



Project Engineering


KDash

Armen Petrosyan

Bachelor (Honours) of Software & Electronic Engineering

Galway-Mayo Institute of Technology

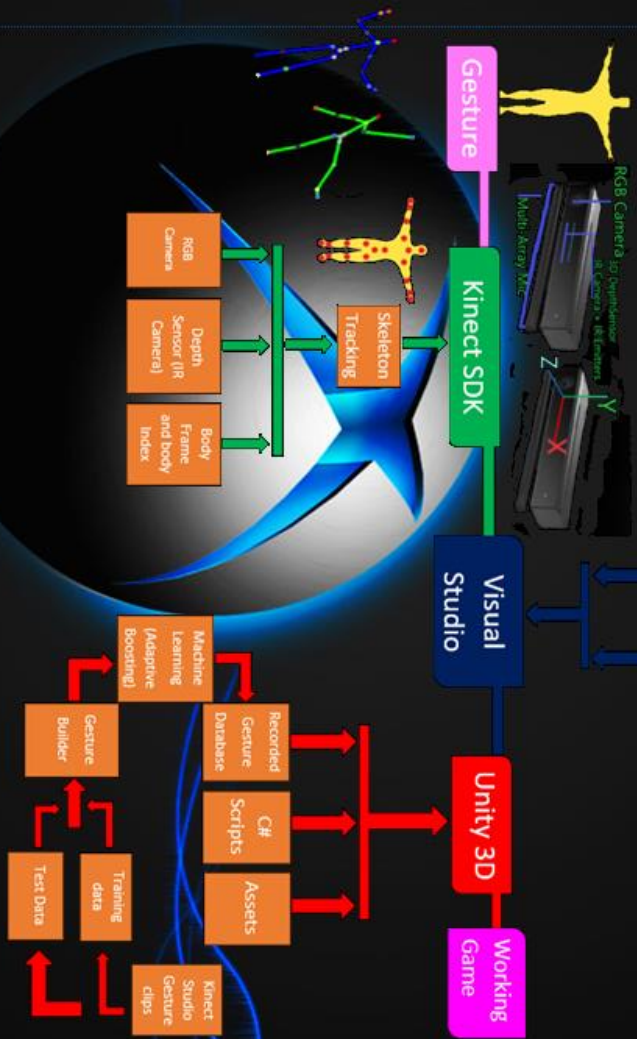
2019/2020



INSTITUTO TECNOLÓGICO NA GALTUHE-MAGHE EO
GALVIN - AMO INSTITUTE OF TECHNOLOGY

BEng (Hons) Software & Electronic Engineering

KDash



Description

KDash is a 3D Kinect game created by unity. In my project I am using the Kinect V2's Skeleton Tracking in conjunction with the Unity game engine. KDash is a endless runner game where the player controls a running character, and the character must avoid obstacles and gather candy. The Kinect records multiple gestures, which are used to interact with the game by controlling the player when he need to avoid obstacles by leaning either side or jumping.

How it works

- I made an endless 3D running map.
- A script to check when the player hit an invisible object.
- It instantiates the running map.
- Recoding gestures in Kinect Studios.
- Two sets of videos for training data and test data.
- Dumping video files in an interface called gesture builder.
- Machine learning algorithm called "Adaptive Boosting" to help find the perfect confidence and sets two Booleans.
- Built video and got a gdb (geodatabases) file which is imported into unity.
- Two Kinect scripts (Kinect Manger, Gesture Detection).
- Kinect script turns on the Kinects skeletons tracking waiting and looking for gesture detection script to recognize a gesture happening by the player which in theory moves the player by setting a Boolean value true in a moving script.
- To make the game more fun I wrote a script to auto generate candy and obstacles for the game.

Technologies

- Kinect v2
- Gesture builder
- Gestures
- Skeletal tracking
- Unity
- State machine
- Machine Learning(Adaptive Boosting)

Results

KDash is now functional in that you can use different gestures such as lean left, lean right and even jump. The character will run infinitely and all you have to do is lean and jump and the game will do the rest for you.

By: Armen Petrosyan

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Acknowledgements

I would like to express my utmost appreciation to all those who provided me the possibility to complete this report. I would like to give a special thanks to my lectures Michelle, Brian, Paul, and the other supervisors, who contribution stimulating suggestions and encouragement, helped me to coordinate my project especially in typing this report.

Furthermore, I would also like to acknowledge with appreciation, the crucial role of the staff of GMIT, who gave me permission to use all the required equipment and the necessary material to complete KDash.

Special thanks go to my friends, who gave me suggestions about my project. Finally, I must give appreciation for the guidance given by Mona and other supervisors, as well as the panels especially in our project presentation, which in turn has improved our presentation skills thanks to their comments and advices.

Table of Contents

1	Summary.....	6
2	Introduction	7
3	Project Architecture	8
4	Development Platform and Tools.....	9
5	Kinect V2.....	10
5.1	Hardware.....	10
5.2	Kinect Studio	13
5.3	Visual Gesture Builder	14
6	Unity	18
7	Web Server	24
7.1	Server	25
7.2	Client	26
8	Problem Solving	27
9	System Integration	29
10	Conclusion.....	32
11	References	33

1 Summary

KDash is a 3D Kinect game created in unity. I used the Kinect V2's Skeleton Tracking in conjunction with the Unity game engine. The goal for KDash is an endless 3D running game where the player controls a running character, and the character must avoid obstacles and gather candy. The Kinect records multiple gestures, which are used to interact with the game by controlling the player when he needs to avoid obstacles by leaning either side or jumping.

I made the endless 3D running map by creating a script to check that every time the player hit an invisible game object it clones the game map object original and returns the clone of the game map object, and adds it to the second invisible game object point at the end of the original map until the character dies or the player turns off the game.

Then by recording your gestures (leaning and jumping etc..) in Kinect studio, you need two sets of videos for training data and test data. Then you dump the video files in an interface called gesture builder. The poses of the human skeleton are recognized with the help of a machine learning algorithm called "AdaBoostTrigger" to help find the perfect confidence and set two Boolean values. Then I built the video and got a GDB (geodatabases) file which I imported into unity.

Once completed, I imported two Kinect scripts into the endless 3D running game. Then I modified the two Kinect scripts to read Kinect data coming in and recognize gestures from the GDB file. The Kinect script then turns on the Kinects skeletons tracking waiting and looking for the gesture detection script to recognize a gesture happening by the player. Once a gesture has been detected in theory then moves the Character by setting a Boolean value true in a moving script that I created to move the Character. To make the game more fun I wrote a script to auto generate candy, power ups and obstacles for the game.

2 Introduction

For my project I made KDash a 3D Kinect game created in unity. KDash is an endless 3D running game where the player controls a running character, and the character must avoid obstacles and gather candy.

The reason why I choose to do this project was because I have a huge interest in gaming and hardware so, this was my way for combing the two. Gaming is no longer just for the agile finger and thumb crowd.

I got motivated to do this project because as a child I always played the Kinect and the games you got with it for free but always wanted to make my own game using the Kinect ever since I had the Kinect v1 for my xbox360. My reason for choosing the idea of making an endless 3D running map was a game called temple run. Temple run was an endless 3D running game that players controlled an explorer who had obtained an ancient relic and is running from evil demon monkeys who are chasing him.

The idea I had with KDash Is to combine my own version of temple run and the Kinect to make an endless 3D running game where the player controls a running character, and the character must avoid obstacles, gather candy and collect power ups.

The Kinect V2 has three vital sensors that work together to detect your motion and create your physical image on the screen: an RGB camera, Depth Sensor (IR camera and IR emitter) and a multi-array microphone. In KDash I am only using two to three sensors to move character left, right and even jump.

3 Project Architecture

For my project I used the Kinect V2, gesture recognition and unity.

The Kinect V2 has three vital sensors that work together to detect your motion. It then creates your physical image on the screen: an RGB camera, Depth Sensor (IR camera and IR emitter) and a multi-array microphone. I am only using two of the three sensors in my project. The Kinect v2 can track up six skeletons at one time, each skeleton has 25 joints. Each joint has 11 properties: Colour (x, y), depth (x, y), camera (x, y, z) and orientation (x, y, z, w). The two cameras, a color (RGB) camera and a depth camera and they have different resolutions. The color camera is 1920 x 1080. The depth camera is 512 x 424.

Gesture recognition is a fundamental element when developing Kinect-based applications (I.e. Unity). Gestures are used for navigation, interaction, or data input. Some gesture examples would be Leaning, jumping, ducking, and waving. This all done through the gesture builder and Kinect SDK with videos combined with machine learning.

