```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5
6  public class MyVector3
7  {
8      //Declares 3 float variables as each of the vector components
9      public float x, y, z;
10     //A constructor is created to instantiate each of the values
11     //Any time a new MyVector3 object is created it references this
         constructor
12     public MyVector3(float x, float y, float z)
13     {
14         this.x = x;
15         this.y = y;
16         this.z = z;
17
18 ;    }
19
20     //Basic Vector
21     //Adds one vector from another
22     public static MyVector3 AddVector(MyVector3 a, MyVector3 b)
23     {
24         MyVector3 rv = new MyVector3(0, 0, 0);
25
26         rv.x = a.x + b.x;
27         rv.y = a.y + b.y;
28         rv.z = a.z + b.z;
29
30         return rv;
31     }
32     //Subtracts one vector from another
33     public static MyVector3 SubtractVector(MyVector3 a, MyVector3 b)
34     {
35
36         MyVector3 rv = new MyVector3(0, 0, 0);
37
38         rv.x = a.x - b.x;
39         rv.y = a.y - b.y;
40         rv.z = a.z - b.z;
41         return rv;
42     }
43     public static MyVector3 ScaleVector(MyVector3 v, float scalar)
44     {
45         //Takes Vector and scales it up or down with scalar variable
46         MyVector3 rv = new MyVector3(0, 0, 0);
47
48         rv.x = v.x * scalar;
49         rv.y = v.y * scalar;
50         rv.z = v.z * scalar;
51         return rv;
52     }
```

```
53      public static MyVector3 DivideVector(MyVector3 v, float divisor)
54      {
55          //Takes Vector and scales it up or down with divisor variable
56          MyVector3 rv = new MyVector3(0, 0, 0);
57
58          rv.x = v.x / divisor;
59          rv.y = v.y / divisor;
60          rv.z = v.z / divisor;
61          return rv;
62      }
63      //Operator overloads
64      public static MyVector3 operator +(MyVector3 overload1, MyVector3
          overload2)
65      {
66          return AddVector(overload1, overload2);
67      }
68      public static MyVector3 operator -(MyVector3 overload1, MyVector3
          overload2)
69      {
70          return SubtractVector(overload1, overload2);
71      }
72      public static MyVector3 operator /(MyVector3 overload1, float overload2)
73      {
74          return DivideVector(overload1, overload2);
75      }
76      public static MyVector3 operator *(MyVector3 overload1, float overload2)
77      {
78          return ScaleVector(overload1, overload2);
79      }
80
81      public float Length()
82      {
83          //Calculates the hypotenuse of Vector
84          float rv = 0.0f;
85          rv = Mathf.Sqrt(x * x + y * y + z * z);
86
87          return rv;
88
89      }
90      public float LengthSq()
91      {
92          float rv = 0.0f;
93
94          rv = x * x + y * y + z * z;
95
96          return rv;
97      }
98      public Vector3 ToUnityVector()
99      {
100         //Converts class MyVector3 to the Vector3 unity
101         Vector3 rv = new Vector3(x, y, z);
102
103
```

```
104         return rv;
105     }
106     public Vector3[] ToUnityArrayVector(int ArraySize)
107     {
108         //Converts class MyVector3 to the Vector3 unity
109 
110             Vector3[] rv = new Vector3[ArraySize];
111 
112         return rv;
113     }
114     public static MyVector3 Zero()
115     {
116         //Converts class MyVector3 to the Vector3 unity
117         MyVector3 rv = new MyVector3(0, 0, 0);
118 
119 
120         return rv;
121     }
122     public MyVector3 NormalizeVector()
123     {
124         MyVector3 rv = new MyVector3(x, y, z);
125         rv = DivideVector(rv, rv.Length());
126         return rv;
127     }
128     public static float VectorDot(MyVector3 a, MyVector3 b)
129     {
130         float rv = 0.0f;
131 
132         //if (ShouldNormalize)
133         //{
134             MyVector3 normA = a.NormalizeVector();
135             MyVector3 normB = b.NormalizeVector();
136 
137             rv = normA.x * normB.x + normA.y * normB.y + normA.z * normB.z;
138         //}
139         //else
140         //{
141         //    rv = a.x * b.x + a.y + b.y + a.z * b.z;
142         //}
143 
144         return rv;
145     }
146     public static float Dot(MyVector3 a, MyVector3 b)
147     {
148         float rv = 0.0f;
149 
150         rv = a.x * b.x + a.y + b.y + a.z * b.z;
151 
152         return rv;
153     }
154     public MyVector3 MoveCube(MyVector3 cube1, MyVector3 cube2)
155     {
156 
```

```
157         if (Input.GetKey(KeyCode.Space))
158         {
159             if (cube1 != cube2)
160             {
161                 cube1 = new MyVector3(cube2.x, cube2.y, cube2.z);
162             }
163             else
164             {
165 
166             }
167         }
168         return cube1;
