IGB283 Assignment 1 – Statement of Completion

Student 1 Name: Findabhair Doolan

Student 1 ID: [N11957557]

Student 2 Name: Alex O'Donnell
Student 2 ID: [N12015148]

Demonstration Video

The following is the link to the demonstration video for IGB283 Assignment1 main project.

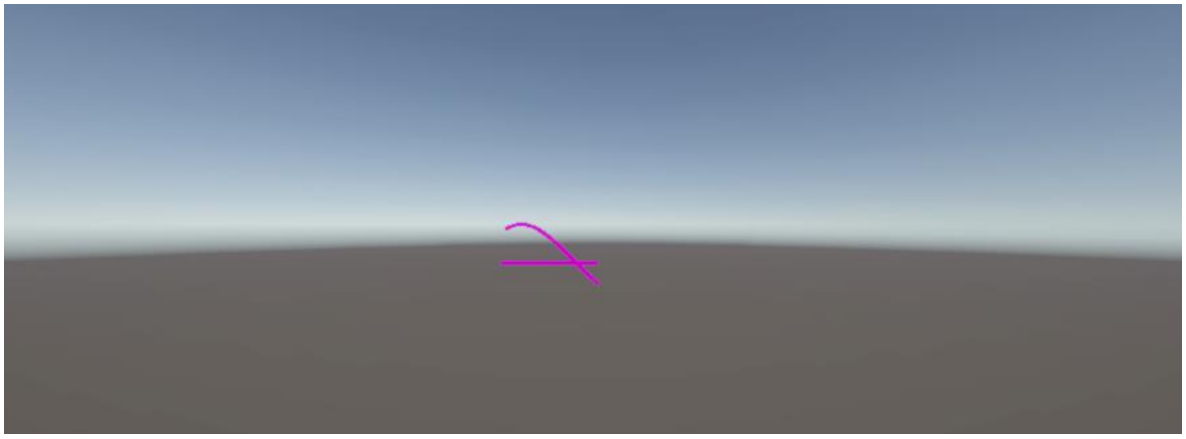
https://youtu.be/hS-GDC-eMLI

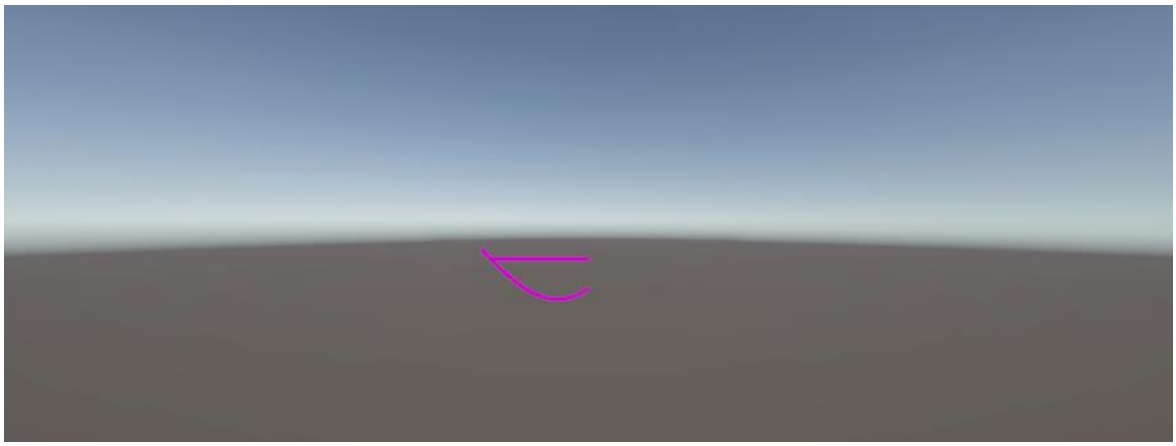
Workshop Activities (Student 1)

Unity Version: 6000.2.2f1

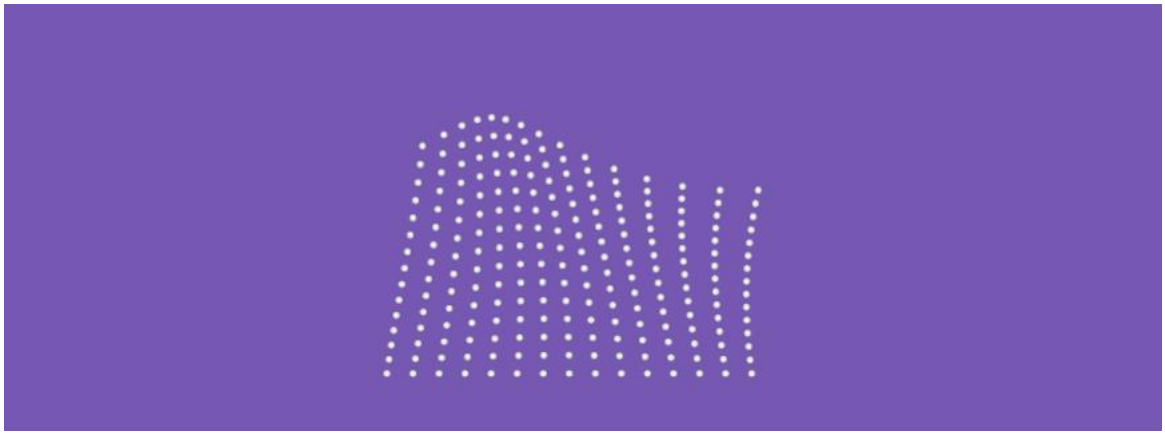
Workshop 2:

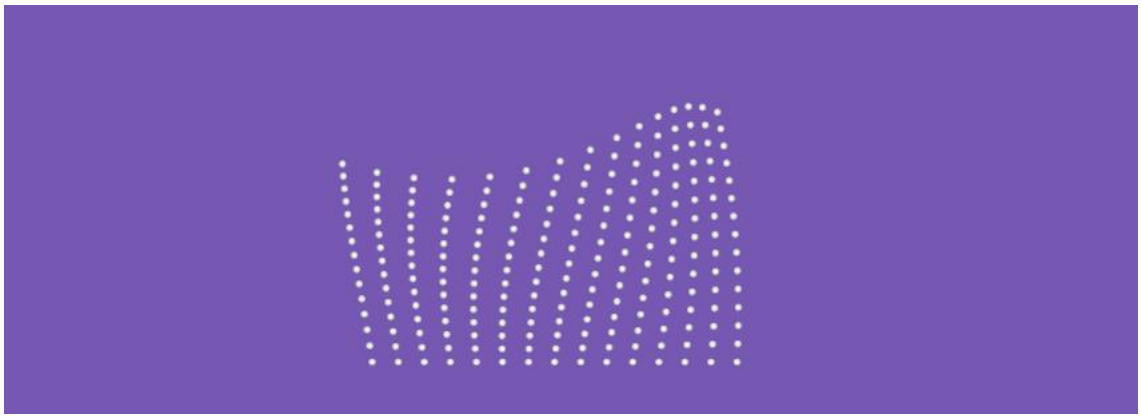
Activity 1 – Spiral:





Activity 2 – Water Wave:





Workshop 3:

Activity 1 – Find the Normal and Equation of a plane given three points:

$P(1,3,2)$ $Q(3,-1,6)$
$R(5,2,0)$

$\overrightarrow{PR} = R - P$

$= \begin{bmatrix} 5-1 \\ 2-3 \\ 0-2 \end{bmatrix}$

$= \begin{bmatrix} 4 \\ -1 \\ -2 \end{bmatrix}$

$\therefore \overrightarrow{PQ} \times \overrightarrow{PR} = n$

$\therefore n = \begin{bmatrix} 2 \\ -4 \\ 4 \end{bmatrix} \times \begin{bmatrix} 4 \\ -1 \\ -2 \end{bmatrix}$

$\overrightarrow{PQ} = Q - P$

$= \begin{bmatrix} 3-1 \\ -1-3 \\ 6-2 \end{bmatrix}$

$= \begin{bmatrix} 2 \\ -4 \\ 4 \end{bmatrix}$

$= \begin{bmatrix} -4 \times -2 - 4 \times -1 \\ 4 \times 4 - 2 \times -2 \\ 2 \times -1 - -4 \times 4 \end{bmatrix}$

$= \begin{bmatrix} 8+4 \\ 16+4 \\ -2+16 \end{bmatrix}$

$= \begin{bmatrix} 12 \\ 20 \\ 14 \end{bmatrix}$

$\therefore$ Plane equation ↓

$0 = n_x(x - p_x) + n_y(y - p_y) + n_z(z - p_z)$

$12(x-1) + 20(y-3) + 14(z-2) = 0$

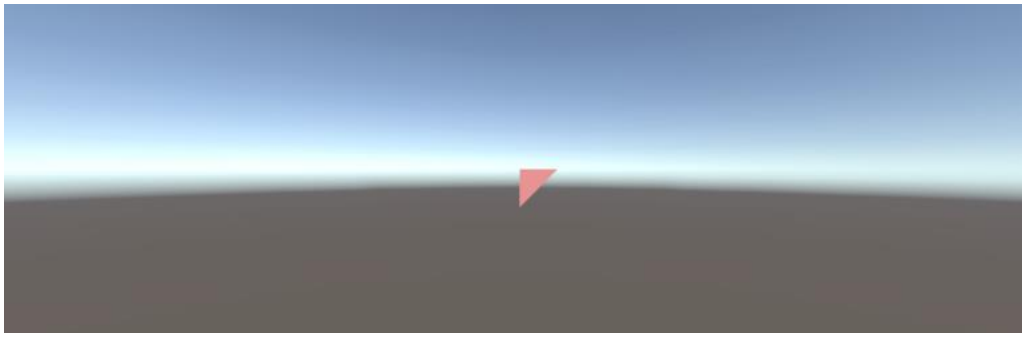
$12x - 12 + 20y - 60 + 14z - 28 = 0$

$12x + 20y + 14z - 100 = 0$

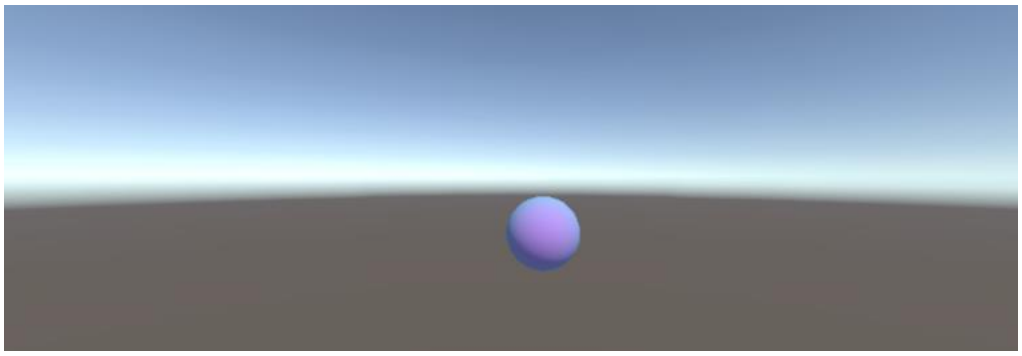
$12x + 20y + 14z = 100$

$6x + 10y + 7z = 50$

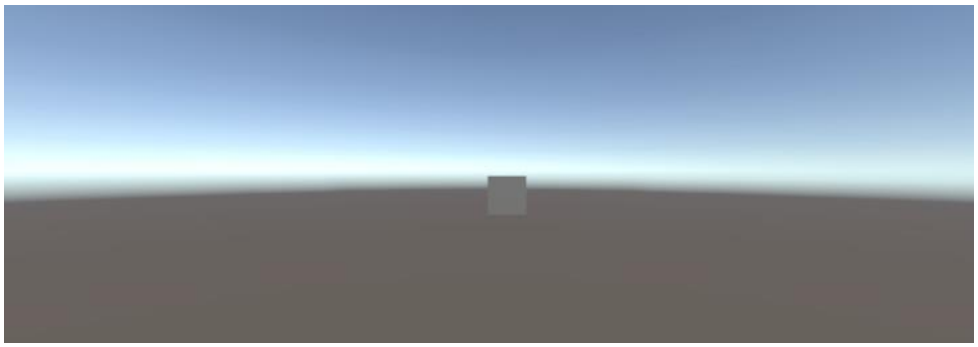
Activity 2 – Mesh Triangle:

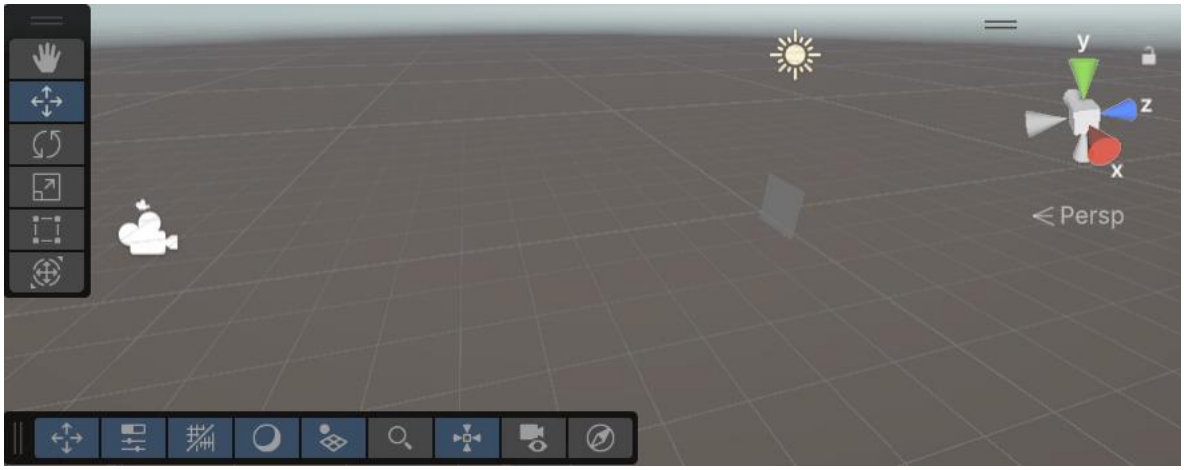


Activity 3 – Coloured Sphere:



Activity 4 – backface culling:





Workshop 4:

Activity 1 – Matrix operations:

$c_1 = 2$

$$M_1 = \begin{bmatrix} 3 & 12 & 4 \\ 5 & 6 & 8 \\ 1 & 0 & 2 \end{bmatrix} \quad M_2 = \begin{bmatrix} 7 & 3 & 8 \\ 11 & 9 & 5 \\ 6 & 8 & 4 \end{bmatrix}$$

2) $M_2^T = \begin{bmatrix} 7 & 11 & 6 \\ 3 & 9 & 8 \\ 8 & 5 & 4 \end{bmatrix}$

3) $M_2 \times c_1 = \begin{bmatrix} 7 & 3 & 8 \\ 11 & 9 & 5 \\ 6 & 8 & 4 \end{bmatrix} \times 2$

$$= \begin{bmatrix} 7\times2 & 3\times2 & 8\times2 \\ 11\times2 & 9\times2 & 5\times2 \\ 6\times2 & 8\times2 & 4\times2 \end{bmatrix}$$

$$= \begin{bmatrix} 14 & 6 & 16 \\ 22 & 18 & 10 \\ 12 & 16 & 8 \end{bmatrix}$$

4) $M_2 \times M_1$

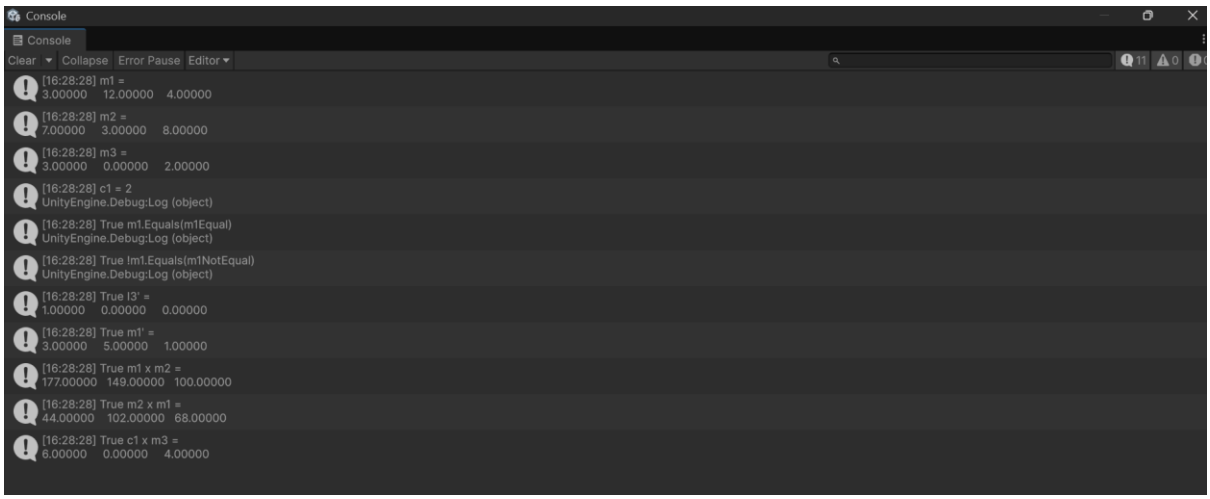
$$= \begin{bmatrix} 7 & 3 & 8 \\ 11 & 9 & 5 \\ 6 & 8 & 4 \end{bmatrix} \times \begin{bmatrix} 3 & 12 & 4 \\ 5 & 6 & 8 \\ 1 & 0 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 7\times3+3\times5+8\times1 & 7\times12+3\times6+8\times0 & 7\times4+3\times8+8\times2 \\ 11\times3+9\times5+5\times1 & 11\times12+9\times6+5\times0 & 11\times4+9\times8+5\times2 \\ 6\times3+8\times5+4\times1 & 6\times12+8\times6+4\times0 & 6\times4+8\times8+4\times2 \end{bmatrix}$$

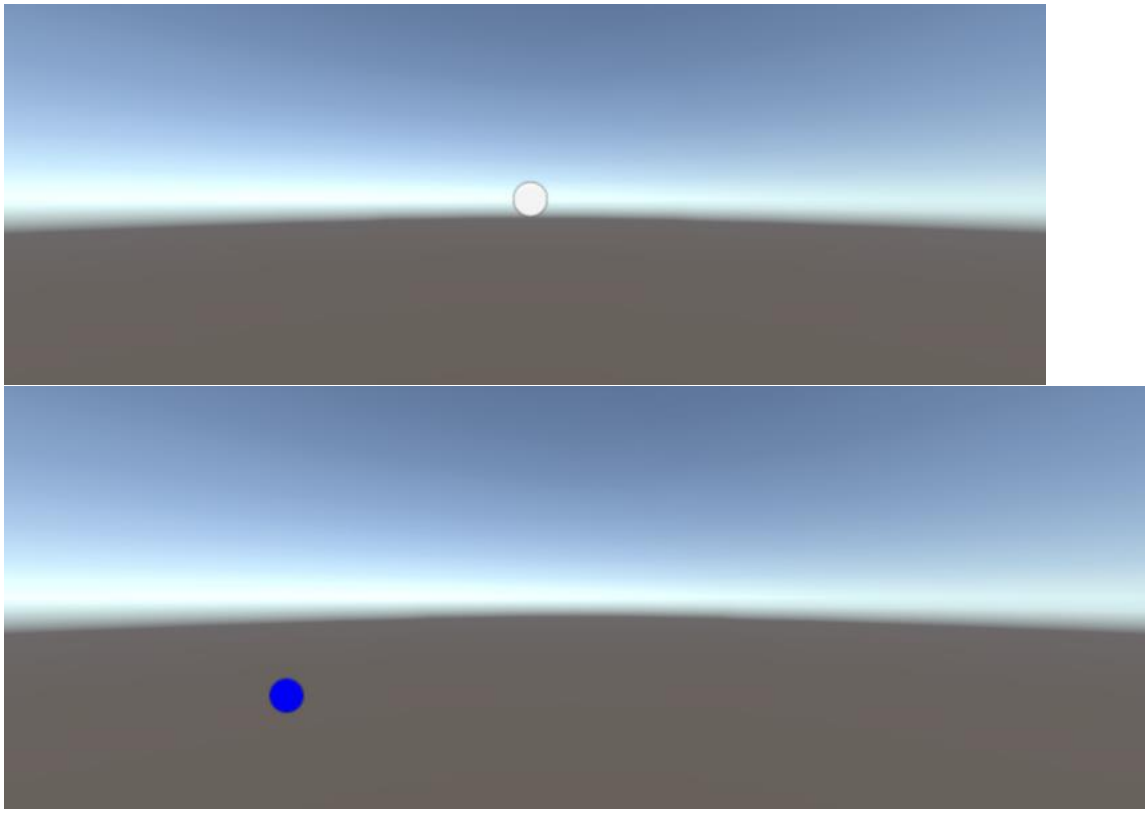
$$= \begin{bmatrix} 21+15+8 & 84+18+0 & 28+24+16 \\ 33+45+5 & 132+54+0 & 44+72+10 \\ 18+40+4 & 72+48+0 & 24+64+8 \end{bmatrix}$$

$$= \begin{bmatrix} 44 & 102 & 68 \\ 83 & 186 & 126 \\ 62 & 120 & 96 \end{bmatrix}$$

Activity 2

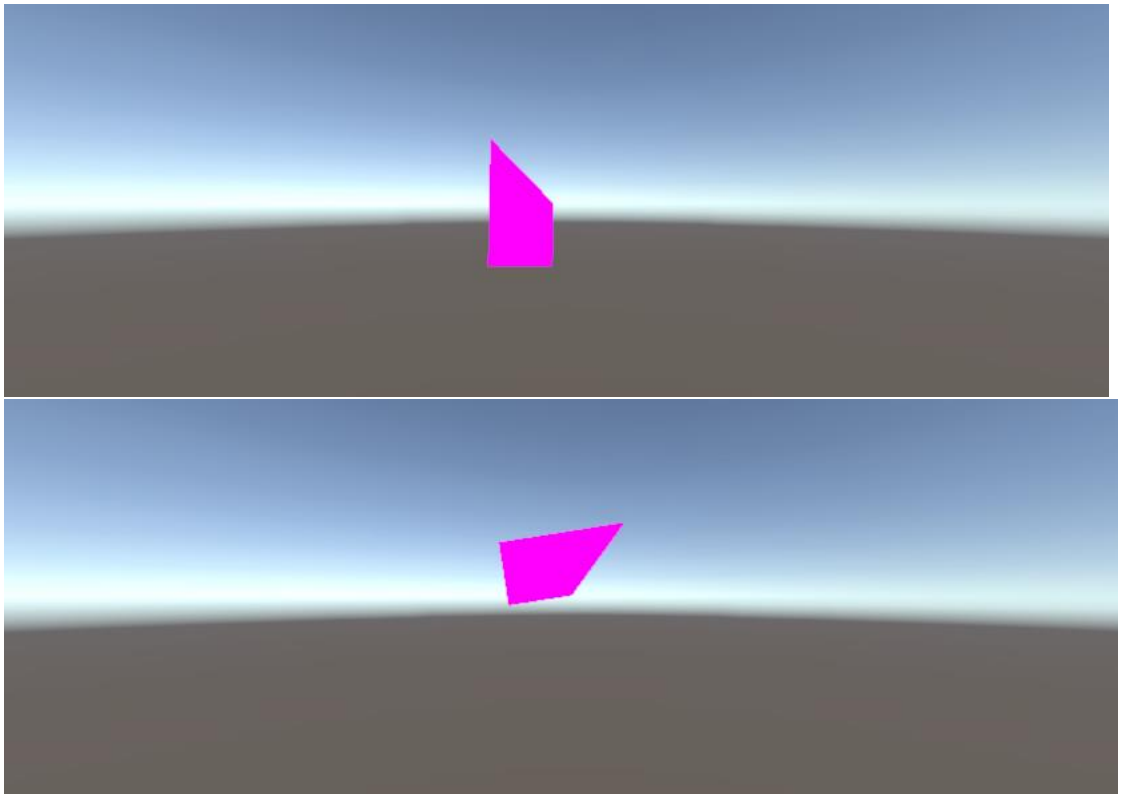


Activity 3 – Interaction (Change object colours and drag them):

