Ankara Yıldırım Beyazıt University
Department of Computer Engineering

CENG 201 – Object Oriented Programming
Course Project

KIT-201

Project Class Design Report

Uğur Kuş – 22050111002
Aziz Önder – 22050141021
Emin Salih Açıkgöz – 22050111032

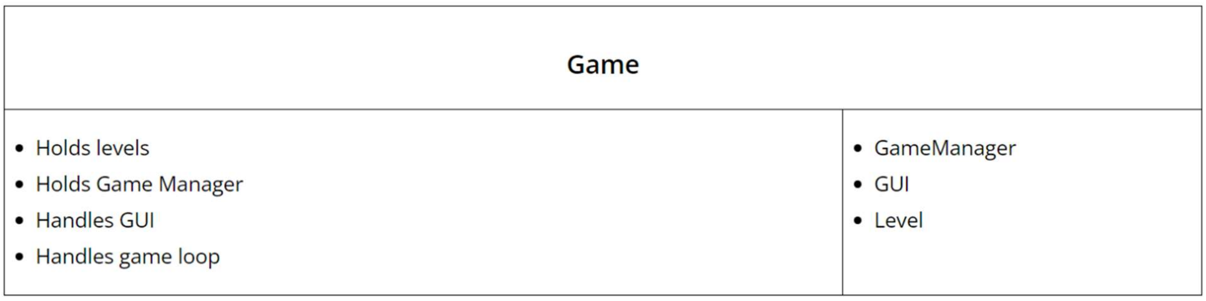
Instructor: Muhammed Abdullah Bülbül
Teaching Assistant: Elif Şanlıalp
Date: 11/12/2023

Table of Contents

1. Introduction:

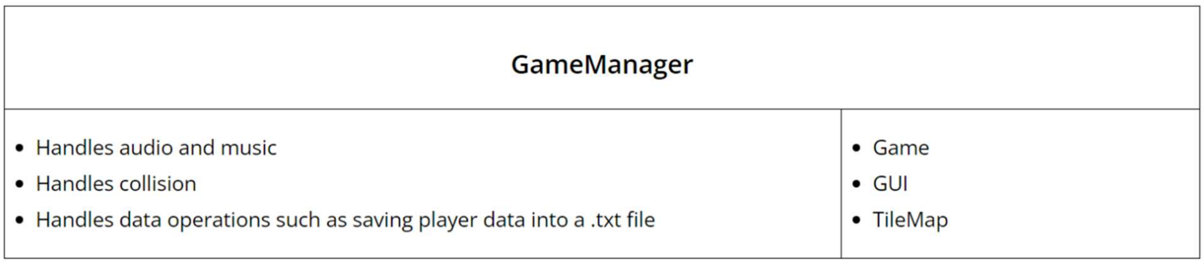
In this report, we will dive into the behaviors and interactions of each class within our system. Our CRC (Class-Responsibility-Collaboration) cards describe the names, and responsibilities between these classes. For example, the Game class is the backbone of our project. The Game class has highly important responsibilities such as holding levels, holding the GameManager, Handling the GUI, and handling the game loop itself. The UML Class diagram visually describes the methods and attributes that facilitate these behaviors. The aforementioned content will be covered in detail in their respective chapters.

2. Class-Responsibility-Collaboration (CRC) Cards:



Game	
• Holds levels • Holds Game Manager • Handles GUI • Handles game loop	• GameManager • GUI • Level

Figure 1: Game CRC Card



GameManager	
• Handles audio and music • Handles collision • Handles data operations such as saving player data into a .txt file	• Game • GUI • TileMap

Figure 2: GameManager CRC Card

GUI	
• Handles GUI and events	• Game
• Updates and renders itself	• GameManager

Figure 3: GUI CRC Card

Level	
• Generates the current level	• Game
• Loads the next level	• TileMap
• Holds Tile Map	• Entity
• Loads Tile Map	
• Holds Player and enemies	
• Spawns Player and enemies	
• Updates and renders itself	

Figure 4: Level CRC Card

TileMap	
• Holds different tiles	• Level
• Loads and updates the Tile Map	• Tile
• Detects collision	• Game Manager
• Renders itself	

Figure 5: TileMap CRC Card

Tile	
• Holds different textures and sprites for different tile types	• TileMap
• Sets the tile type	• HazardousTile
• Renders itself	

