Github Link: https://github.com/conorshortt123/gbui-final-project

Final Project Document

Year 4, Semester 2

CONTENTS

Conor Shortt – G00360253

4-1-2021

INTRODUCTION

This document outlines the development of my final project for Gesture Based UI Development, written in Unity and utilizing different technologies and patterns learned over the course of the past four years. In this document I will be outlining the:

- Purpose of the application
- Gestures identified as appropriate
- Hardware used in creating this application
- Architecture of the solution
- Conclusions & Recommendations

GITHUB REPOSITORY LINK:

https://github.com/conorshortt123/gbui-final-project

Unity Version Game was Developed with:

2019.4.21f1

PROJECT DOCUMENT:

Purpose of the Application:

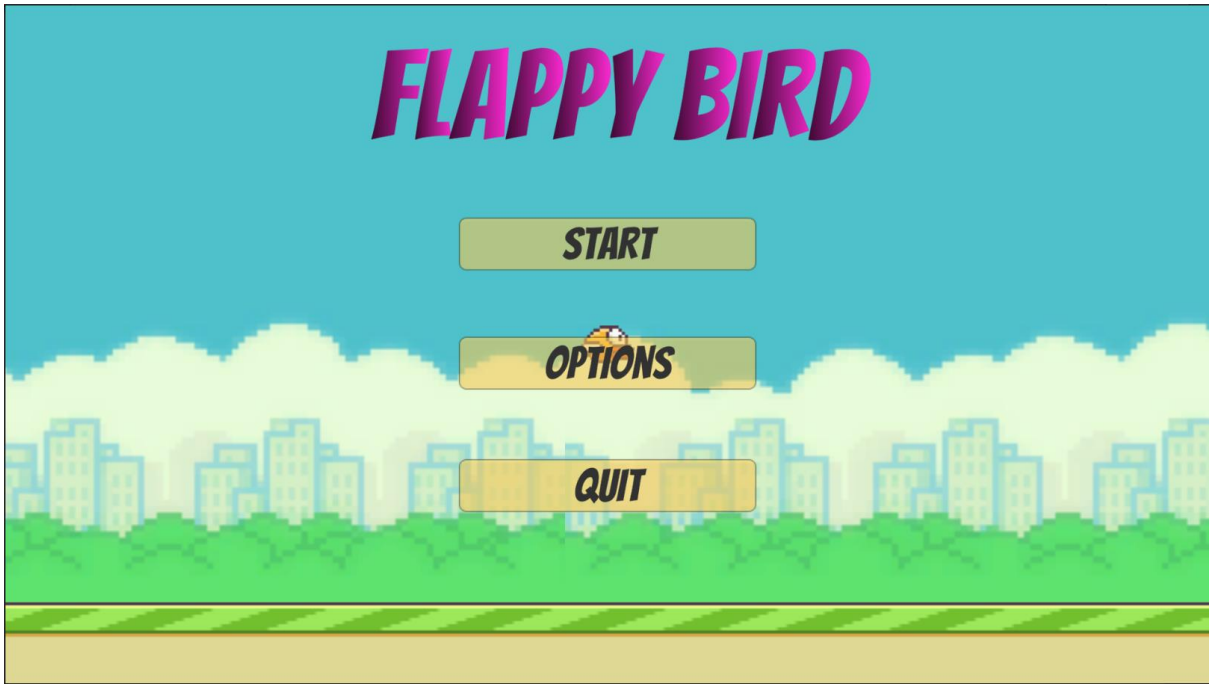
The purpose of this project was kept simple, clean, and effective. I chose to redevelop a very popular game, ***Flappy Bird.*** This was a mobile game, developed by a Vietnamese game developer called Dong Nguyen, under his company ***dotGears***. Flappy Bird is a side-scroller, featuring a yellow bird that is controlled by tapping the screen on a mobile device, causing the bird to flap each time. The player must attempt to fly between columns of green pipes without hitting them. The game was created in just a few days, using a bird he had for a cancelled game in 2012.

The game was released in mid-2013, but saw a massive popularity growth in 2014, as it became the most downloaded free game on the Google Play store and Apple Store.

The reason I chose to develop my own version of this popular game was because of its simplistic nature, and I figured it would be easy to incorporate a gesture-based interface into its control scheme.