Unity is a cross-platform game engine developed by unity technologies it is used to make three dimensional, two-dimensional, virtual reality as well as augmented reality games. I am using unity to make a three-dimensional game so I can control a character using the Kinect combined with gestures.

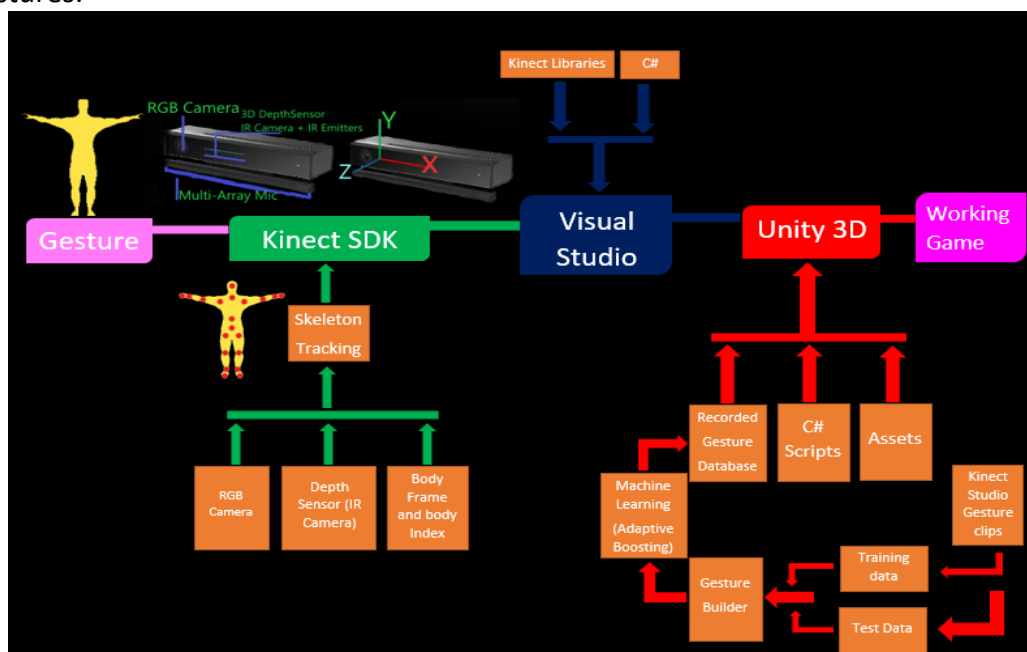


Figure 3-1 Architecture Diagram

4 Development Platform and Tools

- Unity (2019)
- Unity Asset Store
- Kinect SDK
- Kinect studio
- Kinect gesture builder.
- Visual Studios (2019)

5 Kinect V2

The only hardware I used for KDash Is the Kinect V2.

5.1 Hardware

The Kinect V2 has three vital sensors that work together to detect your motion and create your physical image on the screen: an RGB camera, Depth Sensor (IR camera and IR emitter) and a multi-array microphone.

The camera detects the red, green, and blue colour components as well as body-type and facial features. It has a pixel resolution of 1920x1080 and a frame rate of 30 fps. This helps in facial recognition and body recognition.

The depth sensor contains a monochrome CMOS sensor and infrared projector that help create the 3D imagery throughout the room. It also measures the distance of each point of the player's body by transmitting invisible near-infrared light and measuring its "time of flight" after it reflects off the objects.

The microphone is an array of four microphones that can isolate the voices of the player from other background noises allowing players to use their voices as an added control feature.

These components come together to detect and track 48 different points on each player's body and repeats 30 times every second.

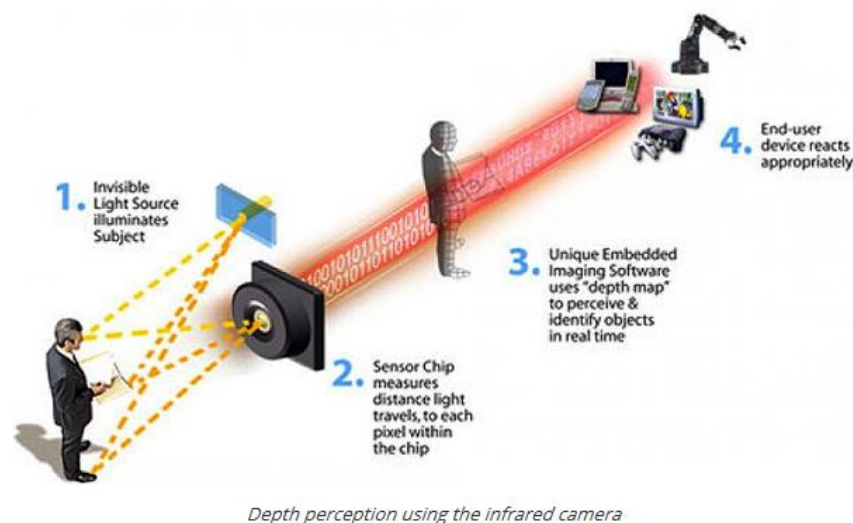


Figure 5-1-1 Depth Perception using the Infrared Camera

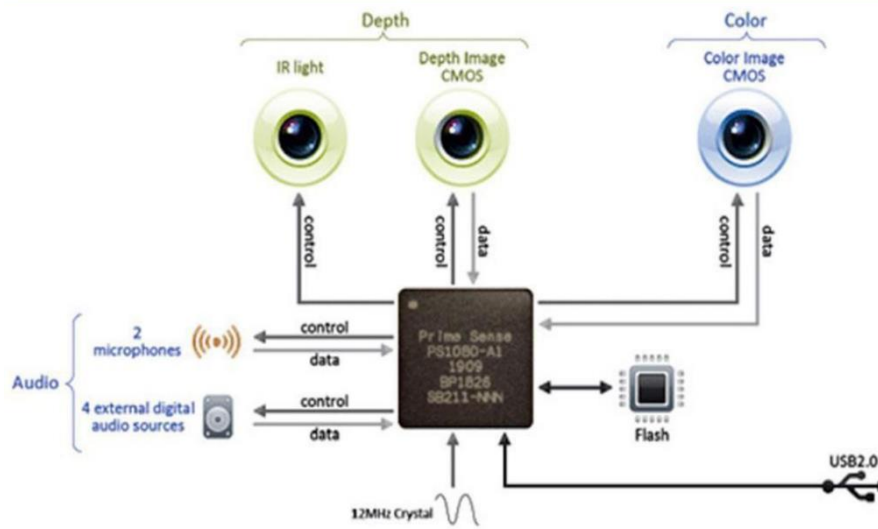


Figure 5-1-2 Kinect Anatomy



Figure 5-1-3 Kinect Scene image

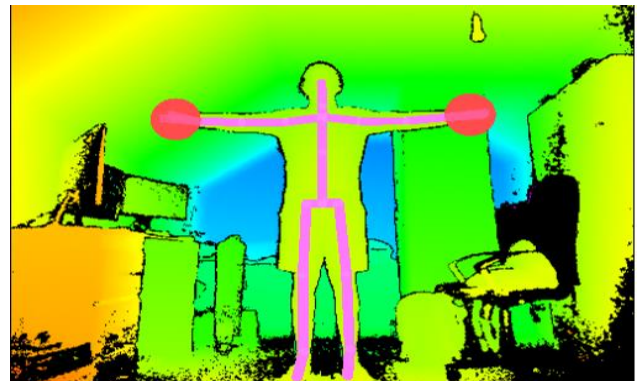


Figure 5-1-3 Kinect Scene Depth Image

```
Kinect Opened
{
  bodyIndex: 5,
  tracked: true,
  trackingId: '72057594037928992',
  leftHandState: 1,
  rightHandState: 1,
  joints: [
    {
      depthX: 0.5051928162574768,
      depthY: 0.5137186050415039,
      colorX: 0.5179657340049744,
      colorY: 0.541925847530365,
      cameraX: -0.015620222315192223,
      cameraY: -0.05089008808135986,
      cameraZ: 1.5552568435668945,
      orientationX: -0.0018134672427549958,
      orientationY: 0.9982876181602478,
      orientationZ: 0.03676481172442436,
      orientationW: -0.04546322673559189,
      jointType: 0,
      trackingState: 2
    },
    {
      depthX: 0.5052429437637329,
      depthY: 0.3433794379234314,
      colorX: 0.5190386176109314,
      colorY: 0.3478032946586609,
      cameraX: -0.01570604369044304,
      cameraY: 0.25717124342918396,
      cameraZ: 1.5779820680618286,
      orientationX: -0.001616719993762672,
      orientationY: 0.9985165596008301,
      orientationZ: 0.036773987114429474,
      orientationW: -0.0401209257543087,
      jointType: 1,
      trackingState: 2
    },
    {
      depthX: 0.50527024269104,
      depthY: 0.18280431628227234,
      colorX: 0.520110011100769,
      colorY: 0.1661166399717331,
```

5.2 Kinect Studio

For KDash I used application called Kinect studio. Kinect studio can capture the data coming into that application from the Kinect unit. Kinect Studio captures the data feeds from the colour stream and the depth stream. I used Kinect studio in KDash to record two types of gestures lean and jump, save them as .xed files and reuse them in the gesture database.