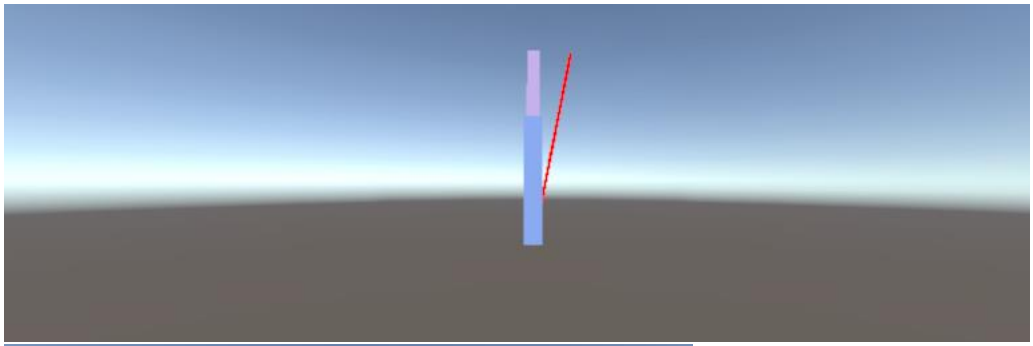


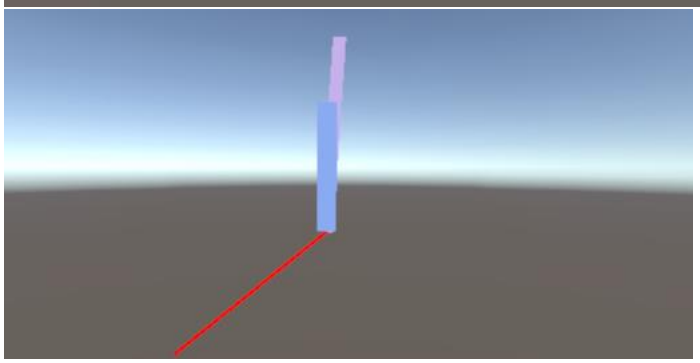
Workshop 5:

Activity 1 – Rotating Triangle:



Activity 2 – Clockface:





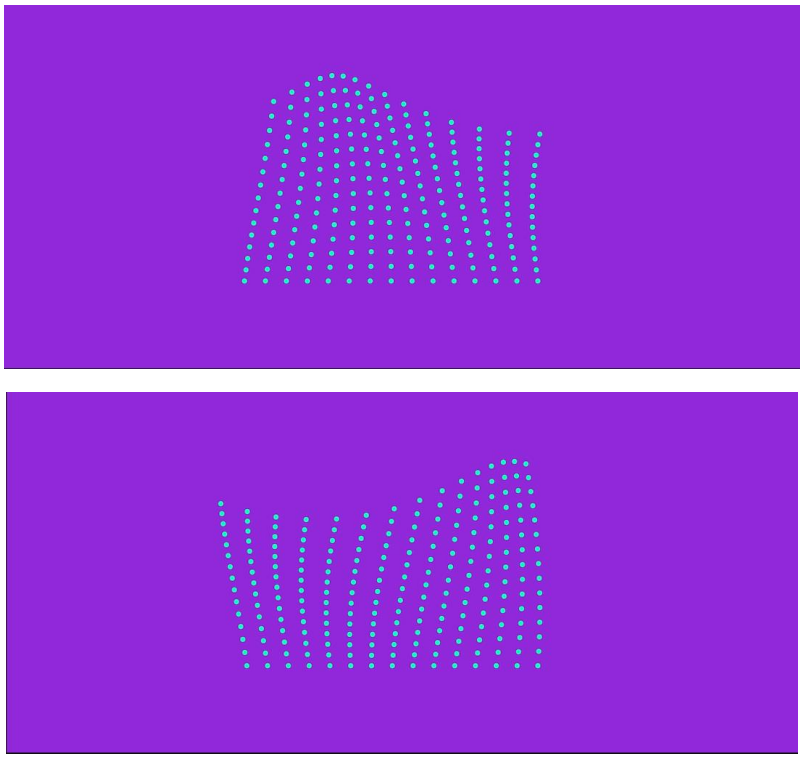
Workshop Activities (Student 2)

Unity Version: 6000.2.2f1

Workshop 2:

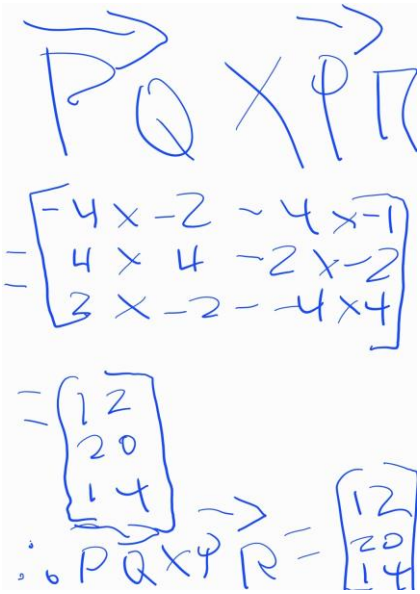
Activity 1 – Spiral:

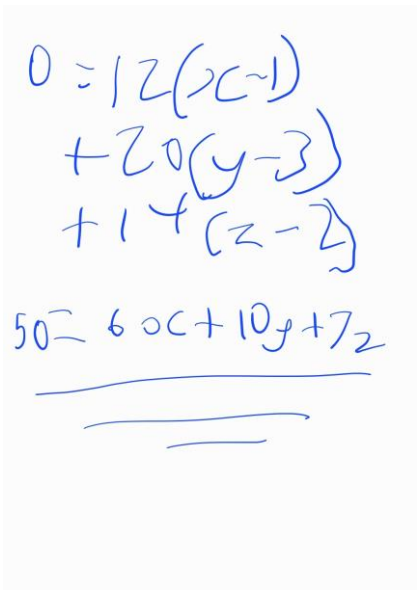
Activity 2 – Water Wave:



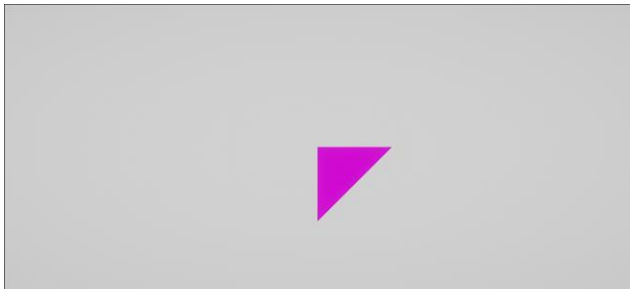
Workshop 3:

Activity 1 – Find the Normal and Equation of a plane given three points:

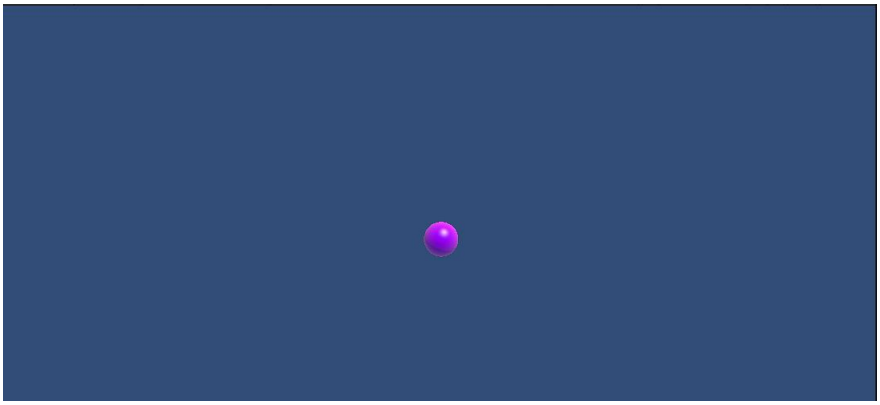




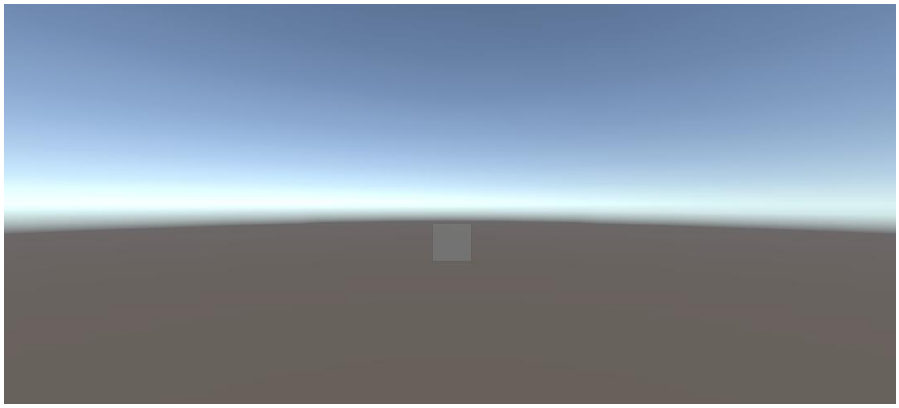
Activity 2 – Mesh Triangle:

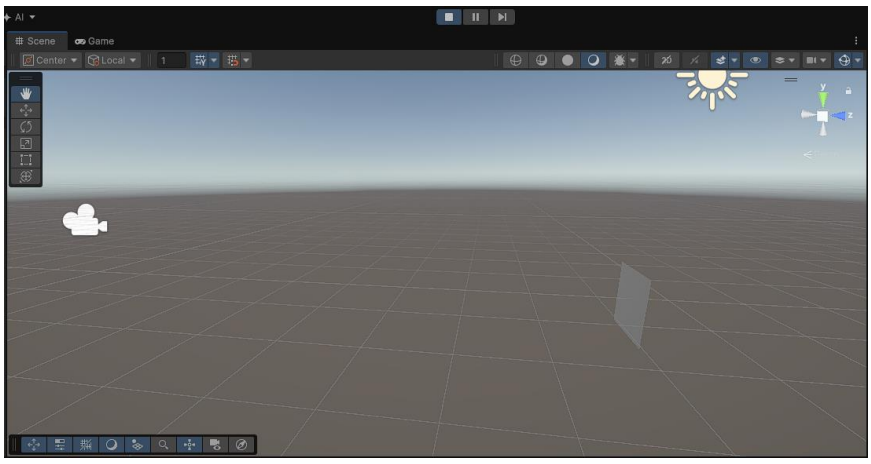


Activity 3 – Coloured Sphere:



