```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PhysicsLibrary
{


    //Displacement Calculations
    public static float DisplacementCalculation(float Velocity, float Time)
    {
        float Displacement = Velocity * Time;

        return Displacement;
    }
    //Velocity Calculations
    public static float VelocityCalculation(float VelocityZero,float
      Acceleration,float Time)
    {
        float Velocity = VelocityZero + (Acceleration * Time);
        return Velocity;
    }
    public static float VelocityCalculation(float Displacement, float Time)
    {
        float Velocity = Displacement / Time;
        return Velocity;
    }
    public static MyVector3 VelocityCalculation(MyVector3 VelocityZero,
      MyVector3 Acceleration, float Time)
    {
        MyVector3 Velocity = VelocityZero + (Acceleration * Time);
        return Velocity;
    }
    public static float ForceCalculation(float Mass, float Acceleration)
    {
        float Force = Mass * Acceleration;
        return Force;
    }
    public static MyVector3 ForceCalculation(float Mass, MyVector3
      Acceleration)
    {
        MyVector3 Force = Acceleration * Mass;
        return Force;
    }
    public static MyVector3 MomentumCalculation(float Time, MyVector3
      Acceleration)
    {
        MyVector3 Force = Acceleration * Time;
        return Force;
    }

    //public static MyVector3 SphereBounds(float sphereradius, MyVector3
      SphereOffset, MyVector3 Translate, MyVector3 NormalizedRocketVector)
```

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

```
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
      SphereOffset, MyVector3 Translate, MyVector3 NormalizedRocketVector)
149     {
150         NormalizedRocketVector = new MyVector3(Translate.x + SphereOffset.x,
           Translate.y + SphereOffset.y, Translate.z +
          SphereOffset.z).NormalizeVector();
151         //Checks if the player is within the bounds of the circle
152         float AngleToRocket = MyVector2.VectorsToRadians(new MyVector2
          (Translate.x-SphereOffset.x, Translate.y - SphereOffset.y));
153         if (Translate.x <= (SphereOffset.x) +Mathf.Cos(AngleToRocket)
          *sphereradius && Translate.x >= (SphereOffset.x) + Mathf.Cos
          (AngleToRocket) * -sphereradius)
154         {
155                 Translate.x = (SphereOffset.x) + Mathf.Cos(AngleToRocket) *
                  -sphereradius;
156         }
157 
158         if (Translate.y <=(SphereOffset.y) + Mathf.Sin(AngleToRocket) *
          sphereradius && Translate.y >= (SphereOffset.y) + Mathf.Sin
          (AngleToRocket) * -sphereradius)
159         {
160                 Translate.y = (SphereOffset.y) + Mathf.Sin(AngleToRocket) *
                  -sphereradius;
161         }
162         //if (Translate.z <= (SphereOffset.x) + Mathf.Cos(AngleToRocket) *
          sphereradius && Translate.x >= (SphereOffset.z) + Mathf.Cos
          (AngleToRocket) * -sphereradius)
163         //{
164         //    Translate.z = (SphereOffset.x) + Mathf.Cos(AngleToRocket) * -
          sphereradius;
165         //}
166         Debug.Log("  " + Mathf.Cos(AngleToRocket));
167 
168 
169 
170         MyVector3 combo = NormalizedRocketVector;
171         //Debug.Log(combo.ToUnityVector());
172 
173        // Debug.Log(Thruster.IsGrounded);
```

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

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