

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PhysicsLibrary
6 {
7
8
9     //Displacement Calculations
10    public static float DisplacementCalculation(float Velocity, float Time)
11    {
12        float Displacement = Velocity * Time;
13
14        return Displacement;
15    }
16    //Velocity Calculations
17    public static float VelocityCalculation(float VelocityZero, float Acceleration, float Time)
18    {
19        float Velocity = VelocityZero + (Acceleration * Time);
20        return Velocity;
21    }
22    public static float VelocityCalculation(float Displacement, float Time)
23    {
24        float Velocity = Displacement / Time;
25        return Velocity;
26    }
27    public static MyVector3 VelocityCalculation(MyVector3 VelocityZero, MyVector3 Acceleration, float Time)
28    {
29        MyVector3 Velocity = VelocityZero + (Acceleration * Time);
30        return Velocity;
31    }
32    public static float ForceCalculation(float Mass, float Acceleration)
33    {
34        float Force = Mass * Acceleration;
35        return Force;
36    }
37    public static MyVector3 ForceCalculation(float Mass, MyVector3 Acceleration)
38    {
39        MyVector3 Force = Acceleration * Mass;
40        return Force;
41    }
42    public static MyVector3 MomentumCalculation(float Time, MyVector3 Acceleration)
43    {
44        MyVector3 Force = Acceleration * Time;
45        return Force;
46    }
47
48    //public static MyVector3 SphereBounds(float sphereradius, MyVector3 SphereOffset, MyVector3 Translate, MyVector3 NormalizedRocketVector)
```

```
49 // {
50 //     NormalizedRocketVector = new MyVector3(Translate.x+SphereOffset.x,
51 //     Translate.y + SphereOffset.y, Translate.z +
52 //     SphereOffset.z).NormalizeVector();
53 //     //Checks if the player is within the bounds of the circle
54 //     if (Translate.x <= (sphereradius * NormalizedRocketVector.x) +
55 //     SphereOffset.x && Translate.x >= (-sphereradius *
56 //     NormalizedRocketVector.x) + SphereOffset.x)
57 //     {
58 //         if (Translate.x <= (sphereradius * NormalizedRocketVector.x) +
59 //         SphereOffset.x)
60 //         {
61 //             Translate.x = (sphereradius * NormalizedRocketVector.x) +
62 //             SphereOffset.x;
63 //         }
64 //         else if (Translate.x < 0.0f)
65 //         {
66 //             Translate.x = (-sphereradius * NormalizedRocketVector.x) +
67 //             SphereOffset.x;
68 //         }
69 //     }
70 //     if (Translate.y <= (sphereradius * NormalizedRocketVector.y) +
71 //     SphereOffset.y && Translate.y >= (-sphereradius *
72 //     NormalizedRocketVector.y) + SphereOffset.y)
73 //     {
74 //         if (Translate.y <= (sphereradius * NormalizedRocketVector.y) +
75 //         SphereOffset.y)
76 //         {
77 //             Translate.y = (sphereradius * NormalizedRocketVector.y) +
78 //             SphereOffset.y;
79 //         }
80 //         else if (Translate.y < 0.0f)
81 //         {
82 //             Translate.y = (-sphereradius * NormalizedRocketVector.y) +
83 //             SphereOffset.y;
84 //         }
85 //     }
86 //     if (Translate.z <= (sphereradius * NormalizedRocketVector.z) +
87 //     SphereOffset.z && Translate.z >= (-sphereradius *
88 //     NormalizedRocketVector.z) + SphereOffset.z)
89 //     {
90 //         Translate.z = (sphereradius * NormalizedRocketVector.z) +
91 //         SphereOffset.z;
92 //     }
93 //     Debug.Log(NormalizedRocketVector.ToUnityVector());
94 //     // Debug.Log(Thruster.IsGrounded);
95 //     return Translate;
96 // }
```

```
87
88     public static MyVector3 SphereBounds(float sphereradius, MyVector3 SphereOffset, MyVector3 Translate, MyVector3 NormalizedRocketVector)
89     {
90
91         NormalizedRocketVector = new MyVector3(Translate.x + SphereOffset.x, Translate.y + SphereOffset.y, Translate.z + SphereOffset.z).NormalizeVector();
92         MyVector3 RadiusVector = new MyVector3((sphereradius * NormalizedRocketVector.x)+SphereOffset.x, (sphereradius * NormalizedRocketVector.y) + SphereOffset.y, (sphereradius * NormalizedRocketVector.z) + SphereOffset.z);
93
94         //Checks if the player is within the bounds of the circle
95
96
97         if (Translate.x < 0)
98         {
99             if ((Translate.x >= RadiusVector.x))
100             {
101                 Translate.x = RadiusVector.x;
102             }
103         }
104         else
105         {
106             if ((Translate.x <= RadiusVector.x))
107             {
108                 Translate.x = RadiusVector.x;
109             }
110         }
111         if (Translate.y < 0)
112         {
113             if ((Translate.y>= RadiusVector.y))
114             {
115                 Translate.y = RadiusVector.y;
116             }
117         }
118         else
119         {
120             if ((Translate.y<= RadiusVector.y))
121             {
122                 Translate.y = RadiusVector.y;
123             }
124         }
125         if (Translate.z < 0)
126         {
127             if ((Translate.z >= RadiusVector.z))
128             {
129                 Translate.z = RadiusVector.z;
130             }
131         }
132         else
133         {
```

