APPVIS: ENABLING DATA-RICH APPS IN APP INVENTOR

BY

FARZEEN HARUNANI
B.S. MARQUETTE UNIVERSITY (2015)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF MASSACHUSETTS LOWELL

Signature of
Author:______________________________**Date:** ____________

Signature of Thesis:
Supervisor:______________________________
Name Typed: Dr. Fred Martin

Signatures of Other Thesis Committee Members:

Committee Member Signature:______________________________
Name Typed: Dr. Haim Levkowitz

Committee Member Signature:______________________________
Name Typed: Dr. Sayamindu Dasgupta

APPVIS: ENABLING DATA-RICH APPS IN APP INVENTOR

BY

FARZEEN HARUNANI

ABSTRACT OF A THESIS SUBMITTED TO THE FACULTY OF THE
DEPARTMENT OF COMPUTER SCIENCE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE
UNIVERSITY OF MASSACHUSETTS LOWELL
2016

Thesis Supervisor: Fred Martin, Ph.D.
Professor, Department of Computer Science

Abstract

MIT App Inventor has enabled middle school students to learn computing while creating their own apps—including apps that serve community needs. However, few resources exist for building apps that gather and share data. There is a need for new tools and an instructional materials for students to build data-enabled, community-focused apps. We developed an extension for App Inventor, called AppVis, which allows app-makers to publish and retrieve data from iSENSE, our existing web-based collaborative data visualization platform. We used AppVis and supporting instructional materials in two one-week summer camps attended by a total of 33 middle school students. Based on student interview data and analysis of their final apps, our approach was broadly accessible to a diverse population of students. Students were motivated to build apps that could be used by their own communities. This thesis presents the design of AppVis and results from students' work in summer camps.

Acknowledgments

I would like to thank:

- My thesis advisor, Fred Martin, who supported me in countless ways throughout this research.
- My dear friend Ashley Hale, who supported with data organization and graphs, and remembered to feed me when I forgot.
- Lijun Ni, with interview data gathering and analysis.
- Mark Sherman, with software tool integration.
- The iSENSE developers at University of Massachusetts Lowell and the rest of the Engaging Computing Group.
- The MIT App Inventor development team.
- The Medford and Everett school districts.
- The CS Pathways team, including Molly Laden, Diane Schilder, and Akira Kamiya.
- The CS Pathways summer teachers: Damian Demarco, Lori Blank, Amy Lieberman, and Jessica Hammerly.

- My thesis readers Haim Levkowitz and Sayamindu Dasgupta.
- My family, especially my parents. I would not be where I am today without the support of those around me.
- My friends, whose names are too numerous to insert here, for providing distractions when I needed them most. Every tank needs a healer.
- This material is based upon work supported by the National Science Foundation under Grants DRL-1433592 and IIS-1123972. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Tables

List of Figures

Chapter 1: Introduction

Since the work of the late Seymour Papert, more and more researchers have been focusing on ways to democratize computing. Papert introduced the idea of "constructionism", which is built upon Jean Piaget's principles of constructivism. He also pioneered the LOGO programming language, the first in a long list of computational design environments for children (MIT Media Lab, 2016).

Informal education is a recognized approach for strengthening students' computational thinking abilities, especially in the form of after-school activities and summer camps. According to Janet Batsleer, the author of Informal Learning in Youth Work, "in informal education...[learning] occurs because the learning is of immediate significance to those involved, rather from a pre-established curriculum" (Batsleer, 2008). Batsleer also explains that informal learning can be a road to equity and diversity. Informal education allows a pedagogy of collegiality to occur between students and "co-conspirators," rather than the traditional pedagogy between student and teacher (Ito, et al., 2009). Furthermore, informal science education in particular can aid in the student's "empowerment and the development of skills for participating in civic activities" (McCallie, Bell, & Lohwater, 2009).

Using social good motivation to teach computing has worked well for a number of educators, as seen by the success of the "Mobile CSP" AP CS Principles course (Morelli, Wolber, Rosato, Uche, & Lake, 2014).

MIT App Inventor is a web-based platform for development of Android applications (Wolber, Abelson, & Friedman, 2014). It uses a blocks-based programming language that provides a low barrier to entry, and it has become a widely-used platform for introductory computer science. It is also often used in informal education (A-Ghamdi, Al-Rajhi, Al-Onaizy, & Al-Khalifa, 2016; Grover & Pea, 2013; Morelli, et al., 2011). App Inventor works well in this context, because "'what is being built' is the driving force, not the abstract goal of learning to code... many more [students] get excited about solving real-world problems and improving their lives and those of their friends" (Wolber, Abelson, & Friedman, 2014). Students enjoy working with hands-on projects that give them "an opportunity to be creative within the structured program curriculum" (Martin, et al., 2011). App Inventor very much allows this, and lends itself to the "use-modify-create" model for engaging youth in computational thinking (Lee, et al., 2011).

Nonetheless, most App Inventor users "tend not to create apps that are functionally different from completed tutorials" (Xie, Shabir, & Abelson, 2015). Similarly, Blikstein warns (Blikstein, 2013): "educators should shy away from quick demonstration projects and push students towards more complex endeavors," else the students will continue to build variations of demo projects and not progress forward. This idea, combined with Barba's advice (Barba & Chancellor, 2015): "tutorials are best treated as a kind of scaffolding rather than a primary source and, when used, should not be scaffolded any further," provide a solid groundwork for where to start in designing materials for introducing students to App Inventor concepts.

However, many educational programming environments, including App Inventor, do not have much support in the way of data science. The iSENSE Project is a collaborative effort of the University of Massachusetts Lowell and Machine Science Inc. (Michalka, Dalphond, & Martin, 2016). It provides an easy, user-friendly data collaboration and visualization tool that is meant for use in the middle and high school classrooms. Tools like iSENSE are crucial in an educational environment in which "meaningful student encounters with data analysis are rare" (Finzer, 2013).