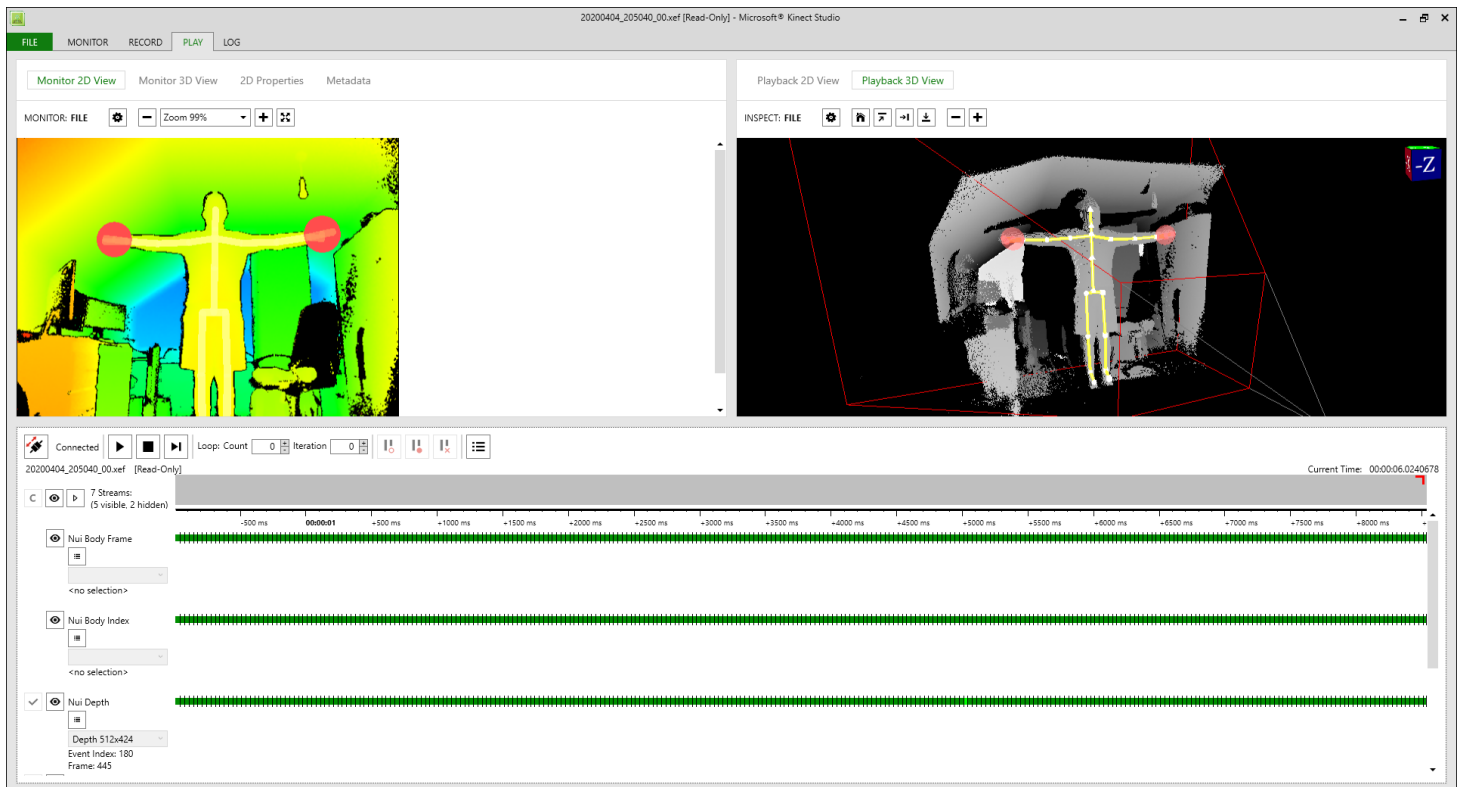


Figure 5-2 Kinect Studio Record Application

5.3 Visual Gesture Builder

Visual Gesture Builder (VGB) generates data that are used by applications to perform run-time gesture detection. By using a data-driven model, VGB shifts the emphasis from writing code to building gesture detection that is testable, repeatable, configurable, and database-driven. This method provides better gesture recognition. VGB uses several detection technologies. The user selects the detection technologies to use—namely AdaBoostTrigger or RFRProgress—and tags frames in a clip related to a meaningful gesture, such as a lean or jump. At the end of the tagging process, VGB builds a gesture database; with this database, an application can process body input from a user to, for example, detect a lean or jump.

“AdaBoostTrigger - The AdaBoostTrigger is a binary technology type. It uses an Adaptive Boosting (AdaBoost) machine learning algorithm to determine when a user performs a certain gesture.”

“RFRProgress - The RFRProgress is a detection technology that produces an analog or continuous result. It uses the Random Forest Regression (RFR) machine learning algorithm to determine the progress of a gesture performed by a user.”

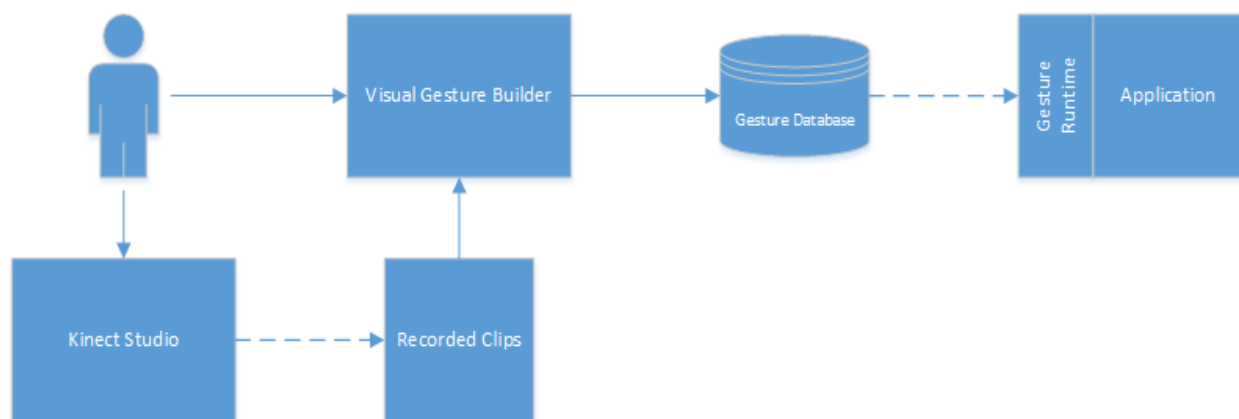


Figure 5-3-1 Visual Gesture Builder in Context

In KDash I am only using adaBoostTrigger [13] for leaning and jumping because it is easier to manipulate as well as During training time it accepts input tags, Boolean values, which mark the occurrence of a gesture, such as a lean. This marking or tagging is used to evaluate whether a

gesture has happened or not and determines the confidence value of the event. For KDash I used myself as training data and three of my classmates for test data making sure we all looked different for better test data and so I could get the perfect confidence. I decided to go with a confidence level of 65% because after looking at the test data that my classmates helped me with, I got the confidence of when the lean would start from about 0.40-0.60 percent. So, in my code I set it to look for 0.65% so that it could be modified for different body types and it was perfect for me in the sense that I would be in leaning/jumping motion when my character would either move left or right. I repeated this for my jump gesture.

