CSM 4 Kinect

June 17, 2014

Client: Dr. Cyndi Rader
Teaching Professor, Colorado School of Mines

Advisor: JK Slyby
Graduate Student, Colorado School of Mines

Team:
James Dang
Daniel Mawhirter
Spencer Kerr
Eric Koerlin

Table of Contents

I. Introduction

1. Client Description

This Microsoft Kinect project initially began under the title Recycler Robbie in Fall 2012 as part of the course Readings in Software Engineering, taught by Professor Cyndi Rader. The initial goal of this project was to create a game using the Microsoft Kinect. The project was continued in the 2013 field session with Dr. Rader and Assistant Professor Yong Bakos as clients. The previous field session team's goal was to create a standalone, open-source framework that built upon the original Recycler Robbie code base to allow new games and new sensors to be easily added. After this field session, the project was continued by acmX. Problems began when OpenNI, the open source Kinect library that was being used, stopped being supported when PrimeSense was bought by Apple in 2013. With the Kinect project having many lingering issues and still using the now unsupported OpenNI Kinect libraries, a decision was made to migrate the current system from Linux to Windows in order to use Microsoft's own Kinect drivers and Software Development Kit (SDK). This migration was only partially finished when the project was proposed to the new field session team, with the hopes of starting a game building competition in the fall that might interest new students to join the Computer Science department.

2. Product Vision

We were tasked with improving upon an interactive installation designed to get people interested in computer science. The basis of this installation is the Microsoft Kinect, and it will attract people through two different competitions, one for the best student designed game, and one for the best player high score on each scorable game. We have had two basic goals for this project. First, to build a bulletproof, reliable framework that is easy to use and easily maintainable. The second goal was to develop a system to enable the competition to take place with both an online component and a component on the Kinect system itself.

II. Requirements

1. Functional Requirements

- Usable, Robust Framework
    - Reset the demo machine remotely without needing a key
    - Make the system universally navigable (exit games, goto main menu, etc.)
- Update demo machine to run Windows and have stable network access
- Games
    - 3 existing games must still run
    - Improve the games to playability
    - Create 1 new game
        - Develop documentation on creating a new game
- Updatable Splash Screen
    - Greet people on approach
    - Grab attention of students walking by
- Competition
    - Design and implement a Voting Mechanism for Games
    - Keep track of player's scores
        - Scorable game vs. competition game
        - Normalized scores so no game awards more points
    - Leaderboard
        - Best Game (by votes)
        - Best Player (by scores)
    - Website access (Either CS Connect or on EECS Student Life Page)
        - Display high scores online
        - Voting for best game on webpage

2. Non-Functional Requirements

- Game interaction
    - Make Wave to sync faster, and more consistent
    - Scale hand tracking relative to body, instead of whole frame
- Sufficient testing to remove bugs
- Top Games
    - Ordered by high to low votes
    - Displays all games in database
- Top Scores
    - Displayed in high-score to low-score order
    - Display only top ten players

III. System Architecture

1. Overall Architecture

Our system consists of multiple different code sets and a database (See Figure 1). The main class is Interface SDK which handles interaction between modules and the Kinect system. The Kinect Hardware relays the information it receives from the camera to the Kinect Driver which relays it to the Kinect SDK. This information then gets sent to the Interface SDK which finally relays it for use in the modules. Each time the information gets relayed, it is processed to become more usable to the modules. The modules and the Interface SDK have access to a database to allow game scores to be saved and to display the high scores for a given game. Finally, the addition to the CS Connect website has similar access to the database for displaying high scores and for displaying and voting for individual games.

