

```
1 //Maze Game
2 //Rhyder Quinlan
3 //C00223030
4
5 #include "pch.h"
6 #include "Player.h"
7 #include "Enemy.h"
8 #include "Bomb.h"
9 #include <iostream>
10 #include <cstdlib>
11 #include <string>
12 #include <sstream>
13 #include <iterator>
14 #include <fstream>
15 using namespace std;
16
17 #include <SFML/Graphics.hpp>
18 using namespace sf;
19
20 int **grid;
21 int rows, columns;
22
23 void setGrid() {
24     rows = 20;
25     columns = 20;
26     grid = new int*[rows];
27
28     for (int row = 0; row < rows; row++)
29     {
30         grid[row] = new int[columns];
31     }
32
33     ifstream gridfile("grid/grid.txt");
34     string readline;
35
36     if (gridfile.is_open())
37     {
38         int i = 0;
39         while (getline(gridfile, readline))
40         {
41
42             for (int row = 0; row < rows; row++)
43             {
44                 for (int column = 0; column < columns; column++)
45                 {
46                     grid[row][column] = (readline.at(int(i)) - 48);
47                     i++;
48                 }
49             }
50             cout << endl;
51         }
52         i = 0;
53         gridfile.close();
```

```
54     }
55     else
56     {
57         cout << "Unable to open file";
58     }
59 }
60
61 void print_array(int **g) {
62     for (int x = 0; x < 20; x++)
63     {
64         for (int y = 0; y < 20; y++)
65         {
66             cout << g[x][y];
67         }
68         cout << endl;
69     }
70 }
71
72 class Tilemap : public Drawable, public Transformable
73 {
74
75 public:
76
77     bool generate(const std::string tileset, sf::Vector2u tileSize, int** tiles, unsigned int width, unsigned int height)
78     {
79         if (!m_tileset.loadFromFile(tileset))
80             return false;
81
82         m_vertices.setPrimitiveType(sf::Quads);
83         m_vertices.resize(width * height * 4);
84
85         for (unsigned int i = 0; i < width; ++i)
86             for (unsigned int j = 0; j < height; ++j)
87             {
88                 int tileNumber = tiles[j][i];
89
90                 int tu = tileNumber % (m_tileset.getSize().x / tileSize.x);
91                 int tv = tileNumber / (m_tileset.getSize().x / tileSize.x);
92
93                 sf::Vertex* quad = &m_vertices[(i + j * width) * 4];
94
95                 quad[0].position = sf::Vector2f(i * tileSize.x + 10, j * tileSize.y + 100);
96                 quad[1].position = sf::Vector2f((i + 1) * tileSize.x + 10, j * tileSize.y + 100);
97                 quad[2].position = sf::Vector2f((i + 1) * tileSize.x + 10, (j + 1) * tileSize.y + 100);
98                 quad[3].position = sf::Vector2f(i * tileSize.x + 10, (j + 1) * tileSize.y + 100);
99 }
```

```
100         quad[0].texCoords = sf::Vector2f(tu * tileSize.x, tv *  
        tileSize.y);  
101         quad[1].texCoords = sf::Vector2f((tu + 1) * tileSize.x, tv *  
        tileSize.y);  
102         quad[2].texCoords = sf::Vector2f((tu + 1) * tileSize.x, (tv + 1) *  
        tileSize.y);  
103         quad[3].texCoords = sf::Vector2f(tu * tileSize.x, (tv + 1) *  
        tileSize.y);  
104     }  
105  
106     return true;  
107 }  
108  
109 private:  
110  
111     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const  
112     {  
113         states.transform *= getTransform();  
114         states.texture = &m_tileset;  
115         target.draw(m_vertices, states);  
116     }  
117  
118     sf::VertexArray m_vertices;  
119     sf::Texture m_tileset;  
120 };  
121  
122 int main()  
123 {  
124     //player enemy grid section  
125     Player player;  
126     Bomb bomb;  
127     Enemy *enemy;  
128     enemy = new Enemy[10];  
129  
130     setGrid();  
131     grid = player.p_spawn(grid);  
132  
133     //spawn first 4 enemies  
134     for (int i = 0; i < 4; i++)  
135     {  
136         grid = enemy[i].spawn(grid);  
137     }  
138  
139     //sfml section  
140     Tilemap map;  
141     Font roboto;  
142     Text playerText;  
143     Text bombText;  
144     Text timeText;  
145     Clock gameclock;  
146  
147     if (!roboto.loadFromFile("Roboto-Regular.ttf"))
```

```
148     {
149         cout << "could not load roboto file" << endl;
150     }
151
152     playerText.setFont(roboto);
153     bombText.setFont(roboto);
154     timeText.setFont(roboto);
155     playerText.setString("Lives: " + to_string(player.getLives()) + "
        Score: " + to_string(bomb.getScore()));
156     playerText.setCharacterSize(24);
157     bombText.setCharacterSize(24);
158     timeText.setCharacterSize(24);
159     playerText.setFillColor(Color::White);
160     bombText.setFillColor(Color::White);
161     timeText.setFillColor(Color::White);
162     playerText.setPosition(50, 60);
163     bombText.setPosition(400, 60);
164     timeText.setPosition(700, 60);
165
166     RenderWindow window(VideoMode(980,1050), "Maze Game");
167     RenderWindow end_window(VideoMode(600, 300), "Maze Game");
168     end_window.setVisible(false);
169     window.setKeyRepeatEnabled(false);
170
171     Clock enemyclock;
172     Clock bombclock;
173     Clock explosionclock;
174     Clock playerclock;
175
176     int bomb_count = 0;
177     int explosion = 0;
178     int enemy_count = 4;
179     while (window.isOpen()) {
180         if (bomb_count == 0)
181         {
182             bombclock.restart();
183         }
184         if (explosion == 0)
185         {
186             explosionclock.restart();
187         }
188
189         if (bomb.getActiveEnemies() != 4)
190         {
191             if (enemy_count < 10)
192             {
193                 for (int i = 0; i < (4 - bomb.getActiveEnemies()); i++)
194                 {
195                     grid = enemy[enemy_count].spawn(grid);
196                     enemy_count++;
197                 }
198                 bomb.setActiveEnemies(4);
199             }