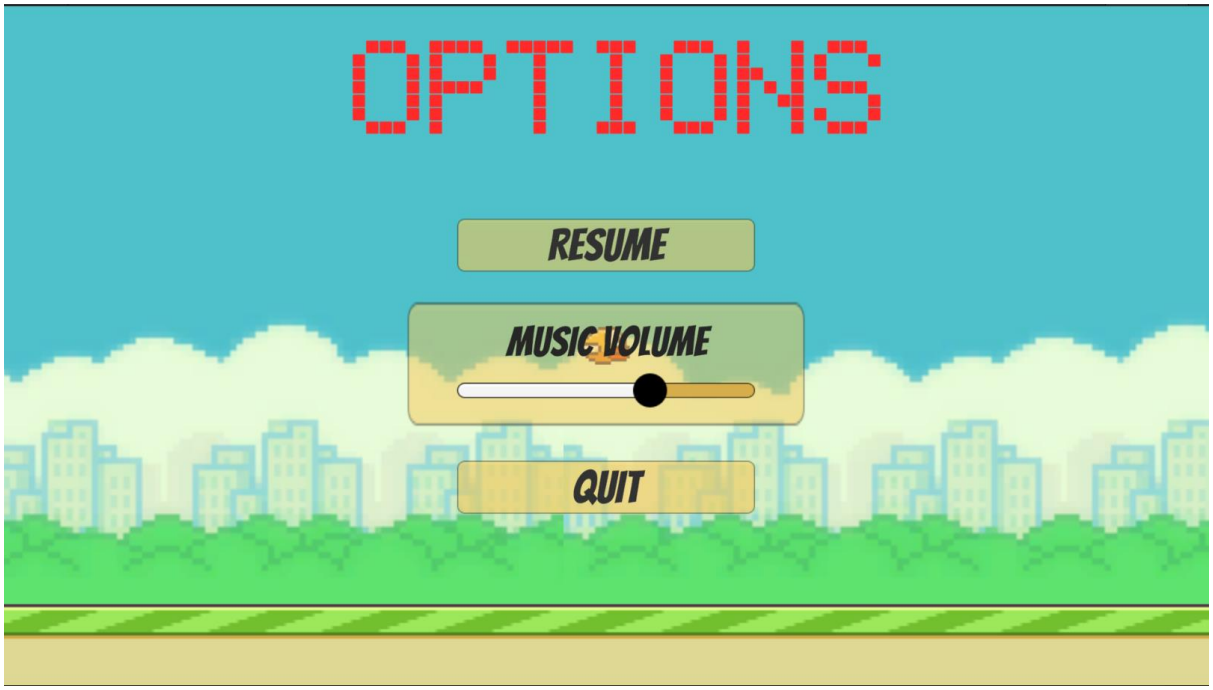
The game itself is controlled using a ***Kinect*** camera to control the bird's movements. Also implemented is a grammar recognizer that can be used to navigate the game menus. The reason I chose to implement grammar to navigate the menus was because after experimentation with gesture navigation with the Kinect, and navigation through grammar; I found the grammar to be significantly more efficient, and easier to configure.

Main Menu:



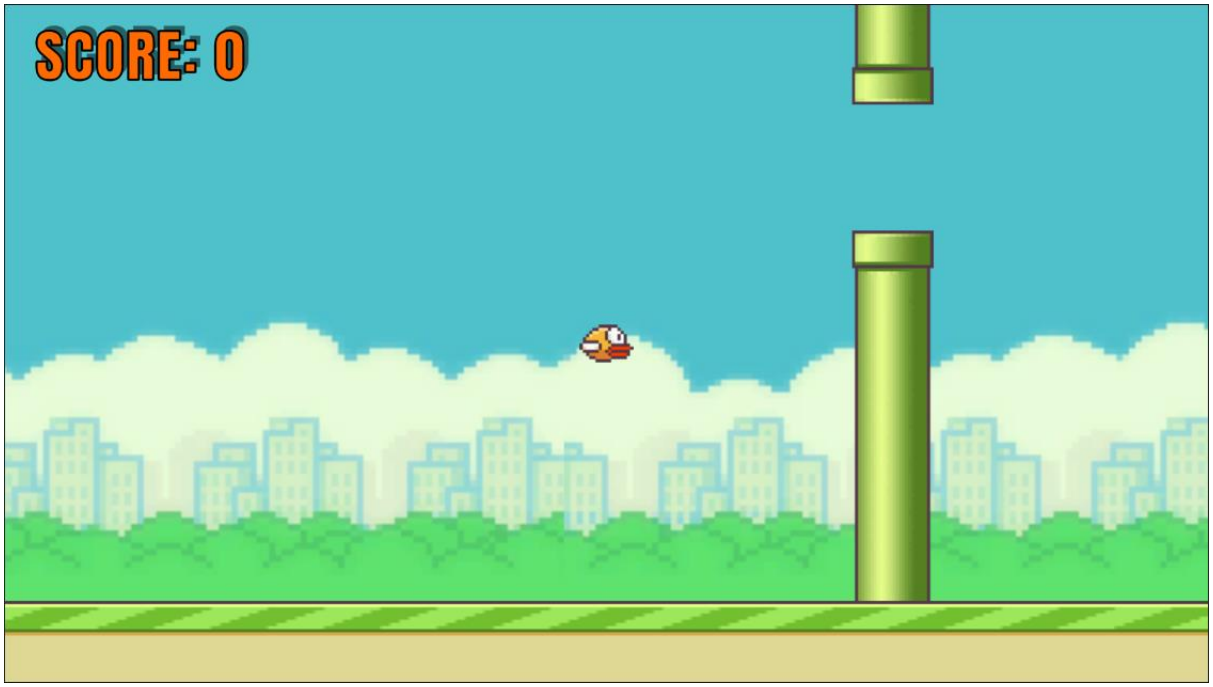
In keeping with the simplistic nature of the original game, the main menu is kept bare bones, with three buttons: Start, Options and Quit. Start begins the game, Options allows the user to access the options menu containing volume control and Quit leaves the application. The menu can be navigate using the mouse or can be navigate with voice commands such as: “Start the game”, “Options menu”, or “Quit Game”.