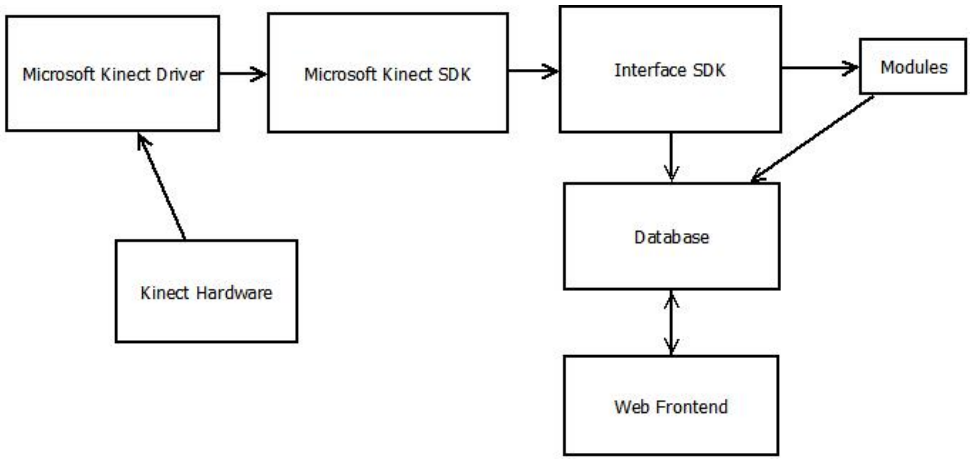

Figure 1: System Architecture

2. Interface SDK Architecture

Interface SDK is the main component in controlling most of functions of this system ((See Figure 2). Within it are the module manager, event manager, and hardware manager. The module manager determines whether or not modules have their required dependencies and is responsible for loading and running them. The events manager allows the communication of components in an event-oriented fashion. The hardware manager keeps track of drivers, their functionalities, and connecting the communication of the modules with their requested functionalities.

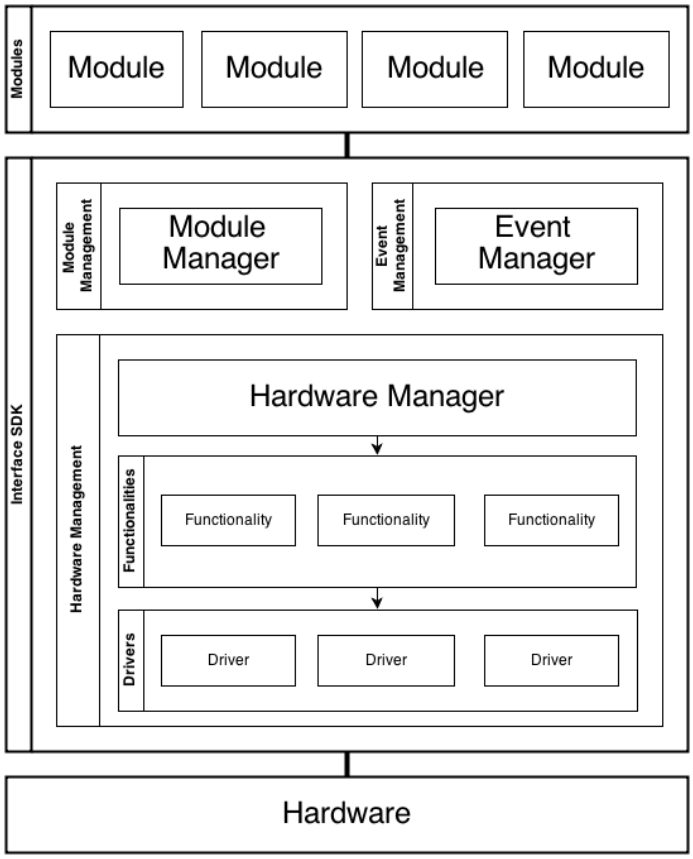

Figure 2: Interface SDK Architecture
(From 2013 Field Session)

IV. Technical Design

1. Migration and Interaction

The structure of this project dictated the design significantly. Previous versions of the Interface SDK used OpenNI and NITE to achieve hand tracking using the Kinect. The company PrimeSense was a major driving force behind Microsoft's production of the Kinect, and the main contributor to the OpenNI and NITE libraries. When PrimeSense was acquired by Apple in late 2013, the OpenNI project was discontinued, leaving the libraries on which the Interface SDK depended unsupported and ultimately unavailable.

The only other functional and long-term supportable driver and hand tracking library for the Microsoft Kinect is Microsoft's Kinect SDK. OpenNI was cross-platform, but Microsoft's Kinect SDK is Windows-only. This meant a migration of the demo machine from Ubuntu Linux to Microsoft Windows 7. Also inherent in OpenNI's design was cross-language compatibility which made interfacing the Java-based Interface SDK with the OpenNI and NITE frameworks relatively painless. However, the Microsoft SDK is designed to work with C, C++, C# and Visual Basic, not Java. Most of the interfacing framework was in place before this field session began, but it constitutes a significant portion of the technical complexity of the project.

JNA (Java Native Access) is an open-source library based on JNI (Java Native Interface) that allows Java to interface with native code in DLL files. The premise is that the programmer writes an interface in Java that matches the function prototypes for the native library. JNA is able to match those interfaces with the natively-implemented classes, convert many of the major native data types to Java types, and allow a Java application to use the interfaces as though they were fully-implemented classes. This is how the Interface SDK gets its data from the Kinect sensor.

