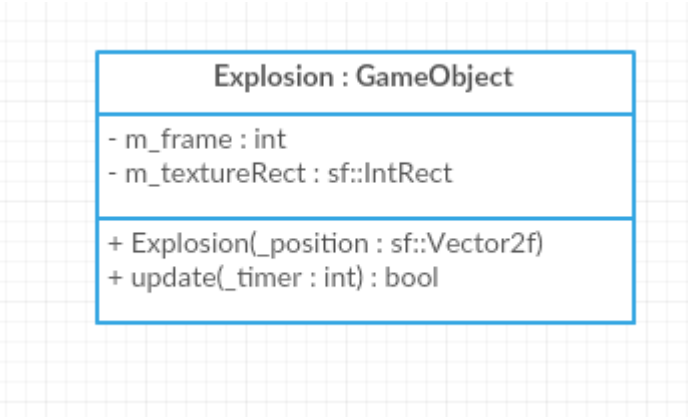
IF m_enemyType == m_weak
	Create a pointer to a bullet
	Initialize the pointer with the X position of m_sprite, and Y position plus 60 of m_sprite, Direction m_down and m_bulletspeed
	Push the pointer to the vector

ELSE IF m_enemyType == m_normal
	Create a pointer to a bullet
	Initialize the pointer with the X position minus 30 of m_sprite, and Y position plus 60 of m_sprite, Direction m_down and m_bulletspeed
	Create another pointer to a bullet
	Initialize the pointer with the X position plus 30 of m_sprite, and Y position plus 60 of m_sprite, Direction m_down and m_bulletspeed
	Push both pointers to the vector

ELSE IF m_enemyType == m_strong
	Create a pointer to a bullet
	Initialize the pointer with the X position minus 50 of m_sprite, and Y position plus 60 of m_sprite, Direction m_down and m_bulletspeed
	Create another pointer to a bullet
	Initialize the pointer with the X position of m_sprite, and Y position plus 60 of m_sprite, Direction m_down and m_bulletspeed
	Create another pointer to a bullet
	Initialize the pointer with the X position plus 50 of m_sprite, and Y position plus 60 of m_sprite, Direction m_down and m_bulletspeed
	Push the three pointers to the vector
END IF
RETURN The STL vector createdBullets

END

This function returns a vector filled with bullets, which will be later on added to a vector of all enemy bullets in the game loop. I used pointers to ensure that the sprites and textures of the bullets do not get lost in memory.



The Explosion class. This is an animated sprite that will be shown when a spaceship is destroyed.

The constructor function:

BEGIN

Load “explosion.png” to m_texture
Set the texture of m_sprite to m_texture
Set the m_sprite to show the first frame of the animation
Scale the m_sprite to 2x
Set the position of m_sprite to _position
Set the origin of m_sprite to center
Set m_frame to 1

END

The constructor simply loads the animation when created.

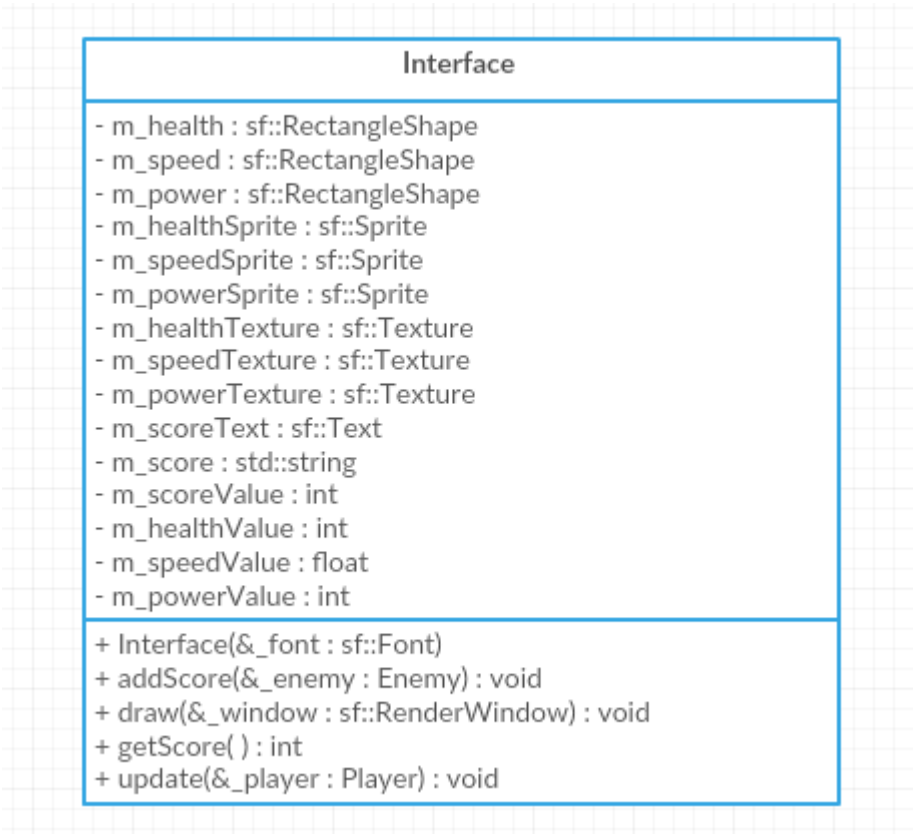
The update() function:

BEGIN

IF m_frame <= 5 and _timer divided by 40 does not leave any remainder
	Increment m_frame
	Set the m_sprite to the m_frame’th frame of the animation
	RETURN true
ELSE IF m_frame <=
	Set the m_sprite to the m_frame’th frame of the animation
	RETURN true
ELSE
	RETURN false
END IF

END

The returns of the function are used to check if the animation is still playing. If it is not, the game loop will delete the explosion.



The Interface class is the class responsible for drawing the Heads-Up Display.

The constructor:

BEGIN

Set m_scoreText's origin to center
Set m_scoreText's position to the upper right corner
Set m_score to the string of m_scoreText
Set m_scoreValue to the integer cast of m_score

Set m_health's size
Set m_health's origin to the left bound of m_health
Set its position to the left side
Rotate it around 180 degrees
Set the fill color to red

Set m_power's size
Set m_power's origin to the left bound of m_power
Set its position to the left side
Rotate it around 180 degrees
Set the fill color to yellow

Set m_speed's size
Set m_speed's origin to the left bound of m_speed
Set its position to the left side
Rotate it around 180 degrees
Set the fill color to cyan

Load the textures for power-up sprites
Set their origins to center
Set them under corresponding power-up bar

END

This function initializes the Heads-Up Display. The values are pre-defined as the constructor runs only once at the start of the game. If the game had more levels and the power-up values would stay after finishing one, this should be changed.

The addScore() function:

```
BEGIN

IF The type of the destroyed enemy == m_weak
        Add 3000 to m_scoreValue
END IF
IF The type of the destroyed enemy == m_normal
        Add 5000 to m_scoreValue
END IF
IF The type of the destroyed enemy == m_strong
        Add 10000 to m_scoreValue
END IF

Set the score on HUD to the new score

END
```

As one may already see, the score value is a local value of the HUD, even though the power-ups and player's health are values of the player. This is done deliberately since the interface is the representation of the game state.

The draw() function:

```
BEGIN

Draw the HUD on the screen

END
```

This function draws the HUD. I wanted to keep it separate from update() for debugging purposes.

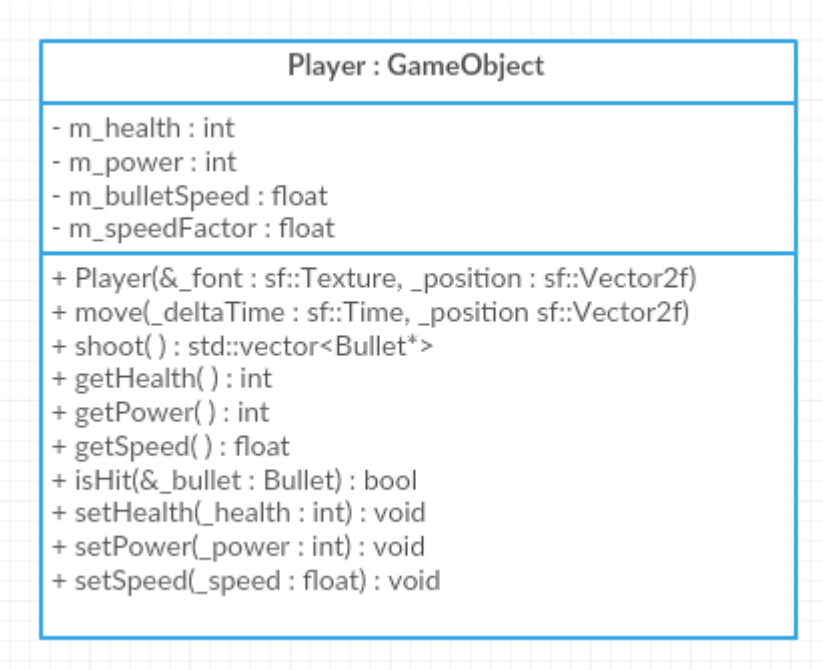
The getScore() function returns the m_scoreValue. This is used for the addScore() function.

