

**Joint DSA & Programming Assignment**

**Title of Assignment:**

**Maze Game**

Student Name: Osama Abou Hajar

Student Number: C00220135

BSc. Hons Software Development

Institute of Technology Carlow, Ireland

Tutors:

Aine Byrne & Noel O Hara

Date of Submission: Monday 26th November @5pm.

Contents

[Contents 2](#_Toc530915355)

[Description of the problem. 3](#_Toc530915356)

[Description of Data Structures Used. 4](#_Toc530915357)

[Pseudocode of the Algorithms used. 4](#_Toc530915358)

[Functions Pseudocode 4](#_Toc530915359)

[Classes Pseudocode 5](#_Toc530915361)

[The Main Code Body Pseudocode 6](#_Toc530915362)

[Diagrams 7](#_Toc530915363)

[C++ code. 10](#_Toc530915364)

[Description of all the functions/routines which have been used. 28](#_Toc530915365)

[The output Screen: 29](#_Toc530915366)

[The Game screen 29](#_Toc530915367)

[Win screen & Game Over Screen 30](#_Toc530915368)

# Description of the problem.

Store the 2d maze as a file. The file must store the size of the maze in rows and columns. It must store whether each array element is either a corridor or a wall. The file must be able to store 1 maze of at least 10 by 20, and store the number of enemies per maze.

The maze is stored as an array of ints. The file must be store 1 maze. Look at Dynamic 2D array in Labs 2018 in blackboard.

Load in the file and draw the maze.

**The player** is represented by an instance of a Player class (you create) that either inherits from sf::Sprite or has a sf::Sprite object as a member variable.

The player character is drawn as a sprite that moves left, right, up and down along the corridors. The player cannot walk through walls. The player moves from one grid position into the adjacent one.

The player moves from one grid position to the next and must not move too fast.

**Each enemy** is represented by an instance of an Enemy class (you create) that either inherits from sf::Sprite or has a sf::Sprite object as a member variable.

The enemies are stored as an array of Enemy objects.

At the start of each game the enemies are randomly placed in the corridors (anything that’s not a wall). The enemies cannot be placed on a grid position that already contains a player or another enemy.

The enemy randomly chooses a direction to move. If there isn't a wall in its way it will move into to the next grid position, the enemies must not move too fast.

When an enemy dies it is brought back to life and is repositioned in a random corridor, not on top of the player or another enemy.

**The bomb** (only 1 bomb) is represented by an instance of a bomb class (you create).

The player can drop a bomb; the bomb has a fuse of 3 seconds. During this time the player cannot change the bombs position. When the bomb explodes it, it explodes in 8 directions, the diagonals and left, right, up and down. This explosion is instantaneous, it destroys any enemy or player within a distance of 4 grid tiles in those directions. The bomb however cannot go through the walls i.e. a player or enemy with a wall between it and the bomb will be unaffected. The player’s score is increased for each enemy destroyed.

**A player wins when a set number of enemies (e.g. 10) have been destroyed. The game is lost when the players lose all their lives.**

# Description of Data Structures Used.

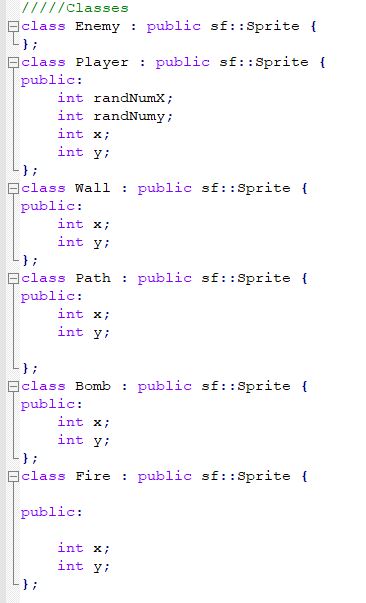
* 2D Array to draw the maze
  + Hold integer values: 1s and 0s to represent the wall and the path
* Array of Enemy object to create Enemy class instance.
  + Hold objects values integer and string
* Array of Fire object to create Enemy class instance.
  + Hold objects values integer and string