The Microsoft SDK has a concept of a skeleton, a wireframe set of points corresponding to the trackable motion of a human standing in front of the sensor. Each skeleton has an ID associated with it, as defined by the Kinect sensor. Since each skeleton has a right and a left hand, we aggregate the portions of trackable skeletons that correspond to hands and give them an ID for use within the Interface SDK and its modules that corresponds to the order in which hands came into the Kinect's interaction frame (See Figure 3).

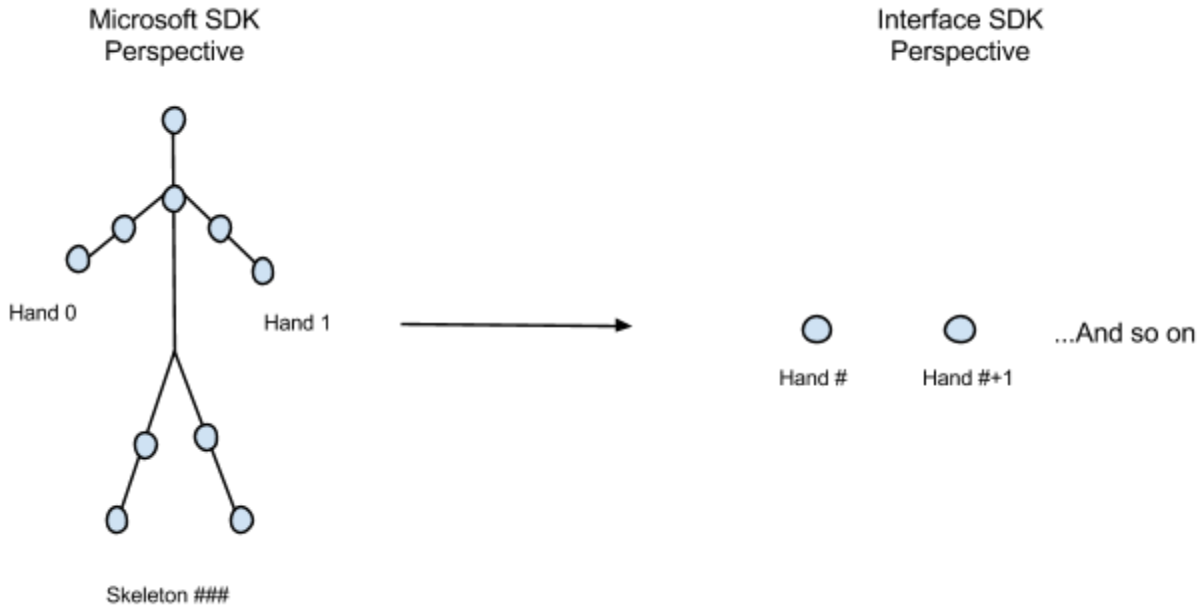

Figure 3: Visual representation of Hand aggregation

These hands are then continuously created, updated, and destroyed, and the Interface SDK fires events corresponding to these actions. Modules that use hand tracking can register for these events and will have methods of the hand receiver class called when these events are fired. The Interface SDK also tracks two basic gestures, a wave and a push , which it watches for in terms of the 3D coordinates of the hands being tracked.

A wave, consisting of a right-left-right-left sequence, is used as a way to 'synchronize' the hand. A hand created event will not be thrown for a tracked hand until that hand waves. Given the positioning of the demo machine in Brown building, it sees lots of hands going by. The wave approach ensures that the Kinect only responds to someone actively trying to use it.

A push consists of a hand moving closer to the sensor and screen. This gesture is very simple, and is currently used by Bug Squash to squash the bugs that move across the screen.

2. Scoring System

The Interface SDK is designed to be highly extensible and compatible with a multitude of different systems. The latest addition to its repertoire, coming out of this field session, is the possibility of saving scores and having a competition. The purpose of this competition is to spur interest in the Kinect and CS-Connect systems by having people or groups develop games. People who had played the games could vote on their favorites by logging in to CS-Connect, and the games where the player can score points allow that user to save their score to their CS-Connect account and compete for the high score in each game. The way this is currently implemented, a module that implements scoring uses the ScoreSaver class in the Interface SDK's stdlib to save a score to a local flat file (See Figure 4).