The update() function:

```
BEGIN

Set the values for the power-ups by using player.getHealth() etc.

BEGIN SWITCH (m_healthValue)
        CASE 3
                Set health bar size to full
        END CASE
        CASE 2
                Set health bar size to 2/3
        END CASE
        CASE 1
                Set health bar size to 1/3
        END CASE
```

```
        CASE DEFAULT
            OUTPUT An error
        END CASE
END SWITCH

BEGIN SWITCH (m_powerValue)
        CASE 5
            Set the power bar size to full
        END CASE
        CASE 4
            Set the power bar size to 4/5
        END CASE
        CASE 3
            Set the power bar size to 3/5
        END CASE
        CASE 2
            Set the power bar size to 2/5
        END CASE
        CASE 1
            Set the power bar size to 1/5
        END CASE
        CASE DEFAULT
            OUTPUT An error
        END CASE
END SWITCH

INITIALIZE int temp

Set temp to 10 times m_speedValue (since it is a float)

BEGIN SWITCH (temp)
        CASE 20
            Set speed bar size to full
        END CASE
        CASE 15
            Set speed bar size to 2/3
        END CASE
        CASE 10
            Set speed bar size to 1/3
        END CASE
        CASE DEFAULT
            OUTPUT An error
        END CASE
END SWITCH

END
```

This function updates the interface with the values received from the player. It will be called with each iteration of the game loop.



The Player class. It is quite similar to the enemy class, with the difference being that the player's movements are dependent on key inputs and that the player can pick-up power-ups.

The constructor function:

BEGIN

Set m_sprite's texture to _texture
Set m_sprite's origin to center
Set m_sprite's position to _position
Scale m_sprite by 2/3

Set m_health to 3
Set m_power to 1
Set m_speed to 300
Set m_bulletSpeed to 400
Set m_speedFactor to 1

END

A simple constructor setting the values for a new game.

The move() function:

BEGIN

BEGIN SWITCH (_direction)
    CASE m_up
        Move the Y coordinate of the sprite by -m_speed times m_speedFactor times _deltaTime
    END CASE
    CASE m_down
        Move the Y coordinate of the sprite by m_speed times m_speedFactor times _deltaTime
    END CASE
    CASE m_left

```
            Move the X coordinate of the sprite by -m_speed times m_speedFactor
            times _deltaTime
        END CASE
        CASE m_right
            Move the X coordinate of the sprite by m_speed times m_speedFactor times
            _deltaTime
        END CASE
        CASE DEFAULT
            OUTPUT An error
        END CASE
END SWITCH

END
```

This move mechanic bases the movement on _direction, which is passed by the game loop depending on which key is pressed.

The getHealth(), getPower(), getSpeed() functions return: m_health, m_power, m_speedFactor (since m_speed is the moving speed, initialised by GameObject).

The isHit() function:

```
BEGIN

IF Bullet's sprite intersects the m_sprite, and the direction of the bullet is not up
        Subtract one from m_health
        IF m_speedFactor is not equal to 1
            Subtract 0.5 from m_speedFactor
        END IF
        IF m_power is not equal to 1
            Subtract 1 from m_power
        END IF
        RETURN true
ELSE
        RETURN FALSE
END IF

END
```

The Boolean return is used to delete the bullet that hit the player.

The shoot() function:

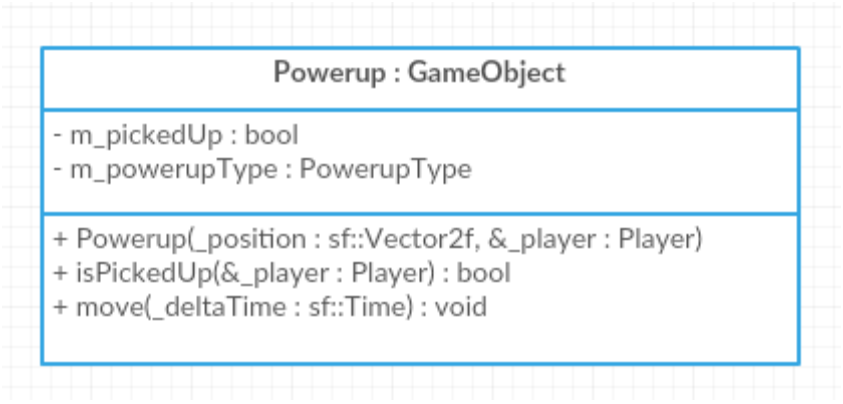
```
BEGIN
INITIALIZE STL vector<Bullet*> createdBullets

BEGIN SWITCH (m_power)
        CASE 1
            Create a pointer to a bullet
            Initialize the pointer with the X position of m_sprite, and Y position minus 60
            of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
            Push the pointer to the vector
        END CASE
        CASE 2
            Create a pointer to a bullet
```

Initialize the pointer with the X minus 10 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet
Initialize the pointer with the X plus 10 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Push the pointers to the vector
END CASE
CASE 3
Create a pointer to a bullet
Initialize the pointer with the X minus 20 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet
Initialize the pointer with the X position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet
Initialize the pointer with the X plus 20 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Push the pointers to the vector
END CASE
CASE 4
Create a pointer to a bullet
Initialize the pointer with the X minus 30 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet
Initialize the pointer with the X minus 10 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet
Initialize the pointer with the X plus 10 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet
Initialize the pointer with the X plus 30 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Push the pointers to the vector
END CASE
CASE 5
Create a pointer to a bullet
Initialize the pointer with the X minus 40 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet
Initialize the pointer with the X minus 20 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet
Initialize the pointer with the X position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet

Initialize the pointer with the X plus 20 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Create another pointer to a bullet
Initialize the pointer with the X plus 40 position of m_sprite, and Y position minus 60 of m_sprite, Direction m_up and m_bulletspeed times m_speedFactor
Push the pointers to the vector
END CASE
END SWITCH
RETURN createdBullets

END

Just as in case of the Enemy's shoot() function, this one as well returns a vector to add later to a larger vector in the game loop.



The Powerup class. It allows the player to increase his stats.

The header file also contains an enum for the powerup types.

ENUM PowerupType
m_health, m_power, m_haste
END ENUM

The constructor function:

BEGIN

Set m_pickedUp to false

Set the seed of srand to time-based
INITIALIZE int Random

Randomize a value between 1 and 10 inclusive
Set it to random

IF Health of player < 2
IF Random <= 2
Set m_powerupType to m_power
Load the texture "power.png"
ELSE IF Random > 2 and Random <= 8
Set m_powerupType to m_health
Load the texture "health.png"
ELSE IF Random > 8

```
            Set m_powerupType to m_haste
            Load the texture “speed.png”
        END IF
ELSE
        IF Random <= 4
            Set m_powerupType to m_power
                Load the texture “power.png”
            ELSE IF Random > 4 and Random <= 7
                Set m_powerupType to m_health
                Load the texture “health.png”
            ELSE IF Random > 7
                Set m_powerupType to m_haste
                Load the texture “speed.png”
            END IF
END IF

Set the m_sprite’s texture to m_texture
Set the origin of m_sprite to center
Set the position of m_sprite to _position

Set m_speed to 100

END
```

The constructor randomizes the type of the powerup with each powerup creation. Also, if the health of the player is low, the odds of getting a health powerup are higher.

The move() function:

```
BEGIN

Move the m_sprite’s Y coordinate by m_speed * _deltaTime

END
```

The isPickedUp() function:

```
BEGIN

IF m_sprite intersects player’s sprite
        IF m_powerupType is equal to m_health and player’s health is under 3
            Add 1 to player’s health
        ELSE IF m_powerupType is equal to m_power and player’s power is under 5
            Add 1 to player’s power
        ELSE IF m_powerupType is equal to m_haste and player’s speed is under 2
            Add 0.5 to player’s speed
        END IF
        Set m_pickedUp to true
        RETURN true
ELSE
        RETURN false
END
```

The returns are used to check if the powerup should be deleted or not in the game loop.

