Computing 3: 2018/2019

Joint DSA & Programming Assignment

Name: Francis Hall

Student Number: C00220910

Lecturers: Áine Byrne / Noel O’Hara

Submission Date: 26/11/2018

**Introduction**

The purpose of this project was to create a game using C++ with SFML libraries as a joint assessment between DSA and Advanced Programming.

This report shall include a description of the game and its entities and the data structures and functions used to create it.

**Description**

The game is comprised of a “tilemap” made from tiles of equal sizes which create a maze. The maze includes either wall tiles or floor tiles.

A “player character” which the user controls using the arrow keys may navigate the maze by moving on top of the floor tiles and may not pass through wall tiles.

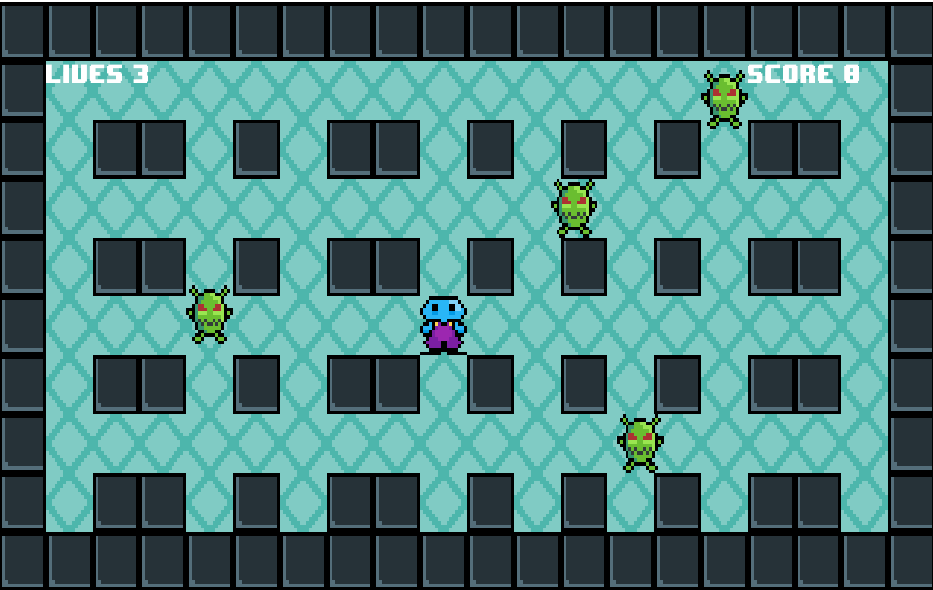
“Enemies” are also present on the map. Each enemy has a path upon which it moves, and like the player character, may not pass through wall tiles.

The player character may place a bomb on the map, which, when the user presses the “Space” key will be placed at the player character’s location. The bomb has a three second timer(fuse), after which time, the bomb detonates, with explosion tiles surrounding the bomb in four directions.

The object of the game is for the player character to place bombs in the path of the enemy characters. Once an enemy collides with an explosion tile, they are destroyed, and the player’s score is increased.

Once all enemies are destroyed, the game is won by the user.

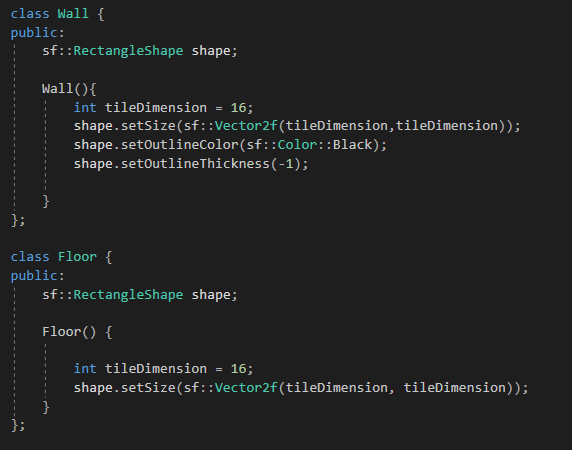
Alternatively, if the player character collides with an enemy, one life is lost. Once all lives are lost, the user loses the game.



**Data Structures**

**Wall and Floor:**

Wall and Floor tiles are created similarly, from classes containing an sf::RectangleShape and an integer value for their size.

Their size is set in the constructor using the shape.setSize() function and their size value. Walls, uniquely contain a couple of extra elements to create a black outline on them for aesthetic purposes.

**Maze:**

The maze is implemented as a 2D array of integers, with each element being either a 1 or a 0. When the game loop starts, a for loop is used to iterate through the array and set tiles at the appropriate tile positions on the map. If the element of the list contains a 1, a wall tile is placed. Otherwise, a floor tile is placed.

