FIRST PERSON RUNNER (SIMULATION GAME)

A Project Report

Submitted by

Saniya Inamdar 1805690225

Qais Sahib 1805690213

Sohail Shaikh 1805690202

Sajiya Shaikh 1805690187

Mahek Shirgawkar 1805690214

Under the guidance of

Shaista Shaikh

(Lecturer, ComputerEngg. Dept.)

Department of Computer Engineering,

Anjuman-I-Islam's Abdul RazzakKalsekarPolytechnic,

Sector- 16, Khandagaon, New Panvel- 410206

Maharashtra State Board of Technical Education

(2017-2018)

A Project Report On

FIRST PERSON RUNNER (SIMULATION GAME)

Submitted By

Saniya Inamdar 1805690225

Qais Sahib 1805690213

Sohail Shaikh 1805690202

Sajiya Shaikh 1805690187

Mahek Shirgawkar 1805690214

In a partial fulfilment for the award of the degree

Of

Diploma in Computer Engineering

Under the guidance of

Shaista Shaikh

(Lecturer, ComputerEngg. Dept.)

ANJUMAN - I - ISLAM'S

A. R. KALSEKAR

POLYTECHNIC, NEW PANVEL

Department of Computer Engineering,

Anjuman-I-Islam's Abdul RazzakKalsekar Polytechnic, Sector- 16, Khandagaon, New Panvel- 410206

Maharashtra State Board of Technical Education

(2017-2018)

CERTIFICATE

This is to certify that the project entitled **"FIRST PERSON RUNNER GAME"** being submitted by **CO6I-A1** is worthy of consideration for the award of the degree of "Diploma in Computer Engineering and is a record of original bonafide carried out under our guidance and supervision. The results contained in this respect have not been submitted in part or full to any other university or institute for the award of any degree, diploma certificate.

Saniya Inamdar 1805690225

Qais Sahib 1805690213

Sohail Shaikh 1805690202

Sajiya Shaikh 1805690187

Mahek Shirgawkar 1805690214

(Internal Examiner) **(External Examiner)**

Ms. Shaista Shaikh

(Project Guide)

Mrs.Shaista Shaikh **Prof.Ramjan A. Khatik**

(HOD, Computer Engg. Dept.) **(Principal, AIARKP)**

Declaration (16)

I declare that this project report entitled "**SIMULATION GAME**" represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any data/fact in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name1: Saniya Inamdar 1805690225
Name2: Qais Sahib 1805690213
Name3: Sohail Shaikh 1805690202
Name4: Sajiya Shaikh 1805690187
Name 5: Mehek Shirgawkar 1805690214

Date: 29/6/2021

Place: New Panvel

Acknowledgement (16)

I consider myself lucky to work under guidance of such talented and experienced people who guided me all through the completion of my dissertation.

I express my deep sense of gratitude to my guide **Mrs.Shaista Shaikh**, Lecturer of Computer Engineering Department his generous assistance, vast knowledge, experience, views& suggestions and for giving me their gracious support. I owe a lot to them for this invaluable guidance in spite of their busy schedule.

I am grateful to **Mr.Ramjan Khatik**, Principal for his support and co-operation and for allowing me to pursue my Diploma Programme besides permitting me to use the laboratory infrastructure of the Institute.

I am thankful to to my H.O.D **Mrs.Shaista Shaikh** (Name of Project Coordinator)for his support at various stages.

Last but not the least my thanks also goes to other staff members of Computer Engineering Department, Anjuman-I-Islam's Kalsekar Polytechnic, Panvel, library staff for their assistance useful views and tips.

I also take this opportunity to thank my Friends for their support and encouragement at every stage of my life.

Date: 29/6/2021

ABSTRACT

The project is made under the guidance of **Mrs. Shaista Shaikh**. The work on the project was started from 5th semester. The title and the topic of the project were finalized by the **Computer Engineering Dept. of A.I.A.R Kalsekar Polytechnic** expert faculties.

The Title of the project is **"FIRST PERSON RUNNER"** to computerize simulation gaming. Software used for Game Development is **Unity.** The language that's used in Unity is **C# (pronounced C-sharp).** In this project our **main genre** is Sports Game. Our sub genres are FPP which is First Person Perspective view, AI which is Artificial Intelligence / NPCs which is non-player character also known as Bots in Gaming. **FPP** is the game mode in which you view the game as the character you are playing as, basically seeing only what the character sees. **AI** in gaming refers to responsive video game experience. These AI-powered interactive experiences are usually generated via non-player characters, or NPCs, that act intelligently or creatively, as if controlled by a human game-player. AI is the engine that determines an NPC's behaviour in the game world.

List of Tables

CHAPTER: 1

INTRODUCTION

What is First Person Perspective (FPP)?

FPP is short for ‘first-person perspective’ which is the game mode in which you view the game as the character you are playing as; basically seeing only what the character sees.

What Does First Person Runner (FPR) Mean?

A first person Runner (FPR) is a video game genre which involves competing in running races that is played from the point of view of the protagonist. FPR games typically map the gamer’s movements and provide a view of what an actual person would see and do in the game. FPR is a runner game where you will compete with other runners or rather bots or AI

This game will feature an athlete running a race.

There are many genre of racing video games, these are the following:-

Arcade-style racing -

- Arcade-style racing games put fun and a fast-paced experience above all else, as cars usually compete in unique ways.
- A key feature of arcade-style racers that specifically distinguishes them from simulation racers is their far more liberal physics.

Simulation Racing -

- Simulation style racing games strive to convincingly replicate the handling of an automobile.
- They often license real cars or racing leagues, but will sometimes use fantasy cars built to resemble real ones if unable to acquire an official license for them.

Kart Racing -

- Kart racing games have simplified driving mechanics while adding obstacles, unusual track designs and various action elements.
- Kart racers are also known to cast characters known from various platform games or cartoon television series as the drivers of "wacky" vehicles.

FPR Pros:

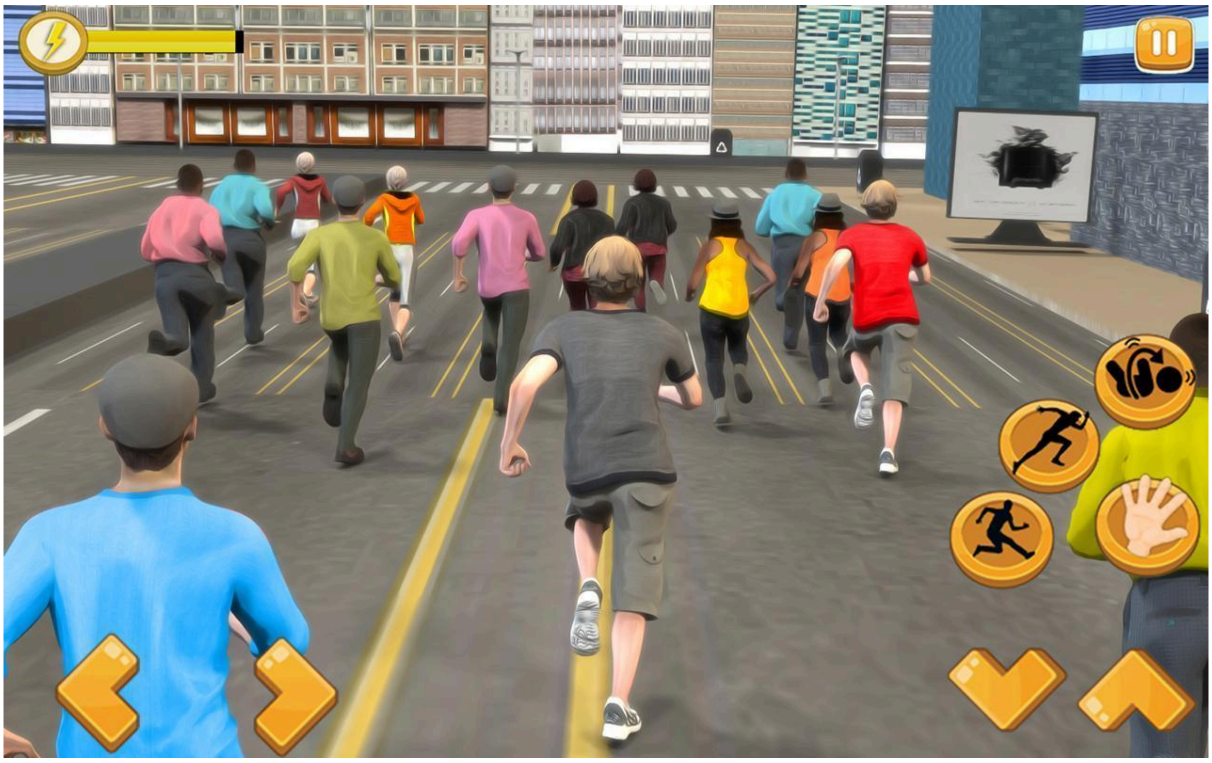
- The first-person person perspective offers a more hard-core gaming experience. It relies a lot more on aim and skill to win games
- It's also more realistic as it limits the field of view of the player to what can be seen if you were the character itself. The mode is certainly more competitive and can make for a better viewing experience, if implemented correctly.