Options Menu:



A simple options menu containing only necessary functionality.

In-Game HUD:



The in-game heads-up display features only a score counter, which is the same as the original game. The Kinect is set up to track the player's right hand, which the bird will follow. Upon successfully navigating through a pipe, the player's score is incremented by one.

Gestures Identified as Appropriate for this Application:

Kinect Gestures:

In the early days of the game's development, the intention was to utilize Kinects gesture detection to detect a "Swipe up" gesture, which I felt was intuitive and easy for a new player to grasp understanding of. But after many days of experimentation the control over the bird in using this gesture, I determined that as Flappy Bird is a game of milliseconds where even the slightest delay in input can kill you, this gesture simply did not suit.

After much deliberation, I decided to change the control mechanism to a tracking controller. This was much more simple to set up, as the Kinect Manager script that I downloaded from the "Kinect with MS-SDK" asset contained a slot for a GameObject that could be tracked along with a choice for the body part to track with. This control scheme worked magnitudes better than the swipe up gesture. However, the implementation of this control mechanism significantly reduced the difficulty of the game. So, after some playing around with the game logic, I added some additional logic to the movement of the pipes so that the higher a players' score is, the faster the pipes move.

Voice Control Gestures:

After implementing the Kinect gestures into the game, I still had the classical method of using mouse to navigate through the menus. But after many iterations of running over to my pc, starting the game, and walking an adequate distance away from the Kinect camera for it to pick up my skeleton, I got tired of this and decided to implement the grammar recognizer that we developed as part of our voice & grammar project. After a lot of tweaking and adjusting of the grammar XML and Grammar Controller logic, the game could now be played entirely from 6 feet away, comfortably.

Hardware Used in Creating the Application:

Kinect Camera:

One piece of hardware used in creating this application was the Xbox Kinect camera. The camera was used to dictate the player's movement by making use of a tracking script that can follow any body part of a player including their legs, hips, right hand, head etc. Kinect is a motion sensing device produced by Microsoft and released in 2010. The hardware contained inside the Kinect includes RGB camera, infrared projectors and detectors that are capable of mapping depth through structured light calculations. Structured light is calculated when a known pattern is projected onto an environment. The deformation of the pattern is then used as input back into the camera which is then able to calculate a depth map. The device also contains Artificial Intelligence software for gesture detection, speech recognition and skeletal detection. The Kinect is an example of an **NUI** (Natural User Interface). An NUI is an interface that is effectively invisible and stays invisible as the user learns to interact with it.

Microphone:

Another piece of hardware I used was your standard microphone. Although a relatively simple piece of hardware that doesn't contain fancy sensors or software, when combined with an SRGS XML grammar, and a speech recognizer, it becomes a lot more. The microphone can be used as a means of interfacing with a Unity game. The reason I chose to implement this as an added interface was because the Kinect wasn't as fast as using voice based control to navigate menus, and I was also tired of walking back and over toward my computer to restart the game. The acronym SRGS stands for Speech Recognition Grammar Specification and is a W3C standard for how speech recognition grammars are specified. The grammar

specification tells a speech recognition system what it expects a person to say.

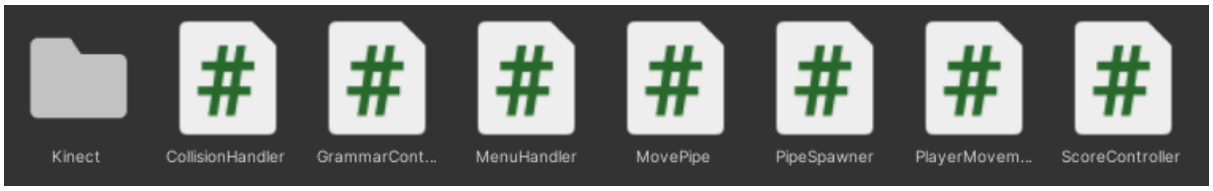
Architecture of the Solution:

Libraries:

- Kinect with MS-SDK – Controls all Kinect Gesture Detection, Tracking etc.
- UnityEngine.Windows.Speech – Grammar Recognizer used for Speech Recognition.
- TextMeshPro – Used for UI elements: Score, Menus, Buttons etc.

