BioKinect

An educational motion controlled game using Kinect for Windows

Nicholas Kolunov

Final Year Project - 2014
B.Sc. in
Computer Science and Software Engineering

NUI MAYNOOTH
Ollscoil na hÉireann Má Nuad

Department of Computer Science
National University of Ireland, Maynooth
Co. Kildare
Ireland

A thesis submitted in partial fulfilment of the requirements for the B.Sc. Single/Double Honours in Computer Science/Computer Science and Software Engineering.

Supervisor: Dr. Joseph Timoney

Abstract

There are so many ways in which we can learn things in today's world. Modern technologies such as tablet computers and smart phones are starting to gain popularity in their application in the class room. Pen and paper are being paralleled with smart boards and touch screens.

BioKinect is my take on another unique way to teach – through the use of interactive games. From the beginning of my computer science days in Maynooth University, I had a keen interest in human-computer interaction. The Kinect sensor provides a unique paradigm of interfacing a user with an application. BioKinect attempts to use all of the strong aspects of this paradigm (ease of use) while working around the weaker sides of motion control (such as latency and accuracy).

This project report describes all of the steps I undertook, in order to realise the user centred game I titled BioKinect. In the end of my report, I detail things that worked out well and list a few things that I feel I could have improved if provided with more time.

Acknowledgements

I would like to thank the staff of the computer science department at Maynooth University. My supervisor Dr Joseph Timoney and the CSSE course coordinator Dr Charles Markham for their advice and guidance during the development of this project. I would also like to thank my mother for sharing her ideas and feedback at various points of the project's lifetime.

Table of contents

Structure of report:

- References are marked with a number enclosed by square brackets, for example [1337]. All references can be found in Appendix A. Note that only the first instances of terms are linked with a reference, repeating terms will not be followed by a number in order to reduce clutter.

- Things that may need further explaining are marked with a superscript number such as [1337]. These are explained in Appendix B or explained on the bottom of the same page.

- All acronyms are listed and expanded in Appendix C. I have decided to include explanation of all acronyms and do not assume that the reader knows even the most common definitions.

- All figures are captioned with a label number and summarising text. For example (Figure 1. Figure). The full list of figures can be found on page…

1. Introduction

This chapter should give the reader an idea of what sort of project this is and what was it's goal.

1.1 Goals

One of the requirements of the project was to make it education focused. There are plenty of things I could possibly base my project around. Education is very broad and I had to find subjects which would work better with the motion controlled aspect while also being innovative in a way. This was where I spent a few weeks trying to filter out subjects that would be too challenging or not intuitive while being presented with a motion controlled interface.

Initially I was planning to make a geography based game and that is the idea I presented at my interim presentation. However, after much work I realised that interfacing a WPF[16] and Kinect Interactions application with a web based app was not as straight forward as I hoped. Even with using some clever tricks like piping the Google Earth app through via a WPF frame, binding the Kinect interaction events with Flash and JavaScript was not at all intuitive and some things like the grip events could only be interfaced using third party libraries. I Had to change my project to be based on biology instead.

1.2 Motivation

My motivation to choosing this project was my interest in Human Computer interaction and user focused design in systems. The Kinect sensor and its possible applications on a personal computer have a lot of potential. A bonus to my interests was that I had previous experience working with the .NET[1] platform and it's family of applications. I was comfortable enough with C# and Visual Studio. Because the Kinect is Microsoft's hardware and uses .NET technologies I had some idea of what kind of developer tools I will be using. These tools combined with Kinect ca produce potentially very interesting systems and give a new way of interacting with them.

1.3 Method

As you can see from figure 1, I had a few potentially good choices. Anything that had a Medium and Good rating came under my consideration. After realising that a geography based

game is not that straight forward to make. I decided to switch over to a new project based around a different subject. I went with the biology game about the human skeleton. It seemed quite intuitive to use motion tracking for its interactions. The Kinect has a separate skeleton data stream hence there would be no need for any third party libraries in order to extract the data I needed. After some brainstorming I developed a clear picture of how the system will work. This idea became BioKinect.

I thought that an agile-like approach would suit this project best. Since I am the only team member, I started with a small specification and would review it at the end of each four-week 'sprint' by removing things that are implemented and adding things that still need to be added.

1.4 Overview

Kinect for Windows[2] is an advanced sensor that uses a set of IR cameras and a colour camera as well as an array of four microphones to sense depth, user's skeletal position and recognise speech. The sensor has its own dedicated SDK[3] (current version 1.8) provided by Microsoft.

Using this SDK, it is possible to write two types of apps which are Kinect compatible. First is the XNA approach which is Microsoft's framework for lightweight games. Second is the WPF approach, which is the framework for standard Windows applications. Both of the approaches use the .NET family of languages. Kinect's SDK is compatible with C#[4] and Visual C++[5] (Microsoft's .NET version of C++ which is also known as 'Managed C++'). I chose the WPF and C# approach, because the things I wanted to do were more straight-forward using this particular combination of tools. I will justify my choice throughout this report.

1.5 Report Overview

In this report I will describe the development journey I took in order to meet a set specification. Throughout the report I will attempt to cover both the high and lower technical details and concepts. I assume no prior experience with Kinect and its SDK. However, I assume the reader is at least aware of C# and certain general programming concepts. Where applicable – diagrams, additional explanations and citations to further reading are included. (Please see Page 2 'Structure of Report' for a guide on how to identify each citation.) While this is a large report, sometimes getting to the point is more important than grinding in minute details but the reader has the option of delving into the technical depths if he or she so desires.

1. Background and set specification

In this chapter I will give the idea of the kind of project I wanted to make and what similar games are already available on the market. I provide details of Kinect internals to help the reader understand what sort of input device my project will be using and how it can be used. I note some sources which provide information on why game based learning is so successful and what are some of the strategies to follow to make a fun and innovative educational game.

2.1 Introduction

The requirement of the project was to design an innovative and educational motion controlled game. Kinect provides very different ways of interacting with applications. The use of Kinect in GBL is nothing new. (I provide examples of such existing apps in Part 2.2). These games are very successful with younger kids as they provide an interactive way of learning.[8,11] I needed to identify my target age group and the academic area I would like to target.