169     }
170     public static MyVector3 NewVectorPos(MyVector3 currentpos, MyVector3
      newpos)
171     {
172         currentpos = new MyVector3(currentpos.x + newpos.x, currentpos.y +
          newpos.y, currentpos.z + newpos.z);
173         return currentpos;
174     }
175     public static MyVector3 Increment(MyVector3 currentpos, float x, float
      y, float z)
176     {
177         currentpos = new MyVector3(currentpos.x + x, currentpos.y + y,
          currentpos.z + z);
178         return currentpos;
179     }
180     public static float GetX(MyVector3 v)
181     {
182         //Returns X component
183         float x = 0.0f;
184         x = v.x;
185         return x;
186     }
187     public static float GetY(MyVector3 v)
188     {
189         //Returns Y component
190         float y = 0.0f;
191         y = v.y;
192         return y;
193     }
194     public static float GetZ(MyVector3 v)
195     {
196         //Returns Z component
197         float z = 0.0f;
198         z = v.z;
199         return z;
200     }
201     //Workshop3
202     public static MyVector3 EulerAnglesToDirection(MyVector3 EulerAngles)
203     {
204         MyVector3 rv = new MyVector3(0,0,0);
205 
```

```
206         rv.x = Mathf.Cos(EulerAngles.y) * Mathf.Cos(EulerAngles.x);
207         rv.y = Mathf.Sin(EulerAngles.x);
208         rv.z = Mathf.Cos(EulerAngles.x) * Mathf.Cos(EulerAngles.y);
209 
210         return rv;
211     }
212     public static MyVector3 VectorCrossProduct(MyVector3 A, MyVector3 B)
213     {
214         MyVector3 C = new MyVector3(0, 0, 0);
215 
216         C.x = A.y * B.z - A.z * B.y;
217         C.y = A.z * B.x - A.x * B.z;
218         C.z = A.x * B.y - A.y * B.x;
219 
220         return C;
221     }
222     public static MyVector3 Lerp(MyVector3 A, MyVector3 B, float T)
223     {
224         //Equation for linear Interpolation(LERP)
225         //C = A(1-T) + BT
226         MyVector3 rv = new MyVector3(0, 0, 0);
227         rv.x = A.x * (1.0f - T) + B.x * T;
228         rv.y = A.y * (1.0f - T) + B.y * T;
229         rv.z = A.z * (1.0f - T) + B.z * T;
230 
231         return rv;
232     }
233     public static MyVector3 GetMyVector3(float x, float y, float z)
234     {
235         MyVector3 v = new MyVector3(0, 0, 0);
236         v.x = x;
237         v.y = y;
238         v.z = z;
239 
240         return v;
241     }
242     public static MyVector3 GetAxis(MyQuaternion MyQuat)
243     {
244         MyVector3 rv = new MyVector3(0, 0, 0);
245 
246         rv.x = MyQuat.x;
247         rv.y = MyQuat.y;
248         rv.z = MyQuat.z;
249 
250         return rv;
251     }
252     public MyVector3 PrintStats()
253     {
254         MyVector3 rv = new MyVector3(0, 0, 0);
255 
256         Debug.Log(rv.x + " " + rv.y + " " + rv.z);
257 
258         return rv;
```

```
259     }
260     public static MyVector3 RotateVertexAroundAxis(float Angle,MyVector3
          Axis,MyVector3 Vertex)
261     {
262         MyVector3 rv = (Vertex * Mathf.Cos(Angle)) + ScaleVector(Axis,Dot
            (Vertex, Axis)) * (1 - Mathf.Cos(Angle)) + VectorCrossProduct
            (Axis, Vertex) * Mathf.Sin(Angle);
263
264         return rv;
265     }
266     public static MyVector3 RotateVertex(float angle, MyVector3 v, MyVector3
          N)
267     {
268         MyVector3 NPrime = N * Mathf.Cos(angle) +
269             v * MyVector3.Dot(N, v) *
270             (1.0f - Mathf.Cos(angle)) +
271             VectorCrossProduct(v,N)*Mathf.Sin(angle);
272
273         return NPrime;
274     }
275     public static MyVector3 Right()
276     {
277         MyVector3 rv = new MyVector3(1, 0, 0);
278         return rv;
279     }
280     public static MyVector3 Left()
281     {
282         MyVector3 rv = new MyVector3(-1, 0, 0);
283         return rv;
284     }
285     public static MyVector3 Up()
286     {
287         MyVector3 rv = new MyVector3(0, -1, 0);
288         return rv;
289     }
290     public static MyVector3 Down()
291     {
292         MyVector3 rv = new MyVector3(0, -1, 0);
293         return rv;
294     }
295     public static MyVector3 Forward()
296     {
297         MyVector3 rv = new MyVector3(0, 0, 1);
298         return rv;
299     }
300     public static MyVector3 Backward()
301     {
302         MyVector3 rv = new MyVector3(0, 0, -1);
303         return rv;
304     }
305     //public static MyVector3 RotateVertexAroundAxis(float Angle,MyVector3
          Axis,MyVector3 Vertex)
306     //{
```

```
307     //    MyVector3 rv = (Vertex * Mathf.Cos(Angle)) +VectorDot(Vertex,
          Axis) * Axis * (1 - Mathf.Cos(Angle))
308     //                     + VectorCrossProduct(Axis, Vertex) * Mathf.Sin
          (Angle);
309     //    return rv;
310     //}
311 }
```