**BAHRIA UNIVERSITY**

A logo of a university

Description automatically generated

A video game screen with planets and a spaceship

Description automatically generated**ISLAMABAD E-8 CAMPUS**

**Object Oriented Programming**

**CSC-210**

**Project report BS-IT 2A**

**2D space shooter game**

# **SUBMISSION DETAILS**

## **SUBMITTED TO:**

**Ms, Nabia Khalid**

## **SUBMITTED BY:**

1. **Kashmallah Shah**

* **01-135231-041**

1. **Sami-Ur-Rehman Raja**

* **01-135231-084**



**DEPARTMENT OF COMPUTER SCIENCE**



**Contents**

[**SUBMISSION DETAILS** 2](#_Toc153058482)

[**ACKNOWLEDGMENTS** 7](#_Toc153058485)

[**ABSTRACT** 8](#_Toc153058486)

[**INTRODUCTION** 9](#_Toc153058487)

[ **DOMAIN** 9](#_Toc153058488)

[ **Background Study and Literature review** 10](#_Toc153058489)

[ **Background Study:** 10](#_Toc153058490)

[ **Literature Review** 12](#_Toc153058498)

[**Project Application Idea** 13](#_Toc153058504)

[ **Justifications** 13](#_Toc153058505)

[ **Boundary of the project** 14](#_Toc153058510)

[**Project Plan and Team** 15](#_Toc153058514)

[ **Project Plan** 15](#_Toc153058515)

[ **Responsibilities among team members** 16](#_Toc153058522)

[ **Role playing** 16](#_Toc153058523)

[**Audience of our Project** 17](#_Toc153058525)

[ **Intended Audience of your Project** 17](#_Toc153058526)

[ **Identified Stakeholders:** 18](#_Toc153058532)

[**System Design and Functional Requirements** 19](#_Toc153058536)

[ **Hardware and Software** 19](#_Toc153058537)

[ **Describe at least FIVE Functional Requirements** 19](#_Toc153058538)

[ **Mapping of the functional requirements with four pillars of OOP** 20](#_Toc153058544)

[ **Class Diagram (UML Notation) with description** 21](#_Toc153058549)

[ **Uml diagram Description** 21](#_Toc153058550)

[ **Use Case with description** 22](#_Toc153058557)

[**Conclusion** 22](#_Toc153058558)

**TABLE OF CAPTIONS**

[Figure a Global Games Market from 2012 to 2017 8](#_Toc153058662)

[Figure b Age and Gender of game players 9](#_Toc153058663)

[Figure d SPACE INVADERS 9](#_Toc153058664)

[Figure c GALAGA 9](file:///C:\Users\Administrator\Desktop\Kashmallah%20UNI\OOP%20PROJECT\BAHRIA%20UNIVERSITY.docx#_Toc153058665)

[Figure e Project Requirements 14](#_Toc153058666)

[Figure f UML/Class Diagram 18](#_Toc153058667)

[Figure g USE CASE diagram 19](#_Toc153058668)

# **ACKNOWLEDGMENTS**

Although this project was an effort of two individuals, it would not have been possible without the support of elders and friends. First, we would like to thank **Ms. Nabia Khalid** and the entire computer department for their useful suggestion and encouragement.

We would like to thank our **OBJECT-ORIENTED PROGRAMMING LECTURER** for helping us with learning Classes and Objects in C++, to our computer department and the HOD sir for their encouragement.

In addition to all the help from our teacher, we received a lot of help and support from our friends and especially our seniors. We would like to thank Muhammad Saqlain Arshad with helping us in correcting our code and Muhammad Abdullah Shah as he also helped us a lot through our work. Without their critical analysis and timely suggestions this project would not have been a success. Their suggestions and comments have been most helpful for the development of the program, and they deserve a significant part of the credit that goes to this program. We would like to thank them for sharing their ideas and knowledge without much of a problem.

**For all the people who helped us a lot, thank you very much and may ALLAH bless you all.**

Sami Ur Rehman Raja (01-135231-084)

Kashmallah Shah (01-135231-041)

# **ABSTRACT**

Nowadays, the video-game industry is one of the biggest and fastest growing around, providing jobs to many people, and having a very big market. From the **1950s** until now, many different video-game genres were created suiting different people’s tastes. Among them, Shoot ’em up games mainly consist of a player trying to complete the game while evading different obstacles which can include enemies, environmental objects, or different types of projectiles.

During our whole lives, we have spent countless hours playing a great variety of videogames. We were always interested in developing one by ourselves, which led us to consider making this project: **a 2D video game** in shoot ’em up genre, which we have named **SPACE SHOOTER.** By making **SPACE SHOOTER**, we have learned how the videogames can be designed and developed by anyone who has basic info of programming.

