## Joint DSA & Programming Assignment

Assignment Report

Rhyder Quinlan C00223030

Contents

[Joint DSA & Programming Assignment 1](#_Toc531004509)

[Problem Description 3](#_Toc531004510)

[Data Structures 3](#_Toc531004511)

[2D Maze Array 3](#_Toc531004512)

[Enemies Array 3](#_Toc531004513)

[Enemies Hit in Explosion Structure 3](#_Toc531004514)

[Pseudocode 4](#_Toc531004515)

[The Player Class 4](#_Toc531004516)

[Move Player 4](#_Toc531004517)

[Spawn 4](#_Toc531004518)

[The Bomb Class 5](#_Toc531004519)

[Drop Bomb 5](#_Toc531004520)

[Explode Up Example 5](#_Toc531004522)

[C++ Code 6](#_Toc531004523)

# Problem Description

The aim of this project is to build an interactive “Maze Game” where a Player (User controlled) can traverse a “Maze” like screen, in this maze there are enemies (Non-Player characters) that move around with the aim of hitting the player to remove a life. A player can drop “Bombs” to kill the enemies, with the goal of killing 10 enemies in the shortest amount of time.

This project was built with C++, integrating the visual abilities of SFML (Simple and Fast Multimedia Library) to display the game elements.

# Data Structures

### 2D Maze Array

33333333333333333333

30303030300303030303

0 = Grass/Path

1 = Player character

2 = Enemy

3 = Tree/Barrier

4 = Bomb

5 = Explosion

6 = Bomb&Player Sprite

30000000002000000003

30333330000003333303

30300000000000000303

30300330033003300303

30300330033003300303

30000000033000000003

30001000033000000003

30303030333303030303

30000000000000020003

30303030333303030303

30000000033000000003

30300330033003300303

30300330033003300303

30300000000000000303

30333330002003333303

30002000000000000003

30303030300303030303

33333333333333333333

The “Maze” is represented by a 2D array data structure. Upon starting up the game the array is populated by the values in ‘grid.txt’ file which represent the barriers and paths of the maze.

The array is made up of integers from 0 to 6, each representing a different game element (See above legend).

### Enemies Array

A criteria of the assignment was to store the enemies in an enemy class, and each enemy object stored in an array of this class:

Enemy \*enemy;

enemy = new Enemy[10];

This code instantiates an array of 10 enemy objects. Because only 4 enemies are in play at a time in my game, I used a Boolean flag that marked if the enemy object within this array was currently active (in play) or not.

### Enemies Hit in Explosion Structure

Due to the fact many enemies can bit hit in an explosion there was a need to record which enemies were hit and then to deactivate them and respawn more. This was done using an array 2x4 array, recording the x and y co-ordinates of the hit enemy.

# Pseudocode

## The Player Class

### Move Player

int\*\* Player::move(int x, int y, int \*\*g) {

//mark the position to move to

x\_move = x\_coord + x;

y\_move = y\_coord + y;

if (g[x\_move][y\_move] == 0) //Check if it is a valid position to move to

{

if (g[x\_coord][y\_coord] == 4) //check if bomb was just placed

{

g[x\_move][y\_move] = 1;

}

else if (g[x\_coord][y\_coord] == 6)

//transition between Wizard/Bomb sprite to moving wizard

{

g[x\_move][y\_move] = 1;

g[x\_coord][y\_coord] = 4;

}

else {

g[x\_move][y\_move] = 1;

g[x\_coord][y\_coord] = 0;

}

//update new co ordinates of player

x\_coord = x\_coord + x;

y\_coord = y\_coord + y;

}

return g;

}

### Spawn

int\*\* Player::spawn(int \*\*g) {

bool valid = false;

while (!valid) {

generate random number

x\_coord = random number

y\_coord = random number

//check if the random co-ordinates are a valid placement.

if (g[x\_coord][y\_coord] == 0)

{

g[x\_coord][y\_coord] = 1;

valid = true;

}

//if valid then break loop.

}

return g;

}

## The Enemy Class

### Spawn Enemy

int\*\* Enemy::spawn(int \*\*g) {

active = true;

int placement;

bool valid = false;

while (!valid) {

x\_pos = random integer;

y\_pos = random integer;

if (g[x\_pos][y\_pos] == 0)

//if random co-ords are a valid placement then spawn

{

g[x\_pos][y\_pos] = 2;

valid = true;

//break loop

}

}

return g;

### }

### Move up example

int\*\* Enemy::move(int\*\* g) {

int direction = 0;

bool valid = false;

while (!valid) {

direction = generate random integer between 1 and 4 (including)

switch (direction)

{

case 1:

//move up

if (g[x\_pos - 1][y\_pos] == 0)

//x\_pos – 1 represents move up one row

{

g[x\_pos - 1][y\_pos] = 2;

g[x\_pos][y\_pos] = 0;

x\_pos = x\_pos - 1;

valid = true;

}

### break;

}

}

return g;

}

### Enemy movement

Within the main function the enemies will move every 700000 microseconds.

if (enemytime.asMicroseconds() > 700000)

{

for (int j = 0; j < 10; j++)