# Pseudocode of the Algorithms used.

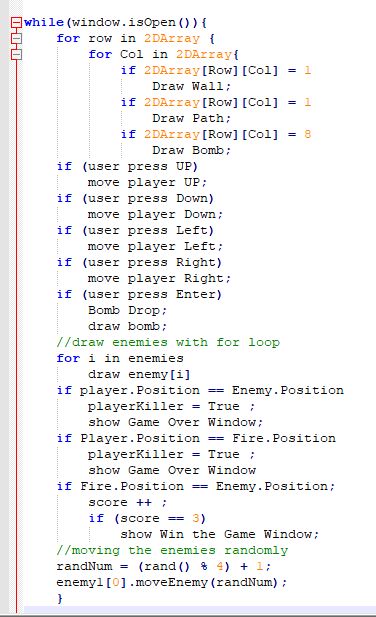
## Functions Pseudocode

## 

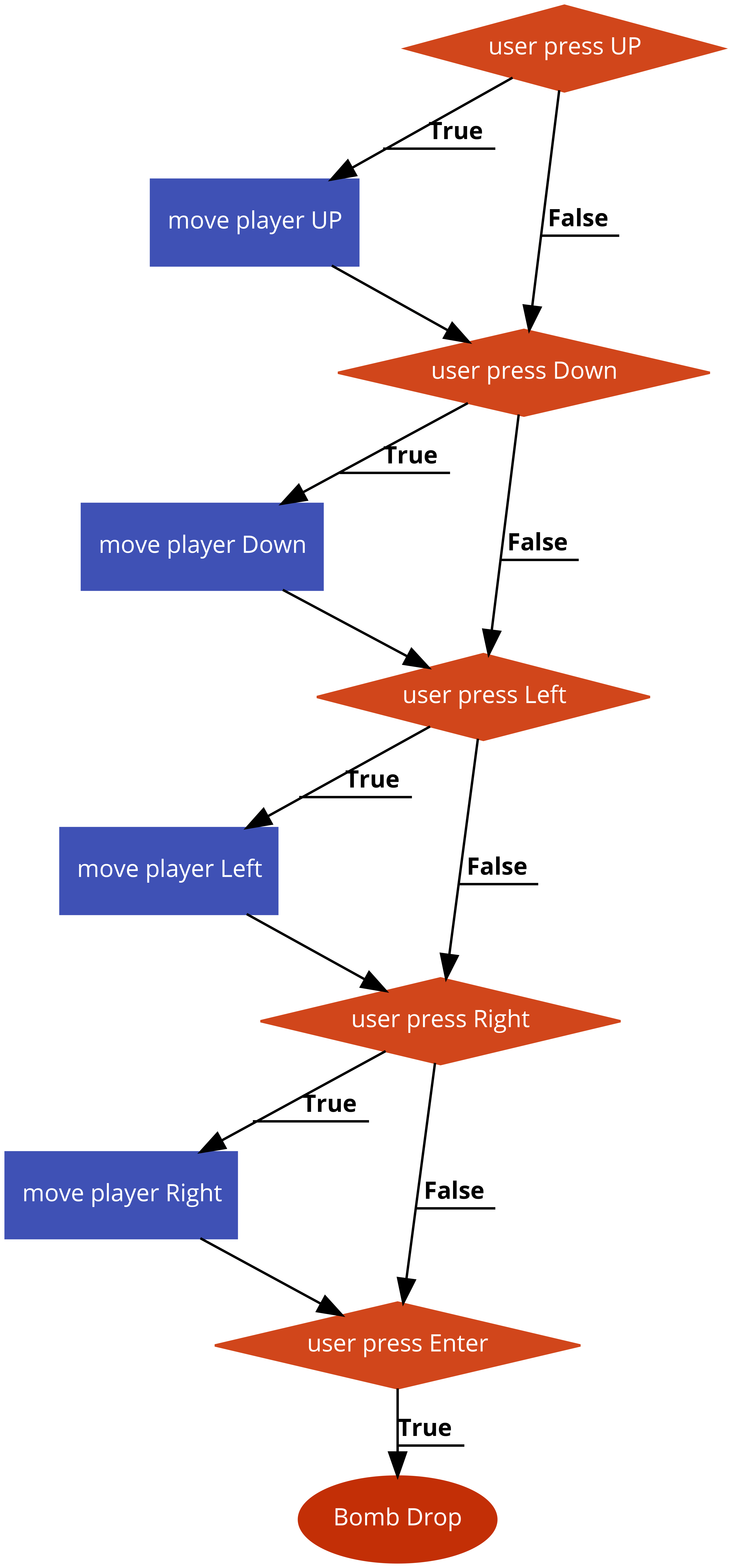
## Classes Pseudocode

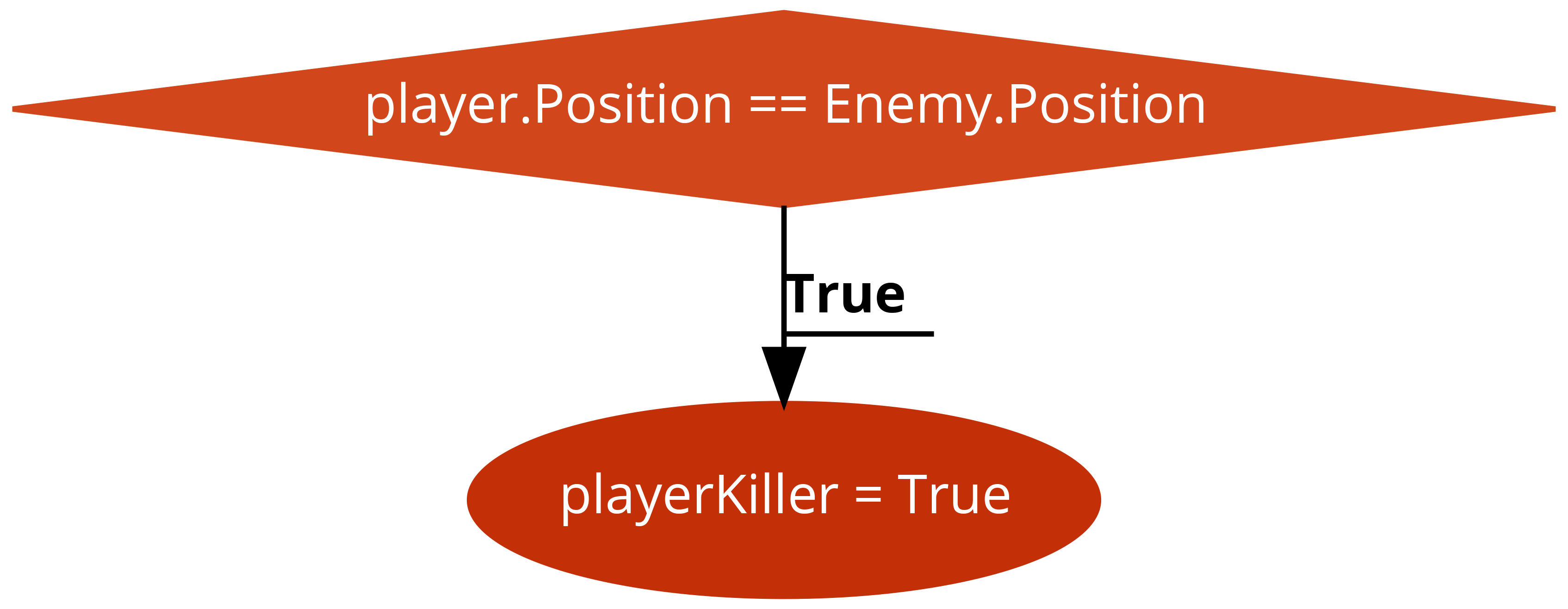


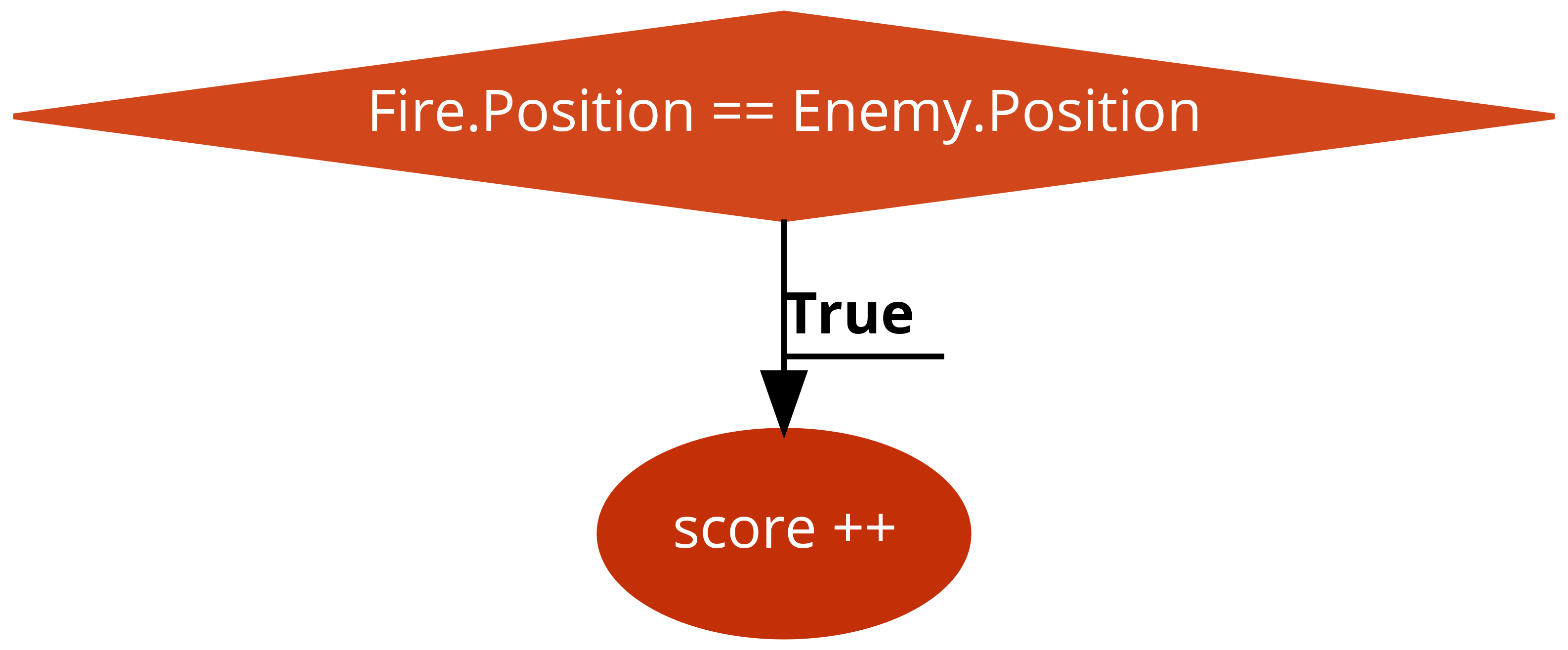
