Space-Shooters

Technical Design Document – Outline

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author(s) | Changes |
| 1.0 | Jan 2022 | D. Burchill |  |
| 2.0 | Feb 2022 | Arth |  |
|  |  |  |  |

# Table of Contents

Table of Contents

[1 Document History 2](#_Toc96912644)

[2 Table of Contents 3](#_Toc96912645)

[3 Game Summary 4](#_Toc96912646)

[Space-Shooters is right-left side scroller. The big picture of the game is to defend the spaceship from the different enemies and defend the final boss to win the game. The spaceship introduced with different types of pickups during the mission. The pickups will be not generated randomly, spaceship must kill some enemies to generate some pickups. The pickups generally boost speed, upgrades the bullet type, and increases health of the player spaceship. The Player must keep their health safe (stay alive) to survive till the end. 4](#_Toc96912647)

[Development Environment 4](#_Toc96912648)

[3.1 Development Hardware 4](#_Toc96912649)

[3.2 Development Tools 4](#_Toc96912650)

[3.3 External Code 4](#_Toc96912651)

[3.4 Game Engine 4](#_Toc96912652)

[4 Architectural Analysis 5](#_Toc96912653)

[4.1 Classes 5](#_Toc96912654)

[4.2 Class Diagrams 10](#_Toc96912655)

[4.2.1 Game Class 10](#_Toc96912656)

[4.2.2 Objects 11](#_Toc96912657)

[4.2.3 File References .h & .cpp 13](#_Toc96912658)

[4.3 Behavioral Analysis 15](#_Toc96912659)

[4.3.1 Game States 15](#_Toc96912660)

[4.4 Game Loop 16](#_Toc96912661)

[5 Technical Risks 16](#_Toc96912662)

# Game Summary

# Space-Shooters is right-left side scroller. The big picture of the game is to defend the spaceship from the different enemies and defend the final boss to win the game. The spaceship introduced with different types of pickups during the mission. The pickups will be not generated randomly, spaceship must kill some enemies to generate some pickups. The pickups generally boost speed, upgrades the bullet type, and increases health of the player spaceship. The Player must keep their health safe (stay alive) to survive till the end.

# Development Environment

## Development Hardware

*Specify the primary programming language(s), including version (e.g. C++03), to be used for development. Also specify any scripting languages that are used (if applicable).*

Player control is through the keyboard and no special hardware has been used. This game and game engine have been developed in a MS Windows environment. It’s built using SFML 2.5 and C++17, so it could be built for Linux or MacOS platforms too. Programming Languages

## Development Tools

*List the software needed for development, such as IDEs, compilers, debuggers, version control software, graphical editors, sound editors, etc.*

* Microsoft Visual Studio 2019 and C++ compiler was used as the IDE and compiler.
* Git was used for Version control

## External Code

List the libraries to be used, including the URLs of the official library site, and the version number of the library used in this project.

SFML, the simple fast multimedia library was used for all graphics, <https://www.sfml-dev.org>. SFML is a simple cross platform wrapper around Open-GL providing a simple interface for graphics programing.

## Game Engine

The Space-Shooters game engine consists of a game class to hold all game objects and manage update and draw calls for all objects in the game, a group of State classes to represent different game scenes or game states. The sample game has only one scene or game state, along with a title state, a menu state, a pause state, and a game over state.

All the Collisions are handles in game class. The game engine is built on C++ programming language and SFML (Simple and Fast Multimedia Library) API. SFML is a cross-platform software development library designed to provide a simple application programming interface to various multimedia components in computers. It is a 2D rendering game engine.

# Architectural Analysis

## Classes

Describe the classes that will have to be implemented. For each class, provide:

* Its responsibilities
* How it collaborates with other classes

**Game Class**

|  |  |
| --- | --- |
| Class Constructor | Responsibilities |
| Game::Game(sf::RenderWindow\* window) | Provides a window to the game.  RenderWindow is pointer in game.h. |
| Accessors | Responsibilities |
| Inline sf::RenderWindow& getWindow()  { return \*this->window; } | RenderWindow is pointer in game.h and de-reference it because it will not create separate memory. It will directly copy code from calling function. |
| Functions | Responsibilities |
| InitUI(); | It contains Player Health Texts, enemy health text, game over text with size, font, color. |
| UpdateUIPlayer(int index) | It has update of player Ui texts. Also, position for player UI texts and string for Ui texts. |
| UpdateUIEnemy(int index); | It has update of enemy Ui texts. Also, position for enemy UI texts and string for Ui texts. |
| update(); | Update function contains,  Enemy spawn timer, enemy spawn, reset enemy spawn timer,  Player update, bullets update,   Enemy collision check with bullets,  Enemy take damage check,  checks if enemy dead or not (if yes, erase enemy),  checks if bullets are out of boundary, (if yes then erase it),  Checks collision between player-enemy,  Checks if enemies are out of boundary, (if yes then erase it), |
| draw(); | This function,  Draws the enemies,  Updates UIEnemy and draws enemy text. |
| initTextures(); | This function contains all the textures of the game. Player, enemy, gun, laser, bullets textures. Also, it contains aura textures also. |
| Enums | Responsibilities |
| enum textures  { player = 0, laser1, missile1, mainGun1, enemy1 }; | Texture for player, laser, missile1, maingun1, enemy1 |