This process includes the usage of different tools for graphic and audio design such as **SFML libraries** as well as **Visual Studio** for combining them into a seamlessly working game. As a result, we have obtained a fully working game, gained a lot of experience and feel more motivated than ever before in continuing making games.

# **INTRODUCTION**

The Video-game industry is one of the fastest growing industries, allowing people from all around the world to play video-games of many different genres. There is a video-game for every person: from first person shooters to racing games, from fighting games to sports games, from strategy games to puzzle games, etc. Among them, this project is focused on shoot ’em up genre, which is one of my favorite ones and whose origins are explained in this report.

This Project report discusses the development of a two-dimensional space-shooter game, and the implementation of several graphical effects and techniques that were used to make the game look interesting. When developing the game we decided that, if there is a conflict between visuals and realism, visuals will be prioritized if it seems likely to result in a more engaging game. This introductory chapter covers the background, purpose, problems, limitations, and the method of working that form the basis of this project report.

## **DOMAIN**

In the domain section, we outline the context and boundaries of our space shooter game project. This game is set in the dynamic world of interactive entertainment, specifically within the action-packed, interstellar adventure genre. The gameplay involves players navigating a virtual cosmos and engaging in intense combat against extraterrestrial adversaries.

The game's subject matter centers around outer space exploration, alien encounters, and strategic maneuvers for survival and success. Key elements include spaceship dynamics, weaponry, celestial environments, and enemy behaviors. The game is designed to be immersive and captivating, with visually stimulating graphics, compelling audio effects, and a well-crafted storyline set against the backdrop of the vast cosmos.

The project's scope covers various aspects, including gameplay mechanics, graphical design, audio integration, and user interface elements. The game will feature multiple levels or stages, each with unique challenges and increasing levels of difficulty. The project also includes considerations for player progression, scoring mechanisms, and the implementation of power-ups to enhance the gaming experience.

The domain section also acknowledges the influence of existing space shooter games in the gaming industry. The game aims to offer a unique and innovative perspective within the genre, examining established conventions, identifying opportunities for differentiation, and infusing the project with creative elements to set it apart in the broader space shooter gaming landscape.

In essence, this section provides a comprehensive understanding of the thematic and operational boundaries that define the virtual universe in which the game will unfold.

## **Background Study and Literature review**

### **Background Study:**

From 2012, the global game market revenue went up over the 30% with different mobile platforms, taking up to the 34% of its market share in 2017 as can be seen in Figure 1.1. Due to its continued growth, software developing companies started making more applications for game development and, at the present time, there are many different tools that allow a single person or a very small team of people to successfully develop and sale video-games, which accelerates the market growth of this industry even more. Games are played by people of all ages and genders alike, as depicted in the charts included in Figure 1.2. This fact provides an easy way to connect parents and children, or different people around the globe. Considering this information, it is easy to see why game development is a good industry to get into.

## 

Figure a Global Games Market from 2012 to 2017

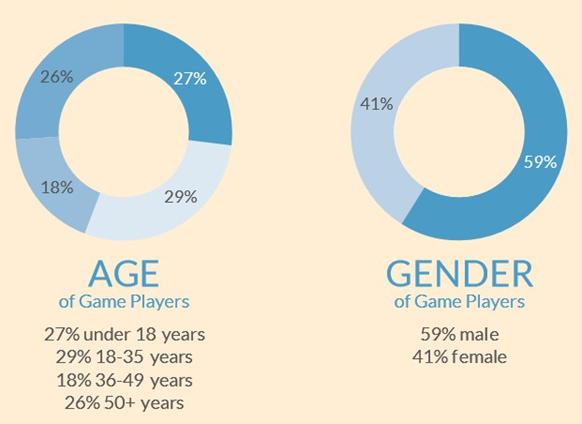


Figure b Age and Gender of game players

Some of most significant games of the 1980s included:

* **Defender 1981**: It was one of the most difficult games of its time.
* **Scramble 1981**: It was the first to offer multiple distinct levels.
* **Tempest 1981[3]:** It was the first to incorporate a 3D perspective into the genre, and it is the game that inspired a sub-genre of rail shooters.
* **Zaxxon 1982**: It was the first isometric scrolling shooter.
* **Xenovious 1982**: It became the most influential vertically scrolling shooter.
* ******Time Pilot 1982:** It became the first multi-directional shooter. 