# The Main Code Body Pseudocode

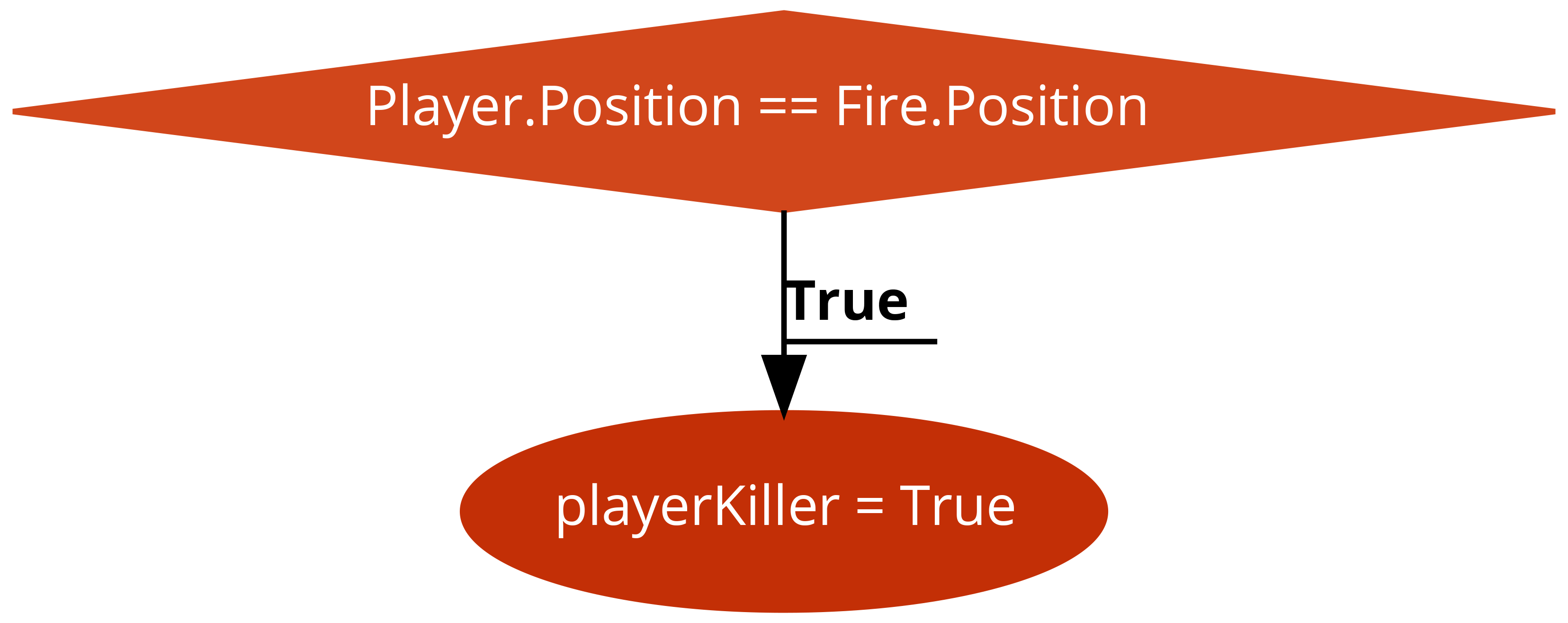


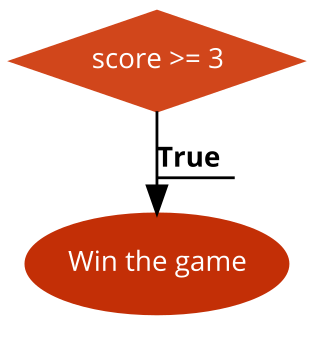
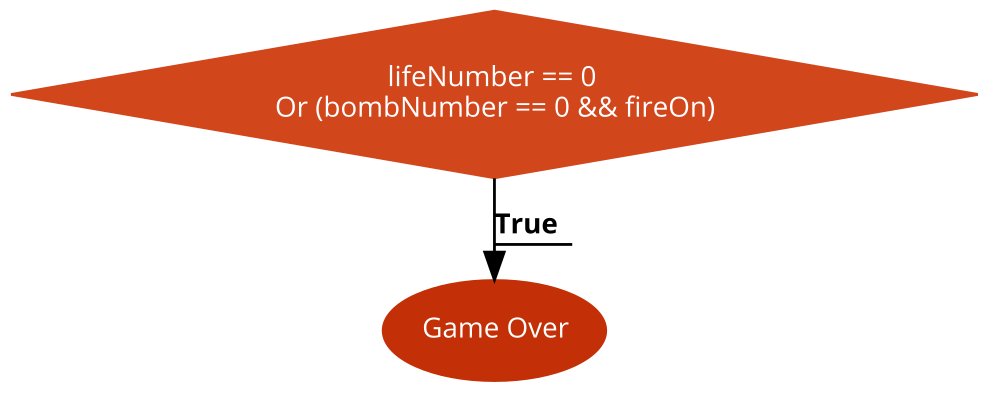
## Diagrams

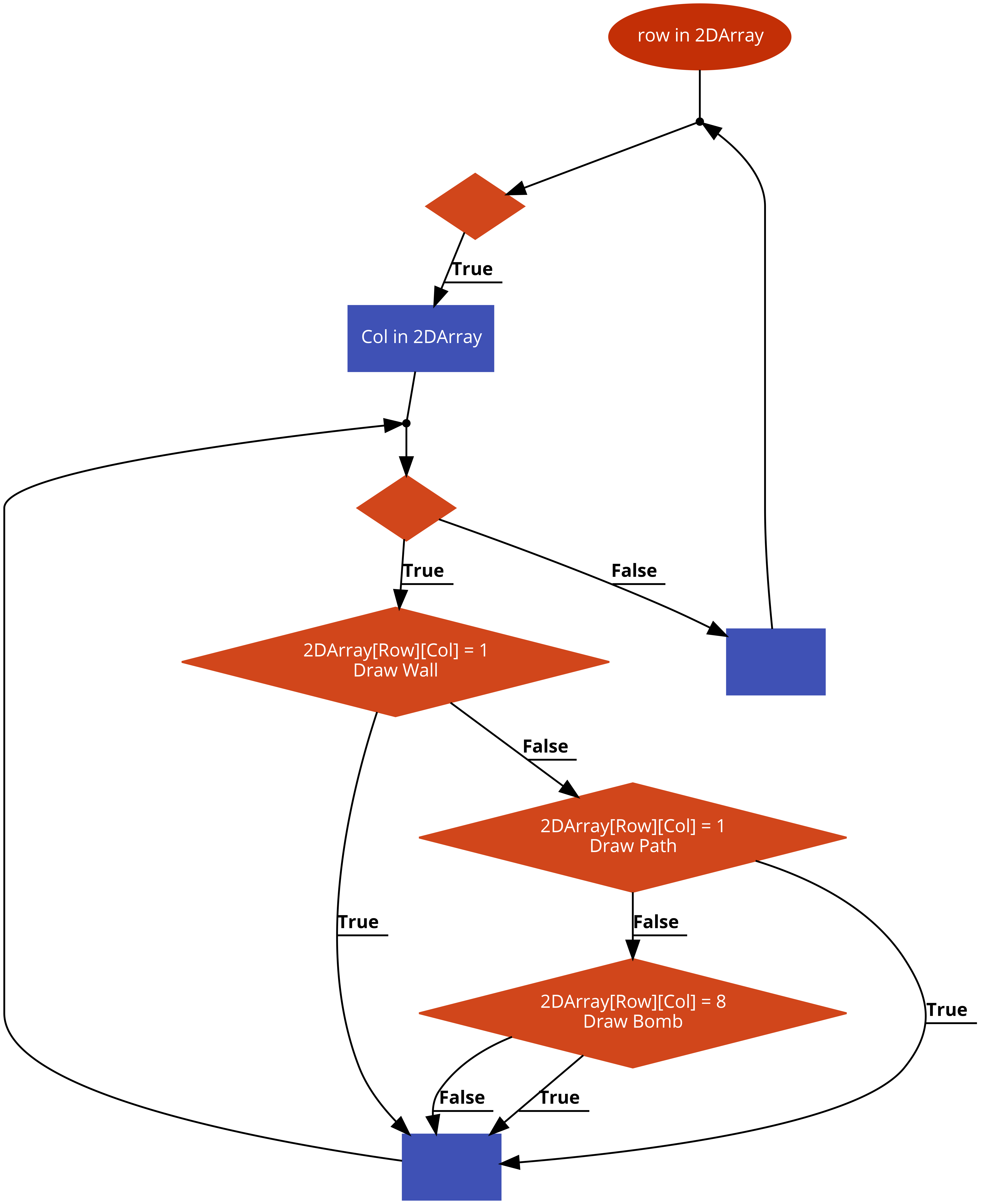