**Player Class**

|  |  |
| --- | --- |
| Class Constructor | Responsibilities |
| Player::Player(std::vector<sf::Texture>& textures,  int UP, int DOWN,  int LEFT, int RIGHT,  int SHOOT)  :level(1)  , exp(0)  , expNext(100)  , hp(10)  , hpMax(10)  , damage(1)  , damageMax(2)  , score(0) | Contains the vector of texture reference which is define in game class. It has player controls with player level, exp, hp, damage and score.  Initial level is defined as 1, hp and hp max is defined as 10. Damage is set to 1 and max damage is to 2.  Player score is 0 initially.  Also, player, bullet, missile, and player gun texture are  Defined in player class. Player shoot timing is defined as 25. In addition, player animation, current weapon, number of player are implemented. |
| Accessors | Responsibilities |
| vector<Bullet>& getBullets()  { return this->bullets; } | It returns bullets and it is used by game class to check collision and clash. |
| sf::Vector2f& getPosition() const  { return this->sprite.getPosition(); } | It returns current position on sprite and accessed by many classes. |
| sf::String getHpAsString() const  { return std::to\_string(this->hp) + "/" + std::to\_string(this->hpMax); } | It returns current hp and hpMax in string format. It is used by UpdateUIPlayer function in game cpp. |
| int getdamage()const | Initial damage by player is set to 0. It contains different cases of damage.  LASER, MISSILE1 (double damage), MISSILE2 (3 time damage) |
| sf::FloatRect getGlobalBounds()const  { return this- >sprite.getGlobalBounds();} | Returns GlobalBounds (rectangle) of entity (player). To check the collision.   (Returns the bounds of the entity in the entity's coordinate system.) |
| const int& getHp()const  { return this->hp; } | Returns current hp of player. |
| const int& getHpMax()const  { return this->hpMax; } | Returns HpMax which is 10. |
| void takeDamage(int damage)  { this->hp -= damage; } | Returns int damage and subtract it from current Hp of player. |
| bool isAlive()const  { return this->hp > 0; } | Checks that player is alive or not. Only returns true if hp >0. |
| Functions | Responsibilities |
| UpdateAccessories(); | This function responsible for,  Gun position with the player,  animation of the gun and correct after firing. |
| Combact(); | Checks the current weapon of the player and initialize the weapon. Then animate the gun.  Weapons – LASER, MISSILE1, MISSILE2. Resets the timer. |
| Movement(); | This function contains movement of the player. UP, DOWN, LEFT, RIGHT with stabilizer force effect. |
| Update(sf::Vector2u windowBounds); | Updates timers for shoot and damage first and then calls the  Movement() UpdateAccessories() Combate() |
| Draw(sf::RenderTarget& target); | Draw the bullet sprite , mainGunSprite and then player sprite. |
| Enums | Responsibilities |
| enum controls  { UP = 0, DOWN, LEFT, RIGHT, SHOOT }; | types of controls. |
| enum Weapons  { LASER = 0, MISSILE1, MISSILE2 }; | Types of weapons for player. |

