```
 1  using Final_Project;
 2  using Final_Project.Properties;
 3  using System;
 4  using System.Collections.Generic;
 5  using System.Linq;
 6  using System.Security.Cryptography.X509Certificates;
 7  using System.Security.Policy;
 8  using System.Text;
 9  using System.Timers;
10  using static Final_Project.Form1;
11
12  namespace BasicAI
13  {
14      //class used to combine everything needed to make an enemy to keep
          most things self contained
15      public class Gunner
16      {
17          //declare variables
18          public PictureBox picEnemy;
19          PictureBox player;
20          Label projectile;
21          Label[] projectiles;
22          Label healthBar;
23          Label playerHealth;
24          public int healthcap;
25          public double speed;
26
27          double bulletX = 1;
28          double bulletY = 1;
29
30          Boolean bulletFired = false;
31          public Point waypoint;
32
33          public int width = 0;
34          public int height = 0;
35
36          public WeaponSelected currentWeapon;
37
38          Form1 form = Form1.getInstance();
39
40          System.Random r = new System.Random((int)
              System.DateTime.Now.Ticks);
41
42          public Gunner(PictureBox picEnemy, PictureBox player, Label
              projectile, Label[] projectiles, Label healthBar, double speed,
              int healthcap, Label playerHealth)
43          {
44              //assign each variable to the parameters
45              this.picEnemy = picEnemy;
```

```
46              this.player = player;
47              this.projectile = projectile;
48              this.projectiles = projectiles;
49              this.healthBar = healthBar;
50              this.playerHealth = playerHealth;
51              currentWeapon = WeaponSelected.SWORD;
52              waypoint = new Point(500,500);
53
54              this.healthcap = healthcap;
55              this.speed = speed;
56
57              healthBar.Width = healthcap;
58
59              picEnemy.Visible = false;
60              healthBar.Visible = false;
61              projectile.Visible = false;
62          }
63
64          public void Update()
65          {
66              //auto updates depending on the health
67              if (healthBar.Width > 0)
68              {
69                  pursuit();
70
71
72              }
73
74              else
75              {
76                  //MessageBox.Show("Congrats! The tyrant has been
                      killed!");
77
78              }
79          }
80
81          private void pursuit()
82          {
83              //tries to go to the player's position
84              double posX = picEnemy.Left + (speed * getDeltaX(true));
85              double posY = picEnemy.Top + (speed * getDeltaY(true));
86
87
88              if (enemyTouch() || swordTouch())
89              {
90                  //posX -= (1 * getDeltaX(false));
91                  //posY -= (1 * getDeltaY(false));
92
93                  picEnemy.Left = (int)posX;
```

```
94                  picEnemy.Top = (int)posY;
95              }
96
97              if (!bulletFired)
98              {
99
100                 bulletX = getDeltaX(false);
101                 bulletY = getDeltaY(false);
102
103                 bulletFired = true;
104                 projectile.Left = (int)posX + picEnemy.Width / 2;
105                 projectile.Top = (int)posY + picEnemy.Height / 2;
106             }
107             else
108                 fireProj();
109
110
111
112                 picEnemy.Left = (int)posX;
113                 picEnemy.Top = (int)posY;
114
115         }
116
117         private void fireProj()
118         {
119
120             //shoots the projectile
121             //change the rounding to a serate variable
122
123             //double deltaX = getDeltaX(false);
124             //double deltaY = getDeltaY(false);
125
126             double x = projectile.Left + (0.02 * bulletX);
127             double y = projectile.Top + (0.02 * bulletY);
128
129             projectile.Left = (int)x;
130             projectile.Top = (int)y;
131
132             if(bulletOOB() || bulletHit() || shieldBlock())
133             {
134                 if(bulletHit())
135                 {
136                     playerHealth.Width -= 40;
137                 }
138                 resetBullet();
139             }
140         }
141
142
```

```
143         public void move(int x, int y)
144         {
145             //move the enemy to the given point
146             picEnemy.Left = x;
147             picEnemy.Top = y;
148         }
149 
150         public void respawn(int x, int y)
151         {
152             //reset the enemy
153             picEnemy.Visible = true;
154             healthBar.Visible = true;
155             projectile.Visible = true;
156 
157             healthBar.Width = healthcap;
158             move(x, y);
159         }
160 
161         public void reset()
162         {
163             //turns off the enemy
164             picEnemy.Visible = false;
165             healthBar.Visible = false;