Figure 6: Tile CRC Card

HazardousTile	Tile
• Holds different textures and sprites for different hazardous tile types	• Tile
• Sets the tile type	• TileMap
• Renders itself	• Player
• Damages the player	

Figure 7: HazardousTile CRC Card

Abstract **Entity**	
• Holds different textures and sprites for different entity types	• Level
• Holds data like hp, maximum hp, position, velocity, acceleration, gravity, etc. for different entity types	• Player
• Initializes variables	• Enemy
• Initializes textures	• Enemy_A
• Initializes physics	• Enemy_B
• Initializes animation	• Enemy_C
• Renders and updates itself	

Figure 8: Entity CRC Card

Player	Entity
• Handles player input	• Level
• Moves	• Entity
• Loses HP	• Enemy
• Gains HP	• Enemy_A
• Shoots Bullet	• Enemy_B
• Jumps	• Enemy_C
• Collides with the enemies and the environment	• Bullet

Figure 9: Player CRC Card

Abstract **Enemy**	Entity
• Moves • Attacks player character • Takes damage • Collides with the player and the environment	• Level • Entity • Player • Enemy_A • Enemy_B • Enemy_C • Bullet

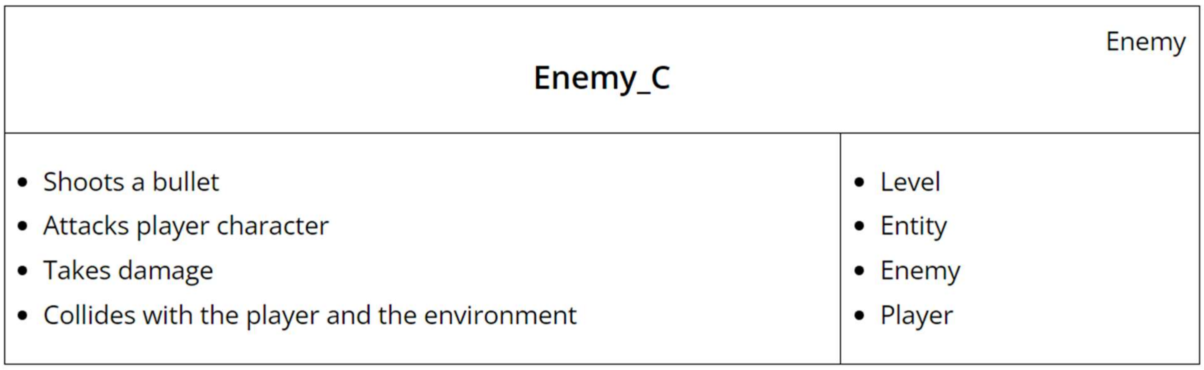
Figure 10: Enemy CRC Card

Enemy_A	Enemy
• Moves • Jumps • Attacks player character • Takes damage • Shoots a Bullet • Collides with the player and the environment	• Level • Entity • Enemy • Player • Bullet

Figure 11: Enemy_A CRC Card

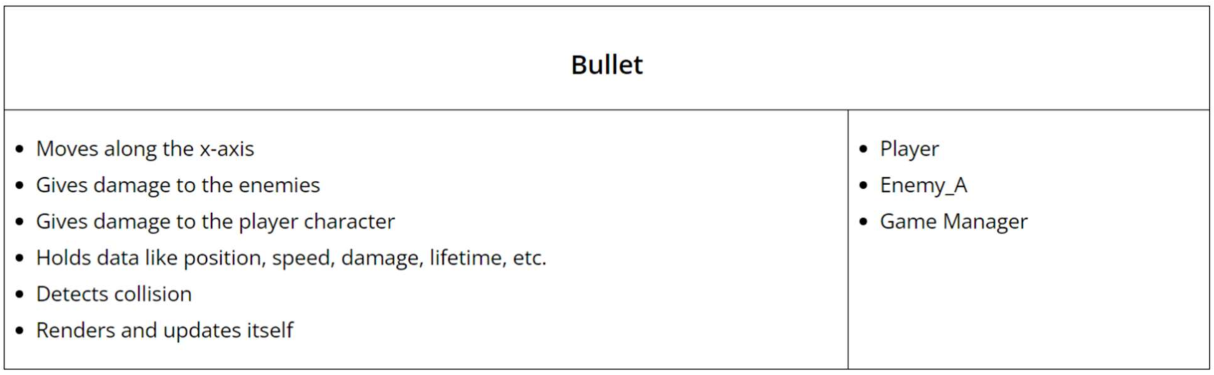
Enemy_B	Enemy
• Moves • Attacks player character • Takes damage • Collides with the player and the environment	• Level • Entity • Enemy • Player

