Department of Computer Science and Engineering
Institute of Technical Education & Research, SOA, Deemed to be University

MINOR ASSIGNMENT-06

Game Programming with C++ (CSE 3545)

Publish on: 10-05-2025 **Submission on:** 15-05-2025
Course Outcome: CO_5 **Program Outcome:** PO_4 **Learning Level:** L_5

Problem Statement:

Experiment with SFML **`sf::View`** (i.e. 2D camera that defines what region is shown on screen) to divide the game up into layers and development of player & zombie classes for the Zombie Arena game.

Learning Objectives:

Students will be able to scroll, rotate or zoom the entire scene without altering the way that their drawable objects are drawn and also able to design the required classes for building the Zombie Arena game engine.

Answer the followings:

1. The graphical assets make up the parts of the scene of the Zombie Arena game. State how many graphical assets are present in our game. Also give a count of different images in `background_sheet.png` file.

Count of graphical assets and images

Total: 12 main graphical assets 4 distinct images

2. The sound files in Zombie Arena game are all in `.wav` format. These are files that contain the sound effects that will be played when certain **events** are triggered. Write the sound file names and when it would be played.

Sound files & purpose

```
/*
hit.wav
pickup.wav
powerup.wav
reload_failed.wav
reload.wav
shoot.wav
splat.wav
*/
```

3. Write the code snippet to create a view from a rectangle as well as a vew from its center and size.

Create view

```
// From a rectangle
sf::View view1(sf::FloatRect(0, 0, 1920, 1080));
// From center and size
sf::View view2;
view2.setCenter(sf::Vector2f(960, 540));
view2.setSize(sf::Vector2f(1920, 1080));
```

4. The view in SFML is like a 2D camera. It controls which part of the 2D scene is visible, and how it is viewed in the render target. The new view will affect everything that is drawn, until another view is set. Write the SFML-C++ statement(s) to set a view to be displayed in the window and draw everything related to it.

Create view

```
sf::View cameraView(sf::FloatRect(0.f, 0.f, 800.f, 600.f));

window.setView(cameraView);
window.draw(sprite);
window.draw(player);
window.draw(background);
```

5. Write the appropriate variable name(s) and/or function name(s) in the place of **?** symbol assuming the image **player.png** has a resolution of 50×50 pixels.

```
class Player{
  private:
     Texture ?
     Sprite ?
  public:
     Player();
};
Player::Player(){
        ? . ?("player.png");
        ? .setTexture(?);
        ?. setOrigin(?,?);
        ? . ?(120,234);
}
```

Complete the code snippet

```
class Player {
private:
   sf::Texture texture;
   sf::Sprite sprite;
public:
   Player();
};

Player::Player() {
   texture.loadFromFile("player.png");
   sprite.setTexture(texture);
   sprite.setOrigin(25.f, 25.f);
   sprite.setPosition(120.f, 234.f);
}
```

6. Design a player class with optimal number of private and public members to draw the player sprite at the center of the defined view of the window.

Player class

```
class Player {
private:
   sf::Sprite m_Sprite;
   sf::Texture m_Texture;
   sf::Vector2f m_Position;

public:
   Player();
   void draw(sf::RenderWindow& window);
};
```

```
Player::Player() {
   m_Texture.loadFromFile("player.png");
   m_Sprite.setTexture(m_Texture);
   m_Sprite.setOrigin(25, 25);
   m_Position = sf::Vector2f(960, 540);
   m_Sprite.setPosition(m_Position);
}

void Player::draw(sf::RenderWindow& window) {
   window.draw(m_Sprite);
}
```

7. The following Figure:1 shows event handling by polling. Develop a code snippet to instantiate an object of `Event` type (`Event` is a SFML class type) to poll for the system events. Additionally, include a loop with the condition `window.pollEvent(...)` to keep looping each frame until there are no events to process and display the appropriate message when there is a state change.