Figure d SPACE INVADERS

Figure c GALAGA

## **Literature Review**

The literature review extends to various sources relevant to 2D game development using C++ and SFML . Key aspects include:

* ***SFML Documentation and Tutorials:*** Exploring official SFML documentation and tutorials to gain a deep understanding of the library's capabilities, features, and best practices for 2D game development. This includes insights into rendering graphics, handling user input, and managing game states.
* ***C++ Game Development Books and Face to Face Lectures:*** Reviewing literature that focuses on C++ game development, particularly books and face to face lectures that discuss design patterns, object-oriented programming principles, and optimization techniques for 2D games.
* ***BUPC Club in University:*** Engaging with BUPC club discussions, forums, and platforms where seniors share experiences, troubleshoot challenges, and provide insights into effectively using SFML for 2D game development.
* ***2D Space Shooter Game Post-Mortems:*** Analyzing post-mortems of other 2D space shooter games developed using similar technologies. Understanding the challenges faced, design decisions made, and lessons learned by other developers in the genre provides valuable guidance for our project.
* ***Game Design Principles for 2D Games:*** Exploring literature that outlines effective game design principles specific to 2D games. This includes considerations for level design, player progression, enemy behavior, and overall player engagement in the context of a 2D space shooter.

By synthesizing knowledge from these sources, we aim to build a strong foundation for the development of our 2D space shooter game. This literature review will inform our design choices, help us avoid common pitfalls, and guide us toward implementing a compelling and technically sound gaming experience.

# **Project Application Idea**

The project application idea centers around the development of an engaging 2D space shooter game using C++ and SFML. The core concept involves placing players in the cockpit of a customizable spaceship, navigating through a visually dynamic interstellar environment, and engaging in intense battles against waves of alien adversaries. The game aims to provide an immersive and enjoyable experience for players who appreciate classic arcade-style space shooter gameplay.

## **Justifications**

* ***Classic Appeal:*** The 2D space shooter genre holds a timeless appeal, reminiscent of classic arcade games. Leveraging this nostalgia, our project aims to capture the essence of the genre while incorporating modern design elements to cater to both seasoned gamers and a new generation of players.
* ***Accessible Development:*** Choosing C++ and SFML for game development offers a balanced combination of performance and accessibility. C++ provides low-level control and efficiency, while SFML simplifies the process of handling graphics, input, and audio. This choice ensures that the development process is approachable for both novice and experienced developers.
* ***Learning Opportunities:*** The project provides an excellent learning opportunity for developers interested in game development. By utilizing widely-used languages and frameworks like C++ and SFML, individuals can enhance their programming skills while gaining insights into game design, physics simulation, and user interface development.
* ***Customization and Upgrades:*** The inclusion of a customizable spaceship and the potential for in-game upgrades introduce strategic elements to the gameplay. This justifies the project by offering players

a sense of progression and personalization, contributing to increased

player engagement and satisfaction.

## **Boundary of the project**

Defining the boundaries of the 2D space shooter game project is essential to manage expectations and scope effectively.

* ***Gameplay Elements:*** The project will focus on core gameplay elements such as player navigation, enemy behaviors, shooting mechanics, power-ups, and scoring. While incorporating engaging features, the scope will not extend to complex subsystems that may compromise the project's feasibility.
* ***Graphics and Sound:*** The project will include visually appealing 2D graphics for the space environment, spaceships, and enemy entities. Sound effects and background music will enhance the gaming experience. However, cinematic sequences or extensive audio-visual effects beyond the scope of a classic 2D space shooter will be excluded.
* ***Single-Player Focus:*** The primary focus of the project will be on delivering a robust single-player experience. Multiplayer or online features, while desirable, are beyond the initial project scope.

By clearly delineating the project boundaries, we aim to maintain a manageable scope that allows for successful development and a polished product in the form of an enjoyable 2D space shooter game.

# **Project Plan and Team**

## **Project Plan**

### ***Pre-Production (Week 1):***

1. Conceptualization of game design, mechanics, and visual aesthetics.
2. Research and documentation on C++ and SFML features relevant to 2D game development.
3. Initial team meetings to establish a shared vision and objectives.

### ***Design Phase (Weeks 2-3):***

1. Detailed game design document outlining player interactions, enemy behaviors, power-ups, and scoring mechanisms.
2. Designing visual assets, including sprites for spaceships, enemies, and environmental elements.
3. Prototyping core gameplay mechanics.

### ***Development Phase (Weeks 2-4):***

1. Implementing game mechanics using C++ and SFML.
2. Iterative development cycles for continuous playtesting and feedback.
3. Concurrent development of visual assets and audio elements.

### ***Testing and Debugging (Weeks 3-4) whereas testing went on with the making of code:***

1. Rigorous testing of gameplay mechanics, graphics, and sound.
2. Identifying and addressing bugs and glitches.
3. Performance testing and optimization.

### ***Polishing and Refinement (Week 4):***

1. Refining user interface elements.
2. Fine-tuning visual and audio elements for a polished experience.

### ***Documentation and Deployment (Week 5):***

1. Finalizing documentation, including a user manual and developer documentation.
2. Preparing for desktop deployment on Windows, Linux, and macOS.

## **Responsibilities among team members**

We are 2 group members in total so both of us had to work together as to save time and finish the project in specified time.