{

if (enemy[j].getActive())//only active enemies will move

{

grid = enemy[j].move(grid);

}

### }

### }

## The Bomb Class

### Drop Bomb

int\*\* Bomb::dropBomb(int\*\* grid, int x, int y) {

setBomb\_Coord(x, y); //update bomb co-ordinates.

grid[bomb\_x][bomb\_y] = 6;

return grid;

### }

### Explosion Up Example

int\*\* Bomb::explode up(int\*\* g) {

//constant as a pointer for the array of enemies hit in this explosion.

int a = 0;

g[bomb\_x][bomb\_y] = 5;

//explosion up

for (int j = 1; j < 4; j++)

{

//current bomb row co-ord – j value represents moving up with the explosions.

if (g[bomb\_x - j][bomb\_y] == 0) {

g[bomb\_x - j][bomb\_y] = 5;

}

//hit enemy in explosion

if (g[bomb\_x - j][bomb\_y] == 2) {

//update the array storing enemies hit in this explosion

enemyHit(bomb\_x - j, bomb\_y, a);

a++;

g[bomb\_x - j][bomb\_y] = 5;

}

//hit player in explosion

if (g[bomb\_x - j][bomb\_y] == 1)

{

playerhit = true;

g[bomb\_x - j][bomb\_y] = 5;

}

//if barrier reached then break loop

if(g[bomb\_x - j][bomb\_y] == 3){

j = 4;