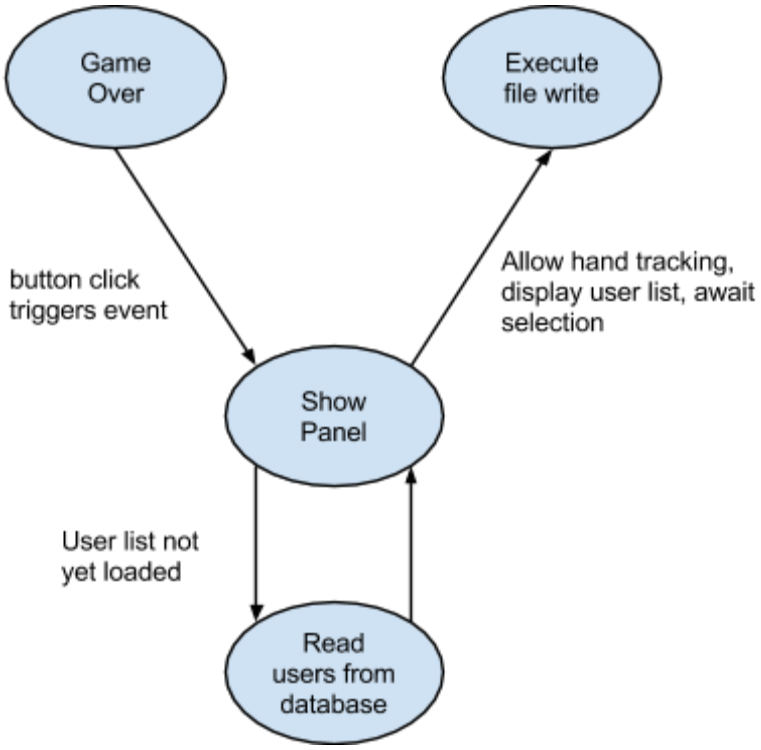

Figure 4: Logical flow of ScoreSaver's showPanel

In order to accomplish this and associate scores with names, Score Saver needs to allow the user to select his or her name from a list, since typing is not an option with the exhibit's setup. All of the currently running modules are implemented in Processing, which takes exclusive control over the JFrame it is operating within. This necessitates spawning an additional JFrame on top of the existing module frame if this implementation is to be abstracted away from module developers. In that JFrame, a scrollable window is drawn for users to select their name and submit. All of this drawing is done with AWT, but something as simple as a JComboBox could not work for something like the Kinect, where it does not emulate the mouse and the hand tracking had a certain amount of lag and jumpiness.

The way names are selected is through a similar model to click on rectangles implemented in other modules: virtual/hover clicking. Once the ‘cursor’ has been within the rectangle for a certain amount of time, the rectangle has been clicked, and the action corresponding to it will be performed.

Currently, this system pulls the list of names from the CS-Connect database of registered users, allows the user to select one, and then saves the score and name to a flat file in the same folder as the modules. In the future, this would save back into the CS-Connect database so that scores would be viewable on the website.

3. Exhibit Configuration

The Kinect demo machine in Brown has been configured with Windows 7 Professional Edition 64-bit. It is running Oracle JDK version 1.7 (Java 7) 64-bit. It has a fully-functional development environment installed with IntelliJ IDEA Community Edition and Maven. The Microsoft Kinect Driver, SDK, and Developer Toolkit are loaded as well

It is set up with a password-protected Admin user called TheKinectTeam and a no-password user called KinectUser. At boot, KinectUser is logged in automatically and a batch file controlled command window warns users to avoid standing in front of the Kinect while it starts so it can establish a baseline. The batch file then starts the Interface SDK application and the Home Screen module appears on the screen, ready for use.

V. Design and Implementation Decisions

1. We chose to use php for our initial website design as suggested by our client, but in order to easily incorporate it into the CS Connect website, we decided to write the final website in the same framework, Laravel.
2. We also decided to use a text document as our testing database, as it is easier to manipulate in the beginning, but for the final product, we used a database for storing all the information, this allowed us to get started easily, but create a robust solution in the end.
3. Use the existing CS-Connect website as a platform for our website, as opposed to creating a new website as users were pulled from the CS-Connect database
4. In our single process implementation, we decided to close all of our frames including the home screen, and then open a new home screen, instead of trying to keep the previous home screen running. This was in order to create a new driver for the home screen module so it will be able to recognize hand gestures
5. Our demo machine has a fully functional development environment, which allows us to make changes and test them on the demo machine without having to repackage everything every time we want to implement new changes
6. We chose to make most of our functionality changes in interface_sdk to minimize the changes that would have to be made to each module individually and maintain ease of ongoing development