Having all of the classes done, we can finally use them in the gameloop.

Almost all of the classes have been used in the game loop as pointers to ensure that the lifetime of the classes carries on.

4. Commented Source Code

Complete this section using Arial font 12pts

```
#pragma once
#ifndef _BACKGROUNDSPARKLE_H_
#define _BACKGROUNDSPARKLE_H_

#include <SFML/Graphics.hpp>
#include "GameObject.h"

class BackgroundSparkle : public GameObject
{
	public:
		BackgroundSparkle(sf::Vector2f _position);
		~BackgroundSparkle();
		void move(sf::Time _deltaTime);
		sf::Vector2f getPosition();
		void setPosition(sf::Vector2f _position);
		sf::CircleShape getShape();

	private:
		sf::CircleShape m_shape;
};

#endif
```

```
#pragma once
#ifndef _BULLET_H_
#define _BULLET_H_

#include <SFML/Graphics.hpp>
#include "GameObject.h"

class Bullet : public GameObject
{
	public:
		Bullet(sf::Vector2f _position, Direction _direction, int _power, float
_speed);
		~Bullet();
		Direction getDirection();
		void move(sf::Time _deltaTime);

	private:
		sf::Vector2f m_position;
		Direction m_direction;
};

#endif
```

```
#pragma once
#ifndef _BUTTON_H_
#define _BUTTON_H_

#include <SFML/Graphics.hpp>
#include <string>

class Button
{
	public:
		Button(int _action, sf::Texture &_texture, sf::Text &_text, sf::Vector2f
_position);
		~Button();
		int getAction();
		sf::Sprite getSprite();
		bool isClicked(sf::RenderWindow &_window);
```

```
		void isHovered(sf::RenderWindow &_window);

	private:
		int m_action;
		sf::Sprite m_sprite;
};

#endif


#include "Bullet.h"
#include "GameObject.h"

enum EnemyType
{
	m_weak, m_normal, m_strong
};

class Enemy : public GameObject
{
	public:
		Enemy();
		~Enemy();
		EnemyType getEnemyType();
		int getHealth();
		bool isHit(Bullet &_bullet);
		void move(sf::Time _deltaTime);
		void setHealth(int _health);
		std::vector<Bullet*> shoot();

	private:
		int m_health;
		float m_bulletSpeed;
		EnemyType m_enemyType;
};

#endif


#pragma once
#ifndef _EXPLOSION_H_
#define _EXPLOSION_H_

#include <SFML/Graphics.hpp>
#include "GameObject.h"

class Explosion : public GameObject
{
	public:
		Explosion(sf::Vector2f _position);
		~Explosion();
		bool update(int _timer);

	private:
		int m_frame;
		sf::IntRect m_textureRect;
};

#endif


#pragma once
#ifndef _GAMELOOP_H_
#define _GAMELOOP_H_

#include <SFML/Graphics.hpp>

void GameLoop(sf::RenderWindow &_window);

#endif
```

```
#pragma once
#ifndef _GAMEOBJECT_H_
#define _GAMEOBJECT_H_

#include <SFML/Graphics.hpp>

enum Direction
{
	m_left, m_right, m_up, m_down
};

class GameObject
{
	public:
	sf::Sprite getSprite();
	sf::Texture getTexture();
	protected:
		sf::Sprite m_sprite;
		sf::Texture m_texture;
		float m_speed;
};

#endif


#pragma once
#ifndef _GAMEOVERSCREEN_H_
#define _GAMEOVERSCREEN_H_

#include <SFML/Graphics.hpp>

void GameOverScreen(sf::RenderWindow &_window, int _score);

#endif


#pragma once
#ifndef _INSTRUCTIONS_H_
#define _INSTRUCTIONS_H_

#include <SFML/Graphics.hpp>

void Instructions(sf::RenderWindow &_window);

#endif

#pragma once
#ifndef _INTERFACE_H_
#define _INTERFACE_H_

#include <SFML/Graphics.hpp>
#include <string>
#include "Enemy.h"
#include "Player.h"

class Interface
{
	public:
		Interface(sf::Font &_font);
		~Interface();
		void addScore(Enemy &_enemy);
		void draw(sf::RenderWindow &_window);
		int getScore();
		void update(Player &_player);
	private:
		sf::RectangleShape m_health;
		sf::RectangleShape m_speed;
		sf::RectangleShape m_power;
		sf::Sprite m_healthSprite;
		sf::Sprite m_speedSprite;
```

```
		sf::Sprite m_powerSprite;
		sf::Texture m_healthTexture;
		sf::Texture m_speedTexture;
		sf::Texture m_powerTexture;
		sf::Text m_scoreText;
		std::string m_score;
		int m_scoreValue;
		int m_healthValue;
		float m_speedValue;
		int m_powerValue;
};

#endif


#pragma once
#ifndef _MENU_H_
#define _MENU_H_

#include <SFML/Graphics.hpp>

int MainMenu(sf::RenderWindow &_window);

#endif


#pragma once
#ifndef _PLAYER_H_
#define _PLAYER_H_

#include <SFML/Graphics.hpp>
#include "Bullet.h"
#include "GameObject.h"

class Player : public GameObject
{
	public:
		Player(sf::Texture &_texture, sf::Vector2f _position);
		~Player();
		void move(sf::Time _deltaTime, Direction _direction);
		std::vector<Bullet*> shoot();
		int getHealth();
		int getPower();
		float getSpeed();
		bool isHit(Bullet &_bullet);
		void setHealth(int _health);
		void setPower(int _power);
		void setSpeed(float _speed);

	private:
		int m_health;
		int m_power;
		float m_bulletSpeed;
		float m_speedFactor;
};

#endif


#pragma once
#ifndef _POWERUP_H_
#define _POWERUP_H_

#include <SFML/Graphics.hpp>
#include "GameObject.h"
#include "Player.h"

enum PowerupType
{
	m_health, m_power, m_haste
```

```
};

class Powerup : public GameObject
{
	public:
		Powerup(sf::Vector2f _position, Player &_player);
		~Powerup();
		bool isPickedUp(Player &_player);
		void move(sf::Time _deltaTime);
	private:
		PowerupType m_powerupType;
		bool m_pickedUp;
};

#endif

#pragma once
#ifndef _SETORIGINTOCENTER_H_
#define _SETORIGINTOCENTER_H_

#include <SFML/Graphics.hpp>

void SetOriginToCenter(sf::Sprite &_sprite);
void SetOriginToCenter(sf::Text &_text);
void SetOriginToCenter(sf::RectangleShape &_shape);

#endif

#include "BackgroundSparkle.h"

BackgroundSparkle::BackgroundSparkle(sf::Vector2f _position)
{
	m_shape.setPosition(_position);
	m_shape.setRadius(5.f);
	m_shape.setFillColor(sf::Color(255, 255, 255, 194));
	m_speed =	150.f;
}

BackgroundSparkle::~BackgroundSparkle()
{

}

void BackgroundSparkle::setPosition(sf::Vector2f _position)
{
	m_shape.setPosition(_position);
}

sf::Vector2f BackgroundSparkle::getPosition()
{
	return m_shape.getPosition();
}

sf::CircleShape BackgroundSparkle::getShape()
{
	return m_shape;
}

void BackgroundSparkle::move(sf::Time _deltaTime)
{
	m_shape.move(0, m_speed * _deltaTime.asSeconds());
}

#include "Bullet.h"
#include "SetOriginToCenter.h"
#include <iostream>

Bullet::Bullet(sf::Vector2f _position, Direction _direction, float _speed)
{
	m_speed = _speed;
```

```
	m_direction = _direction;

	if (_direction == m_up)
	{
		m_texture.loadFromFile("Textures/bullet.png");
	}
	else if (_direction == m_down)
	{
		m_texture.loadFromFile("Textures/enemyBullet.png");
	}

	sf::Sprite temp(m_texture);
	SetOriginToCenter(temp);

	temp.setPosition(_position);

	m_sprite = temp;
}

Bullet::~Bullet()
{
}

Direction Bullet::getDirection()
{
	return m_direction;
}

void Bullet::move(sf::Time _deltaTime)
{
	if (m_direction == m_up)
	{
		m_sprite.move(0, -m_speed * _deltaTime.asSeconds());
	}

	else if (m_direction == m_down)
	{
		m_sprite.move(0, m_speed * _deltaTime.asSeconds());
	}

	else
	{
		std::cout << "Something has gone horribly wrong!\n";
		std::cout << "Check bullet::move in bullet.cpp\n";
	}
}

#include "Button.h"
#include "SetOriginToCenter.h"

Button::Button(int _action, sf::Texture &_texture, sf::Text &_text, sf::Vector2f _position)
{
	m_action = _action;

	SetOriginToCenter(_text);

	sf::Sprite button(_texture);
	SetOriginToCenter(button);

	button.setPosition(_position);
	_text.setPosition(_position);

	m_sprite = button;
}

Button::~Button()
{

}
```

```
int Button::getAction()
{
	return m_action;
}

sf::Sprite Button::getSprite()
{
	return m_sprite;
}

bool Button::isClicked(sf::RenderWindow &_window)
{
	if
(m_sprite.getGlobalBounds().contains(_window.mapPixelToCoords(sf::Mouse::getPosition(_windo
w))))
	{
		return true;
	}
	else
	{
		return false;
	}
}

void Button::isHovered(sf::RenderWindow &_window)
{
	if
(m_sprite.getGlobalBounds().contains(_window.mapPixelToCoords(sf::Mouse::getPosition(_windo
w))))
	{
		m_sprite.setColor(sf::Color(255, 255, 255, 128));
	}
	else
	{
		m_sprite.setColor(sf::Color(255, 255, 255, 200));
	}
}

#include "Enemy.h"
#include "SetOriginToCenter.h"
#include <cstdlib>
#include <ctime>

Enemy::Enemy()
{
	srand(time(NULL));
	int temp = 1 + rand() % 10;

	if (temp <= 4)
	{
		m_enemyType = m_weak;
	}
	else if (temp > 4 && temp <= 7)
	{
		m_enemyType = m_normal;
	}
	else if (temp > 7)
	{
		m_enemyType = m_strong;
	}

	if (m_enemyType == m_weak)
	{
		m_texture.loadFromFile("Textures/alien1.png");
		m_health = 3;
		m_speed = 225.f;
		m_bulletSpeed = 400.f;
	}
	else if (m_enemyType == m_normal)
	{
```

```