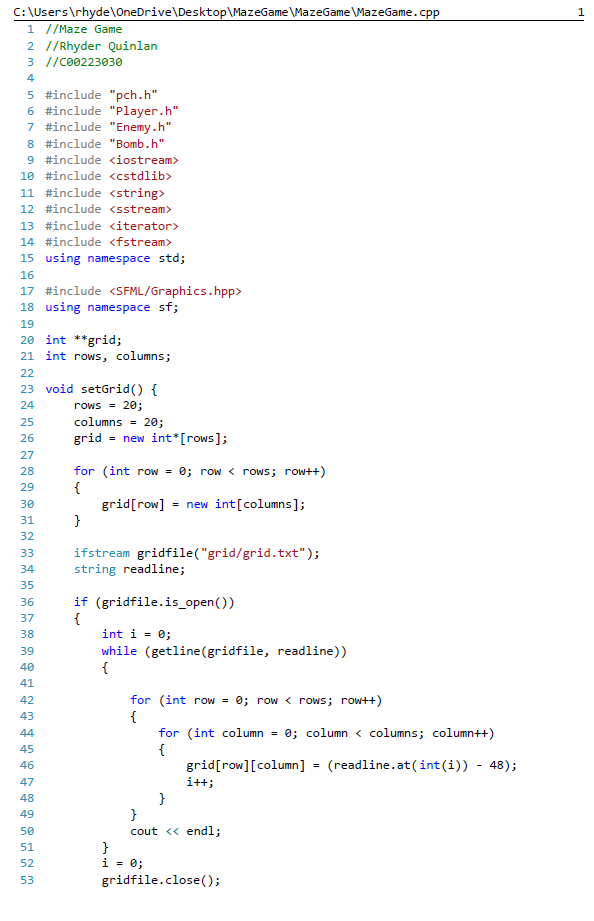
}

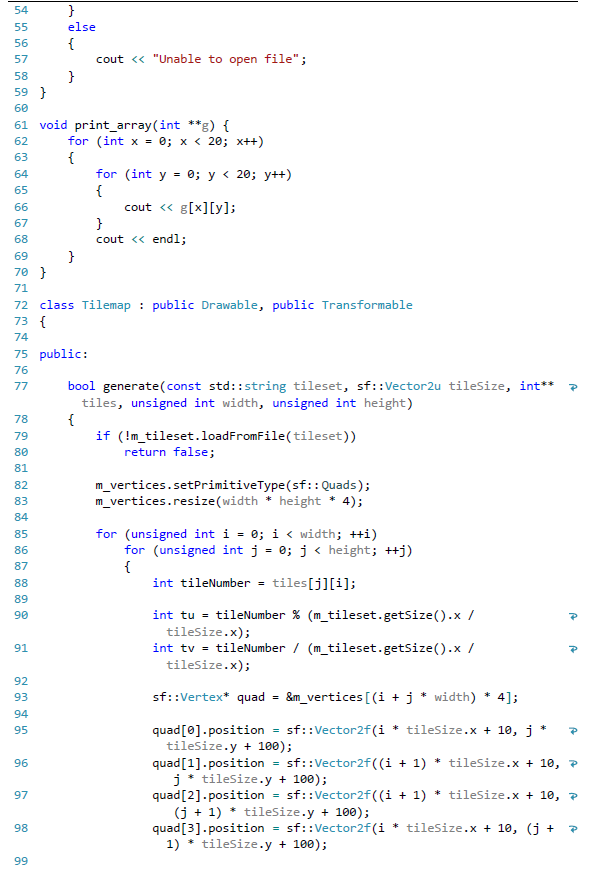
}

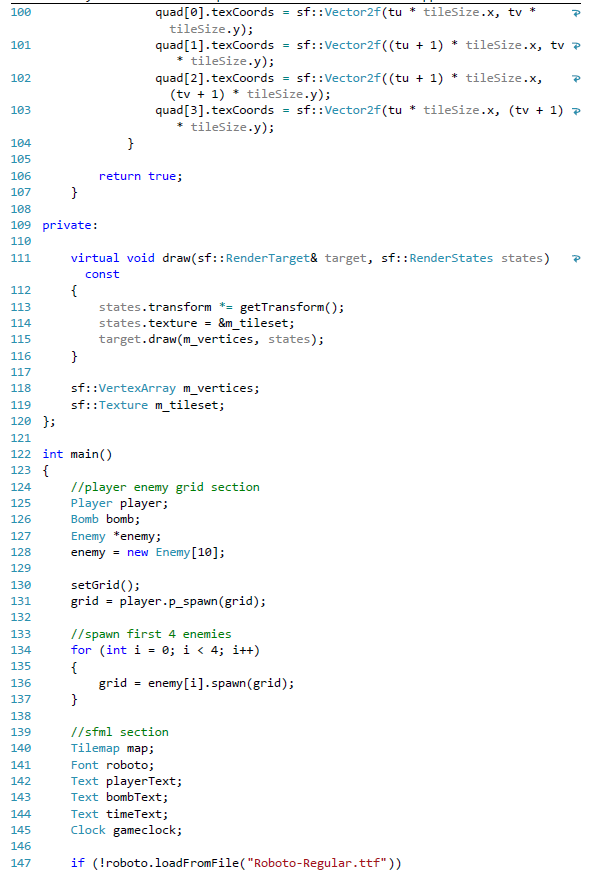
}

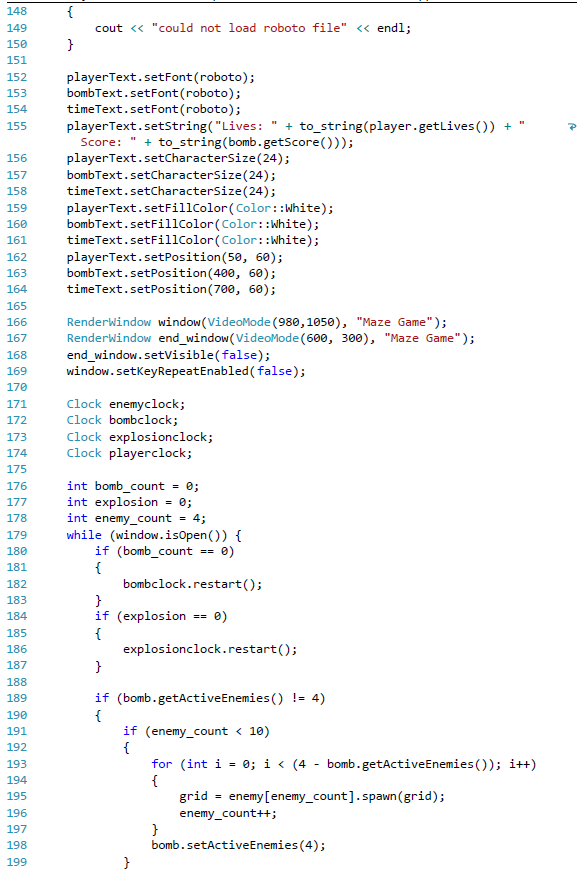
# C++ Code

MazeGame.cpp



