HCI: Gesture Enabled Games

UROP 1000

Students

Prasenjit Das, pdas@connect.ust.hk

Xu Muyu, mxuai@connect.ust.hk

Li Xiang, xlidh@connect.ust.hk

Fan Ziqian, zfanag@connect.ust.hk

Hong Kong University of Science and Technology

Supervised by

Prof. Xiaojuan MA, mxj@cse.ust.hk

Hong Kong University of Science and Technology

Abstract

With every passing day, the world is coming closer to the best of motion control and gesture recognition technologies. Specifically, the world of gaming has been changing exponentially. From tilting mobile phones back and forth to punching aimlessly in the air to control characters, gaming technology has come a long way.

The purpose of this research was to understand one such technology and develop a game that would use this technology. The communication between *Welle* and PC was tested using *CoolTerm*. A more refined communication was established using C#. An adventure game was then developed using *Unity2D* as the final game for the project.

The results of this research can be used to further improve the accuracy of the device and consider the feasibility of use it with games.

Introduction

Human Computer Interaction is the study of how people interact with computers and to what extent computers are or are not developed for successful interaction with human beings.[1] Over the past few decades there has been a rapid growth and diversification in this field. From Graphical User Interface (GUI) to Voice User Interface(VUI), the different branches of HCI has had different impacts in the world of technology.

Another such diversification of HCI came to be by using Sound Navigation And Ranging (SONAR) technique. This technique uses sound to navigate and communicate with other devices. The device that we had been working with in our UROP 1000 project is called *Welle*

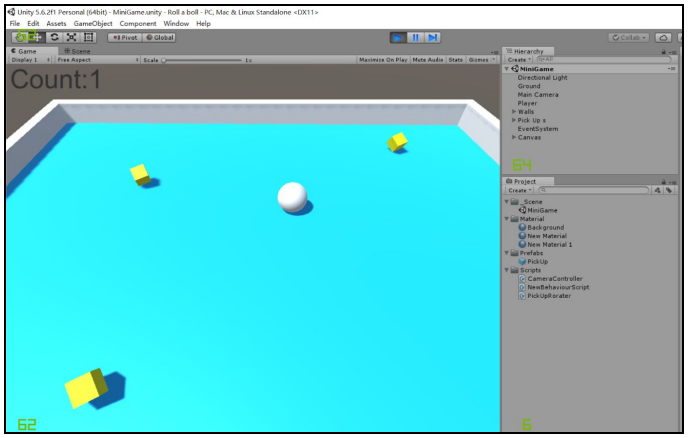
[1] *What is HCI (human-computer interaction)? - Definition from WhatIs.com*. (2017). *SearchSoftwareQuality*. Retrieved 9 August 2017, from http://searchsoftwarequality.techtarget.com/definition/HCI-human-computer-interaction

provided by a company named *MaxusTech*. *Welle* uses SONAR to detect human motion and thus uses this to interact with various smart devices and applications. Our task was to explore the ways that *Welle* could be used in a game and develop a new enriching and exciting gesture enabled game that used *Welle*.

Method

Design Process

For investigating purposes, we first made a trial game-Roll a Ball-using Unity 3D tutorials so that it would familiarize us with Unity's working and also provide us with a simple game to test the responses from Welle. This game just had a ball and a floor where the ball would move and collect all the gems.



A snapshot of the trial game

To control the ball, we thought of using a coordinate mapping system that could be used instead of the hex code directly. Thus, using the translated coordinates from the hex code, the ball could move in the direction of the finger motion welle detects.

We decided to make a 2D game as our **final game** because of the ease of using it with *Welle*. The game would be a multiplayer adventure game where each player would be controlled by one *Welle*. This idea of the game was chosen because it would enable the use of two *Welle*s and thus provide an exciting multiplayer platform.

The game starts with a menu scene, which includes the game name-*Cave Raider*-, the buttons of *Play* or *Quit*, and the controller to be used- *Welle* or *Keyboard.*

The characters in the game are a boy and a girl. Both of the characters and their respective animations were developed in Adobe Photoshop. We analysed many animations from other games to learn how pixel-style characters act. For each character we made 4 frames for the idle animation, 1 frame for the jump animation and 10 frames for the walk animation.

The background assets used in the game are taken from Unity Asset Store. The background's similarity to a cave was giving the right kind of atmosphere that we wanted for the game. Other assets such as the boxes were also taken from the Asset Store because of being readily available and free of cost.

