Object-Oriented Programming Project

<Shmup Game>

01286131 Computer Programming

Software Engineering Program

By

Chanasorn	Howattanakulphong	65011277
Napatr	Sapprasert	65011381
Kanokjan	Singhsuwan	65011320

Introduction:

In our project, we made an arcade shmup game by using the sfml library together with c++ programming language. By using what we learn in the course, building class- objects using inheritance and polymorphism, that allows us to construct solid playable arcade games.

How the game works:

When the game app is opened, user will be prompted with “press enter to start the game” screen. Once enter is pressed, the game starts.

The player can move with WASD and shoot by holding spacebar or Z. The user can also use “bomb” by pressing X, removing all the bullets and non-boss enemies from the screen. The player starts with 10 lives, which decreases every time the enemies’ bullet hits the player’s hitbox or when the enemy makes contact with the player.

The player will have to survive different types of enemy that will come at the player, avoiding their bullets or killing them. If the enemy is shot down, they drop points which the player can pick up to get extra scores. When the Player’s score reaches a certain threshold, Player’s shooting will be upgraded. After surviving for a while, the player will be met with the boss, which serves as the last enemy of the game. Upon killing the

boss, the player will win the game and get their score recorded (if it is higher than the previous highscore)

Features:

1. Arcade Action: In the code we build a player class and enemy class which are the base of classic shmup games. Each of the classes can provide an object that has a unique function for the gameplay.

2. Diverse Enemy Types: The different enemy types are polymorph from the enemy class which makes each of them differ in look, health, damage, and behavior. The enemy AI differs for each enemy class, and new enemies with new patterns will be made everytime the level increases.

The enemy types are:

Red enemy: low health, moves in a straight line, doesn't shoot, but spawn a lot of bullets when killed

Blue enemy: low health, doesn't shoot but runs towards the player.

Black enemy: moderate health, shoots 3 bullets in a cone shape downwards.

Purple enemy:high health, shoots 3 bullets with different speed at the player's current position.

Mini boss: unkillable, shoots 2 rows of cone shaped bullet pattern at the player's current position. Only shoots once, then leave the screen.

3. Bullets management. The bullets the enemies and player shoots are objects of class Bullets which are managed by the BulletManager class that ensures smooth bullet spawn and efficient memory management. Enemies' bullets and Player's bullets are objects of class Enemybullets and Playerbullets respectively. Enemybullet class will be further differentiated with enum type that classifies the bullets into normal enemies' bullets and boss' bullets. To ensure smooth gameplay and memory management, All the bullets will be stored in the BulletManager class that manages the bullets behavior and reuses the bullet objects to manage memory.

4.Boss Battles: At the last stage, the player will meet the boss. Differ from normal enemies, the boss has higher health, damage and range, with many different bullet patterns. That will add some more

challenges to our game. The boss has 3 different phases which will activate upon reaching the set health threshold

5. Score Challenges: Compete against yourself to achieve the highest scores. Our game has a built the scoring system so that the player can always feel active to continue to play the game and aim for best. Highscores are written and read on a txt file to save the data.

6. Soundtrack and Visual Effects: We did a cosmetic addition for the background music, background scrolling and also the visual effect of the time when the bullet hit and killed the enemy.

7.error checking: We ensure that there wouldn't be errors by setting conditions for different events, such as in order to proceed to the next level, we would cross-check the variables such as current enemy count, current enemy kill count, current bullets on screen to ensure that the numbers add up.

We also made it so that the enemy that leaves the screen will self-destruct to make sure the game proceeds naturally even if the player fails to shoot every enemy. And also made it so that the player cannot move beyond the screen size.

To manage memory, we also check for offscreen bullets and reuse the already made bullets to avoid an unnecessary large number of objects being created.

Conclusion:

In conclusion, developing a shoot 'em up (shmup) game can be an exciting and rewarding coding project. By following a structured code description and incorporating the suggested features, we can create an immersive and action-packed gaming experience.