CHAPTER: 2

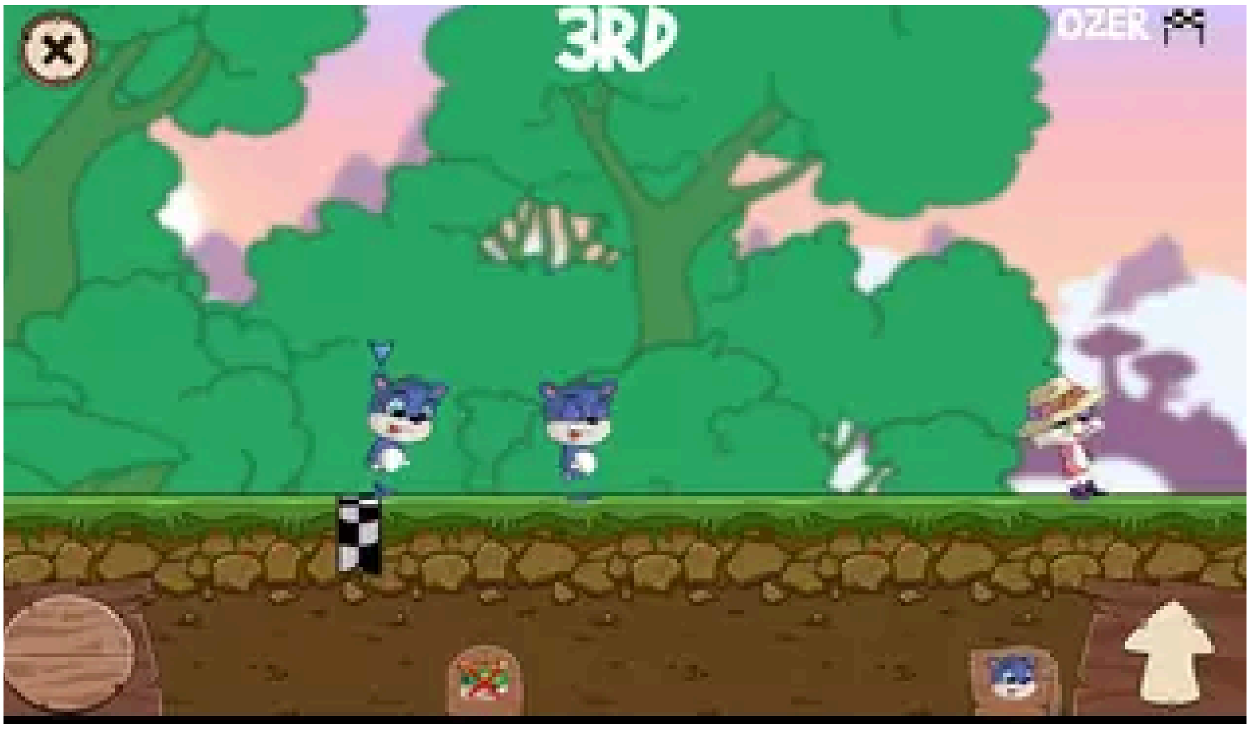
Existing System

Running Race Games only delivers a third person or 2D view which never gave the full experience to the user about racing in a stadium. To be specific there is never a game which gave a realistic running race experience.

Third person racing games look like:-



2D Running Race Games looks like:-



CHAPTER: 3

Problem Definition

We are making a game which will gave a good running race experience form the eyes of the racer. Sticking with some physics as realistic as possible