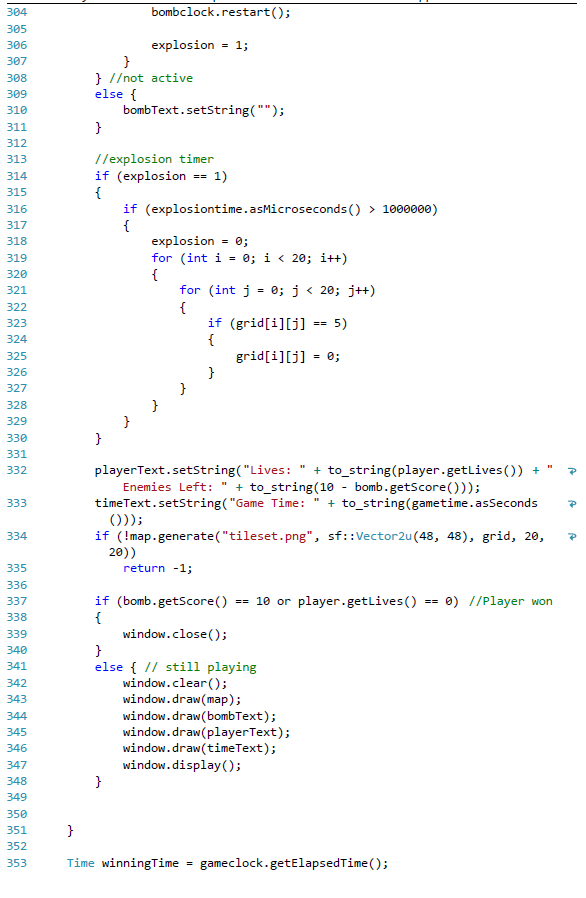


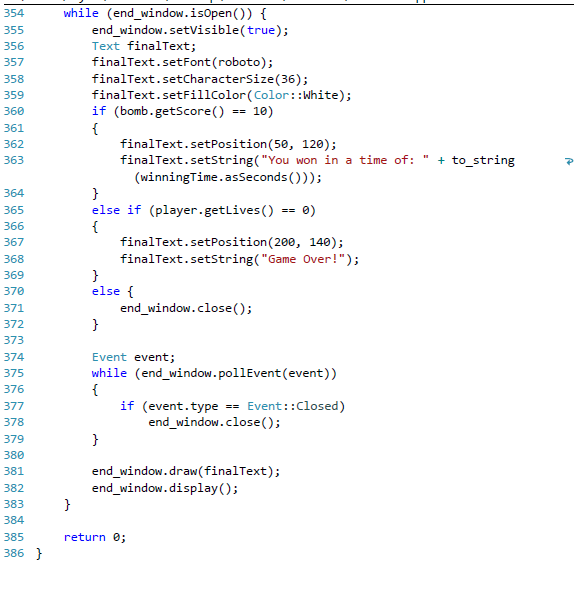


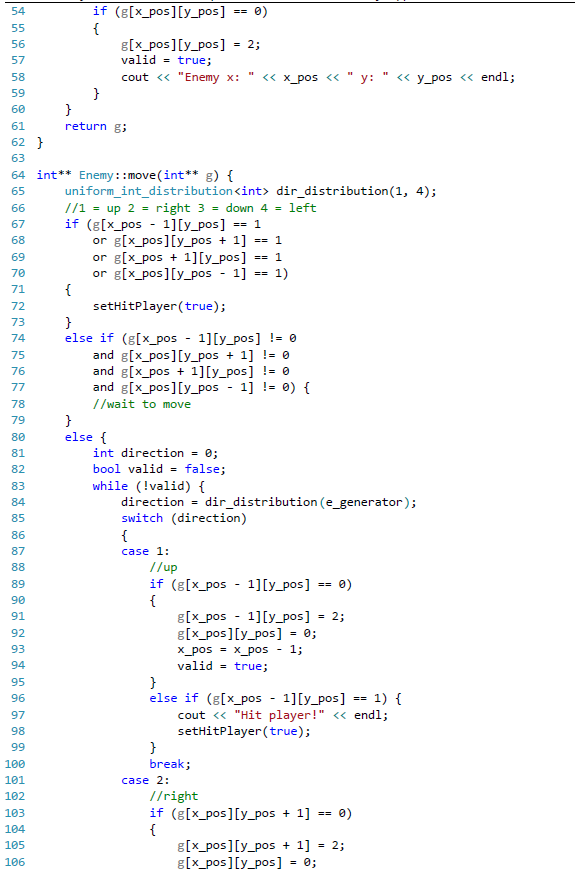
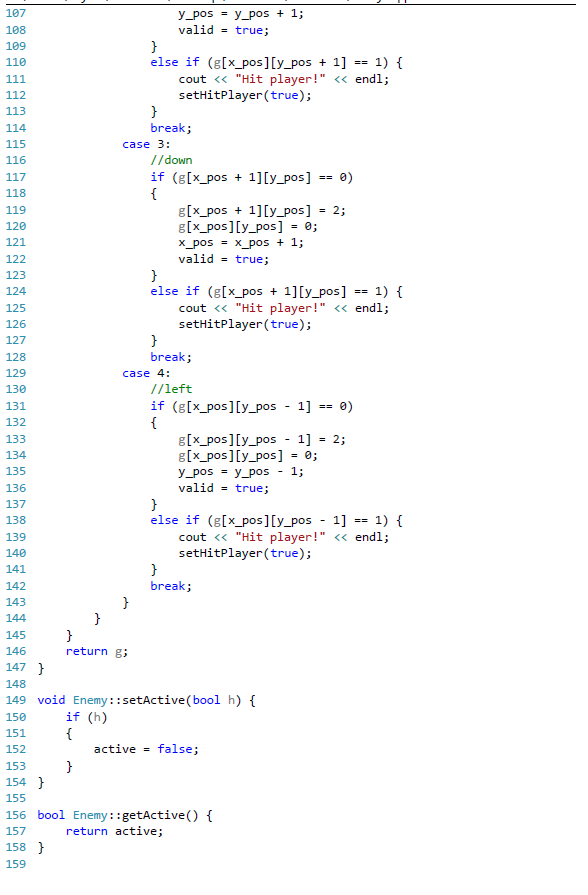
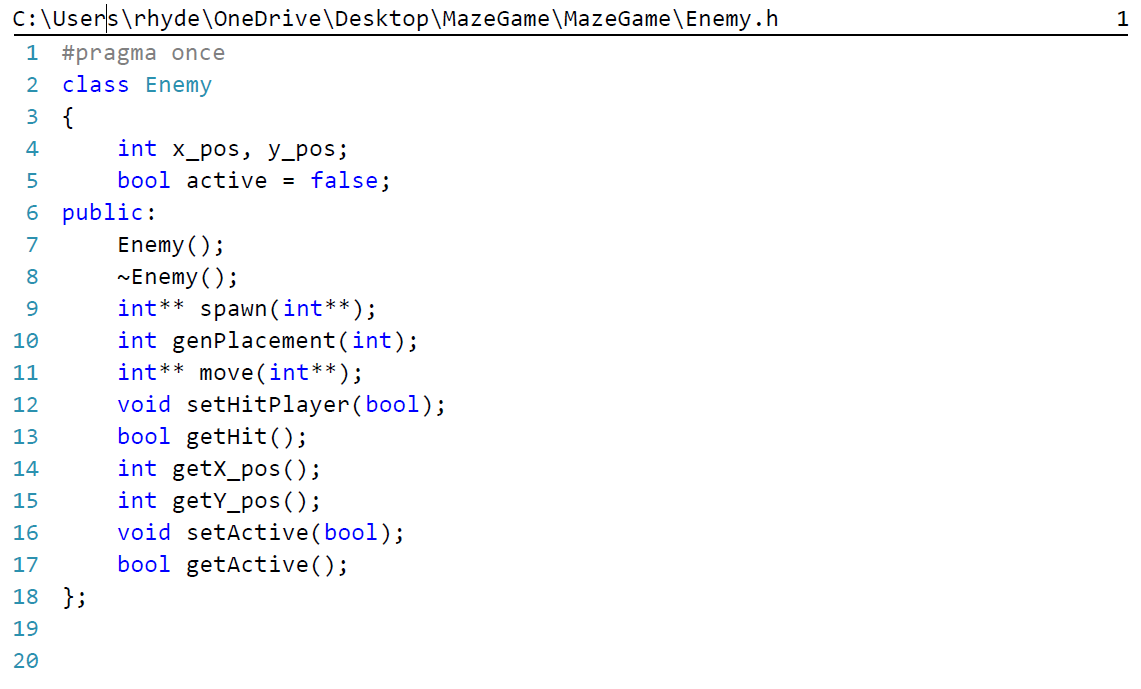