2.2 Literature Review

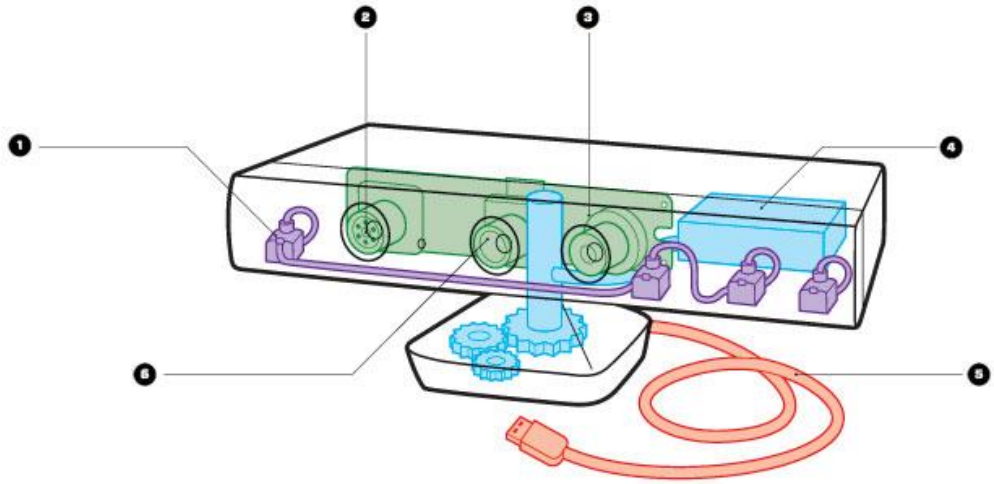
There exist full fledged Kinect enabled educational games made by professional development teams and published by companies on the Xbox 360 console. These games target various age groups. For example the game "Body and Brain Connection"[6] is more targeted at an older audience while "Kinect Sesame Street TV"[7] is meant for small children. Along with these large games, exist smaller games made by lone developers or a small team of developers.

One such game is "River Crossing"[9] made by a small team called Kinems, a team of developers who focus on developing educational games[10]. This is exactly the kind of game I would be aiming to develop. Its is straight forward and focused while at the same time interactive and educational. Sadly, I do not have the artistic skills to make my game as colourful as River Crossing, but it showed me that there do exist educational Kinect games that are focused in a smaller area than the large published games on the market, albeit being a niche market. Both category of games follows some common rules such as intuitive interfaces which can be operated without of use of controllers or mouse and keyboards.

2.3 Specification

2.3.1 Overview of the Kinect sensor internals

The Kinect sensor consists of three cameras and a four-microphone array. The full internals are presented below.



(Figure 1. Kinect sensor internals [21])

1) The four microphone array which make the Kinect capable of recognising various speakers and the direction of the sound.
2) IR emitter projects a pattern of infrared light into a room. As the light hits a surface, the pattern becomes distorted, and the distortion is read by the depth camera.
3) Depth camera which reads the distortions to produce a 3D map.
4) Tilt motor which adjusts the angle of the sensor depending on user's height
5) USB 2.0 cable which pipes the data down to the machine. Note that the sensor requires and external power source. The USB does not provide the hardware with enough power.
6) RGB camera that can be used as a colour camera to view the user.

2.3.2 Project specification

The objective is to develop a game that can be used in a classroom. Is intuitive to use and user/child friendly. The game should focus on teaching a single subject and should do so in an innovative way by means of motion control.

The interface should be operable without the use of mouse and keyboard or controller. The Kinect sensor is the primary input source with C# and WPF as the back-end of the application. The game should not be complex to require a manual to operate i.e. the rules are straight forward to follow. There should be a feedback system rewarding the player for completing objectives. There should be a way to lose at the game. The use of the game should not be physically exhausting.

3. Project Management

In this chapter I describe how I managed my time to have sufficient space for research, development, testing and evaluation.

3.1 Approach

My planning started with a list of subjects I could possibly include in the game. Below I present the list of subjects I looked at.

Subject	Compatibility with Kinect[EA]	Comments
Chemistry	Poor	Don't see a concept for this
Biology	Good	Can make some games about the skeleton and its parts
Mathematics	Poor	Don't see a concept for this
Computer Science	Poor	Don't see a concept for this
Physics	Good	Could make an astrophysics themed game
Geography	Good	Could make an exploration simulator or an interactive globe
Languages	Medium	Could make a game that teaches a language. Need to think about the motion controlled aspect

(Figure 2. Subject list)

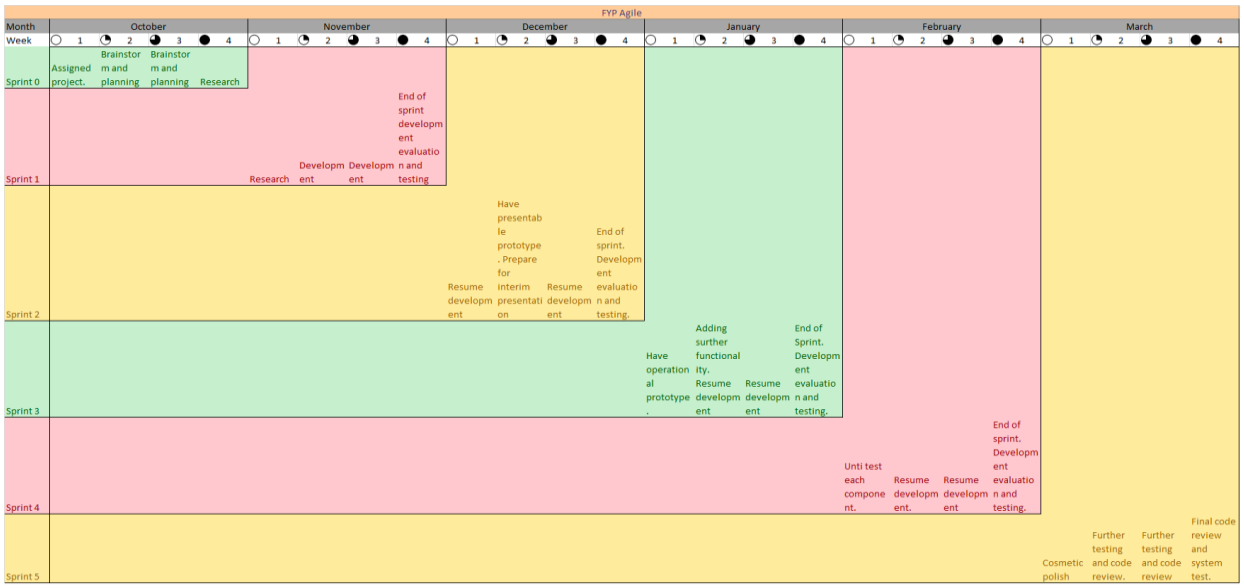
The way I evaluated a subject's compatibility with Kinect is explained in Appendix B.
I thought that this is a good starting point because knowing the subject I would develop for, would guide me in the direction of the sort of game I will make. It forced me to think of concepts for the sort of game I will be making. With these prototype ideas, I could tell if a certain subject would work well or not so well or in fact, not work at all.

After choosing what I think is the correct subject, I would solidify the prototype specification in a loose draft. This would help me in establishing in what sort of Kinect and WPF functionality I will need to utilise. It would also give the feeling of how complex this certain functionality is to implement using

Kinect or WPF. Basically, I would try to make pseudo Hello World applications which do a separate part of the game I want to make using Kinect and WPF. Being completely new to Kinect, I had to learn its boiler-plate level code as well as the way it interacts with WPF. This was a rather lengthy process made a little less painful by the MSDN[12] and StackOverflow community.