166             projectile.Visible = false;
167 
168             healthBar.Width = healthcap;
169 
170             move(0, 0);
171         }
172 
173         public void dynamicWaypoint(int width, int height)
174         {
175             //changes where the point is based off of where the player is
                  to avoid it
176             if (player.Left >= width / 2)
177                 waypoint.X = 0 + (picEnemy.Width * 2);
178             else if (player.Left < width / 2)
179                 waypoint.X = width - (picEnemy.Width * 2);
180 
181             if (player.Top >= height / 2)
182                 waypoint.Y = 0 + (picEnemy.Height * 2);
183             else if (player.Top < height / 2)
184                 waypoint.Y = height - (picEnemy.Height * 2);
185         }
186 
187         public Boolean enemyTouch()
188         {
189             //returns if the enemy and the player are touching
190             return picEnemy.Bounds.IntersectsWith(player.Bounds);
```

```
191         }
192
193         private Boolean shieldBlock()
194         {
195             //returns if the shield is touching the projectile
196             for(int i = 0; i < projectiles.Length; i++)
197             {
198                 if(currentWeapon == WeaponSelected.SHIELD && projectiles
                      [i].Bounds.IntersectsWith(projectile.Bounds))
199                 {
200                     return true;
201                 }
202             }
203             return false;
204
205
206         }
207         public Boolean swordTouch()
208         {
209             //returns if the sword is touching the enemy
210
211             for (int i = 0; i < projectiles.Length; i++)
212             {
213                 if (picEnemy.Bounds.IntersectsWith(projectiles[i].Bounds))
214                 {
215                     //form.spawnHeart();
216                     healthBar.Width -= 5;
217                     return true;
218                 }
219             }
220
221             return false;
222         }
223
224         public int getHealth()
225         {
226             //returns the healthbar's width aka health
227             return healthBar.Width;
228         }
229
230         public Boolean bulletHit()
231         {
232             //returns if the bullet hits the player
233             return player.Bounds.IntersectsWith(projectile.Bounds);
234         }
235
236         private Boolean bulletCollide()
237         {
238             //returns if the bullet hits anything in general
```

```
239             return player.Bounds.IntersectsWith(projectile.Bounds) ||
                  (projectile.Left > width || projectile.Right < 0 ||
                  projectile.Top > height || projectile.Bottom < 0);
240         }
241
242         private Boolean bulletOOB()
243         {
244             //returns if the bullet is out of bounds aka out of the form
245             return (projectile.Left > width || projectile.Right < 0 ||
                  projectile.Top > height || projectile.Bottom < 0);
246         }
247
248         private void resetBullet()
249         {
250             //puts bullet back to the enemy
251             bulletFired = false;
252
253             projectile.Left = picEnemy.Left + picEnemy.Width / 2;
254             projectile.Top = picEnemy.Top + picEnemy.Height / 2;
255
256         }
257         private double getDeltaY(Boolean usingWaypoint)
258         {
259             //gets the difference in y coordinates of either the waypoint
                  or the player
260             int playerX, playerY;
261             if (usingWaypoint)
262             {
263                 playerX = waypoint.X;// - (player.Width / 2);
264                 playerY = waypoint.Y;// + (player.Height / 2);
265             }
266
267             else
268             {
269                 playerX = player.Left;
270                 playerY = player.Top;
271             }
272
273             int enemyX = picEnemy.Left;// - (picEnemy.Width / 2);
274             int enemyY = picEnemy.Top;// + (picEnemy.Height / 2);
275
276             double deltaX = playerX - enemyX;
277             double deltaY = playerY - enemyY;
278
279             return deltaY;
280
281         }
282
283         private double getDeltaX(Boolean usingWaypoint)
```

```
284         {
285             //returns either the x difference of the waypoint or the
                  player
286             int playerX, playerY;
287             if (usingWaypoint)
288             {
289                 playerX = waypoint.X;// - (player.Width / 2);
290                 playerY = waypoint.Y;// + (player.Height / 2);
291             }
292 
293             else
294             {
295                 playerX = player.Left;
296                 playerY = player.Top;
297             }
298 
299             int enemyX = picEnemy.Left;// - (picEnemy.Width / 2);
300             int enemyY = picEnemy.Top;// + (picEnemy.Height / 2);
301 
302             double deltaX = playerX - enemyX;
303             double deltaY = playerY - enemyY;
304 
305             return deltaX;
306 
307         }
308     }
309 }
310 
```