In 2014, researchers from University of Massachusetts Lowell and the Tri-City Technology Education Collaborative, Inc. partnered with two urban school districts in the CS Pathways project, a three-year initiative to design a middle school curriculum that teaches computational thinking with App Inventor. As part of the project, the summer of 2015 was used to run an App Inventor summer camp in these districts. The goal was to have students build socially relevant apps to help local communities (Ni et al., 2016). At the end, over half of the students had built games. Out of 38 total apps, only one was a utility app (meant to be used as a tool by others), and only about 60% of the students felt that they had built apps that they would like to share with the community. Students commented that they did not feel that they had built real apps.

This work began with the challenge to design a subsequent iteration of the summer camp experience for the CS Pathways project. We took a different approach, with new technology and instruction. We introduced the concept of data-rich apps for social good by designing a curriculum around AppVis, an extension of MIT App Inventor that integrates with the iSENSE Project. In order to encourage the students to build social good apps, we had volunteers from local nonprofits discuss community problems that could be solved by technology. As a result, students built more utility apps, and nearly all students were proud to share their apps with their communities.

Chapter 2: Background

In this chapter, we explore prior work in data collaboration and visualization tools for beginners. Though AppVis provides user-friendly data-rich tools for introductory programming in the Android environment, it is not the first tool to try to fill this gap.

2.1 Scratch

Scratch has the most users of any of the other platforms mentioned. It is a browser-based block programming environment that allows easy sharing of projects among users.

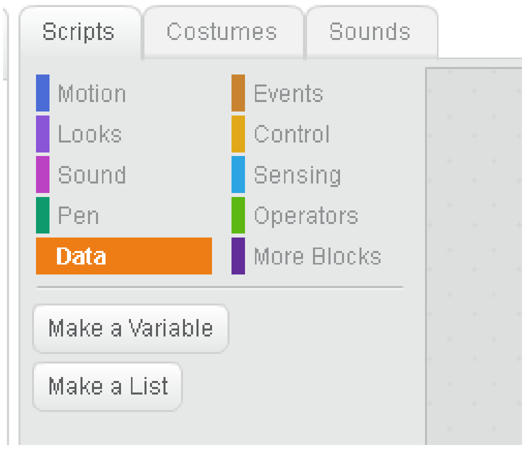


Figure 1: Scratch - Available Data Blocks for Standard Users

While most users are unaware of the data-centric capabilities of Scratch, Scratch actually offers metadata for advanced users to use in their own apps. A user is made eligible for advanced Scratch blocks based on how many projects they have and how active they are in the Scratch community, and so they are not available to the beginner Scratch user ("New Scratchers"). These advanced blocks can be used to build apps that utilize user information about their own users or followers, or how many people speak a certain language. While the visualization tools are not explicitly present, Scratch offers the capability to build complex projects that interface with data about Scratch itself.

Scratch also offers cloud variables to select eligible advanced users. These variables are persistent and stored on the server. For example, this makes it possible to maintain a global high score.

2.1.1 NetScratch

NetScratch is an extension of Scratch that allows shared variables (Stern, 2007). In this environment, students can work with each other's data, manipulate data in real-time, and interact with other users in an entirely new way. However, this also brings up the issue of data ownership and permissions. One user can easily delete data entirely by accident, and lose data that multiple users were working with.

2.2 App Inventor

Before the AppVis component was introduced, several components existed that allowed apps to record data.

2.2.1. TinyWebDB in App Inventor

TinyWebDB provides the ability to store and retrieve data between App Inventor apps. This data can only be accessed by App Inventor apps and there is no visualization capacity.

There are two ways to configure this component: First, to use the default “server”, which has limited storage capacity and is not guaranteed to persist over time; i.e., it could be rewritten by other apps. All apps have the same access to this data. The second way is for a user to create their own instance of a TinyWebDB server, which allows persistent and private data storage, but requires the Google App Engine and user configuration that is beyond the scope of a middle school classroom.

Sample screenshots of TinyWebDB have been provided here. TinyWebDB acts much like the TinyDB in App Inventor, and both utilize a KVP (key-value pair) model of data. The difficulty of TinyWebDB lies in configuration outside of the App Inventor environment, which is not shown here.

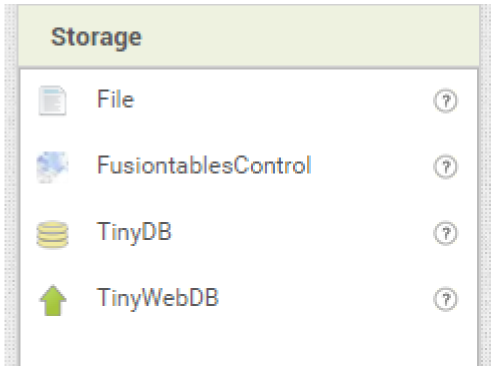
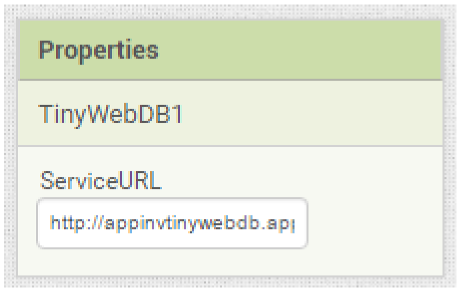


Figure 2: TinyWebDB in App Inventor


Properties

TinyWebDB1

ServiceURL

http://appinvtinywebdb.ap

Figure 3: TinyWebDB Configuration

The figures above show where the component lives in the Components palette of App Inventor, and the Properties of the component. Below, all available blocks for TinyWebDB are shown.

