Project Proposal

Project Group 33

Title: Tower Defence Game
Team Members: Sufiyan Aman, Muhammad Affan, Shaheer Bin Tariq

Project Overview

Our project is a tower defense game, built using the SFML library, where players deploy **Defenders(towers)** strategically along a path to protect their base from waves of advancing enemies. The game's core challenge lies in resource management and strategic placement, as enemies become progressively stronger with each wave. Players earn resources to upgrade or add more Defenders, creating an engaging balance between offense and defense.

Game Description and Features

- The game opens with a **welcome screen** that invites players to start their mission, view instructions, or exit. Once in the game, players can place Defenders(towers) along a predetermined path to intercept and eliminate waves of enemies heading toward the base.
- Each Defender type has unique abilities, such as varied attack ranges, firing speeds, and damage. Players earn resources by defeating enemies, allowing them to upgrade existing Defenders(towers) or place new ones strategically. As waves progress, the player must adjust their defense tactics, utilizing the strengths of different Defender types to handle increasingly resilient enemy waves. The **sound and visual design** immerses players with background music, sound effects for Defender actions, and visually distinct sprites for Defenders and enemies, making each gameplay moment feel dynamic and engaging.
- Enemies follow a set path toward the player's base, and each enemy reaching the base reduces the player's health, increasing the urgency for efficient defense placement. The game's progression is marked by waves, with each new wave becoming more challenging. The player's objective is to survive as many waves as possible.

OOP Design and Structure

The game’s OOP design centers on modular, reusable classes with a clear hierarchy and relationships:

- **Defenders and Enemies**: The `Defender` and `Enemy` classes serve as base classes, with specific subclasses introducing varied behaviors. Each Defender subclass has different firing styles, ranges, and effects, while Enemy subclasses allow for distinct enemy types with unique attributes like speed and health.
- **Class Interactions**: Composition, aggregation, and association are used to manage relationships among `Defender`, `Bullet`, `Enemy`, `Wave`, and other classes. This modularity enhances gameplay and allows for future expansion.

Class Structure

Class	Description
Game	Manages the main game loop, initializing all game elements, updating game state, and rendering objects.
Player	Tracks player stats, including health, score, and resources available for deploying and upgrading Defenders.
Tower	Represents a base class for all Defender types, each with unique attributes like range, damage, and fire rate. Subclasses may represent specialized Defenders with distinct firing styles or damage types.
Enemy	Defines enemy attributes like health, speed, and rewards upon defeat. Each enemy follows the path to the base and interacts with Defenders through combat.

Bullet Represents projectiles fired by Defenders, with attributes for speed and damage. Bullets move toward enemies and apply damage upon impact.

