F20GP: Games Programming
Coursework: Programming Assignment.

Heriot Watt University
Jamie Fraser- H00216091

Table of Contents

Code Listing

Explanations on how a star works and how graphics is separate from the physics are found the in comments of the code.

Bouncing Ball

BallPhysics.cs

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4.
5. public class BallPhysics : MonoBehaviour {
6.
7.     //objects floor and sphere colider
8.     GameObject floor;
9.     GameObject Collider;
10.
11.    //gravity velocity and the energy loss from hitting the floor
12.    Vector3 velocity = new Vector3(0, 0, 0);
13.    Vector3 gravity = new Vector3(0, -9.8f, 0);
14.    float engeryLoss = 0.8f;
15.
16.    //check if the ball is falling, going down and is bouncing
17.    bool motion;
18.    int bounce = 0;
19.
20.    // Use this for initialization
21.    void Start ()
22.    {
23.
```

```
24.             floor =  GameObject.Find("Cube");
25.             Collider = GameObject.Find("sphereCollider");
26.
27.             //begin the ball falling
28.             motion = true;
29.     }
30.
31.
32.     void FixedUpdate(){
33.
34.             //the ball is moving
35.             if(motion){
36.
37.                     //if collider touches floor and has a negitve velocity
38.                     if((Collider.transform.position.y-2.5) <= (floor.transform.position.y)
&& velocity.y <=0){
39.                             bounceBall();
40.                     }
41.
42.                     moveObj();
43.             }
44.     }
45.
46.
47.     void bounceBall(){
48.
49.             bounce++;
50.             Debug.Log(bounce);
51.
52.             //engery is lost(realisticly to sound and heat)
53.             velocity.y = (velocity.y * -1) * engeryLoss;
```

```
54.
55.             //after a time bouncing, stop the ball
56.             if(bounce == 20){
57.                     motion = false;
58.             }
59.     }
60.
61.
62.     void moveObj(){
63.             //velocity accelerated/slow by gravity
64.             velocity = velocity + gravity * Time.fixedDeltaTime;
65.             //move the ball and its collider
66.             transform.Translate(velocity * Time.fixedDeltaTime);
67.             Collider.transform.position = transform.position;
68.
69.     }
70. }
```

Particle Explosion

Explosion.cs

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4.
5. public class explosion : MonoBehaviour {
6.
7.      //physics and movemnt
8.      public GameObject[] particleObjArr;
9.      public GameObject particleObject;
10.     public GameObject cube;
11.
12.     Vector3 velocity;
13.     Vector3[] velocityArr = new Vector3[particleNo];
14.     Vector3 cubevelocity;
15.     Vector3 gravity = new Vector3(0, -9.8f, 0);
16.
17.     public static int particleNo = 1000;
18.     public int timeOut  = 0;
19.
20.     public bool explode;
21.     public bool dropCube;
22.
23.
24.     // Use this for initialization
25.     void Start ()
26.     {
27.             cube = Instantiate(Resources.Load ("cube")) as GameObject;
```

```
28.             particleObjArr = new GameObject[particleNo];
29.
30.     for (int i = 0; i < particleNo; i++)
31.     {
32.                 //loads in the picture as the game object
33.                 particleObject = Instantiate(Resources.Load ("particle")) as
GameObject;
34.                 particleObject.transform.position = new Vector3(0,0,0);
35.
36.                 //adds the obj to the array and hides it on screen
37.                 particleObjArr[i] = particleObject;
38.                 particleObjArr[i].SetActive(false);
39.
40.                 //arr of random vectors for each partical
41.                 velocityArr[i] = new Vector3(Random.Range(-10.0f,
20.0f),Random.Range(-5.0f, 20.0f),Random.Range(-10.0f, 20.0f));
42.     }
43.   }
44.
45.   void FixedUpdate(){
46.             //press to being scene
47.             if (Input.GetKeyDown("space")){
48.                         dropCube = true;
49.             }
50.             //cube is falling
51.             if(dropCube){
52.                 cubeFall();
53.             }
54.             //partical is exploading
55.             if (explode){
56.                 exploading();
```

```
57.             }
58.             //after explosion happens for a while destroy all particals to save memory
59.             if (timeOut == 160){
60.                     explode = false;
61.                     for(int i=0; i < particleNo; i++){
62.                             Destroy(particleObjArr[i]);
63.                     }
64.             }
65.
66.     }
67.
68.     //drops the cube at the start of the scene into positon then sets it to false and begins the explosion
69.     void cubeFall(){
70.
71.             if(cube.transform.position.y <=0){
72.                             dropCube = false;
73.                             explode =true;
74.                             cube.SetActive(false);
75.                     }
76.                     else{
77.                             moveObj(ref cubevelocity, ref cube);
78.
79.                     }
80.     }
81.
82.     //sets all particals to active then is continually called which then mvoes them
83.     void exploading(){
84.                     timeOut++;
85.                     for(int i=0; i < particleNo; i++){
86.                             particleObjArr[i].SetActive(true);
```

```
87.                             moveObj(ref velocityArr[i], ref particleObjArr[i]);
88.                         }
89.
90.     }
91.
92.     //applies gravity and changes velocity then moves obj its passed
93.     void moveObj(ref Vector3 vel,ref GameObject obj){
94.                             vel = vel + gravity * Time.fixedDeltaTime;
95.                             obj.transform.Translate(vel * Time.fixedDeltaTime);
96.     }
97.
98. }
99.
100.
```

Flocking Boids

Flock.cs

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4.
5. public class flock : MonoBehaviour {
6.
7.       public float velocity;
8.
9.       float rotationSpeed = 4.0f;
10.    float neighbourDistance = 5.0f;
11.     bool turning = false;
12.    float gSpeed;
13.    float dist;
14.    int groupSize;
15.
16.    Vector3 averageHeading;
17.    Vector3 averagePosition;
18.    Vector3 direction;
19.
20.    Vector3 goalPos;
21.    Vector3 vcentre;
22.    Vector3 vavoid;
23.
24.    GameObject[] gObj;
25.
26.
27.
```