* Player moves:
* The life and the score logic:







* Game Over and Win Logic:
* Wall and Corridor drawing:



# C++ code.

// Maze game. By Osama Abou Hajar Date : 26/11/2018

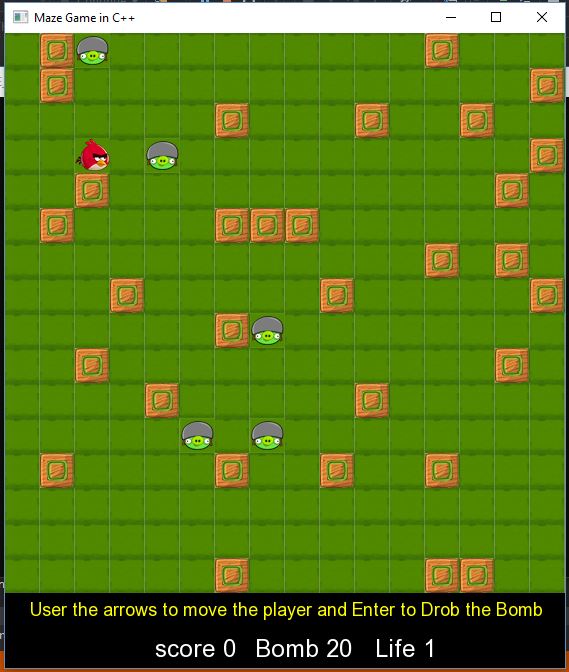
#include "pch.h"  
  