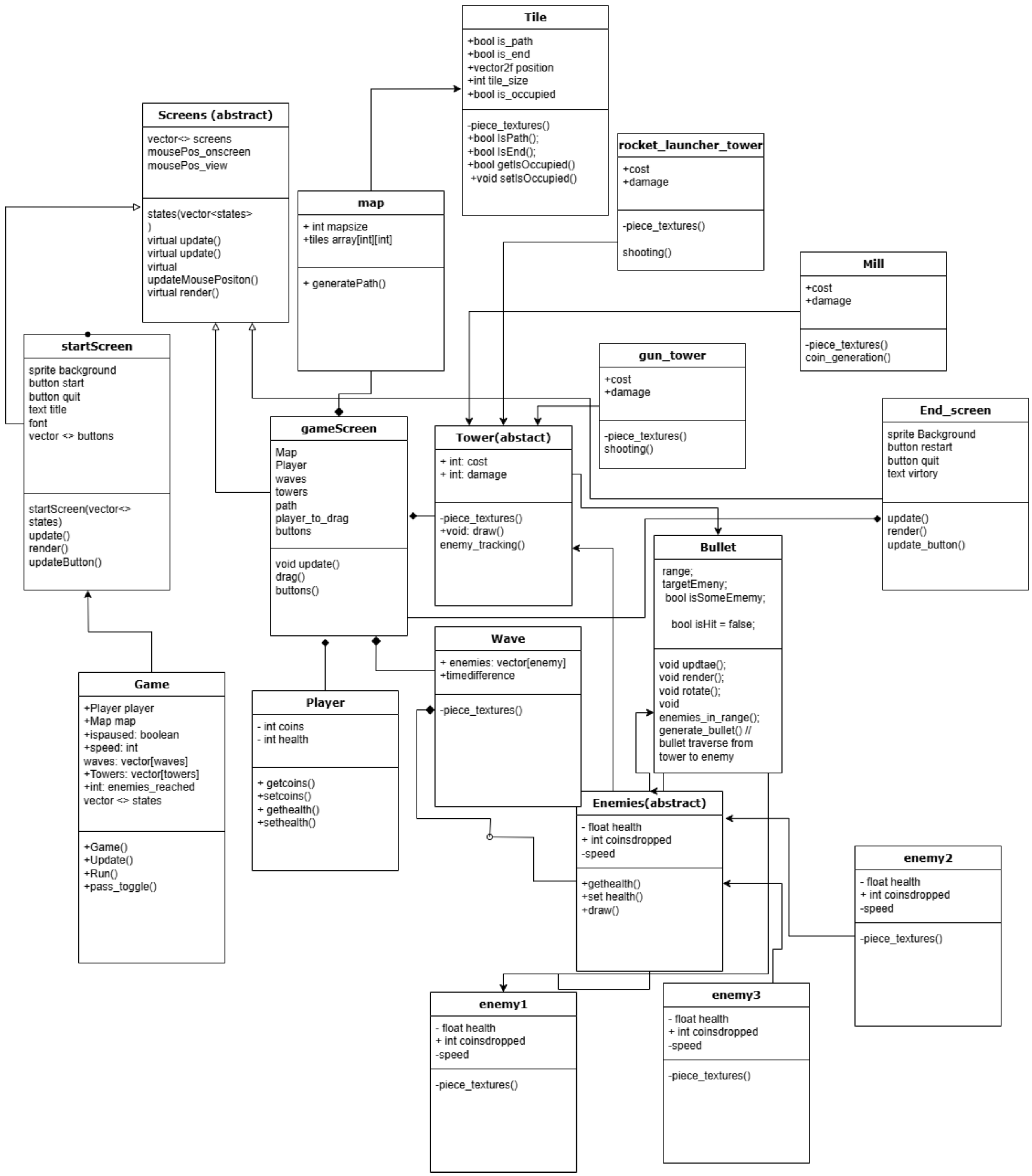
Path Manages the route that enemies follow, providing waypoints for navigation and controlling movement patterns.

Wave Manages the spawning and timing of enemies in sequential waves, increasing difficulty progressively.

Spawner Controls the spawn mechanics for enemies within each wave, dictating spawn timing and initial enemy placement.