Additionally, whenever a wall tile is placed, its position is pushed into a vector which will be used later to check for collisions with walls.

**Player and Enemy:**

The Player and Enemy objects are created similarly to the wall and floor tiles, except that they use an sf::Sprite rather than an sf::RectangleShape.

Their scale is set in the constructor simply as 1:1, given that the texture intended for these sprites use the appropriate dimensions (16x16).

The enemy class is slightly more complex than the player, in that it used a Boolean value to show whether or not it has been destroyed in-game, and a function to “kill” the enemy.

The kill() function is intended to set its Boolean value to true, to denote that it has been destroyed, set its position to that of wall tile outside of the playing area, and finally to indicate that this enemy should no longer be drawn to the window.

**Bomb and Explosion:**

The Bomb and Explosion objects are also made from RectangleShape.

The bomb has four explosion tiles which, after the bomb has been placed for 3 seconds, will appear on the map in four tiles adjacent to the bomb’s position for 1.5 seconds. After which time, the bomb and explosion are moved out of the play area and are no longer drawn until they are placed again.



**Lives and Score:**

The lives and Score values are made from integers passed into a string which is set to an sf::Text object. Lives are intended to decrement when the player collides with an enemy and Score is intended to increment when enemies are destroyed.

These integers ultimately govern the endgame condition as to whether the game is won or lost.

**Functions**

**Movement:**

The move() function on a shape or a sprite will move it along a set axis. It has been used to govern the movement of the player character and the enemy characters.

The player character moves in the appropriate direction on keypresses (arrow keys) and can thus be freely controlled by the user. 

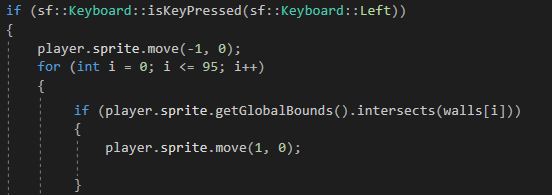
Enemy movement uses the same move function, but each enemy moves in a fixed direction until they collide with a wall, at which point, they will reverse their direction. This gives the effect of the enemy characters “patrolling” a path on the map.



**Collisions:**

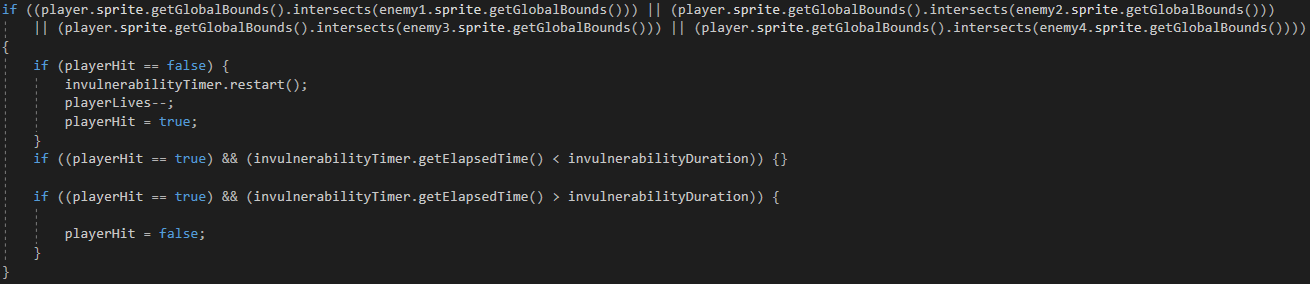
Collisions are one of the most important elements of the game, and certain objects colliding with other objects must have appropriate effects. The “intersects” function has been used throughout the project.

For the Player character, in order to prevent it from becoming spectral and passing through walls, the intersects function was used to effectively undo its movement if it collides with a wall tile.

As mentioned previously, when the wall tiles are placed on the map, their positions are pushed into a vector called “walls”. Whenever the player moves, this vector is iterated through to check, using the intersects() function, whether the player has collided with a wall.

The enemy sprites use a similar function, but when they collide with a wall, they will also reverse their direction.

This method of using the intersects function between two objects’ positions also helps to alter the player’s lives when it collides with an enemy.



Here, there is also an “invulnerability timer” using an sf::Clock and an sf::Time object and a Boolean value. This way, if a player collides with an enemy and a life is lost, the player may not lose another life for a set period, which in this case, is 1.5 seconds. This practice is common in games and helps to prevent a fast and continuous loss of lives upon extended collisions.

Finally, when an enemy collides in the same way with an explosion, that enemy’s kill() function is called, the enemy is effectively destroyed, and the player’s score is increased.

This function governs the win condition of the game.