200         }
201     }
```

```
200
201     }
202
203     Event event;
204     Time gametime = gameclock.getElapsedTime();
205     Time enemytime = enemyclock.getElapsedTime();
206     Time bombtime = bombclock.getElapsedTime();
207     Time explosiontime = explosionclock.getElapsedTime();
208     Time playertime = playerclock.getElapsedTime();
209     while (window.pollEvent(event))
210     {
211         //close command
212         if (event.type == Event::Closed)
213             window.close();
214
215         //player movement
216         if (playertime.asMicroseconds() > 100000)
217         {
218             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
219             {
220                 grid = player.move(0, -1, grid);
221             }
222             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
223             {
224                 grid = player.move(0, 1, grid);
225             }
226             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
227             {
228                 grid = player.move(-1, 0, grid);
229             }
230             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
231             {
232                 grid = player.move(1, 0, grid);
233             }
234             playerclock.restart();
235         }
236
237         if (Keyboard::isKeyPressed(Keyboard::Space))
238         {
239             if (bomb_count == 0)
240             {
241                 bomb_count = 1;
242                 grid = bomb.dropBomb(grid, player.getX(), player.getY
243                 ());
244                 bombclock.restart();
245             }
246
247             if (Keyboard::isKeyPressed(Keyboard::Q))
248             {
249                 bomb.setScore(10);
250             }
251         }
```

```
252
253     //enemy movement timer
254     if (enemytime.asMicroseconds() > 700000)
255     {
256         for (int j = 0; j < 10; j++)
257         {
258             if (enemy[j].getActive())
259             {
260                 grid = enemy[j].move(grid);
261             }
262         }
263         if (enemy[0].getHit()) {
264             grid[player.getX()][player.getY()] = 0;
265             grid = player.p_spawn(grid);
266             enemy[0].setHitPlayer(false);
267             player.minusLife();
268         }
269         enemyclock.restart();
270     }
271
272     //bomb active
273     if (bomb_count == 1)
274     {
275         if (bombtime.asMicroseconds() < 1000000)
276         {
277             bombText.setString("Bomb: 3");
278         } else if (bombtime.asMicroseconds() < 2000000)
279         {
280             bombText.setString("Bomb: 2");
281         } else if (bombtime.asMicroseconds() < 3000000)
282         {
283             bombText.setString("Bomb: 1");
284         }
285         //explode
286         if (bombtime.asMicroseconds() > 3000000)
287         {
288             bomb_count = 0;
289             grid = bomb.explode(grid);
290             for (int i = 0; i < 10; i++) {
291                 if (enemy[i].getActive())
292                 {
293                     enemy[i].setActive(bomb.checkHit(enemy[i].getX_pos
294                     (), enemy[i].getY_pos()));
295                 }
296             }
297             if (bomb.getHitPlayer())
298             {
299                 grid[player.getX()][player.getY()] = 5;
300                 grid = player.p_spawn(grid);
301                 bomb.setHitPlayer(false);
302                 player.minusLife();
303             }
304             bomb.resetHitArray();
305         }
306     }
```

```
304         bombclock.restart();
305
306         explosion = 1;
307     }
308 } //not active
309 else {
310     bombText.setString("");
311 }
312
313 //explosion timer
314 if (explosion == 1)
315 {
316     if (explosiontime.asMicroseconds() > 1000000)
317     {
318         explosion = 0;
319         for (int i = 0; i < 20; i++)
320         {
321             for (int j = 0; j < 20; j++)
322             {
323                 if (grid[i][j] == 5)
324                 {
325                     grid[i][j] = 0;
326                 }
327             }
328         }
329     }
330 }
331
332 playerText.setString("Lives: " + to_string(player.getLives()) + " "
333     Enemies Left: " + to_string(10 - bomb.getScore()));
334 timeText.setString("Game Time: " + to_string(gametime.asSeconds
335     ());
336 if (!map.generate("tileset.png", sf::Vector2u(48, 48), grid, 20,
337     20))
338     return -1;
339
340 if (bomb.getScore() == 10 or player.getLives() == 0) //Player won
341 {
342     window.close();
343 }
344 else { // still playing
345     window.clear();
346     window.draw(map);
347     window.draw(bombText);
348     window.draw(playerText);
349     window.draw(timeText);
350     window.display();
351 }
352
353 Time winningTime = gameclock.getElapsedTime();
```

```
354     while (end_window.isOpen()) {
355         end_window.setVisible(true);
356         Text finalText;
357         finalText.setFont(roboto);
358         finalText.setCharacterSize(36);
359         finalText.setFillColor(Color::White);
360         if (bomb.getScore() == 10)
361         {
362             finalText.setPosition(50, 120);
363             finalText.setString("You won in a time of: " + to_string      ↗
                (winningTime.asSeconds()));
364         }
365         else if (player.getLives() == 0)
366         {
367             finalText.setPosition(200, 140);
368             finalText.setString("Game Over!");
369         }
370         else {
371             end_window.close();
372         }
373
374         Event event;
375         while (end_window.pollEvent(event))
376         {
377             if (event.type == Event::Closed)
378                 end_window.close();
379         }
380
381         end_window.draw(finalText);
382         end_window.display();
383     }
384
385     return 0;
386 }
```