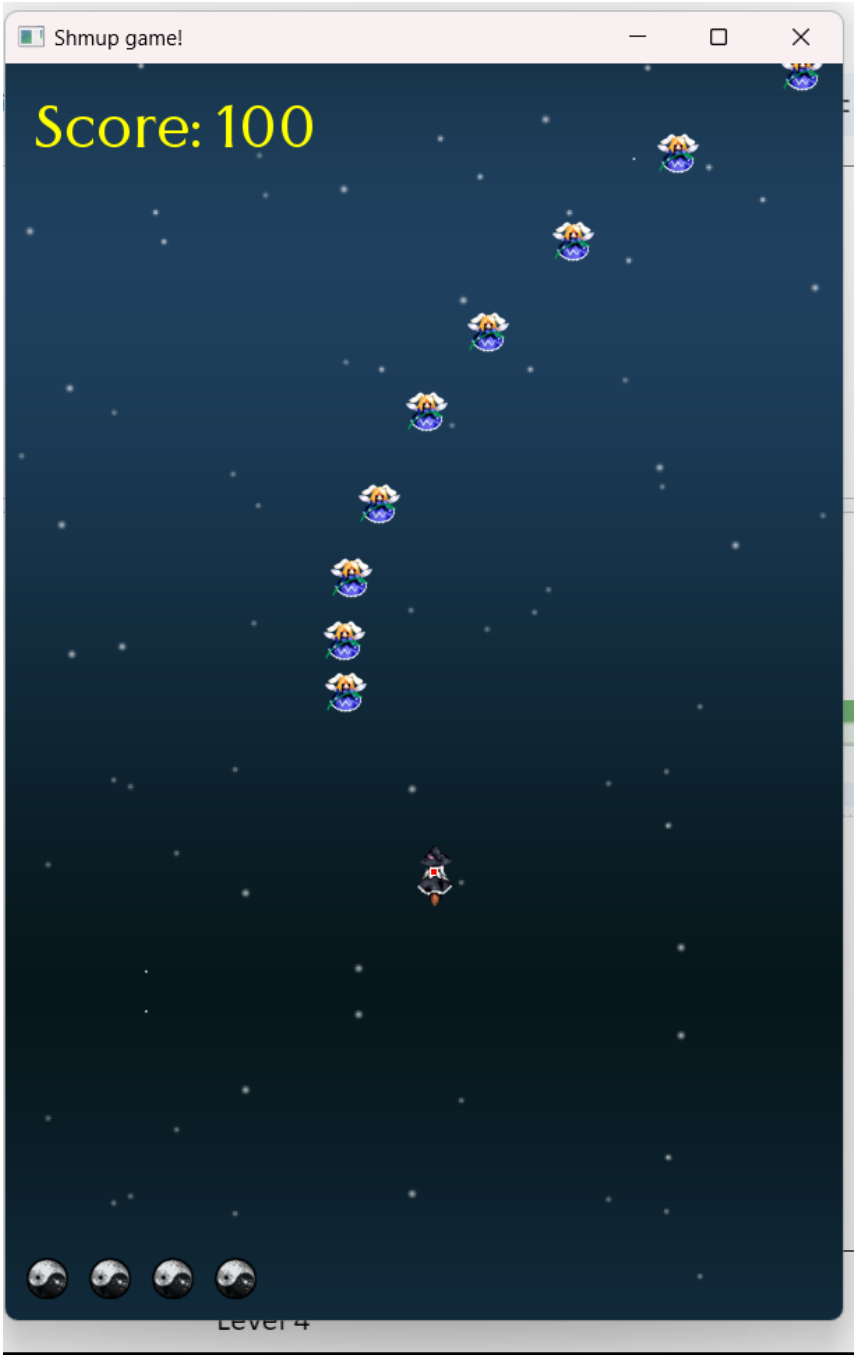
Throughout the development process, we've encountered challenges such as designing enemy behavior, balancing gameplay difficulty, and creating visually appealing effects. However, with careful planning, iterative testing, and attention to detail, we can overcome these obstacles and bring our vision to life.

In Game Interface:

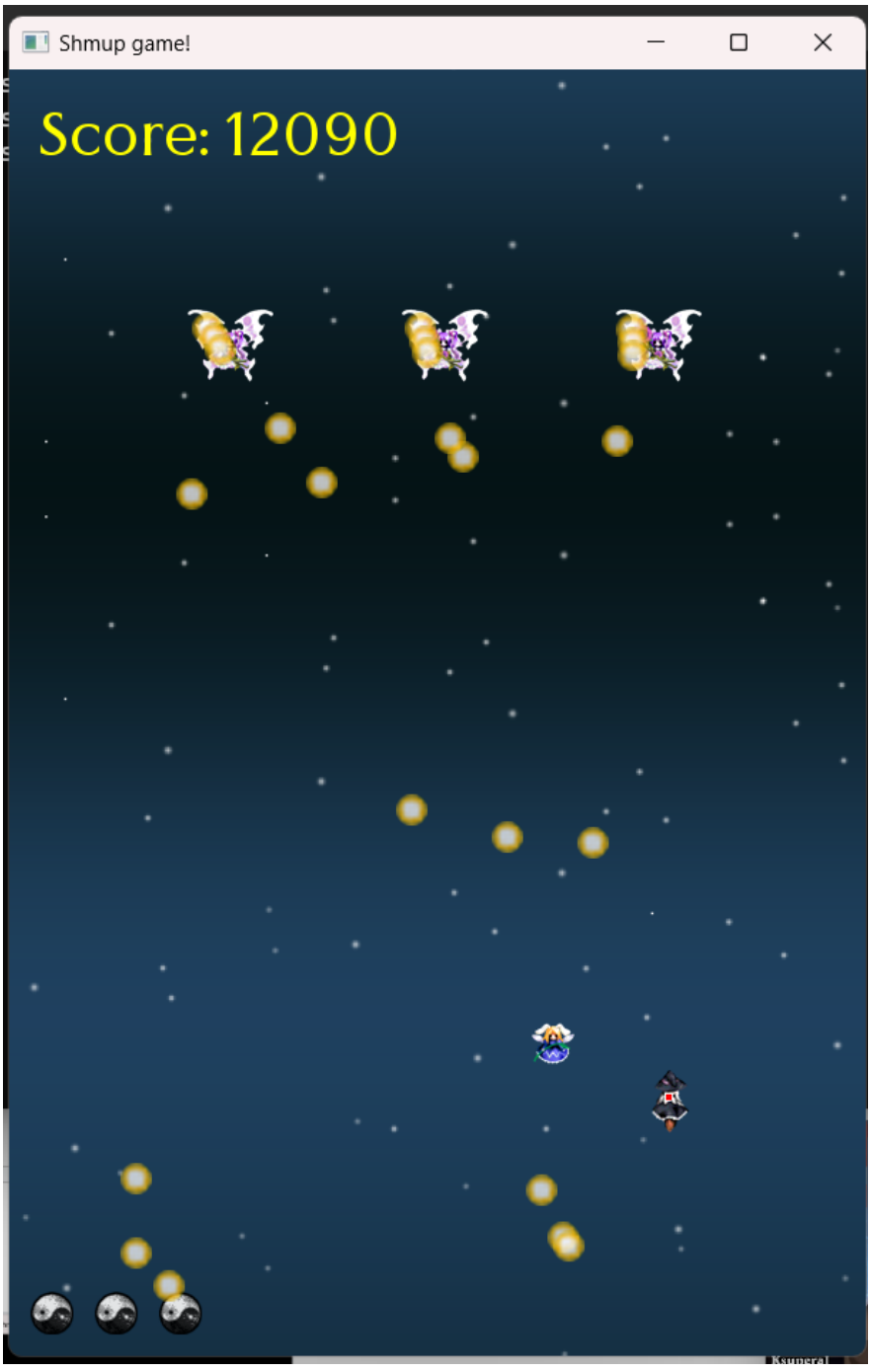
Title Screen:

Level 1 -3

Level 4

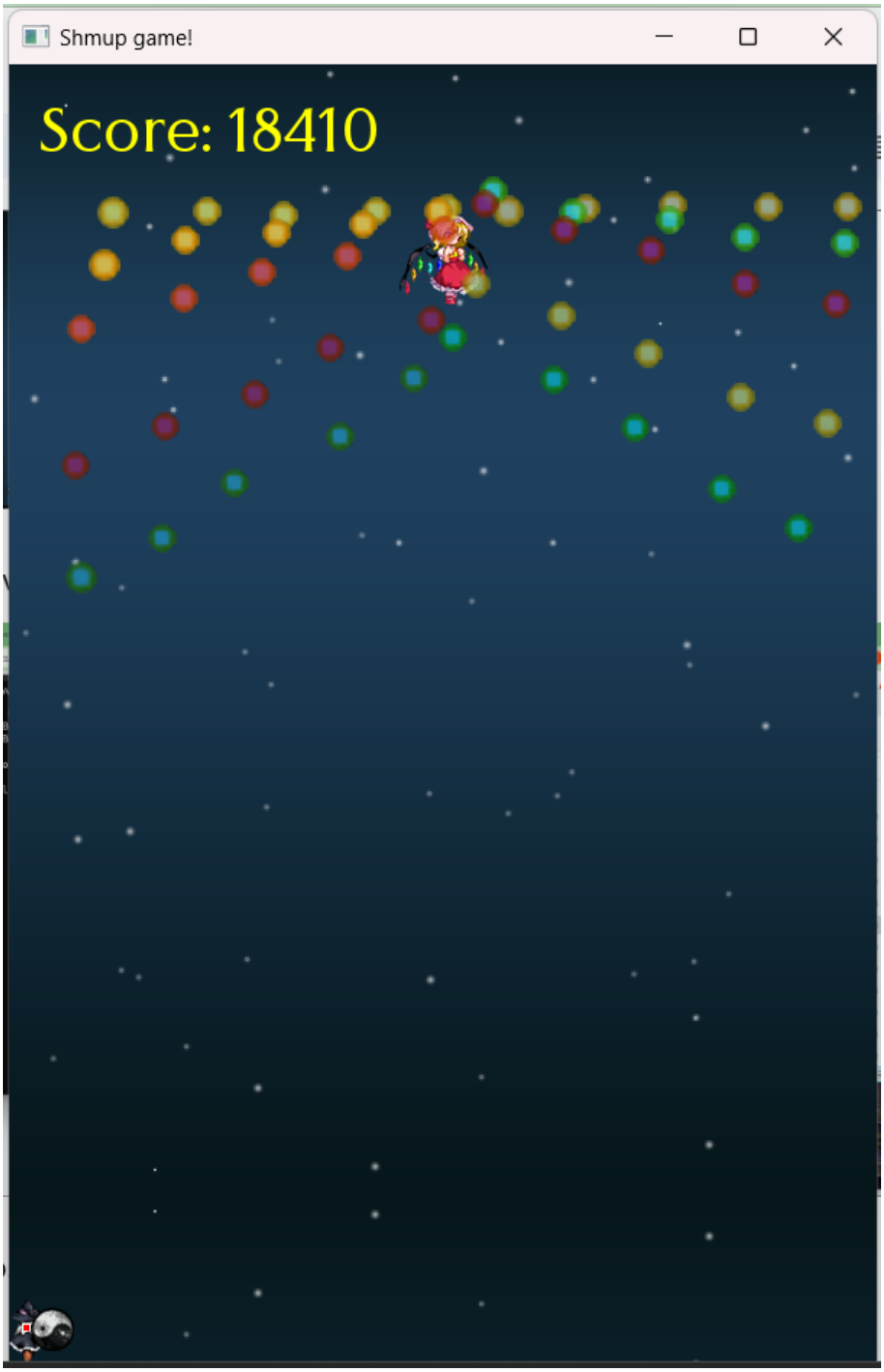


Level 5