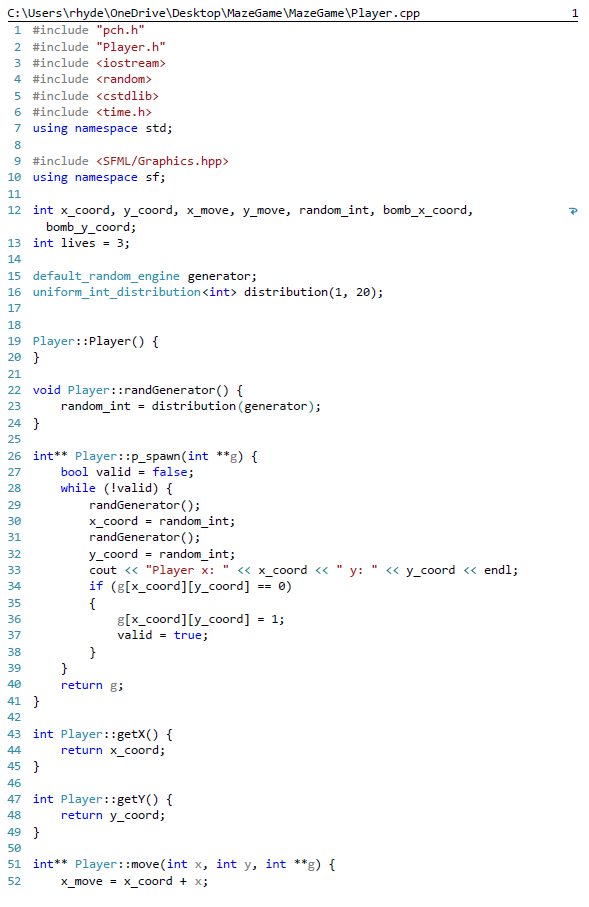
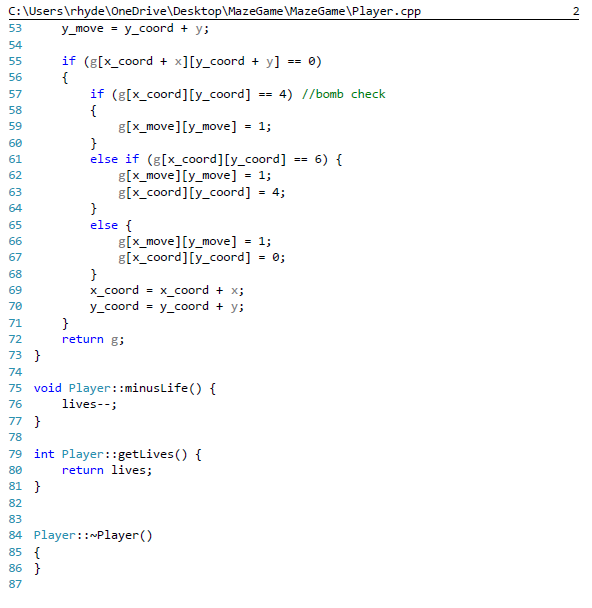