			m_texture.loadFromFile("Textures/alien2.png");
			m_health = 6;
			m_speed = 175.f;
			m_bulletSpeed = 400.f;
		}
		else if (m_enemyType == m_strong)
		{
			m_texture.loadFromFile("Textures/alien3.png");
			m_health = 10;
			m_speed = 100.f;
			m_bulletSpeed = 400.f;
		}
		m_sprite.setTexture(m_texture);

		temp = 300 + rand() % 725;


		m_sprite.setPosition(sf::Vector2f(temp, -200));
		SetOriginToCenter(m_sprite);
}

Enemy::~Enemy()
{

}

EnemyType Enemy::getEnemyType()
{
		return m_enemyType;
}

int Enemy::getHealth()
{
		return m_health;
}

bool Enemy::isHit(Bullet &_bullet)
{
		if (_bullet.getSprite().getGlobalBounds().intersects(m_sprite.getGlobalBounds()) &&
_bullet.getDirection() != m_down)
		{
			m_health -= 1;
			return true;
		}
		else
		{
			return false;
		}
}

void Enemy::move(sf::Time _deltaTime)
{
		m_sprite.move(0, m_speed * _deltaTime.asSeconds());
}

void Enemy::setHealth(int _health)
{
		m_health = _health;
}

std::vector<Bullet*> Enemy::shoot()
{
		std::vector<Bullet*> createdBullets;
		if (this->m_enemyType == m_weak)
		{
			Bullet* bullet = new Bullet(sf::Vector2f(m_sprite.getPosition().x,
m_sprite.getPosition().y + 60), m_down, m_bulletSpeed);
			createdBullets.push_back(bullet);
		}
		else if (this->m_enemyType == m_normal)
```

```
	{
		Bullet* bullet1 = new Bullet(sf::Vector2f(m_sprite.getPosition().x - 30,
m_sprite.getPosition().y + 60), m_down, m_bulletSpeed);
		Bullet* bullet2 = new Bullet(sf::Vector2f(m_sprite.getPosition().x + 30,
m_sprite.getPosition().y + 60), m_down, m_bulletSpeed);
		createdBullets.push_back(bullet1);
		createdBullets.push_back(bullet2);
	}
	else if (this->m_enemyType == m_strong)
	{
		Bullet* bullet1 = new Bullet(sf::Vector2f(m_sprite.getPosition().x-50,
m_sprite.getPosition().y + 60), m_down, m_bulletSpeed);
		Bullet* bullet2 = new Bullet(sf::Vector2f(m_sprite.getPosition().x,
m_sprite.getPosition().y + 60), m_down, m_bulletSpeed);
		Bullet* bullet3 = new Bullet(sf::Vector2f(m_sprite.getPosition().x+50,
m_sprite.getPosition().y + 60), m_down, m_bulletSpeed);
		createdBullets.push_back(bullet1);
		createdBullets.push_back(bullet2);
		createdBullets.push_back(bullet3);
	}
	return createdBullets;
}

#include "Explosion.h"
#include "SetOriginToCenter.h"

Explosion::Explosion(sf::Vector2f _position)
{
	m_texture.loadFromFile("Textures/explosion.png");
	m_sprite.setTexture(m_texture);
	m_sprite.setTextureRect(sf::IntRect(0, 640, 128, 128));
	m_sprite.scale(2, 2);
	m_sprite.setPosition(_position);
	SetOriginToCenter(m_sprite);
	m_frame = 1;
}

Explosion::~Explosion()
{

}

bool Explosion::update(int _timer)
{
	if (m_frame <= 5 && !(_timer % 40))
	{
		m_frame++;
		m_sprite.setTextureRect(sf::IntRect(m_frame * 128, 0, 128, 128));
		return true;
	}
	else if (m_frame <= 5)
	{
		m_sprite.setTextureRect(sf::IntRect(m_frame * 128, 0, 128, 128));
		return true;
	}
	else
	{
		return false;
	}
}

#include "GameLoop.h"
#include "BackgroundSparkle.h"
#include "Player.h"
#include "Enemy.h"
#include "Interface.h"
#include "GameOverScreen.h"
#include "Powerup.h"
#include "Explosion.h"
#include <SFML/Graphics.hpp>
```

```
#include <SFML/Audio.hpp>
#include <ctime>
#include <cstdlib>
#include <vector>

void GameLoop(sf::RenderWindow &_window)
{
	srand(time(NULL));
	sf::Clock clock;
	sf::SoundBuffer fireBuffer, powerupBuffer, hitBuffer, explosionBuffer;
	fireBuffer.loadFromFile("Sounds/fire.wav");
	sf::Sound fire1, fire2, powerup, hit1, hit2, explosion;
	fire1.setBuffer(fireBuffer);
	fire2.setBuffer(fireBuffer);

	powerupBuffer.loadFromFile("Sounds/powerup.wav");
	powerup.setBuffer(powerupBuffer);

	hitBuffer.loadFromFile("Sounds/hit.flac");
	hit1.setBuffer(hitBuffer);
	hit2.setBuffer(hitBuffer);

	explosionBuffer.loadFromFile("Sounds/explosion.wav");
	explosion.setBuffer(explosionBuffer);

	sf::Music gameMusic;
	gameMusic.openFromFile("Music/gameMusic.ogg");

	gameMusic.play();
	gameMusic.setLoop(true);

	bool quitGame = false;
	bool gameOver = false;
	bool repeatGame = false;

	int endScore;

	sf::Font agency;
	agency.loadFromFile("Fonts/agency.ttf");

	sf::FloatRect boundaries(sf::Vector2f(0, 0), sf::Vector2f(1280, 930));

	int eventTimer = 0;

	sf::Time deltaTime = sf::seconds(1.0f);
	sf::Texture playerTexture;
	playerTexture.loadFromFile("Textures/player.png");

	sf::Texture gameBkgTexture;
	gameBkgTexture.loadFromFile("Textures/gameBackground.jpg");

	sf::Sprite gameBackground(gameBkgTexture);
	gameBackground.setColor(sf::Color(255, 255, 255, 128));

	Interface hud(_window, agency);
	Player *player = new Player(playerTexture, sf::Vector2f(640, 867));

	sf::Texture columnTexture;
	columnTexture.loadFromFile("Textures/column.png");

	sf::Sprite column1(columnTexture);
	sf::Sprite column2(columnTexture);

	column2.setPosition(sf::Vector2f(1088, 0));

	std::vector<BackgroundSparkle> sparkles;
	std::vector<Bullet*> enemyBullets;
	std::vector<Bullet*> playerBullets;
	std::vector<Enemy*> enemies;
	std::vector<Powerup*> powerups;
```

```
	std::vector<Explosion*> explosions;

	while (!quitGame)
	{
		for (int i = 0; i < 70; i++)
		{
			int tempx = rand() % 866 + 212;
			int tempy = rand() % 960;
			BackgroundSparkle temp(sf::Vector2f(tempx, tempy));
			sparkles.push_back(temp);
		}

		while (_window.isOpen() && !gameOver)
		{
			sf::Event event;

			while (_window.pollEvent(event))
			{
				if (event.type == sf::Event::Closed)
				{
					_window.close();
					return;
				}
			}

			_window.clear();
			_window.draw(gameBackground);

			for (std::vector<BackgroundSparkle>::iterator itr = sparkles.begin();
itr != sparkles.end(); itr++)
			{
				if ((*itr).getPosition().y > 960)
				{
					sf::Vector2f temp = (*itr).getPosition();
					temp.x += (rand() % 40) - 20;
					temp.y = -12;
					temp.y += (rand() % 3) - 2;
					(*itr).setPosition(temp);
				}
				_window.draw((*itr).getShape());
				(*itr).move(deltaTime);
			}

			_window.draw(column1);
			_window.draw(column2);
			hud.update(*player);
			hud.draw(_window);

			_window.draw(player->getSprite());

			if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left) && !player-
>getSprite().getGlobalBounds().intersects(column1.getGlobalBounds()))
			{
				player->move(deltaTime, m_left);
			}

			else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right) && !player-
>getSprite().getGlobalBounds().intersects(column2.getGlobalBounds()))
			{
				player->move(deltaTime, m_right);
			}

			if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up) && player-
>getSprite().getGlobalBounds().top > (_window.getSize().y - 600))
			{
				player->move(deltaTime, m_up);
			}

			else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down) && player-
>getSprite().getGlobalBounds().intersects(boundaries))
```

```
                {
                        player->move(deltaTime, m_down);
                }

                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))
                {
                        if (playerBullets.size() < (int)(10 * player->getPower()) &&
!(eventTimer % (20 / (int)player->getSpeed())))
                        {
                                fire1.play();
                                std::vector<Bullet*> shotBullets = player->shoot();
                                playerBullets.insert(playerBullets.end(),
shotBullets.begin(), shotBullets.end());
                        }
                }

                std::vector<Enemy*>::iterator enmitr = enemies.begin();
                std::vector<Bullet*>::iterator pbltitr = playerBullets.begin();

                for (enmitr; enmitr != enemies.end();)
                {
                        pbltitr = playerBullets.begin();
                        for (pbltitr; pbltitr != playerBullets.end();)
                        {
                                if ((**enmitr).isHit(**pbltitr))
                                {
                                        hit1.play();
                                        delete *pbltitr;
                                        pbltitr = playerBullets.erase(pbltitr);
                                }
                                else
                                {
                                        pbltitr++;
                                }
                        }

                        if ((**enmitr).getSprite().getPosition().y-20 >