We designed three levels of the game in an increasing difficulty in completing the levels. The players would need to collaboratively overcome the obstacles in each levels and reach the end point with as many collectibles (diamonds) as possible to finish with a higher score. The obstacles in the game include *spikes*, that the players would need to jump over, *two-way button-boards*, which pull out a bridge that serves as a way for the players to collaboratively cross large gaps in the game, and *boxes*, that may be pushed by the players to jump over high walls. There are also some "diamond" shaped collectibles in the game which the players may collect to get a higher score.

For the camera angle in the game, we first used a split-screen using *Normalized Viewport Setting* in Unity which would divide the gameplay screen into two for each player. However, we found that it was troublesome to look at two different screens for the players to collaborate. Thus, we found another solution by increasing the size of the normal camera and setting it up at the midpoint of the two players. Another variable stored the distance between the two players. A rectangular space was then set at the default view of the camera when the distance between the players was less than the size of that rectangle. When the distance between the players increased over the size of that rectangle, the camera would zoom out and form a bigger rectangular view, keeping both of the players in view.

The audio in the game is simple. Each level has a background audio that plays on awake of the level. Furthermore, the players have an audiosource that plays on hitting their respective coloured diamonds. There is another type of audio that plays when the players have completed level three, which is the end of the game.

A basic scoring scheme was also employed in the game. The were four scoring levels, *A, B, C,* and *F*. *A* being the highest score and *F* being the least. Pixel images of these scores were drawn on photoshop and then used in the game. The scores would be based on the number of diamonds collected by the two players. A public variable called *score* was used to count the number of diamonds. As the level would be over, the score variable would be checked and as a response to the number stored in the variable, a picture of the score would be enabled to the screen.

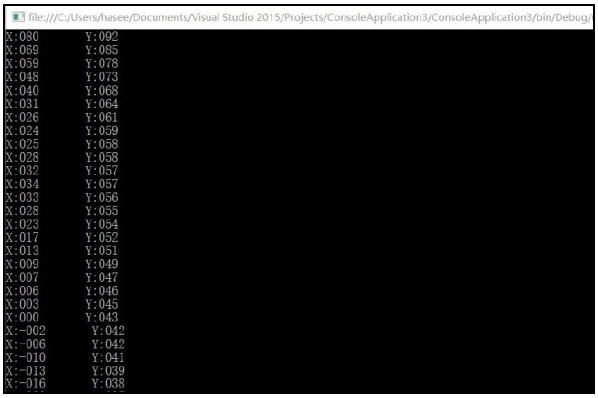
The start scene of the game

Implementation

Welle uses an ultrasonic wave that detects the human motion by bouncing back from its sensing area. The bounced back waves are then translated into specific commands using an advanced algorithms that converts them into Hex code. The Hex code can then be used by applications and other devices to perform certain tasks.

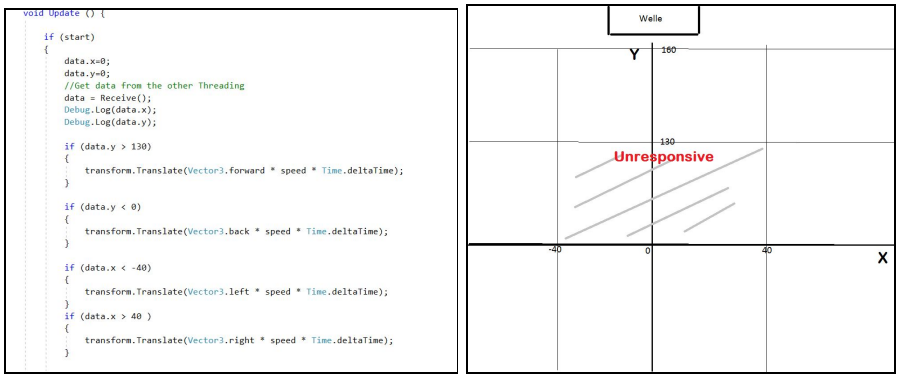
We first used a software called *CoolTerm* to get familiar with the responses of the device in Hex. We found that the responses were as expected but with some occasional delays and anomalies. Further, we used Visual Studio to communicate with the device in C# because in the future we would be using Unity (which uses C#) to make the device interact with the game.

In our attempt to interact with the device using Visual Studio, we successfully converted the hex code into decimal numbers and figured a xy-coordinate mapping that would be useful for interactions with the player of our game.



An image of the responses after converting hex code into decimal numbers and using a xy-coordinate map.