VI. Results

We were able to improve the current framework to be stable on Windows machines. It runs all of the old games, which we improved to the point that they are all playable. Adding games to the system is still very easy, requiring simply that a compiled JAR file of the game be placed in interface SDK's "modules" folder. A new, more accessible power button was installed so the system can be reset more easily, without needing a key. The demo computer has been setup to automatically log in to a limited user account and run our game on startup, thereby allowing the system to be reset by someone unfamiliar with the project and administrator passwords. We created a new game running on the current framework which implements the new scoring system. We were hoping to make modifications to the splash screen to draw attention to the installation, but we were not able to accomplish that due to time constraints. We had also been hoping to implement a universal navigation scheme, but were unable able to do so due to time constraints. However,we have made navigation easier by changing the current games' exit behavior. Currently, games exit to the home screen either after displaying "game over" and score screens, or when the Kinect has lost its connected hands for a period of time.

For the leaderboard, we were able to add a new tab on the CS-Connect website and allow users to vote for each game. The list of games must currently be manually updated. The functionality also exists to display the users with the highest scores on a particular game, and works with a test database. Currently the Kinect demo machine does not publish the high scores to this page however. This page can pull from a database to get the scores and the votes and also checks to make sure that no one is able to vote for the same game multiple times. High scores are currently associated with a name selected from the remote CS-Connect database accessed from the Kinect machine and then saved to local flat files.

Future Work

This project could be improved upon in the future by allowing Interface SDK to use both mouse inputs and Kinect hand tracking to allow for easier development of games without needing a Kinect. Also adding a native code layer to interact with system events would speed up processing of the Kinect data and result in fewer tracking errors. Currently Java is not always fast enough to process this data, which causes tracking frames to timeout resulting in dropped frames. This would also simplify the higher level code, abstracting out the lower system level events, and make the system less time sensitive. Another useful improvement would be to modify 'HoverClick' to use reflection so that a method could be specified to be called when a hover click was registered.

VII. Appendices

1. Setting up a Development Environment

To setup a development environment, a Microsoft Kinect and a computer running Windows XP, 7, or 8 are required. Basic instructions are provided here, for more in depth instructions, as well as helpful tips and useful links, see the Interface SDK wiki

i. You will need Git and JDK 7 to get started

ii. Drivers and Developer Toolkit
Download the Kinect SDK, and Developer Toolkit.These downloads include the necessary drivers, install the SDK first. Interface SDK uses version 1.8 of the Kinect SDK as of 0.4.3.

iii. Clone the Repositories
Clone the Interface SDK Github repository, as well as home_screen_module and any other modules you want access to. These will need to be imported using IntelliJ.

iv. Setting up your Coding Platform

Installing Intellij

1. Download IntelliJ
2. Select your OS at the top and download the community version.
3. Install it.

Creating Project

You can create an Intellij project using the built-in Maven plugin. To create project:

1. Open up Intellij.
2. You should be presented with a screen that says "Import Project". If you are not (you have used Intellij before), go to "File"->"Import Project".
3. Select the path of the Maven "pom.xml" file.
4. Hit ok.

Installing Maven

This SDK uses Maven 3. It is recommended that you install the most recent stable release, such as Maven 3.0.5. Unless you are planning on modifying the functions of Maven itself, you should download the binary zip file and extract it into a separate folder. The recommended location to place Maven is C:\Program Files\Apache Software Foundation\apache-maven-3.0.5.

Once the bin is extracted, ensure that Maven is included in your system's PATH variable by navigating to (in Windows 7) Control Panel -> System -> Advanced system settings -> Environment Variables. Edit the PATH user variable to include the path to your Maven folder, i.e. C:\Program Files\Apache Software Foundation\apache-maven-3.0.5. Typing

mvn -version in a command line should print your version of Maven and verify that Maven is correctly configured. If this command does not work, repeat the steps in this paragraph.

Configuring Maven

Add an environment variable M2_HOME with the value being the path to your Maven install. If you are unsure how to do this, you can configure the path directly within Intellij on a per-module basis:

1. Go to "File"->"Settings" and on the side you will see Maven listed. Click on Maven.
2. You will see a field named "Maven home directory". Check the "Override" box.
3. Enter in the path to your Maven install.

Using Maven

To build the project, you must go through the Maven plugin.