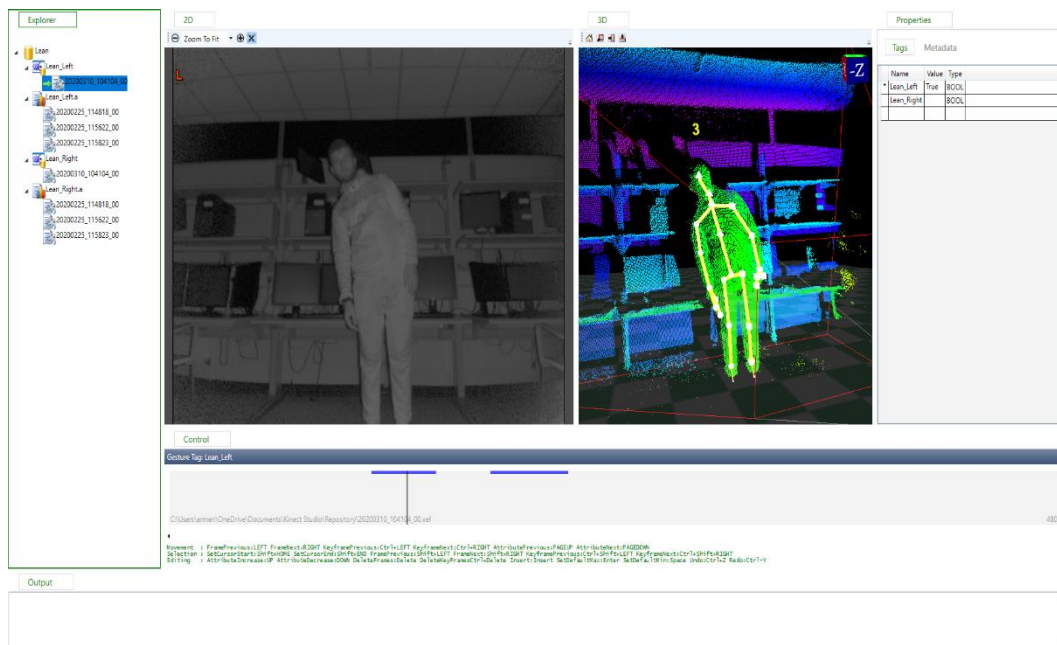


Figure 5-3-2 Training Data of myself leaning left.