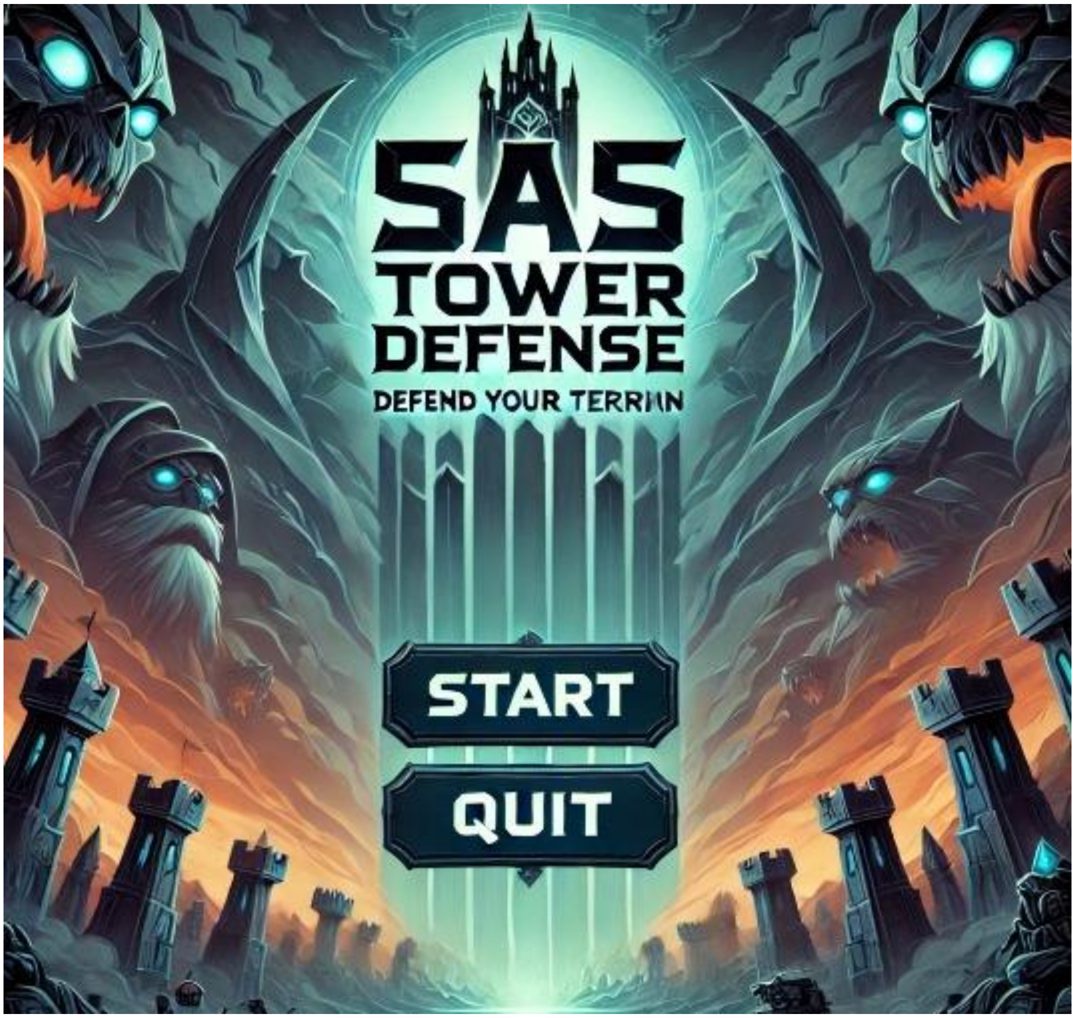
GameController Oversees game states like start, pause, and end, as well as overall game progression and user input management.

UML:


Tile
+bool is_path
+bool is_end
+vector2f position
+int tile_size
+bool is_occupied
-piece_textures()
+bool IsPath();
+bool IsEnd();
+bool getIsOccupied()
+void setIsOccupied()
Screens (abstract)
vector<> screens
mousePos_onscreen
mousePos_view
states(vector<states>
)
virtual update()
virtual update()
virtual
updateMousePositon()
virtual render()
map
+ int mapsize
+tiles array[int][int]
+ generatePath()
rocket_launcher_tower
+cost
+damage
-piece_textures()
shooting()
Mill
+cost
+damage
-piece_textures()
coin_generation()
startScreen
sprite background
button start
button quit
text title
font
vector <> buttons
startScreen(vector<>
states)
update()
render()
updateButton()
gun_tower
+cost
+damage
-piece_textures()
shooting()
gameScreen
Map
Player
waves
towers
path
player_to_drag
buttons
void update()
drag()
buttons()
Tower(abstact)
+ int: cost
+ int: damage
-piece_textures()
+void: draw()
enemy_tracking()
End_screen
sprite Background
button restart
button quit
text virtory
update()
render()
update_button()
Bullet
range;
targetEmeny;
bool isSomeEmemy;
bool isHit = false;
void updtae();
void render();
void rotate();
void
enemies_in_range();
generate_bullet() //
bullet traverse from
tower to enemy
Wave
+ enemies: vector[enemy]
+timedifference
-piece_textures()
Game
+Player player
+Map map
+ispaused: boolean
+speed: int
waves: vector[waves]
+Towers: vector[towers]
+int: enemies_reached
vector <> states
+Game()
+Update()
+Run()
+pass_toggle()
Player
- int coins
- int health
+ getcoins()
+setcoins()
+ gethealth()
+sethealth()
Enemies(abstract)
- float health
+ int coinsdropped
-speed
+gethealth()
+set health()
+draw()
enemy2
- float health
+ int coinsdropped
-speed
-piece_textures()
enemy1
- float health
+ int coinsdropped
-speed
-piece_textures()
enemy3
- float health
+ int coinsdropped
-speed
-piece_textures()