So we distributed the Responsibilities in such a way that we work together efficiently and both of us have equal Responsibilities.

## **Role playing**

|  |  |
| --- | --- |
| Sami-Ur-Rehman Raja | Kashmallah Shah |
| 1. REQUIREMENT GATHERING  2. ANALYSIS  3. IMPLEMENTATION  4. DEPLOYEMENT  5. MAINTENANCE  6. REPORT | 1. ANALYSIS  2. DESIGN  3. IMPLEMENTATION  4. TESTING  5. MAINTENANCE  6. REPORT |

# **A diagram of a process Description automatically generated**

Figure e Project Requirement

# **Audience of our Project**

## **Intended Audience of your Project**

The 2D space shooter game is designed to cater to a broad audience, appealing to individuals with different gaming experiences and preferences. The main target audience includes:

### ***Casual Gamers:***

1. Individuals seeking an enjoyable and accessible gaming experience without the need for extensive learning curves.
2. Those who appreciate classic arcade-style games and want a modern take on the 2D space shooter genre.

### ***Fans of Space-themed Games:***

1. Enthusiasts who have a fondness for space-themed games and are drawn to the immersive experience of navigating through a visually dynamic interstellar environment.

### ***Nostalgia Seekers:***

1. Players who have a nostalgic connection to classic 2D space shooter games and are looking for a contemporary rendition that captures the essence of the genre.

### ***Casual Competitive Players:***

1. Individuals who enjoy competitive elements prefer a more relaxed gaming experience, where they can engage in challenging space battles without the intensity of competitive eSports.

### ***All Age Groups:***

1. The game is designed to be family-friendly, suitable for players of all age groups, ensuring that both younger and older audiences can enjoy the gameplay.

## **Identified Stakeholders:**

### ***Development Team:***

Game developers, graphic designers, sound designers, and game designers actively involved in the creation and implementation of the 2D space shooter game.

### ***Players and Gamers:***

The end-users who engage with and experience the game. Their feedback and preferences are essential for shaping the game's design and future updates.

### ***Investors and Sponsors:***

1. Individuals or organizations providing financial support or investment for the project.
2. Their stake involves the game's commercial success and potential returns on investment.

# **System Design and Functional Requirements**

## **Hardware and Software**

1. Visual Studio IDE 2022 Professional

[Download Visual Studio Tools - Install Free for Windows, Mac, Linux (microsoft.com)](https://visualstudio.microsoft.com/downloads/)

1. SFML (Visual C++ 17 (2022) - 64-bit)

[SFML 2.6.1 (SFML / Download) (sfml-dev.org)](https://www.sfml-dev.org/download/sfml/2.6.1/)

1. 64-bit Operating System

## **Describe at least FIVE Functional Requirements**

### ***Player Movement and Controls:***

The game should allow players to control the spaceship's movement using keyboard input. Functionalities include forward, backward, left, and right movements.

### ***Enemy AI and Behavior:***

Implement an intelligent enemy AI system that generates dynamic and challenging behaviors for adversaries. Enemies should exhibit varied movement patterns and shooting strategies, increasing difficulty as players progress through the levels.

### ***Power-ups and Upgrades:***

Integrate power-ups such as shield enhancements, weapon upgrades, and score multipliers. Functional requirements include defining the effects of each power-up and ensuring they are appropriately distributed throughout the game.

### ***Scoring System:***

Develop a scoring mechanism that tracks and rewards players based on their performance. Functionalities include accurate point calculations for defeating enemies, achieving milestones, and successfully completing levels.

### ***User Interface (UI):***

Design an intuitive and visually appealing user interface that provides essential information, including score, health, and power-up status. Functionalities include displaying relevant game statistics, such as the number of lives remaining and current level.

## **Mapping of the functional requirements with four pillars of OOP**

### ***Encapsulation:***

The player’s movement and controls, as well as the enemy AI and behavior, can be encapsulated within dedicated classes. This encapsulation helps organize related functionalities and hides the internal implementation details, promoting modularity.

### ***Inheritance:***

The power-ups and upgrades can be implemented using inheritance, with a base class for power-ups and specific derived classes for each type of enhancement. This facilitates code reuse and supports the extension of functionalities in the future.

### ***Polymorphism:***

The scoring system can leverage polymorphism by defining a base scoring class and allowing for multiple derived classes representing different scoring criteria. This enables flexibility in adding new scoring rules without modifying existing code.

### ***Abstraction:***

The user interface (UI) can be abstracted into a separate class, encapsulating the details of how information is displayed and updated. Abstraction allows for a clear separation between the visual presentation and the underlying game logic.