Which will more likely look like this:-

Note: - This is not the game we made

CHAPTER: 4

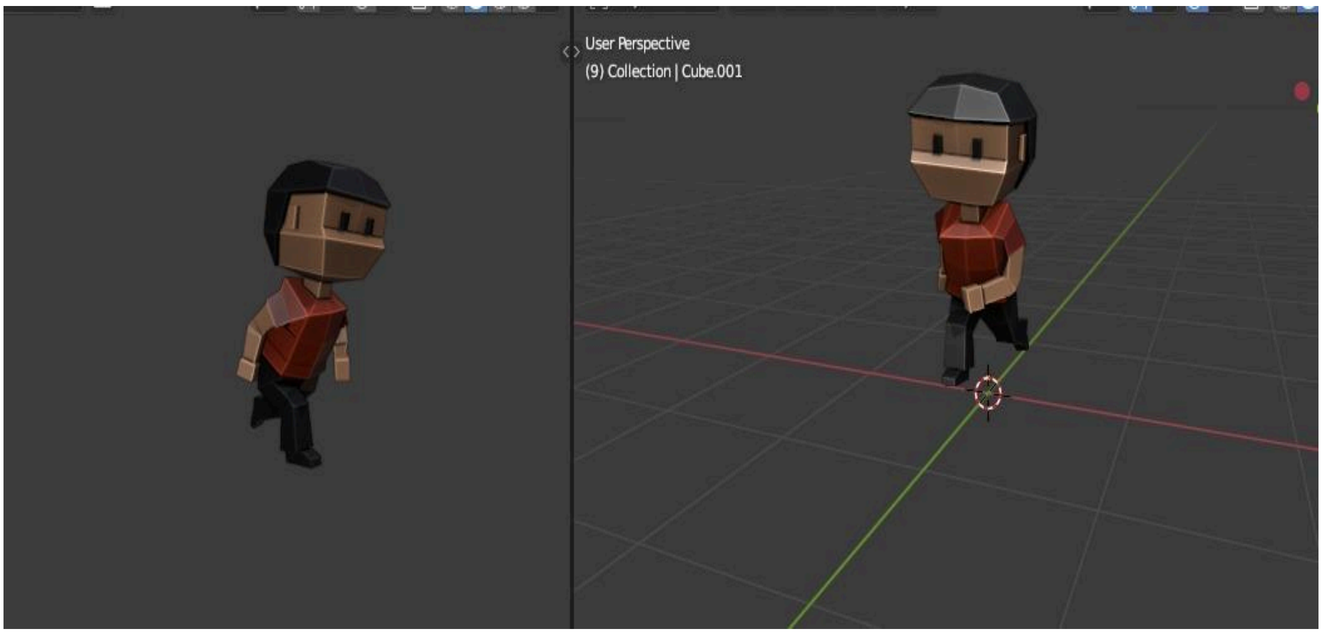
Developed System

Analysis and Design

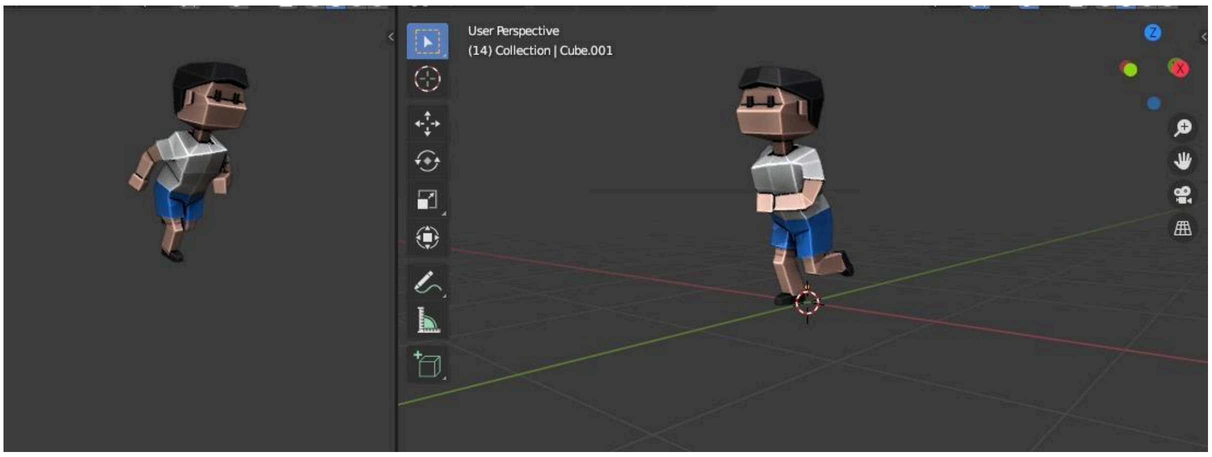
We used blender to design the game Blender is a popular program because it’s free but also robust, making its 3D modelling tools accessible to anyone who wants to learn.

