

```

1 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////~
2 //
3 // Author:          Sharome Burton
4 // Email:           koulkoudakis@gmail.com
5 // Label:           P01
6 // Title:           Game: Game
7 // Course:          CMPS 2143
8 // Semester:        Spring 2021
9 //
10 // Description:
11 //      This program implements a simple game where a green player ball aims
12 //      to dodge oncoming debris
13 // Usage:
14 //      - $ ./main filename
15 //      - Use up/down arrow keys to move green player ball
16 //
17 // Files:
18 //      P01.cpp      : driver program
19 //////////////////////////////////////////////////////////////////////~
20 /
21 #include <SFML/Graphics.hpp>
22 #include <iostream>
23 #include <vector>
24 #include <time.h>
25 #include <string.h>
26
27 class Player : public sf::Drawable {
28 public:
29     Player(int w, int h) {
30
31         width = w;            // window size
32         height = h;
33         dx = -1;              // direction
34         dy = -1;
35         x = (rand() % int(width / 4)); // location (Left quarter of screen)
36         y = int(height / 2);
37         d = .1;                // distance
38
39         // define a circle with radius = 200
40         circle = new sf::CircleShape(20.f);
41
42         circle->setFillColor(sf::Color::Green);
43
44         // set the radius to whatever
45         circle->setRadius(20.f);
46
47         // change the number of sides (points) to 100

```

```
48     circle->setPointCount(100);
49
50     circle->setPosition(sf::Vector2f(x, y));
51
52 }
53
54 // Updates Player values
55 void updateP() {
56
57     position = circle->getPosition();
58     bool hit = false;
59
60     //circle->move(sf::Vector2f(x, y));
61     circle->setPosition(sf::Vector2f(x, y));
62 }
63
64 // Allows player ball to translate up or down after key input
65 void moveVert(int dir) {
66
67     switch (dir) {
68     case 0: // Move player up
69         y = position.y - 10;
70         break;
71     case 1: // Move player down
72         y = position.y + 10;
73         break;
74     }
75 }
76
77 // Checks bounding box for debris for collision-checking purposes
78 sf::FloatRect getBounds() {
79     return circle->getGlobalBounds();
80 }
81
82 private:
83     sf::CircleShape *circle; // reference to our "ball"
84     float x; // x location
85     float y; // y location
86     float dx; // "change" in x
87     float dy; // "change" in y
88     float width;
89     float height;
90     float d; // distance to move
91     sf::Vector2f position;
92
93 /**
94  * virtual = A virtual function a member function which is
95  * declared within base class and is re-defined (Overriden)
96  * by derived class.
```

```
97     * function draw:
98     *     draw an SFML object to some window
99     */
100     virtual void draw(sf::RenderTarget &target, sf::RenderStates states) const
101     {
102         //states.transform *= getTransform();
103         target.draw(*circle, states);
104     }
105 };
106 class Debris : public sf::Drawable {
107 public:
108     Debris(int w, int h) {
109
110         width = w;           // window size
111         height = h;
112         dx = -1;             // direction
113         dy = 0;
114         x = width - (rand() % 200); // location (Far right of screen)
115         y = (rand() % int(height));
116         d = 0.03*(rand() % 10 + 1); // speed
117         rwidth = 350;        // minimum width
118         rheight = 350;       // minimum height
119
120         srand(time(NULL));    // randomize size
121         rheight = rand() % rhmin + 150; // rect height
122         rwidth = rand() % rwmin + 150;   // and width
123         counter = 0;          // number of debris passed
124
125         // define a rectangle
126         rectangle = new sf::RectangleShape(sf::Vector2f(rwidth, rheight));
127
128         rectangle->setFillColor(sf::Color::Red);
129
130         rectangle->setPosition(sf::Vector2f(x, y));
131     }
132
133     // Update Debris position
134     void updateD() {
135
136         position = rectangle->getPosition();
137         bool hit = false;
138
139         if (position.x < -1 * (rwmin)) {
140             srand(time(NULL));
141             //position.x = 0;
142             //dx = 0;
143             //hit = true;
144             x = width - (rand() % 200); // location (Far right of screen)
```

```
145         position.x = x;
146         y = (rand() % int(height));
147         position.y = y;
148         d = 0.03*(counter + rand() % 10 + 1);           // speed
149         rheight = rhmin + rand() % 50;                 // rect height
150         rwidth = rwmin + rand() % 50;                  // and width
151         counter += 1;
152
153         rectangle->setSize(sf::Vector2f(rwidth, rheight));
154         // std::cout << counter << '\n';
155     }