3.2 Initial Project Plan

Being a one man team, I thought that an agile approach would best to follow. A consequence of choosing this cycle is the flexibility of certain artificially set dead-lines. In other words, if something unexpected was to happen, I could accommodate for it in my end-of-sprint review and plan my next sprint accordingly. Below I present my initial Gantt chart.



(Figure 3. Overall Gantt Chart)

You can find a full version of the chart in a .xlsx format attached in Appendix D. Because of the rather short development time, I decided to have four week sprints with a sprint review at the end of each month (week 4 of a month).

A few notable milestones from this chart: December – week 2. I proposed to have a presentable prototype by that time. If no such prototype is possible, I would still have time to go back and re-evaluate my chosen system. This feedback system is one of the perks of agile methods and I found it to be very helpful. February – week 1. I proposed to unit test all of the components I have in my system by that time. I thought this would be a good time to push for stability with the release being just a month away. This way I could focus on polishing the system cosmetically during the pre-release period.

March – week 4. I proposed to go though a complete code review, before the release of the source code. Maintainability is a very important aspect of good software. Having readable and self documented code is essential to that. Just before the release, all of my components would be in place and the call hierarchy is established. Having the entire system in this static state makes it easier to documenting it.

3.3 Problems and Changes to the Plan

Sprint 0 went exactly according to plan. I was looking into all the possible shapes my games could take while also learning the Kinect hardware. At the beginning of Sprint 1, I began work on a prototype for my geography game that would track the users palm gestures. This proved to be a challenge and I had to spend some time in learning how to make an XNA application to detect a sensor and also display the user's palm and its status. Progress was slow and I realised that I would not meet the proposed prototype deadline in December.

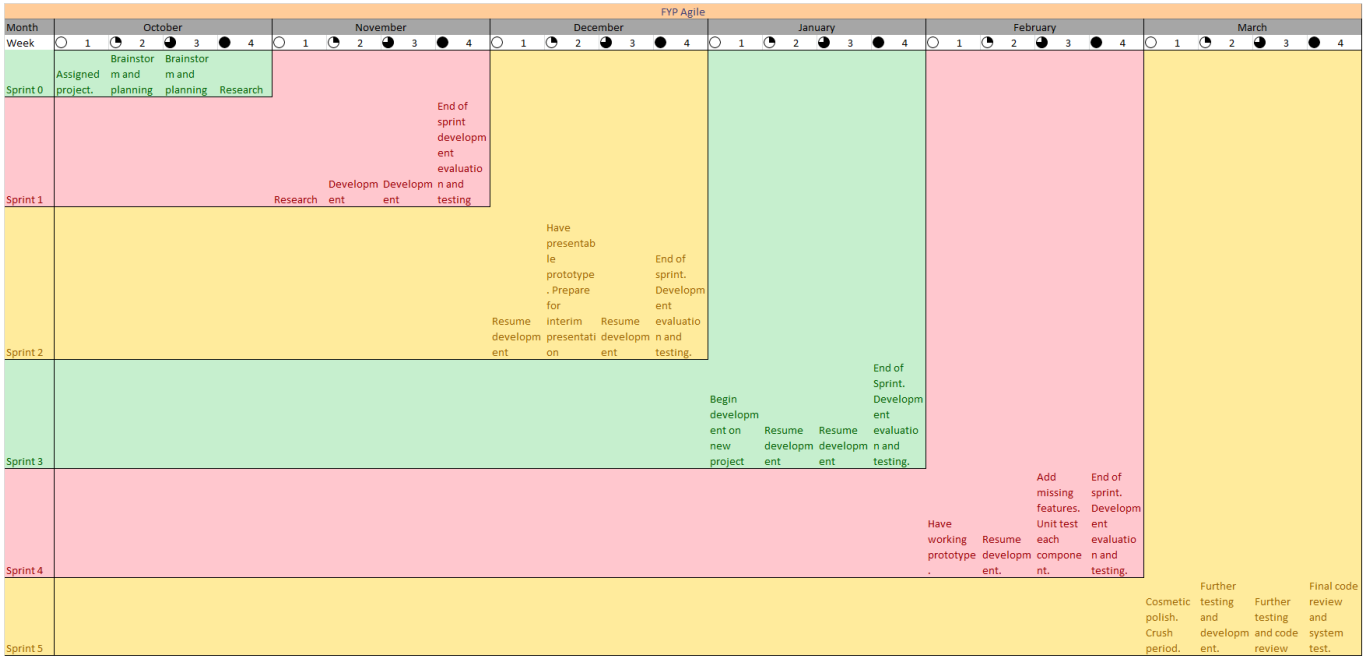
At the end of November I did get a working XNA app that could track palm position and gestures. My next challenge was to interface the tracked data with a web application. This is what made me re-evaluate the entire system and switch over to a different game entirely. My main reason for the switch was that I could not find a straight forward way of getting Kinect data into a Flash[13] application such as Google Earth[15].

While this was a disappointing experience, I had enough time to start over. Following an agile method, I re-evaluated what worked and what didn't. I began work on a new game. This time, I knew what was straight forward and what was not so straight forward using Kinect. One major change, was the switch to WPF instead of XNA as the main platform. XNA was one of the parts that just didn't work and had to be left behind. Some of the code I wrote for my geography game could be either directly copied into WPF or at least ported[EB]. This meant that I did not begin work from a total zero. Most importantly, the things I learned could be reused in this new biology themed game. As a consequence of that, I was able to stick to my initially outlined plan and timeline.

3.4 Final Project Record

After my switch over, my plan only had to change slightly to accommodate the extra time needed for testing and cosmetic changes. This period of testing towards the end has resulted in some crunch-time nearing the release. Once again, the sprints were 4 weeks long with testing and

a review of progress at week 4. Figure 3 represents the actual Gant Chart I followed. Figure 4 represents the project plan I followed.



(Figure 4. Final Gantt chart)