These are the characters we have designed:-

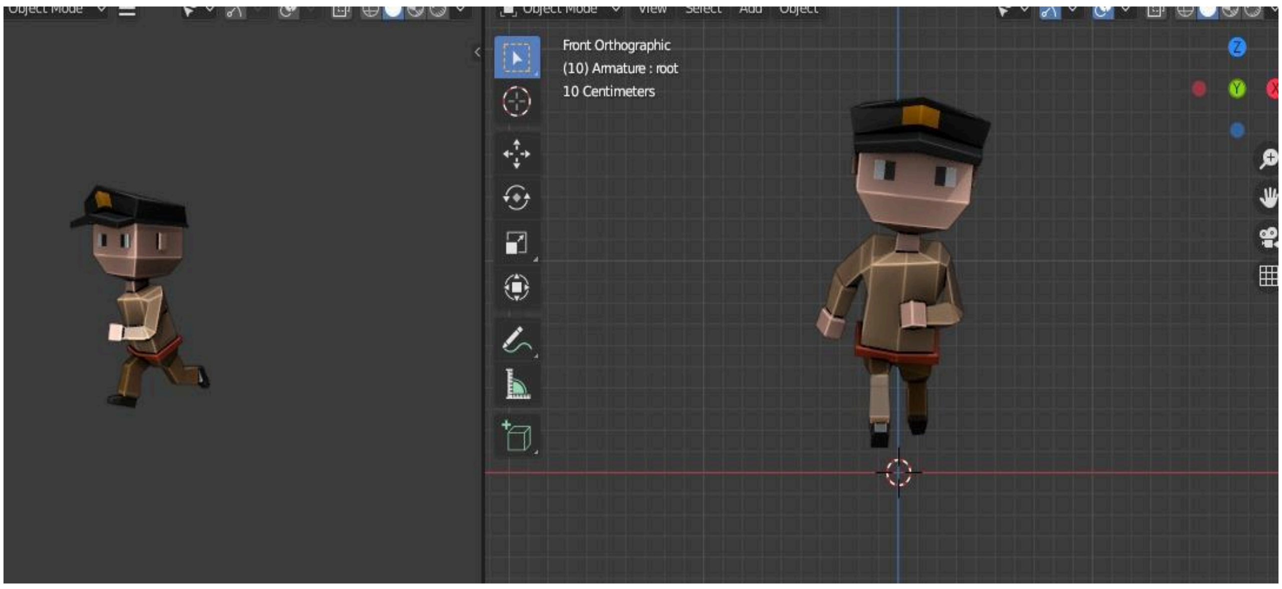
James Jacobs



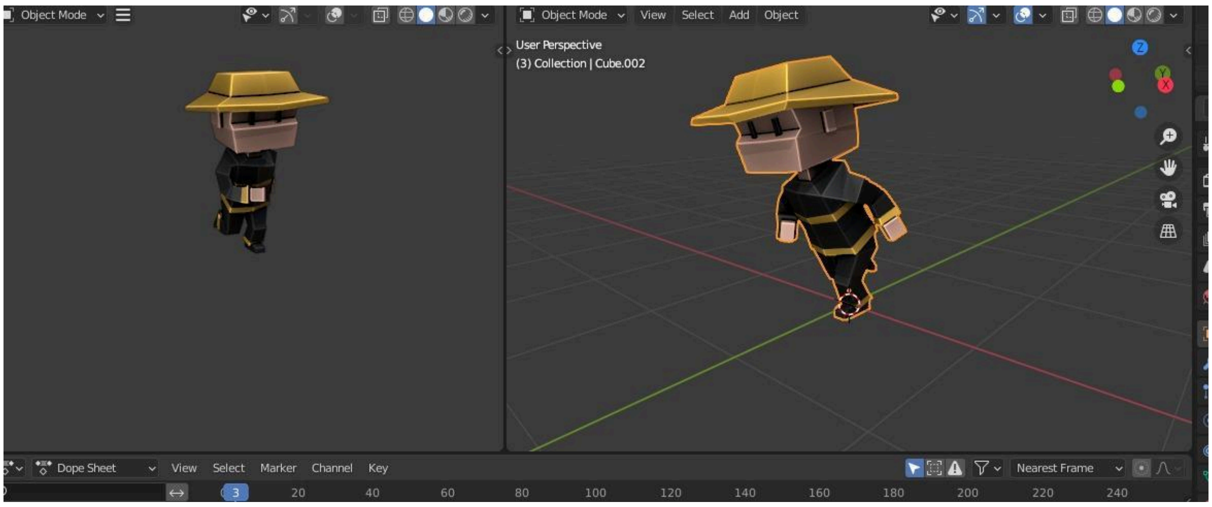
Carlos White



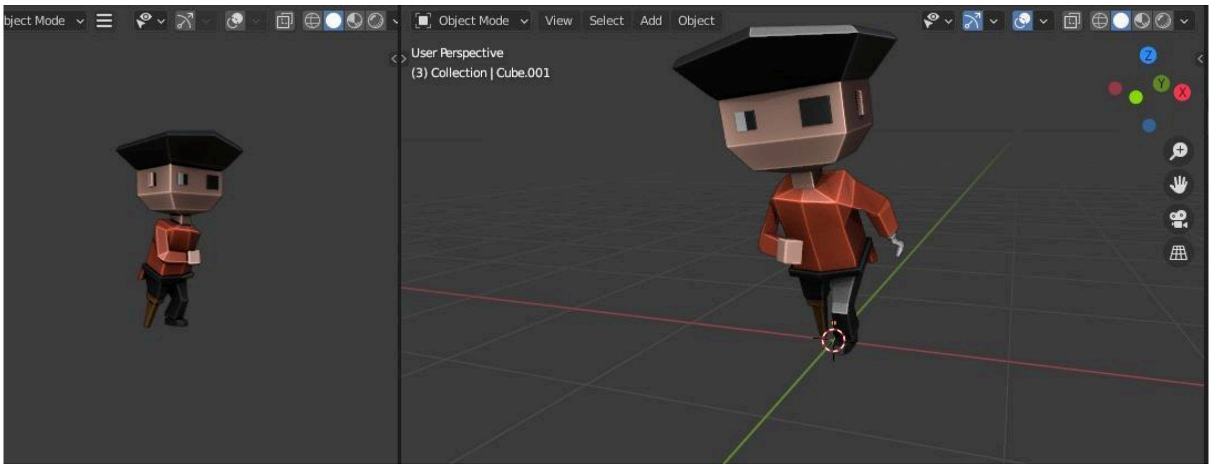
Peter Cop



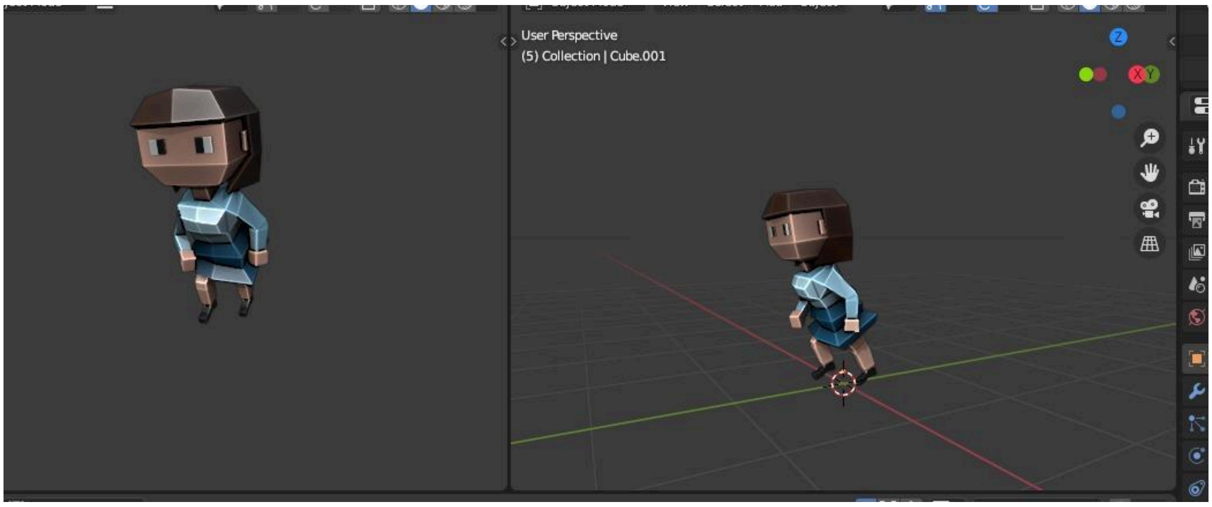
Dexter Morgan



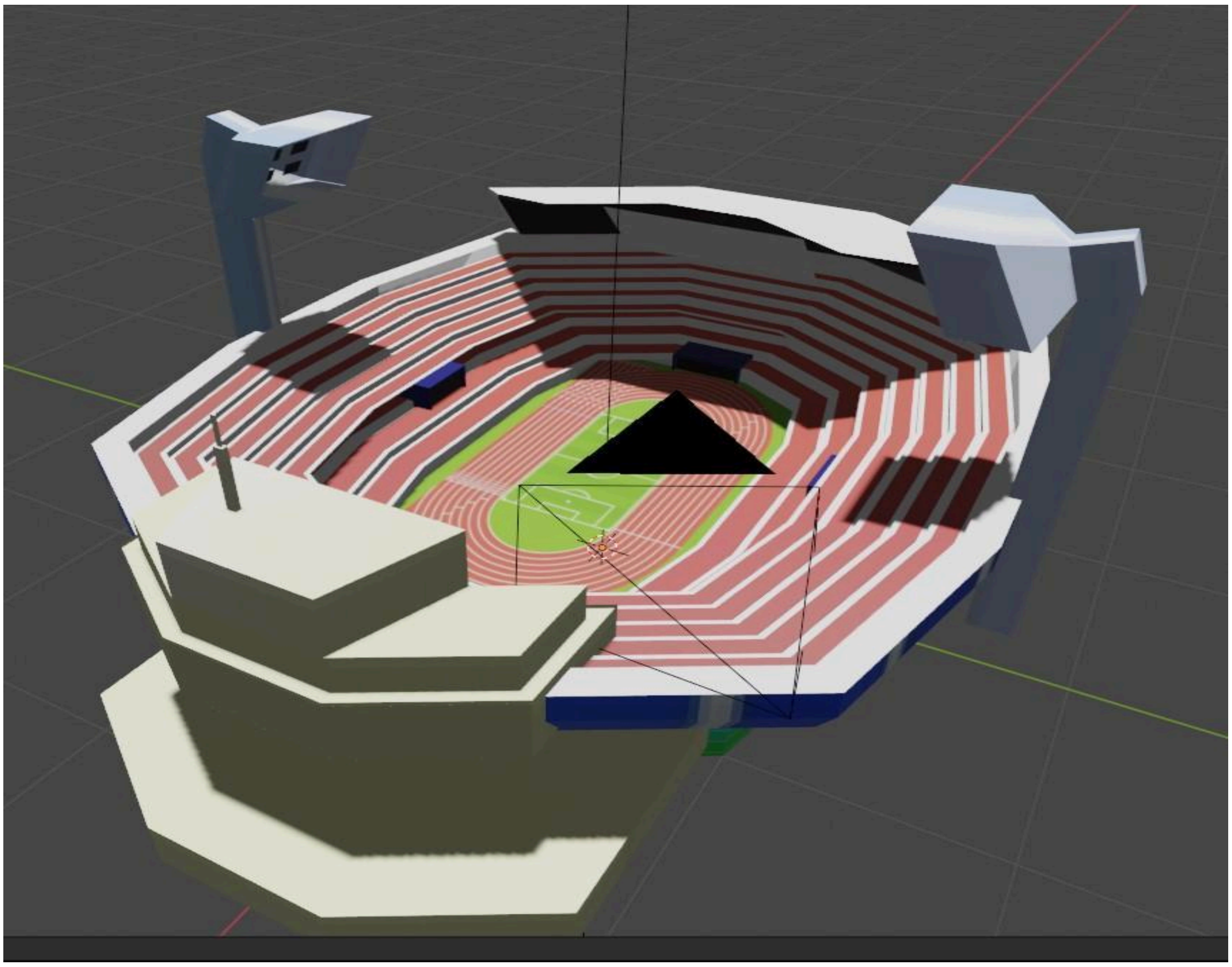
Pirate jack



Alexby



And we have the stadium



Requirements

Things which are required to make this simulator are:-

- **Blender**
- **Krita**
- **Visual Studio**
- **Unity**

Blender

- This software will help us to make our 3D assets or object (Character, environment, etc.)
- It will also help us to create character animations

Krita

- This software will allow us to give colour to our assets.
- Using this software we will give textures or images to the Objects we use in this project.

Visual Studio

- We write our program in visual studio.
- The program we are going to use is C#.
- Program will be used to make the key binding and to make an animation is a loop or if there is a condition the player must finish to get to next level.

Unity

- Unity is a software where we will integrate our program with the objects or assets.
- Here is where the object will come to life and interact with other object.
- With this software this game will be actually made.

Implementation

We have implement all the characters and environment assets in Unity with the following steps:-

- Save your .blend file in your project's Assets folder.
- Switch back into Unity, the file is imported automatically

After implementation we have created path for the characters to run in specific speed in Visual Studios using C#.