Figure 12: Enemy_B CRC Card



Enemy_C	Enemy
• Shoots a bullet • Attacks player character • Takes damage • Collides with the player and the environment	• Level • Entity • Enemy • Player

Figure 13: Enemy_C CRC Card



Bullet	
• Moves along the x-axis • Gives damage to the enemies • Gives damage to the player character • Holds data like position, speed, damage, lifetime, etc. • Detects collision • Renders and updates itself	• Player • Enemy_A • Game Manager

Figure 14: Bullet CRC Card

3. Class Diagram:

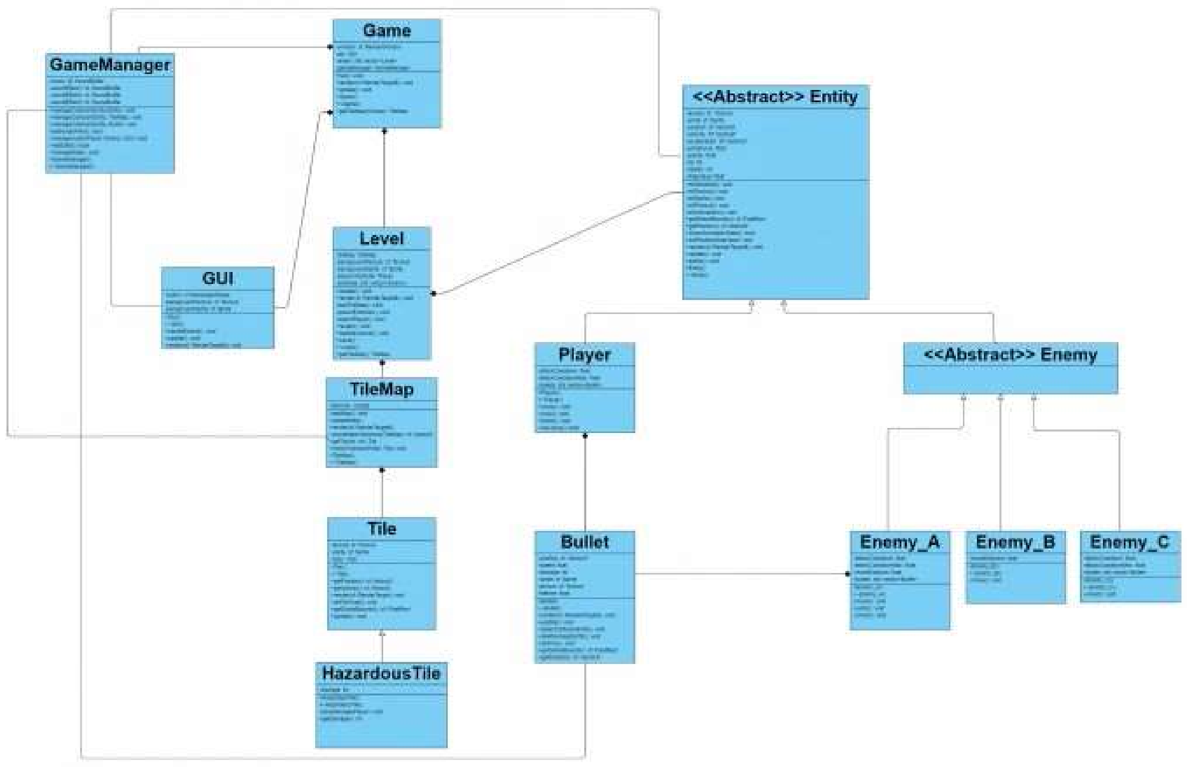


Figure 15: The full UML Class diagram. It depicts the relationship between each class.

- This is our entire UML Class diagram. It shows the relation between every class, method, and data field that each class will have. Let's break it down into each section.