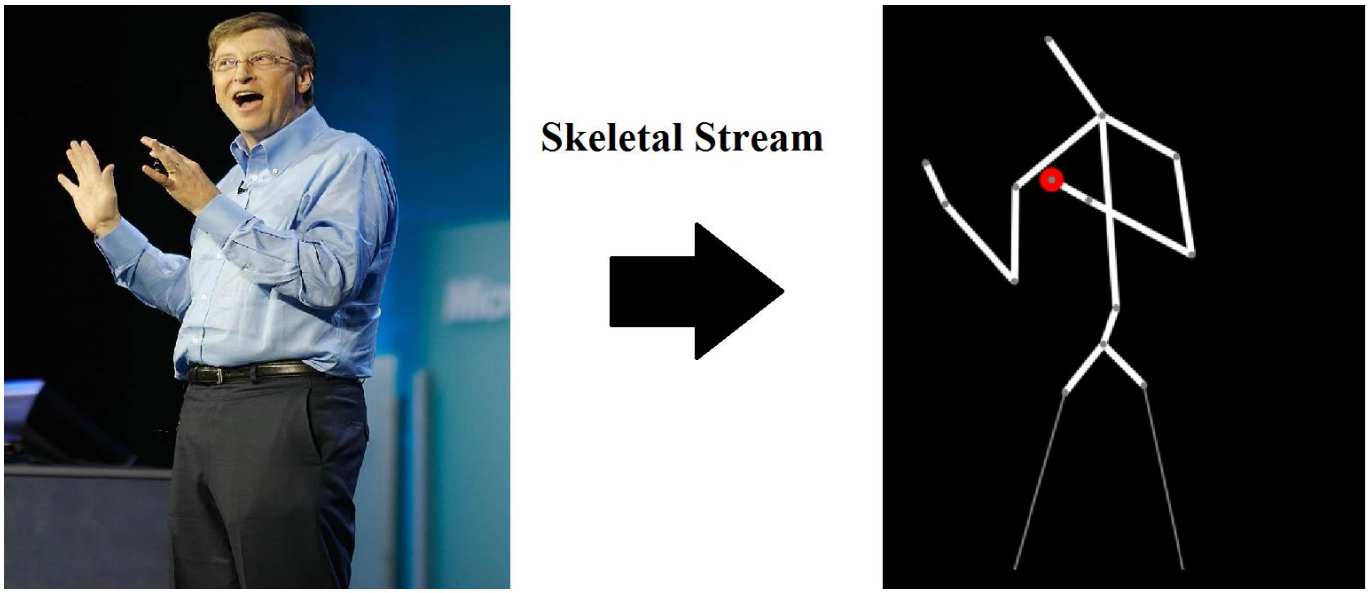
Note that only the later half of the plan had to be revised as prior to my system change, everything was going according to the set plan. In this revised plan, my working prototype deadline was moved to February – week 1. As a result of this move, the unit test date had to be pushed ahead to February – week 3. Sprint 5 remained largely intact for the exception of week 1 where I had to include a crunch week in order to be able to perform a complete code review before the release date.

Following this plan, I had produced a total of five progress reports. These are included in Appendix D. Each report represents a) what has been done, b) what needs to be done and c) the testing results (if any tests were performed prior to the writing of that progress report.) and any feedback of the sprint.

4. Analysis and Solution

This chapter describes the high and low details of my project. This will describe what I started working with and finally show the result of my work. I make extensive use of diagrams and annotations which I hope will help the reader in understanding the context.

4.1 Problem modelling



(Figure 5 [18]. Converting analogue data into a digital representation)

It all begins with a user standing in front of the sensor. Kinect is able to extract all of the data that each of its cameras detect per frame. This data (Skeletal frame, colour camera image, voice) can be packaged up and used in some processing, otherwise they are discarded on the next frame.[19] BioKinect is focused on the skeleton hence it processes the skeletal frame data.

At this point, its worth mentioning some limitations of the sensor. The Kinect v1 device has no finger tracking due to the low resolution and the Xbox 360 version of the Kinect used for this project has no 'near mode'. With this in mind, I had to make sure that my game will not require either of these.

Neither of these weaknesses had any great impact on the overall game design. My main focus was on the skeletal tracking, especially the joint tracking. The Kinect v1 with SDK 1.8 provides access to 20 joints. This is just enough for my particular application. Below I present the step-by-step Use case of BioKinect.

4.1.1 BioKinect Use case

1) The executable is launched. All of the necessary WPF components are loaded into memory and then rendered to the screen.

2) The Main Menu window launches the SensorChooser() method which sniffs the USB ports for a potential Kinect sensor it can use. The first Kinect found will be the one it uses. Ideally there is only one sensor available.

3) The dedicated Kinect sensor is chosen and its IR emitter now waits for a user to step into the operation envelope.

4) Once a user is detected, he can begin using the UI by the use of his hand as a cursor.

5) There are two KinectTileButtons presented to the user at the Main Menu as well as a KinectScrollViewer with instructions on how to play the game.

6) The user presses the 'Learn' button. A new screen opens which performs steps 2 to 4. The Learn screen presents a model of the human skeleton where the user can learn about the various bones in the human body.

7) The user presses the 'X' button in the 'Learn' screen. The screen closes and Main Menu re-opens.

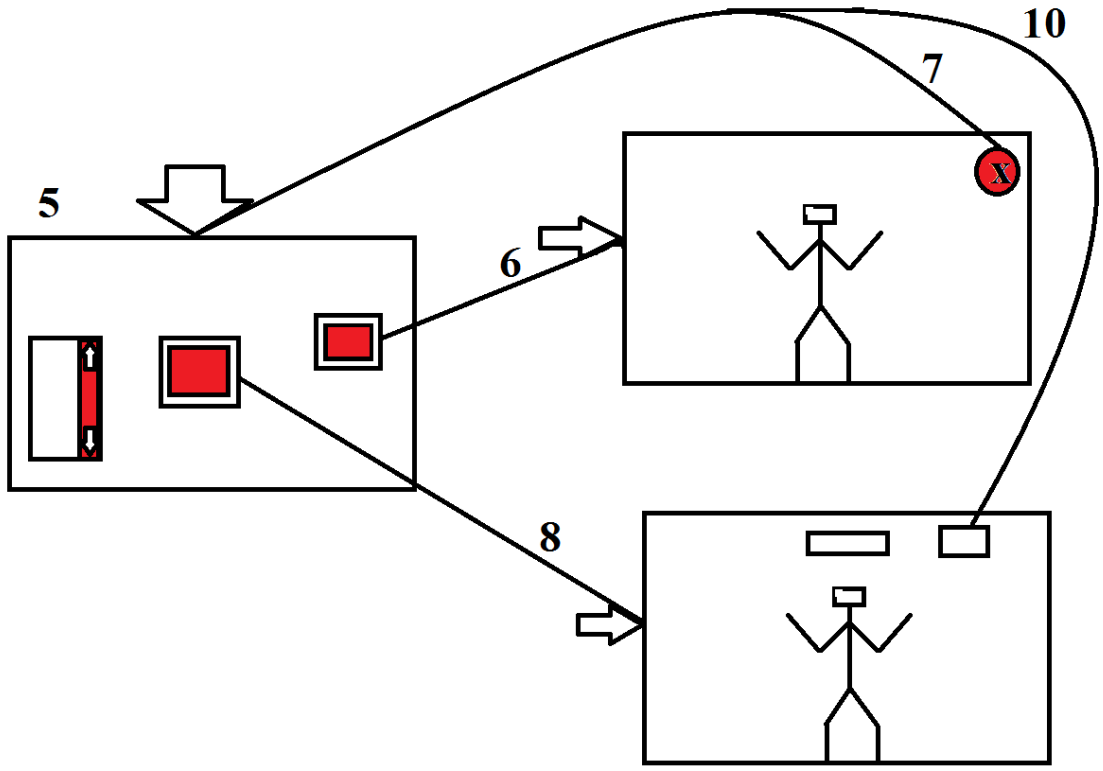
8) The user clicks on Play BioKinect. The main game screen opens.