Path Creator Program

```
using System.Collections.Generic;

using PathCreation;

using PathCreation.Utility;

using UnityEditor;

using UnityEditor.IMGUI.Controls;

using UnityEngine;


namespace PathCreationEditor {

    /// Editor class for the creation of Bezier and Vertex paths


    [CustomEditor (typeof (PathCreator))]
    public class PathEditor : Editor {

        #region Fields

        // Interaction:
        const float segmentSelectDistanceThreshold = 10f;
        const float screenPolylineMaxAngleError = .3f;
        const float screenPolylineMinVertexDst = .01f;

        // Help messages:
        const string helpInfo = "Shift-click to add or insert new points. Control-click to delete
points. For more detailed infomation, please refer to the documentation.";
        static readonly string[] spaceNames = { "3D (xyz)", "2D (xy)", "Top-down (xz)" };
        static readonly string[] tabNames = { "Bézier Path", "Vertex Path" };
        const string constantSizeTooltip = "If true, anchor and control points will keep a
constant size when zooming in the editor.";

        // Display
        const int inspectorSectionSpacing = 10;
        const float constantHandleScale = .01f;
        const float normalsSpacing = .2f;
        GUIStyle boldFoldoutStyle;
```

```
// References:

PathCreator creator;

Editor globalDisplaySettingsEditor;

ScreenSpacePolyLine screenSpaceLine;

ScreenSpacePolyLine.MouseInfo pathMouseInfo;

GlobalDisplaySettings globalDisplaySettings;

PathHandle.HandleColours splineAnchorColours;

PathHandle.HandleColours splineControlColours;

Dictionary<GlobalDisplaySettings.HandleType, Handles.CapFunction> capFunctions;

ArcHandle anchorAngleHandle = new ArcHandle ();

VertexPath normalsVertexPath;


// State variables:

int selectedSegmentIndex;

int draggingHandleIndex;

int mouseOverHandleIndex;

int handleIndexToDisplayAsTransform;


bool shiftLastFrame;

bool hasUpdatedScreenSpaceLine;

bool hasUpdatedNormalsVertexPath;

bool editingNormalsOld;
```

```
Vector3 transformPos;

Vector3 transformScale;

Quaternion transformRot;


Color handlesStartCol;


// Constants

const int bezierPathTab = 0;

const int vertexPathTab = 1;


#endregion


#region Inspectors


public override void OnInspectorGUI () {

   // Initialize GUI styles

   if (boldFoldoutStyle == null) {

      boldFoldoutStyle = new GUIStyle (EditorStyles.foldout);

      boldFoldoutStyle.fontStyle = FontStyle.Bold;

   }
```

```
        Undo.RecordObject (creator, "Path settings changed");


        // Draw Bezier and Vertex tabs

        int tabIndex = GUILayout.Toolbar (data.tabIndex, tabNames);

        if (tabIndex != data.tabIndex) {

            data.tabIndex = tabIndex;

            TabChanged ();

        }


        // Draw inspector for active tab

        switch (data.tabIndex) {

            case bezierPathTab:

                DrawBezierPathInspector ();

                break;

            case vertexPathTab:

                DrawVertexPathInspector ();

                break;

        }


        // Notify of undo/redo that might modify the path

        if (Event.current.type == EventType.ValidateCommand &&
Event.current.commandName == "UndoRedoPerformed") {
```

```
            data.PathModifiedByUndo ();
        }
    }

    void DrawBezierPathInspector () {
        using (var check = new EditorGUI.ChangeCheckScope ()) {
            // Path options:
            data.showPathOptions = EditorGUILayout.Foldout (data.showPathOptions, new
GUIContent ("Bézier Path Options"), true, boldFoldoutStyle);
            if (data.showPathOptions) {
                bezierPath.Space = (PathSpace) EditorGUILayout.Popup ("Space", (int)
bezierPath.Space, spaceNames);
                bezierPath.ControlPointMode = (BezierPath.ControlMode)
EditorGUILayout.EnumPopup (new GUIContent ("Control Mode"),
bezierPath.ControlPointMode);
                if (bezierPath.ControlPointMode == BezierPath.ControlMode.Automatic) {
                    bezierPath.AutoControlLength = EditorGUILayout.Slider (new GUIContent
("Control Spacing"), bezierPath.AutoControlLength, 0, 1);
                }

                bezierPath.IsClosed = EditorGUILayout.Toggle ("Closed Path",
bezierPath.IsClosed);
                data.showTransformTool = EditorGUILayout.Toggle (new GUIContent ("Enable
Transforms"), data.showTransformTool);
```

```
            Tools.hidden = !data.showTransformTool;


            // Check if out of bounds (can occur after undo operations)
            if (handleIndexToDisplayAsTransform >= bezierPath.NumPoints) {
                handleIndexToDisplayAsTransform = -1;
            }


            // If a point has been selected
            if (handleIndexToDisplayAsTransform != -1) {
                EditorGUILayout.LabelField ("Selected Point:");


                using (new EditorGUI.IndentLevelScope ()) {
                    var currentPosition =
creator.bezierPath[handleIndexToDisplayAsTransform];
                    var newPosition = EditorGUILayout.Vector3Field ("Position",
currentPosition);
                    if (newPosition != currentPosition) {
                        Undo.RecordObject (creator, "Move point");
                        creator.bezierPath.MovePoint (handleIndexToDisplayAsTransform,
newPosition);
                    }
                    // Don't draw the angle field if we aren't selecting an anchor point/not in 3d
space
                    if (handleIndexToDisplayAsTransform % 3 == 0 &&
creator.bezierPath.Space == PathSpace.xyz) {
                        var anchorIndex = handleIndexToDisplayAsTransform / 3;
                        var currentAngle = creator.bezierPath.GetAnchorNormalAngle
(anchorIndex);
                        var newAngle = EditorGUILayout.FloatField ("Angle", currentAngle);
                        if (newAngle != currentAngle) {
                            Undo.RecordObject (creator, "Set Angle");
                            creator.bezierPath.SetAnchorNormalAngle (anchorIndex, newAngle);
                        }
                    }
                }
            }

            if (data.showTransformTool & (handleIndexToDisplayAsTransform == -1)) {
                if (GUILayout.Button ("Centre Transform")) {

                    Vector3 worldCentre = bezierPath.CalculateBoundsWithTransform
(creator.transform).center;
                    Vector3 transformPos = creator.transform.position;
                    if (bezierPath.Space == PathSpace.xy) {
```

```
            transformPos = new Vector3 (transformPos.x, transformPos.y, 0);
        } else if (bezierPath.Space == PathSpace.xz) {
            transformPos = new Vector3 (transformPos.x, 0, transformPos.z);
        }
        Vector3 worldCentreToTransform = transformPos - worldCentre;


        if (worldCentre != creator.transform.position) {
            //Undo.RecordObject (creator, "Centralize Transform");
            if (worldCentreToTransform != Vector3.zero) {
                Vector3 localCentreToTransform =
MathUtility.InverseTransformVector (worldCentreToTransform, creator.transform,
bezierPath.Space);
                for (int i = 0; i < bezierPath.NumPoints; i++) {
                    bezierPath.SetPoint (i, bezierPath.GetPoint (i) +
localCentreToTransform, true);
                }
            }


            creator.transform.position = worldCentre;
            bezierPath.NotifyPathModified ();
        }
    }
}
```

```
                if (GUILayout.Button ("Reset Path")) {

                    Undo.RecordObject (creator, "Reset Path");

                    bool in2DEditorMode = EditorSettings.defaultBehaviorMode ==
EditorBehaviorMode.Mode2D;

                    data.ResetBezierPath (creator.transform.position, in2DEditorMode);

                    EditorApplication.QueuePlayerLoopUpdate ();

                }



                GUILayout.Space (inspectorSectionSpacing);

            }



            data.showNormals = EditorGUILayout.Foldout (data.showNormals, new
GUIContent ("Normals Options"), true, boldFoldoutStyle);

            if (data.showNormals) {

                bezierPath.FlipNormals = EditorGUILayout.Toggle (new GUIContent ("Flip
Normals"), bezierPath.FlipNormals);

                if (bezierPath.Space == PathSpace.xyz) {

                    bezierPath.GlobalNormalsAngle = EditorGUILayout.Slider (new GUIContent