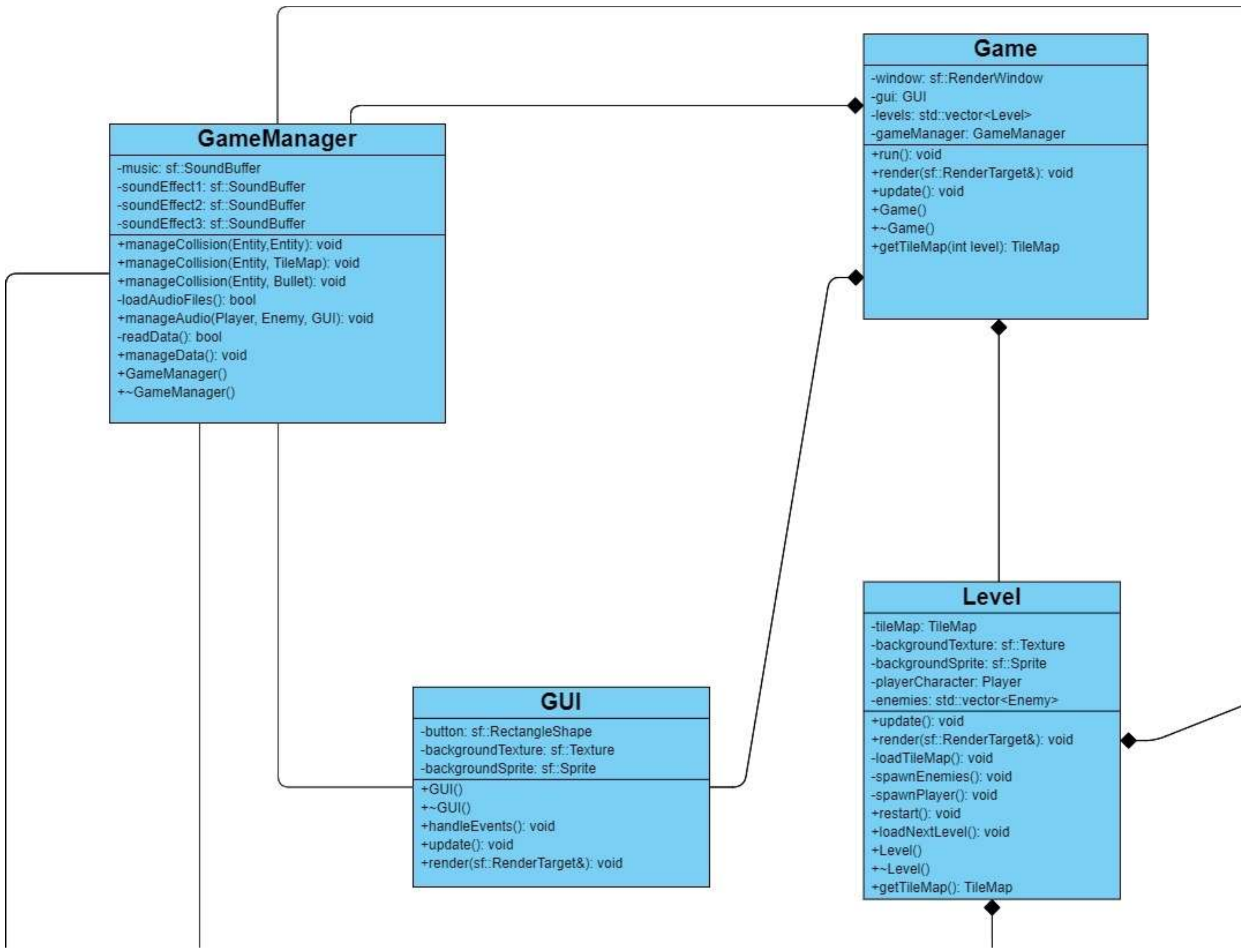

Game
-window: sf::RenderWindow
-gui: GUI
-levels: std::vector<Level>
-gameManager: GameManager
+run(): void
+render(sf::RenderTarget&): void
+update(): void
+Game()
+~Game()
+getTileMap(int level): TileMap
GameManager
-music: sf::SoundBuffer
-soundEffect1: sf::SoundBuffer
-soundEffect2: sf::SoundBuffer
-soundEffect3: sf::SoundBuffer
+manageCollision(Entity,Entity): void
+manageCollision(Entity, TileMap): void
+manageCollision(Entity, Bullet): void
-loadAudioFiles(): bool
+manageAudio(Player, Enemy, GUI): void
-readData(): bool
+manageData(): void
+GameManager()
+~GameManager()
GUI
-button: sf::RectangleShape
-backgroundTexture: sf::Texture
-backgroundSprite: sf::Sprite
+GUI()
+~GUI()
+handleEvents(): void
+update(): void
+render(sf::RenderTarget&): void
Level
-tileMap: TileMap
-backgroundTexture: sf::Texture
-backgroundSprite: sf::Sprite
-playerCharacter: Player
-enemies: std::vector<Enemy>
+update(): void
+render(sf::RenderTarget&): void
-loadTileMap(): void
-spawnEnemies(): void
-spawnPlayer(): void
+restart(): void
+loadNextLevel(): void
+Level()
+~Level()
+getTileMap(): TileMap