//#include "stdafx.h"  
  
#ifdef \_DEBUG   
#pragma comment(lib,"sfml-graphics-d.lib")   
#pragma comment(lib,"sfml-audio-d.lib")   
#pragma comment(lib,"sfml-system-d.lib")   
#pragma comment(lib,"sfml-window-d.lib")   
#pragma comment(lib,"sfml-network-d.lib")   
#else   
#pragma comment(lib,"sfml-graphics.lib")   
#pragma comment(lib,"sfml-audio.lib")   
#pragma comment(lib,"sfml-system.lib")   
#pragma comment(lib,"sfml-window.lib")   
#pragma comment(lib,"sfml-network.lib")   
#endif   
  
#include <SFML/Graphics.hpp>  
#include <iostream>  
#include <stdlib.h>   
#include <time.h>   
//#include <Clock.hpp>  
#include <iostream>  
#include <fstream>  
#include <SFML/Audio.hpp>  
  
//using namespace std;  
  
  
int const ARRAYSIZE = 16;  
  
int dynamicArrraySizeRows;  
int dynamicEnemyNumber;  
int dynamicArrraySizeColumns;  
  
int \*\*mazeGraph;  
int const DIMINTION = 35;  
int SIDES = 525;  
int randNum = 0;  
//int mazeGraph[ARRAYSIZE][ARRAYSIZE] = {};  
sf::Music music;  
sf::Sound bombsound;  
sf::Sound firesound;  
  
/////Classes  
  
class Enemy : public sf::Sprite {  
public:  
int randNumX;  
int randNumy;   
int x;  
int y;  
  
Enemy()  
{   
  
do  
{  
randNumX = (rand() % dynamicArrraySizeColumns-1) + 1;  
randNumy = (rand() % dynamicArrraySizeRows-1) + 1;  
x = randNumX;  
y = randNumy;  
} while (mazeGraph[randNumy][randNumX] == 1 );  
  
  
}  
bool moveEnemy(int direction) {  
this->x = (this->getPosition().x) / DIMINTION;  
this->y = (this->getPosition().y) / DIMINTION;  
  
std::cout << "ENEMY x" << x << std::endl;  
std::cout << "ENEMY y" << y << std::endl;  
  
if (direction == 1)  
{  
if (this->getPosition().x != 0 && mazeGraph[y][x - 1] != 1 && mazeGraph[y][x - 1] != 8)  
{  
this->move(-DIMINTION, 0);  
}  
else  
return false;  
}  
else if (direction == 2)  
{  
if (this->getPosition().x != SIDES && mazeGraph[y][x + 1] != 1 && mazeGraph[y][x + 1] != 8)  
{  
this->move(DIMINTION, 0);  
}  
else  
return false;  
}  
else if (direction == 3)  
{  
if (this->getPosition().y != 0 && mazeGraph[y - 1][x] != 1 && mazeGraph[y - 1][x] != 8)  
{  
this->move(0, -DIMINTION);  
}  
else  
return false;  
}  
else if (direction == 4)  
{  
if (this->getPosition().y != SIDES && mazeGraph[y + 1][x] != 1 && mazeGraph[y + 1][x] != 8)  
{  
this->move(0, DIMINTION);  
}  
else  
return false;  
}  
}  
  
  
};  
  
class Player : public sf::Sprite {  
public:  
int randNumX;  
int randNumy;   
int x;  
int y;  
Player()  
{  
do  
{  
randNumX = (rand() % dynamicArrraySizeColumns - 1) + 1;  
randNumy = (rand() % dynamicArrraySizeRows - 1) + 1;  
x = randNumX;  
y = randNumy;  
} while (mazeGraph[randNumy][randNumX] == 1);  
}  
  
bool movePlayer(int direction) {  
this -> x = (this -> getPosition().x) / DIMINTION;  
this-> y = (this -> getPosition().y) / DIMINTION;  
  
std::cout << "x" << x << std::endl;  
std::cout << "y" << y << std::endl;  
  
if (direction == 1)  
{  
if (this -> getPosition().x != 0 && mazeGraph[y][x - 1] != 1 && mazeGraph[y][x - 1] != 8)  
{//left  
this -> move(-DIMINTION, 0);  
}  
else  
return false;  
}  
else if (direction == 2)  
{  
if (this->getPosition().x != SIDES && mazeGraph[y][x + 1] != 1 && mazeGraph[y][x + 1] != 8)  
{//right  
this->move(DIMINTION, 0);  
}  
else  
return false;  
}  
else if (direction == 3)  
{  
if (this->getPosition().y != 0 && mazeGraph[y - 1][x] != 1 && mazeGraph[y - 1][x] != 8)  
{//up  
this->move(0, -DIMINTION);  
}  
else  
return false;  
}  
else if (direction == 4)  
{//down  
if (this->getPosition().y != SIDES && mazeGraph[y + 1][x] != 1 && mazeGraph[y + 1][x] != 8 )  
{  
this->move(0, DIMINTION);  
}  
else  
return false;  
}  
}  
void setRanPos()  
{  
do  
{  
randNumX = (rand() % dynamicArrraySizeColumns - 1) + 1;  
randNumy = (rand() % dynamicArrraySizeRows - 1) + 1;  
x = randNumX;  
y = randNumy;  
} while (mazeGraph[randNumy][randNumX] == 1);  
}  
};  
  
class Wall : public sf::Sprite {  
public:  
int x;  
int y;  
};  
  
  
class Path : public sf::Sprite {  
public:  
int x;  
int y;  
  
};  
  
class Bomb : public sf::Sprite {  
public:  
int x;  
int y;  
Bomb()  
{  
x = 0;  
y = 0;  
}  
  
};  
  
class Fire : public sf::Sprite {  
  
public:  
  
int x;  
int y;  
Fire()  
{  
x = -20;  
y = -20;  
}  
void fireOn(int bombY, int bombX) {  
  