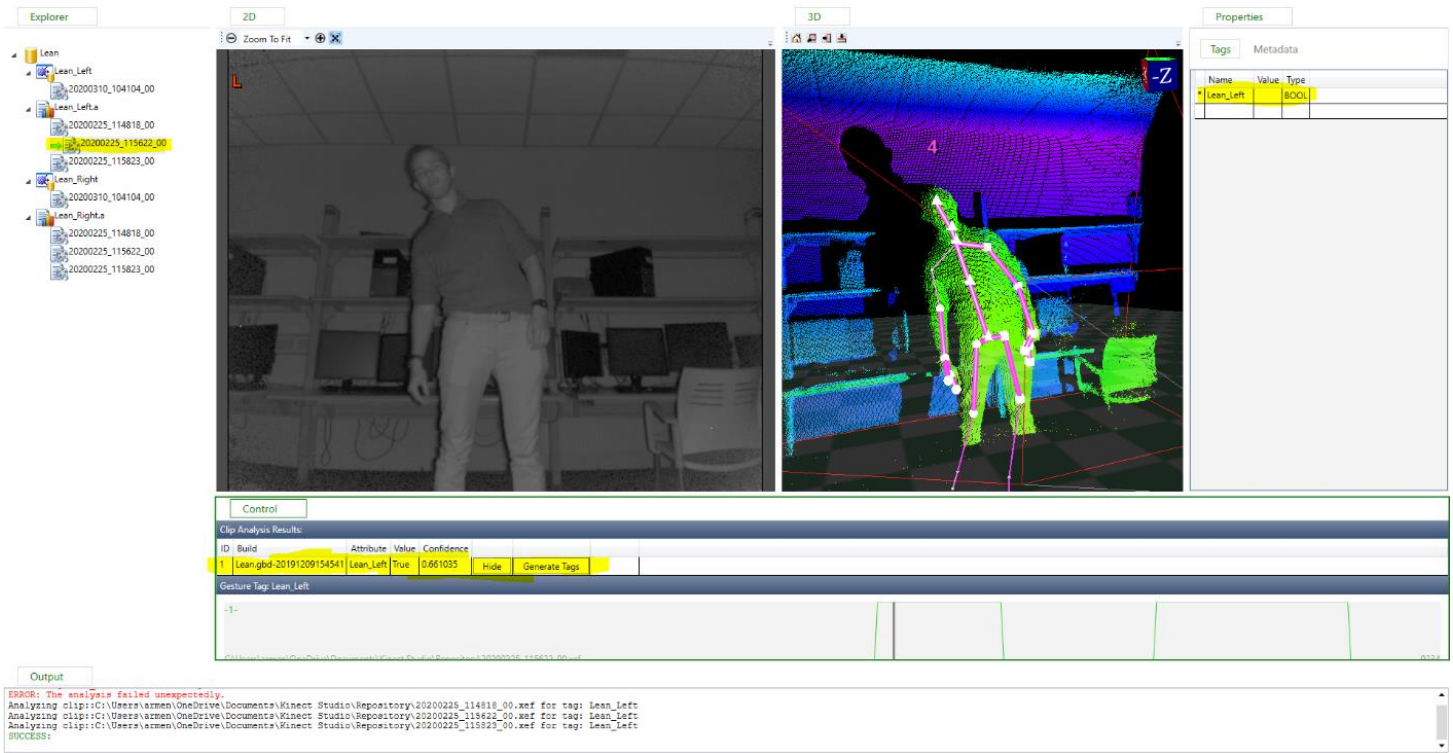


Figure 5-3-3 Test Data one showing confidence at 0.66%

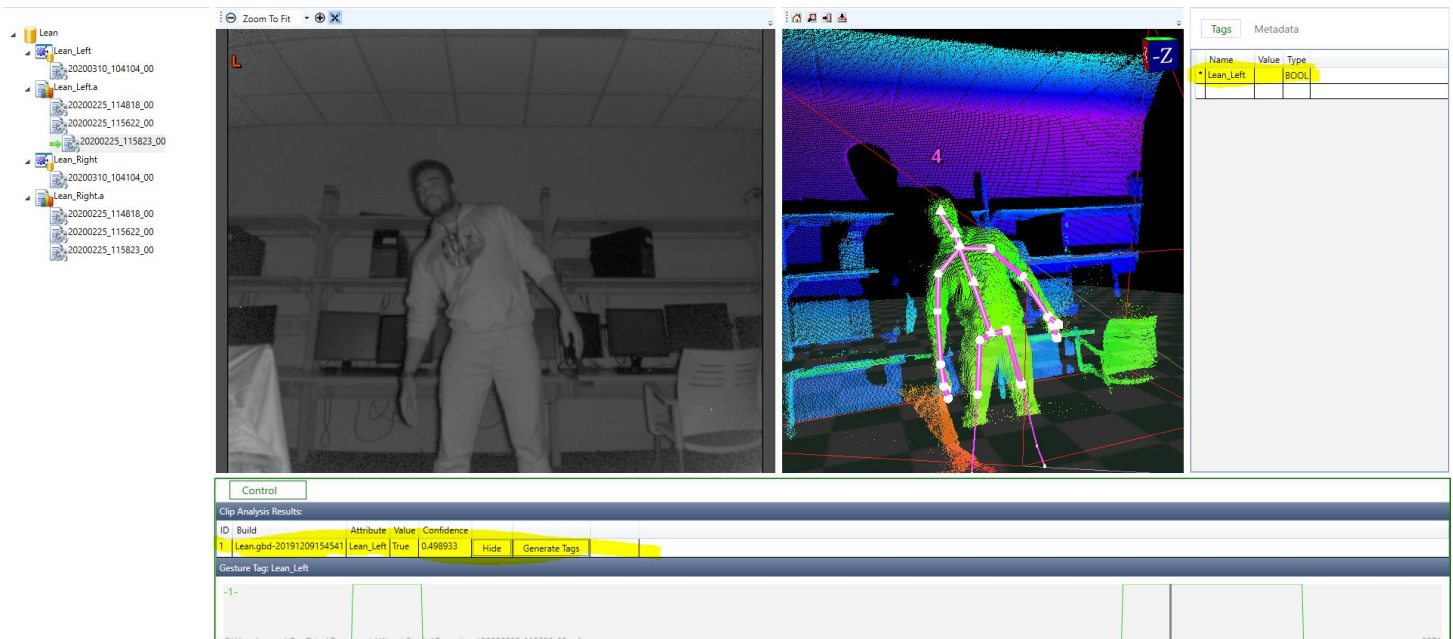


Figure 5-3-4 Test Data two showing confidence at 0.49%

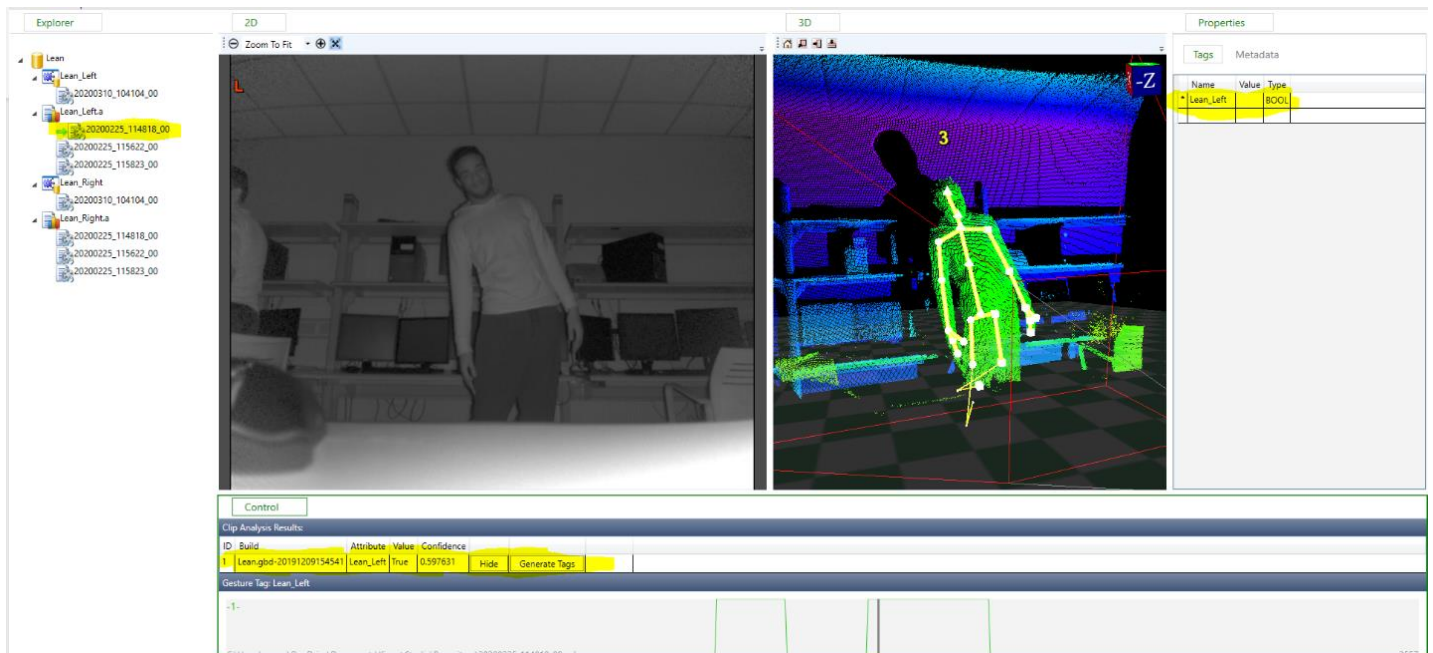


Figure 5-3-5 Test Data three showing confidence at 0.59%

```

1 reference
private void OnGestureDetected(object sender, GestureEventArgs e, int bodyIndex)
{
    if (e.GestureID == leanLeftGestureName)
    {
        //NEW UI FOR GESTURE DETECTED
        if (e.DetectionConfidence > 0.65f) //65%
        {
            turnScript.moveLeft = true;
            turnScript.moveRight = false;
        }
    }

    //Debug.Log(e.GestureID);
    if (e.GestureID == leanRightGestureName)
    {
        //NEW UI FOR GESTURE DETECTED
        if (e.DetectionConfidence > 0.65f)
        {
            turnScript.moveRight = true;
            turnScript.moveLeft = false;
        }
    }

    if (e.GestureID == jumpGestureName)
    {
        if (e.DetectionConfidence > 0.20f)
        {
            turnScript.jump = true;
        }
        else
        {
            turnScript.jump = false;
        }
    }
}

```

Figure 5-3-6 OnGestureDetected Method

6 Unity

“Unity is a cross-platform game engine developed by Unity Technologies. It is used to develop video games for web plugins, desktop platforms, consoles, and mobile devices. Unity is programmed using C# using visual studio.”

Unity [11] is the platform I choose to make my game because it was more compatible with the Kinect V2 and its easier to use. I used unity3D to make my endless 3D running map. To make my endless 3D running map I started off by using cube game objects to build the platform as well as a wall either side so my character would not fall out of the map. I later used the asset store to get a character. After finding a character I liked, I soon found out later that I had to add animation to the character. Unity offers an animator controller that allows you to arrange and maintain a set of Animation Clips and associated Animation Transitions for a character or object. So, based off my game I only needed 4 states my character could be in entry, idle, run, jump, exit.

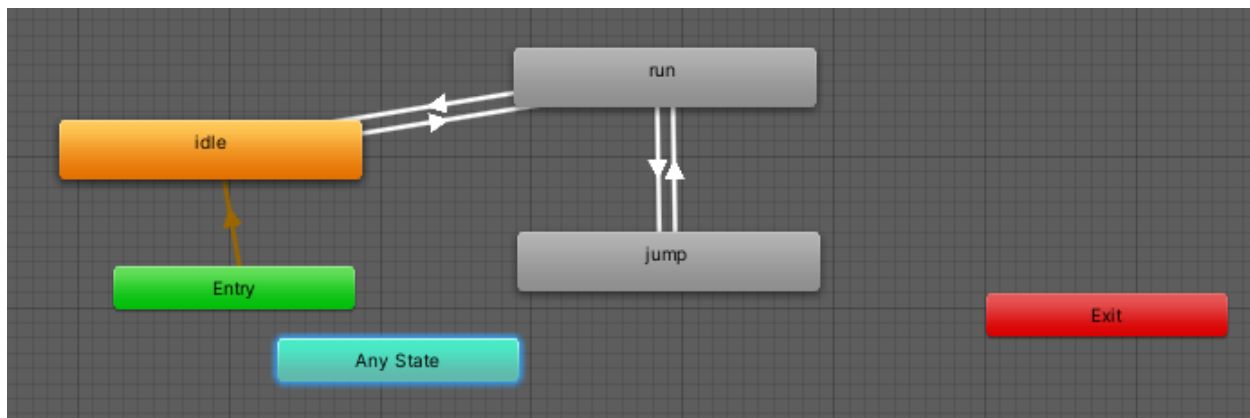


Figure 6-1 Animator Controller

Once my character was in, I started making him run on the map that I had built so he would change state from running, jumping and idle.

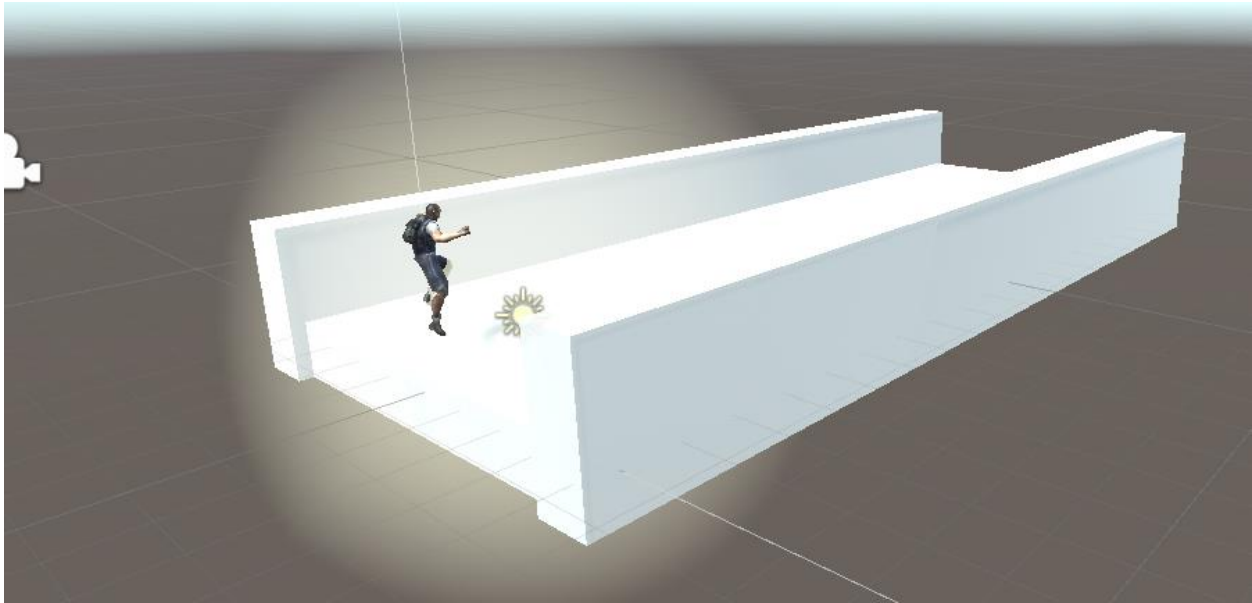


Figure 6-1-3 Starting Map

Once I got him changing states on the map, I decide to think of ways to make the map run endlessly. My first idea was to write a script that would just copy and paste the map repeatedly. while it a good Idea I ran into glitches appearing on the map where my player would just fall out of the world and also a minor issue I ran into was a memory and CPU issue because I kept generating game object which took up memory space and caused the CPU to spike high which caused my game to crash. So, I came up with an idea to place a moving invisible game object in front of character and one at the end of the map, then I created two scripts. One is just a simple C# script to destroy game objects after 10 seconds and the other script I used object collision to detect when the character collides with the invisible game object it would clone the game map object original and return the clone of the game map object and add it to the second game object point at the end of the original map.