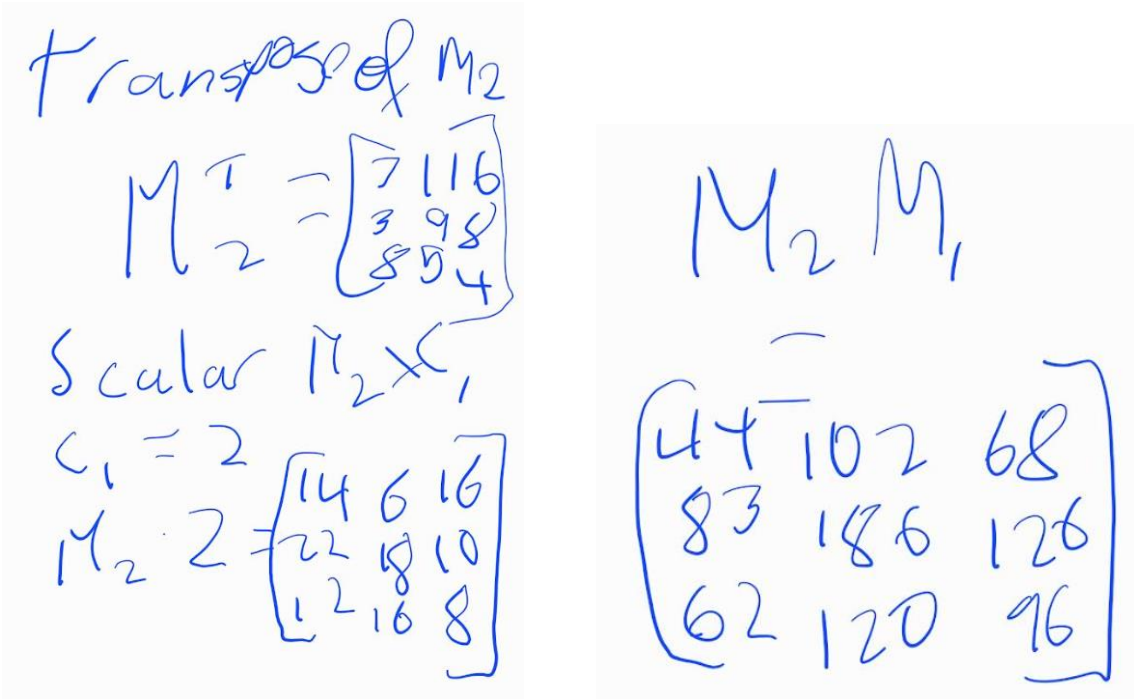
Activity 4 – Backface Culling:



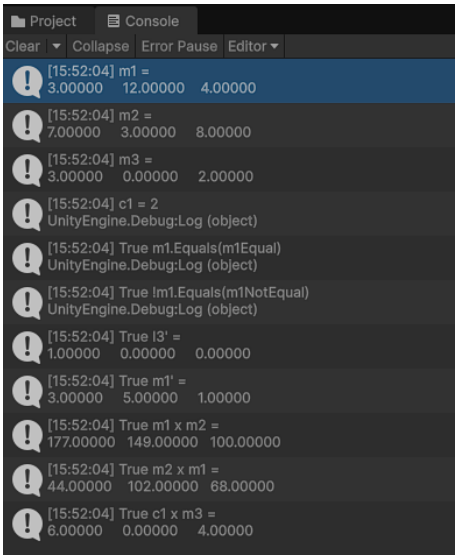


Workshop 4:

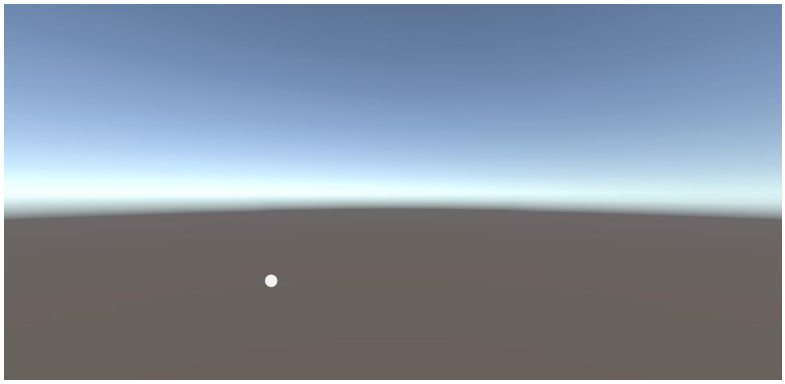
Activity 1 – Matrix operations:

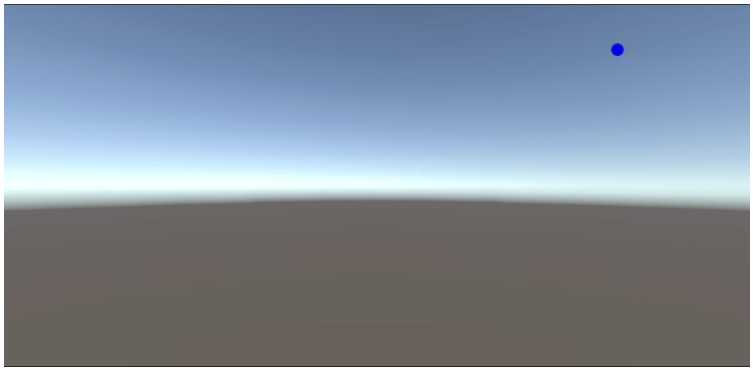


Activity 2



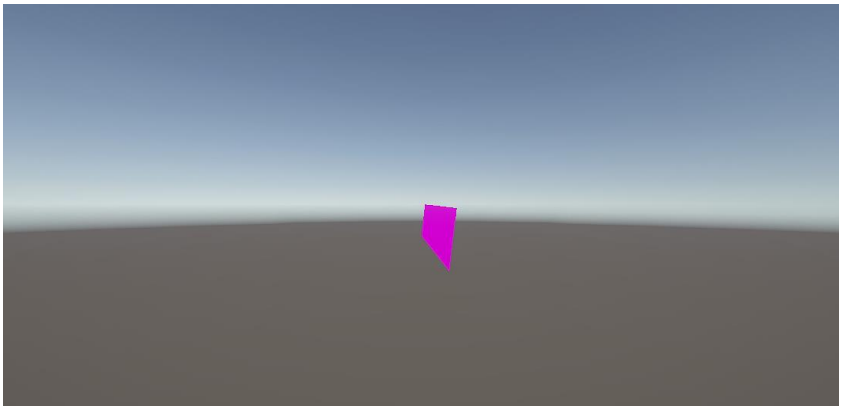
Activity 3 – Interaction (Change object colours and drag them):

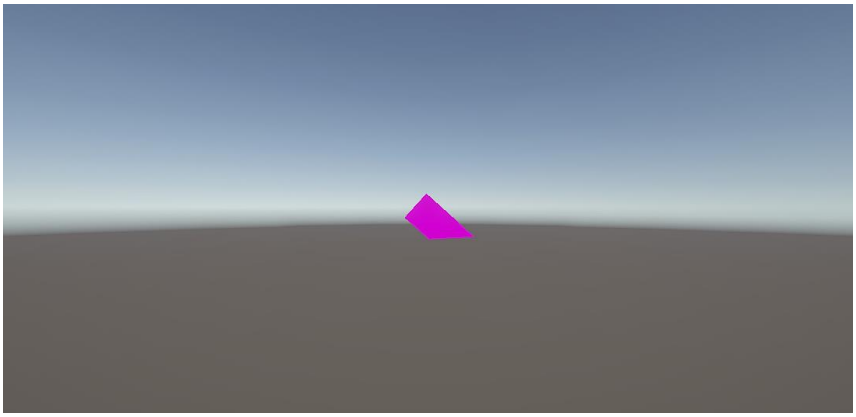




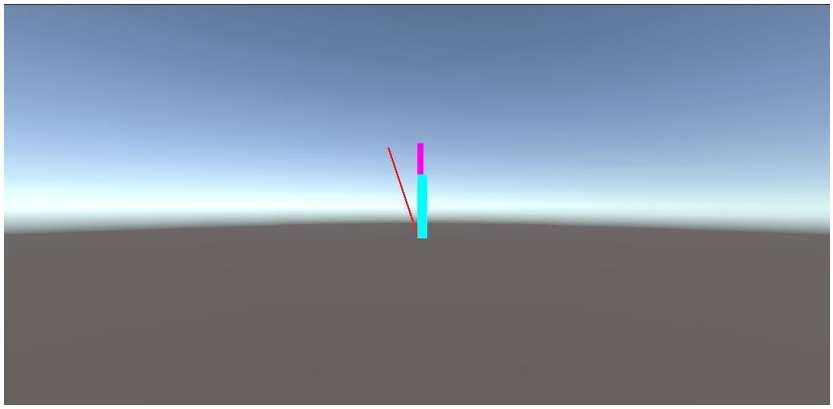
Workshop 5:

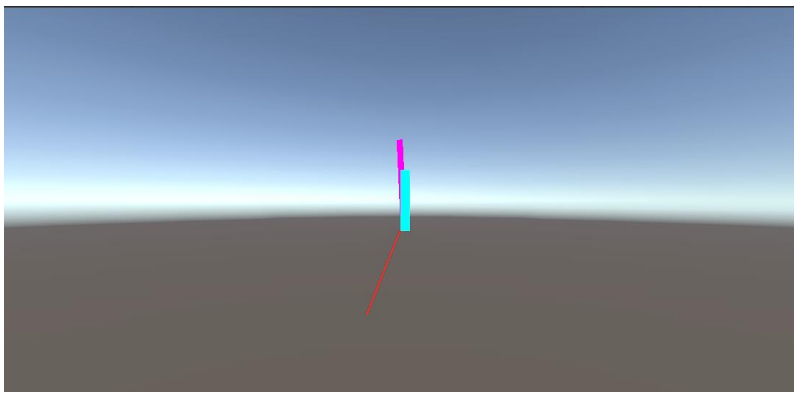
Activity 1 – Rotating Triangle:





Activity 2 – Clockface:





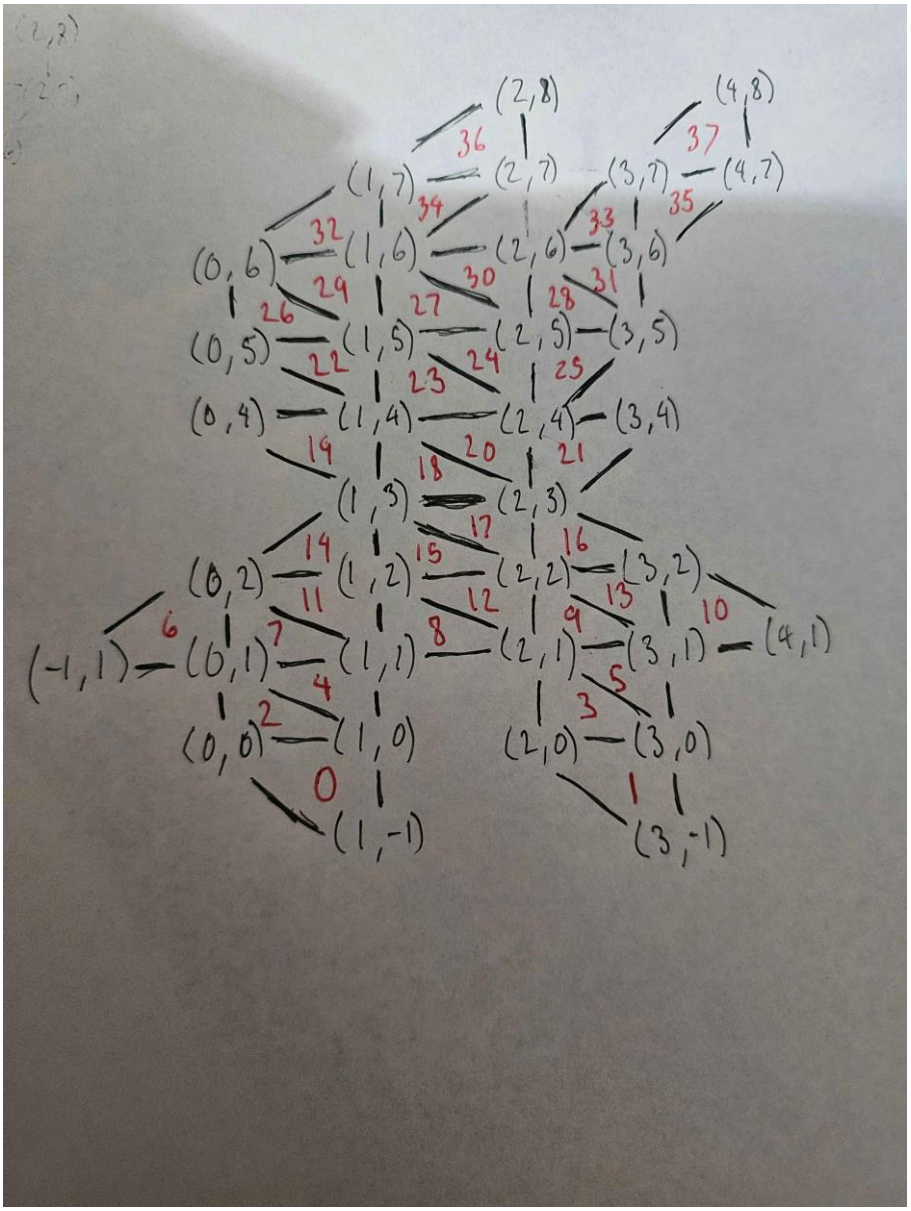
IGB283 Assignment Report

Program Design:

Task 2:

A 2D graphical object is created and its vertices are added to a mesh of triangles.