Figure 16: The Game, GameManager, GUI, and Level classes.

- This section contains the Game, GameManager, GUI, and Level classes. The Game class contains the main loop for the game, the GameManager, the GUIs, the Levels, and the Render Window that our game will be running inside of. The GameManager class contains methods that handle audio, saving data, loading data, and collisions. The Level class contains the tiles and entities that make up the level, sprites for the level, and methods to render and update the level. The GUI class contains an event Handler and methods to render itself and update.

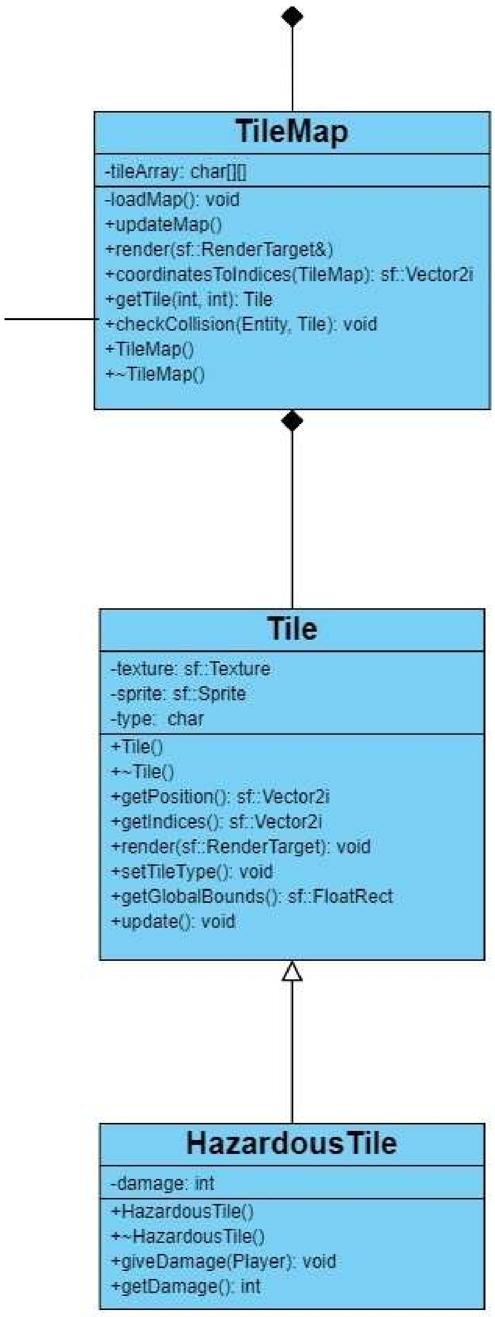


Figure 17: The TileMap, Tile, and HazardousTile classes.

- This section contains the TileMap, Tile, and HazardousTile classes. The TileMap class contains a 2D array of Tiles that will represent the level. TileMap also contains methods for checking collision, modifying tiles, obtaining and manipulating coordinates, updating, and rendering. The Tile class contains methods and attributes for rendering, getting position, and updating. The HazardousTile class has methods that deal damage to the player if a collision with the HazardousTile is made.