Our next task was to transplant the code used to interact with *Welle* into a game in Unity. After successfully transplanting the code to Unity, we found that there were six specific coordinates (2, 14, 16, 18, 35, and 48) that we were getting as anomalies in both the x and y coordinates. After repeated attempts to fix our code after trying different approaches, we discovered that these anomalies were only being received in Unity as the same code was working perfectly in Visual Studio. Thus, we decided to omit these anomalies by not reading them. We then decided to use the code in the game we had built using the Unity tutorials - Roll a Ball. We used the xy-coordinate to move the ball and found the code to be working fine with this game.

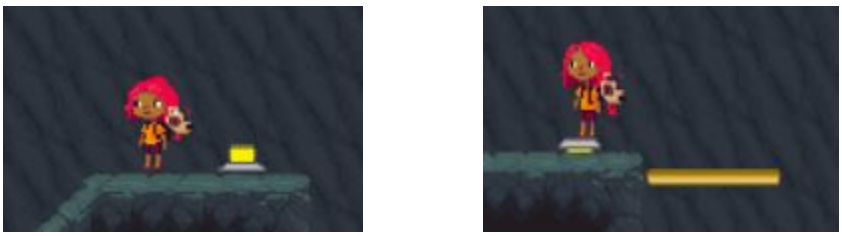


A snapshot of the code used in the Roll a Ball game and the coordinate mapping developed used Welle's responses in the Roll a Ball game

The grid points for the x and y coordinates were decided by trial and error method. After investigating many responses from *Welle* in Unity, we found that the coordinates from 0 to 130 for the y-axis were not accurate enough to be used to move the ball. Thus, only coordinate readings after 130 in the y-axis, were used to move the ball forward.

In the final game, the same coordinate axis was used to control the two players. However, instead of moving in four directions, the player in the final game needed to move only in 2 directions because of the game being 2D. Anything sensed above 130 in the y-axis by *Welle* was used for a "jump" action of the players in the game. The right and left movement of players were similar to the functions used to move the ball in these directions.

In the game, there were several functions built for the players to interact with other objects. To push the boxes, we initially tried to use a 2D Raycast, however we found that the push would be more realistic using the AddForce() function. Thus, we employed the onCollosionEnter/Stay/Exit2D() function along with the AddForce() function for the players to push the boxes. For the *button-boards* we used the onTriggerEnter/Stay/Exit2D() function and set the players as the trigger. When the button is triggered there is a boolean value named *OnPress* that would set to be "true". This value would then set the bridge to move out using a transform() function.



A picture of how the button-boards work

To make the game easier to handle during the live demo period, we added a few keyboard shortcuts in the game using the function Input.GetKeyDown(KeyCode.*key*). The shortcuts were mainly used for getting to different levels without having to play the game. The *F1* key would get us to the first level, *F2* to second, and *F3* to the third level, at any moment the respective keys were pressed during the game.

Evaluation

While setting up the communication between *Welle* and PC, we found that response of the "stop" request sent to *Welle* was never accurate. The response would always differ, and even after consulting with Calvin and Tom, we could not solve this problem. Thus we ignored the stop response and directly closed the serialport. Furthermore, we could not use the BLE (bluetooth) function in *Welle* as the device's bluetooth connection could not be discovered in Android, PC, or Iphone.

Over the course of trial and testing the game using *Welle*, we found that it was a little difficult to accurately control the players. We also called upon some of our friends to test the game using *Welle* and found that they had the same issue. Moreover, we also found that sometimes the game was receiving some random noise from *Welle* which again made the players very difficult to accurately control. Thus, the overall user game experience was not very good.

Apart from the occasional noise and the inaccuracy of the stop response, *Welle* was working fine with the game we created. We found that setting the communication between *Welle* and the PC was a little troublesome as we used C#, which was a new programming language for all of us.

We hope that the accuracy of *Welle* can be improved in the future. It would provide a great alternative platform for playing games if it's connection to pre-existing games and small devices becomes more convenient by using functions such as bluetooth.

Individual contribution

LI Xiang :

In this game, I'm mainly in charge of constructing controlling system with Welle. In the beginning, I achieved communication between PC and Welle through the example application provided by MaxTech, and had some basic ideas about the communication progress. Then regarding that we were going to build our game using Unity which mostly uses C# as the language for scripts, I started learning C# and SerialPort. Through many weeks tough work, I successfully finished the communication script. With the help of Prasenjit, we managed to decode the raw data to accurate position data after many trials. Then we transplanted the script to Unity and tried to reduce the noise caused by Unity, and made corresponding changes to be able to control our characters. Then I helped Hans to build our game by writing scripts for the obstacles such as PushableBox and Button. Finally I made efforts to improve the GameUI by adding a starting scene including game title and 4 UI buttons, and showing score, time and diamonds collected at the end of each level.

Appendix

First Level

Second Level

Third Level