The following mesh contains 36 vertices, and 38 triangles.
The original mesh coordinates are as listed in the image however in the coded vertex list they were downscaled by 1/8 to fit within the bounds of [-1,1] and then were mathematically centred on (0,0) using code that was applied to all vertices in the list.



```
using UnityEngine;

public class Triangle : MonoBehaviour
{
    [SerializeField] private Material material;

    void Start()
    {
        Mesh mesh = gameObject.AddComponent<MeshFilter>().mesh;
        gameObject.AddComponent<MeshRenderer>().material = material;
        mesh.Clear();

        // Define all vertices - calculated by using original vertex map then downscaling it to fit in the [-1,1]
        mesh.vertices = new Vector3[]
        {
            new Vector3(0.125f, -0.125f, 0),
            new Vector3(0.375f, -0.125f, 0),
            new Vector3(0f, 0f, 0),
            new Vector3(0.125f, 0f, 0),
            new Vector3(0.25f, 0f, 0),
            new Vector3(0.375f, 0f, 0),
            new Vector3(-0.125f, 0.125f, 0),
            new Vector3(0f, 0.125f, 0),
            new Vector3(0.125f, 0.125f, 0),
            new Vector3(0.25f, 0.125f, 0),
            new Vector3(0.375f, 0.125f, 0),
            new Vector3(0.5f, 0.125f, 0),
            new Vector3(0f, 0.25f, 0),
            new Vector3(0.125f, 0.25f, 0),
            new Vector3(0.25f, 0.25f, 0),
            new Vector3(0.375f, 0.25f, 0),
            new Vector3(0.125f, 0.375f, 0),
            new Vector3(0.25f, 0.375f, 0),
            new Vector3(0f, 0.5f, 0),
            new Vector3(0.125f, 0.5f, 0),
            new Vector3(0.25f, 0.5f, 0),
            new Vector3(0.375f, 0.5f, 0),
            new Vector3(0f, 0.625f, 0),
            new Vector3(0.125f, 0.625f, 0),
            new Vector3(0.25f, 0.625f, 0),
            new Vector3(0.375f, 0.625f, 0),
            new Vector3(0f, 0.75f, 0),
            new Vector3(0.125f, 0.75f, 0),
            new Vector3(0.25f, 0.75f, 0),
            new Vector3(0.375f, 0.75f, 0),
            new Vector3(0.125f, 0.875f, 0),
            new Vector3(0.25f, 0.875f, 0),
            new Vector3(0.375f, 0.875f, 0),
            new Vector3(0.5f, 0.875f, 0),
            new Vector3(0.25f, 1.0f, 0),
            new Vector3(0.5f, 1.0f, 0)
        };

        //get copy of mesh
        Vector3[] vertCopy = mesh.vertices;

        //Get bounds
        float minX = float.MaxValue, maxX = float.MinValue;
        float minY = float.MaxValue, maxY = float.MinValue;

        foreach (var v in vertCopy)
        {
            if (v.x < minX) minX = v.x;
            if (v.x > maxX) maxX = v.x;
            if (v.y < minY) minY = v.y;
            if (v.y > maxY) maxY = v.y;
        }

        //Get center
        Vector3 center = new Vector3((minX + maxX) / 2f, (minY + maxY) / 2f, 0f);
```

```
71
72          //Recenter vertices
73          for (int i = 0; i < vertCopy.Length; i++)
74              vertCopy[i] -= center;
75
76          //Update mesh
77          mesh.vertices = vertCopy;
78          mesh.RecalculateBounds();
79
80
81
82          // Default vertices colour
83          Color black = new Color(0.0f,0.0f,0.0f,1f);
84          Color red = new Color(0.5f, 0.0f, 0.0f, 1f);
85          Color white = new Color(1f, 1f, 1f, 1f);
86          mesh.colors = new Color[]
87          {
88              black,
89              black,
90              black,
91              black,
92              black,
93              black,
94              red,
95              black,
96              black,
97              black,
98              black,
99              red,
100             red,
101             red,
102             red,
103             red,
104             red,
105             red,
```

```
106                 red,
107                 red,
108                 red,
109                 red,
110                 white,
111                 white,
112                 white,
113                 white,
114                 white,
115                 white,
116                 white,
117                 white,
118                 white,
119                 white,
120                 white,
121                 white,
122                 white,
123                 white
124             };
125
126             // Define Triangles
127             mesh.triangles = new int[]
128             {
129                 0,2,3,
130                 1,4,5,
131                 3,2,7,
132                 5,4,9,
133                 3,7,8,
134                 5,9,10,
135                 7,6,12,
136                 8,7,12,
137                 9,8,13,
138                 10,9,14,
139                 11,10,15,
140                 8,12,13,
141                 9,13,14,
142                 10,14,15,
143                 13,12,16,
144                 14,13,16,
145                 15,14,17,
146                 17,14,16,
147                 17,16,19,
148                 19,16,18,
149                 20,17,19,
150                 21,17,20,
151                 23,19,22,
152                 20,19,23,
153                 20,23,24,
154                 25,20,24,
155                 23,22,26,
156                 24,23,27,
157                 25,24,28,
158                 27,23,26,
159                 28,24,27,
160                 29,25,28,
161                 27,26,30,
162                 29,28,32,
163                 31,27,30,
164                 33,29,32,
165                 31,30,34,
166                 33,32,35,
167             };
168
169         }
170     }
```

Custom Transformation methods were added using matrices including:

Translation:

```
14          //translation
15          public static IGB283Transform Translation(float x, float y, float z)
16          {
17              IGB283Transform translated = new IGB283Transform();
18              translated.m[0, 3] = x;
19              translated.m[1, 3] = y;
20              translated.m[2, 3] = z;
21              return translated;
22          }
```

Rotation:

```
//next 3 are roations on each axis
public static IGB283Transform RotationZ(float degrees)
{
    float rad = degrees * Mathf.Deg2Rad;
    IGB283Transform rotate = new IGB283Transform();
    rotate.m[0, 0] = Mathf.Cos(rad);
    rotate.m[0, 1] = -Mathf.Sin(rad);
    rotate.m[1, 0] = Mathf.Sin(rad);
    rotate.m[1, 1] = Mathf.Cos(rad);
    return rotate;
}

public static IGB283Transform RotationX(float degrees)
{
    float rad = degrees * Mathf.Deg2Rad;
    IGB283Transform rotate = new IGB283Transform();
    rotate.m[1, 1] = Mathf.Cos(rad);
    rotate.m[1, 2] = -Mathf.Sin(rad);
    rotate.m[2, 1] = Mathf.Sin(rad);
    rotate.m[2, 2] = Mathf.Cos(rad);
    return rotate;
}

public static IGB283Transform RotationY(float degrees)
{
    float rad = degrees * Mathf.Deg2Rad;
    IGB283Transform rotate = new IGB283Transform();
    rotate.m[0, 0] = Mathf.Cos(rad);
    rotate.m[0, 2] = Mathf.Sin(rad);
    rotate.m[2, 0] = -Mathf.Sin(rad);
    rotate.m[2, 2] = Mathf.Cos(rad);
    return rotate;
}
```

And Scaling:

```
//scaling
public static IGB283Transform Scaling(float x, float y, float z)
{
    IGB283Transform transform = new IGB283Transform();
    transform.m[0, 0] = x;
    transform.m[1, 1] = y;
    transform.m[2, 2] = z;
    return transform;
}
```

The object continually rotates and translates between two points.

Function handling continual rotation/translation/scaling of the object:

```
229     //One function to change Hornet's size based on x position while also moving Hornet and rotating her
230     private void changeTransformScale()
231     {
232         Mesh mesh = GetComponent<MeshFilter>().mesh;
233         Vector3[] verts = new Vector3[originalVertices.Length];
234
235         //calculate the pivot
236         Vector3 pivot = Vector3.zero;
237         for (int i = 0; i < originalVertices.Length; i++)
238             pivot += originalVertices[i];
239         pivot /= Mathf.Max(1, originalVertices.Length);
240
241         //Determine scaling based on x position
242         float minX = PointA.position.x;
243         float maxX = PointB.position.x;
244         float t = Mathf.InverseLerp(minX, maxX, position.x);
245         float scale = Mathf.Lerp(0.5f, 1.5f, t);
246
247         //2D rotation about Z applied in space around pivot
248         float rad = angleZDeg * Mathf.Deg2Rad;
249         float s = Mathf.Sin(rad);
250         float c = Mathf.Cos(rad);
251
252         //Calculate offset between desired position and hornet's original position.
253         Vector3 localOffset = position.ToUnityVector() - transform.position;
254
255         for (int i = 0; i < verts.Length; i++)
256         {
257             Vector3 v0 = originalVertices[i];
258
259             //translate to pivot
260             float px = v0.x - pivot.x;
261             float py = v0.y - pivot.y;
262             float pz = v0.z - pivot.z;
263
264             //uniform scale in XY
265             px *= scale;
266             py *= scale;
267
268             //rotate in XY
269             float rx = px * c - py * s;
270             float ry = px * s + py * c;
271             float rz = pz;
272
273             //translate back from pivot, apply translation offset
274             verts[i] = new Vector3(rx + pivot.x, ry + pivot.y, rz + pivot.z) + localOffset;
275         }
276         //Update vertices with scaling change, translation change, and rotation change
277         mesh.vertices = verts;
278         mesh.RecalculateBounds();
279     }
280 }
281
```

Functions handling the object flipping:

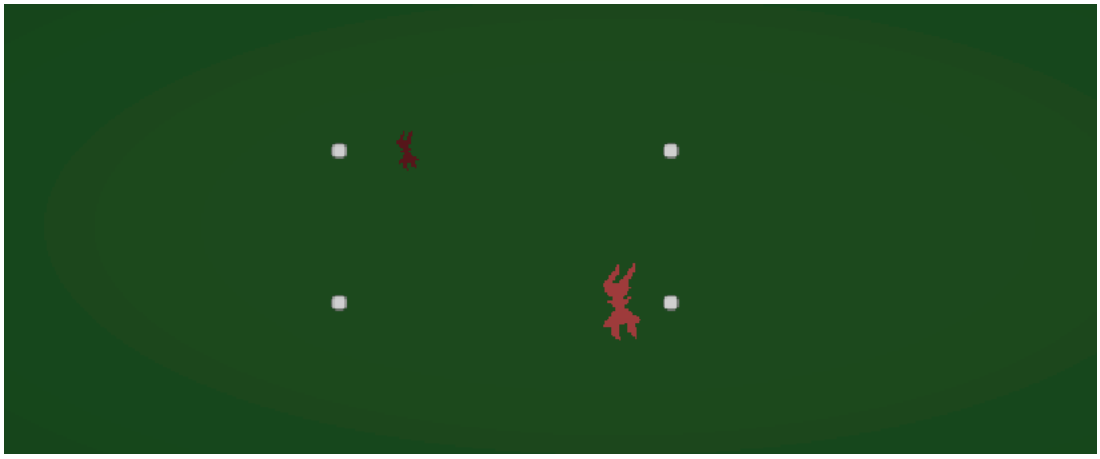
```
95      //change target
96      private void FlipTarget()
97      {
98          currentTarget = (currentTarget == PointA) ? PointB : PointA;
99      }
100
101     //bounce off point
102     private void OnTriggerEnter2D(Collider2D other)
103     {
104         // Only flip when we hit the CURRENT target, ignore the other point
105         if (other.transform == currentTarget)
106         {
107             FlipTarget();
108         }
109
110     }
```