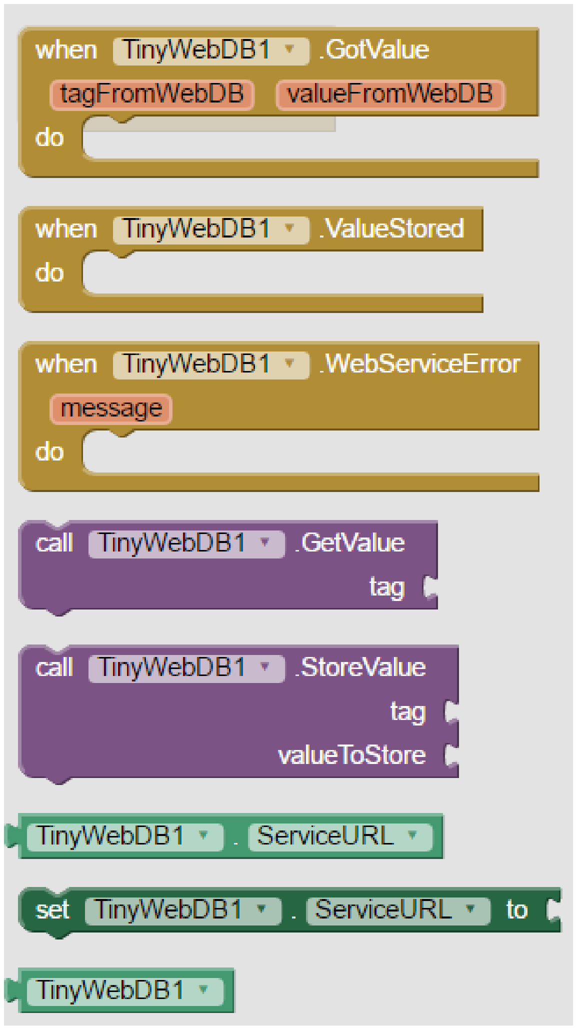

when TinyWebDB1 .GotValue
tagFromWebDB valueFromWebDB
do
when TinyWebDB1 .ValueStored
do
when TinyWebDB1 .WebServiceError
message
do
call TinyWebDB1 .GetValue
tag
call TinyWebDB1 .StoreValue
tag
valueToStore
TinyWebDB1 . ServiceURL
set TinyWebDB1 . ServiceURL to
TinyWebDB1

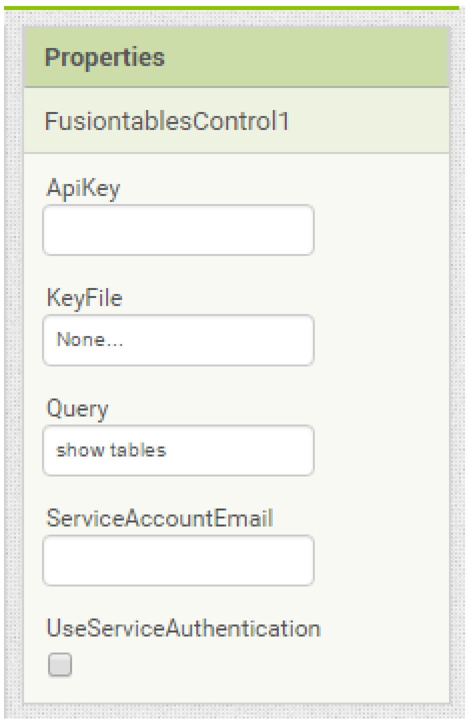
Figure 4: TinyWebDB - Available Blocks

2.2.2 FusiontablesControl in App Inventor

FusionTablesControl is an interface in App Inventor to Google's Fusion Tables. Fusion Tables is a sophisticated, powerful Google web application that allows real-time data collaboration, gathering, sharing, and visualization. Before AppVis, this was the only way to visualize data in App Inventor. However, configuration and usage of Fusion Tables is not appropriate for novices and a challenge even for advanced App Inventor programmers. When introduced to college students during an Android

Application Development course, 43% of the students agreed or strongly agreed that a databases course should be a required prerequisite (Soares & Martin, 2014). As a databases course is not feasible in a summer camp for middle schoolers, Fusion Tables are not a viable option for introducing students to working with data-rich apps.

The figures below show configuration and the available blocks for working with Fusion Tables in App Inventor.


Properties

FusiontablesControl1

ApiKey:

KeyFile: None...

Query: show tables

ServiceAccountEmail:

UseServiceAuthentication: ☐

Figure 5: FusiontablesControl Configuration

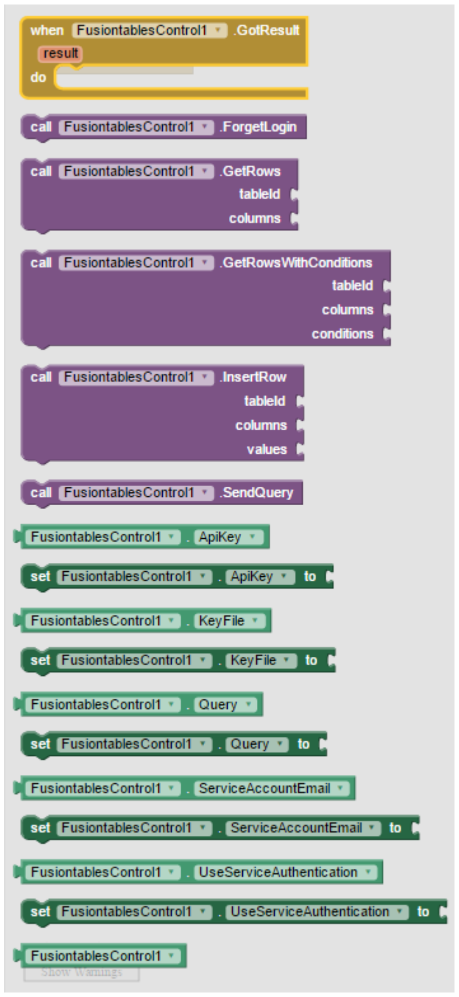
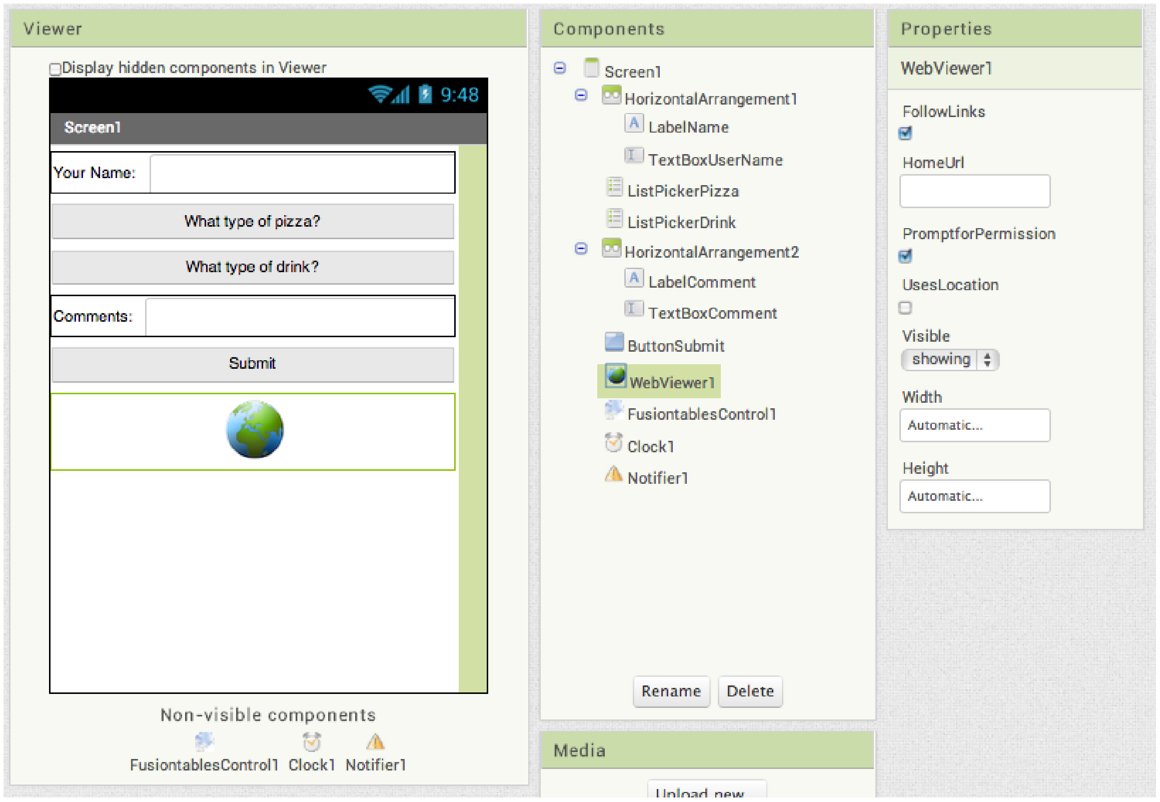

```
when FusiontablesControl1 .GotResult
  result
do

call FusiontablesControl1 .ForgetLogin

call FusiontablesControl1 .GetRows
                          tableId
                          columns

call FusiontablesControl1 .GetRowsWithConditions
                          tableId
                          columns
                          conditions

call FusiontablesControl1 .InsertRow
                          tableId
                          columns
                          values

call FusiontablesControl1 .SendQuery

FusiontablesControl1 . ApiKey
set FusiontablesControl1 . ApiKey to
FusiontablesControl1 . KeyFile
set FusiontablesControl1 . KeyFile to
FusiontablesControl1 . Query
set FusiontablesControl1 . Query to
FusiontablesControl1 . ServiceAccountEmail
set FusiontablesControl1 . ServiceAccountEmail to
FusiontablesControl1 . UseServiceAuthentication
set FusiontablesControl1 . UseServiceAuthentication to
FusiontablesControl1
```

Figure 6: FusiontablesControl - Available Blocks

As an example usage case, user interface and selected parts of the code for the Pizza Party tutorial[1] is displayed here.

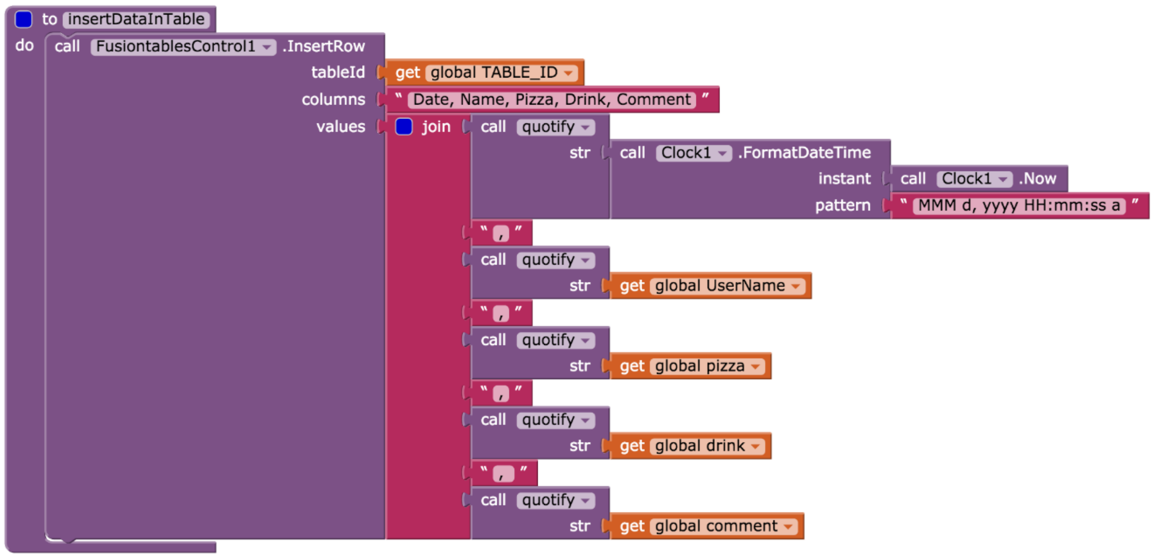


FollowLinks ✓
HomeUrl
PromptforPermission ✓
UsesLocation ☐
Visible: showing
Width: Automatic...
Height: Automatic...

Figure 7: User Interface for Pizza Party Tutorial

The above figure shows the Pizza Party tutorial's user interface layout, which is provided as context for the following sample code blocks. Not all of the code has been presented here; instead, only parts of the Fusiontablescontrol-specific code have been shown. Correct usage of this component requires SQL to manipulate the data, unlike the key-value pairs of TinyWebDB.