```
28.     // Use this for initialization
29.     void Start () {
30.       //start them all with a diffrent velocity
31.             velocity = Random.Range(1f, velocity);
32.     }
33.
34.     // Update is called once per frame
35.     void Update () {
36.
37.       //if they are about to hit the edge of the space, turn them about
38.             if (Vector3.Distance(transform.position, Vector3.zero) >= globalFlock.space)
39.         turning = true;
40.       else
41.         turning = false;
42.
43.       if (turning){
44.
45.         //turn them around and change velocity
46.         direction = Vector3.zero - transform.position;
47.         transform.rotation = Quaternion.Slerp(transform.rotation,
Quaternion.LookRotation(direction), rotationSpeed * Time.deltaTime);
48.         velocity = Random.Range(1f, velocity);
49.
50.       }
51.       else{
52.
53.                   if (Random.Range(0,3) < 1)
54.                         reDirect();
55.
56.
57.             }
```

```
58.
59.         //moves everything depenat on velocity
60.         transform.Translate(0, 0, Time.deltaTime * velocity);
61.     }
62.
63.
64.
65.     void reDirect() {
66.
67.         gObj = globalFlock.allBoids;
68.         vavoid = Vector3.zero;
69.         vcentre = Vector3.zero;
70.         goalPos = globalFlock.centerSpace;
71.         groupSize = 0;
72.         gSpeed = 0.1f;
73.
74.
75.         //checks all the other boids
76.         foreach (GameObject go in gObj){
77.
78.             //for other boids check the how far away they are
79.             if (go != this.gameObject){
80.                 dist = Vector3.Distance(go.transform.position, this.transform.position);
81.
82.                 //if they are close enough group them up
83.                 if (dist <= neighbourDistance){
84.
85.                     vcentre += go.transform.position;
86.                     groupSize++;
87.
```

```
88.                 //if they are too close then avoid
89.                 if (dist < 1.5f)
90.                     vavoid = vavoid + (this.transform.position - go.transform.position);
91.
92.                 //creates another flock
93.                 flock anotherFlock = go.GetComponent<flock>();
94.                 gSpeed = gSpeed + anotherFlock.velocity;
95.
96.             }
97.
98.         }
99.
100.     }
101.
102.
103.     //if there is a group formed then s
104.     if (groupSize > 0){
105.
106.       //  Debug.Log("group size : "+ groupSize);
107.         vcentre = vcentre / groupSize + (goalPos - this.transform.position);
108.         //creates an avrage speed of group
109.         velocity = gSpeed / groupSize;
110.
111.         //moves direction
112.         direction = (vcentre + vavoid) - transform.position;
113.         if (direction != Vector3.zero)
114.             transform.rotation = Quaternion.Slerp(transform.rotation,
Quaternion.LookRotation(direction), rotationSpeed * Time.deltaTime);
115.
116.     }
117.
```

```
118.     }
119.
120.
121. }
122.
```

GlobalFlock.cs

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4.
5. public class globalFlock : MonoBehaviour {
6.
7.      //used for the demo to toggle 3d
8.      public bool threeDimensional;
9.
10.     // Use this for initialization
11.     public GameObject BoidPrefab;
12.
13.     //area of movemnt
14.     public static int space = 20;
15.
16.     //number of spiders in this case
17.   static int numBoids = 10;
18.   public static GameObject[] allBoids = new GameObject[numBoids];
19.     public static Vector3 centerSpace = Vector3.zero;
20.
21.
22.
23.     // Use this for initialization
24.     void Start () {
25.
26.             if(threeDimensional){
27.                     //if 3d then hide the plane so they cant walk on it
28.                     GameObject.Find("Plane").SetActive(false);
29.             }
```

```
30.
31.             //loops through and makes how every many instanaces of the boids,  and spawns them in a certain location
32.         for (int i = 0; i < numBoids; i++) {
33.                         Vector3 pos = new Vector3(Random.Range(-space, space), 0 ,Random.Range(-space, space));
34.
35.                         if(threeDimensional){
36.                                 //if 3d then they can go on a random y cordinate too
37.                                 pos.y = Random.Range(-space, space);
38.                         }
39.
40.                         //add them to and array of gameObjects and rotates them to a randrom roation
41.             allBoids[i] = (GameObject) Instantiate(BoidPrefab, pos, Quaternion.identity);
42.                         allBoids[i].transform.rotation =  Quaternion.Euler(0, Random.Range(0, 360),0);
43.         }
44.     }
45.
46.     // Update is called once per frame
47.     void Update () {
48.
49.             //if one in 10 chance the move the certer space to change the rough direction of all of them
50.             if (Random.Range (0, 10) < 1) {
51.
52.                         if(threeDimensional)
53.                                 centerSpace = new Vector3 (Random.Range (-space, space),Random.Range (-space, space),Random.Range (-space, space));
54.                         else
55.                                 centerSpace = new Vector3 (Random.Range (-space, space),0,Random.Range (-space, space));
```

```
56.                 }
57.     }
58. }
```

A * Pathfinding

Sebastian Lague's YouTube tutorials were used to learn how to do this.
https://www.youtube.com/watch?v=-L-WgKMFuhE

Node.cs

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. using UnityEditor.Build;
5.
6.
7.
8. public class Node {
9.
10.
11.     public Vector3 worldPosition;
12.
13.     //keeps track of own position
14.     public int gCost;
15.     public int hCost;
16.     public int Xlocation;
17.     public int Ylocation;
18.
19.     //whether it can is walkable or not
20.     public bool notwall;
21.
22.     //keeps track of its parent node
23.     public Node parent;
24.
25.
26.     public Node(bool _notwall, Vector3 _worldPosition, int _Xlocation, int _Ylocation)
```