## **Class Diagram (UML Notation) with description**

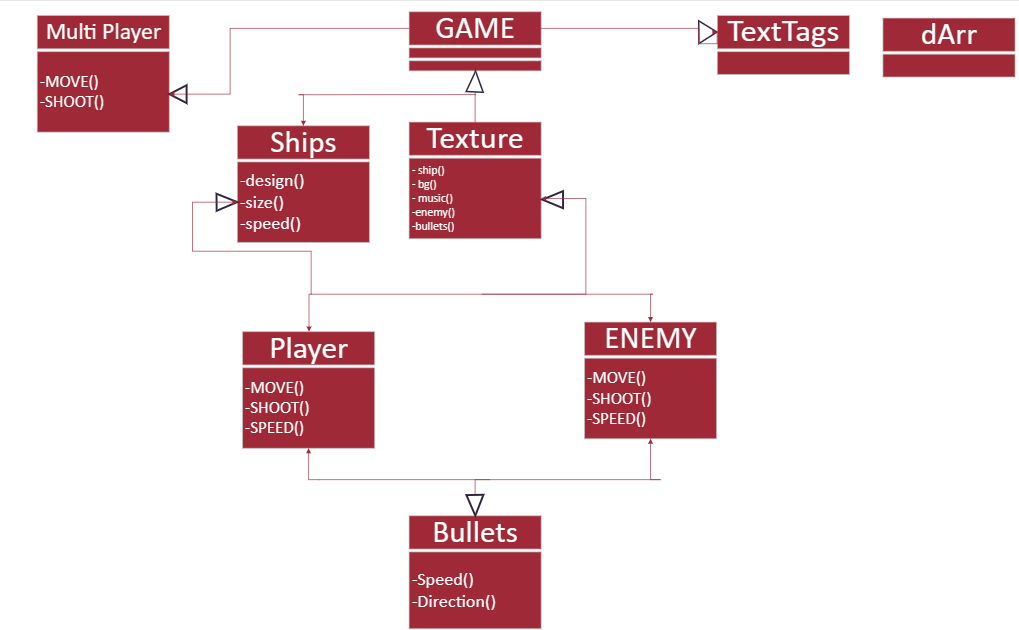


Figure f UML/Class Diagram

## **Uml diagram Description**

* ***Game:*** Represents the main game class that orchestrates the interactions between various game components.
* ***Player:*** Manages player input and controls for spaceship movement and shooting.
* ***Texture:*** Represents the base class for all the images, audios and fonts to be added and the display of the code.
* ***ENEMY:*** Manages enemy input and controls for spaceship movement and shooting.
* ***dArr:*** Defines a dynamic array class template called dArr. This class is a container for holding elements of a templated type T in an array-like structure.
* ***TextTags:*** Defines the written elements to be printed on game screen.

## **Use Case with description**

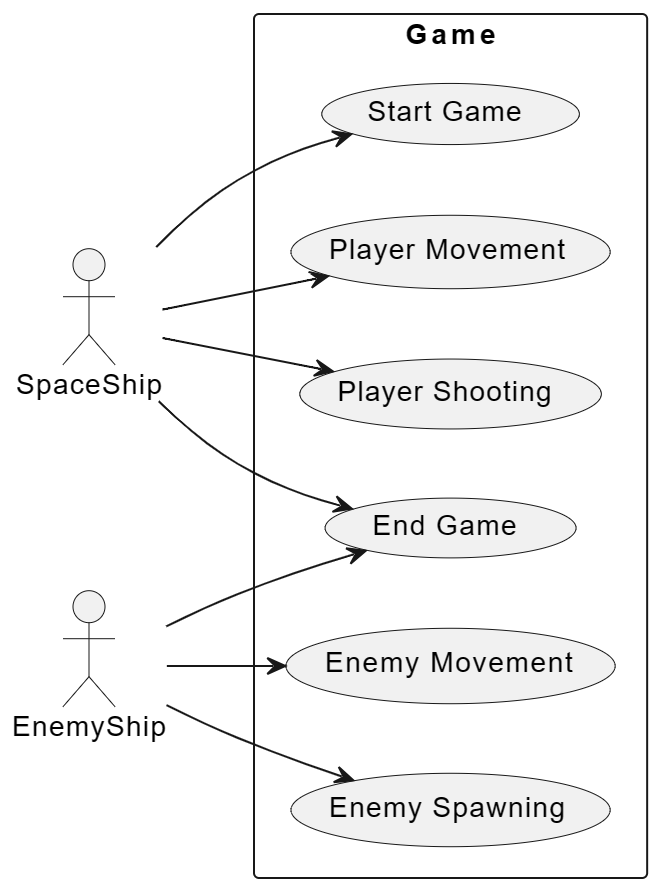


Figure g USE CASE diagram

# **Construction of Project**

We’ve started working in this project by making ***“.h”*** and ***“.cpp”*** files separately.

## **.h FILES (HEADER FILES)**

**“.h”** header files are used to declare functions, data structures, constants, and other elements that can be shared across multiple source code files. These header files typically contain function prototypes, type definitions, and macro definitions.