Update function containing movement and flipping logic:

```
46      void Update()
47      {
48          //point positions
49          IGB283Vector pointA = new IGB283Vector(PointA.position);
50          IGB283Vector pointB = new IGB283Vector(PointB.position);
51          IGB283Vector target = (currentTarget == PointA) ? pointA : pointB;
52
53          //spin
54          angleZDeg += rotateSpeed * Time.deltaTime;
55          if (angleZDeg >= 360f) angleZDeg -= 360f;
56
57          //course correct based on movement
58          IGB283Vector toTarget = target - position;
59          float distSqr = IGB283Vector.SqrMagnitude(toTarget);
60
61          if (distSqr > 1e-8f)
62          {
63              IGB283Vector dir = IGB283Vector.Normalize(toTarget);
64              IGB283Vector step = IGB283Vector.Scale(dir, moveSpeed * Time.deltaTime);
65              IGB283Transform T = IGB283Transform.Translation(step.x, step.y, step.z);
66              position = T.Apply(position);
67          }
68
69          //rotate scale and move
70          changeTransformScale();
```

```
87
88          //a distance based flip in case the weird mesh makes flip not work
89          if (IGB283Vector.SqrMagnitude(toTarget) <= 0.01f)
90          {
91              FlipTarget();
92          }
93      }
```

Output:



Task 3:

A second mesh object has been created based on the original through the script “Duplicate Hornet”. This script finds the original mesh object (and boundaries), copies these objects and moves them by an offset, updates the clone’s script to bounce between the new boundaries instead of the original ones and additionally modifies the slide and rotation speed for distinguishment.

```
using UnityEngine;

public class DuplicateHornet : MonoBehaviour
{
    [SerializeField] Vector3 cloneHornetOffset = new Vector3(6f, 0f, 0f);
    [SerializeField] float cloneSlideSpeed = 2.0f;
    [SerializeField] float cloneRotationSpeed = 120f;

    private GameObject clone;

    void Start()
    {
        //finds hornet
        HornetSpinMove original = FindObjectOfType<HornetSpinMove>();

        //clones the mesh
        clone = Instantiate(
            original.gameObject,
            original.transform.position + cloneHornetOffset,
            original.transform.rotation,
            this.transform
        );
        clone.name = "HornetClone";

        //clones points
        Transform origA = original.PointA;
        Transform origB = original.PointB;

        GameObject pointC = Instantiate(origA.gameObject,
            origA.position + cloneHornetOffset, origA.rotation, this.transform);
        pointC.name = "PointC";

        GameObject pointD = Instantiate(origB.gameObject,
            origB.position + cloneHornetOffset, origB.rotation, this.transform);
        pointD.name = "PointD";

        //changes targets of cloned hornet
        HornetSpinMove hornet2 = clone.GetComponent<HornetSpinMove>();
        hornet2.PointA = pointC.transform;
        hornet2.PointB = pointD.transform;
        hornet2.moveSpeed = cloneSlideSpeed;
        hornet2.rotateSpeed = cloneRotationSpeed;

    }
}
```

Four small 2D boundary objects have been implemented for the mesh objects to bounce between. This was done by defining the original 2 boundaries for the original object and then using the duplication script (as given above) to clone these boundaries and their behaviour, therefore giving four small boundaries in total.

The original boundaries code located in the “Hornet Spin Move” script:

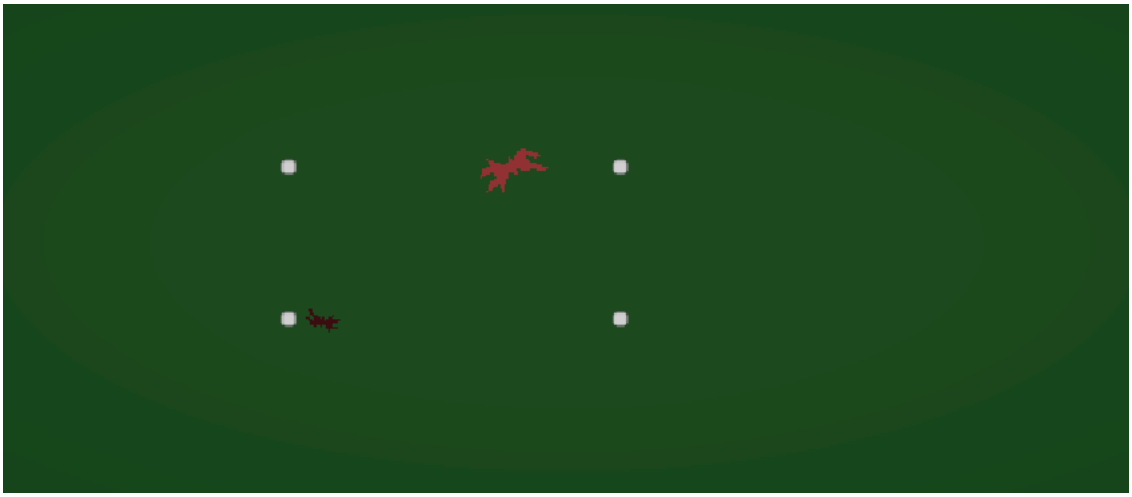
```
46      void Update()
47      {
48          //point positions
49          IGB283Vector pointA = new IGB283Vector(PointA.position);
50          IGB283Vector pointB = new IGB283Vector(PointB.position);
51          IGB283Vector target = (currentTarget == PointA) ? pointA : pointB;
52
```

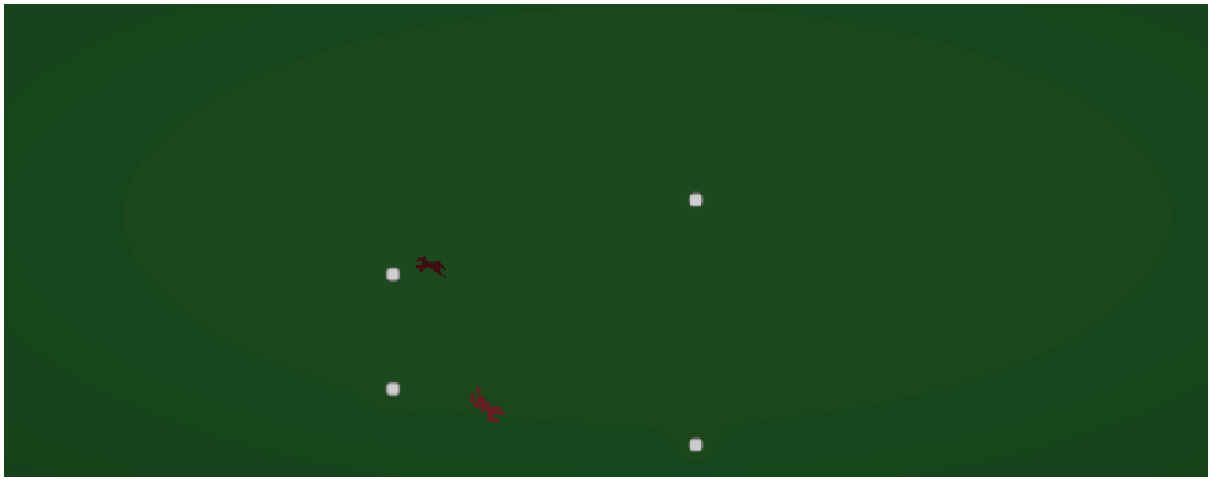
```
88          //a distance based flip in case the weird mesh makes flip not work
89          if (IGB283Vector.SqrMagnitude(toTarget) <= 0.01f)
90          {
91              FlipTarget();
92          }
93      }
94
95      //change target
96      private void FlipTarget()
97      {
98          currentTarget = (currentTarget == PointA) ? PointB : PointA;
99      }
100
101     //bounce off point
102     private void OnTriggerEnter2D(Collider2D other)
103     {
104         // Only flip when we hit the CURRENT target, ignore the other point
105         if (other.transform == currentTarget)
106         {
107             FlipTarget();
108         }
109
110     }
```

The original boundaries code located in the “Point” script:

```
using UnityEngine;
using UnityEngine.InputSystem;

public class Knob : MonoBehaviour
{
    [SerializeField] private InputAction mouseClick;
    private SpriteRenderer spriteRenderer;
    [HideInInspector] public bool IsMoving = false;

    void Start()
    {
        //get sprite
        spriteRenderer = GetComponent<SpriteRenderer>();
    }

    void Update()
    {
        //checks specifically this frame to stop confusion with other knobs
        if (Mouse.current.leftButton.wasPressedThisFrame)
            MouseClickAction(default);

        if (Mouse.current.leftButton.wasReleasedThisFrame)
            MouseReleaseAction(default);

        Move();
    }

    private void OnEnable()
    {
        mouseClick.performed += MouseClickAction;
        mouseClick.canceled += MouseReleaseAction;
        mouseClick.Enable();
    }

    private void OnDisable()
    {
        mouseClick.Disable();
        mouseClick.performed -= MouseClickAction;
        mouseClick.canceled -= MouseReleaseAction;
    }

    private Vector3 GetMousePosition()
    {
        //Get the mouse position with appropriate z coordinate
        Vector3 mouseInput = Mouse.current.position.ReadValue();
        mouseInput.z = transform.position.z - Camera.main.transform.position.z;
        //find position in world space
        Vector3 mouseInWorld = Camera.main.ScreenToWorldPoint(mouseInput);
        return mouseInWorld;
    }

    private void MouseClickAction(InputAction.CallbackContext context)
    {
        Vector3 mouseInWorld = GetMousePosition();
        Collider2D hitCollider = Physics2D.OverlapPoint(mouseInWorld);
        // Only start moving if this specific knob was clicked
        if (hitCollider != null && hitCollider.gameObject == gameObject)
        {
            IsMoving = true;
        }
    }

    private void MouseReleaseAction(InputAction.CallbackContext context)
    {
        IsMoving = false;
    }

```

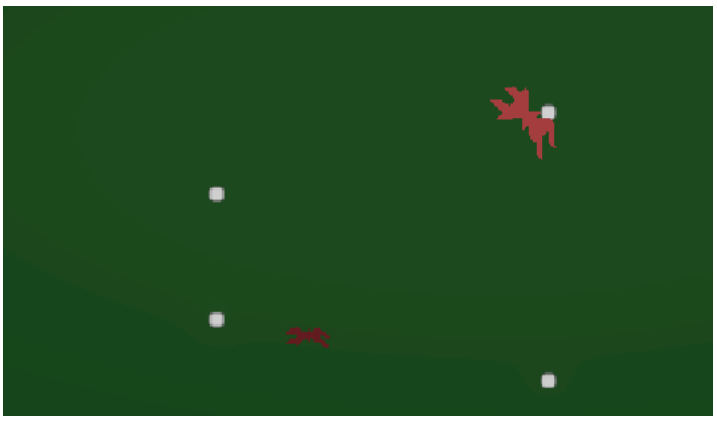
```
68      private void Move()
69      {
70          // Move the knob to the mouse
71          if (IsMoving)
72          {
73              Vector3 mousePosition = GetMousePosition();
74              transform.position = new Vector3(
75                      transform.position.x,  //current X
76                      mousePosition.y,            //mouse Y
77                      transform.position.z   //current Z
78                  );
79          }
80      }
81  }
82
```