_window.getSize().y)
                        {
                                delete *enmitr;
                                enmitr = enemies.erase(enmitr);
                        }

                        else if
((**enmitr).getSprite().getGlobalBounds().intersects(player-
>getSprite().getGlobalBounds()))
                        {
                                explosion.play();
                                player->setHealth(0);
                                (**enmitr).setHealth(0);
                                delete *enmitr;
                                enmitr = enemies.erase(enmitr);
                        }

                        else if ((**enmitr).getHealth() <= 0)
                        {
                                explosion.play();
                                explosions.push_back(new
Explosion((**enmitr).getSprite().getPosition()));
                                hud.addScore(**enmitr);
                                int random = 1 + rand() % 10;
                                if (random <= 3)
                                {
                                        powerups.push_back(new
Powerup((**enmitr).getSprite().getPosition(), *(player)));
                                }
                                delete *enmitr;
                                enmitr = enemies.erase(enmitr);
                        }
```

```
                        else
                        {
                                _window.draw((**enmitr).getSprite());
                                (**enmitr).move(deltaTime);

                                if (!(eventTimer % 600))
                                {
                                        fire2.play();
                                        std::vector<Bullet*> enemyShotBullets =
(**enmitr).shoot();
                                        enemyBullets.insert(enemyBullets.end(),
enemyShotBullets.begin(), enemyShotBullets.end());
                                }
                                enmitr++;
                        }
                }

                pbltitr = playerBullets.begin();

                for (pbltitr; pbltitr != playerBullets.end();)
                {
                        if ((**pbltitr).getSprite().getPosition().y < 0)
                        {
                                delete *pbltitr;
                                pbltitr = playerBullets.erase(pbltitr);
                        }
                        else
                        {
                                _window.draw((**pbltitr).getSprite());
                                (**pbltitr).move(deltaTime);
                                pbltitr++;

                        }
                }

                if (!(eventTimer % 800))
                {
                        enemies.push_back(new Enemy);
                }

                if (eventTimer == INT_MAX)
                {
                        eventTimer = 0;
                }

                std::vector<Bullet*>::iterator ebltitr = enemyBullets.begin();

                for (ebltitr; ebltitr != enemyBullets.end();)
                {
                        if ((**ebltitr).getSprite().getPosition().y >
_window.getSize().y)
                        {
                                delete *ebltitr;
                                ebltitr = enemyBullets.erase(ebltitr);
                        }
                        else
                        {
                                _window.draw((**ebltitr).getSprite());
                                (**ebltitr).move(deltaTime);
                                if (player->isHit(**ebltitr))
                                {
                                        hit2.play();
                                        delete *ebltitr;
                                        ebltitr = enemyBullets.erase(ebltitr);
                                }
                                else
                                {
                                        ebltitr++;
                                }
                        }
```

```
                        }

                        std::vector<Powerup*>::iterator pwrupitr = powerups.begin();
                        for (pwrupitr; pwrupitr != powerups.end();)
                        {
                                if ((**pwrupitr).isPickedUp(*player))
                                {
                                        powerup.play();
                                        delete *pwrupitr;
                                        pwrupitr = powerups.erase(pwrupitr);
                                }
                                else
                                {
                                        _window.draw((**pwrupitr).getSprite());
                                        (**pwrupitr).move(deltaTime);
                                        pwrupitr++;
                                }
                        }

                        std::vector<Explosion*>::iterator explitr = explosions.begin();
                        for (explitr; explitr != explosions.end();)
                        {
                                if (!(**explitr).update(eventTimer))
                                {
                                        delete *explitr;
                                        explitr = explosions.erase(explitr);
                                }
                                else
                                {
                                        _window.draw((**explitr).getSprite());
                                        explitr++;
                                }
                        }

                        deltaTime = clock.restart();
                        eventTimer++;
                        _window.display();


                        if (player->getHealth() <= 0 || hud.getScore() > 300000)
                        {
                                gameOver = true;
                                endScore = hud.getScore();
                        }
                }

                gameMusic.stop();
                GameOverScreen(_window, endScore);
                quitGame = true;

        }

        delete player;
        sparkles.clear();
        enemyBullets.clear();
        playerBullets.clear();
        enemies.clear();
        powerups.clear();

}

#include "GameObject.h"

sf::Sprite GameObject::getSprite()
{
        return m_sprite;
}

sf::Texture GameObject::getTexture()
{
```

```
	return m_texture;
}

#include "Button.h"
#include "GameOverScreen.h"
#include "SetOriginToCenter.h"
#include <string>

void GameOverScreen(sf::RenderWindow &_window, int _score)
{
	sf::Sprite background;
	sf::Texture btnTexture, lostBackground, wonBackground;
	sf::Font agency;
	lostBackground.loadFromFile("Textures/lostBackground.png");
	wonBackground.loadFromFile("Textures/wonBackground.png");
	agency.loadFromFile("Fonts/agency.ttf");
	btnTexture.loadFromFile("Textures/button.png");

	_window.clear();
	if (_score > 300000)
	{
		background.setTexture(wonBackground);
	}
	else
	{
		background.setTexture(lostBackground);
	}

	std::string scoreText("Score: ");

	scoreText += std::to_string(_score);
	sf::Text score(scoreText, agency, 48);
	SetOriginToCenter(score);

	score.setPosition(sf::Vector2f(640, 275));

	sf::Text buttonText("Quit to Menu", agency, 36);
	Button button(100, btnTexture, buttonText, sf::Vector2f(640, 800));

	std::vector<sf::Text> textList;

	textList.push_back(score);
	textList.push_back(buttonText);

	while (_window.isOpen())
	{
		sf::Event event;
		while (_window.pollEvent(event))
		{
			if (event.type == sf::Event::Closed)
			{
				_window.close();
			}

			else if (event.type == sf::Event::MouseButtonPressed &&
event.mouseButton.button == sf::Mouse::Left)
			{
				if (button.isClicked(_window))
				{
					return;
				}
			}
			button.isHovered(_window);

			_window.clear();
			_window.draw(background);

			_window.draw(button.getSprite());

			for (int i = 0; i < 2; i++)
```

```
                        {
                                _window.draw(textList[i]);
                        }

                        _window.display();
                }
        }
}

#include "Button.h"