When you write code that spans multiple files or modules, you can use header files to declare the interfaces (functions, structures, etc.) that are used by other parts of the program. By including the appropriate **.h** header file in your source code, you provide access to the declarations contained within that file.

### ***“.h” Header Files Used In Our Program:***

* **Game.h**

Game.h is derived from Player.h and Enemy.h classes. All kind of movements and bounds required by the textures added in the Player.h and Enemy.h classes are configured in this class. The fonts, textures of player ship,lasers,missiles and enemy ships are loaded in the game through this class.

#### **DATA MEMBERS**

The Data Members are set to **Private** access specifier.

* **Game**

RenderWindow\* window;

float dtMultiplier;

* **Score**

unsigned score;

unsigned scoreMultiplier;

float multiplierTimerMax;

float multiplierTimer;

int multiplierAdder;

int multiplierAdderMax;

* **Fonts**

Font font;

Text followPlayerText;

Text staticPlayerText;

Text enemyText;

Text gameOverText;

Text scoreText;

* **Bars**

RectangleShape playerExpBar;

* **Players**

dArr<Player>players;

int playersAlive;

* **Textags**

dArr<TextTag>textTags;

* **Enemies**

dArr<Enemy>enemies;

vector<Enemy>enemiesSaved;

float enemySpawnTimer;

float enemySpawnTimerMax;

* **Pickups**

dArr<Pickup>pickups;

* **Textures**

vector<Texture>textures;

dArr<Texture>enemyTextures;

dArr<Texture>enemyBulletTextures;

dArr<Texture>lWingTextures;

dArr<Texture>rWingTextures;

dArr<Texture>cPitTextures;

dArr<Texture>auraTextures;

dArr<Texture>pickupTextures;

#### **MEMBER FUNCTIONS**

The Member Functions are set to **Public** access specifiers.

* **Parameterized Constructor**

Game(RenderWindow\* window);

* **Destructor**

virtual~Game();

* **Accessors**

inline RenderWindow& getWindow()

* **Functions**

void InitTextures();

void InitUI();

void UpdateUIPlayer(int index);

void UpdateUIEnemy(int index);

void Update(const float& dt);

void DrawUI();

void Draw();

* **Player.h**

Player.h contains Bullet.h class. This class initializes the movement of the player ship through Enumeration and defines the weapons for the player ship. In this class the levels,lives,score,damage and other powers are managed.

#### **DATA MEMBERS**

* **Timers**

float dtMultiplier;

unsigned playerNr;

float shootTimer;

float shootTimerMax;

float damageTimer;

float damageTimerMax;

* **Position**

Vector2f playerCenter;

Sprite sprite;

RectangleShape hitBox;

* **Accessories**

Sprite mainGunSprite;

vector<Bullet>bullets;

Texture\* laserTexture;

Texture\* missile01Texture;

Texture\* missile02Texture;

int controls[5];

* **Velocity**

Vector2f currentVelocity;

float maxVelocity;

* **Acceleration**

float accelaration;

* **Direction**

Vector2f direction;

float stabilizerForce;

* **Leveling**

int level;

* **Experience Points**

int exp;

int expNext;

int statPoints;

* **Additionals**

int cooling; (used for endurance of player ship)

int plating; (used for vigor of player ship)

int power; (used for strength of players ship)

int wiring; (used for agility of player ship)

* **Health**

int hp;

int hpMax;

* **Damage**

int damage;

int damageMax;

* **Score**

int score;

* **Loaded Weapon**

int currentWeapon;

* **Upgrades**

int mainGunLevel;

bool dualMissiles01;

bool dualMissiles02;

#### **MEMBER FUNCTIONS**

The Member Functions are set to **Public** access specifiers.

* **Parameterized constructor**

Player(vector<Texture> &textures,

dArr<Texture> &lWingTextures,

dArr<Texture> &rWingTextures,

dArr<Texture> &cPitTextures,

dArr<Texture> &auraTextures,

int UP = 22, int DOWN = 18,

int LEFT = 0, int RIGHT = 3,

int SHOOT = 57);

* **Destructor**

virtual ~Player();

* **Accessors**

Bullet& getBullet(unsigned index);

void removeBullet(unsigned index);

inline const int getBulletsSize()const

inline const Vector2f& getPosition()const

inline const String getHPAsString() const

int getDamage()const;

inline FloatRect getGlobalBounds()const

inline const int& getHp()const

inline const int& getHpMax()const

void takeDamage(int damage);

inline bool isAlive()const

inline const int& getPlayerNr()const

inline const int& getLevel()const

inline const int& getExp()const

inline const int& getExpNext()const

inline bool gainExp(int exp)

inline void gainScore(int score)

inline const int getScore()const

inline bool isDamageCoolDown()

* **Functions**

bool UpdateLeveling();

void ChangeAccessories();

void UpdateAccessories(const float& dt);

void Combat(const float& dt);

void Movement(Vector2u windowBounds,const float& dt);

void Update(Vector2u windowBounds, const float& dt);

void Draw(RenderTarget& target);

* **Statics**

static unsigned players;

* **Regular Functions**

float vectorlength(Vector2f v)

Vector2f normalize(Vector2f v, float length)

* **Bullet.h**

This class is associated with TexTag.h header file. This class defines the bullets, their acceleration, velocity and direction.