156
157
158     x = position.x + (d * dx);
159     y = position.y + (d * dy);
160
161     //circle->move(sf::Vector2f(x, y));
162     rectangle->setPosition(sf::Vector2f(x, y));
163
164 }
165
166 // "Destroys" one piece of debris (basically sends it to right side of screen
167 //with new starting position)
168 void destroy() {
169     rectangle->setPosition(sf::Vector2f(-1 * (rwmin), y));
170 }
171
172 // Checks bounding box for debris for collision-checking purposes
173 sf::FloatRect getBounds() {
174     return rectangle->getGlobalBounds();
175 }
176
177 // Returns number of debris that have been 'generated'
178 int getCounter() {
179     return counter;
180 }
181
182 private:
183     sf::RectangleShape *rectangle;    // reference to debris
184     float x;                          // x location
185     float y;                          // y location
186     float dx;                         // "change" in x
187     float dy;                         // "change" in y
188     float width;
189     float height;
190     int rwmin;
191     int rhmin;
192     float rheight;
```

```
193     float rwidth;
194     float d;                // distance to move
195     int counter;
196     sf::Vector2f position;
197
198     /**
199     * virtual = A virtual function a member function which
200     * is declared within base class and is re-defined (Overriden)
201     * by derived class.
202     * function draw:
203     * draw an SFML object to some window
204     */
205     virtual void draw(sf::RenderTarget &target, sf::RenderStates states) const
206     {
207         //states.transform *= getTransform();
208         target.draw(*rectangle, states);
209     };
210
211     int main() {
212         int window_width = 600;
213         int window_height = 600;
214
215         sf::RenderWindow window(sf::VideoMode(window_width, window_height),
216                                 "P01");
217
218         Player B(window_width, window_height);
219         Debris D(window_width, window_height);
220
221         sf::Font font;
222         if (!font.loadFromFile("arial.ttf"))
223         {
224             std::cout << "Error: no font file!\n";
225         }
226         sf::Text text;
227         text.setFont(font); // font is a sf::Font
228         text.setFillColor(sf::Color::Red);
229         text.setString("Game Score: 0");
230         text.setCharacterSize(30);
231         text.setPosition(10, 10);
232
233         int score = 0;        // Increments when debris is successfully avoided
234         int penalty = 0;      // Decrements when debris is collided with
235         std::string scoreString; // Score is stored as string for display
236
237         while (window.isOpen()) {
238             sf::Event event;
239             while (window.pollEvent(event)) {
```

```

240         if (event.type == sf::Event::Closed)
241             window.close();
242
243         else if (event.type == sf::Event::KeyPressed)
244             if (event.text.unicode < 128)
245                 if (event.text.unicode == 73) {
246                     std::cout << "int(" << event.key.code << ") letter("
247                     << char(event.key.code + 65) << ")" << std::endl;
248                     std::cout << "UP\n";
249                     B.moveVert(0);
250                 }
251                 else if (event.text.unicode == 74) {
252                     std::cout << "int(" << event.key.code << ") letter("
253                     << char(event.key.code + 65) << ")" << std::endl;
254                     std::cout << "DOWN\n";
255                     B.moveVert(1);
256                 }
257             }
258
259         B.updateP();
260         D.updateD();
261
262         sf::FloatRect recB = B.getBounds();
263         sf::FloatRect recD = D.getBounds();
264
265         // Checks for collision and updates score
266         if (recB.intersects(recD)) {
267             std::cout << "Minus one point!\n";
268             penalty += 2; // Penalty is incremented by 2 because counter
269                         // increments
270                         // by 1 for each debris 'created' so subtracting 2
271                         // when
272                         // a collision takes place is the practical way of
273                         // lowering
274                         // the score by 1 (counter - penalty = score)
275             D.destroy();
276         }
277
278         score = D.getCounter() - penalty; // if collision
279         // std::cout<<"Score: " << score << '\n';
280
281         scoreString = "Game Score: " + std::to_string(score);
282         std::cout << scoreString << '\n';
283         text.setString(scoreString);

```

```
284         window.draw(D);
285         window.draw(text);
286         window.display();
287     }
288
289     return 0;
290 }
```