Keystroke: Project Proposal
Group 3: Sensor

Tyler Alterio - Tyler_Alterio@student.uml.edu
Therese Kuczynski - Therese_Kuczynski@student.uml.edu
Steven Scheffelaar Klots - Steven_ScheffelaarKlots@student.uml.edu
Jesse Schaffer - HinesJesse_SchafferHines@student.uml.edu
Wen Shao - Wen_Shao@student.uml.edu
Clayton Smith - Clayton_smith@student.uml.edu

May 9, 2015

Contents

Introduction

1.1 Document Description

1.1.1 Introduction

Project Keystroke was built to allow users a way to be more physically active at their desks. This system works by mapping users movements to a predefined set of keystrokes. Once the keystrokes have been captured, the keystrokes will be broadcast to the application the user wished to type. The usage and movement data will also be sent to the cloud where words per minute, accuracy, and other usage statistics will be calculated. The user will also have the companion app installed on their phone which will allow them to view their data in the cloud.

Keystroke is designed for desk-bound workers who would like to be more physical active during the day while still being productive in the work place or at home. This product in geared towards people who spend long hours each day in front of their computers.. These dedicated workers would be given the chance to incorporate physical activity into their work routine.

For the purpose of this project we will creating a bare-bones user interface for Keystroke. Keystroke will be able to detect a users action and transmit those actions to user's application of choice. The user will then be able to type alphanumeric text into their selected application. We DO NOT plan to take advantage of the Kinect facial recognition or any of the other advanced features the Kinect has to offer. Keystroke is intended to be a proof of concept.

Keystroke is currently still in beta. The most current version of Keystroke is v0.5.6. Below is a list of all of the external resources that Keystroke uses along with a link to their respective websites.

- Unity: V5.1
- C #: V.2.0

- XBox One Kinect: V2.0
- XBox One Kinect SDK: V2.0

Below is a list of document that were generated during the course of the Keystroke development period.

- Group-3-Sensor_Project_Proposal.pdf
- Group-3-Sensor_Project_Assignment2.pdf
- Group-3-Sensor_Project_Assignment4.pdf
- Group-3-Sensor_Final-Report.pdf

With the given report, we as a group believe our plan to use a Kinect to track movements and turn those movements into keyboard strokes is feasible. There are enough uses in many different fields for this type of product, including uses in the workplace to encourage healthy habits and in-home physical therapy. Our deadline to have a working, bug free product is May 9th, 2015. Once released, we will continue working on software updates and customer concerns as needed.

1.2 System overview

In order to ensure that the Client and the Group are on the same page, the following walk-through has been prepared to illustrate the Group's understanding of the product desired by the Client. The walk-through is not necessarily a reflection of the exact interactions for the final product; rather it should serve as a rough overview of the functionality required by the final product as the Group currently understands it.

When users first load Keystroke, they will be greeted by a minimalistic UI that allows users to activate the product. Once Keystroke has been activated, users must then select the application they wish to type in. The Kinect will have been activated at this point and will begin capturing users movements once the user is a sufficient distance away from the sensor. It is at this point users will be able to begin typing using Keystroke.

While Keystroke is being used, a display of the live video feed will be shown back to the user with an added overlay of Keystroke's virtual keyboard. The overlay will allow users to see where characters are in relation to their body, making the product easier to use. It is now up to the user to decide what it is he or show would like to type. By simply holding ones hand over the virtual keyboard, the corresponding character will produce in the users application.

User will also have access to Keystroke's companion android app which will allow users to track their progress. Statistics like words per minute, acuracie, and general usage information will be accessible to users.

Design Considerations

2.1 Assumptions and Dependencies

Keystroke is built to run on Microsoft's 32 bit version Windows 8.1 operating system. We assume that the beta testers/users at this phase will posses the technical skills to install all Keystroke's dependencies onto their system. These dependancies include but are not limited to:

- Unity: V5.1
- C #: V.2.0
- XBox One Kinect: V2.0
- XBox One Kinect SDK: V2.0

As we are unaware of what issues may arise due to conflicts between Keystroke's dependencies and users personal settings or configurations, it is recommended for users to reinstall the 32 bit Windows 8.1 to ensure Keystroke works as intended.

2.2 General Constrains

During the latest development cycle, we discovered that Keystroke may posses a major bottle neck in its gesture recognition algorithm. This constraint was discovered when users tried to move excessively fast or in an unpredictable manner. When users moved or acted in such ways, the application would become slow and unresponsive. This issue is reproducible across a wide variety of low-end machines and could cause users frustration if the problem is not resolved.

A possible solution to this issue might be to limit the Kinect's sample rate in order to decrease the amount of data needing to be computed. This solution could potentially cause the users skeleton to appear jumpy or jittery. This solution, if implemented would only be a short term fix until we are able to find a more sustainable solution.

2.3 Goals and Guidelines

Keystroke aims to provide users with a simple and intuitive UI. The UI will connect users directly to our powerful gesture recognition algorithms that will capture users actions and transform their actions into recognizable gesture. These gestures will be the tools that users will use to interact with Keystroke's virtual keyboard.

Our primary goal for Keystroke is it must be able to allow users to select from at all of the lowercase letters in the English alphabet. If we are able to successfully implement the lowercase keyboard, we will then implement a full alphanumeric keyboard. This would provide users with the full range of functionality that a standard keyboard has to offer.