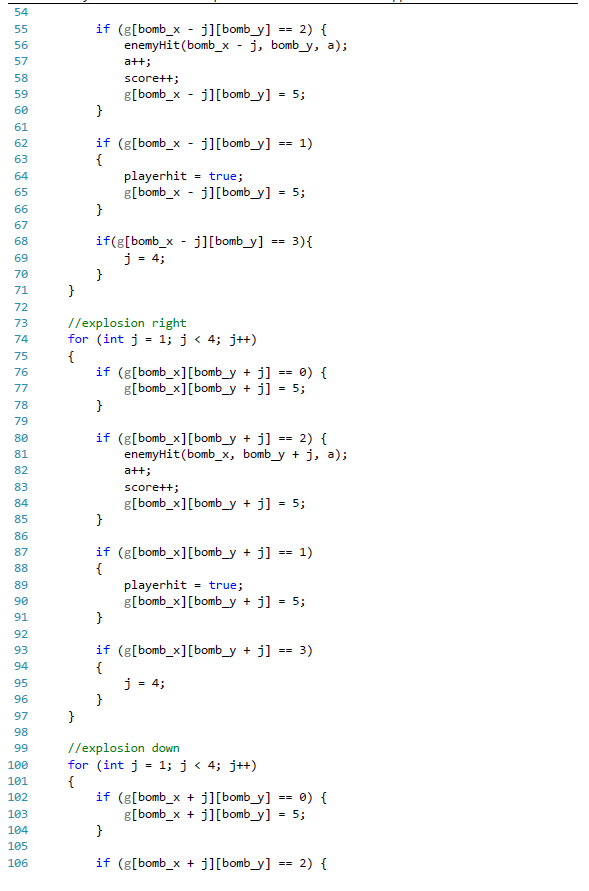
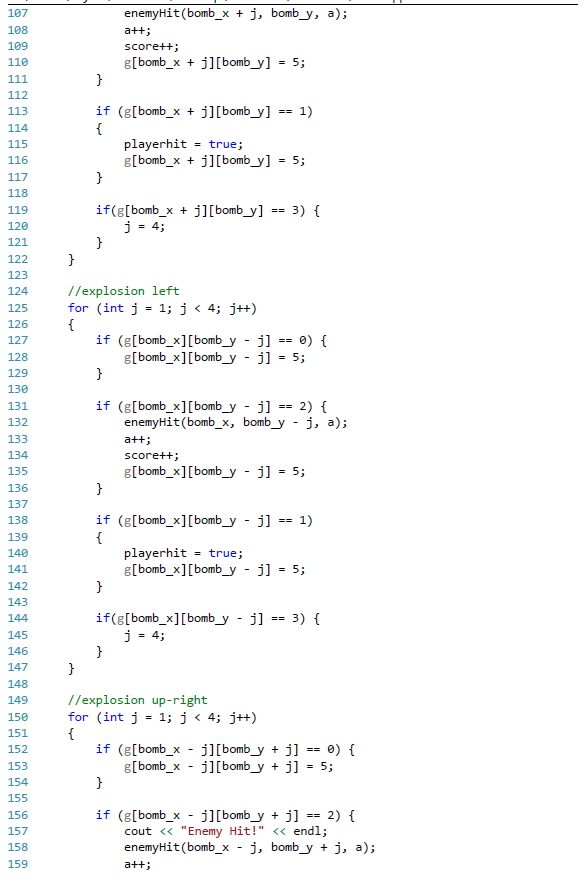
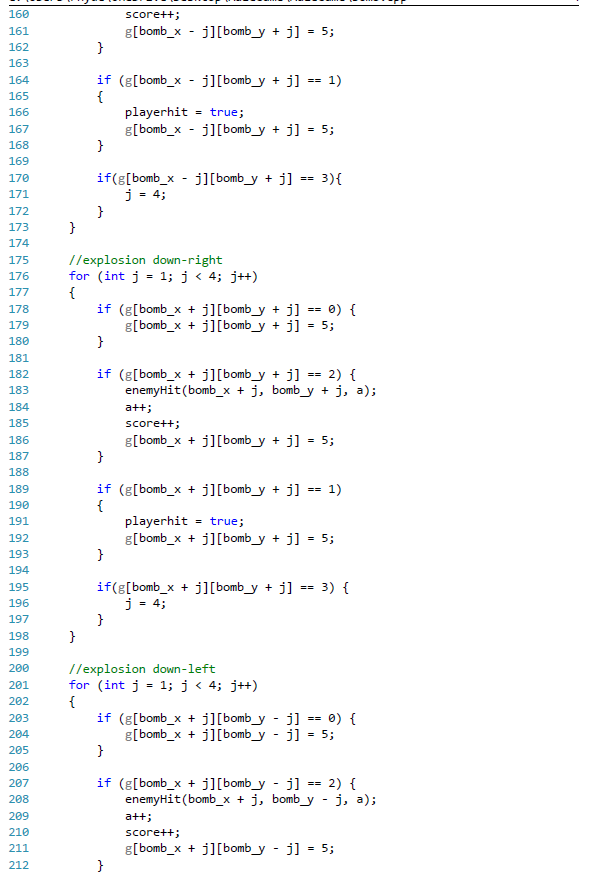
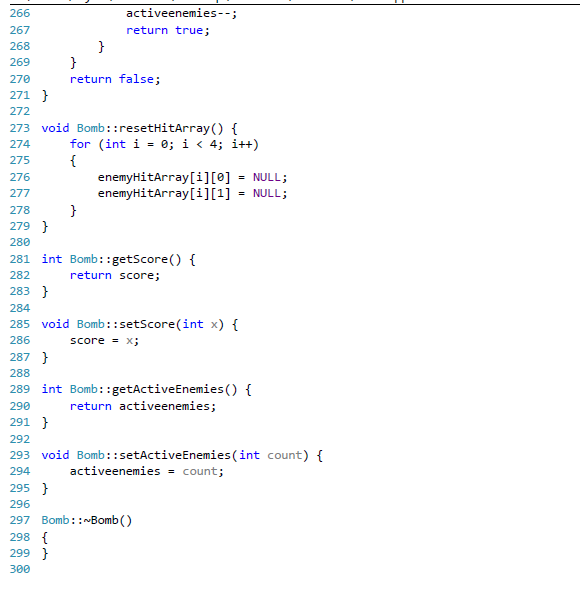
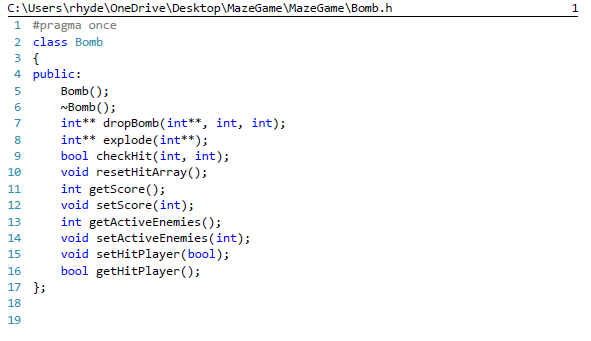
   