```
27.     {
28.
29.             notwall = _notwall;
30.
31.             //sets nodes postions
32.             worldPosition = _worldPosition;
33.             Xlocation = _Xlocation;
34.             Ylocation = _Ylocation;
35.
36.     }
37.
38.     //cost to goal from start
39.     public int fCost
40.     {
41.
42.             get{
43.                     return gCost + hCost;
44.             }
45.     }
46.
47. }
48.
49.
50.
```

Pathfinding.cs

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. using UnityEditor.Build;
5.
6. public class Pathfinding : MonoBehaviour {
7.
8.      Grid grid;
9.      public Transform seeker,target;
10.     Node startingNode;
11.     Node targetNode;
12.
13.
14.     void Start()
15.     {
16.             //loads the grid
17.             grid = GetComponent<Grid>();
18.     }
19.
20.     void Update()
21.     {
22.             //seeker
23.             startingNode = grid.WorldPointNode(seeker.position);
24.             //target
25.             targetNode = grid.WorldPointNode(target.position);
26.
27.             //sorts the nodes into checked and unchecked
28.             List<Node> open = new List<Node>();
29.             List<Node> closed = new List<Node>();
```

```
30.
31.             //add the seeker node to open
32.             open.Add(startingNode);
33.
34.
35.             //loops through all nodes
36.             while(open.Count > 0){
37.                     Node currentNode = open[0];
38.
39.                     //Compares cost of open nodes and if there is a better option then add
it.
40.                     for(int i = 1; i < open.Count; i++){
41.                             if(open[i].fCost < currentNode.fCost || open[i].fCost ==
currentNode.fCost && open[i].hCost < currentNode.hCost){
42.                                     currentNode = open[i];
43.                             }
44.                     }
45.
46.                     //remove that node from the open and add it to the close to show its
been checked
47.                     open.Remove(currentNode);
48.                     closed.Add(currentNode);
49.
50.                     //if the target is found the call retrace path
51.                     if (currentNode == targetNode){
52.                             RetracePath(startingNode, targetNode);
53.                             return;
54.                     }
55.
56.                     //searches the nabouors that are left, i.e the ones that are walkable and
that are not closed
57.                     foreach(Node neighbour in grid.GetNeighbour(currentNode)){
```

```
58.                     if(!neighbour.notwall || closed.Contains(neighbour)){
59.                         continue;
60.                     }
61.
62.                     //gets the cost using the ecludian huristic
63.                     int newMovementCostToNeighbour = currentNode.gCost +
Distance(currentNode, neighbour);
64.                     //if the cost is smaller that the other ones then choose that to be
the best node for the path
65.                     if (newMovementCostToNeighbour < neighbour.gCost ||
!open.Contains(neighbour)){
66.                         neighbour.gCost = newMovementCostToNeighbour;
67.                         neighbour.hCost = Distance(neighbour, targetNode);
68.                         neighbour.parent = currentNode;
69.
70.
71.                         //if its not been added alread then add it
72.                         if (!open.Contains(neighbour))
73.                             open.Add(neighbour);
74.
75.
76.                     }
77.                 }
78.             }
79.
80.     }
81.
82.     //create path List then, loops throuhg the nodes backwards and draws the path in
reverse
83.     void RetracePath(Node startNode, Node endNode)
84.     {
85.
```

```
86.             List<Node> Paths = new List<Node>();
87.             Node currentNode = endNode;
88.
89.             //loop through the path untill the get to the start node
90.             while (currentNode != startNode){
91.                     Paths.Add(currentNode);
92.                     currentNode = currentNode.parent;
93.                     //uses the parent to find the previous node to find the path
94.
95.             }
96.             Paths.Reverse();
97.
98.             grid.path = Paths;
99.     }
100.
101.
102.
103.    int Distance (Node nodeA, Node nodeB)
104.    {
105.
106.            //heuristic is used here to in distance calucation.
107.            //a diagonal move is worth 2 and an staight movemnt is 1
108.
109.
110.            int Xdistance = Mathf.Abs(nodeA.Xlocation -nodeB.Xlocation);
111.            int ydistance = Mathf.Abs(nodeA.Ylocation -nodeB.Ylocation);
112.            int diagonal = 2;
113.            int straight = 1;
114.
115.            if (Xdistance > ydistance)
```

```
116.                    return diagonal*ydistance + straight* (Xdistance-ydistance);
117.
118.            else
119.                    return diagonal*Xdistance + straight* (ydistance-Xdistance);
120.
121.
122.            //In place of using the vector3 GetDistance which uses euclidean heuristics,
This function uses the manhattan distanace
123.
124.    }
125.
126. }
127.
```

Grid.cs

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. using UnityEditor.Build;
5.
6. public class Grid : MonoBehaviour {
7.
8.
9.      //layer for walls for unity visualation
10.     public LayerMask wall;
11.
12.     //Grid for nodes
13.     Node[,] grid;
14.     public Vector2 gridSize;
15.     int gridX,gridY;
16.
17.     //specfic node size
18.     public float nodeRadius;
19.     float nodeDiameter;
20.
21.     Vector3 gridCorner;
22.     Vector3 worldPoint;
23.
24.
25.     void Start()
26.     {
27.             //Create a grid at the right size for the nodes
28.             nodeDiameter = nodeRadius*2;
29.             gridX = Mathf.RoundToInt(gridSize.x/nodeDiameter);
```