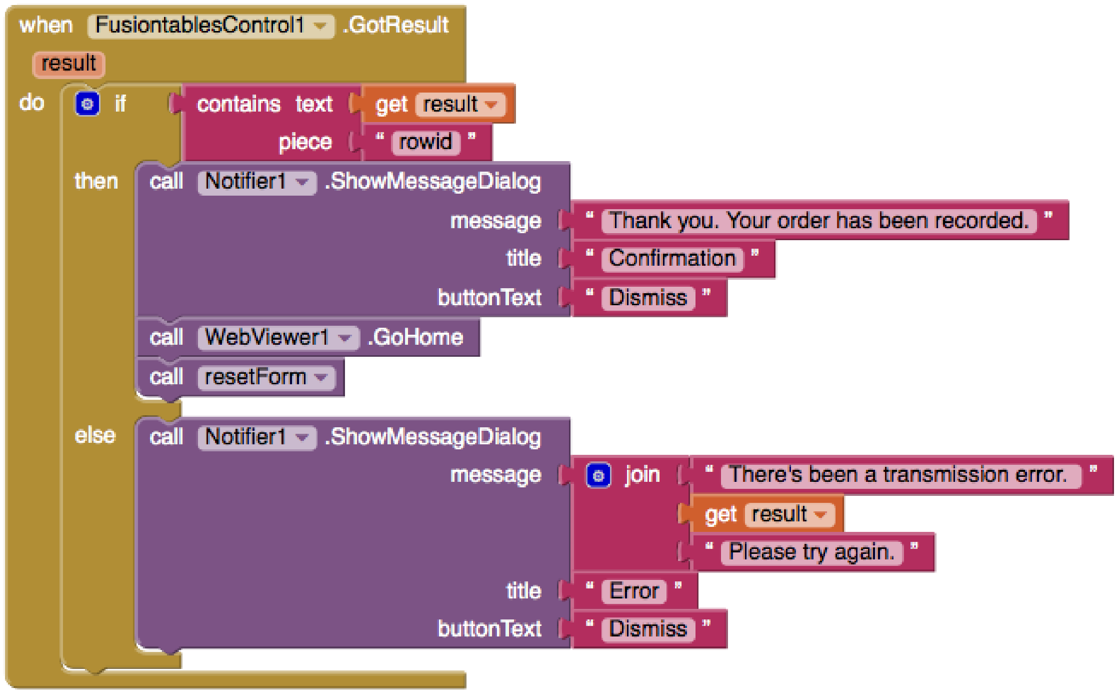
[1] Available at http://appinventor.mit.edu/explore/ai2/pizzaparty.html.



```
to insertDataInTable
do  call FusiontablesControl1 .InsertRow
                        tableId   get global TABLE_ID
                        columns   " Date, Name, Pizza, Drink, Comment "
                        values    join  call quotify
                                             str  call Clock1 .FormatDateTime
                                                          instant  call Clock1 .Now
                                                          pattern  " MMM d, yyyy HH:mm:ss a "
                                        " , "
                                        call quotify
                                             str  get global UserName
                                        " , "
                                        call quotify
                                             str  get global pizza
                                        " , "
                                        call quotify
                                             str  get global drink
                                        " , "
                                        call quotify
                                             str  get global comment
```

Figure 8: Fusiontablescontrol Insert Data Example

Above is the InsertDataInTable function that uploads data into the Fusion Table.



```
when FusiontablesControl1 .GotResult
  result
do  if   contains  text   get result
                   piece  " rowid "
    then  call Notifier1 .ShowMessageDialog
                              message     " Thank you. Your order has been recorded. "
                              title       " Confirmation "
                              buttonText  " Dismiss "
          call WebViewer1 .GoHome
          call resetForm
    else  call Notifier1 .ShowMessageDialog
                              message     join  " There's been a transmission error. "
                                                get result
                                                " Please try again. "
                              title       " Error "
                              buttonText  " Dismiss "
```

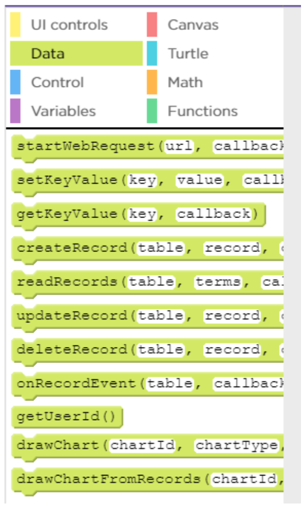
Figure 9: Example Usage of Fusiontablescontrol GotResult Callback

Above is the GotResult callback function, which is triggered on a positive or negative upload result.

2.3 Code.org's AppLab

The AppLab is a browser-based environment that allows students to create "apps" that run in the browser with either blocks or JavaScript. It is similar to Scratch in that it has a browser-based, visual programming environment and students can see their code run right within the browser as well. However, AppLab features different functionality; specifically, JavaScript integration. It also enables data-rich apps. The data is based on key-value pairs (similar to TinyDB in App Inventor). Moreover, it includes methods that allow data visualization. The apps created by AppLab are shareable among other AppLab users, but they are restricted to the browser.