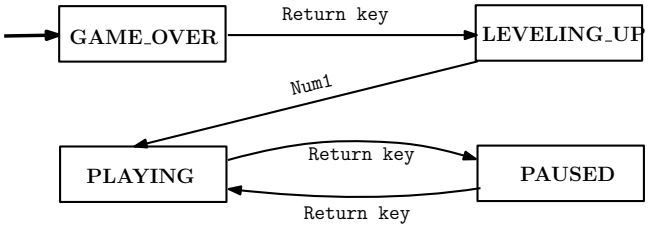

Figure 1: Game state transition for handling events by polling

Code Snippet

```
enum GameState { PLAYING, PAUSED, LEVELING_UP, GAME_OVER };
int main() {
    sf::RenderWindow window(sf::VideoMode(800, 600), "Zombie Arena - State Handling");
    sf::Event event;
    GameState currentState = GAME_OVER;
    while (window.isOpen()) {
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
            if (event.type == sf::Event::KeyPressed) {
                switch (currentState) {
                    case GAME_OVER:
                        if (event.key.code == sf::Keyboard::Enter) {
                            currentState = LEVELING_UP;
                            std::cout << "State changed to LEVELING_UP\n";
                        }
                        break;

                    case LEVELING_UP:
                        if (event.key.code == sf::Keyboard::Num1) {
                            currentState = PLAYING;
                            std::cout << "State changed to PLAYING\n";
                        }
                        break;

                    case PLAYING:
                        if (event.key.code == sf::Keyboard::Return) {
                            currentState = PAUSED;
                            std::cout << "State changed to PAUSED\n";
                        }
                        break;
                    case PAUSED:
                        if (event.key.code == sf::Keyboard::Return) {
                            currentState = PLAYING;
                            std::cout << "State changed to PLAYING\n";
                        }
                        break;
                }
            }
        }
        window.clear();
        window.display();
    }
    return 0;
}
```

8. Re-write the `spawn` public member function for the **Player** class to spawn 5 players as shown in the below Figure-2. Also state the SFML statements to call the **`spawn(...)`** function and **`window.draw(...)`** for the player.

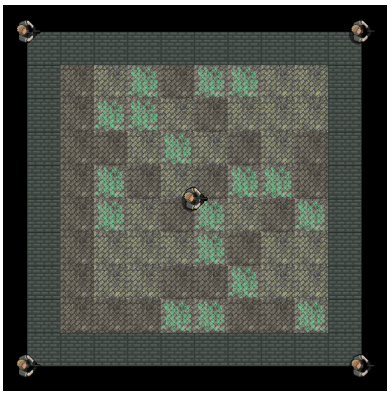

Figure 2: Player spawn at different position of the arena

Code Snippet

```
#pragma once
#include <SFML/Graphics.hpp>

class Player {
private:
    sf::Texture texture;
    sf::Sprite sprite;

public:
    Player(const std::string& texturePath);
    void spawn(float x, float y);
    void draw(sf::RenderWindow& window);
};
```

```
#include "Player.hpp"

Player::Player(const std::string& texturePath) {
    texture.loadFromFile(texturePath);
    sprite.setTexture(texture);

    sf::Vector2u size = texture.getSize();
    sprite.setOrigin(size.x / 2.f, size.y / 2.f);
}

void Player::spawn(float x, float y) {
    sprite.setPosition(x, y);
}

void Player::draw(sf::RenderWindow& window) {
    window.draw(sprite);
}
```

Code Snippet

```
#include "Player.hpp"
#include <vector>

int main() {
    sf::RenderWindow window(sf::VideoMode(600, 600), "Player Spawning");

    std::vector<sf::Vector2f> positions = {
        {75.f, 75.f},
        {525.f, 75.f},
        {300.f, 300.f},
        {75.f, 525.f},
        {525.f, 525.f}
    };

    std::vector<Player> players;
    for (int i = 0; i < 5; ++i) {
        Player p("player.png");
        p.spawn(positions[i].x, positions[i].y);
        players.push_back(p);
    }

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();
        for (auto& p : players)
            p.draw(window);
        window.display();
    }

    return 0;
}
```

9. Design a public member function, **`void spawn(float startX, float startY, int type, int seed)`**, for the **Zombie** class to draw the 3 kinds of zombies over the arena as shown in the below Figure-3. Also write **THREE** function call statements to invoke the **`spawn(...)`** function with three independent objects of that class **Zombie**.