  
  
}  
};  
  
/////functions  
sf::Time frame() {  
sf::Time timePerFrame = sf::seconds(10.0f / 60.0f);  
return timePerFrame;  
}  
  
void create\_the\_sound() {  
  
if (!music.openFromFile("gameStart.wav"))  
{  
std::cout << "Sound Not Found";  
}  
// increase the pitch  
music.setVolume(35);  
music.play();  
  
////sound when drop the bomb  
sf::SoundBuffer bombbuffer;  
if (!bombbuffer.loadFromFile("bombsound.wav")) {  
std::cout << "Bomb Sound Not Found";  
}  
  
bombsound.setBuffer(bombbuffer);  
  
  
////sound when drop the bomb  
sf::SoundBuffer firebuffer;  
if (!firebuffer.loadFromFile("Explosion.wav")) {  
std::cout << "Bomb Sound Not Found";  
}  
  
firesound.setBuffer(firebuffer);  
}  
  
bool player\_touch\_the\_fire(Player player, Fire fireSprite[])  
{  
for (int index = 0; index < 36; index++)  
{  
if (player.getPosition() == fireSprite[index].getPosition()) {  
  
return  true;  
std::cout << std::endl << "\n\n player touch the fire" << std::endl;  
}   
}  
return false;  
}  
  
void setFirePos(Fire fireSprite[] ,int bombY,int bombX)  
{  
//// to assign the fire on the four different positions around the bomb  
std::cout << "mazeGraph[bombY][bombX]" << mazeGraph[bombY][bombX];  
  
int indexFortheArrayOfFire = 0;  
for (int totalFireToPrint = 0; totalFireToPrint < 36; totalFireToPrint ++ ) {  
fireSprite[totalFireToPrint].x = -20;  
fireSprite[totalFireToPrint].y = -20;  
}  
//left  
int left = 1;  
while (mazeGraph[bombY][bombX - left ] != 1 && left <= 4)  
{  
fireSprite[indexFortheArrayOfFire].x = bombY;  
fireSprite[indexFortheArrayOfFire].y = bombX - left;  
left++;  
indexFortheArrayOfFire++;  
}  
  
int right = 1;  
while (mazeGraph[bombY][bombX + right] != 1 && right <= 4)  
{  
fireSprite[indexFortheArrayOfFire].x = bombY;  
fireSprite[indexFortheArrayOfFire].y = bombX + right;  
right++;  
indexFortheArrayOfFire++;  
}  
  
int up = 1;  
while (bombY - up > -1 &&mazeGraph[bombY - up][bombX] != 1 && up <= 4)  
{  
fireSprite[indexFortheArrayOfFire].x = bombY - up;  
fireSprite[indexFortheArrayOfFire].y = bombX;  
up++;  
indexFortheArrayOfFire++;  
}  
int down = 1;  
// if to make suer dont go out of the sides  
while (bombY + down < dynamicArrraySizeRows && mazeGraph[bombY + down][bombX] != 1 && down <= 4)  
{  
fireSprite[indexFortheArrayOfFire].x = bombY + down;  
fireSprite[indexFortheArrayOfFire].y = bombX;  
down++;  
indexFortheArrayOfFire++;  
}  
  
int upleft = 1;  
while (bombY - upleft > -1 && mazeGraph[bombY - upleft][bombX + upleft] != 1 && upleft <= 4)  
{  
fireSprite[indexFortheArrayOfFire].x = bombY - upleft;  
fireSprite[indexFortheArrayOfFire].y = bombX + upleft;  
upleft++;  
indexFortheArrayOfFire++;  
}  
  
int upright = 1;  
while (bombY - upright > -1 && mazeGraph[bombY - upright][bombX - upright] != 1 && upright <= 4)  
{  
fireSprite[indexFortheArrayOfFire].x = bombY - upright;  
fireSprite[indexFortheArrayOfFire].y = bombX - upright;  
upright++;  
indexFortheArrayOfFire++;  
}  
  
int downleft = 1;  
// if to make suer dont go out of the sides  
while (bombY + downleft < dynamicArrraySizeRows && mazeGraph[bombY + downleft][bombX - downleft] != 1 && downleft <= 4)  
{  
fireSprite[indexFortheArrayOfFire].x = bombY + downleft;  
fireSprite[indexFortheArrayOfFire].y = bombX - downleft;  
downleft++;  
indexFortheArrayOfFire++;  
}  
int downright = 1;  
// if to make suer dont go out of the sides  
while (bombY + downright < dynamicArrraySizeRows && mazeGraph[bombY + downright][bombX + downright] != 1 && downright <= 4)  
{  
fireSprite[indexFortheArrayOfFire].x = bombY + downright;  
fireSprite[indexFortheArrayOfFire].y = bombX + downright;  
downright++;  
indexFortheArrayOfFire++;  
}  
  
  
}  
  
void read\_the\_Array\_from\_the\_file()  
{  
////// to read the array from the file ///////////  
std::ifstream file{ "array.txt" };  
  
if (!file.is\_open())  
std::cout << "Array File not found";  
  
file >> dynamicArrraySizeColumns;  
file >> dynamicArrraySizeRows;  
file >> dynamicEnemyNumber;  
mazeGraph = new int\*[dynamicArrraySizeRows];  
  
for (int row = 0; row < dynamicArrraySizeRows; row++) {  
mazeGraph[row] = new int[dynamicArrraySizeColumns];  
}  
  
  
for (int mazeCol = 0; mazeCol < dynamicArrraySizeRows; mazeCol++) {  
for (int mazeRow = 0; mazeRow < dynamicArrraySizeColumns; mazeRow++) {  
file >> mazeGraph[mazeCol][mazeRow];  
}  
}  
  
  
}  
  
int main() ///////////////////////////////////////////////////// MAIN ///////////////////////////////////  
{  
start:  
int score = 0;  
int lifeNumber = 3;  
int bombNumber = 20;  
bool playerKilled = false;  
bool enemyKilled = false;  
bool fireOn = false;  
create\_the\_sound();  
read\_the\_Array\_from\_the\_file();  
int counter = 0;  
//create Window  
sf::RenderWindow window(sf::VideoMode(560, 635), "Maze Game in C++");  
srand(time(NULL));  
float randomNum;  
sf::Time timePerFrame = sf::seconds(1.0f / 60.0f); //Don't change this code  
sf::Time timeSinceLastUpdate = sf::Time::Zero;  
// the clock object keeps the time.  
sf::Clock clock;  
clock.restart();  
  
double speed = 0.25;  
  