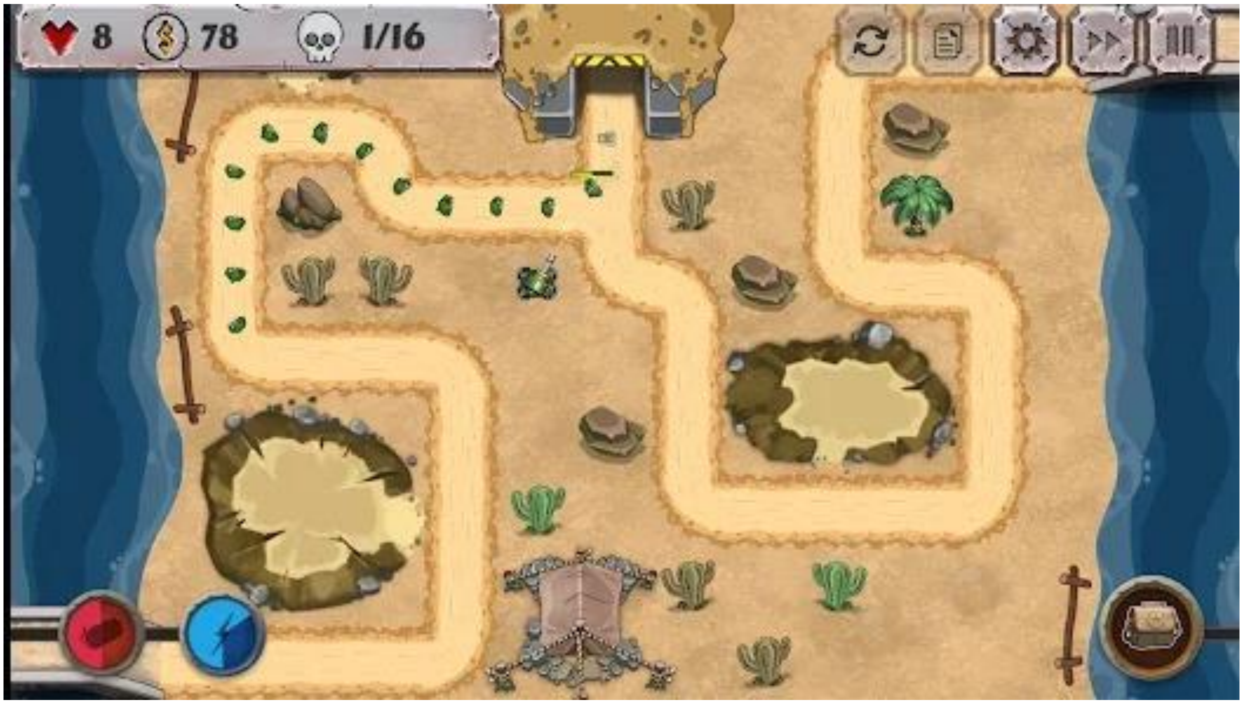
Sample Screens:

Welcome Screen:



Welcome Screen: This png will be uploaded for the main menu screen and two buttons “start” and “quit” will be added. By pressing “start”, the game screen will appear.

Game Screen:



Game Screen: Game screen will look like this one but will be simpler than this. In this screen, there will be a tower defense setup where the player's defenders (towers) will be positioned along a winding path, ready to intercept and attack incoming enemy units. Each defender has a specific range and damage level, indicated by their positions and attack animations. Enemies move along the path towards a base or endpoint, and the player must strategically place defenders to prevent enemies from reaching this destination. Key information like player resources, wave count, and health status will be displayed at the top, allowing players to monitor their progress and adjust their strategies accordingly.