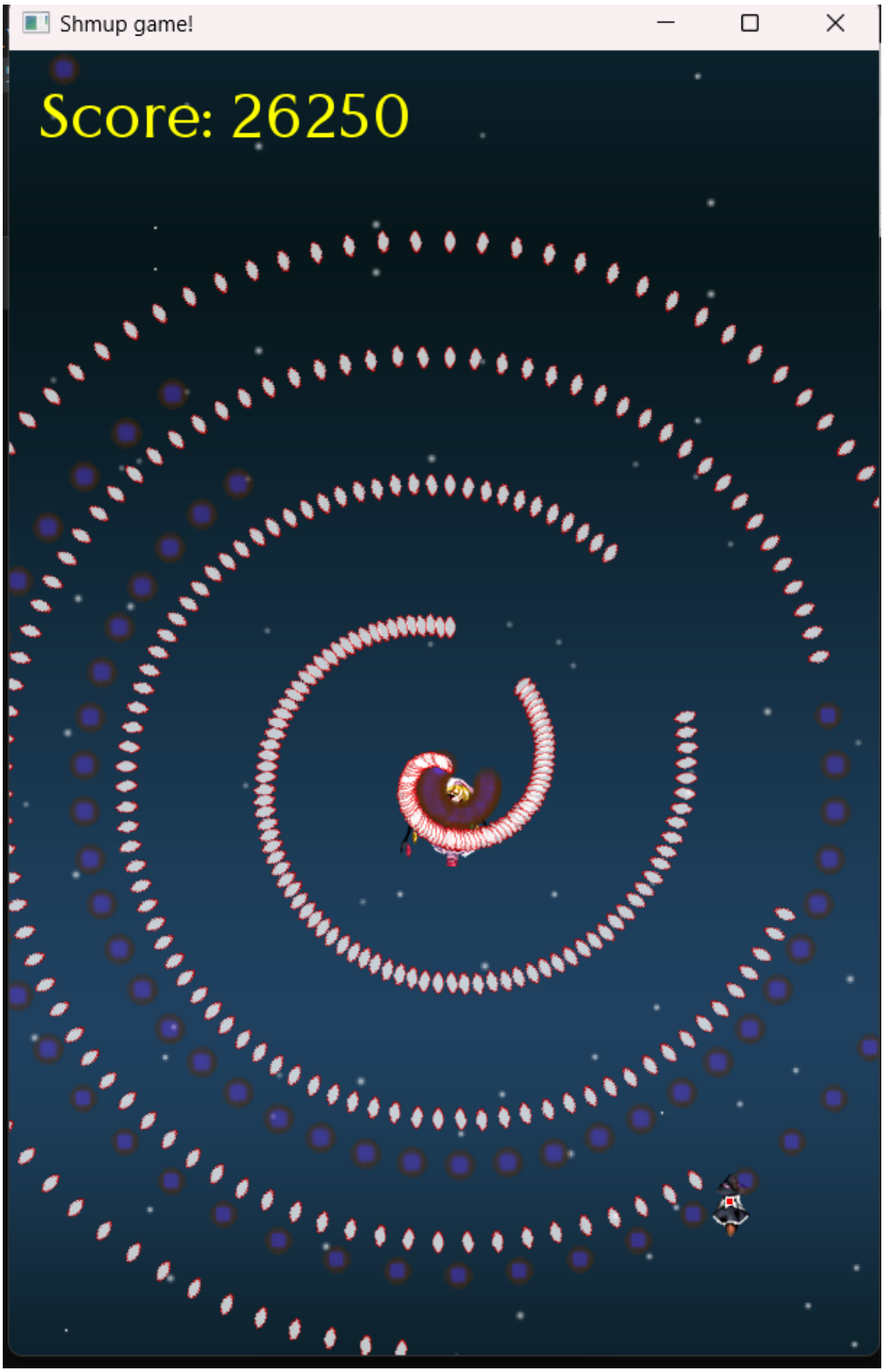


Level 6

Boss Level 1

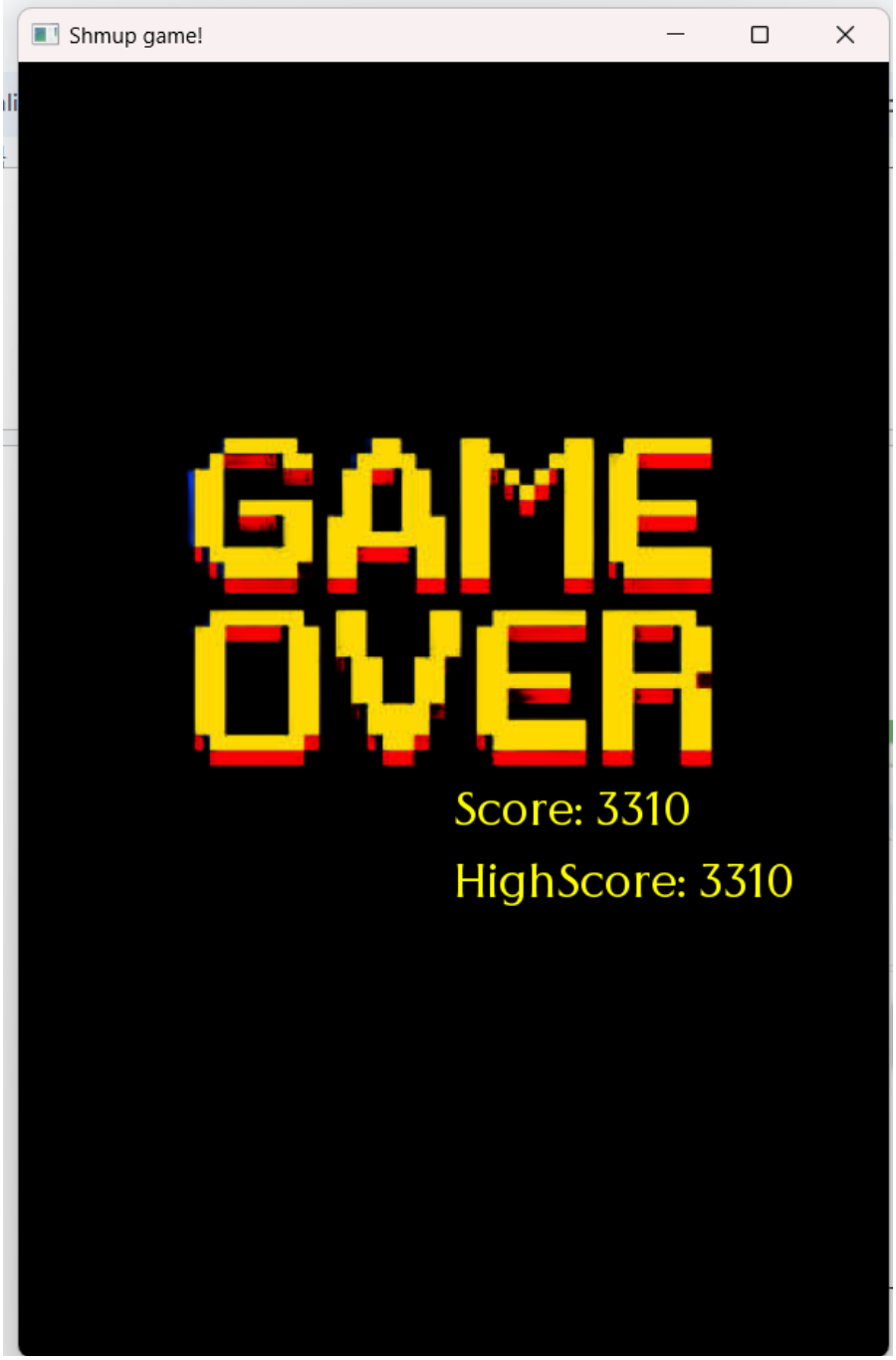