#include "Instructions.h"
#include "SetOriginToCenter.h"

void Instructions(sf::RenderWindow &_window)
{
        sf::Font agency;
        agency.loadFromFile("Fonts/agency.ttf");
        sf::Text exitText("Exit to Menu", agency, 36);

        SetOriginToCenter(exitText);
        exitText.setPosition(sf::Vector2f(525, 880));

        sf::Texture btnTexture;
        btnTexture.loadFromFile("Textures/button.png");

        Button exitToMenu(0, btnTexture, exitText, sf::Vector2f(640, 800));

        sf::Texture menuBkgTexture;
        menuBkgTexture.loadFromFile("Textures/menuBackground.png");
        sf::Sprite background(menuBkgTexture);

        sf::Texture instructionsTexture;
        instructionsTexture.loadFromFile("Textures/instructions.png");
        sf::Sprite instructionsSprite(instructionsTexture);

        while (_window.isOpen())
        {
                sf::Event event;
                while (_window.pollEvent(event))
                {
                        if (event.type == sf::Event::Closed)
                        {
                                _window.close();
                        }
                        else if (event.type == event.MouseButtonPressed &&
event.mouseButton.button == sf::Mouse::Left)
                        {
                                if (exitToMenu.isClicked(_window))
                                {
                                        return;
                                }
                        }

                        exitToMenu.isHovered(_window);

                        _window.clear();
                        _window.draw(background);
                        _window.draw(instructionsSprite);
                        _window.draw(exitToMenu.getSprite());
                        _window.draw(exitText);
                        _window.display();
                }
        }
}

#include "Interface.h"
#include "SetOriginToCenter.h"
#include <iostream>

Interface::Interface(sf::Font &_font)
```

```
{
	sf::Text temp("0", _font, 60);
	temp.setOutlineThickness(4.f);
	sf::FloatRect boundsRect;

	SetOriginToCenter(temp);
	temp.setPosition(sf::Vector2f(1130, 72));

	m_scoreText = temp;
	m_score = m_scoreText.getString();
	m_scoreValue = std::stoi(m_score);

	m_health.setSize(sf::Vector2f(128, 720));
	boundsRect = m_health.getLocalBounds();
	m_health.setOrigin(boundsRect.left + boundsRect.width / 2.0f, boundsRect.height);
	m_health.setPosition(sf::Vector2f(96, 90));
	m_health.rotate(180);
	m_health.setFillColor(sf::Color(255, 0, 0, 255));

	m_power.setSize(sf::Vector2f(72, 144));
	boundsRect = m_power.getLocalBounds();
	m_power.setOrigin(boundsRect.left + boundsRect.width / 2.0f, boundsRect.height);
	m_power.setPosition(sf::Vector2f(1140, 750));
	m_power.rotate(180);


	m_power.setFillColor(sf::Color(255, 255, 0, 255));

	m_speed.setSize(sf::Vector2f(72, 144));
	boundsRect = m_speed.getLocalBounds();
	m_speed.setOrigin(boundsRect.left + boundsRect.width / 2.0f, boundsRect.height);
	m_speed.setPosition(sf::Vector2f(1228, 750));
	m_speed.rotate(180);


	m_speed.setFillColor(sf::Color(0, 255, 255));

	sf::Texture tempTex;
	tempTex.loadFromFile("Textures/health.png");
	m_healthTexture = tempTex;
	tempTex.loadFromFile("Textures/power.png");
	m_powerTexture = tempTex;
	tempTex.loadFromFile("Textures/speed.png");
	m_speedTexture = tempTex;

	m_healthSprite.setTexture(m_healthTexture);
	SetOriginToCenter(m_healthSprite);

	m_healthSprite.setPosition(sf::Vector2f(96, 930));

	m_powerSprite.setTexture(m_powerTexture);
	SetOriginToCenter(m_powerSprite);

	m_powerSprite.setPosition(sf::Vector2f(1140, 930));

	m_speedSprite.setTexture(m_speedTexture);
	SetOriginToCenter(m_speedSprite);

	m_speedSprite.setPosition(sf::Vector2f(1228, 930));
}


Interface::~Interface()
{

}

void Interface::addScore(Enemy &_enemy)
{
	if (_enemy.getEnemyType() == m_weak)
```

```
	{
		m_scoreValue += 3000;
	}

	if (_enemy.getEnemyType() == m_normal)
	{
		m_scoreValue += 5000;
	}

	if (_enemy.getEnemyType() == m_strong)
	{
		m_scoreValue += 10000;
	}

	m_score = std::to_string(m_scoreValue);
	m_scoreText.setString(m_score);
}

void Interface::draw(sf::RenderWindow &_window)
{
	_window.draw(m_health);
	_window.draw(m_speed);
	_window.draw(m_power);
	_window.draw(m_healthSprite);
	_window.draw(m_speedSprite);
	_window.draw(m_powerSprite);
	_window.draw(m_scoreText);
}

int Interface::getScore()
{
	return m_scoreValue;
}

void Interface::update(Player &_player)
{
	m_healthValue = _player.getHealth();
	m_powerValue = _player.getPower();
	m_speedValue = _player.getSpeed();

	switch (m_healthValue)
	{
		case 3:
		{
			m_health.setSize(sf::Vector2f(128, 720));
			break;
		}
		case 2:
		{
			m_health.setSize(sf::Vector2f(128, 480));
			break;
		}
		case 1:
		{
			m_health.setSize(sf::Vector2f(128, 240));
			break;
		}
		case 0:
		{
			m_health.setSize(sf::Vector2f(0, 0));
			break;
		}
		default:
		{
			std::cout << "The health bar receives invalid values from _player.\n";
			std::cout << "Check Interface.cpp -> update\n";
			break;
		}
	}
```

```
	switch (m_powerValue)
	{
		case 5:
		{
			m_power.setSize(sf::Vector2f(72, 720));
			break;
		}
		case 4:
		{
			m_power.setSize(sf::Vector2f(72, 576));
			break;
		}
		case 3:
		{
			m_power.setSize(sf::Vector2f(72, 432));
			break;
		}
		case 2:
		{
			m_power.setSize(sf::Vector2f(72, 288));
			break;
		}
		case 1:
		{
			m_power.setSize(sf::Vector2f(72, 144));
			break;
		}
		default:
		{
			std::cout << "The power bar receives invalid values from _player.\n";
			std::cout << "Check Interface.cpp -> update\n";
			break;
		}
	}
	int temp = m_speedValue * 10;
	switch (temp)
	{
		case 20:
		{
			m_speed.setSize(sf::Vector2f(72, 720));
			break;
		}
		case 15:
		{
			m_speed.setSize(sf::Vector2f(72, 432));
			break;
		}
		case 10:
		{
			m_speed.setSize(sf::Vector2f(72, 144));
			break;
		}
		default:
		{
			std::cout << "The power bar receives invalid values from _player.\n";
			std::cout << "Check Interface.cpp -> update\n";
			break;
		}
	}
}

#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include "Instructions.h"
#include "Menu.h"
#include "GameLoop.h"

int main()
{
	int choice;
```

```
	bool quitGame = false;

	sf::Music intro;
	intro.openFromFile("Music/intro.ogg");

	intro.play();
	intro.setLoop(true);

	sf::RenderWindow window(sf::VideoMode(1280, 960), "Aliens From Outer Space!");
	window.setPosition(sf::Vector2i(310, 10));
	while (window.isOpen() && !quitGame)
	{
		sf::Event event;
		while (window.pollEvent(event))
		{
			if (event.type == sf::Event::Closed)
			{
				window.close();
			}

		}
		choice = MainMenu(window);
		switch (choice)
		{
			case 1:
			{
				intro.stop();
				GameLoop(window);
				intro.play();
				break;
			}
			case 2:
			{
				Instructions(window);
				break;
			}
			case 3:
			{
				intro.stop();
				quitGame = true;
				break;
			}
		}
	}
	if (window.isOpen())
	{
		window.close();
	}
	return 0;
}

#include "Button.h"
#include "Menu.h"
#include "GameLoop.h"
#include "SetOriginToCenter.h"
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <vector>

int MainMenu(sf::RenderWindow &_window)
{
	sf::Font agency;
	agency.loadFromFile("Fonts/agency.ttf");
	sf::Texture menuBkgTexture, btnTexture;

	btnTexture.loadFromFile("Textures/button.png");
	menuBkgTexture.loadFromFile("Textures/menuBackground.png");
	sf::Sprite background(menuBkgTexture);

	sf::Text title("Aliens from Outer Space!", agency, 78);
```

```
	title.setOutlineThickness(8);
	SetOriginToCenter(title);
	title.setPosition(sf::Vector2f(640, 400));

	sf::Text buttonText1("Play", agency, 36);
	sf::Text buttonText2("Instructions", agency, 36);
	sf::Text buttonText3("Quit Game", agency, 36);
	Button button1(1, btnTexture, buttonText1, sf::Vector2f(640, 550));
	Button button2(2, btnTexture, buttonText2, sf::Vector2f(640, 650));
	Button button3(3, btnTexture, buttonText3, sf::Vector2f(640, 750));

	std::vector<Button> buttonList;
	std::vector<sf::Text> textList;

	buttonList.push_back(button1);
	buttonList.push_back(button2);
	buttonList.push_back(button3);

	textList.push_back(title);
	textList.push_back(buttonText1);
	textList.push_back(buttonText2);
	textList.push_back(buttonText3);

	while (_window.isOpen())
	{
		sf::Event event;
		while (_window.pollEvent(event))