Enemy.cpp

Enemy.h

Player.cpp

Player.h

Bomb.cpp

Bomb.h

# Test Data

## Player and Enemy movement

Example of a the player character (represented by a wizard) moving up in the maze, and the enemies (represented by soldiers) moving randomly.

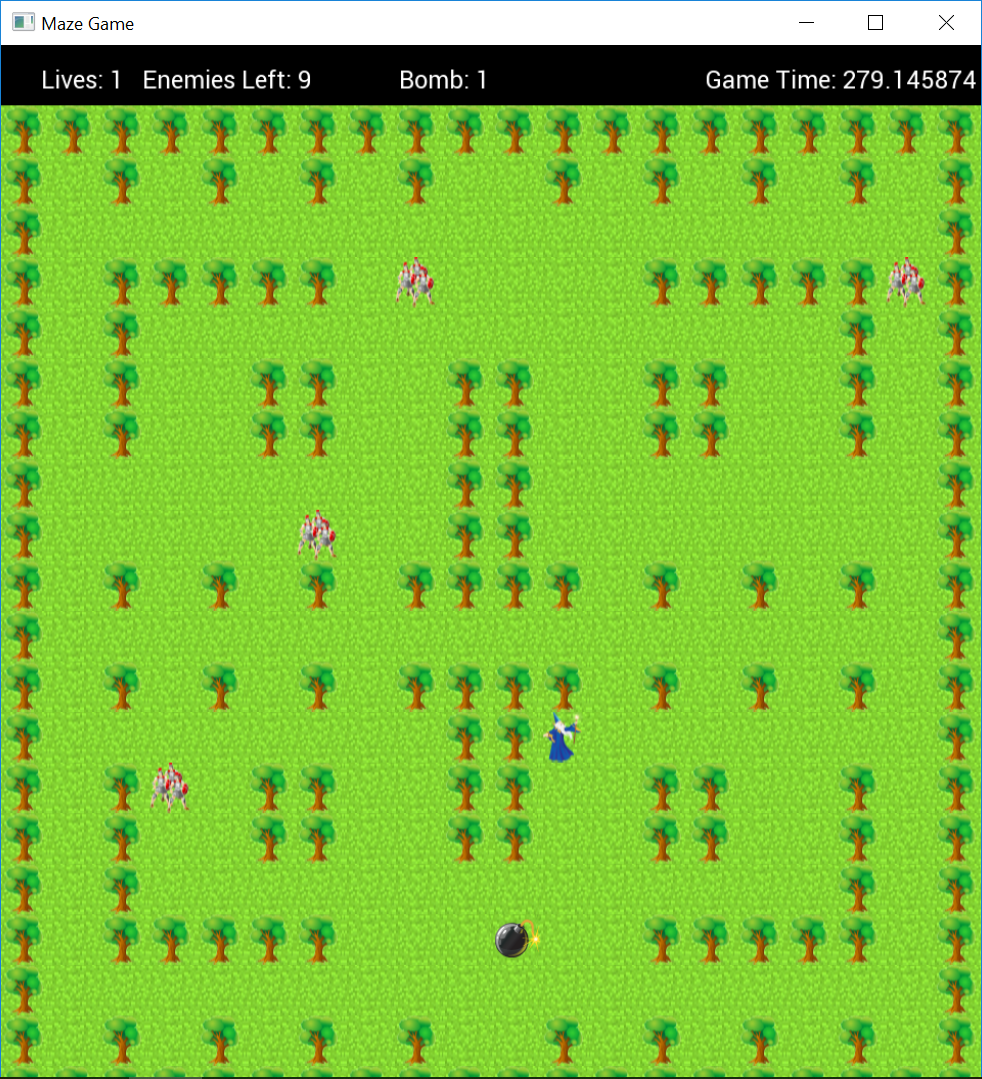
Before movement:



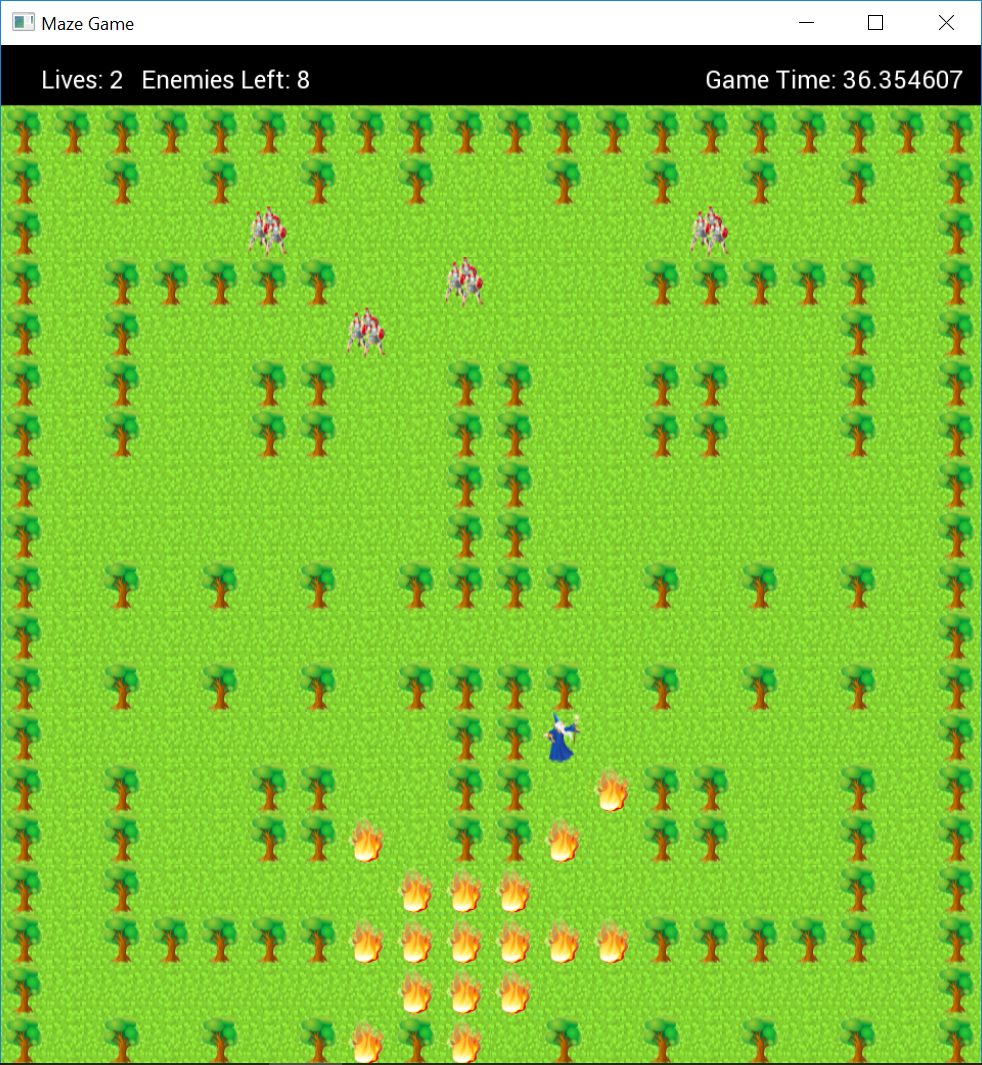
After movement:



### Bomb explosion test



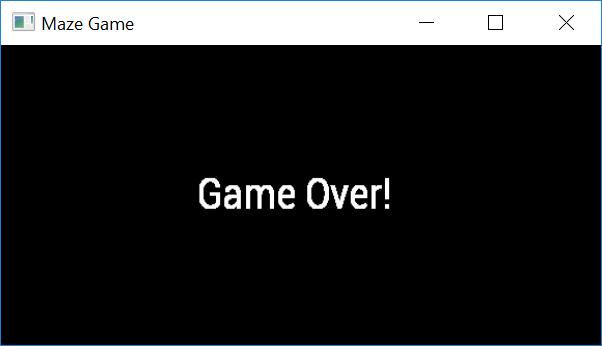
If the user presses ‘space’ then the player character should drop a ‘bomb’ sprite. The image to the left represents this action.



After a countdown of 3 seconds (Represented by the bomb count down timer in the above image) the bomb is expected to ‘explode’, killing enemies and player up to 4 tiles away in 8 directions shown in the image to the left.

### Win/Lose Screen

Game Over:



Winning Screen:

