```cpp
//Maze Game
//Rhyder Quinlan
//C00223030

#include "pch.h"
#include "Player.h"
#include "Enemy.h"
#include "Bomb.h"
#include <iostream>
#include <cstdlib>
#include <string>
#include <sstream>
#include <iterator>
#include <fstream>
using namespace std;

#include <SFML/Graphics.hpp>
using namespace sf;

int **grid;
int rows, columns;

void setGrid() {
    rows = 20;
    columns = 20;
    grid = new int*[rows];

    for (int row = 0; row < rows; row++)
    {
        grid[row] = new int[columns];
    }

    ifstream gridfile("grid/grid.txt");
    string readline;

    if (gridfile.is_open())
    {
        int i = 0;
        while (getline(gridfile, readline))
        {

            for (int row = 0; row < rows; row++)
            {
                for (int column = 0; column < columns; column++)
                {
                    grid[row][column] = (readline.at(int(i)) - 48);
                    i++;
                }
            }
            cout << endl;
        }
        i = 0;
        gridfile.close();
    }
    else
    {
        cout << "Unable to open file";
    }
}

void print_array(int **g) {
    for (int x = 0; x < 20; x++)
    {
        for (int y = 0; y < 20; y++)
        {
            cout << g[x][y];
        }
        cout << endl;
    }
}

class Tilemap : public Drawable, public Transformable
{

public:

    bool generate(const std::string tileset, sf::Vector2u tileSize, int** tiles, unsigned int width, unsigned int height)
    {
        if (!m_tileset.loadFromFile(tileset))
            return false;

        m_vertices.setPrimitiveType(sf::Quads);
        m_vertices.resize(width * height * 4);

        for (unsigned int i = 0; i < width; ++i)
            for (unsigned int j = 0; j < height; ++j)
            {
                int tileNumber = tiles[j][i];

                int tu = tileNumber % (m_tileset.getSize().x / tileSize.x);
                int tv = tileNumber / (m_tileset.getSize().x / tileSize.x);

                sf::Vertex* quad = &m_vertices[(i + j * width) * 4];

                quad[0].position = sf::Vector2f(i * tileSize.x + 10, j * tileSize.y + 100);
                quad[1].position = sf::Vector2f((i + 1) * tileSize.x + 10, j * tileSize.y + 100);
                quad[2].position = sf::Vector2f((i + 1) * tileSize.x + 10, (j + 1) * tileSize.y + 100);
                quad[3].position = sf::Vector2f(i * tileSize.x + 10, (j + 1) * tileSize.y + 100);

                quad[0].texCoords = sf::Vector2f(tu * tileSize.x, tv * tileSize.y);
                quad[1].texCoords = sf::Vector2f((tu + 1) * tileSize.x, tv * tileSize.y);
                quad[2].texCoords = sf::Vector2f((tu + 1) * tileSize.x, (tv + 1) * tileSize.y);
                quad[3].texCoords = sf::Vector2f(tu * tileSize.x, (tv + 1) * tileSize.y);
            }

        return true;
    }

private:

    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
        states.transform *= getTransform();
        states.texture = &m_tileset;
        target.draw(m_vertices, states);
    }

    sf::VertexArray m_vertices;
    sf::Texture m_tileset;
};

int main()
{
    //player enemy grid section
    Player player;
    Bomb bomb;
    Enemy *enemy;
    enemy = new Enemy[10];

    setGrid();
    grid = player.p_spawn(grid);

    //spawn first 4 enemies
    for (int i = 0; i < 4; i++)
    {
        grid = enemy[i].spawn(grid);
    }

    //sfml section
    Tilemap map;
    Font roboto;
    Text playerText;
    Text bombText;
    Text timeText;
    Clock gameclock;

    if (!roboto.loadFromFile("Roboto-Regular.ttf"))
    {
        cout << "could not load roboto file" << endl;
    }

    playerText.setFont(roboto);
    bombText.setFont(roboto);
    timeText.setFont(roboto);
    playerText.setString("Lives: " + to_string(player.getLives()) + "
      Score: " + to_string(bomb.getScore()));
    playerText.setCharacterSize(24);
    bombText.setCharacterSize(24);
    timeText.setCharacterSize(24);
    playerText.setFillColor(Color::White);
    bombText.setFillColor(Color::White);
    timeText.setFillColor(Color::White);
    playerText.setPosition(50, 60);
    bombText.setPosition(400, 60);
    timeText.setPosition(700, 60);

    RenderWindow window(VideoMode(980,1050), "Maze Game");
    RenderWindow end_window(VideoMode(600, 300), "Maze Game");
    end_window.setVisible(false);
    window.setKeyRepeatEnabled(false);

    Clock enemyclock;
    Clock bombclock;
    Clock explosionclock;
    Clock playerclock;

    int bomb_count = 0;
    int explosion = 0;
    int enemy_count = 4;
    while (window.isOpen()) {
        if (bomb_count == 0)
        {
            bombclock.restart();
        }
        if (explosion == 0)
        {
            explosionclock.restart();
        }

        if (bomb.getActiveEnemies() != 4)
        {
            if (enemy_count < 10)
            {
                for (int i = 0; i < (4 - bomb.getActiveEnemies()); i++)
                {
                    grid = enemy[enemy_count].spawn(grid);
                    enemy_count++;
                }
                bomb.setActiveEnemies(4);
            }

        }

        Event event;
        Time gametime = gameclock.getElapsedTime();
        Time enemytime = enemyclock.getElapsedTime();
        Time bombtime = bombclock.getElapsedTime();
        Time explosiontime = explosionclock.getElapsedTime();
        Time playertime = playerclock.getElapsedTime();
        while (window.pollEvent(event))
        {
            //close command
            if (event.type == Event::Closed)
                window.close();

            //player movement
            if (playertime.asMicroseconds() > 100000)
            {
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
                {
                    grid = player.move(0, -1, grid);
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
                {
                    grid = player.move(0, 1, grid);
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
                {
                    grid = player.move(-1, 0, grid);
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
                {
                    grid = player.move(1, 0, grid);
                }
                playerclock.restart();
            }

            if (Keyboard::isKeyPressed(Keyboard::Space))
            {
                if (bomb_count == 0)
                {
                    bomb_count = 1;
                    grid = bomb.dropBomb(grid, player.getX(), player.getY
                  ());
                    bombclock.restart();
                }
            }

            if (Keyboard::isKeyPressed(Keyboard::Q))
            {
                bomb.setScore(10);
            }
        }

        //enemy movement timer
        if (enemytime.asMicroseconds() > 700000)
        {
            for (int j = 0; j < 10; j++)
            {
                if (enemy[j].getActive())
                {
                    grid = enemy[j].move(grid);
                }
            }
            if (enemy[0].getHit()) {
                grid[player.getX()][player.getY()] = 0;
                grid = player.p_spawn(grid);
                enemy[0].setHitPlayer(false);
                player.minusLife();
            }
            enemyclock.restart();
        }

        //bomb active
        if (bomb_count == 1)
        {
            if (bombtime.asMicroseconds() < 1000000)
            {
                bombText.setString("Bomb: 3");
            } else if (bombtime.asMicroseconds() < 2000000)
            {
                bombText.setString("Bomb: 2");
            } else if (bombtime.asMicroseconds() < 3000000)
            {
                bombText.setString("Bomb: 1");
            }
            //explode
            if (bombtime.asMicroseconds() > 3000000)
            {
                bomb_count = 0;
                grid = bomb.explode(grid);
                for (int i = 0; i < 10; i++) {
                    if (enemy[i].getActive())
                    {
                        enemy[i].setActive(bomb.checkHit(enemy[i].getX_pos
                      (), enemy[i].getY_pos()));
                    }
                }
                if (bomb.getHitPlayer())
                {
                    grid[player.getX()][player.getY()] = 5;
                    grid = player.p_spawn(grid);
                    bomb.setHitPlayer(false);
                    player.minusLife();
                }
                bomb.resetHitArray();
```

```cpp
                bombclock.restart();

                explosion = 1;
            }
        } //not active
        else {
            bombText.setString("");
        }

        //explosion timer
        if (explosion == 1)
        {
            if (explosiontime.asMicroseconds() > 1000000)
            {
                explosion = 0;
                for (int i = 0; i < 20; i++)
                {
                    for (int j = 0; j < 20; j++)
                    {
                        if (grid[i][j] == 5)
                        {
                            grid[i][j] = 0;
                        }
                    }
                }
            }
        }

        playerText.setString("Lives: " + to_string(player.getLives()) + " Enemies Left: " + to_string(10 - bomb.getScore()));
        timeText.setString("Game Time: " + to_string(gametime.asSeconds()));
        if (!map.generate("tileset.png", sf::Vector2u(48, 48), grid, 20, 20))
            return -1;

        if (bomb.getScore() == 10 or player.getLives() == 0)  //Player won
        {
            window.close();
        }
        else { // still playing
            window.clear();
            window.draw(map);
            window.draw(bombText);
            window.draw(playerText);
            window.draw(timeText);
            window.display();
        }


    }

    Time winningTime = gameclock.getElapsedTime();
    while (end_window.isOpen()) {
        end_window.setVisible(true);
        Text finalText;
        finalText.setFont(roboto);
        finalText.setCharacterSize(36);
        finalText.setFillColor(Color::White);
        if (bomb.getScore() == 10)
        {
            finalText.setPosition(50, 120);
            finalText.setString("You won in a time of: " + to_string
              (winningTime.asSeconds()));
        }
        else if (player.getLives() == 0)
        {
            finalText.setPosition(200, 140);
            finalText.setString("Game Over!");
        }
        else {
            end_window.close();
        }

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

        end_window.draw(finalText);
        end_window.display();
    }

    return 0;
}
```