```
30.             gridY = Mathf.RoundToInt(gridSize.y/nodeDiameter);
31.
32.
33.             grid = new Node[gridX,gridY];
34.     }
35.
36.
37.     void Update()
38.     {
39.
40.             //starts at the corner of the grid then moves throgugh the rest of the of the grid to
create nodes
41.             gridCorner = transform.position - Vector3.right * gridSize.x/2 - Vector3.forward *
gridSize.y/2;
42.
43.             for(int x = 0; x < gridX; x++){
44.                     for(int y = 0; y < gridY; y++){
45.
46.                             //loops throught the nodes and checks if they are walkable or not
47.                             //then creates a node at that point in the grid
48.                             worldPoint = gridCorner + Vector3.right * (x * nodeDiameter +
nodeRadius) + Vector3.forward * (y * nodeDiameter + nodeRadius);
49.                             grid[x,y] = new
Node(!(Physics.CheckSphere(worldPoint,nodeRadius,wall)),worldPoint,x,y);
50.                     }
51.             }
52.     }
53.
54.
55.
56.     public List<Node> GetNeighbour(Node node)
57.     {
```

```
58.             //will have a list of node neihbours
59.             List<Node> nodeNeighbours = new List<Node>();
60.
61.             //loops through left and right nodes, skips over the cuernt node and then adds
nodes to the list.
62.             for(int x = -1; x <= 1; x++){
63.                     for(int y = -1; y <= 1; y++){
64.                             if(x == 0 && y == 0){
65.                                     continue;
66.                             }
67.
68.                             int checkX = node.Xlocation + x;
69.                             int checkY = node.Ylocation + y;
70.
71.                             if(checkX >= 0 && checkX < gridX && checkY >= 0 && checkY <
gridY){
72.                                     nodeNeighbours.Add(grid[checkX,checkY]);
73.                             }
74.
75.                     }
76.             }
77.
78.             return nodeNeighbours;
79.
80.     }
81.
82.
83.
84.     public Node WorldPointNode(Vector3 worldPosition)
85.     {
86.             //
87.             float xPercentage = (worldPosition.x + gridSize.x/2) / gridSize.x;
```

```
88.                 float yPercentage = (worldPosition.z + gridSize.y/2) / gridSize.y;
89.                 //makes whole number
90.                 xPercentage = Mathf.Clamp01(xPercentage);
91.                 yPercentage = Mathf.Clamp01(yPercentage);
92.
93.                 return grid[Mathf.RoundToInt((gridX-1) * xPercentage),Mathf.RoundToInt((gridY-1)
* yPercentage)];
94.         }
95.
96.
97.
98.         public List<Node> path;
99.
100.
101.
102.
103. }
104.
```

Vector Class Listing

Vector calls taken from
http://www.technologicalutopia.com/sourcecode/xnageometry/vector3.cs.htm

Unity vector3 was used in the coursework however

```
1. using System;
2. using System.ComponentModel;
3. using System.Diagnostics;
4. using System.Text;
5.
6. namespace XnaGeometry
7. {
8.     [Serializable]
9.     public struct Vector3 : IEquatable<Vector3>
10.    {
11.        #region Private Fields
12.
13.        private static  Vector3 zero = new Vector3(0f, 0f, 0f);
14.        private static  Vector3 one = new Vector3(1f, 1f, 1f);
15.        private static  Vector3 unitX = new Vector3(1f, 0f, 0f);
16.        private static  Vector3 unitY = new Vector3(0f, 1f, 0f);
17.        private static  Vector3 unitZ = new Vector3(0f, 0f, 1f);
18.        private static  Vector3 up = new Vector3(0f, 1f, 0f);
19.        private static  Vector3 down = new Vector3(0f, -1f, 0f);
20.        private static  Vector3 right = new Vector3(1f, 0f, 0f);
21.        private static Vector3 left = new Vector3(-1f, 0f, 0f);
22.        private static Vector3 forward = new Vector3(0f, 0f, -1f);
23.        private static Vector3 backward = new Vector3(0f, 0f, 1f);
24.
25.        #endregion Private Fields
```

```
26.
27.
28.     #region Public Fields
29.
30.     public double X;
31.     public double Y;
32.     public double Z;
33.
34.     #endregion Public Fields
35.
36.
37.     #region Properties
38.
39.     public static Vector3 Zero
40.     {
41.         get { return zero; }
42.     }
43.
44.     public static Vector3 One
45.     {
46.         get { return one; }
47.     }
48.
49.     public static Vector3 UnitX
50.     {
51.         get { return unitX; }
52.     }
53.
54.     public static Vector3 UnitY
55.     {
56.         get { return unitY; }
```

```
57.     }
58.
59.     public static Vector3 UnitZ
60.     {
61.         get { return unitZ; }
62.     }
63.
64.     public static Vector3 Up
65.     {
66.         get { return up; }
67.     }
68.
69.     public static Vector3 Down
70.     {
71.         get { return down; }
72.     }
73.
74.     public static Vector3 Right
75.     {
76.         get { return right; }
77.     }
78.
79.     public static Vector3 Left
80.     {
81.         get { return left; }
82.     }
83.
84.     public static Vector3 Forward
85.     {
86.         get { return forward; }
87.     }
```

```
88.
89.     public static Vector3 Backward
90.     {
91.         get { return backward; }
92.     }
93.
94.     #endregion Properties
95.
96.
97.     #region Constructors
98.
99.     public Vector3(double x, double y, double z)
100.     {
101.         this.X = x;
102.         this.Y = y;
103.         this.Z = z;
104.     }
105.
106.
107.     public Vector3(double value)
108.     {
109.         this.X = value;
110.         this.Y = value;
111.         this.Z = value;
112.     }
113.
114.
115.     public Vector3(Vector2 value, double z)
116.     {
117.         this.X = value.X;
118.         this.Y = value.Y;
```