		{
			if (event.type == sf::Event::Closed)
			{
				_window.close();
				return 3;
			}


			else if (event.type == sf::Event::MouseButtonPressed &&
event.mouseButton.button == sf::Mouse::Left)
			{
				for (int i = 0; i < 3; i++)
				{
					if (buttonList[i].isClicked(_window))
					{
						return buttonList[i].getAction();
					}
				}
			}
		}

			for (int i = 0; i < 3; i++)
			{
				buttonList[i].isHovered(_window);
			}

			_window.clear();
			_window.draw(background);

			for (int i = 0; i < 3; i++)
			{
				_window.draw(buttonList[i].getSprite());
			}

			for (int i = 0; i < 4; i++)
			{
				_window.draw(textList[i]);
			}
			_window.display();

		}
	}
```

```cpp
#include "Player.h"
#include "SetOriginToCenter.h"
#include <iostream>

Player::Player(sf::Texture &_texture, sf::Vector2f _position)
{
	m_sprite.setTexture(_texture);
	SetOriginToCenter(m_sprite);
	m_sprite.setPosition(_position);
	m_sprite.scale(0.66, 0.66);

	m_health = 3;
	m_power = 1;
	m_speed = 300.0f;
	m_bulletSpeed = 400.0f;
	m_speedFactor = 1.0f;
}

Player::~Player()
{

}

void Player::move(sf::Time _deltaTime, Direction _direction)
{
	switch (_direction)
	{
		case m_up:
		{
			m_sprite.move(0, -m_speed * m_speedFactor * _deltaTime.asSeconds());
			break;
		}
		case m_down:
		{
			m_sprite.move(0, m_speed * m_speedFactor * _deltaTime.asSeconds());
			break;
		}
		case m_left:
		{
			m_sprite.move(-m_speed * m_speedFactor * _deltaTime.asSeconds(), 0);
			break;
		}
		case m_right:
		{
			m_sprite.move(m_speed * m_speedFactor * _deltaTime.asSeconds(), 0);
			break;
		}
		default:
		{
			std::cout << "Something has gone horribly wrong!\n";
			std::cout << "Check the enums for player.cpp.\n";
			break;
		}
	}
}

int Player::getHealth()
{
	return m_health;
}

int Player::getPower()
{
	return m_power;
}

float Player::getSpeed()
{
	return m_speedFactor;
}
```

```
bool Player::isHit(Bullet &_bullet)
{
	if (_bullet.getSprite().getGlobalBounds().intersects(m_sprite.getGlobalBounds()) &&
_bullet.getDirection() != m_up)
	{
		m_health -= 1;
		if (m_speedFactor != 1.f)
		{
			m_speedFactor -= 0.5;
		}
		if (m_power != 1)
		{
			m_power -= 1;
		}
		return true;
	}
	else
	{
		return false;
	}
}

std::vector<Bullet*> Player::shoot()
{
	std::vector<Bullet*> createdBullets;
	switch (m_power)
	{
		case 1:
		{
			Bullet* bullet = new Bullet(sf::Vector2f(m_sprite.getPosition().x,
m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			createdBullets.push_back(bullet);
			break;
		}
		case 2:
		{
			Bullet* bullet1 = new Bullet(sf::Vector2f(m_sprite.getPosition().x -
10, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			Bullet* bullet2 = new Bullet(sf::Vector2f(m_sprite.getPosition().x +
10, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			createdBullets.push_back(bullet1);
			createdBullets.push_back(bullet2);
			break;
		}
		case 3:
		{
			Bullet* bullet1 = new Bullet(sf::Vector2f(m_sprite.getPosition().x -
20, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			Bullet* bullet2 = new Bullet(sf::Vector2f(m_sprite.getPosition().x,
m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			Bullet* bullet3 = new Bullet(sf::Vector2f(m_sprite.getPosition().x +
20, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			createdBullets.push_back(bullet1);
			createdBullets.push_back(bullet2);
			createdBullets.push_back(bullet3);
			break;
		}
	case 4:
		{
			Bullet* bullet1 = new Bullet(sf::Vector2f(m_sprite.getPosition().x -
30, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			Bullet* bullet2 = new Bullet(sf::Vector2f(m_sprite.getPosition().x -
10, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			Bullet* bullet3 = new Bullet(sf::Vector2f(m_sprite.getPosition().x +
10, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			Bullet* bullet4 = new Bullet(sf::Vector2f(m_sprite.getPosition().x +
30, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
			createdBullets.push_back(bullet1);
			createdBullets.push_back(bullet2);
```

```
                        createdBullets.push_back(bullet3);
                        createdBullets.push_back(bullet4);
                        break;
                }
                case 5:
                {
                        Bullet* bullet1 = new Bullet(sf::Vector2f(m_sprite.getPosition().x -
40, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
                        Bullet* bullet2 = new Bullet(sf::Vector2f(m_sprite.getPosition().x -
20, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
                        Bullet* bullet3 = new Bullet(sf::Vector2f(m_sprite.getPosition().x + 0,
m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
                        Bullet* bullet4 = new Bullet(sf::Vector2f(m_sprite.getPosition().x +
20, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
                        Bullet* bullet5 = new Bullet(sf::Vector2f(m_sprite.getPosition().x +
40, m_sprite.getPosition().y - 60), m_up, (m_bulletSpeed * m_speedFactor));
                        createdBullets.push_back(bullet1);
                        createdBullets.push_back(bullet2);
                        createdBullets.push_back(bullet3);
                        createdBullets.push_back(bullet4);
                        createdBullets.push_back(bullet5);
                        break;
                }
        }
        return createdBullets;
}

void Player::setHealth(int _health)
{
        m_health = _health;
}

void Player::setSpeed(float _speed)
{
        m_speedFactor = _speed;
}

void Player::setPower(int _power)
{
        m_power = _power;
}

#include "Powerup.h"
#include "SetOriginToCenter.h"
#include <ctime>
#include <cstdlib>

Powerup::Powerup(sf::Vector2f _position, Player &_player)
{
        m_pickedUp = false;

        srand(time(NULL));
        int random = 1 + rand() % 10;

        if (_player.getHealth() < 2)
        {
                if (random <= 2)
                {
                        m_powerupType = m_power;
                        m_texture.loadFromFile("Textures/power.png");
                }
                else if (random > 2 && random <= 8)
                {
                        m_powerupType = m_health;
                        m_texture.loadFromFile("Textures/health.png");
                }
                else if (random > 8)
                {
                        m_powerupType = m_haste;
                        m_texture.loadFromFile("Textures/speed.png");
```

```
		}
	}
	else
	{
		if (random <= 4)
		{
			m_powerupType = m_power;
			m_texture.loadFromFile("Textures/power.png");
		}
		else if (random > 4 && random <= 7)
		{
			m_powerupType = m_health;
			m_texture.loadFromFile("Textures/health.png");
		}
		else if (random > 7)
		{
			m_powerupType = m_haste;
			m_texture.loadFromFile("Textures/speed.png");
		}
	}

	m_sprite.setTexture(m_texture);

	SetOriginToCenter(m_sprite);

	m_sprite.setPosition(_position);

	m_speed = 100;
}

Powerup::~Powerup()
{

}

void Powerup::move(sf::Time _deltaTime)
{
	m_sprite.move(0, m_speed * _deltaTime.asSeconds());
}

bool Powerup::isPickedUp(Player &_player)
{
	if (m_sprite.getGlobalBounds().intersects(_player.getSprite().getGlobalBounds()) &&
!m_pickedUp)
	{
		if (m_powerupType == m_health && _player.getHealth() < 3)
		{
			_player.setHealth(_player.getHealth() + 1);
		}
		else if (m_powerupType == m_power && _player.getPower() < 5)
		{
			_player.setPower(_player.getPower() + 1);
		}
		else if (m_powerupType == m_haste && _player.getSpeed() < 2)
		{
			_player.setSpeed(_player.getSpeed() + 0.5);
		}
		m_pickedUp = true;
		return true;
	}
	else
	{
		return false;
	}
}

#include "SetOriginToCenter.h"

void SetOriginToCenter(sf::Sprite &_sprite)
{
```

```
	sf::FloatRect boundsRect = _sprite.getLocalBounds();
	_sprite.setOrigin(boundsRect.left + boundsRect.width / 2.0f, boundsRect.top +
boundsRect.height / 2.0f);
}

void SetOriginToCenter(sf::Text &_text)
{
	sf::FloatRect boundsRect = _text.getLocalBounds();
	_text.setOrigin(boundsRect.left + boundsRect.width / 2.0f, boundsRect.top +
boundsRect.height / 2.0f);
}

void SetOriginToCenter(sf::RectangleShape &_shape)
{
	sf::FloatRect boundsRect = _shape.getLocalBounds();
	_shape.setOrigin(boundsRect.left + boundsRect.width / 2.0f, boundsRect.top +
boundsRect.height / 2.0f);
}
```