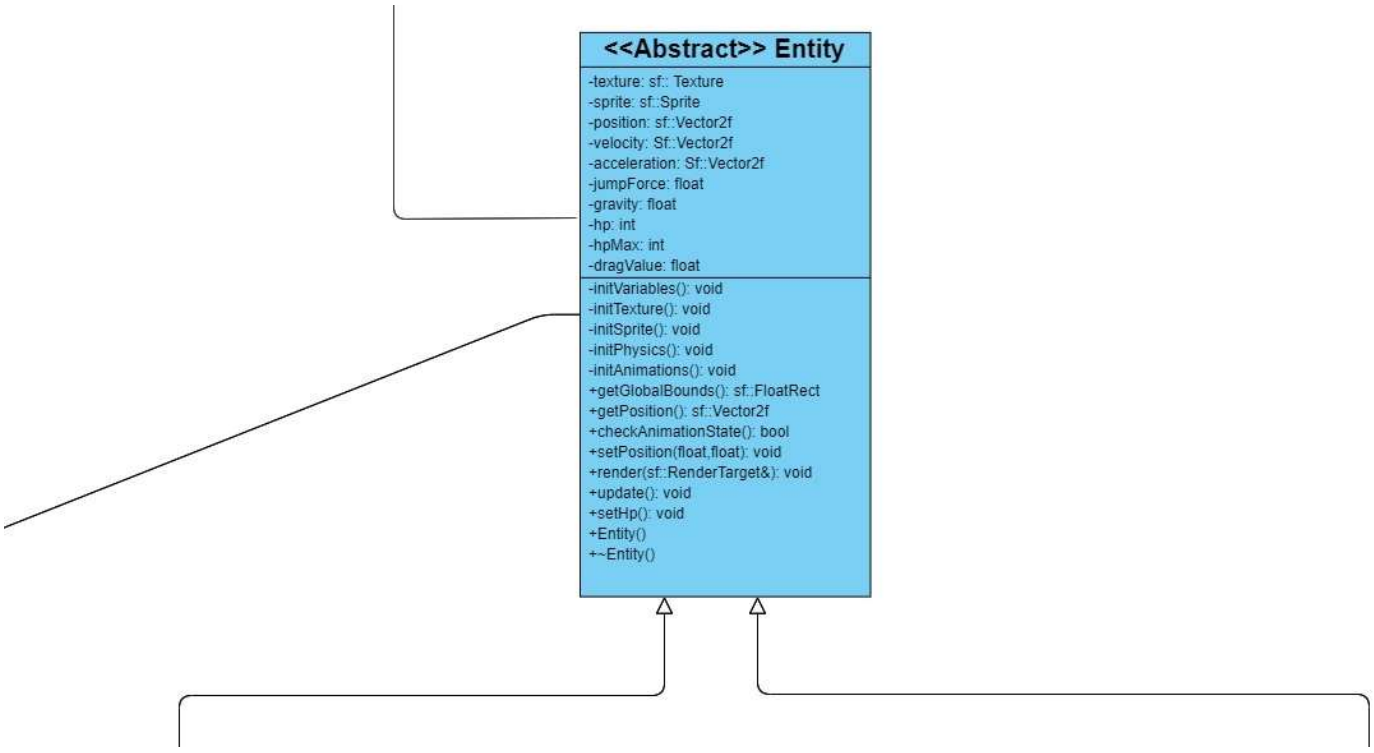


Figure 18: The Abstract Entity class.

- This section contains the Abstract Entity class. It contains attributes that allow it to have sprites, textures, velocity, position, jumpForce, health, etc. It contains methods for initializing attributes, checking the animation state, manipulating its position, modifying hp, updating, and rendering.