Screenshots from a sample cat/dog voting app are shown below to demonstrate usage of AppLab's data storage and visualization capabilities. Because the data tables are stored in the apps themselves, the data is persistent (cloud storage) even if the app is closed or changed. While the AppLab provides a very similar strategy to what AppVis provides, there is no standalone visualization feature—that is, the visualizations can only exist within the apps.



```
startWebRequest(url, callbac
setKeyValue(key, value, call
getKeyValue(key, callback)
createRecord(table, record,
readRecords(table, terms, ca
updateRecord(table, record,
deleteRecord(table, record,
onRecordEvent(table, callbac
getUserId()
drawChart(chartId, chartType,
drawChartFromRecords(chartId,
```

Figure 10: AppLab – Available Data Blocks

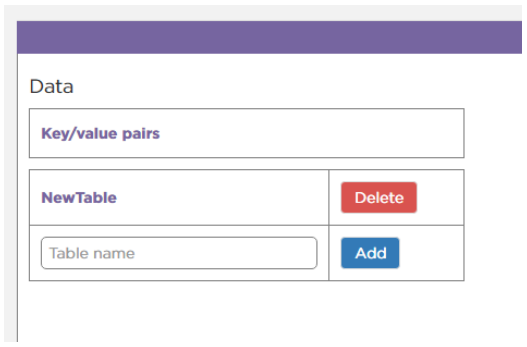
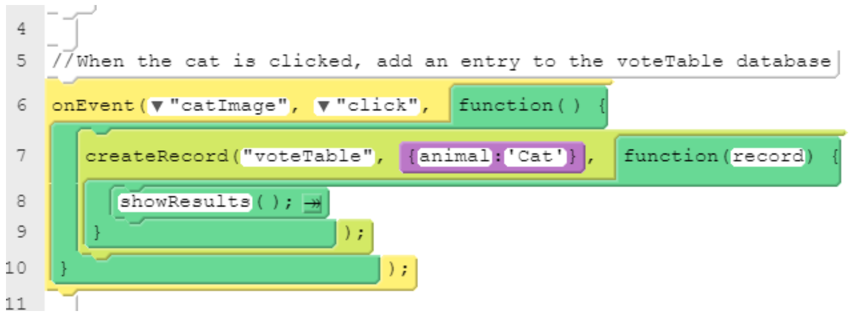


Figure 11: AppLab – Data Browser

The above and below figures show some of the background workings of a data-rich app in AppLab. While AppLab is a powerful tool, it is much more complicated than AppVis.



```
//When the cat is clicked, add an entry to the voteTable database
onEvent("catImage", "click", function() {
  createRecord("voteTable", {animal:'Cat'}, function(record) {
    showResults();
  });
});
```

Figure 12: Adding Data in AppLab

Below are screenshots of the user interface of the example pet voting app in AppLab. Conceptually, it is actually very similar to one of the tutorials designed for the AppVis camp.

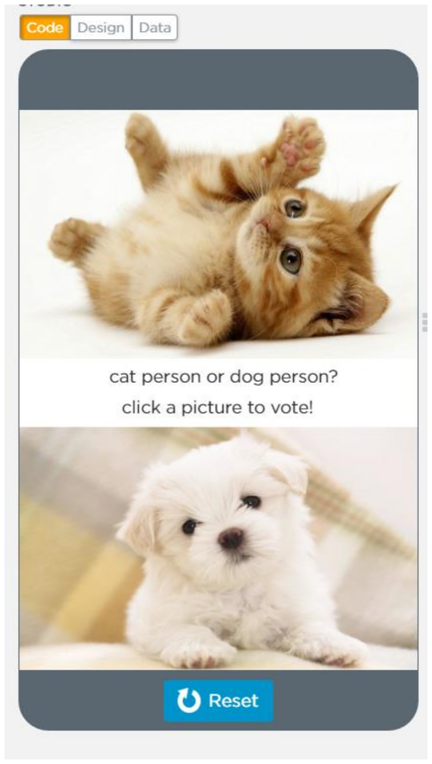


Figure 13: Example AppLab Layout

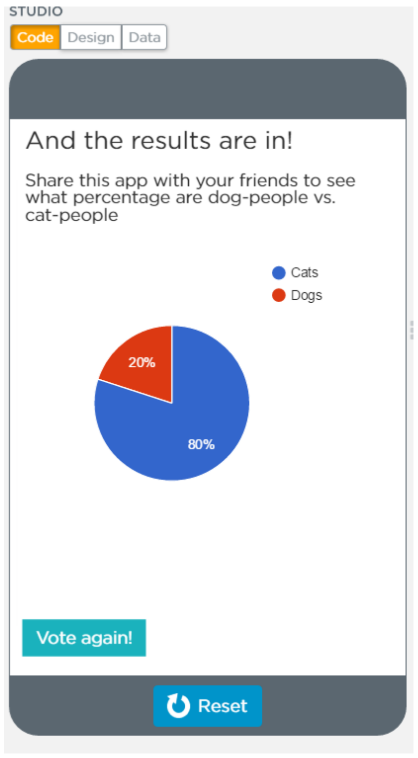
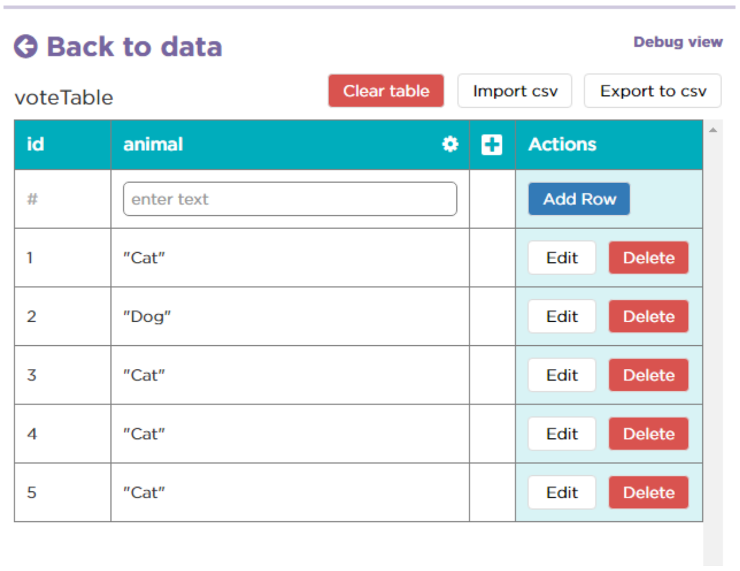

STUDIO
Code
Design
Data
And the results are in!
Share this app with your friends to see what percentage are dog-people vs. cat-people
Cats
Dogs
20%
80%
Vote again!
Reset

Figure 14: Example AppLab Visualization

Finally, this screenshot below shows some of the sample data taken by the pet voting app.



voteTable

id	animal	Actions
#	enter text	
1	"Cat"	
2	"Dog"	
3	"Cat"	
4	"Cat"	
5	"Cat"	

Figure 15: Example AppLab Database

2.4 iSENSE

iSENSE, the data science platform used within AppVis, differs from the above examples in that is does not utilize key-value pairs or SQL at all—instead, data are stored in a project. Each project corresponds to one table of data. Column names of the table are the fields of the project. Data are inserted by uploading new datasets. When uploading a dataset, the user needs to provide a matched list of column names (field names) and data.