5. Testing and Conclusions (Marks: 5%)

When I firstly met each function, I usually tested them out with a break point to see how does it work and what can I expect from it. For example, getLocalBounds() and getGlobalBounds() may sound quite similar but a mistake in using one of them instead of the other costed me two hours of finding where did I give the coordinates wrongly until I have found out the mistake.

The biggest problem that I encountered was with using vectors and vector iterators for bullets. I firstly created one big vector for all bullets from both the enemy and the player, but when both classes used their shoot() function the game usually broke down. After initially thinking it was fixed with creating two separate vectors, the problem still arose, which made me real confused. It turned out that the problem was hidden in checking whether an enemy was hit. While iterating through the bullets vector, I nested a while loop checking collision with each enemy for the current bullet. The solution was to swap the loops, to make every bullet check collision with the current enemy.

My initial idea was to created angled shots for the enemies, but as the development grew, the linear shots were here to stay. It makes more of a challenge in my personal opinion, as it is harder to shoot at something when there are shots incoming at the player at the same time.

6. Self-Assessment of Performance (Marks: 5%)

Tutor **:** Andrew Watson

Student's ID:	s4922675

Indicate the appropriate response:

Did I submit the assignment on time?	**Yes**	**No**		
Did I complete the assignment?	**Yes**	**No**		
If no, approx. how much did I complete	**85%**			
How happy am I with what I submitted?	**Very happy**	**Satisfied**	**Disappointed**	**Ashamed**
What mark do I expect?	**60%**			
Did I spend enough time on the assignment?	**Yes**	**No**		
Did I get it proof-read by someone else?	**Yes**	**No**		
Have I properly 'referenced' it?	**Yes**	**No**		
Could I improve the presentation?	**Yes**	**No**		

Answer the following questions:

The best part of my performance was:	The implementation of the game. Even though I never used SFML, I managed to create a game, googling only a couple of times for the documentation. And it was incredibly fun too!
The worst part of my performance was:	The design part of the report. Even though I have shown why we need the classes I used in the game loop in the analysis, I did not show the final usage of them, since I spent way too much time on diagrams and pseudocode. Shame on me.
One way in which I could improve the content of my assignment is:	Increase the analysis done between classes and modules, show why is it better to have modulated code instead of everything in one function, and explain more often why I used a class instead of a function.
One way in which I could improve the presentation of my assignment is:	Organise the chaos that has awoken upon this work. The word count got way out of hand at one point, and I should have realised that earlier.
One thing I will do to improve my performance in my next assignment is:	Start the assignment way earlier on to get myself acquittanced with the documentation of given libraries and the assignment criteria, to make sure I can mark everything as done.
Another thing I will do to improve my performance in my next assignment is:	Comment the code. I have completely forgotten about comments and got reminded by my friend one hour before the deadline.

7. References

Cormier, E., 2013. *Centering text on the screen with SFML* [online]. Stack Overflow. Available from: https://stackoverflow.com/a/15253837 [Accessed 12 May 2018].

Dundee, 2017. *How do you make a clickable sprite in SFML?* [online]. Stack Overflow. Available from: https://stackoverflow.com/a/23578981 [Accessed 12 May 2018].

MarcusM, 2013. *[Sprite] Modifying the bounds of a sprite* [online]. En.sfml-dev.org. Available from: https://en.sfml-dev.org/forums/index.php?topic=13633.0 [Accessed 12 May 2018].

Button Image: Sub Dimension Studios, not dated. Marble 125 [online]. 3DXO. Available from: http://www.3dxo.com/textures/10529_marble_125 [Accessed 12 May 2018].

Game Background Image: OpenClipart-Vectors, 2014. Deep Space [online]. Pixabay. Available from: https://pixabay.com/pl/galaktyka-deep-space-przestrzeń-575235/ [Accessed 12 May 2018].

Main Menu Image: myersalex216, 2017. Space [online]. Pixabay. Available from: https://pixabay.com/pl/przestrzeń-deep-space-galaktyka-2638126/ [Accessed 12 May 2018].

Spaceships: MillionthVector, 2013. Free Sprites [online]. Blogspot. Available from: http://millionthvector.blogspot.co.uk/p/free-sprites.html [Accessed 12 May 2018].

Bullets: Master484, 2013. M484BulletCollection1 [online]. OpenGameArt. Available from: https://opengameart.org/content/bullet-collection-1-m484 [Accessed 12 May 2018].

Explosion Sound: LittleRobotSoundFactory, 2015. 8-bit Explosion_00.wav [online]. Freesound. Available from: https://freesound.org/s/270308/ [Accessed 12 May 2018].

Hit Sound: qubodup, 2013. Damage [online]. Freesound. Available from: https://freesound.org/people/qubodup/sounds/211634/ [Accessed 12 May 2018].

Shoot Sound: wildweasel, 2007. Dual Neutron Disruptor.wav [online]. Freesound. Available from: https://freesound.org/people/wildweasel/sounds/39023/ [Accessed 12 May 2018].

Power-up Sound: freezefast65, 2018. 8-bit Powerup 2 [online]. Freesound. Available from: https://freesound.org/people/freezefast65/sounds/422090/ [Accessed 12 May 2018].

Game Music: SiriusBeat, 2013. Ep. 08: Royalty Free Music [Electronic/Technology/Sci-Fi] - Future Club [online]. SiriusBeat. Available from: https://siriusbeat.com/track/855252/ep-08-royalty-free-music-electronic-technology-sci-fi-future-club [Accessed 12 May 2018].

Menu Music: SiriusBeat, 2013. Ep. 14: Royalty Free Music [Ambient/Fantasy/New Age] - The Cosmos [online]. SiriusBeat. Available from: https://siriusbeat.com/track/855246/ep-14-royalty-free-music-ambient-fantasy-new-age-the-cosmos [Accessed 12 May 2018].