("Global Angle"), bezierPath.GlobalNormalsAngle, 0, 360);



                    if (GUILayout.Button ("Reset Normals")) {

                        Undo.RecordObject (creator, "Reset Normals");
                        bezierPath.FlipNormals = false;

                        bezierPath.ResetNormalAngles ();

                    }

                }

                GUILayout.Space (inspectorSectionSpacing);

            }


            // Editor display options

            data.showDisplayOptions = EditorGUILayout.Foldout (data.showDisplayOptions,
new GUIContent ("Display Options"), true, boldFoldoutStyle);

            if (data.showDisplayOptions) {

                data.showPathBounds = GUILayout.Toggle (data.showPathBounds, new
GUIContent ("Show Path Bounds"));

                data.showPerSegmentBounds = GUILayout.Toggle
(data.showPerSegmentBounds, new GUIContent ("Show Segment Bounds"));

                data.displayAnchorPoints = GUILayout.Toggle (data.displayAnchorPoints, new
GUIContent ("Show Anchor Points"));

                if (!(bezierPath.ControlPointMode == BezierPath.ControlMode.Automatic &&
globalDisplaySettings.hideAutoControls)) {

                    data.displayControlPoints = GUILayout.Toggle (data.displayControlPoints,
new GUIContent ("Show Control Points"));

                }

                data.keepConstantHandleSize = GUILayout.Toggle
(data.keepConstantHandleSize, new GUIContent ("Constant Point Size",
constantSizeTooltip));
```

```
                data.bezierHandleScale = Mathf.Max (0, EditorGUILayout.FloatField (new
GUIContent ("Handle Scale"), data.bezierHandleScale));
                DrawGlobalDisplaySettingsInspector ();
            }


            if (check.changed) {
                SceneView.RepaintAll ();
                EditorApplication.QueuePlayerLoopUpdate ();
            }
        }
    }


    void DrawVertexPathInspector () {


        GUILayout.Space (inspectorSectionSpacing);
        EditorGUILayout.LabelField ("Vertex count: " + creator.path.NumPoints);
        GUILayout.Space (inspectorSectionSpacing);


        data.showVertexPathOptions = EditorGUILayout.Foldout
(data.showVertexPathOptions, new GUIContent ("Vertex Path Options"), true,
boldFoldoutStyle);
        if (data.showVertexPathOptions) {
            using (var check = new EditorGUI.ChangeCheckScope ()) {
                data.vertexPathMaxAngleError = EditorGUILayout.Slider (new GUIContent
("Max Angle Error"), data.vertexPathMaxAngleError, 0, 45);
                data.vertexPathMinVertexSpacing = EditorGUILayout.Slider (new GUIContent
("Min Vertex Dst"), data.vertexPathMinVertexSpacing, 0, 1);

                GUILayout.Space (inspectorSectionSpacing);
                if (check.changed) {
                    data.VertexPathSettingsChanged ();
                    SceneView.RepaintAll ();
                    EditorApplication.QueuePlayerLoopUpdate ();
                }
            }
        }

        data.showVertexPathDisplayOptions = EditorGUILayout.Foldout
(data.showVertexPathDisplayOptions, new GUIContent ("Display Options"), true,
boldFoldoutStyle);
        if (data.showVertexPathDisplayOptions) {
            using (var check = new EditorGUI.ChangeCheckScope ()) {
                data.showNormalsInVertexMode = GUILayout.Toggle
(data.showNormalsInVertexMode, new GUIContent ("Show Normals"));
                data.showBezierPathInVertexMode = GUILayout.Toggle
(data.showBezierPathInVertexMode, new GUIContent ("Show Bezier Path"));
```

```
                if (check.changed) {
                    SceneView.RepaintAll ();
                    EditorApplication.QueuePlayerLoopUpdate ();
                }
            }
            DrawGlobalDisplaySettingsInspector ();
        }
    }

    void DrawGlobalDisplaySettingsInspector () {
        using (var check = new EditorGUI.ChangeCheckScope ()) {
            data.globalDisplaySettingsFoldout = EditorGUILayout.InspectorTitlebar
(data.globalDisplaySettingsFoldout, globalDisplaySettings);
            if (data.globalDisplaySettingsFoldout) {
                CreateCachedEditor (globalDisplaySettings, null, ref
globalDisplaySettingsEditor);
                globalDisplaySettingsEditor.OnInspectorGUI ();
            }
            if (check.changed) {
                UpdateGlobalDisplaySettings ();
                SceneView.RepaintAll ();
            }
        }
```

```
}

#endregion

#region Scene GUI

void OnSceneGUI () {
   if (!globalDisplaySettings.visibleBehindObjects) {
      Handles.zTest = UnityEngine.Rendering.CompareFunction.LessEqual;
   }

   EventType eventType = Event.current.type;

   using (var check = new EditorGUI.ChangeCheckScope ()) {
      handlesStartCol = Handles.color;
      switch (data.tabIndex) {
         case bezierPathTab:
            if (eventType != EventType.Repaint && eventType != EventType.Layout) {
               ProcessBezierPathInput (Event.current);
            }

            DrawBezierPathSceneEditor ();
```

```
                break;
            case vertexPathTab:
                if (eventType == EventType.Repaint) {
                    DrawVertexPathSceneEditor ();
                }
                break;
        }

        // Don't allow clicking over empty space to deselect the object
        if (eventType == EventType.Layout) {
            HandleUtility.AddDefaultControl (0);
        }

        if (check.changed) {
            EditorApplication.QueuePlayerLoopUpdate ();
        }
    }

    SetTransformState ();
}

void DrawVertexPathSceneEditor () {
```

```
Color bezierCol = globalDisplaySettings.bezierPath;
bezierCol.a *= .5f;

if (data.showBezierPathInVertexMode) {
   for (int i = 0; i < bezierPath.NumSegments; i++) {
      Vector3[] points = bezierPath.GetPointsInSegment (i);
      for (int j = 0; j < points.Length; j++) {
         points[j] = MathUtility.TransformPoint (points[j], creator.transform,
bezierPath.Space);
      }
      Handles.DrawBezier (points[0], points[3], points[1], points[2], bezierCol, null,
2);
   }
}

Handles.color = globalDisplaySettings.vertexPath;

for (int i = 0; i < creator.path.NumPoints; i++) {
   int nextIndex = (i + 1) % creator.path.NumPoints;
   if (nextIndex != 0 || bezierPath.IsClosed) {
      Handles.DrawLine (creator.path.GetPoint (i), creator.path.GetPoint (nextIndex));
   }
```

```
        }


        if (data.showNormalsInVertexMode) {
            Handles.color = globalDisplaySettings.normals;
            Vector3[] normalLines = new Vector3[creator.path.NumPoints * 2];
            for (int i = 0; i < creator.path.NumPoints; i++) {
                normalLines[i * 2] = creator.path.GetPoint (i);
                normalLines[i * 2 + 1] = creator.path.GetPoint (i) + creator.path.localNormals[i]
* globalDisplaySettings.normalsLength;
            }
            Handles.DrawLines (normalLines);
        }
    }


    void ProcessBezierPathInput (Event e) {
        // Find which handle mouse is over. Start by looking at previous handle index first, as
most likely to still be closest to mouse
        int previousMouseOverHandleIndex = (mouseOverHandleIndex == -1) ? 0 :
mouseOverHandleIndex;
        mouseOverHandleIndex = -1;
        for (int i = 0; i < bezierPath.NumPoints; i += 3) {

            int handleIndex = (previousMouseOverHandleIndex + i) % bezierPath.NumPoints;

            float handleRadius = GetHandleDiameter (globalDisplaySettings.anchorSize *
data.bezierHandleScale, bezierPath[handleIndex]) / 2f;

            Vector3 pos = MathUtility.TransformPoint (bezierPath[handleIndex],
creator.transform, bezierPath.Space);

            float dst = HandleUtility.DistanceToCircle (pos, handleRadius);

            if (dst == 0) {

                mouseOverHandleIndex = handleIndex;

                break;

            }

        }


        // Shift-left click (when mouse not over a handle) to split or add segment

        if (mouseOverHandleIndex == -1) {

            if (e.type == EventType.MouseDown && e.button == 0 && e.shift) {

                UpdatePathMouseInfo ();

                // Insert point along selected segment

                if (selectedSegmentIndex != -1 && selectedSegmentIndex <
bezierPath.NumSegments) {

                    Vector3 newPathPoint = pathMouseInfo.closestWorldPointToMouse;

                    newPathPoint = MathUtility.InverseTransformPoint (newPathPoint,
creator.transform, bezierPath.Space);