#### **DATA MEMBERS**

The Data Members are set to **Private** access specifier.

* **Stabilizer**

float dtMultiplier;

* **Textures**

Texture\* texture;

Sprite sprite;

* **Velocity**

float maxVelocity

Vector2f currentvelocity;

* **Acceleration**

float acceleration;

* **Move Direction**

Vector2f direction;

float initialVelocity;

#### **MEMBER FUNCTIONS**

* **Parameterized Constructor**

Bullet(Texture\* texture,

Vector2f position, Vector2f scale,

Vector2f moveDirection, float initialVelocity,

float maxVelocity, float acceleration);

* **Destructor**

virtual ~Bullet();

* **Accessors**

inline FloatRect getGlobalBounds()const

inline const Vector2f& getPosition()const

* **Functions**

void Movement(const float& dt);

void Update(const float& dt);

void Draw(RenderTarget& target);

* **Enemy.h**

The Enemy.h class is associated with Bullet.h class. It manages the movement of enemy ship.

#### **DATA MEMBERS**

* **Stabilizer**

float dtMultiplier;

* **Textures**

dArr<Texture>\* textures;

Sprite sprite;

* **Bounds**

Vector2u windowBounds;

Vector2f moveDirection;

Vector2f normalizedMoveDIR;

Vector2f lookDirection;

Vector2f normalizedlookDIR;

float maxVelocity;

dArr<Texture> \*bulletTextures;

dArr<Bullet>bullets;

float damageTimerMax;

float damageTimer;

float shootTimerMax;

float shootTimer;

* **Damages and healths**

int type;

int hp;

int hpMax;

int damageMin;

int damageMax;

int playerFollowNr;

#### **MEMBER FUNCTIONS**

* **Parameterized constructor**

Enemy(

dArr < Texture>& textures,

dArr < Texture>& bulletTextures,

Vector2u windowBounds,

Vector2f position,

Vector2f moveDirection,

int type,

int scaler,

int playerFollowNr);

* **Destructor**

virtual ~Enemy();

* **Accessors**

inline const int getDamage()const

inline const int getHP()const

inline const int getHPMax()const

inline const bool isDead()const

inline FloatRect getGloabalBounds()const

inline Vector2f getPosition()const

inline const int& getPlayerFollowNr()const

* **Functions**

void collision();

void takeDamage(int damage);

void Update(const float& dt, Vector2f playerPosition);

void Draw(RenderTarget& target);

float vectorlength(Vector2f v)

Vector2f normalize(Vector2f v, float length)

**TexTag.h**

#### **DATA MEMBERS**

* **Stabilizer**

float dtMultiplier;

* **fonts**

Font\* font;

Text text;

* **Speed**

float speed;

Vector2f direction;

* **Timer**

float timerMax;

float timer;

bool accelarate;

#### **MEMBER FUNCTIONS**

* **Parameterized constructor**

TextTag(Font\* font, string text,

const Color color,

Vector2f position,

unsigned int size,

float timerMax);

* **Destructor**

virtual ~TextTag();

* **Accessors**

inline const float& getTimer()const

void Update(const float& dt);

void Draw(RenderTarget &target);

* **dArr.h**

Defines a dynamic array class template called dArr. This class is a container for holding elements of a templated type T in an array-like structure. The class provides functionalities for dynamic memory management, element addition, removal, and various operations on the array.

#### **DATA MEMBERS**

unsigned cap;

unsigned nrofEl;

T\*\* arr;

void initialize(unsigned from);

void expand();

#### **MEMBER FUNCTIONS**

* **Default constructor**

dArr(unsigned cap = 5);

* **Parameterized constructor**

dArr(const dArr& obj);

* **Destructor**

~dArr();

* **Functions**

T& operator[](int index);

void operator=(const T& obj);

void add(const T element);

void remove(int index);

unsigned size()const;

void clear();

## **.cpp FILES (HEADER FILES)**

**“.cpp”** files are source code files that contain C++ code. The .cpp extension is commonly used for C++ source files to distinguish them from header files (.h or .hpp).

These files typically contain the actual implementations of functions, classes, and other constructs declared in header files. They can also include other .cpp files, header files (using #include directives), and libraries necessary for the program.