9) The game screen performs steps 2 to 4. As soon as a user is detected, the game begins. The user is asked to point at a bone. A countdown timer is initialised and the current score is displayed.

10) If the time runs out, the game is over and the user's final score is displayed. Closing the score message should return to the Main Menu.

4.1.2 BioKinect Use case mock up diagram

Below is a mock up diagram that follows the Use Case steps. Numbers indicate which Use case action was performed.



(Figure 6. Diagram of the use case)

4.2 Non-functional requirements

- The latency of motion detection should be kept to a minimum
- The game should accommodate for user positioning errors made by the sensor
- It should be easy to expand functionality beyond just the human skeleton
- The error present in motion detection should be accounted for in the layout of the UI

5. Product/System Design

This chapter details the design decisions I made and my justification of these choices.

5.1 Introduction

I chose and Agile development cycle which would suit a one man team perfectly. Agile pattern also made sense because I had no previous experience with Kinect and it would allow me to adapt to sudden changes in my project plan.

The process began with exploratory programming. By making mock-up mini games which were themed around chose subjects. These basic ‘paper models’ contained the most necessary user interactions which gave me an idea of the kind of game this will result in. After the best mini game was chosen, I proceeded with solidifying its specification by outlining exact requirements and developing a more solid user task list. I mainly focused on geography as it provided a lot of possibilities. An interactive globe was my initial idea which was later scrapped.

I instead chose to work with skeletal tracking because it immediately trigger an education application in my mind. The main application is tracking the user and asking him or her to point at various bones on their body. This is done by calculating the midpoint between specific joints of the tracked user. By using just the data streams that the Kinect provided, I could avoid using any third party libraries on top of the Kinect SDK.

5.2 Product Features

BioKinect makes extensive use of the Kinect interactions toolkit and the skeletal tracking data stream.

1) The user is presented with a main menu screen where he can choose what part of the application he desires to use.
2) The application will accommodate for the Kinect sensor plugged in during run time regardless of which screen the user is on
3) The user has the option of using a mouse to interact with the UI.
4) The application will give feedback on the Kinect’s connection status

5) The user can back out of any screen at any point in time.
6) The game will not begin until a full skeleton is detected by the Kinect sensor

5.3 User Interface

What follows is a list of the action-effect events of the UI of BioKinect

1) There are KinectTile buttons, pressing these will open their corresponding windows and close the screen the user is currently on.
   Action: User's hand is displayed as a cursor on the screen. By positioning the cursor over a button and pushing their hand towards the sensor, the button will be pressed.
   Effect: The current screen is closed and a new corresponding screen is opened.

2) There is a KinectScrollViewer used to display instructions on how to play the game.
   Action: User's hand is displayed as a cursor on the screen. The user is able to grab the scroll window with their hand,
   Effect: The user grabs the window and is ale to scroll up or down by moving their hand up or down (y-axis)

3) There are three KinectCircle buttons. Each button represents a difficulty level.
   Action: Same operation as the KinectTileButton. The button is pressed by moving the hand closer to the sensor while having the cursor hovering over the button.
   Effect: The difficulty level of the game is changed depending on which button is pressed. The difficulty indicator will point to the set difficulty level.

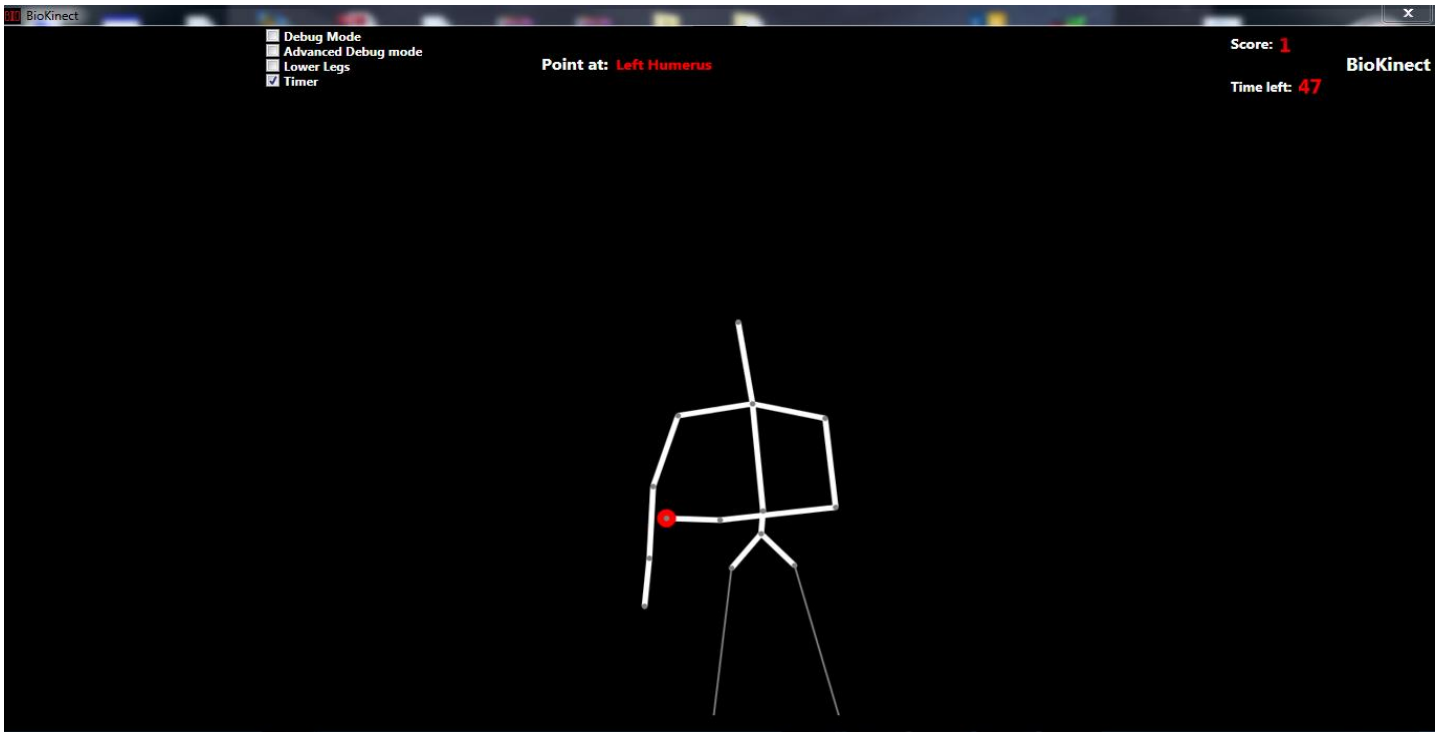
5.4 Design Verification

Below is a walkthrough of a single game of BioKinect.

1) The executable is launched and the user is informed that the Kinect sensor is ready.
2) The user uses the scroll viewer on the left of the screen to view the game instructions and learns how to play the game.
3) The user uses their hand (left or right) to select a difficulty level (easy, medium or hard) using the circle buttons on the right of the screen.