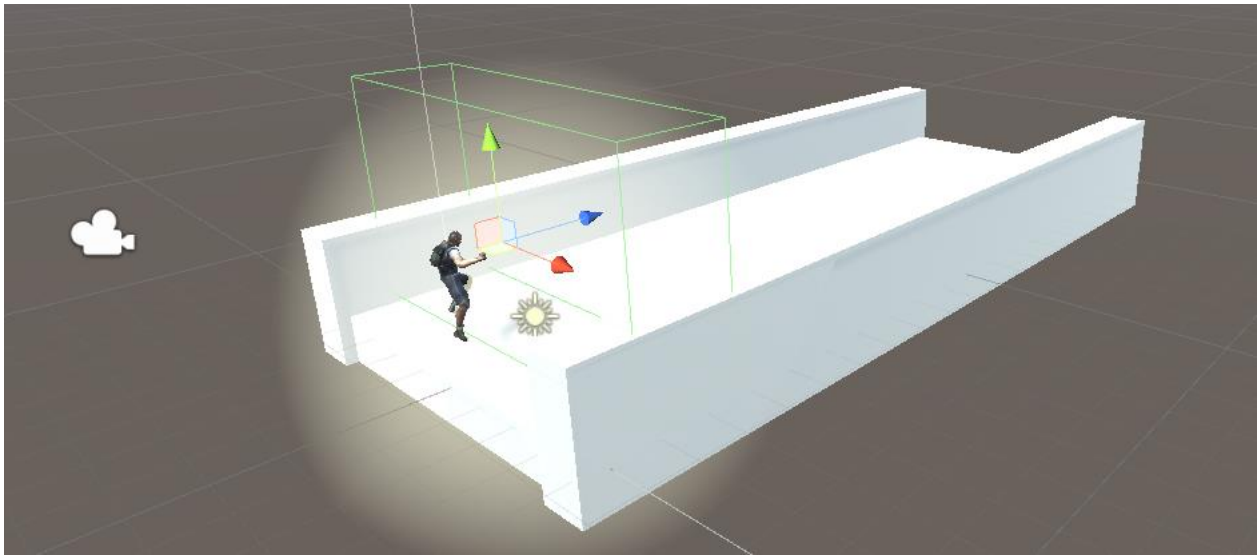


Figure 6-1-2 Path Collider Point

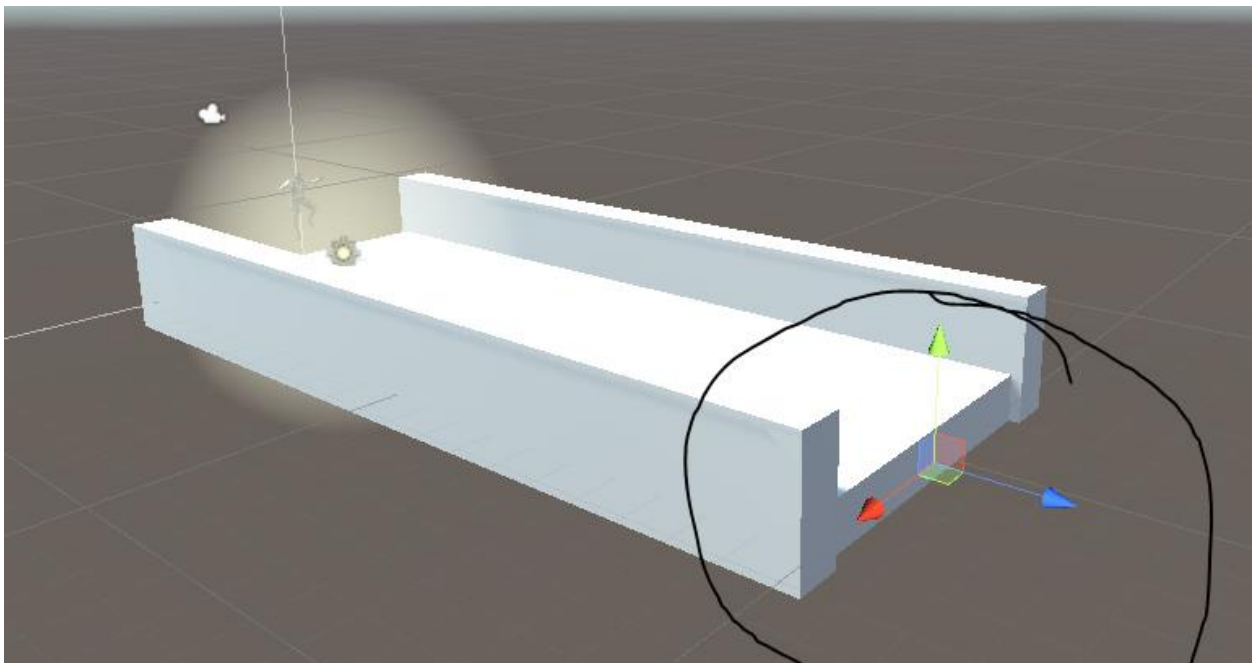


Figure 6-1-3 Path Spawn Point

```

1  using UnityEngine;
2
3  public class PathSpawnCollider : MonoBehaviour
4  {
5      public Transform[] PathSpawnPoints;
6
7      public GameObject Path;
8
9      internal void OnTriggerEnter(Collider hit)
10     {
11         //player has hit the collider
12         if (hit.gameObject.tag == Constants.PlayerTag)
13         {
14             //find whether the next path will be straight, left or right
15             int SpawnPoint = Random.Range(0, PathSpawnPoints.Length);
16             for (int i = 0; i < PathSpawnPoints.Length; i++)
17             {
18                 //instantiate the path
19                 if (i == SpawnPoint)
20                     Instantiate(Path, PathSpawnPoints[i].position, PathSpawnPoints[i].rotation);
21                 else
22                 {
23                     Vector3 position = PathSpawnPoints[i].position;
24                 }
25             }
26         }
27     }
28 }
29
30

```

Figure 6-1-4 Path Spawn Collider Code

Once the map was built, I decided to add candy for point and obstacles so the player could dodge them by leaning or jumping. For this I created two scripts. The first script made sure that when you picked you candy; you would get certain points for different candies and I added a little animation to make them spin. The second script was to randomly generate and spawn the obstacles, candy, and power ups. First, I made sure that obstacles only had a 50 percent chance to spawn so there would not be too many on the field. Then I gave candy and powerups 33 percent each so, many would not spawn and crowd the field and I saw by using 33 percent it randomly picks between them.

```

1 using UnityEngine;
2 using System.Collections;
3
4 [References]
5 public class StuffSpawner : MonoBehaviour
6 {
7     //points where stuff will spawn
8     public Transform[] StuffSpawnPoints;
9     //next game objects
10    public GameObject[] Bonus;
11    public GameObject[] PowerUps;
12    public GameObject[] Obstacles;
13
14    public bool RandomX = false;
15    public float minx = -2f, maxx = 2f;
16
17    // Use this for initialization
18    void Start()
19    {
20        //First, let's decide whether we'll place an obstacle
21        //Random.Range is exclusive for integers but inclusive for
22        bool placeObstacle = Random.Range(0, 2) == 0; //50% chances
23        int obstacleIndex = -1;
24        if (placeObstacle)
25        {
26            //Select a random spawn point, apart from the first one
27            //since we do not want an obstacle there
28            obstacleIndex = Random.Range(1, StuffSpawnPoints.Length);
29
30            CreateObject(StuffSpawnPoints[obstacleIndex].position, Obstacles[Random.Range(0, Obstacles.Length)]);
31        }
32
33        for (int i = 0; i < StuffSpawnPoints.Length; i++)
34        {
35            //Don't instantiate if there's an obstacle
36            if (i == obstacleIndex) continue;
37            if (Random.Range(0, 3) == 0) //33% chances to create candy
38            {
39                CreateObject(StuffSpawnPoints[i].position, Bonus[Random.Range(0, Bonus.Length)]);
40            }
41
42            for (int i = 0; i < StuffSpawnPoints.Length; i++)
43            {
44                //Don't instantiate if there's an obstacle
45                if (i == obstacleIndex) continue;
46                if (Random.Range(0, 3) == 0) //33% chances to create candy
47                {
48                    CreateObject(StuffSpawnPoints[i].position, PowerUps[Random.Range(0, PowerUps.Length)]);
49                }
50            }
51        }
52
53        //References
54        void CreateObject(Vector3 position, GameObject prefab)
55        {
56            if (RandomX) //True on the wide path
57                position = new Vector3(Random.Range(minx, maxx), 0, 0);
58            //Clone the object original and returns the clone
59            Instantiate(prefab, position, Quaternion.identity);
60        }
61    }