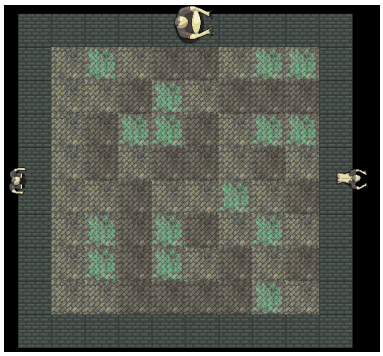

Figure 3: Zombies spawning over the arena wall

Code Snippet

```
void Zombie::spawn(float startX, float startY, int type, int seed) {
    m_Position.x = startX;
    m_Position.y = startY;
    // Choose texture based on type
    switch(type) {
        case BLOATER: m_Texture.loadFromFile("bloater.png"); break;
        case CHASER: m_Texture.loadFromFile("chaser.png"); break;
        case CRAWLER: m_Texture.loadFromFile("crawler.png"); break;
    }
    m_Sprite.setTexture(m_Texture);
    m_Sprite.setPosition(m_Position);
}

// Function calls
z1.spawn(100, 100, BLOATER, 1);
z2.spawn(300, 100, CHASER, 2);
z3.spawn(500, 100, CRAWLER, 3);
```

10. Redesign the above code snippet to use array of objects rather than 3 individual objects of the **Zombie** class. Don't write the **`spawn(...)`** function again.

Code Snippet

```
//1. Declare an Array of Zombie Objects
const int NUM_ZOMBIES = 3;
Zombie zombies[NUM_ZOMBIES];

//2. Call spawn(...) Using a Loop
float spawnX[NUM_ZOMBIES] = {300.f, 75.f, 525.f};
float spawnY[NUM_ZOMBIES] = {75.f, 300.f, 300.f};
int types[NUM_ZOMBIES]   = {0, 1, 2};
int seeds[NUM_ZOMBIES]   = {10, 20, 30};

for (int i = 0; i < NUM_ZOMBIES; ++i) {
    zombies[i].spawn(spawnX[i], spawnY[i], types[i], seeds[i]);
}

//3. Draw All Zombies in the Game Loop
for (int i = 0; i < NUM_ZOMBIES; ++i) {
    zombies[i].draw(window);
}
```

11. Redesign the above code snippet to use dynamic allocation of objects (using new) rather than array of objects of the **Zombie** class. Don't write the **`spawn(...)`** function again.

Code Snippet

```
Zombie* zombies[3];
zombies[0] = new Zombie();
zombies[1] = new Zombie();
zombies[2] = new Zombie();

zombies[0]->spawn(100, 100, BLOATER, 1);
zombies[1]->spawn(300, 100, CHASER, 2);
zombies[2]->spawn(500, 100, CRAWLER, 3);
```

12. Write SFML-C++ statement to compute the angle between the player location **`(x1,y1)`** to the BLOATER position **`(x2,y2)`**. Additionally, set the rotation of the BLOATER zombie sprite (i.e. `m_Sprite`) to that angle.

Code Snippet

```
float dx = x2 - x1;
float dy = y2 - y1;
float angle = atan2(dy, dx) * 180 / 3.14159265f;
m_Sprite.setRotation(angle);
```

13. Assume that a zombie sprite, `m_Sprite`, is to the left of the player's position (i.e. `Vector2f playerLocation`). Write SFML-C++ statement to update the zombie position variable (`m_Position`) w.r.t. the player.

Code Snippet

```
sf::Vector2f direction = playerLocation - m_Position;

float length = std::sqrt(direction.x * direction.x + direction.y * direction.y);
sf::Vector2f unitDirection = direction / length;

float speed = 1.0f;
m_Position += unitDirection * speed;

m_Sprite.setPosition(m_Position);
```

14. State the code segment to keep the player (**`m_Position.x`** & **`m_Position.y`**) is NOT beyond any of the edges of the current arena (`m_Arena`) with the surrounding wall tile size, `m_TileSize=50`.

Code Snippet

```
if (m_Position.x < m_Arena.left + m_TileSize)
    m_Position.x = m_Arena.left + m_TileSize;

if (m_Position.x > m_Arena.left + m_Arena.width - m_TileSize)
    m_Position.x = m_Arena.left + m_Arena.width - m_TileSize;

if (m_Position.y < m_Arena.top + m_TileSize)
    m_Position.y = m_Arena.top + m_TileSize;

if (m_Position.y > m_Arena.top + m_Arena.height - m_TileSize)
    m_Position.y = m_Arena.top + m_Arena.height - m_TileSize;
```

15. Write the code segment to generate a random number between 80 and 100.

Code Snippet

```
srand(time(0)); // Once in main
int number = (rand() % 21) + 80;
```