1. "Maven Projects" should be listed on the right side of the screen. Click it.
2. A pop-out windows should appear with different commands. Expand "Lifecycle".
3. Select package.
4. Click green arrow at top. This will compile your module

Running compile for the first time will download all the needed libraries.

v. Create a Module

We recommend looking at a module like BrickBreaker and changing the name to suit your needs. Working from there is likely to give the best starting point. Additional instructions can be found on the Interface SDK wiki, and in the next section of this document.

vi. Run Your Module

Create a new directory structured as the one below. The files will be added in the following steps.

1. Ensure the most recent version of Interface SDK has been pulled from Github
2. Create a directory named modules/, and place your module and any other modules you want to run in this folder. In the above example, the Home Screen Module has also been added.
3. Create a run configuration in IntelliJ by going to “Run”->”Edit Configurations…” and clicking on the “+” at the top left to add a new run configuration.
    a. Type a name in the “Name:” field.
    b. Check the box that says “Single Instance Only”
    c. Select ModuleManagerRunner for the Main Class
    d. In “Program arguments:” type:
        “--manifest src\main\resources\example_module_manager_manifest.xml”
4. Run your configuration by going to “Run”->”Run ‘<name>’

2. Information on Creating a New Game

To create a new game, a working development environment is necessary. It may be possible to create a game without a Kinect and convert it to work with the Kinect, but that has not been tried, and those instructions will not be provided here. A good way to start creating a new game is using the source code from one of the existing games to get all of the necessary functions and calls, and then modifying the setup, update and draw functions to have the desired game behavior. The games for this installation are made using Processing, more information about Processing and its available functions can be found on Processing's website, listed in the Relevant Links section. The setup function is called by default whenever a game is loaded, and then the draw function is called a predetermined number of times every second. Typically the draw function will draw all necessary objects on the screen, and will also call update. Update is responsible for making any calculations and moving any objects around the screen as needed, update also typically updates hand positions. Other important functions to include and call are registerTracking, which starts the Kinect driver and allows handX and handY positions to be determined. It can be useful to make functions that will draw things like "game over" screens as well that can be called by checking a simple boolean flag. Due to the fact that Processing has very specific timing constraints, timers should not be used, instead time can be tracked by monitoring changes in Processing's millis() function.

Drawing in Processing works like drawing in Microsoft Paint, once something is drawn on the screen, it stays there until something is drawn over it. To make objects move across the screen, set the background and then draw the objects. By setting the background first, the screen is essentially cleared, and objects will not draw tails.

To add a game icon, place the image in src/main/resources/images in the module directory. The icon name must also be set in the manifest.xml, which should be located in the resources file. The icon should be named something other than "icon", as this can cause conflicts with other game's icons.

If Maven is being used to compile the game module, the pom.xml file must be edited to include the Interface SDK as a dependency, it is important that the correct version number of the Interface SDK is listed in the version section, multiple versions can be listed as long as they are separated by commas. The appropriate Interface SDK version number can be found in Interface SDK's pom.xml file, listed at the top of the file under version. As of this writing, the current version of Interface SDK is 0.4.3. To compile the game module in IntelliJ, look for the project lifecycle under Maven Projects, select "package" and click the "Run Maven Build" button. this will create a JAR file of the game module.

Once a JAR file of the game module has been created, make a copy of the JAR file in Interface SDK's "modules" folder. This will direct interface SDK to load the game module on the Home Screen.

3. Relevant Links

1. Interface SDK GitHub
   https://github.com/ColoradoSchoolOfMines/interface_sdk
2. Interface SDK Wiki
   https://github.com/ColoradoSchoolOfMines/interface_sdk/wiki
3. CS-Connect website
   http://connect.mines.edu/
4. Git Download
   https://help.github.com/articles/set-up-git
5. JDK 7 Information
   https://github.com/ColoradoSchoolOfMines/interface_sdk/wiki/Setup:-Installing-Java
6. Microsoft Kinect SDK and Developer Toolkit
   http://www.microsoft.com/en-us/kinectforwindowsdev/Downloads.aspx
7. Processing
   www.processing.org
8. Pong
   https://github.com/ColoradoSchoolOfMines/Pong
9. Home Screen
   https://github.com/ColoradoSchoolOfMines/home_screen_module
10. Brick Breaker
    https://github.com/ColoradoSchoolOfMines/brickBreaker
11. Bug Squash
    https://github.com/ColoradoSchoolOfMines/bugSquash
12. Fall
    https://github.com/EricKoerlin/Fall