**Enemy Class**

|  |  |
| --- | --- |
| Class Constructor | Responsibilities |
| Enemy::Enemy(  sf::Texture\* texture,  sf::Vector2u windowBounds,  sf::Vector2f position,  sf::Vector2f direction,  sf::Vector2f scale, int type,  int hpMax, int damageMax,  int damageMin) | Enemy class constructor contains, textures of enemy, windowBounds,  Position of enemy (random on Y coordinate), direction of enemy, scale, enemy type, enemy hp max which is 3, damage max, damage min. |
| Accessors | Responsibilities |
| const int getDamage()const  { return rand() % this->damageMax +  this->damageMin; } | This function returns random damage from damageMin to damageMax. |
| const int getHP()const  { return this->hp; } | This function returns current hp of enemy. |
| const int getHPMax()const  { return this->hpMax; } | This function returns max hp of enemy. |
| const bool isDead()const  { return this->hp <= 0; } | Checks if the enemy is dead or not. Returns true only if hp <= 0. |
| sf::FloatRect getGlobalBounds() const  { return this->sprite.getGlobalBounds(); } | Returns GlobalBounds (rectangle) of entity (enemy). To check the collision.   (Returns the bounds of the entity in the entity's coordinate system.) |
| sf::Vector2f getPosition() const  { return this->sprite.getPosition(); } | This vector returns current sprite position. |
| Functions | Responsibilities |
| takeDamage (int damage); | Checks if enemy is hitted, substracts the hp. Resets the damage timer. |
| Update(); | Checks the enemy type and updates the movement of sprite. while hitted , sets the color of sprite to Red and moves little backward to feel the game more real. |
| Draw(sf::RenderTarget& target); | This function simply draws the enemy sprite. |
| Enums | Responsibilities |
| enum eTypes  { MOVELEFT = 0, FOLLOW, FOLLOWFAST, FOLLOWSLOW, FOLLOWSHOOT, FOLLOWFASTSHOOT }; | Enemy type. |

**Bullet Class**

|  |  |
| --- | --- |
| Class Constructor | Responsibilities |
| Bullet(  sf::Texture\* texture,  sf::Vector2f position,  sf::Vector2f scale,  sf::Vector2f direction,  float initialVelocity,  float maxVelocity,  float acceleration); | Class constructor contains texture of bullet,  position of bullets, scale of bullets, direction, initial velocity, max velocity, and acceleration of bullets. |
| Accessors | Responsibilities |
| sf::FloatRect getGlobalBounds() const  { return this->sprite.getGlobalBounds(); } | Returns GlobalBounds (rectangle) of entity (bullet). To check the collision.   (Returns the bounds of the entity in the entity's coordinate system.) |
| sf::Vector2f getPosition() const  { return this->sprite.getPosition(); } | Returns current position of sprite. |
| Functions | Responsibilities |
| Movement(); | Updates the bullet when,  Accelerate in the pressed direction, velocity increasing by acceleration and multiply by direction. For both X and Y cordinate |
| Update(); | Updates the movement of bullet. |
| Draw(sf::RenderTarget& target); | Draws the bullet sprite. |

**Game Loop**

|  |  |
| --- | --- |
| main() | Responsibilities |
| sf::RenderWindow | Displays the window of size 1720 x 880 videoMode with frame limit of 65. |
| game.update(); | Calls the Update() of the game class in main game loop. |
| game.draw(); | Calls the Draw() of the game class in main game loop. |