```

Figure 6-1-5 Generates candy, obstacles and power ups

```

1 using UnityEngine;
2 using System.Collections.Generic;
3
4 [References]
5 public class Candy : MonoBehaviour
6 {
7     public int ScorePoints = 100;
8     public float rotateSpeed = 50f;
9
10    // Update is called once per frame
11    internal void Update()
12    {
13        transform.Rotate(Vector3.up, Time.deltaTime * rotateSpeed);
14    }
15
16    //References
17    internal void OnTriggerEnter(Collider col)
18    {
19        UIManager.Instance.IncreaseScore(ScorePoints);
20        Destroy(this.gameObject);
21    }
22 }

```

Figure 6-1-6 Adding Point to Candy

```
1  using UnityEngine;
2  using System.Collections.Generic;
3  public class PowerUps : MonoBehaviour
4  {
5      public int ScoreMultipliers;
6      public float rotateSpeed = 50f;
7
8      // Update is called once per frame
9      internal void Update()
10     {
11         transform.Rotate(Vector3.up, Time.deltaTime * rotateSpeed);
12     }
13
14     internal void OnTriggerEnter(Collider col)
15     {
16         UIManager.Instance.IncreaseScore(ScoreMultipliers);
17         Destroy(this.gameObject);
18     }
19 }
20
```

Figure 6-1-7 Adding Powerups to Candy

7 Web Server

For my web server I decided to setup a demo aimed at skeleton tracking using the Kinect V2. I did most of this by using examples and tutorials I found online and combining them to work. This was only used to show off how the Kinect tracked each joint in the body.

A web server is a computer that runs websites. The reason for a web server is to store, process and deliver web pages to users. This intercommunication is done by using Hypertext Transfer protocol (HTTP). These web pages done by using HTML (Hypertext Markup Language), JavaScript and CSS (Cascading Style Sheets).

The reason why I decided to use a webpage to showcase skeleton tracking was because it was easier to manipulate and show off each joint in the body on a webpage as seen in Figure 7-0-1



Figure 7-0-1 Output of Skeleton Tracking on webpage

7.1 Server

As seen in Figure 7-1-1, I decided to run this JavaScript locally on port 8080 because it is easier and local host are faster to work with because you do not need an internet connection. You can also set up as many sites as your computer has space for.

I then check if a Kinect is connected to my computer and if it is, the server starts up locally on localhost 8080 and starts taking in data from the Kinect if a body is in within frame. Then it sends it using sockets because it sockets enables Realtime, bi-directional communication between web clients and servers.

```
1  var Kinect2 = require('kinect2'),  
2      kinect = new Kinect2(),  
3      express = require('express'),  
4      app = express(),  
5      server = require('http').createServer(app),  
6      io = require('socket.io').listen(server);  
7  
8  if (kinect.open()) {  
9      server.listen( port: 8080);  
10  
11     console.log('Server listening on port 8080');  
12  
13     app.get('/', function (req, res) {  
14         res.sendFile( path: 'C:\\nodejs\\MiniProject\\Files\\Kinect\\Kinect.html');  
15     });  
16  
17     kinect.on('bodyFrame', function (bodyFrame) {  
18         io.sockets.emit('bodyFrame', bodyFrame);  
19     });  
20  
21     //request body frames  
22     kinect.openBodyReader();  
23 }
```

Figure 7-1-1 Server-Side Code

7.2 Client

```

6      <title>Node Kinect2 Client</title>
7      <link rel="stylesheet" href="">
8  </head>
9  <body>
10     <canvas id="bodyCanvas" width="512" height="424"></canvas>
11     <script src="/socket.io/socket.io.js"></script>
12     <script>
13         var socket = io.connect('/');
14         var canvas = document.getElementById('bodyCanvas');
15         var ctx = canvas.getContext('2d');
16         var colors = ['000000', '000000', '0000ff', '0000ff', '0000ff', '0000ff'];
17         // handstate circle size
18         var HANDSIZE = 20;
19         // closed hand state color
20         var HANDCLOSEDCOLOR = "red";
21         // open hand state color
22         var HANDOPENCOLOR = "green";
23         // lasso hand state color
24         var HANDLASSOCOLOR = "blue";
25
26         function updateHandState(handState, jointPoint) {
27             switch (handState) {
28                 case 3:
29                     drawHand(jointPoint, HANDCLOSEDCOLOR);
30                     break;
31                 case 2:
32                     drawHand(jointPoint, HANDOPENCOLOR);
33                     break;
34                 case 4:
35                     drawHand(jointPoint, HANDLASSOCOLOR);
36                     break;
37             }
38         }
39         function drawHand(jointPoint, handColor) {
40             // draw semi transparent hand circles
41             ctx.globalAlpha = 0.75;
42             ctx.beginPath();
43             ctx.fillStyle = handColor;
44             ctx.arc(jointPoint.depthX * 512, jointPoint.depthY * 424, HANDSIZE, 0, Math.PI * 2, true);
45             ctx.fill();
46             ctx.closePath();
47             ctx.globalAlpha = 1;
48         }
49         socket.on('bodyFrame', function (bodyFrame) {
50             ctx.clearRect(0, 0, canvas.width, canvas.height);
51             var index = 0;
52             bodyFrame.bodies.forEach(function (body) {
53                 if (body.tracked) {
54                     for (var jointType in body.joints) {
55                         var joint = body.joints[jointType];
56                         ctx.fillStyle = colors[index];
57                         ctx.fillRect(joint.depthX * 512, joint.depthY * 424, 10, 10);
58                     }
59                     //draw hand states
60                     updateHandState(body.leftHandState, body.joints[7]);
61                     updateHandState(body.rightHandState, body.joints[11]);
62                     index++;
63                 }
64             });
65         });
66     </script>
67 </body>
68 </html>

```

Figure 7-1-1 Client-Side Code

8 Problem Solving

During the creation of KDash I ran in a couple minor issues.

The first problem I ran into was a hardware issue, I was unable to use the Kinect v1 because the drivers were too old to run on my computer and it would have been too much hassle to run it on a virtual machine. I also found that the Kinect V1 compatibility with windows 10 was very slow and buggy. So my solution to this problem was to buy and upgrade to the KinectV2 which had better colour video camera, depth camera, it could track up to 3x what the Kinect v1 could do and it was also compatible with windows 10 and easier to used overall over the V1.

The second problem I ran into was later in the project. I ran into a memory and CPU problem in unity. The program kept crashing a minute in after starting it and I couldn't figure it out for a while, until I figured out that when I started running the program, I was instantiating candy, obstacles and even the map, this basically meant that I was cloning the object original and returning the clone which was taking up memory and causing the CPU to spike high and crash unity. The way I decided to fix this problem was creating a script called "Time Destroyer" that would invoke a method called "Destroy Object". The method "Destroy Object" would check if the player had died, if it had not then it would call a method destroy, which basically meant it removes the game Object, component, or asset. In the invoke I set it up to invoke every 10 seconds which meant the character would have already passed it after it got destroyed.

```

1  using UnityEngine;
2  using System.Collections;
3
4  0 references
5  public class TimeDestroyer : MonoBehaviour
6  {
7      // Use this for initialization
8      0 references
9      void Start()
10     {
11         Invoke("DestroyObject", LifeTime);
12     }
13
14     0 references
15     void DestroyObject()
16     {
17         if (GameManager.Instance.GameState != GameState.Dead)
18             Destroy(gameObject);
19     }
20
21     public float LifeTime = 10f; //LifeTime determines how many seconds this game object will be alive for..
22 }

```

Figure 8-1-1 Time Destroyer Script

Another way I decided to fix the problem was to create a script and singleton implementation, this is a method I found that helped a lot to fix memory and CPU problems with unity. singleton pattern is one of the best-known patterns in software engineering. Essentially, a singleton is a class which only allows a single instance of itself to be created, and usually gives simple access to that instance.

```

internal void Awake()
{
    if (instance == null)
    {
        instance = this;
    }
    else
    {
        DestroyImmediate(this);
    }
}

//singleton implementation
private static UIManager instance;

9 references
public static UIManager Instance
{
    get
    {
        if (instance == null)
            instance = new UIManager();

        return instance;
    }
}

1 reference
protected UIManager()
{
}

```

Figure 8-1-1 UIManager Script

9 System Integration

For KDash I had two major parts for my project, I had the Kinect V2 and unity. I first started off with trial and error with trying to get the Kinect V1 working with my laptop, which in the end I could not get working so I had to order the Kinect V2.