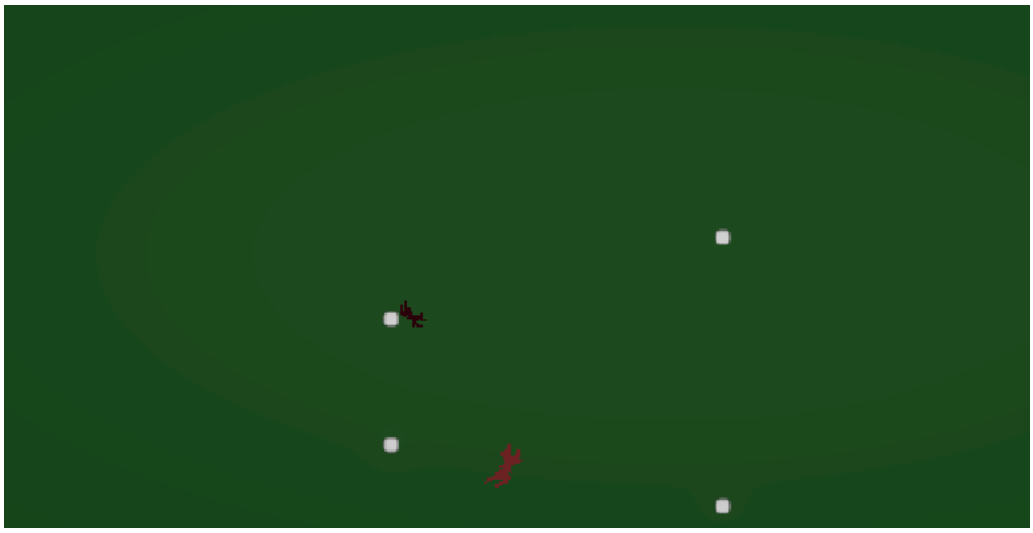




The two objects change their colour and scale based on their current x-coordinate as they translate from one side of the screen to the other. In order to alter the colour of the mesh object, the function first gets the mesh's current x position between the boundary objects. Then it calculates a blend factor between the two chosen colours based on the position and applies it to all vertices in the mesh.

```
112     //Change hornet colour depending on x position
113     private void changeColour()
114     {
115         Color black = new Color(0.0f, 0.0f, 0.0f, 1f);
116         Color red = new Color(0.459f, 0.051f, 0.051f, 1f);
117
118         Mesh mesh = GetComponent<MeshFilter>().mesh;
119         Color[] colours = new Color[mesh.vertexCount]; //List to put new vertex colours in
120
121         float minX = PointA.position.x;
122         float maxX = PointB.position.x; //Boundary x positions
123
124         for (int i = 0; i < colours.Length; i++) //For all vertices
125         {
126             float t = Mathf.InverseLerp(minX, maxX, position.x);
127             colours[i] = Color.Lerp(black, red, t); //Set colour dependent on the position between boundaries
128         }
129
130         mesh.colors = colours; //Update vertex colours
131     }
```





Task 4:

The boundary objects are able to be moved vertically by holding down the left mouse button and dragging them with the mouse up or down.

```
using UnityEngine;
using UnityEngine.InputSystem;

public class Point : MonoBehaviour
{
    [SerializeField] private InputAction mouseClick;
    private SpriteRenderer spriteRenderer;
    [HideInInspector] public bool IsMoving = false;

    void Start()
    {
        //get sprite
        spriteRenderer = GetComponent<SpriteRenderer>();
    }

    void Update()
    {
        //checks specifically this frame to stop confusion with other knobs
        if (Mouse.current.leftButton.wasPressedThisFrame)
            MouseClickAction(default);

        if (Mouse.current.leftButton.wasReleasedThisFrame)
            MouseReleaseAction(default);

        Move();
    }

    private void OnEnable()
    {
        mouseClick.performed += MouseClickAction;
        mouseClick.canceled += MouseReleaseAction;
        mouseClick.Enable();
    }

    private void OnDisable()
    {
        mouseClick.Disable();
        mouseClick.performed -= MouseClickAction;
        mouseClick.canceled -= MouseReleaseAction;
    }

    private Vector3 GetMousePosition()
    {
        //Get the mouse position with appropriate z coordinate
        Vector3 mouseInput = Mouse.current.position.ReadValue();
        mouseInput.z = transform.position.z - Camera.main.transform.position.z;
        //find position in world space
        Vector3 mouseInWorld = Camera.main.ScreenToWorldPoint(mouseInput);
        return mouseInWorld;
    }

    private void MouseClickAction(InputAction.CallbackContext context)
    {
        Vector3 mouseInWorld = GetMousePosition();
        Collider2D hitCollider = Physics2D.OverlapPoint(mouseInWorld);
        // Only start moving if this specific knob was clicked
        if (hitCollider != null && hitCollider.gameObject == gameObject)
        {
            IsMoving = true;
        }
    }

    private void MouseReleaseAction(InputAction.CallbackContext context)
    {
        IsMoving = false;
    }

```

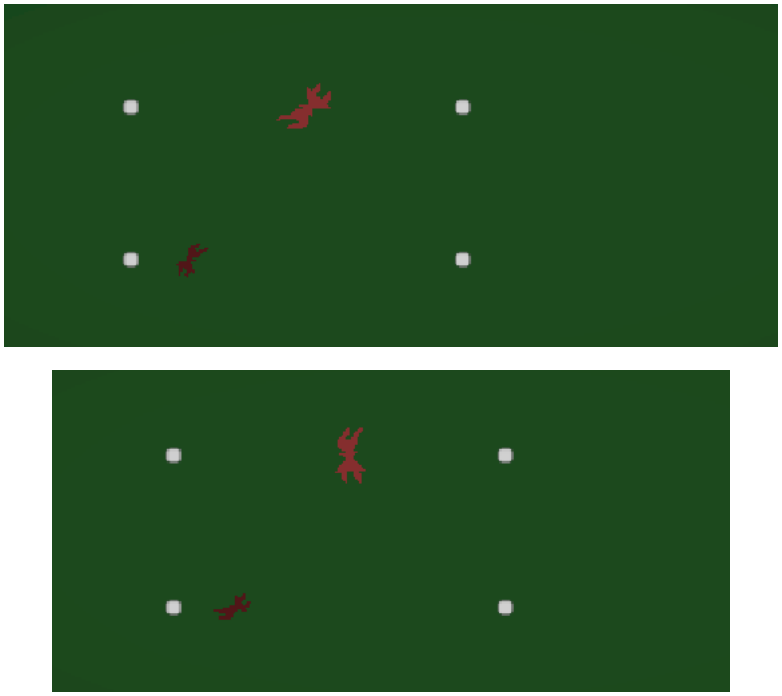
```
68      private void Move()
69      {
70          // Move the knob to the mouse
71          if (IsMoving)
72          {
73              Vector3 mousePosition = GetMousePosition();
74              transform.position = new Vector3(
75                      transform.position.x,  //current X
76                      mousePosition.y,            //mouse Y
77                      transform.position.z   //current Z
78                  );
79          }
80      }
81  }
82
```

The speed of the moving objects is controlled by pressing the ‘a’ key to slow down the movement speed and pressing the ‘d’ key to speed up the movement speed.

```
133     private void OnEnable()
134     {
135         // Enable the inputs
136         aKey.Enable();
137         dKey.Enable();
138         pKey.Enable();
139         spaceKey.Enable();
140
141         // Trigger the events
142         aKey.performed += decreaseSpeed;
143         dKey.performed += increaseSpeed;
144         pKey.performed += showPseudoCrossProduct;
145         spaceKey.performed += throwKnife;
146     }
147
148     private void OnDisable()
149     {
150         // Disable the inputs
151         aKey.Disable();
152         dKey.Disable();
153         pKey.Disable();
154         spaceKey.Disable();
155
156         //Stop triggering the events
157         aKey.performed -= decreaseSpeed;
158         dKey.performed -= increaseSpeed;
159         pKey.performed -= showPseudoCrossProduct;
160         spaceKey.performed -= throwKnife;
161     }
```

```
169     //Increase hornet move speed
170     private void increaseSpeed(InputAction.CallbackContext context)
171     {
172         moveSpeed += 1f;
173     }
174
175     //Decrease hornet move speed
176     private void decreaseSpeed(InputAction.CallbackContext context)
177     {
178         moveSpeed = Mathf.Max(0f, moveSpeed - 1f);
179     }
180
```

User lowering speed to 0 by pressing ‘a’ multiple times (mesh rotates on spot):



Task 5:

pseudo-cross product function can be toggled by pressing the ‘p’ key which displays the green/red triangles.

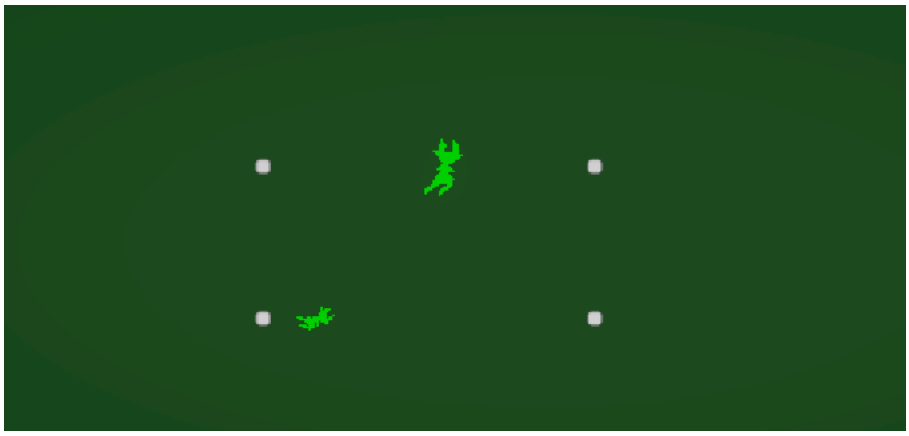
```
72              if (showingPseudoCrossProduct)
73                  applyPseudoCrossProduct();
74              else
75                  changeColour();
76
```

```
181     private void applyPseudoCrossProduct()
182     {
183         //Get mesh
184         Mesh mesh = GetComponent<MeshFilter>().mesh;
185         Vector3[] vertices = mesh.vertices;
186         int[] triangles = mesh.triangles;
187         Color[] colours = new Color[vertices.Length];
188 
189         //Get camera direction
190         Vector3 cameraDirection = Camera.main.transform.forward;
191 
192         for (int i = 0; i < triangles.Length; i += 3) //For every triangle in mesh
193         {
194             //Get triangle vertices
195             Vector3 a = vertices[triangles[i]];
196             Vector3 b = vertices[triangles[i + 1]];
197             Vector3 c = vertices[triangles[i + 2]];
198 
199             //Determine triangle edges
200             Vector3 s0 = b - a;
201             Vector3 s1 = c - a;
202 
203             //Determine normal
204             Vector3 normal = Vector3.Cross(s1, s0);
205 
206             //Rotate normal
207             Vector3 rotatedNormal = transform.rotation * normal;
208 
209             //Dot the normal with the camera direction
210             float dotProduct = Vector3.Dot(rotatedNormal, cameraDirection);
211 
212             //Positive dot product is front facing = green, back facing is red
213             Color faceColour = (dotProduct > 0f) ? Color.green : Color.red;
214 
215             //Apply colour to the vertices of the triangle
216             colours[triangles[i]] = faceColour;
217             colours[triangles[i + 1]] = faceColour;
218             colours[triangles[i + 2]] = faceColour;
219         }
220 
221         mesh.colors = colours; //Update colours
222     }
223 
224     private void showPseudoCrossProduct(InputAction.CallbackContext context)
225     {
226         showingPseudoCrossProduct = !showingPseudoCrossProduct;
227     }
```