4) The user presses the Play BioKinect button
5) The game screen opens and the tracked user skeleton is displayed, the timer begins to count down.
6) The user points at a requested bone and earns points which are added to the score, displayed in the top right corner of the screen.
7) The timer runs out and the game stops.
8) A game over message is displayed that tells the user that the time ran out andwhat was his final score.
9) The user presses okay on the game over message and is returned to the Main Menu.

Below is the final UI of the main game. The objective is shown in the top centre. The current score and remaining time is displayed in top right corner. Debug options are in the top right of the screen. The player's current skeletal state is rendered on the bottom middle of the screen.



(Figure 7. The final version of the BioKinect game screen)

6. Software Design

This chapter will describe the High and low details of the internals of BioKinect. Every design decision is justified through the use of high and low level analysis of components.

6.1 Introduction

6.1.1 The back-end

Central to the entire game is the skeletal tracking data. All of the game resolves around processing that data and depending on the position of the requested bone and the coordinates of the pointer hand, the game would perform different actions. The SDK contained all the necessary objects required for analysis of skeletal data as well as some useful methods that can be applied to those objects. All that was missing were the methods operating on the data. Joints and Points had the necessary methods of data extraction from them, I wrote the methods to perform calculations on the data returned. The entire back-end is written using C#.

6.1.2 The front-end

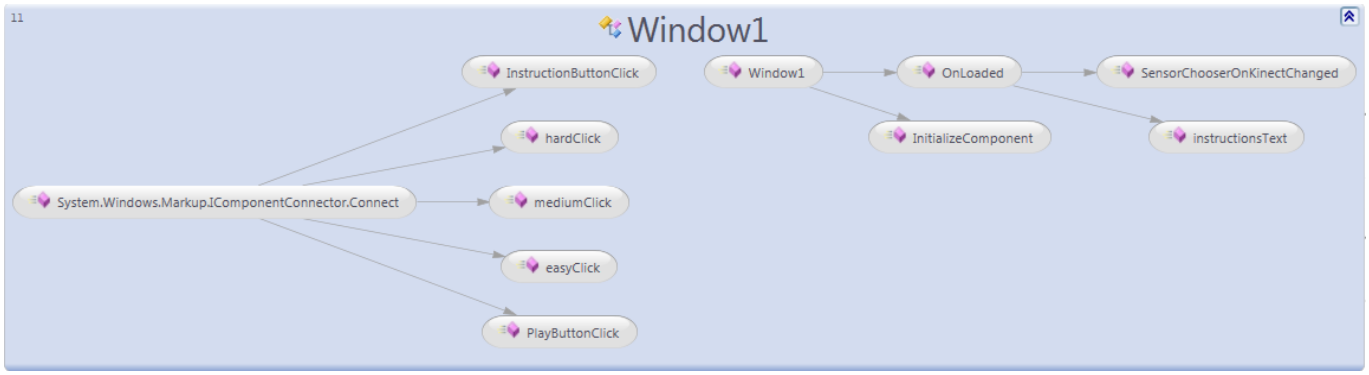
With WPF being the platform fro the entire application, all of the GUI is defined using XAML [20]. XAML was very easy to use. I wanted to make an intuitive, tidy UI and XAML let me do just that. Using XAML I could easily reposition UI components and resize them. Both of these abilities were vital to the success of my project.

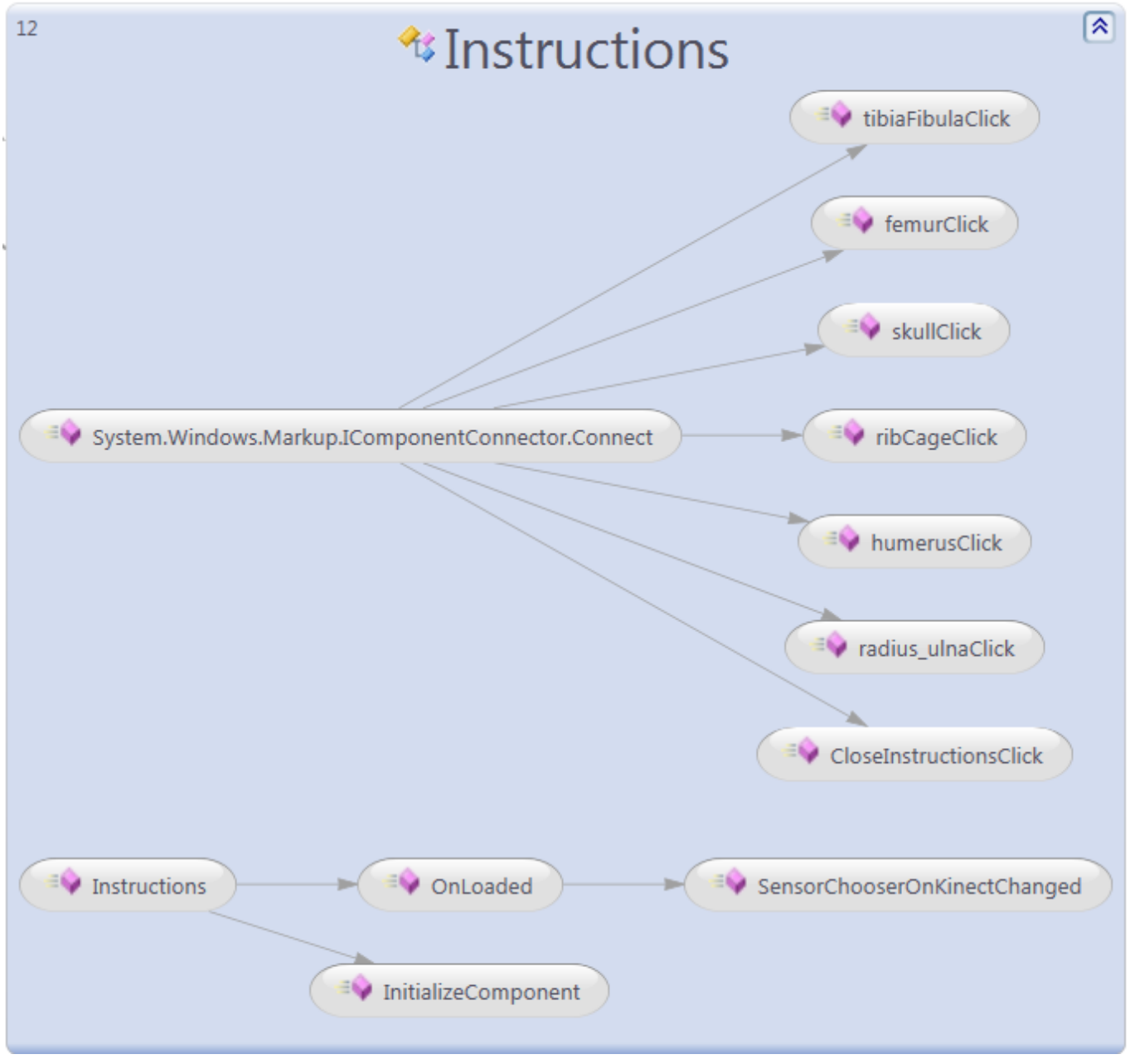
As I stated in 4.2 Non-functional requirements, the error with which the Kinect detects the user's palm should be noted and the UI must accommodate for that. As a result, all of the UI elements in BioKinect are large to ensure that the user can easily point at the desired UI component and are sufficiently spread out in order to prevent the user clicking the wrong button.