Boss Level 2

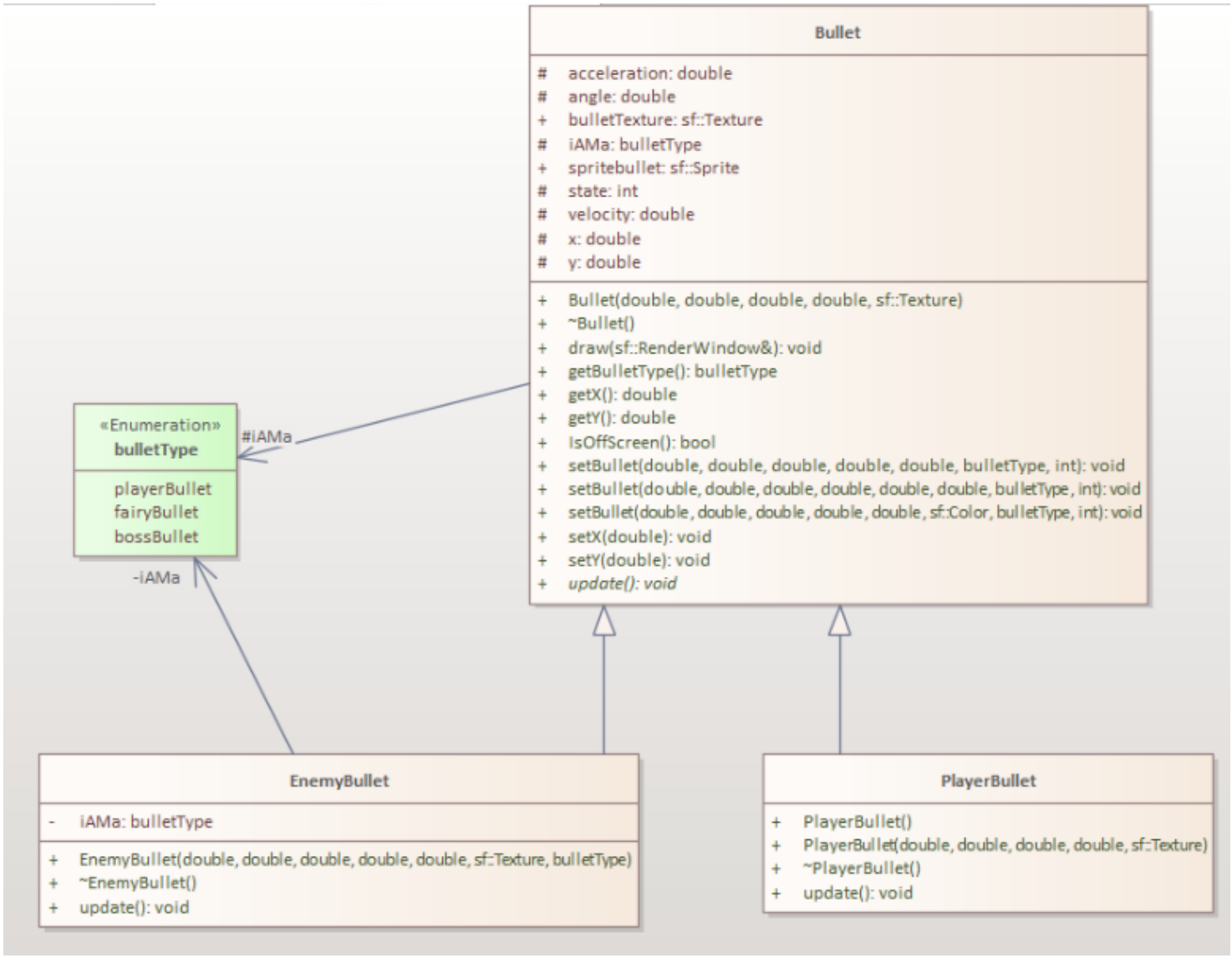