/////////////////////////////elements //////////////////////////////  
////Player /////  
sf::Texture playerTexture;  
if (!playerTexture.loadFromFile("player.png"))  
{  
std::cout << "file not found";  
}  
Player player;  
player.setTexture(playerTexture);  
player.setPosition(player.x \* player.getGlobalBounds().width, player.y \* player.getGlobalBounds().height);  
  
////Enemy//////  
sf::Texture enemyTexture;  
if (!enemyTexture.loadFromFile("enemy.png"))  
{  
std::cout << "file not found";  
}  
Enemy enemy1[100];  
int enemyCounter = 0;  
  
/////wall////  
sf::Texture wallTexture;  
if (!wallTexture.loadFromFile("wall.png"))  
{  
std::cout << "file not found";  
}  
Wall wallSprite;  
wallSprite.setTexture(wallTexture);  
///// Path //////  
sf::Texture pathTexture;  
if (!pathTexture.loadFromFile("Path.png"))  
{  
std::cout << "file not found";  
}  
Path pathSprite;  
pathSprite.setTexture(pathTexture);  
////// Bomb //////////////  
sf::Texture bombTexture;  
if (!bombTexture.loadFromFile("Bomb.png"))  
{  
std::cout << "file not found";  
}  
Bomb bombSprite ;  
bombSprite.setTexture(bombTexture);  
sf::Clock clockBomb;  
sf::Time timeBomb;  
int bombX = 0;  
int bombY = 0;  
bool bombDropped = false;  
  
////// Fire //////////////  
sf::Texture fireTexture;  
if (!fireTexture.loadFromFile("fire.png"))  
{  
std::cout << "file not found";  
}  
Fire fireSpriteDraw;  
fireSpriteDraw.setTexture(fireTexture);  
Fire fireSprite[36];  
  
//to get the font  
sf::Font font;  
if (!font.loadFromFile("arial.ttf"))  
{  
std::cout << "file not found";  
}  
  
sf::Text instractionsText;  
instractionsText.setFont(font);  
instractionsText.setString("User the arrows to move the player and Enter to Drob the Bomb ");  
instractionsText.setCharacterSize(18);  
instractionsText.setFillColor(sf::Color::Yellow);  
instractionsText.setPosition(25, 565);  
  
for (int indexForenemey = 0; indexForenemey < dynamicEnemyNumber; indexForenemey++)   
{  
enemy1[indexForenemey].setTexture(enemyTexture);  
enemy1[indexForenemey].setPosition(enemy1[indexForenemey].x \* enemy1[indexForenemey].getGlobalBounds().width, enemy1[indexForenemey].y \* enemy1[indexForenemey].getGlobalBounds().height);  
}  
  
int counterForTime = 0;  
///// while window is open /////  
while (window.isOpen())  
{  
//// Create a random  number each time run between 0 -> 4 ///////////  
randNum = (rand() % 4) + 1;  
/////the life OUTPUT ///////////  
sf::Text lifeText;  
lifeText.setFont(font);  
lifeText.setString("Life " + std::to\_string(lifeNumber));  
lifeText.setCharacterSize(24);  
lifeText.setFillColor(sf::Color::White);  
lifeText.setPosition(370, 600);  
/////the bomb OUTPUT ///////////   
sf::Text bombText;  
bombText.setFont(font);  
bombText.setString("Bomb " + std::to\_string(bombNumber));  
bombText.setCharacterSize(24);  
bombText.setFillColor(sf::Color::White);  
bombText.setPosition(250, 600);  
/////the score OUTPUT ///////////   
sf::Text scoreText;  
scoreText.setFont(font);  
scoreText.setString("score " + std::to\_string(score));  
scoreText.setCharacterSize(24);  
scoreText.setFillColor(sf::Color::White);  
scoreText.setPosition(150, 600);  
// check if the close window button is clicked on.  
sf::Event event;  
while (window.pollEvent(event))  
{  
if (event.type == sf::Event::Closed)  
window.close();  
}  
//add to the time since last update and restart the clock  
timeSinceLastUpdate += clock.restart();  
////To clear each iteration   
window.clear();// dont change this code..  
////// to print the wall and path from the array  
for (int i = 0; i < dynamicArrraySizeRows; i++)  
{  
for (int j = 0; j < dynamicArrraySizeColumns; j++)  
{  
if (mazeGraph[i][j] == 0)  
{  
pathSprite.setPosition(j \* pathSprite.getGlobalBounds().width, i \* pathSprite.getGlobalBounds().height);  
window.draw(pathSprite);  
}  
if (mazeGraph[i][j] == 1)  
{  
wallSprite.setPosition(j \* wallSprite.getGlobalBounds().width, i \* wallSprite.getGlobalBounds().height);  
window.draw(wallSprite);  
}  
if (mazeGraph[i][j] == 8) //to draw the bomb after drop with Enter key  
{  
bombSprite.setPosition(j \* wallSprite.getGlobalBounds().width, i \* wallSprite.getGlobalBounds().height);  
window.draw(bombSprite);  
}  
}  
}  
//// to output the enemies /////  
for (int indexToDrawTheEnemy = 0; indexToDrawTheEnemy < dynamicEnemyNumber; indexToDrawTheEnemy++) {  
window.draw(enemy1[indexToDrawTheEnemy]);  
}  
window.draw(player);  
////////////////////////////// PLAYER MOVMENTE /////////////////////////  
int dirctionMove = 0;  // to decid where to move as the key pressed  
int xPlayer = (player.getPosition().x) / DIMINTION;  
int yPlayer = (player.getPosition().y) / DIMINTION;  
if (timeSinceLastUpdate > frame())  
{  
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))  
{  
dirctionMove = 1;  
player.movePlayer(dirctionMove);  
}  
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))  
{  
dirctionMove = 2;  
player.movePlayer(dirctionMove);  
}  
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))  
{  
dirctionMove = 3;  
player.movePlayer(dirctionMove);  
}  
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))  
{  
dirctionMove = 4;  
player.movePlayer(dirctionMove);  
}  
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Enter) && bombDropped == false && bombNumber > 0)  
{  
bombsound.play();  
mazeGraph[yPlayer][xPlayer] = 8;  
bombY = yPlayer;  
bombX = xPlayer;  
//// assign within the class  
bombSprite.y = yPlayer;  
bombSprite.x = xPlayer;  
// to decrement the bomb number on the screen  
bombNumber--;  
std::cout << "the Bomb Droped" << bombNumber;  
//// to reset the bomb timer   
timeBomb = clockBomb.restart();  
bombDropped = true;  
}  
//// For My use to check the timer  
timeBomb = clockBomb.getElapsedTime();  
std::cout << "the Bomb Droped  " << int(timeBomb.asSeconds()) << std::endl;  
//// to check if the time since the bomb on is 5 seconds  
if (int(timeBomb.asSeconds()) == 3 && bombDropped)  
{  
firesound.play();  
setFirePos(fireSprite, bombY, bombX);  
int randNumX = (rand() % dynamicArrraySizeColumns - 1) + 1;  
int randNumy = (rand() % dynamicArrraySizeRows - 1) + 1;  
///// to draw the bomb and check at the same time if the enemy on the fire  
for (int ixd = 0; ixd < 36; ixd++)  
{  
fireSprite[ixd].setTexture(fireTexture);  
fireSprite[ixd].setPosition(fireSprite[ixd].y \* fireSprite[ixd].getGlobalBounds().width, fireSprite[ixd].x \* fireSprite[ixd].getGlobalBounds().height);  
window.draw(fireSprite[ixd]);  
  