```
119.            this.Z = z;
120.        }
121.
122.
123.        #endregion Constructors
124.
125.
126.        #region Public Methods
127.
128.        public static Vector3 Add(Vector3 value1, Vector3 value2)
129.        {
130.            value1.X += value2.X;
131.            value1.Y += value2.Y;
132.            value1.Z += value2.Z;
133.            return value1;
134.        }
135.
136.        public static void Add(ref Vector3 value1, ref Vector3 value2, out Vector3 result)
137.        {
138.            result.X = value1.X + value2.X;
139.            result.Y = value1.Y + value2.Y;
140.            result.Z = value1.Z + value2.Z;
141.        }
142.
143.        public static Vector3 Barycentric(Vector3 value1, Vector3 value2, Vector3 value3, double
amount1, double amount2)
144.        {
145.            return new Vector3(
146.                MathHelper.Barycentric(value1.X, value2.X, value3.X, amount1, amount2),
147.                MathHelper.Barycentric(value1.Y, value2.Y, value3.Y, amount1, amount2),
148.                MathHelper.Barycentric(value1.Z, value2.Z, value3.Z, amount1, amount2));
```

```
149.        }
150.
151.        public static void Barycentric(ref Vector3 value1, ref Vector3 value2, ref Vector3 value3,
double amount1, double amount2, out Vector3 result)
152.        {
153.            result = new Vector3(
154.                MathHelper.Barycentric(value1.X, value2.X, value3.X, amount1, amount2),
155.                MathHelper.Barycentric(value1.Y, value2.Y, value3.Y, amount1, amount2),
156.                MathHelper.Barycentric(value1.Z, value2.Z, value3.Z, amount1, amount2));
157.        }
158.
159.        public static Vector3 CatmullRom(Vector3 value1, Vector3 value2, Vector3 value3, Vector3
value4, double amount)
160.        {
161.            return new Vector3(
162.                MathHelper.CatmullRom(value1.X, value2.X, value3.X, value4.X, amount),
163.                MathHelper.CatmullRom(value1.Y, value2.Y, value3.Y, value4.Y, amount),
164.                MathHelper.CatmullRom(value1.Z, value2.Z, value3.Z, value4.Z, amount));
165.        }
166.
167.        public static void CatmullRom(ref Vector3 value1, ref Vector3 value2, ref Vector3 value3,
ref Vector3 value4, double amount, out Vector3 result)
168.        {
169.            result = new Vector3(
170.                MathHelper.CatmullRom(value1.X, value2.X, value3.X, value4.X, amount),
171.                MathHelper.CatmullRom(value1.Y, value2.Y, value3.Y, value4.Y, amount),
172.                MathHelper.CatmullRom(value1.Z, value2.Z, value3.Z, value4.Z, amount));
173.        }
174.
175.        public static Vector3 Clamp(Vector3 value1, Vector3 min, Vector3 max)
176.        {
177.            return new Vector3(
```

```
178.            MathHelper.Clamp(value1.X, min.X, max.X),
179.            MathHelper.Clamp(value1.Y, min.Y, max.Y),
180.            MathHelper.Clamp(value1.Z, min.Z, max.Z));
181.        }
182.
183.        public static void Clamp(ref Vector3 value1, ref Vector3 min, ref Vector3 max, out Vector3
result)
184.        {
185.            result = new Vector3(
186.                MathHelper.Clamp(value1.X, min.X, max.X),
187.                MathHelper.Clamp(value1.Y, min.Y, max.Y),
188.                MathHelper.Clamp(value1.Z, min.Z, max.Z));
189.        }
190.
191.        public static Vector3 Cross(Vector3 vector1, Vector3 vector2)
192.        {
193.            Cross(ref vector1, ref vector2, out vector1);
194.            return vector1;
195.        }
196.
197.        public static void Cross(ref Vector3 vector1, ref Vector3 vector2, out Vector3 result)
198.        {
199.            result = new Vector3(vector1.Y * vector2.Z - vector2.Y * vector1.Z,
200.                                 -(vector1.X * vector2.Z - vector2.X * vector1.Z),
201.                                 vector1.X * vector2.Y - vector2.X * vector1.Y);
202.        }
203.
204.        public static double Distance(Vector3 vector1, Vector3 vector2)
205.        {
206.            double result;
207.            DistanceSquared(ref vector1, ref vector2, out result);
```

```
208.            return (double)Math.Sqrt(result);
209.        }
210.
211.        public static void Distance(ref Vector3 value1, ref Vector3 value2, out double result)
212.        {
213.            DistanceSquared(ref value1, ref value2, out result);
214.            result = (double)Math.Sqrt(result);
215.        }
216.
217.        public static double DistanceSquared(Vector3 value1, Vector3 value2)
218.        {
219.            double result;
220.            DistanceSquared(ref value1, ref value2, out result);
221.            return result;
222.        }
223.
224.        public static void DistanceSquared(ref Vector3 value1, ref Vector3 value2, out double
result)
225.        {
226.            result = (value1.X - value2.X) * (value1.X - value2.X) +
227.                     (value1.Y - value2.Y) * (value1.Y - value2.Y) +
228.                     (value1.Z - value2.Z) * (value1.Z - value2.Z);
229.        }
230.
231.        public static Vector3 Divide(Vector3 value1, Vector3 value2)
232.        {
233.            value1.X /= value2.X;
234.            value1.Y /= value2.Y;
235.            value1.Z /= value2.Z;
236.            return value1;
237.        }
```