* ***“Main.cpp” In Our Program:***

#include"Game.h"

using namespace std;

using namespace sf;

int main()

{

RenderWindow window(VideoMode(1920, 1080), "WingMan Game", Style::Default);

Clock clock;

float dt = 0.f;

Game game(&window);

Music music;

if (!music.openFromFile("Textures/game1.mp3"))

{

cout << "ERROR" << endl;

}

music.setVolume(1000.f);

music.getLoop();

music.play();

//Game Loop

while (window.isOpen())

{

Event event;

while (window.pollEvent(event))

{

if (event.type == Event::Closed)

window.close();

if (event.type == Event::KeyPressed && event.key.code == Keyboard::Escape)

window.close();

}

dt = clock.restart().asSeconds();

game.Update(dt);

game.Draw();

}

return 0;

}

* ***Making Dynamic Array Header File:***

#pragma once

template<typename T>

class dArr

{

private:

unsigned cap;

unsigned nrofEl;

T\*\* arr;

void initialize(unsigned from);

void expand();

public:

dArr(unsigned cap = 5);

dArr(const dArr& obj);

~dArr();

T& operator[](int index);

void operator=(const T& obj);

void add(const T element);

void remove(int index);

unsigned size()const;

void clear();

};

template<typename T>

dArr<T>::dArr(unsigned cap)

{

this->cap = cap;

this->nrofEl = 0;

this->arr = new T \* [this->cap];

this->initialize(0);

}

template<typename T>

dArr<T>::dArr(const dArr& obj)

{

this->cap = obj.cap;

this->nrofEl = obj.nrofEl;

this->arr = new T \* [this->cap];

for (int i = 0; i < nrofEl; i++)

{

this->arr[i] = new T(\*obj.arr[i]);

}

this->initialize(this->nrofEl);

}

template<typename T>

dArr<T>::~dArr()

{

for (int i = 0; i < this->nrofEl; i++)

{

delete this->arr[i];

}

delete[]this->arr;

}

template<typename T>

T& dArr<T>::operator[](int index)

{

if (index < 0 || index >= this->nrofEl)

throw("OUT OF BOUNDS EXCEPTION");

return \*this->arr[index];

}

template<typename T>

void dArr<T>::operator=(const T& obj)

{

for (int i = 0; i < this->nrofEl; i++)

{

delete this->arr[i];

}

delete[] this->arr;

this->cap = obj.cap;

this->nrofEl = obj.nrofEl;

this->arr = new T \* [this->cap];

for (int i = 0; i < nrofEl; i++)

{

this->arr[i] = new T(\*obj.arr[i]);

}

this->initialize(this->nrofEl);

}

template<typename T>

void dArr<T>::initialize(unsigned from)

{

for (int i = from; i < this->cap; i++)

{

this->arr[i] = nullptr;

}

}

template<typename T>

void dArr<T>::expand()

{

this->cap \*= 2;

T\*\* tempArr = new T \* [this->cap];

for (int i = 0; i < this->nrofEl; i++)

{

tempArr[i] = this->arr[i];

}

delete[]this->arr;

this->arr = tempArr;

initialize(this->nrofEl);

}

template<typename T>

void dArr<T>::add(const T element)

{

if (this->nrofEl >= this->cap)

this->expand();

this->arr[this->nrofEl++] = new T(element);

}

template<typename T>

void dArr<T>::remove(int index)

{

if (index < 0 || index >= this->nrofEl)

throw("OUT OF BOUNDS EXCEPTION!");

delete this->arr[index];

this->arr[index] = this->arr[this->nrofEl - 1];

this->arr[--nrofEl] = nullptr;

}

template<typename T>

unsigned dArr<T>::size()const

{

return this->nrofEl;

}

template<typename T>

void dArr<T>::clear()

{

for (int i = 0; i < this->nrofEl; i++)

{

delete this->arr[i];

}

initialize(0);

this->nrofEl = 0;

}

# **Conclusion**

In conclusion, the development of the 2D space shooter game using C++ and SFML has been a journey marked by creativity, collaboration, and technical ingenuity. The project aimedto capture the classic essence of arcade-style space shooters while integrating modern design elements to cater to a diverse gaming audience.

The process began with a thorough exploration of the gaming domain, establishing the context and scope within which the project unfolded. Through an extensive background study and literature review, we gained valuable insights into the rich history and evolution of 2D space shooter games, informing our design decisions and ensuring our project aligned with both tradition and innovation.

As we conclude this report, the 2D space shooter game stands as a testament to the collaborative efforts, creative innovation, and technical proficiency of the development team. It is poised to offer players a nostalgic yet refreshing gaming experience, navigating through the vast cosmos and engaging in thrilling space battles. The journey from conceptualization to deployment has not only resulted in a successful game but has also enriched the skills and knowledge of the development team, paving the way for future endeavors in the dynamic field of game development.

------THE END------