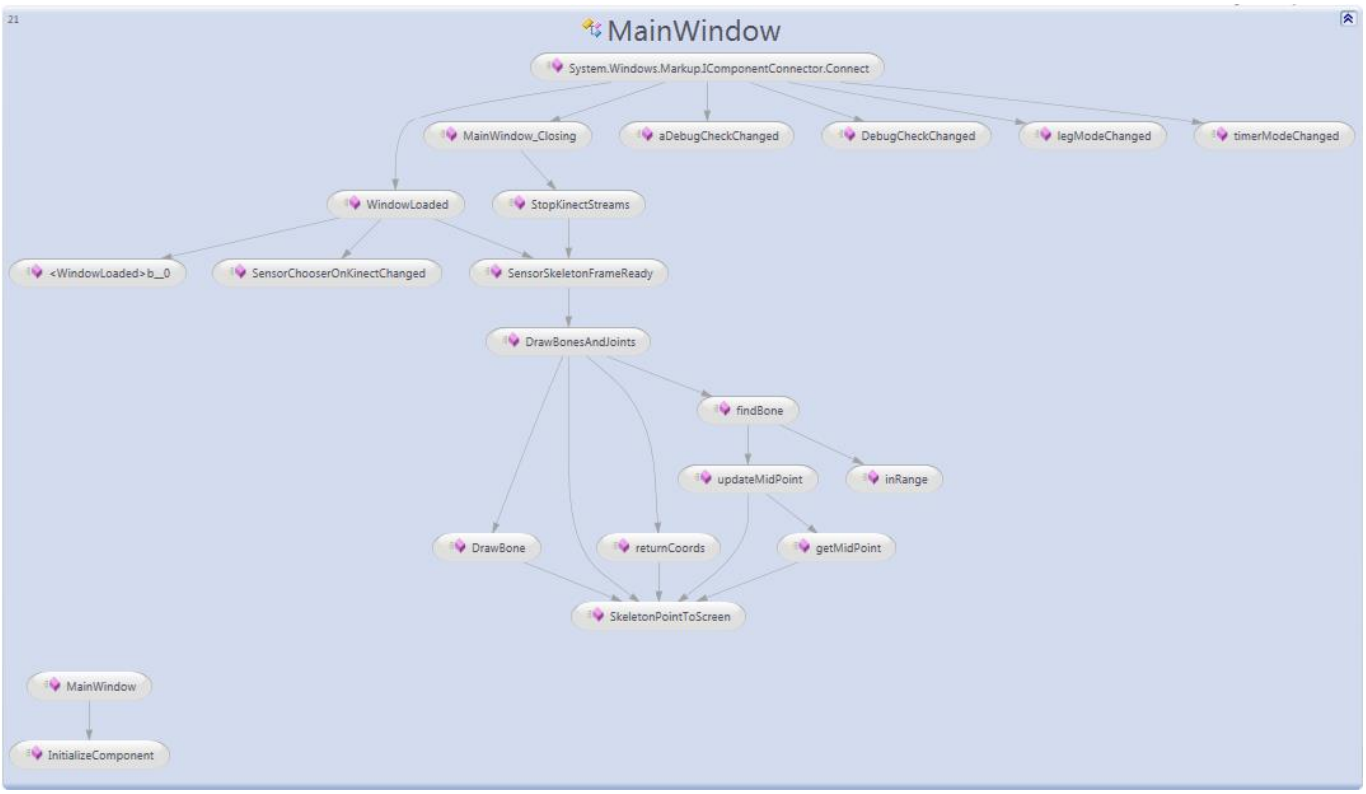
I decided to spread the system out into multiple screens (total of three). I think that this worked out well because each screen has a dedicated function and all of the UI components were able follow the principles outlined in the non-functional requirements. If it were a single screen system, the UI would be a cluttered mess and not only not look aesthetically presentable but would also hinder the user-friendliness aspect as well.

6.2 High-level Design

What follows are three UML diagrams which should give the reader a clear picture of the internals of my project. Each diagram represents a separate window. Within the separate diagrams are all the variables and methods within that window.







The main window is the game screen which executes various calculations on the user's tracked skeletal data. At start-up, the screen initialises all of the variables and objects that will be necessary for the game. These variables include all of the on-screen text labels such as the current score, the time remaining and the bone that the user is asked to pint at. Various parameters for drawing such as the thickness of brushes are also initialised at start-up. Objects include the `DispatcherTimer` used to monitor how much time the user has before the end of the game and the `KinectSensorChooser`. It is the most computationally expensive, but I managed to keep it running at smooth thirty frames per second as per Kinect's specification.

The instructions screen consists only of buttons and event handlers for them. Each event handler acts on a frame present on the screen by opening a URL in it. This is the simplest screen of all and contains only several local variables and event handlers.

The Window 1 is the initial app screen and acts as a main menu for the game. It contains a Kinect scroll viewer and several buttons. Each button is attached to an individual event handler. There is a global variable `difficultyValue` present on the main menu which represents the difficulty setting. This difficulty setting is passed to the game screen. There, it determines the value of `TimeSpan time` which is the countdown timer's set time. The harder the difficulty, the less time will be given to the player.

6.3 Low-level design

The low level details of every class are available in Appendix E. Enclosed in that appendix are the three architecture diagrams generated by Visual Studio which show the flow direction from one screen to another.

6.4 Design verification

The following walkthrough of the final version of BioKinect should convince the reader of the robustness of the application.

1) The BioKinect executable is launched. Window1 is called and begins its initialisation procedures. One important procedure is the 'sniffing' USB ports for a Kinect sensor. The application will work normally regardless if a Kinect sensor is detected or not. The stability is not broken even if the sensor is unplugged during operation. The user has the option of using the mouse to interact with the UI.
2) From Window1, the user has several choices – Window1 will remain open until the user decides to open a different screen or to close the app.
3) Pressing "Play BioKinect" or "Learn", will open up new screens and close Window1 (Main Menu). Lets assume the user presses the "Learn" button
4) Window1 lets go of the Kinect sensor and its data streams. The Learn screen opens and starts 'sniffing' for a sensor it can use for motion controls. The user has the option to use the mouse to interact with the UI. A back arrow is present which will return the user to the main menu. From my study of UI elements[22], presenting the user with an option to return to the previous screen is critical to the successful front-end design. It is just as important to present this back button in an intuitive way. I use an arrow that points inwards, indicating a backwards move.
5) On the Learn screen, the user has a choice of various buttons positioned on top of a drawn skeleton. These buttons open a URL that is displayed within the screen in a frame. This frame can only be operated using the mouse. I thought it would make sense because reading a lot of text while standing is not very comfortable when the option of sitting down is available.