Scripts:

The architecture of the solution was kept simple, with only a handful of scripts to control the game logic. I wrote seven scripts for the game in total:



CollisionHandler:

Handles logic for a trigger box collider in between each pipe, that when the player passes through the player's score is incremented by one.

GrammarController:

Controls all the logic for voice recognition. A Grammar Recognizer is instantiated and passed in the XML file "GameGrammar.xml" as its SRGS. Once a phrase is recognized by the GrammarRecognizer it's added to a dictionary with the action being passed into the value variable e.g "start". This value is then passed into a switch statement which handles function calls. All the function calls are made to the external class *MenuHandler.*

MenuHandler:

Contains all the methods for menu navigation and canvas elements. Including: Death Menu, Options, Start Menu, Controls, and the in-

game heads-up display.

MovePipe:

A simple movement script for the pipes. The movement is increased gradually over time in correlation to the player score using the formula speed = speed + (score / 10);

So, if the speed is three and the score is ten, it would evaluate to four.

PipeSpawner:

The logic for the pipe spawner was simplified by creating several prefabs with different heights and opening sizes. The prefabs were then added to an array of GameObjects, which the script then instantiates at random, every two seconds using the InvokeRepeating function.

```
void Start()
{
    InvokeRepeating("SpawnPipes", 0, 2f); // Spawn two pipes every 2 seconds
}
```

PlayerMovement:

Controls bird movement, changes of sprites depending on velocity, and collision detection.

ScoreController:

Controls incrementing of score, with public getters and setter functions.

Conclusions & Recommendations:

In conclusion, the development of new user interfaces is something that will continue to happen, and I look forward to seeing the new and exciting ways of developing games in years to come. In relation to my own game, I learned a significant amount about natural user interfaces and how they can be implemented both well or poorly depending on the choice of gestures, layout and so on. A possible recommendation would be to increase the complexity of the game by adding more levels, game logic could also be improved by adding more variability in pipe distances, spawn times, and perhaps some sort of powerup capability. Overall I enjoyed developing the game and I feel it was a worthwhile project to get some experience of developing applications that are utilized by something other than a keyboard and mouse.

Test Plan Outline:

Scope of test plan:

For the scope of my test plan I'm seperating the game into two areas. The game itself and then all of the UI elements (menus, buttons, score trackers etc).

For the UI elements I plan on implementing **Integration testing**, which is used to expose defects in interfaces and in the interactions between components or systems. I will test the following features:

Game Requirements: (Features to be tested, see Test Plan document)

Requirement ID	Feature Name	Requirements
1.0	Main Menu	Play button and voice commands brings user to game and game begins. Options button or options voice command hides main

		menu and shows options menu. Exit button or exit voice command quits game.
1.1	Options Menu	Music volume slider adjust in game music. Back button brings user back to main menu. High score is displayed top right.
1.2	In-Game UI	Player score is being displayed and updated on top left of screen.
1.3	In-Game Pause Menu	Pause Menu Shows user stats and allows user to navigate to main menu
1.4	Controls Menus	Play button OR spoken voice commands for "Play" grammar brings user into game play scene.
1.5	Death Menu UI	Player score is being displayed on screen, restart and quit UI buttons displayed, buttons allow user to restart game and also voice commands also user to restart.
2.0	Kinect Tracking Mechanic	Player model follows right hand path. Ensure a sufficient distance from Kinect camera.
2.1	Collisions	Upon colliding with an obstacle, player dies.
2.2	Score	User score incremented by one upon passing through pipe opening and score counter UI component is subsequently updated.
2.3	Death Event	Death event is incurred with collision. Death menu is shown and in-game GUI is hidden. Player can play again, enter options menu, or quit game

3.0	Voice Controls	Upon saying "Play Game" the game starts. When user says "Pause Game" the game should pause and an options menu should be displayed. Music is muted upon saying "Music Off"

REFERENCES:

Assets:

Kinect with MS-SDK –

https://assetstore.unity.com/packages/tools/kinect-with-ms-sdk-7747

Flappy bird assets –

https://github.com/samuelcust/flappy-bird-assets

Music:
https://www.bensound.com/royalty-free-music/track/dreams-chill-out

Unity with Kinect Tutorial –

https://www.youtube.com/watch?v=aHGlLxh6a88&ab_channel=VRwithAndrew

SRGS Info –

https://en.wikipedia.org/wiki/Speech_Recognition_Grammar_Specification

Flappy bird Info –

https://en.wikipedia.org/wiki/Flappy_Bird