for (int innerindex = 0; innerindex <= dynamicEnemyNumber; innerindex++) {  
if (fireSprite[ixd].getPosition() == enemy1[innerindex].getPosition())  
{  
enemyKilled = true;  
score++;  
do {  
randNumX = (rand() % dynamicArrraySizeColumns - 1) + 1;  
randNumy = (rand() % dynamicArrraySizeRows - 1) + 1;  
enemy1[innerindex].x = randNumX;  
enemy1[innerindex].y = randNumy;  
enemy1[innerindex].setPosition(randNumX\*DIMINTION, randNumy\*DIMINTION);  
} while (mazeGraph[randNumy][randNumX] == 1);  
}  
}  
  
if (player\_touch\_the\_fire(player, fireSprite))  
{  
randNumX = (rand() % dynamicArrraySizeColumns - 1) + 1;  
randNumy = (rand() % dynamicArrraySizeRows - 1) + 1;  
player.x = randNumX;  
player.y = randNumy;  
player.setPosition(randNumX\*DIMINTION, randNumy\*DIMINTION);  
playerKilled = true;  
lifeNumber--;  
}  
}  
///// to remove the bomb after the fire and check   
mazeGraph[bombY][bombX] = 0;  
bombDropped = false;  
std::cout << "mazeGraph[bombX][bombY]  " << mazeGraph[bombY][bombX] << std::endl;  
fireOn = true;  
}  
  
//if player touch the enemy  
for (int index = 0; index < dynamicEnemyNumber; index++)  
{  
if (player.getPosition() == enemy1[index].getPosition()) {  
lifeNumber--;  
playerKilled = true;  
std::cout << std::endl << "\n\n GAME OVER" << std::endl;  
}  
  
}  
  
// clear the screen and draw all the shapes  
window.draw(instractionsText);  
window.draw(scoreText);  
window.draw(bombText);  
window.draw(lifeText);  
window.display();// dont change this code..  
  
//enemy1[0].moveEnemy(randNum);  
  
for (int indexMovingEnemies = 1; indexMovingEnemies < dynamicEnemyNumber; indexMovingEnemies++) {  
randNum = (rand() % 4) + 1;  
enemy1[indexMovingEnemies].moveEnemy(randNum);  
}  
// reset the timeSinceLastUpdate to 0  // dont change this code..  
timeSinceLastUpdate = sf::Time::Zero;  
  
//// to display the win window  
if (score >= 3)  
{  
sf::RenderWindow windowWin(sf::VideoMode(900, 575), "You Win");  
while (windowWin.isOpen())  
{  
sf::Event event;  
while (windowWin.pollEvent(event))  
{  
if (event.type == sf::Event::Closed)  
{  
windowWin.close();  
window.close();  
}  
}  
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))  
{  
windowWin.close();  
window.close();  
}  
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Enter))  
{  
goto start;  
}  
  
sf::Texture gameOverTexture;  
if (!gameOverTexture.loadFromFile("win.png"))  
{  
std::cout << std::endl << "\n\n Can't load GAME OVER";  
}  
  
sf::Sprite goSprite;  
goSprite.setTexture(gameOverTexture);  
  
windowWin.draw(goSprite);  
windowWin.display();  
  
}  
}  
if (lifeNumber == 0 || (bombNumber == 0 && fireOn)) {  
sf::RenderWindow windowGameOver(sf::VideoMode(900, 506), "Game Over");  
while (windowGameOver.isOpen()) {  
sf::Event event;  
while (windowGameOver.pollEvent(event))  
{  
if (event.type == sf::Event::Closed)  
{  
windowGameOver.close();  
window.close();  
}  
}  
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))  
{  
windowGameOver.close();  
window.close();  
}  
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Enter))  
{  
goto start;  
}  
  
sf::Texture gameOverTexture;  
if (!gameOverTexture.loadFromFile("gameover.png"))  
{  
std::cout << std::endl << "\n\n Can't load GAME OVER";  
}  
  
sf::Sprite goSprite;  
goSprite.setTexture(gameOverTexture);  
  
windowGameOver.draw(goSprite);  
windowGameOver.display();  
}  
}  
  
}   
}  
return 0;  
}

# Description of all the functions/routines which have been used.

/////functions  
  
//to import the sound and assignee the buffer  
void create\_the\_sound() {   
  
  
}  
//to check if the player touch the fire and return True or False  
bool player\_touch\_the\_fire(Player player, Fire fireSprite[])   
{  
  
}  
// to set the fire positions within their classes and draw them  
void setFirePos(Fire fireSprite[] ,int bombY,int bombX)   
{  
  
}  
//to read the Array file and assign the Row, Columns and the enemies number  
void read\_the\_Array\_from\_the\_file()    
{  
  
}

# The output Screen:

## The Game screen



## Win screen: Game Over Screen