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of B.Sc. Single/Double Honours in Computer Science/Computer Science and Software Engineering, is entirely my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____ Date: _____

Appendix A

[1]: https://msdn.microsoft.com/en-us/library/vstudio/w0x726c2(v=vs.110)

[2]: http://blogs.msdn.com/b/kinectforwindows/archive/2012/12/07/inside-the-newest-kinect-for-windows-sdk-infrared-control.aspx

[3]: https://www.microsoft.com/en-gb/download/details.aspx?id=40278

[4]: https://msdn.microsoft.com/en-us/library/kx37x362.aspx

[5]: https://msdn.microsoft.com/en-us/library/vstudio/hh875057(v=vs.110).aspx

[6]: http://marketplace.xbox.com/en-US/Product/Body-and-Brain-Connection/66acd000-77fe-1000-9115-d8024e4d0827

[7]: http://marketplace.xbox.com/en-US/Product/Kinect-Sesame-Street-TV/66acd000-77fe-1000-9115-d8024d5309da

[8]: http://download.microsoft.com/download/4/1/8/4182DF40-7EA3-4C13-91D0-E3B75D639590/CDE12BRIEFMicrosoft_Kinect.pdf

[9]: http://www.kinecteducation.com/blog/2013/02/12/kinect-educational-app-river-crossing-by-kinems/

[10]: http://www.kinems.com/

[11]: http://www.edutopia.org/blog/kinect-classroom-andrew-miller

[12]: https://msdn.microsoft.com/en-us/library/hh855347.aspx

[13]: http://www.adobe.com/uk/products/flashplayer.html

[14]: http://www.google.co.uk/intl/en_uk/earth/

[15]: http://glovepie.org/

[16]: https://msdn.microsoft.com/en-us/library/aa970268(v=vs.110).aspx

[17]: http://projects.ict.usc.edu/mxr/faast/

[18]: http://3.bp.blogspot.com/_RsyKBL2MDYk/Si7-tH5ioFI/AAAAAAAAAQFs/2lyAzhCzH4M/s1600-h/Bill+Gates+11.JPG

[19]: https://msdn.microsoft.com/en-gb/library/hh438998.aspx

[20]: https://msdn.microsoft.com/en-us/library/cc295302.aspx

[21]: http://www.wired.com/2011/06/mf_kinect/all/

[22]: Ellen Isaacs, Alan Walendowski (2001). Designing from Both Sides of the Screen: A Dialogue Between a Designer and an Engineer. Sams.

Appendix B

EA: I based compatibility with the Kinect on a few things. Certain subjects would make good games but I did not see how I could utilise the motion control aspect of them. Hence mathematics and computer science received a poor rating. Subjects like biology and languages received a medium rating because I had solid ideas about what sort of games these would be, but I thought that they would not use the full potential of the Kinect. For example I could the microphone array in the Kinect for voice recognition in the language games but implementing the motion controls would be more of a gimmick than a feature.

EB: I was able to directly move over code for detecting the Kinect sensor and the skeletal tracking code. This saved some time because I could focus on developing the actual game rather than worry about Kinect's boiler plate code. The ported code was mainly composed of the cosmetic XAML code used at the front-end.

Appendix C

WPF – Windows Presentation Forms

IR – Infra-Red

SDK – Source Development Kit

XNA – X Not Acronym

GBL – Game Based Learning

FAAST - Flexible Action and Articulated Skeleton Toolkit

XAML – Extensible Application Markup Language

UML – Unified Modelling Language

URL – Uniform Resource Locator

Appendix D

Report 1/Sprint 1 DATE

a)Established the possible subjects. Watching and reading Kinect tutorials. Made an XNA Visual Studio project with some sample Kinect code.

b)Need to setup a repo to store code. Need to have code that detects Kinect and display a message accordingly.

c)No tests performed. There is a wealth of information on the Kinect and its SDK on MSDN, this makes the research part easier but I still need to spend time learning how to do things the right way.

Report 2/Sprint 2

a)Made a simple program that displays the user's palm and draws a sprite on it. Made a tortoise SVN repo for the code. Made a batch script that automatically compresses my source code into a .rar archive and copies it into a backup folder in the cloud.

b) Need to include a frame that opens up the web app. Need to pipe Kinect data into that web app. Need to accommodate for Kinect being unplugged at run-time.

c)Tested the system for stability, unplugging the Kinect causes the application to stop responding.

Report 3/Sprint 3

a) Included a frame that opens the Google Earth app. The hand cursor now responds to grip state. The system will not halt when the sensor is unplugged during run-time. The system will register a sensor if it is plugged in during run-time.

b) The piping of Kinect data into the web app is proving challenging. I cannot find any standard library for this.

c) Tested the grip functionality, everything responds well. There is a problem with piping the data through to the web app. Similar applications use third party utilities such as GlovePie[15] combined with FAAST[17]. I do not want to use any third party utilities. XNA does not

provide me with the necessary utilities I need for my project. I think a project switch is necessary at this stage.

Report 4/Sprint 4

a) I have started a completely new WPF project titled BioKinect. This will use the skeletal tracking and WPF as the platform instead of XNA. Moved over some of the dynamic Kinect boiler-plate code. Made a new repository. Updated the backup script. Wrote the skeletal tracking function. The user's tracked skeleton and tracked joints are now rendered on screen. Added logic to the displayed skeleton which makes up the actual game component of the app. Add main menu and Instructions screens to make the app feel more complete and developed. Added some debug checkboxes which will show some values for demo/debug purposes.

b) Need to add Kinect controls to all of the screens. c) All of the functionality is stable. Getting the data from the skeletal stream was not simple and there are a few things that need to be performed before the collected data is rendered on screen. The Kinect tracked coordinates are different to those used by WPF. This was something I had to learn because the midpoint() function was not displaying the correct value.

Report 5/Sprint 5

a)Added cosmetic polish to all of the screens. The entire app follows a red and white theme that I think looks aesthetically pleasing. The entire app can be controlled via the Kinect. Added all of the game components (timer, scoreboard and difficulty system). Rewrote a lot of functions to work on Points instead of array values. Finished up the learning screen by adding Kinect button which open appropriate URLs.

b) Review the cosmetics and the code before the very release. Ensure stability of all screens and review spellings in UI components.

c) This is the final report before release. The crunch week helped push the project to more-or-less a completed state. One major bug was revealed by one of the tests which caused the entire app crash. I was able to fix it by adding proper shutting down procedures

to the Kinect sensor. I managed to tidy up a lot of the code by doing an intermediate code review which revealed some unnecessary variables and a few typos which resulted in an incorrect calculation in the midpoint() method. This sprint was the busiest by far.