```

134         if ((Translate.z <= RadiusVector.z))
135         {
136             Translate.z = RadiusVector.z;
137         }
138     }
139
140
141     //Debug.Log(RadiusVector.ToUnityVector());
142     Debug.Log(RadiusVector.ToUnityVector());
143     return Translate;
144
145
146 }
147
148 public static MyVector3 SphereBounds2(float sphereradius, MyVector3
149 SphereOffset, MyVector3 Translate, MyVector3 NormalizedRocketVector)
150 {
151     NormalizedRocketVector = new MyVector3(Translate.x + SphereOffset.x,
152     Translate.y + SphereOffset.y, Translate.z +
153     SphereOffset.z).NormalizeVector();
154     //Checks if the player is within the bounds of the circle
155     float AngleToRocket = MyVector2.VectorsToRadians(new MyVector2
156     (Translate.x-SphereOffset.x, Translate.y - SphereOffset.y));
157     if (Translate.x <= (SphereOffset.x) +Mathf.Cos(AngleToRocket)
158     *sphereradius && Translate.x >= (SphereOffset.x) + Mathf.Cos
159     (AngleToRocket) * -sphereradius)
160     {
161         Translate.x = (SphereOffset.x) + Mathf.Cos(AngleToRocket) *
162         -sphereradius;
163     }
164     if (Translate.y <=(SphereOffset.y) + Mathf.Sin(AngleToRocket) *
165     sphereradius && Translate.y >= (SphereOffset.y) + Mathf.Sin
166     (AngleToRocket) * -sphereradius)
167     {
168         Translate.y = (SphereOffset.y) + Mathf.Sin(AngleToRocket) *
169         -sphereradius;
170     }
171     //if (Translate.z <= (SphereOffset.x) + Mathf.Cos(AngleToRocket) *
172     sphereradius && Translate.x >= (SphereOffset.z) + Mathf.Cos
173     (AngleToRocket) * -sphereradius)
174     //{
175         Translate.z = (SphereOffset.x) + Mathf.Cos(AngleToRocket) * -
176         sphereradius;
177     //}
178     Debug.Log(" " + Mathf.Cos(AngleToRocket));
179
180
181     MyVector3 combo = NormalizedRocketVector;
182     //Debug.Log(combo.ToUnityVector());
183
184     // Debug.Log(Thruster.IsGrounded);

```

```
174     return Translate;
175 }
176 public static MyVector3 SphereBounds4(float sphereradius, MyVector3
177     SphereOffset, MyVector3 Translate, MyVector3 NormalizedRocketVector)
178 {
179     //Checks if the player is within the bounds of the circle
180     float AngleToRocket = MyVector2.VectorsToRadians(new MyVector2
181         (Translate.x - SphereOffset.x, Translate.y - SphereOffset.y));
182
183     if (Translate.x < 0)
184     {
185         if ((Translate.x <= (SphereOffset.x) + Mathf.Cos(AngleToRocket)
186             * sphereradius))
187         {
188             Translate.x = (SphereOffset.x) + Mathf.Cos(AngleToRocket) *
189                 sphereradius;
190         }
191     }
192     else
193     {
194         if ((Translate.x >= (SphereOffset.x) + Mathf.Cos(AngleToRocket)
195             * sphereradius))
196         {
197             Translate.x = (SphereOffset.x) + Mathf.Cos(AngleToRocket) *
198                 sphereradius;
199         }
200     }
201
202     if (Translate.y < 0)
203     {
204         if ((Translate.y <= (SphereOffset.y) + Mathf.Sin(AngleToRocket)
205             * sphereradius))
206         {
207             Translate.y = (SphereOffset.y) + Mathf.Sin(AngleToRocket) *
208                 -sphereradius;
209         }
210     }
211     else
212     {
213         if ((Translate.y <= (SphereOffset.y) + Mathf.Sin(AngleToRocket)
214             * sphereradius))
215         {
216             Translate.y = (SphereOffset.y) + Mathf.Sin(AngleToRocket) *
217                 -sphereradius;
218         }
219     }
220
221     if (Translate.z < 0)
222     {
223         if ((Translate.z <= (SphereOffset.x) + Mathf.Cos(AngleToRocket)
224             * sphereradius))
225         {
226             Translate.z = (SphereOffset.x) + Mathf.Cos(AngleToRocket) *
227                 -sphereradius;
```

```
215     }
216   }
217   else
218   {
219     if ((Translate.z <= (SphereOffset.x) + Mathf.Cos(AngleToRocket) *
220         * sphereradius))
221     {
222       Translate.z = (SphereOffset.x) + Mathf.Cos(AngleToRocket) *
223         -sphereradius;
224     }
225   }
226   return Translate;
227 }
```