While I waited for the Kinect V2 to come in I started off by downloading unity and learning how that worked. Once I got the hang off using unity from completing tutorials and looking at examples, I started to build my endless 3D running map. To build the map I started off by building a runway using 20 cube game objects extends to 8 on the x coordinate so the player would have space to move left and right. I then added in two walls so the player would not fall out of the map and I could implement colliders to block the player from glitching into the wall. So to extend the map endlessly I came up with an idea to place a moving invisible game object in front of character and one at the end of the map, then I created a script that used object collision to detect when the character collides with the invisible game object it clones the game map object original and returns the clone of the game map object and adds it to the second game object point at the end of the original map.

At this point I ran into my first bug. I ran into a memory and CPU problem in unity. The program kept crashing a minute in after starting it and I couldn't figure it out for a while, until I figured out that when I started running the program, I was instantiating candy, obstacles and even the map, this basically meant that I was cloning the object original and returning the clone which was taking up memory and causing the CPU to spike high and crash unity. The way I decided to fix this problem was creating a script called "Time Destroyer" that would invoke a method called "Destroy Object". The method "Destroy Object" would check if the player had died, if it had not then it would call a method destroy, which basically meant it removes the game Object, component, or asset. In the invoke I set it up to invoke every 10 seconds which meant the character would have already passed it after it got destroyed.

Another way I decided to fix the problem was to create a script and singleton implementation, this a method I found helped a lot to fix memory and CPU problems with unity. singleton pattern is one of the best-known patterns in software engineering. Essentially, a singleton is a

class which only allows a single instance of itself to be created, and usually gives simple access to that instance. Once I had a basic temple of my runway built, I went online and found some character skins that I thought was cool and added it as an asset in my game. After finding a character I liked, I soon found out later that I had to add animation to the character. So, based off my game I only needed 4 states my character could be in entry, idle, run, jump, exit.

I first got my character to move using arrow keys just to test the map, candy, and obstacles. At this point my Kinect V2 had already arrived and I started testing and recording myself to make sure it worked with my laptop which it did. While waiting for the Kinect V2, I did a little research on how gestures work and how to implement them into my project. what I found was there was an application built within Kinect Studios called Visual Gesture Builder, which has a wizard that allowed me to specify what kind of gesture I was trying to create (i.e. kick, punch, lean or jump) and whether it was a discrete or continuous gesture. I then imported a set of clips me and my 3 classmates pre-recorded within Kinect studios for test data and training data. At that point, the gesture builder analysed the test data to the original training data using Adaptive Boosting (AdaBoost) machine learning algorithm to find the perfect confidence so I could later use that in my code.

As soon as I had my leaning and jump gesture recorded and analysed using Adaptive Boosting (AdaBoost) machine learning algorithm. Then I went back to Unity and imported two scripts that Microsoft offer when you download their Kinect SDK, known as Kinect Manager and gesture detector. These are default scripts that read in Kinect data and detect gestures. A modification that I done to the KinectManager script so that the Kinect would look for specific gestures (i.e. lean left, lean right and jump) using its depth sensor and skeleton tracking to track when the player has leaned/jumped. I then modified gestureDetector and added gestureID in the gesture builder script so that later if I wanted to add more gestures to KDash I could easily by just storing them into an array and giving each gesture an ID as it stood. I did not need to put them in an array because I only had two gesture and a simple if statement could do the job. I assigned them each a gestureID. I also added the specific path to the two videos I got from the gesture builder one for jump and one for lean as well as specified what type of gesture it was.

Once it could detect gestures, I had to modify the script I wrote for moving the character sideways. I first added three Boolean values Lean Left, Lean Right and jump. Then Instead of checking for an arrow key to be pressed I just checked if lean left was true then move the character as well as for lean right and jump. I then went back to KinectManager and rewrote a function called “OnGestureDetected” so that if the gestureID matched pre-recorded videos of myself leaning and if it detected that the player has gone pass a specific confidence level then it would set each Boolean value to true depending on the confidence level. I used 65% for leaning left or right and 20% for jumping. Because after the gesture builder was done analysing using Adaptive Boosting (AdaBoost) machine learning algorithm the test and training data it gave me the exact confidence level at which point the player was in a leaning motion or jumping motion. So, I used those value and once I ran my game my character started to move left, right, or even jumped when I leaned left, right, or even jumped.



10 Conclusion

So, in the end I got everything working and integrated together which I was happy about. when I evaluated all the technologies I implement in my system, I yielded the result of a working application. Premised on this fact, I was proud of what I could achieved such as detecting different gestures for instance lean left, lean right, and even jump was functioning as expected.

The character will run infinitely on the map I created and all you must do is lean or jump and the game will do the rest for you. By doing this project it has equipped me with new technologies that I never utilized before and as I embark on my professional journey in the technology realm I will definitely integrate the lessons and new ideologies and methodologies that I have picked up from my project.

11 References

- [1] Pietro Polsinelli. (2020). Why is Unity so popular for videogame development? Available: <https://designagame.eu/2013/12/unity-popular-videogame-development/>. Last accessed 07/04/2020.
- [2] Unity Technologies. (2020). Animator Controller. Available: <https://docs.unity3d.com/Manual/class-AnimatorController.html>. Last accessed 08/04/2020
- [3] Ashlee Watts. (2019). How Microsoft has leveraged. Available: <https://slideplayer.com/slide/12905388/>. Last accessed 08/04/2020.
- [4] Stephanie Crawford. (2010). How Microsoft Kinect Works. Available: <https://electronics.howstuffworks.com/microsoft-kinect.htm>. Last accessed 08/04/2020.
- [5] Tim Carmody. (2010). How Motion Detection Works in Xbox Kinect. Available: <https://www.wired.com/2010/11/tonights-release-xbox-kinect-how-does-it-work/>. Last accessed 08/04/2020.
- [6] Pietro Polsinelli. (2013). Why is Unity so popular for videogame development? Available: <https://designagame.eu/2013/12/unity-popular-videogame-development/>. Last accessed 08/04/2020.
- [7] Leland Holmquest. (2012). Kinect - Working with Kinect Studio. Available: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2012/september/kinect-working-with-kinect-studio>. Last accessed 08/04/2020.
- [8] Ben Lower. (2014). Custom Gestures End to End with Kinect and Visual Gesture Builder. Available: <https://channel9.msdn.com/Blogs/k4wdev/Custom-Gestures-End-to-End-with-Kinect-and-Visual-Gesture-Builder>. Last accessed 08/04/2020.
- [9] ben Lower. (2015). Kinect 2 Hands on Labs. Available: <https://kinect.github.io/tutorial/lab12/index.html>. Last accessed 08/04/2020.
- [10] Jason Brownlee. (2016). Boosting and AdaBoost for Machine Learning. Available: <https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/>. Last accessed 08/04/2020.
- [11] David Helgason. (2004). Unity. Available: <https://unity.com/>. Last accessed 08/04/2020.
- [12] Syndicated News. (2017). Visual Gesture Builder (VGB). Available: thewindowsupdate.com/2017/05/04/visual-gesture-builder-vgb/. Last accessed 08/04/2020.

[13] Microsoft. (2014). AdaBoostTrigger. Available: [https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785522\(v=ieeb.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785522(v=ieeb.10)?redirectedfrom=MSDN). Last accessed 08/04/2020.

[14] Microsoft. (2014). RFRProgress. Available: [https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785524\(v=ieeb.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785524(v=ieeb.10)?redirectedfrom=MSDN). Last accessed 08/04/2020.