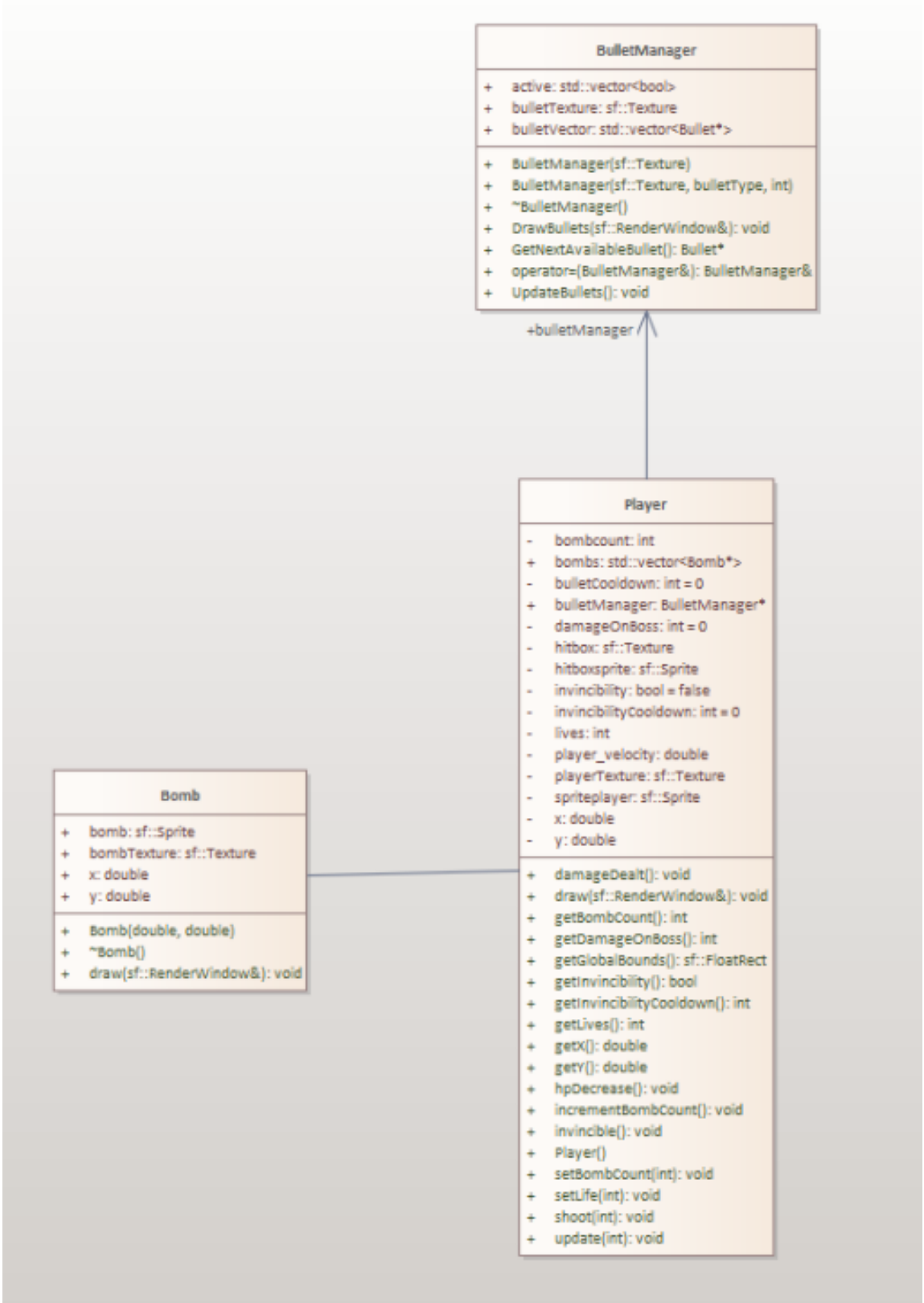
Boss Level 3

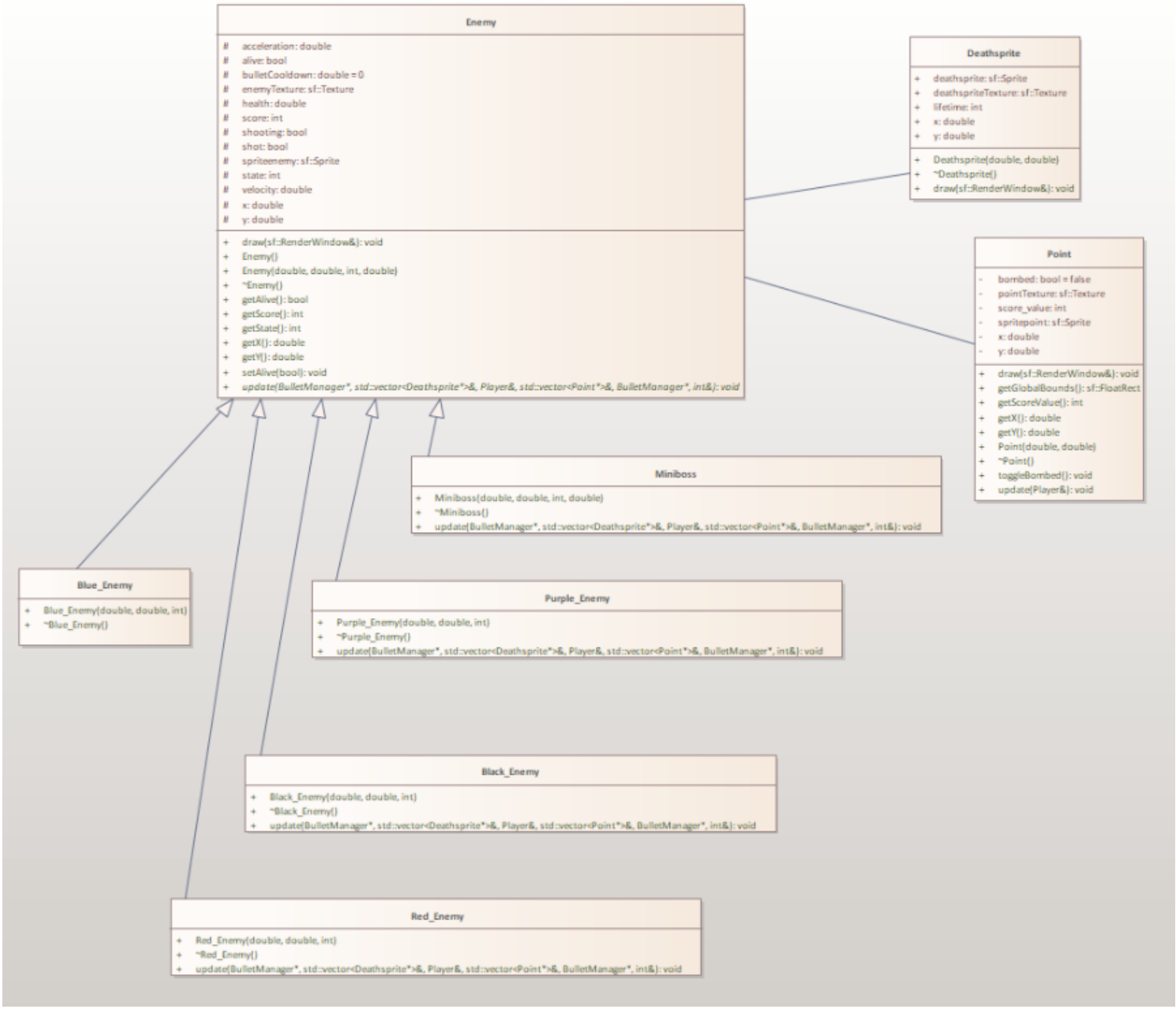