Chapter 3: Technology Developed

This chapter outlines some of the technical aspects of the new and revised technology developed for AppVis. Additional technical details are available in Appendix B. As of August 2015, there already existed an initial version of the iSENSE Publisher component. It was called the "iSENSE component" and existed in the Social category of the App Inventor palette. It was difficult for new users to use, often froze or crashed, and had minimal error handling. Photo uploading also did not work.

The components described in this chapter act as an App Inventor-level API to the iSENSE Project. For the AppVis summer camp, we configured a dedicated instance of App Inventor for student use[2]. The code for AppVis is available publicly on Github[3]. Figure 16 shows how AppVis connects App Inventor and iSENSE.

[2] Available at https://ai.cspathways.org

[3] Available at https://github.com/farxinu/appinventor-sources

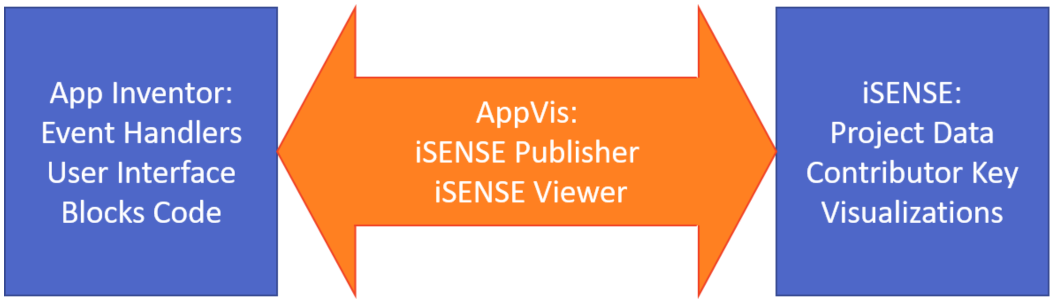


Figure 16: Connecting App Inventor to iSENSE

4.1 iSENSE Publisher

This component was built to replace the iSENSE Component. It exists in the Connectivity category of the App Inventor palette, as seen in Figure 17, and the properties available are shown in Figure 18. The properties options were simplified ot make the component easier to use. In the simplified component, there are two properties that must be filled for the iSENSE Publisher: a project ID for an iSENSE project and the corresponding contributor key (more information is in the next section).

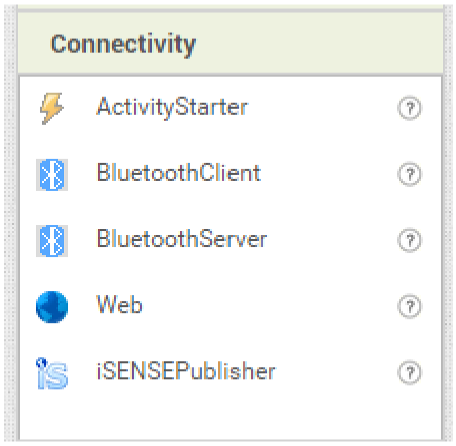
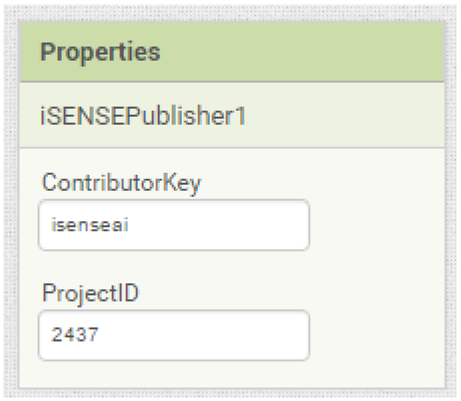


Figure 17: iSENSE Publisher in Palette



ContributorKey
isenseai
ProjectID
2437

Figure 18: iSENSE Publisher Properties

Figure 19 shows the available blocks for the iSENSE Publisher component.

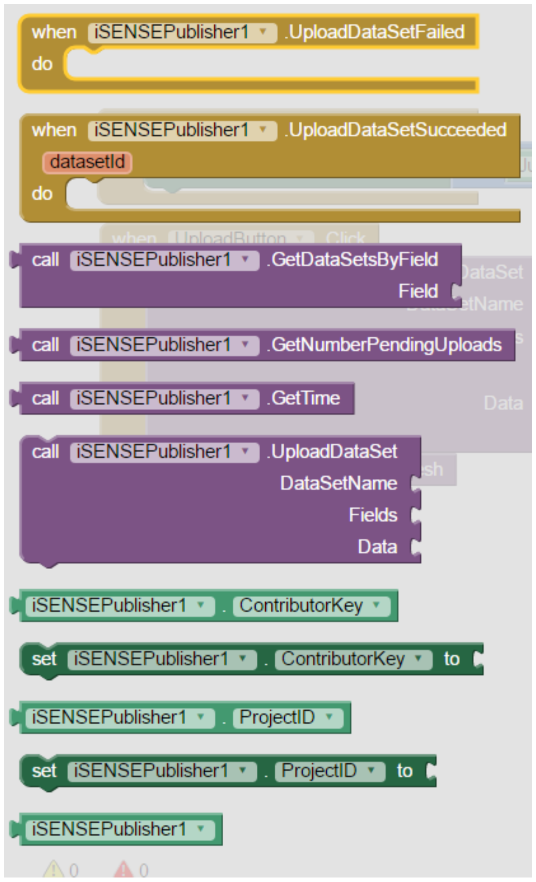

when iSENSEPublisher1 .UploadDataSetFailed
do
when iSENSEPublisher1 .UploadDataSetSucceeded
datasetId
do
call iSENSEPublisher1 .GetDataSetsByField
Field
call iSENSEPublisher1 .GetNumberPendingUploads
call iSENSEPublisher1 .GetTime
call iSENSEPublisher1 .UploadDataSet
DataSetName
Fields
Data
iSENSEPublisher1 . ContributorKey
set iSENSEPublisher1 . ContributorKey to
iSENSEPublisher1 . ProjectID
set iSENSEPublisher1 . ProjectID to
iSENSEPublisher1