                    Undo.RecordObject (creator, "Split segment");

                    bezierPath.SplitSegment (newPathPoint, selectedSegmentIndex,
pathMouseInfo.timeOnBezierSegment);
```

```
            }

            // If path is not a closed loop, add new point on to the end of the path

            else if (!bezierPath.IsClosed) {

                // insert new point at same dst from scene camera as the point that comes
before it (for a 3d path)

                float dstCamToEndpoint = (Camera.current.transform.position -
bezierPath[bezierPath.NumPoints - 1]).magnitude;

                Vector3 newPathPoint = MouseUtility.GetMouseWorldPosition
(bezierPath.Space, dstCamToEndpoint);

                newPathPoint = MathUtility.InverseTransformPoint (newPathPoint,
creator.transform, bezierPath.Space);


                Undo.RecordObject (creator, "Add segment");

                if (e.control || e.command) {

                    bezierPath.AddSegmentToStart (newPathPoint);

                } else {

                    bezierPath.AddSegmentToEnd (newPathPoint);

                }


            }


        }

    }
```

```
        // Control click or backspace/delete to remove point
        if (e.keyCode == KeyCode.Backspace || e.keyCode == KeyCode.Delete || ((e.control ||
e.command) && e.type == EventType.MouseDown && e.button == 0)) {

            if (mouseOverHandleIndex != -1) {
                Undo.RecordObject (creator, "Delete segment");
                bezierPath.DeleteSegment (mouseOverHandleIndex);
                if (mouseOverHandleIndex == handleIndexToDisplayAsTransform) {
                    handleIndexToDisplayAsTransform = -1;
                }
                mouseOverHandleIndex = -1;
                Repaint ();
            }
        }

        // Holding shift and moving mouse (but mouse not over a handle/dragging a handle)
        if (draggingHandleIndex == -1 && mouseOverHandleIndex == -1) {
            bool shiftDown = e.shift && !shiftLastFrame;
            if (shiftDown || ((e.type == EventType.MouseMove || e.type ==
EventType.MouseDrag) && e.shift)) {

                UpdatePathMouseInfo ();
                if (pathMouseInfo.mouseDstToLine < segmentSelectDistanceThreshold) {
                    if (pathMouseInfo.closestSegmentIndex != selectedSegmentIndex) {
                        selectedSegmentIndex = pathMouseInfo.closestSegmentIndex;
                        HandleUtility.Repaint ();
                    }
                } else {
                    selectedSegmentIndex = -1;
                    HandleUtility.Repaint ();
                }

            }
        }

        shiftLastFrame = e.shift;

    }

    void DrawBezierPathSceneEditor () {

        bool displayControlPoints = data.displayControlPoints &&
(bezierPath.ControlPointMode != BezierPath.ControlMode.Automatic ||
!globalDisplaySettings.hideAutoControls);
```

```
        Bounds bounds = bezierPath.CalculateBoundsWithTransform (creator.transform);


        if (Event.current.type == EventType.Repaint) {
            for (int i = 0; i < bezierPath.NumSegments; i++) {
                Vector3[] points = bezierPath.GetPointsInSegment (i);
                for (int j = 0; j < points.Length; j++) {
                    points[j] = MathUtility.TransformPoint (points[j], creator.transform,
bezierPath.Space);
                }


                if (data.showPerSegmentBounds) {
                    Bounds segmentBounds = CubicBezierUtility.CalculateSegmentBounds
(points[0], points[1], points[2], points[3]);
                    Handles.color = globalDisplaySettings.segmentBounds;
                    Handles.DrawWireCube (segmentBounds.center, segmentBounds.size);
                }


                // Draw lines between control points
                if (displayControlPoints) {
                    Handles.color = (bezierPath.ControlPointMode ==
BezierPath.ControlMode.Automatic) ? globalDisplaySettings.handleDisabled :
globalDisplaySettings.controlLine;
                    Handles.DrawLine (points[1], points[0]);
                    Handles.DrawLine (points[2], points[3]);
                }

                // Draw path
                bool highlightSegment = (i == selectedSegmentIndex && Event.current.shift
&& draggingHandleIndex == -1 && mouseOverHandleIndex == -1);
                Color segmentCol = (highlightSegment) ? globalDisplaySettings.highlightedPath
: globalDisplaySettings.bezierPath;
                Handles.DrawBezier (points[0], points[3], points[1], points[2], segmentCol, null,
2);
            }

            if (data.showPathBounds) {
                Handles.color = globalDisplaySettings.bounds;
                Handles.DrawWireCube (bounds.center, bounds.size);
            }

            // Draw normals
            if (data.showNormals) {
                if (!hasUpdatedNormalsVertexPath) {
                    normalsVertexPath = new VertexPath (bezierPath, creator.transform,
normalsSpacing);
                    hasUpdatedNormalsVertexPath = true;
                }

                if (editingNormalsOld != data.showNormals) {
                    editingNormalsOld = data.showNormals;
                    Repaint ();
                }

                Vector3[] normalLines = new Vector3[normalsVertexPath.NumPoints * 2];
                Handles.color = globalDisplaySettings.normals;
                for (int i = 0; i < normalsVertexPath.NumPoints; i++) {
                    normalLines[i * 2] = normalsVertexPath.GetPoint (i);
                    normalLines[i * 2 + 1] = normalsVertexPath.GetPoint (i) +
normalsVertexPath.GetNormal (i) * globalDisplaySettings.normalsLength;
                }
                Handles.DrawLines (normalLines);
            }
        }

        if (data.displayAnchorPoints) {
            for (int i = 0; i < bezierPath.NumPoints; i += 3) {
                DrawHandle (i);
            }
        }
        if (displayControlPoints) {
            for (int i = 1; i < bezierPath.NumPoints - 1; i += 3) {
                DrawHandle (i);
                DrawHandle (i + 1);
            }
        }
    }

    void DrawHandle (int i) {
        Vector3 handlePosition = MathUtility.TransformPoint (bezierPath[i],
creator.transform, bezierPath.Space);


        float anchorHandleSize = GetHandleDiameter (globalDisplaySettings.anchorSize *
data.bezierHandleScale, bezierPath[i]);
        float controlHandleSize = GetHandleDiameter (globalDisplaySettings.controlSize *
data.bezierHandleScale, bezierPath[i]);


        bool isAnchorPoint = i % 3 == 0;
        bool isInteractive = isAnchorPoint || bezierPath.ControlPointMode !=
BezierPath.ControlMode.Automatic;
        float handleSize = (isAnchorPoint) ? anchorHandleSize : controlHandleSize;
        bool doTransformHandle = i == handleIndexToDisplayAsTransform;
        PathHandle.HandleColours handleColours = (isAnchorPoint) ? splineAnchorColours : 
splineControlColours;

        if (i == handleIndexToDisplayAsTransform) {

            handleColours.defaultColour = (isAnchorPoint) ? 
globalDisplaySettings.anchorSelected : globalDisplaySettings.controlSelected;

        }

        var cap = capFunctions[(isAnchorPoint) ? globalDisplaySettings.anchorShape : 
globalDisplaySettings.controlShape];

        PathHandle.HandleInputType handleInputType;

        handlePosition = PathHandle.DrawHandle (handlePosition, bezierPath.Space, 
isInteractive, handleSize, cap, handleColours, out handleInputType, i);


        if (doTransformHandle) {

            // Show normals rotate tool

            if (data.showNormals && Tools.current == Tool.Rotate && isAnchorPoint && 
bezierPath.Space == PathSpace.xyz) {

                Handles.color = handlesStartCol;


                int attachedControlIndex = (i == bezierPath.NumPoints - 1) ? i - 1 : i + 1;

                Vector3 dir = (bezierPath[attachedControlIndex] - handlePosition).normalized;

                float handleRotOffset = (360 + bezierPath.GlobalNormalsAngle) % 360;

                anchorAngleHandle.radius = handleSize * 3;
                anchorAngleHandle.angle = handleRotOffset +
bezierPath.GetAnchorNormalAngle (i / 3);

                Vector3 handleDirection = Vector3.Cross (dir, Vector3.up);

                Matrix4x4 handleMatrix = Matrix4x4.TRS (

                    handlePosition,

                    Quaternion.LookRotation (handleDirection, dir),