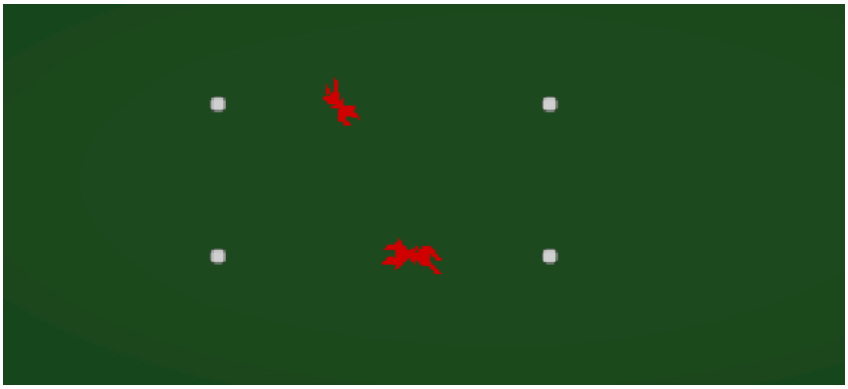
Briefly discuss your functions to display front/back faces:
The *applyPseudoCrossProduct* function calculates the logic behind distinguishing front/back facing triangles in the mesh. While *showPseudoCrossProduct* displays these front/back facing triangles with front facing triangles represented with green colouring and back facing triangles represented with red colouring. The pseudo cross product helps to determine the face direction since the normal of each triangle calculated with the pseudo cross product, when this is then dotted with the camera's direction the result indicates how similar the direction between the camera and the triangle's normal is thus determining if it is back or front facing.

The code for this is as depicted above.

Actual output of game:

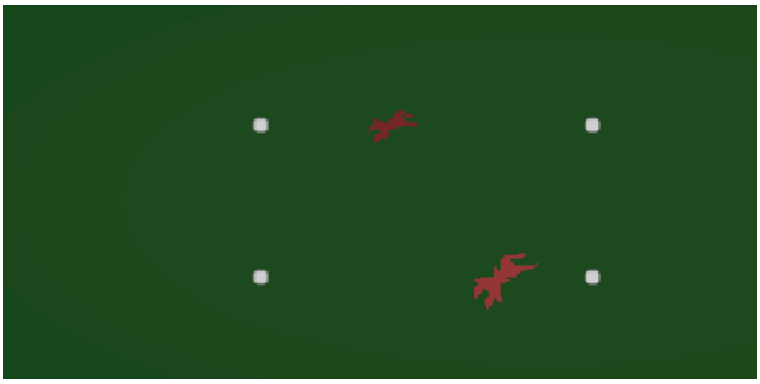


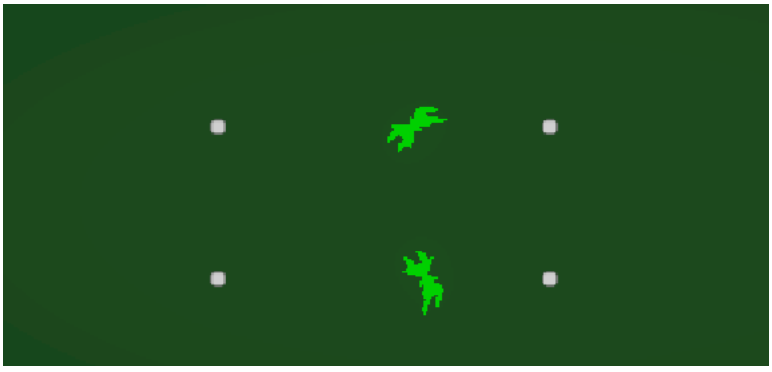
In the instance all triangles in the mesh were defined backwards:



show/hide control for mesh inspection is the key ‘p’

```
132
133     private void OnEnable()
134     {
135         // Enable the inputs
136         aKey.Enable();
137         dKey.Enable();
138         pKey.Enable();
139         spaceKey.Enable();
140
141         // Trigger the events
142         aKey.performed += decreaseSpeed;
143         dKey.performed += increaseSpeed;
144         pKey.performed += showPseudoCrossProduct;
145         spaceKey.performed += throwKnife;
146     }
147
148     private void OnDisable()
149     {
150         // Disable the inputs
151         aKey.Disable();
152         dKey.Disable();
153         pKey.Disable();
154         spaceKey.Disable();
155
156         //Stop triggering the events
157         aKey.performed -= decreaseSpeed;
158         dKey.performed -= increaseSpeed;
159         pKey.performed -= showPseudoCrossProduct;
160         spaceKey.performed -= throwKnife;
161     }
```





Task 6:

The additional feature added is the ability for the mesh object named Hornet, to throw knives in the direction she is facing using the ‘spacebar’ key.

The story of this game is about Hornet, a humanoid bug creature skilled in fighting and hunting. She is the protector of her underground home and comes across an evil clone of herself. To defend her home she fights this evil clone with her signature spin attack and her knife throwing ability.

This is why the additional feature added to this game is the ability to throw knives from the Hornet character, so that you can attack the evil clone from a distance and with good timing. The user must also take care that the clone is an exact copy of hornet and thus knows and copies her every move when she does them, so the user must carefully time their attacks to hit the evil clone while also dodging their attacks that they throw back.

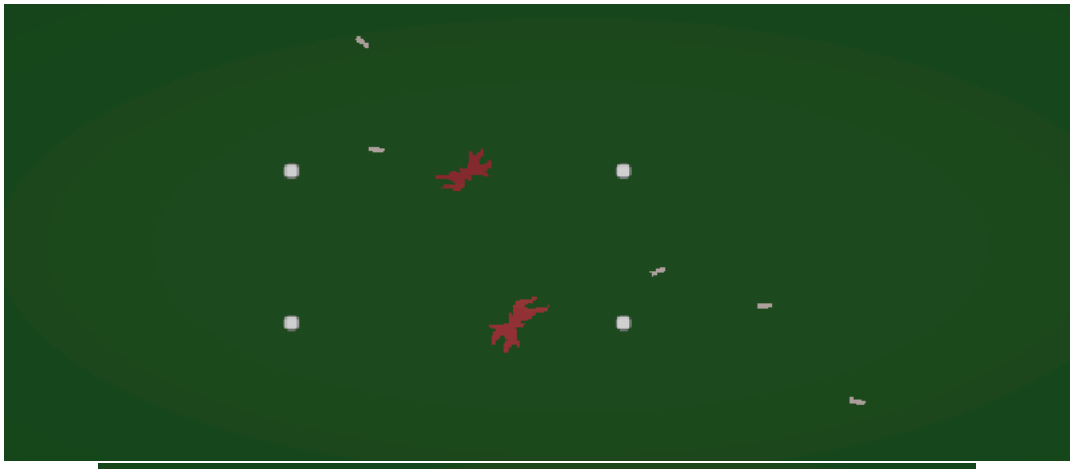
The user dodges by clicking and dragging their spinning attack boundaries and by speeding up or slowing down their movement speed with the ‘a’ and ‘d’ keys.

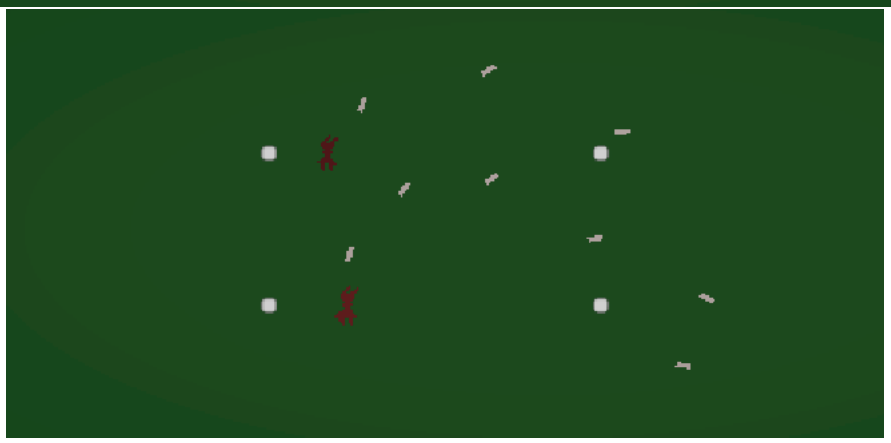
```
132
133     private void OnEnable()
134     {
135         // Enable the inputs
136         aKey.Enable();
137         dKey.Enable();
138         pKey.Enable();
139         spaceKey.Enable();
140
141         // Trigger the events
142         aKey.performed += decreaseSpeed;
143         dKey.performed += increaseSpeed;
144         pKey.performed += showPseudoCrossProduct;
145         spaceKey.performed += throwKnife;
146     }
147
148     private void OnDisable()
149     {
150         // Disable the inputs
151         aKey.Disable();
152         dKey.Disable();
153         pKey.Disable();
154         spaceKey.Disable();
155
156         //Stop triggering the events
157         aKey.performed -= decreaseSpeed;
158         dKey.performed -= increaseSpeed;
159         pKey.performed -= showPseudoCrossProduct;
160         spaceKey.performed -= throwKnife;
161     }
```

```
162
163     //Queue up a knife to be thrown (done in update function to avoid drifting spawn)
164     private void throwKnife(InputAction.CallbackContext context)
165     {
166         queueKnifeSpawn = true;
167     }
168
```

```
76
77          //make knives
78          if (queueKnifeSpawn)
79          {
80              //First line roates the knife spawner in the same way as the hornetmesh
81              Quaternion rotation = Quaternion.Euler(0f, 0f, angleZDeg);
82              //Hornet mesh is the spawnPosition, and the knife is spawned there
83              Vector3 spawnPosition = position.ToUnityVector();
84              Instantiate(knife, spawnPosition, rotation);
85              queueKnifeSpawn = false;
86          }
```

```
using UnityEngine;

public class ThrowingKnives : MonoBehaviour
{
    public float speed = 10f; //Knife travel speed
    public float lifetime = 2f; //Life of knife before destroyed
    private Vector3 direction; //Direction the knife should travel

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {
        //Set the knife's direction to its local 'up' when spawned, so it is thrown in sync with Hornet's spin
        direction = transform.up.normalized;

        Destroy(gameObject, lifetime); //Destroy knife when lifetime has passed
    }

    // Update is called once per frame
    void Update()
    {
        //Travel in a straight line
        transform.position += direction * speed * Time.deltaTime;
    }
}

```





User Instructions:

Altering the Speed of the Objects:

[Keys or Mouse Buttons or others] Used:

The speed of the object that is initially towards the top of the screen can be altered by using the ‘a’ key to slow it down and the ‘d’ key to speed it up.

Similarly, the speed of the object that is initially positioned towards the bottom of the screen can be altered by using the same ‘a’ key to slow it down and the ‘d’ key to speed it up.

Changing the Boundary Objects:

[Keys or Mouse Buttons or others] Used:

There is a total of four boundary objects present within the project that can all be dragged vertically up or down by clicking and holding down the left mouse button and moving the mouse to the desired location.

Toggling the pseudo cross product display:
The toggle for viewing which triangles in the mesh are front/back facing is pressing the ‘p’ key.

Throwing knives (Task 6):
Knives can be thrown by pressing the ‘spacebar’ key.

Statement of Contribution:

Work was split between both students by dividing the tasks in the following way:

Student 1:

- Tasks 3.3, 3.4
- Task 4.2
- Tasks 5.2, 5.3, 5.4
- Tasks 6.1, 6.2, 6.3

Student 2:

- Tasks 2.2, 2.3, 2.4, 2.5
- Tasks 3.1, 3.2
- Task 4.1