Gesture Based UI Project

G00356158

Benas Pelakauskas

Introduction

Before beginning the project, I created a brainstorm of potential ideas I could use for the creation of this project fitting in line with the brief.

Examples include:

- Menu completely operated using XBOX 360 Kinect
- Grand Theft Auto-like game with the ability to roam freely using Kinect and interact with other AI's using your microphone.
- Kinect Fighting game

I finally settled on the idea of creating a Kinect penalty shooter game, which would utilise voice commands to navigate the menu or reset the level.

I began by downloading and installing Microsoft's Kinect SDK v1.8.

Additional resources used include Kinect with MS-SDK and Distant Lands Free Characters, both from the Unity Asset Store.

Implementation Steps

Before progressing further, I tested the tracking on the Kinect sensor to establish any possible difficulties I may encounter later in development.

I created a base (ground) which I dropped the Free Character prefab on and after configuration, all was working.

I then proceeded to create a goal by adding in cubes, stretching and fitting them together, followed by a wall to prevent the ball from falling out of the map.

I added a sphere (ball) and attempted to kick it, which is where I encountered problems.

The prefabricated character would not affect the ball, either with RigidBody or a collider component added.

My first attempt was to create a C# script, which calculates the velocity of the kick, then apply this velocity to the ball using another script.

```
public Transform Object; // gameobject
private Vector3 _position; // last position of target

private void OnEnable()
{
    _position = Object.transform.position;
}

private void Update()
{
    var dt = Time.deltaTime;
    var current = Object.transform.position;
    var delta = Vector3.Distance(current, _position);
    var velocity = delta / dt;
    Debug.Log((velocity).ToString("#,##0.000"));

    // replace current position
    _position = current;
}
```

Unfortunately, this method proved unsuccessful, however the script still has the potential to be used for other applications such as determining the ‘Fastest Kick’ if competing with friends.

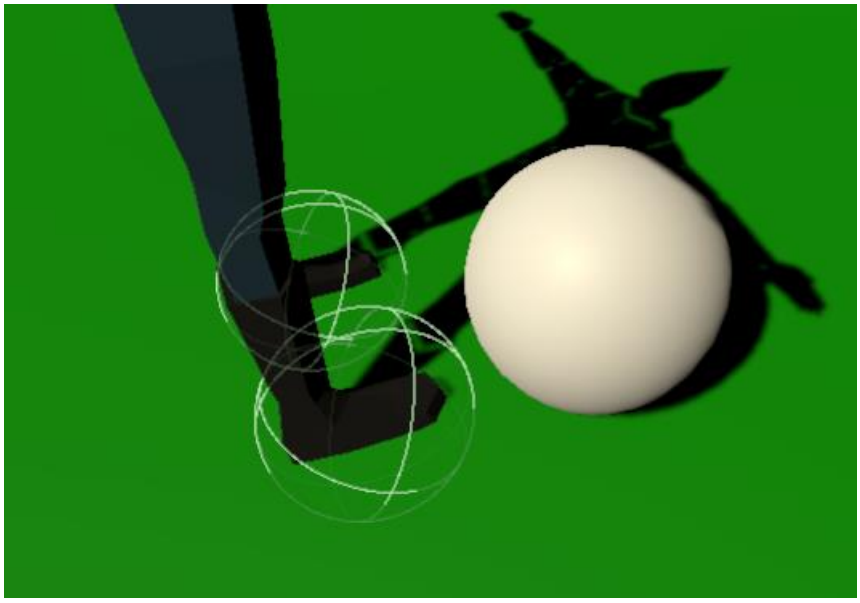
The ‘BallVelocity’ script was created and then edited to apply a velocity boost to the ball due to the players’ weak force when kicking using Kinect.

However, the script is currently unused and could be potentially used as a ‘Power-Up’.

My solution to fix the characters’ inability to kick the ball was fixed by creating a small sphere and implanting it into the players’ foot, both left and right.

Initially, the momentum created from the kick was barely strong enough to move the ball into the goal.

After trial and error of testing different mass values of the sphere and football with no success, I enlarged the two spheres which were inside the characters’ shoe. This negatively impacted the visual player, so I made the spheres invisible, but physically present.



Imaged above; The size of the spheres used.

Once I had the base game working correctly, I proceeded to create a new scene for the menu.

For the menu, I decided to use voice commands instead of having to manually reset or change scenes using a mouse and keyboard. It also allows for a greater variety of gestures.

This way, any actions required by the game could be performed while standing in-front of the Kinect sensor without having to touch the computer.

The menu scene is relatively simple, with a background photo of the ‘MainScene’ used to improve the user experience visually, accompanied by coloured text to help users differentiate the writing.

The game consists of three auditory commands:

- “Start” – to start the game from the menu.
- “Menu” – to return back to the menu.
- “Reset” – to reset the scene and return the ball to the original position.

```
private void Start()
{
    actions.Add("start", StartGame);
    actions.Add("menu", Menu);
    actions.Add("reset", Reset);

    keywordRecognizer = new KeywordRecognizer(actions.Keys.ToArray());
    keywordRecognizer.OnPhraseRecognized += RecognizedSpeech;
    keywordRecognizer.Start();

}
```

Pictured above; how commands are mapped to their corresponding actions, using ‘keywordRecognizer’.

```
// Loads scene 1
private void StartGame()
{
    SceneManager.LoadScene(1);
}
```

Pictured above; the action each command carries out, in this case, saying ‘Start’ will trigger the ‘StartGame()’ method which consists of loading scene 1.

The gestures implemented are simple to use and logical.

E.g to kick a ball, you perform a ‘kick’ motion with the desired foot, which correlates to the same action in game.

Vocal commands are straightforward and easy to memorize, as the wording used is directly correlated to the action E.g ‘Menu’, opens the menu.

Conclusion

In conclusion, I enjoyed undertaking this project and believe that there is nearly an endless amount of possibilities for similar projects using this technology.

I have learned the necessary skills to undertake such project and utilize the technology to create a game and user interface controllable by gestures.

With a much larger scope, I would have liked to use the 'Fastest Kick' script and create a multi-player mode with the ability to roam around the football field and play a full game. Communication could be performed over the microphone between players, allowing for a very immersive football experience, possibly even with the implementation of a virtual reality headset.

Undertaking such project would be better using the Xbox One Kinect instead of the Xbox 360 due to the improved technology, such as the ability to recognise facial expressions and monitor heart rate for intense Kinect sessions.

GitHub Repository:

https://github.com/BenasPelakauskas/Gesture-Based-UI-Project