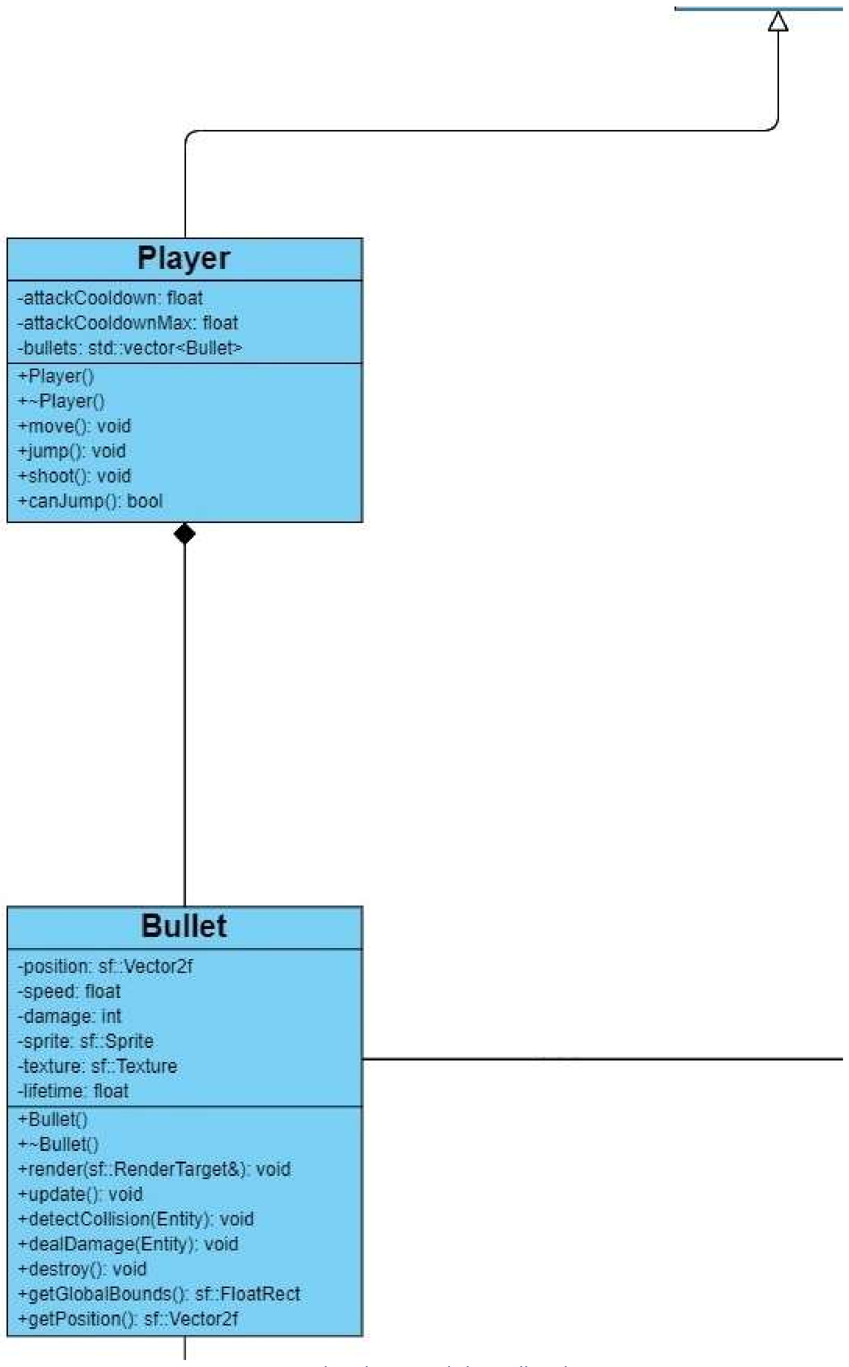


Figure 19: The Player and the Bullet class.

- This section contains the Player and Bullet classes. The Player class has attributes that determine its attack speed and the bullets it can use. Keyboard input allows the player to move the character around, jump, and shoot. The Bullet class contains attributes that hold its position, speed, damage, sprite, texture, and lifetime. The Bullet class has methods that deal damage to entities, check collision, destroy the bullet, render, and update the Bullet.

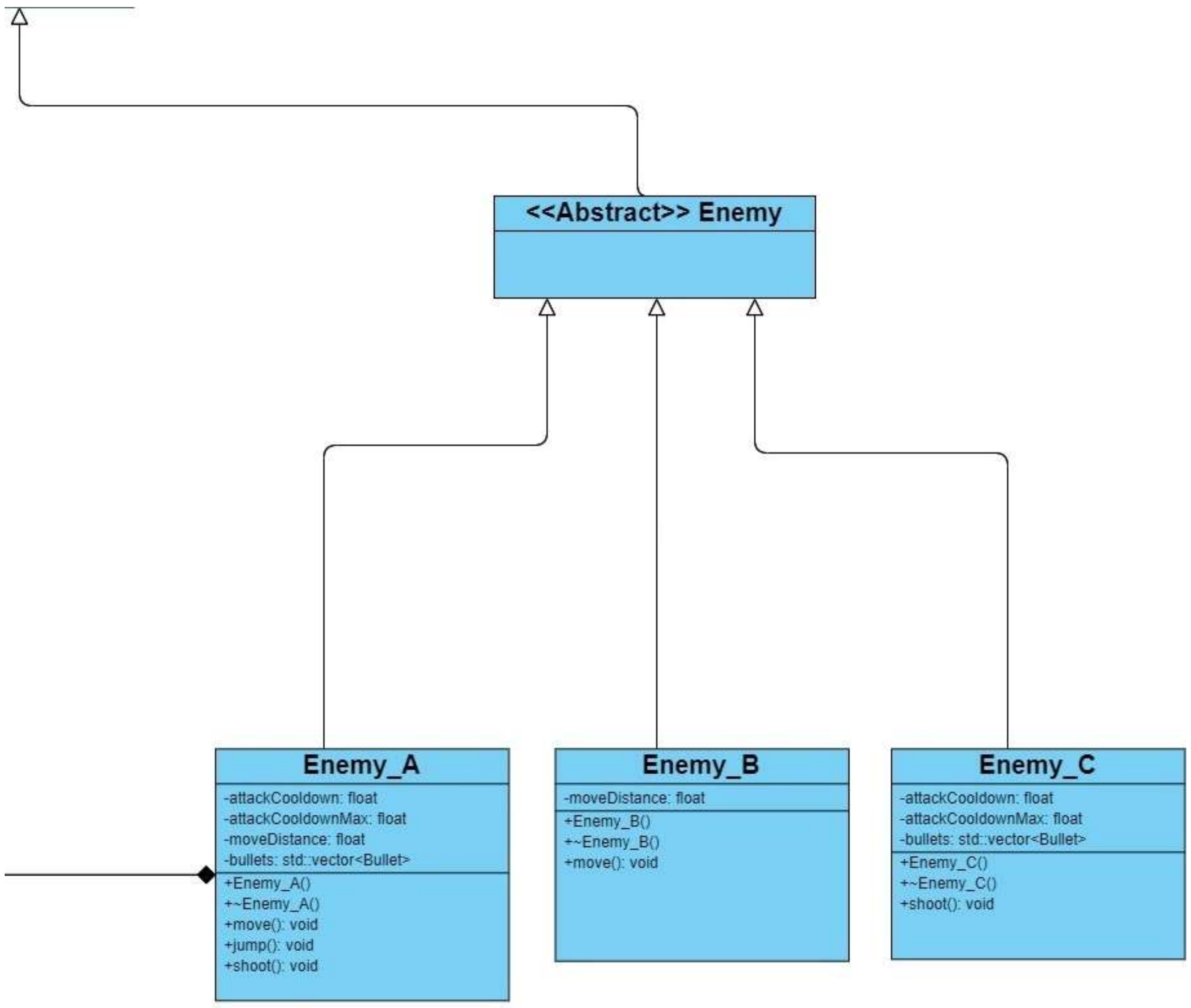


Figure 20: The Enemy class and its subclasses.

- This is the section where the Enemy entities are. Enemy entities are obstacles to the player that deal damage. They come in different subtypes that possess different behaviors. In general, they will attempt to collide with the player, shoot projectiles, and move around.

4. Conclusion:

This report goes over the responsibilities and collaborations of each class we will implement. The main game loop is responsible for maintaining the game itself. The GameManager is responsible for handling various behaviors such as playing audio, handling game physics, loading the game, and saving the game. The Level class contains a TileMap that represents the tiles in the level and the entities that will reside in the level alongside the player. The entity class is an abstract class that holds key methods and data fields all its inheritance will use, such as health. The player class contains methods that allow players to move their characters around and shoot enemies. The bullet class contains methods that make it a projectile that moves along the x-axis and damages entities. The UML Class diagram contains the methods that will be used to have such behavior.

Contributions:

Aziz Önder: Prepared the UML Class Diagram, wrote CRC cards, and revised CRC cards.

Emin Salih Açıkgöz: Wrote the report, wrote CRC cards, and revised CRC cards.

Uğur Kuş: Wrote CRC cards, reviewed the UML Class Diagram, and revised CRC cards.