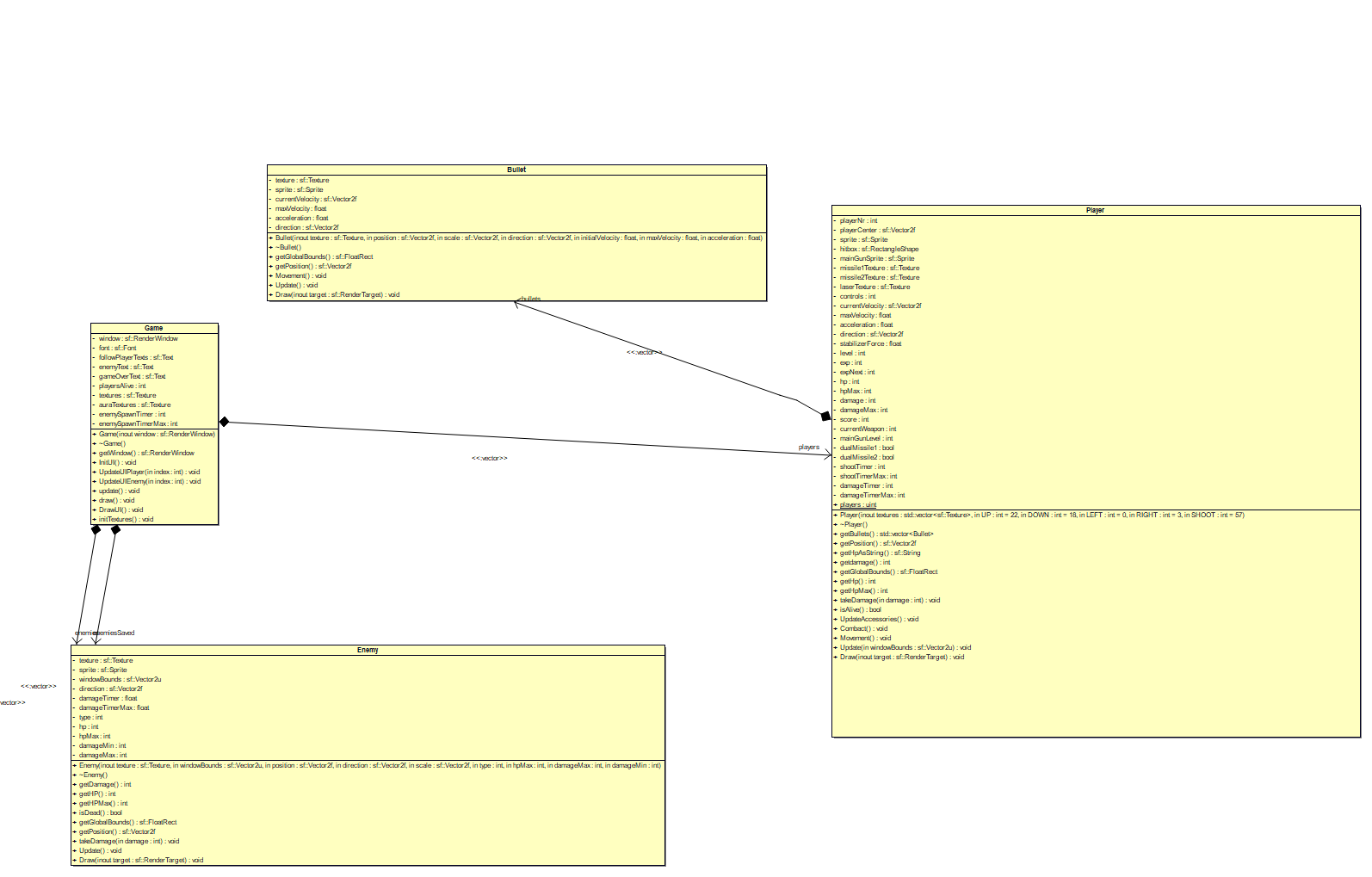
**DataTables:**

|  |  |
| --- | --- |
| Classes and Functions | Responsibilities |
| **Player** | Data related to Aircraft gun type (laser, missile, missile 2), texture, capabilities, and stabilizer force. |
| **Enemy** | Data related to enemy aircraft, texture, capabilities, and position. |
| **Directions** | Random generate enemy aircraft from position Y. |
| **Movement (Player)** | Data related player movement with animation data |
| **PickupData** | Pickup Type and capabilities |
| **ProjectileData** | Bullet or missile and capabilities |
| **ParticleData** | Particle type and decay rate |

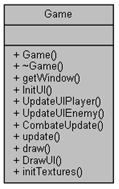
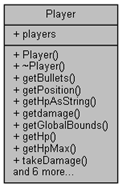
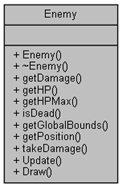
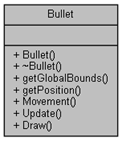
## Class Diagrams

Present class and object diagrams that show the relationships between classes. Show only the most essential attributes and methods for each class. You may use Doxygen to create class diagrams