2.4 Development Methods

We are going to use the incremental development process as we feel it is best suited to the Keystroke project. The incremental development process has redundancy built in, allowing the development team to produce many versions versions of Keystroke during development. The client be about to provide valuable input on each release. The incremental development process yields the following:

- **Multiple versions**: During the course of the development cycle, will have produce several versions of the product to allow for incremental testing. This will aid in the elimination of bugs early on.
- **Feedback**: Client feedback invaluable during the early stages of development.

Architectural Strategies

3.1 Application Type

Keystroke is a standalone application that does not depend on any external application or server once it has been built. We decided to structure Keystroke this way as it would allow for the project to be easily loaded onto any given computer (that Keystroke is supported on). It would also be easier for users to use as starting the application would be as simple as running the single executable.

3.2 Future Plans

In the future, we plan on working closely with the cloud group to provide users with a more customizable feel to Keystroke. When we begin development with the cloud group, Keystroke will no long be a standalone application but would allow us to add some very neat features like customized keyboard layouts. Users would have the the ability to modify their personal settings in the cloud via mobile app or website and see those changes displayed in the Keystroke application. Settings like keyboard layout, keyboard color, and keyboard position would be just some of the settings users would be able to modify.

System Architecture

The system from an architectural view is divided into 2 main parts: The virtual keyboard and the key sender. The virtual keyboard is the 3D rendering of the keyboard that the user sees and interacts with. This part if the system sends the information that it receives from the user colliding with the collision blocks of the keyboard to the key sender. The key sender interprets this information and translates it into a keystroke and sends that keystroke to the system. The reason we decided to split the system into these distinct sections is for a few reasons. One of these reasons is that it compartmentalizes the code so that it is easier to test and translate to be used for possible expansion into other updates. The other reason that we did this is that the system functions separate easily into these 2 parts; either the code is there for the interpretation of the users movements or it is there to send information to the users system. This clear distinction between the two kinds of code we would be writing made it easy to determine that the two distinct parts should be made.

Policies and Tactics

For our system we decided that the best choice for an environment would be Unity for its deep integration and support for the Kinect. Most if not all applications that are written for the Kinect are done within the Unity environment. Unity is a tool used by developers for creating 3D games and environments for a multitude of purposes; ranging from visualizations for architectural design to developing 3D animations. In our case the Unity environment provides a perfect ecosystem for developing Kinect applications as most of the setup required to operate and interact with the Kinect is provided by a pre-built package that you can easily install into Unity. Unitys 3D GUI is also perfect for creating Kinect applications as it is incredibly easy to create and manipulate an object in 3D space.

Our plans for testing our software included having a team member dedicated to manual testing during each of our development cycles in between meetings. This entailed testing the application for errors or crashes and documenting the members findings. This also included writing short unit tests to determine whether our system was operating correctly underneath the GUI interface. After development of our system is completed, we intend to have more rigorous tests to be put into place that test any new code that comes into the Github repository and makes sure that the new code does not break anything that was previously working.

Our current plan for the interface of our system for the end user is to begin development of the interface after the back-end code of our system is completed and fully tested. This way we will know with absolute certainty whether any new bugs that arise during the rest of the development of the system stem from the user interface or the back end of the system. Our user interface will be simple and will various frames designed to give the users feedback on what they have done. This information will include a log of what keys the user has pressed and how quickly, a 3D readout of the keyboard and keys position along with the stream of the user so that the user can determine where they are in relation to the keys they are trying to press, and a tool that will allow the user to reposition the virtual keyboard to somewhere else within the Kinects field of view.

Detailed System Design

The virtual keyboard is one of the 2 modules of the system and is defined as the keyboard that is displayed to the user that they interact with to send keystrokes to their system. This module is responsible for receiving the skeleton rendering of the user from the Kinect and checking it against the collision model of the keyboard and sending that information to the key sender. This module is limited by the fact that the user must have a connected Microsoft Kinect peripheral visible by Keystroke and that the user all of their movements must be constrained to the view of the Kinect. This system is made up of the keyboard image in Unity that the user sees but has no function other than where the user should position their appendages to send the correct keys. The other part of this module is the collision blocks that determines what key the user attempted to press and sends this information to the key sender.

The second module of the system is the key sender which is described as the system that determines what key presses should be sent to the system and then sends those key presses. This system is responsible for all interaction with the users system and is dependent on receiving the keys that it should be sending from the virtual keyboard, or else this module will remain dormant. This module is built up from the receiver that receives the key presses from the keyboard and the sender that actually sends the key commands to the system.

System Installation and Execution

In order to interface with the Kinect, you'll need the Kinect SDK. Go on the Microsoft website and download the SDK. We used v2.0, however any newer version should be backwards-compatible. After that, go to https://www.unity3d.com and create and account. After your account is made download the Unity IDE, which will require you account. We're using Unity v5.1. Once you've installed everything, clone the project and open the "KinectSandbox" directory in unity. Once in unity, use the file explorer to open "Assets/kinectsandbox.unity" which will setup the project objects and scripts

Bibliography

Unity : https://unity3d.com/

C# : http://tinyurl.com/2fnedx

XBox One Kinect :

XBox One Kinect SDK :https://www.microsoft.com/en-us/kinectforwindows/develop/

Design patterns : https://sourcemaking.com

Business considerations

As University of Massachusetts Lowell students, the Group owns the copyright in the software that we create in this project. The Group agrees to transfer the copyright to the Client and to provide the Client with unrestricted license to use the system.

It is just possible that a project may develop concepts that could be patented. If such a situation arises, the Group collectively owns the rights to all patents associated with the System.

We understand that the use of open-source solutions IS a viable option and that there are not any serious licensing issues to this extent.