Ending Screen

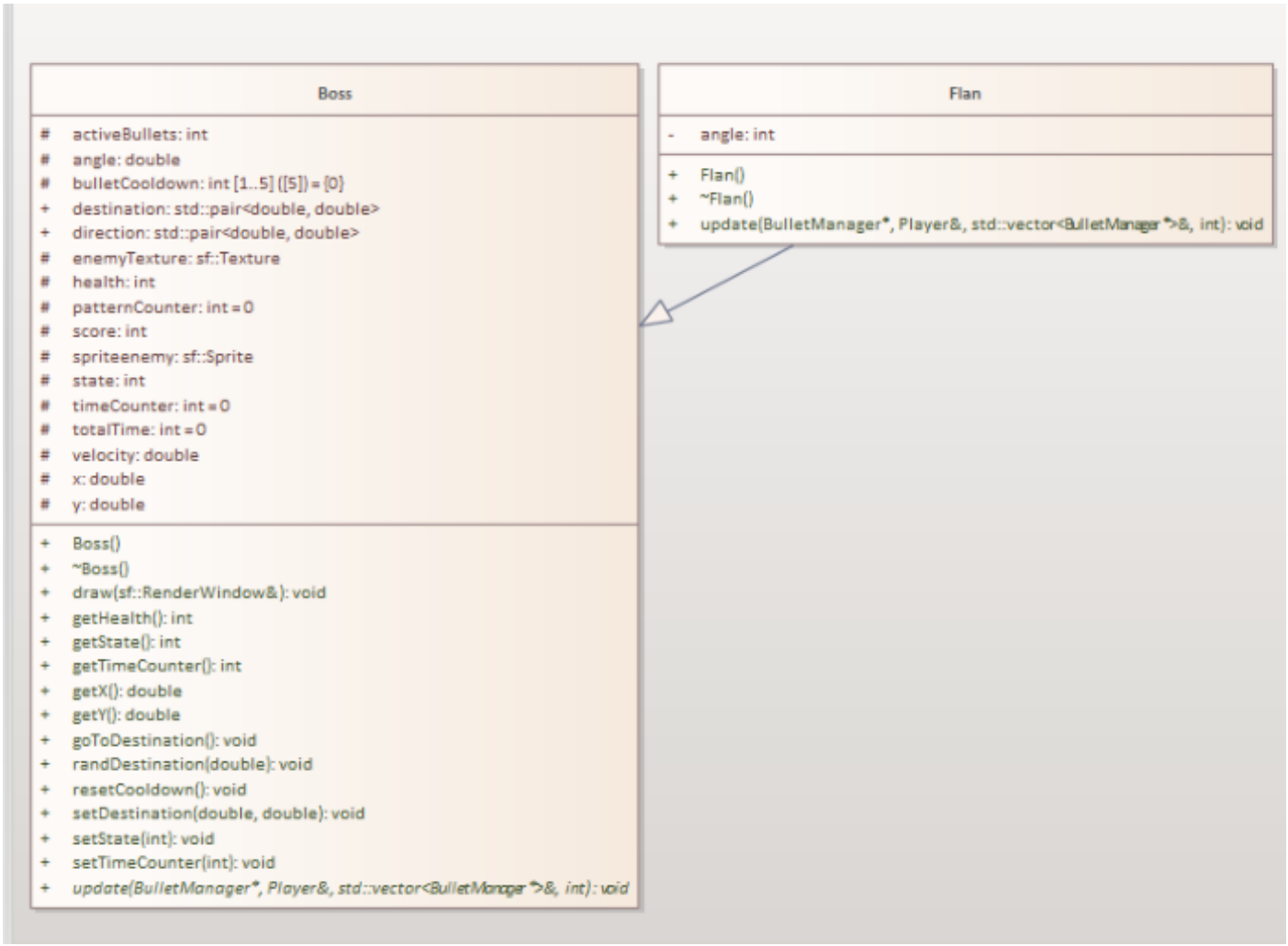


Class Diagram:




BulletManager
+ active: std::vector<bool>
+ bulletTexture: sf::Texture
+ bulletVector: std::vector<Bullet*>
+ BulletManager(sf::Texture)
+ BulletManager(sf::Texture, bulletType, int)
+ ~BulletManager()
+ DrawBullets(sf::RenderWindow&): void
+ GetNextAvailableBullet(): Bullet*
+ operator=(BulletManager&): BulletManager&
+ UpdateBullets(): void
+bulletManager
Player
- bombcount: int
+ bombs: std::vector<Bomb*>
- bulletCooldown: int = 0
+ bulletManager: BulletManager*
- damageOnBoss: int = 0
- hitbox: sf::Texture
- hitboxsprite: sf::Sprite
- invincibility: bool = false
- invincibilityCooldown: int = 0
- lives: int
- player_velocity: double
- playerTexture: sf::Texture
- spriteplayer: sf::Sprite
- x: double
- y: double
+ damageDealt(): void
+ draw(sf::RenderWindow&): void
+ getBombCount(): int
+ getDamageOnBoss(): int
+ getGlobalBounds(): sf::FloatRect
+ getInvincibility(): bool
+ getInvincibilityCooldown(): int
+ getLives(): int
+ getX(): double
+ getY(): double
+ hpDecrease(): void
+ incrementBombCount(): void
+ invincible(): void
+ Player()
+ setBombCount(int): void
+ setLife(int): void
+ shoot(int): void
+ update(int): void
Bomb
+ bomb: sf::Sprite
+ bombTexture: sf::Texture
+ x: double
+ y: double
+ Bomb(double, double)
+ ~Bomb()
+ draw(sf::RenderWindow&): void


Enemy
acceleration: double
alive: bool
bulletCooldown: double = 0
enemyTexture: sf::Texture
health: double
score: int
shooting: bool
shot: bool
spriteenemy: sf::Sprite
state: int
velocity: double
x: double
y: double
+ draw(sf::RenderWindow&): void
+ Enemy()
+ Enemy(double, double, int, double)
+ ~Enemy()
+ getAlive(): bool
+ getScore(): int
+ getState(): int
+ getX(): double
+ getY(): double
+ setAlive(bool): void
+ update(BulletManager*, std::vector<Deathsprite*>&, Player&, std::vector<Point*>&, BulletManager*, int&): void
Deathsprite
+ deathsprite: sf::Sprite
+ deathspriteTexture: sf::Texture
+ lifetime: int
+ x: double
+ y: double
+ Deathsprite(double, double)
+ ~Deathsprite()
+ draw(sf::RenderWindow&): void
Point
- bombed: bool = false
- pointTexture: sf::Texture
- score_value: int
- spritepoint: sf::Sprite
- x: double
- y: double
+ draw(sf::RenderWindow&): void
+ getGlobalBounds(): sf::FloatRect
+ getScoreValue(): int
+ getX(): double
+ getY(): double
+ Point(double, double)
+ ~Point()
+ toggleBombed(): void
+ update(Player&): void
Miniboss
+ Miniboss(double, double, int, double)
+ ~Miniboss()
+ update(BulletManager*, std::vector<Deathsprite*>&, Player&, std::vector<Point*>&, BulletManager*, int&): void
Blue_Enemy
+ Blue_Enemy(double, double, int)
+ ~Blue_Enemy()
Purple_Enemy
+ Purple_Enemy(double, double, int)
+ ~Purple_Enemy()
+ update(BulletManager*, std::vector<Deathsprite*>&, Player&, std::vector<Point*>&, BulletManager*, int&): void
Black_Enemy
+ Black_Enemy(double, double, int)
+ ~Black_Enemy()
+ update(BulletManager*, std::vector<Deathsprite*>&, Player&, std::vector<Point*>&, BulletManager*, int&): void
Red_Enemy
+ Red_Enemy(double, double, int)
+ ~Red_Enemy()
+ update(BulletManager*, std::vector<Deathsprite*>&, Player&, std::vector<Point*>&, BulletManager*, int&): void


Boss
activeBullets: int
angle: double
bulletCooldown: int [1..5] ([5]) = {0}
+ destination: std::pair<double, double>
+ direction: std::pair<double, double>
# enemyTexture: sf::Texture
# health: int
# patternCounter: int = 0
# score: int
# spriteenemy: sf::Sprite
# state: int
# timeCounter: int = 0
# totalTime: int = 0
# velocity: double
# x: double
# y: double
+ Boss()
+ ~Boss()
+ draw(sf::RenderWindow&): void
+ getHealth(): int
+ getState(): int
+ getTimeCounter(): int
+ getX(): double
+ getY(): double
+ goToDestination(): void
+ randDestination(double): void
+ resetCooldown(): void
+ setDestination(double, double): void
+ setState(int): void
+ setTimeCounter(int): void
+ update(BulletManager*, Player&, std::vector<BulletManager*>&, int): void
Flan
- angle: int
+ Flan()
+ ~Flan()
+ update(BulletManager*, Player&, std::vector<BulletManager*>&, int): void

Individual Contributions:

We divided our project down into several parts, so that each of us can help out to complete the project faster though github desktop. There are 3 main parts bullet, enemy & player and boss:

Chanasorn Howattanakulphong 65011277

- Developing bullet class and bullet management, including bullet patterns and level design.
- Collision and enemy-player interactions, along with enemy drops.
- Bux fixes and cleans up code.
- Find suitable pictures for bullets and spikes.

Napatr Sapprasert 65011381

- Developing enemy classes and level design, making different enemies AI along with their spawn algorithms.
- Developing player algorithms, controls and abilities.
- Create effect when enemies die and when players gain score.
- Bug fixes and finalizes code.

Kanokjan Singhsuwan 65011320

- Developing on Boss level, include boss bullet pattern and the AI boss behavior on the latest stage of the game.
- Create Background motion and find suitable sound effects.
- Build Title screen and End screen.
- Clean up and finalize code.