```
238.
239.        public static Vector3 Divide(Vector3 value1, double value2)
240.        {
241.            double factor = 1 / value2;
242.            value1.X *= factor;
243.            value1.Y *= factor;
244.            value1.Z *= factor;
245.            return value1;
246.        }
247.
248.        public static void Divide(ref Vector3 value1, double divisor, out Vector3 result)
249.        {
250.            double factor = 1 / divisor;
251.            result.X = value1.X * factor;
252.            result.Y = value1.Y * factor;
253.            result.Z = value1.Z * factor;
254.        }
255.
256.        public static void Divide(ref Vector3 value1, ref Vector3 value2, out Vector3 result)
257.        {
258.            result.X = value1.X / value2.X;
259.            result.Y = value1.Y / value2.Y;
260.            result.Z = value1.Z / value2.Z;
261.        }
262.
263.        public static double Dot(Vector3 vector1, Vector3 vector2)
264.        {
265.            return vector1.X * vector2.X + vector1.Y * vector2.Y + vector1.Z * vector2.Z;
266.        }
267.
268.        public static void Dot(ref Vector3 vector1, ref Vector3 vector2, out double result)
```

```
269.        {
270.            result = vector1.X * vector2.X + vector1.Y * vector2.Y + vector1.Z * vector2.Z;
271.        }
272.
273.        public override bool Equals(object obj)
274.        {
275.            return (obj is Vector3) ? this == (Vector3)obj : false;
276.        }
277.
278.        public bool Equals(Vector3 other)
279.        {
280.            return this == other;
281.        }
282.
283.        public override int GetHashCode()
284.        {
285.            return (int)(this.X + this.Y + this.Z);
286.        }
287.
288.        public static Vector3 Hermite(Vector3 value1, Vector3 tangent1, Vector3 value2, Vector3
tangent2, double amount)
289.        {
290.            Vector3 result = new Vector3();
291.            Hermite(ref value1, ref tangent1, ref value2, ref tangent2, amount, out result);
292.            return result;
293.        }
294.
295.        public static void Hermite(ref Vector3 value1, ref Vector3 tangent1, ref Vector3 value2, ref
Vector3 tangent2, double amount, out Vector3 result)
296.        {
297.            result.X = MathHelper.Hermite(value1.X, tangent1.X, value2.X, tangent2.X, amount);
298.            result.Y = MathHelper.Hermite(value1.Y, tangent1.Y, value2.Y, tangent2.Y, amount);
```

```
299.            result.Z = MathHelper.Hermite(value1.Z, tangent1.Z, value2.Z, tangent2.Z, amount);
300.        }
301.
302.        public double Length()
303.        {
304.            double result;
305.            DistanceSquared(ref this, ref zero, out result);
306.            return (double)Math.Sqrt(result);
307.        }
308.
309.        public double LengthSquared()
310.        {
311.            double result;
312.            DistanceSquared(ref this, ref zero, out result);
313.            return result;
314.        }
315.
316.        public static Vector3 Lerp(Vector3 value1, Vector3 value2, double amount)
317.        {
318.            return new Vector3(
319.                MathHelper.Lerp(value1.X, value2.X, amount),
320.                MathHelper.Lerp(value1.Y, value2.Y, amount),
321.                MathHelper.Lerp(value1.Z, value2.Z, amount));
322.        }
323.
324.        public static void Lerp(ref Vector3 value1, ref Vector3 value2, double amount, out Vector3
result)
325.        {
326.            result = new Vector3(
327.                MathHelper.Lerp(value1.X, value2.X, amount),
328.                MathHelper.Lerp(value1.Y, value2.Y, amount),
```

```
329.            MathHelper.Lerp(value1.Z, value2.Z, amount));
330.        }
331.
332.        public static Vector3 Max(Vector3 value1, Vector3 value2)
333.        {
334.            return new Vector3(
335.                MathHelper.Max(value1.X, value2.X),
336.                MathHelper.Max(value1.Y, value2.Y),
337.                MathHelper.Max(value1.Z, value2.Z));
338.        }
339.
340.        public static void Max(ref Vector3 value1, ref Vector3 value2, out Vector3 result)
341.        {
342.            result = new Vector3(
343.                MathHelper.Max(value1.X, value2.X),
344.                MathHelper.Max(value1.Y, value2.Y),
345.                MathHelper.Max(value1.Z, value2.Z));
346.        }
347.
348.        public static Vector3 Min(Vector3 value1, Vector3 value2)
349.        {
350.            return new Vector3(
351.                MathHelper.Min(value1.X, value2.X),
352.                MathHelper.Min(value1.Y, value2.Y),
353.                MathHelper.Min(value1.Z, value2.Z));
354.        }
355.
356.        public static void Min(ref Vector3 value1, ref Vector3 value2, out Vector3 result)
357.        {
358.            result = new Vector3(
359.                MathHelper.Min(value1.X, value2.X),
```

```
360.            MathHelper.Min(value1.Y, value2.Y),
361.            MathHelper.Min(value1.Z, value2.Z));
362.        }
363.
364.        public static Vector3 Multiply(Vector3 value1, Vector3 value2)
365.        {
366.            value1.X *= value2.X;
367.            value1.Y *= value2.Y;
368.            value1.Z *= value2.Z;
369.            return value1;
370.        }
371.
372.        public static Vector3 Multiply(Vector3 value1, double scaleFactor)
373.        {
374.            value1.X *= scaleFactor;
375.            value1.Y *= scaleFactor;
376.            value1.Z *= scaleFactor;
377.            return value1;
378.        }
379.
380.        public static void Multiply(ref Vector3 value1, double scaleFactor, out Vector3 result)
381.        {
382.            result.X = value1.X * scaleFactor;
383.            result.Y = value1.Y * scaleFactor;
384.            result.Z = value1.Z * scaleFactor;
385.        }
386.
387.        public static void Multiply(ref Vector3 value1, ref Vector3 value2, out Vector3 result)
388.        {
389.            result.X = value1.X * value2.X;
390.            result.Y = value1.Y * value2.Y;
```

```
391.            result.Z = value1.Z * value2.Z;
392.        }
393.
394.        public static Vector3 Negate(Vector3 value)
395.        {
396.            value = new Vector3(-value.X, -value.Y, -value.Z);
397.            return value;
398.        }
399.
400.        public static void Negate(ref Vector3 value, out Vector3 result)
401.        {
402.            result = new Vector3(-value.X, -value.Y, -value.Z);
403.        }
404.
405.        public void Normalize()
406.        {
407.            Normalize(ref this, out this);
408.        }
409.
410.        public static Vector3 Normalize(Vector3 vector)
411.        {
412.            Normalize(ref vector, out vector);
413.            return vector;
414.        }
415.
416.        public static void Normalize(ref Vector3 value, out Vector3 result)
417.        {
418.            double factor;
419.            Distance(ref value, ref zero, out factor);
420.            factor = 1f / factor;
421.            result.X = value.X * factor;
```

```
422.            result.Y = value.Y * factor;
423.            result.Z = value.Z * factor;
424.        }
425.
426.    public static Vector3 Reflect(Vector3 vector, Vector3 normal)
427.    {
428.        // I is the original array
429.        // N is the normal of the incident plane
430.        // R = I - (2 * N * ( DotProduct[ I,N] ))
431.        Vector3 reflectedVector;
432.        // inline the dotProduct here instead of calling method
433.        double dotProduct = ((vector.X * normal.X) + (vector.Y * normal.Y)) + (vector.Z * normal.Z);
434.        reflectedVector.X = vector.X - (2.0f * normal.X) * dotProduct;
435.        reflectedVector.Y = vector.Y - (2.0f * normal.Y) * dotProduct;
436.        reflectedVector.Z = vector.Z - (2.0f * normal.Z) * dotProduct;
437.
438.        return reflectedVector;
439.    }
440.
441.    public static void Reflect(ref Vector3 vector, ref Vector3 normal, out Vector3 result)
442.    {
443.        // I is the original array
444.        // N is the normal of the incident plane
445.        // R = I - (2 * N * ( DotProduct[ I,N] ))
446.
447.        // inline the dotProduct here instead of calling method
448.        double dotProduct = ((vector.X * normal.X) + (vector.Y * normal.Y)) + (vector.Z * normal.Z);
449.        result.X = vector.X - (2.0f * normal.X) * dotProduct;
450.        result.Y = vector.Y - (2.0f * normal.Y) * dotProduct;
451.        result.Z = vector.Z - (2.0f * normal.Z) * dotProduct;
452.
```

```
453.    }
454.
455.        public static Vector3 SmoothStep(Vector3 value1, Vector3 value2, double amount)
456.        {
457.            return new Vector3(
458.                MathHelper.SmoothStep(value1.X, value2.X, amount),
459.                MathHelper.SmoothStep(value1.Y, value2.Y, amount),
460.                MathHelper.SmoothStep(value1.Z, value2.Z, amount));
461.        }
462.
463.        public static void SmoothStep(ref Vector3 value1, ref Vector3 value2, double amount, out
Vector3 result)
464.        {
465.            result = new Vector3(
466.                MathHelper.SmoothStep(value1.X, value2.X, amount),
467.                MathHelper.SmoothStep(value1.Y, value2.Y, amount),
468.                MathHelper.SmoothStep(value1.Z, value2.Z, amount));
469.        }
470.
471.        public static Vector3 Subtract(Vector3 value1, Vector3 value2)
472.        {
473.            value1.X -= value2.X;
474.            value1.Y -= value2.Y;
475.            value1.Z -= value2.Z;
476.            return value1;
477.        }
478.
479.        public static void Subtract(ref Vector3 value1, ref Vector3 value2, out Vector3 result)
480.        {
481.            result.X = value1.X - value2.X;
482.            result.Y = value1.Y - value2.Y;
```

```
483.            result.Z = value1.Z - value2.Z;
484.        }
485.
486.        public override string ToString()
487.        {
488.            StringBuilder sb = new StringBuilder(32);
489.            sb.Append("{X:");
490.            sb.Append(this.X);
491.            sb.Append(" Y:");
492.            sb.Append(this.Y);
493.            sb.Append(" Z:");
494.            sb.Append(this.Z);
495.            sb.Append("}");
496.            return sb.ToString();
497.        }
498.
499.        public static Vector3 Transform(Vector3 position, Matrix matrix)
500.        {
501.            Transform(ref position, ref matrix, out position);
502.            return position;
503.        }
504.
505.        public static void Transform(ref Vector3 position, ref Matrix matrix, out Vector3 result)
506.        {
507.            result = new Vector3((position.X * matrix.M11) + (position.Y * matrix.M21) + (position.Z
* matrix.M31) + matrix.M41,
508.                                 (position.X * matrix.M12) + (position.Y * matrix.M22) + (position.Z *
matrix.M32) + matrix.M42,
509.                                 (position.X * matrix.M13) + (position.Y * matrix.M23) + (position.Z *
matrix.M33) + matrix.M43);
510.        }
511.
```