                    Vector3.one

                );


                using (new Handles.DrawingScope (handleMatrix)) {

                    // draw the handle

                    EditorGUI.BeginChangeCheck ();

                    anchorAngleHandle.DrawHandle ();

                    if (EditorGUI.EndChangeCheck ()) {

                        Undo.RecordObject (creator, "Set angle");

                        bezierPath.SetAnchorNormalAngle (i / 3, anchorAngleHandle.angle -
handleRotOffset);

                    }

                }


            } else {

                handlePosition = Handles.DoPositionHandle (handlePosition,
Quaternion.identity);
```

```
        }

    }

    switch (handleInputType) {
        case PathHandle.HandleInputType.LMBDrag:
            draggingHandleIndex = i;
            handleIndexToDisplayAsTransform = -1;
            Repaint ();
            break;
        case PathHandle.HandleInputType.LMBRelease:
            draggingHandleIndex = -1;
            handleIndexToDisplayAsTransform = -1;
            Repaint ();
            break;
        case PathHandle.HandleInputType.LMBClick:
            draggingHandleIndex = -1;
            if (Event.current.shift) {
                handleIndexToDisplayAsTransform = -1; // disable move tool if new point
added
            } else {
                if (handleIndexToDisplayAsTransform == i) {
                    handleIndexToDisplayAsTransform = -1; // disable move tool if clicking on
point under move tool
                } else {
                    handleIndexToDisplayAsTransform = i;
                }
            }
            Repaint ();
            break;
        case PathHandle.HandleInputType.LMBPress:
            if (handleIndexToDisplayAsTransform != i) {
                handleIndexToDisplayAsTransform = -1;
                Repaint ();
            }
            break;
    }

    Vector3 localHandlePosition = MathUtility.InverseTransformPoint (handlePosition,
creator.transform, bezierPath.Space);

    if (bezierPath[i] != localHandlePosition) {
        Undo.RecordObject (creator, "Move point");
        bezierPath.MovePoint (i, localHandlePosition);
```

```
            }

        }

        #endregion

        #region Internal methods

        void OnDisable () {
            Tools.hidden = false;
        }

        void OnEnable () {
            creator = (PathCreator) target;
            bool in2DEditorMode = EditorSettings.defaultBehaviorMode ==
EditorBehaviorMode.Mode2D;
            creator.InitializeEditorData (in2DEditorMode);

            data.bezierCreated -= ResetState;
            data.bezierCreated += ResetState;
            Undo.undoRedoPerformed -= OnUndoRedo;
            Undo.undoRedoPerformed += OnUndoRedo;
            LoadDisplaySettings ();

            UpdateGlobalDisplaySettings ();

            ResetState ();

            SetTransformState (true);

        }


        void SetTransformState (bool initialize = false) {

            Transform t = creator.transform;

            if (!initialize) {

                if (transformPos != t.position || t.localScale != transformScale || t.rotation !=
transformRot) {

                data.PathTransformed ();

                }

            }

            transformPos = t.position;

            transformScale = t.localScale;

            transformRot = t.rotation;

        }


        void OnUndoRedo () {

            hasUpdatedScreenSpaceLine = false;
```

```
            hasUpdatedNormalsVertexPath = false;
            selectedSegmentIndex = -1;

            Repaint ();
        }

        void TabChanged () {
            SceneView.RepaintAll ();
            RepaintUnfocusedSceneViews ();
        }

        void LoadDisplaySettings () {
            globalDisplaySettings = GlobalDisplaySettings.Load ();

            capFunctions = new Dictionary<GlobalDisplaySettings.HandleType,
Handles.CapFunction> ();
            capFunctions.Add (GlobalDisplaySettings.HandleType.Circle,
Handles.CylinderHandleCap);
            capFunctions.Add (GlobalDisplaySettings.HandleType.Sphere,
Handles.SphereHandleCap);
            capFunctions.Add (GlobalDisplaySettings.HandleType.Square,
Handles.CubeHandleCap);
        }
        void UpdateGlobalDisplaySettings () {
            var gds = globalDisplaySettings;
            splineAnchorColours = new PathHandle.HandleColours (gds.anchor,
gds.anchorHighlighted, gds.anchorSelected, gds.handleDisabled);
            splineControlColours = new PathHandle.HandleColours (gds.control,
gds.controlHighlighted, gds.controlSelected, gds.handleDisabled);

            anchorAngleHandle.fillColor = new Color (1, 1, 1, .05f);
            anchorAngleHandle.wireframeColor = Color.grey;
            anchorAngleHandle.radiusHandleColor = Color.clear;
            anchorAngleHandle.angleHandleColor = Color.white;
        }

        void ResetState () {
            selectedSegmentIndex = -1;
            draggingHandleIndex = -1;
            mouseOverHandleIndex = -1;
            handleIndexToDisplayAsTransform = -1;
            hasUpdatedScreenSpaceLine = false;
            hasUpdatedNormalsVertexPath = false;

            bezierPath.OnModified -= OnPathModifed;
```

```
    bezierPath.OnModified += OnPathModifed;


    SceneView.RepaintAll ();
    EditorApplication.QueuePlayerLoopUpdate ();
}


void OnPathModifed () {
    hasUpdatedScreenSpaceLine = false;
    hasUpdatedNormalsVertexPath = false;


    RepaintUnfocusedSceneViews ();
}


void RepaintUnfocusedSceneViews () {
    // If multiple scene views are open, repaint those which do not have focus.
    if (SceneView.sceneViews.Count > 1) {
        foreach (SceneView sv in SceneView.sceneViews) {
            if (EditorWindow.focusedWindow != (EditorWindow) sv) {
                sv.Repaint ();
            }
        }
    }
```

```
        }

        void UpdatePathMouseInfo () {

            if (!hasUpdatedScreenSpaceLine || (screenSpaceLine != null &&
screenSpaceLine.TransformIsOutOfDate ())) {
                screenSpaceLine = new ScreenSpacePolyLine (bezierPath, creator.transform,
screenPolylineMaxAngleError, screenPolylineMinVertexDst);
                hasUpdatedScreenSpaceLine = true;
            }
            pathMouseInfo = screenSpaceLine.CalculateMouseInfo ();
        }

        float GetHandleDiameter (float diameter, Vector3 handlePosition) {
            float scaledDiameter = diameter * constantHandleScale;
            if (data.keepConstantHandleSize) {
                scaledDiameter *= HandleUtility.GetHandleSize (handlePosition) * 2.5f;
            }
            return scaledDiameter;
        }

        BezierPath bezierPath {
            get {
```

```
            return data.bezierPath;

         }

      }


      PathCreatorData data {

         get {

            return creator.EditorData;

         }

      }


      bool editingNormals {

         get {

            return Tools.current == Tool.Rotate && handleIndexToDisplayAsTransform % 3
== 0 && bezierPath.Space == PathSpace.xyz;

         }

      }

      #endregion

   }

}
```

CHAPTER: 5

FUTURE WORK

According to the knowledge perspective and capability of being a final year diploma students, we have created, implemented or can be said as developed our final year project which is the Simulation Game-based project named **“FIRST PERSON RUNNER”** upon that. The fact cannot be denied that it is difficult for us as not being a professional developers and diploma level students to completely create a web-based project with such features that are difficult for building and coding.

While creating the project there were some features which needed to be implemented in the project for completely developing it in all aspects were difficult to develop or build or to code. So due to the time limitations we had to skip some of features of the project which are planned to be implemented in the future.

The first feature was to create our project on an advanced level with high security and advanced features.

The second and the last feature or can be said as operation to implement in the future is to create an application of our Simulation Game-based project. The reason to create an application is that our project is based on the real time system, so for any time access and easy to access point it is been planned by us to create an application in the future.