### Game Class



### Objects

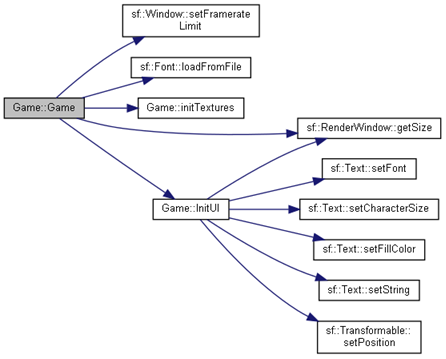
   

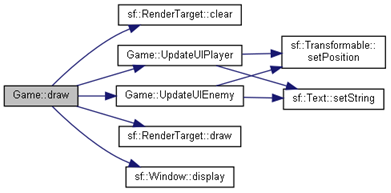
 Diagram

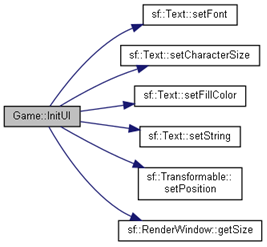
Description automatically generated

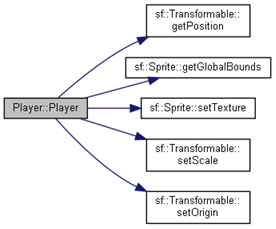
 Diagram

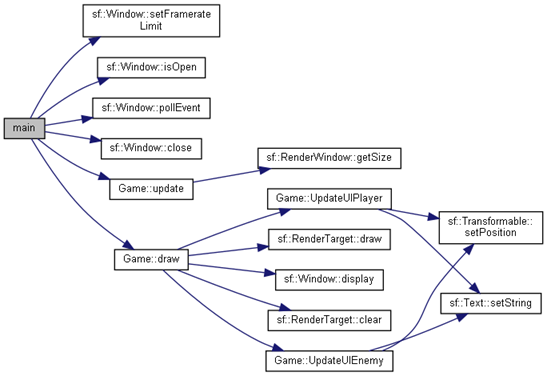
Description automatically generated







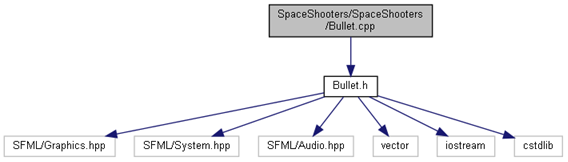




### File References .h & .cpp

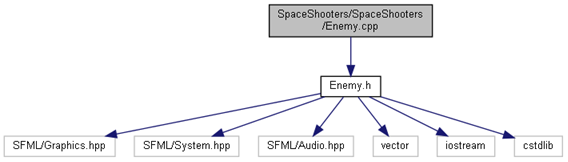
**Bullet.h & .cpp File Reference**

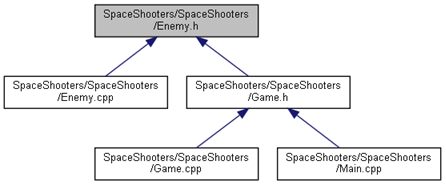
**This graph shows which files directly or indirectly include this file**



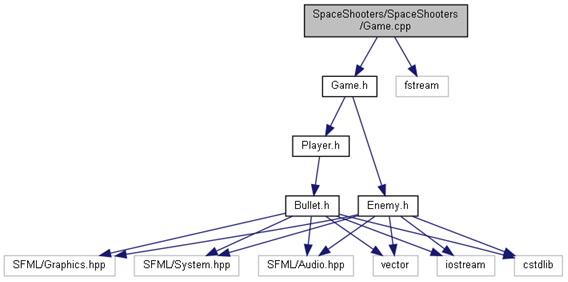
**Enemy.h & .cpp File Reference**

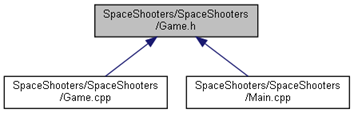
**This graph shows which files directly or indirectly include this file:**



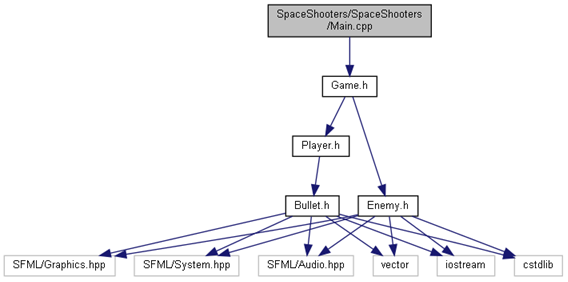


**Game.cpp & .h File Reference**





**Main.cpp File Reference**



## Behavioral Analysis

*Present statecharts, flow charts (activity diagrams), sequence diagrams, etc. that model complicated behavior. If your game has actors that implement a state machine, this would be the section where you’d present the statechart.*

### Game States

Graphical user interface, text, application

Description automatically generated

## Game Loop

*Describe, in order, the sequence of activities that happen during each game loop. You must document this even if you’ll be basing your game on the games developed in class.*

The Space-Shooter game initially generates a window and key pressed events are checked. The main game loop has game objects which updates the game by each frame and draws the game. In game class, all the collision were handled and update. The game class also contains textures for all entities. It creates the enemy before checking the collision and update function. The enemy and bullets are erased when they are out of boundary of the window. UI is updated by each entity class separately and by game class. The game has different state and it behaves according to user inputs.

# Technical Risks

List all technical risks that could make it difficult or impossible to complete the game. Examples:

* Uncertainty on how to implement a certain feature
* Uncertainty on if a certain feature can be executed fast enough in real time
* First time using a certain library

|  |  |  |
| --- | --- | --- |
| Risk | Severity | Mitigation (what is to be done to eliminate or minimize this risk) |
|  |  |  |
|  |  |  |