```
512.        public static void Transform(Vector3[] sourceArray, ref Matrix matrix, Vector3[]
destinationArray)
513.        {
514.            Debug.Assert(destinationArray.Length >= sourceArray.Length, "The destination array is
smaller than the source array.");
515.
516.            // TODO: Are there options on some platforms to implement a vectorized version of this?
517.
518.            for (var i = 0; i < sourceArray.Length; i++)
519.            {
520.                var position = sourceArray[i];
521.                destinationArray[i] =
522.                    new Vector3(
523.                        (position.X*matrix.M11) + (position.Y*matrix.M21) + (position.Z*matrix.M31) +
matrix.M41,
524.                        (position.X*matrix.M12) + (position.Y*matrix.M22) + (position.Z*matrix.M32) +
matrix.M42,
525.                        (position.X*matrix.M13) + (position.Y*matrix.M23) + (position.Z*matrix.M33) +
matrix.M43);
526.            }
527.        }
528.
529.    /// <summary>
530.        /// Transforms a vector by a quaternion rotation.
531.        /// </summary>
532.        /// <param name="vec">The vector to transform.</param>
533.        /// <param name="quat">The quaternion to rotate the vector by.</param>
534.        /// <returns>The result of the operation.</returns>
535.        public static Vector3 Transform(Vector3 vec, Quaternion quat)
536.        {
537.            Vector3 result;
538.            Transform(ref vec, ref quat, out result);
539.            return result;
```

```
540.        }
541.
542.        /// <summary>
543.        /// Transforms a vector by a quaternion rotation.
544.        /// </summary>
545.        /// <param name="vec">The vector to transform.</param>
546.        /// <param name="quat">The quaternion to rotate the vector by.</param>
547.        /// <param name="result">The result of the operation.</param>
548. //      public static void Transform(ref Vector3 vec, ref Quaternion quat, out Vector3 result)
549. //      {
550. //      // Taken from the OpentTK implementation of Vector3
551. //          // Since vec.W == 0, we can optimize quat * vec * quat^-1 as follows:
552. //          // vec + 2.0 * cross(quat.xyz, cross(quat.xyz, vec) + quat.w * vec)
553. //          Vector3 xyz = quat.Xyz, temp, temp2;
554. //          Vector3.Cross(ref xyz, ref vec, out temp);
555. //          Vector3.Multiply(ref vec, quat.W, out temp2);
556. //          Vector3.Add(ref temp, ref temp2, out temp);
557. //          Vector3.Cross(ref xyz, ref temp, out temp);
558. //          Vector3.Multiply(ref temp, 2, out temp);
559. //          Vector3.Add(ref vec, ref temp, out result);
560. //      }
561.
562.        /// <summary>
563.        /// Transforms a vector by a quaternion rotation.
564.        /// </summary>
565.        /// <param name="vec">The vector to transform.</param>
566.        /// <param name="quat">The quaternion to rotate the vector by.</param>
567.        /// <param name="result">The result of the operation.</param>
568.        public static void Transform(ref Vector3 vec, ref Quaternion quat, out Vector3 result)
569.        {
570.            // This has not been tested
```

```
571.         // TODO:  This could probably be unrolled so will look into it later
572.         Matrix matrix = quat.ToMatrix();
573.         Transform(ref vec, ref matrix, out result);
574.     }
575.
576.     public static Vector3 TransformNormal(Vector3 normal, Matrix matrix)
577.     {
578.         TransformNormal(ref normal, ref matrix, out normal);
579.         return normal;
580.     }
581.
582.     public static void TransformNormal(ref Vector3 normal, ref Matrix matrix, out Vector3
result)
583.     {
584.         result = new Vector3((normal.X * matrix.M11) + (normal.Y * matrix.M21) + (normal.Z *
matrix.M31),
585.                              (normal.X * matrix.M12) + (normal.Y * matrix.M22) + (normal.Z *
matrix.M32),
586.                              (normal.X * matrix.M13) + (normal.Y * matrix.M23) + (normal.Z *
matrix.M33));
587.     }
588.
589.     #endregion Public methods
590.
591.
592.     #region Operators
593.
594.     public static bool operator ==(Vector3 value1, Vector3 value2)
595.     {
596.         return value1.X == value2.X
597.             && value1.Y == value2.Y
598.             && value1.Z == value2.Z;
```

```
599.        }
600.
601.        public static bool operator !=(Vector3 value1, Vector3 value2)
602.        {
603.            return !(value1 == value2);
604.        }
605.
606.        public static Vector3 operator +(Vector3 value1, Vector3 value2)
607.        {
608.            value1.X += value2.X;
609.            value1.Y += value2.Y;
610.            value1.Z += value2.Z;
611.            return value1;
612.        }
613.
614.        public static Vector3 operator -(Vector3 value)
615.        {
616.            value = new Vector3(-value.X, -value.Y, -value.Z);
617.            return value;
618.        }
619.
620.        public static Vector3 operator -(Vector3 value1, Vector3 value2)
621.        {
622.            value1.X -= value2.X;
623.            value1.Y -= value2.Y;
624.            value1.Z -= value2.Z;
625.            return value1;
626.        }
627.
628.        public static Vector3 operator *(Vector3 value1, Vector3 value2)
629.        {
```

```
630.            value1.X *= value2.X;
631.            value1.Y *= value2.Y;
632.            value1.Z *= value2.Z;
633.            return value1;
634.        }
635.
636.        public static Vector3 operator *(Vector3 value, double scaleFactor)
637.        {
638.            value.X *= scaleFactor;
639.            value.Y *= scaleFactor;
640.            value.Z *= scaleFactor;
641.            return value;
642.        }
643.
644.        public static Vector3 operator *(double scaleFactor, Vector3 value)
645.        {
646.            value.X *= scaleFactor;
647.            value.Y *= scaleFactor;
648.            value.Z *= scaleFactor;
649.            return value;
650.        }
651.
652.        public static Vector3 operator /(Vector3 value1, Vector3 value2)
653.        {
654.            value1.X /= value2.X;
655.            value1.Y /= value2.Y;
656.            value1.Z /= value2.Z;
657.            return value1;
658.        }
659.
660.        public static Vector3 operator /(Vector3 value, double divider)
```

```
661.        {
662.            double factor = 1 / divider;
663.            value.X *= factor;
664.            value.Y *= factor;
665.            value.Z *= factor;
666.            return value;
667.        }
668.
669.        #endregion
670.    }
671. }
```