Figure 19: Available Blocks for iSENSE Publisher during Summer Camp

4.1.1 Contribution Method

In the iSENSE system, there are two ways to contribute data to a project—with a contributor key (a project-specified password designated by the project owner), or with an email and password (in order to login to the website). In the iSENSE component, it was possible to use either method. However, users tended to get confused about how to use the email/password combination. Further, the email/password login option requires App Inventor to store users' iSENSE passwords in plaintext, which is a security risk. Owing to all of these complications, the email/password contribution method was removed. The iSENSE Publisher only allows data to be contributed via contributor keys, and always lists the contributor as "AppVis".

4.1.2 Data Upload

In the iSENSE component, one data set could be uploaded at a time, and the upload would fail if there was no internet connection. It used the Android UI Handler thread to upload data, which meant that the entire app would freeze during a data upload.

In order to rectify the blocking upload, the iSENSE Publisher was rewritten to use Android's AsyncTask class. The AsyncTask has four important methods: onPreExecute, doInBackground, onPostExecute, and onProgressUpdate. doInBackground, the method that does the actual upload work, is a background task, while the other three methods are on the UI thread. This allows correct management of the UI thread so that the app does not freeze and block.

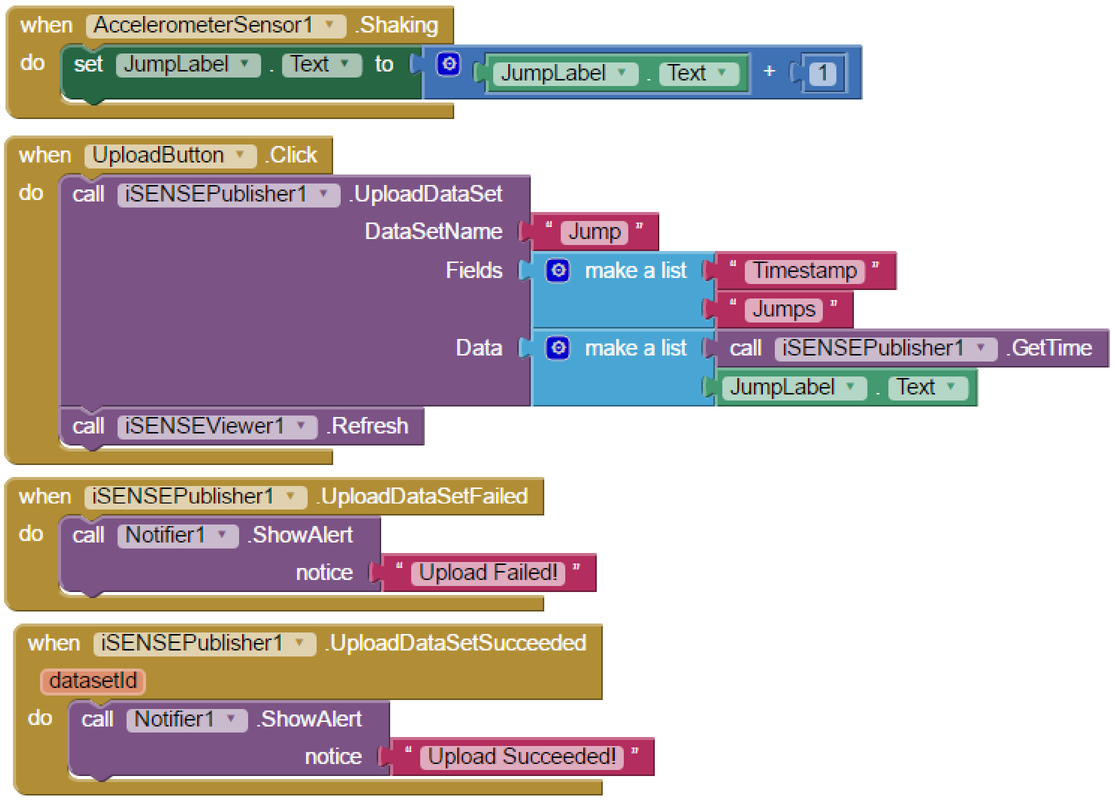
The usage of an AsyncTask also meant that it was possible to block the upload in the absence of an internet connection. The iSENSE Publisher allows up to fifty items to be in a pending upload queue. This restricts users from overzealous uploading (for example, an app that uploads every new accelerometer data point would generate a significant amount of very granular data), and also allows multiple pending uploads to be queued while there is no internet connection available. A method was added to check the size of the pending upload queue from within App Inventor (GetNumberPendingUploads).

The most important function available in the AppVis API is UploadDataSet, as it is the one that allows a user to send data to iSENSE. It requires a name for the new dataset, and a list of fields and a list of data. Each line in the list of fields must correspond to a field designated by the iSENSE project on the website. Each line in the list of data must correspond to the appropriate field in the fields list. The types of data that can be uploaded are number, text, timestamp, latitude, and longitude.

However, should a user want to create a timestamp and upload it, creating a timestamp in App Inventor is more difficult than the average user is prepared for. It requires a clock component and some configuration. The GetTime block provides easy timestamp generation that is formatted exactly how iSENSE expects timestamps to be formatted.

There is also a block named GetDataSetsByField, but it is a more advanced feature than what the average user would need. It retrieves datasets from iSENSE based on a specified field.

Finally, UploadDataSetSucceeded & UploadDataSetFailed are callback blocks that will return on successful or unsuccessful uploads. A successful upload will return the new dataset ID of the uploaded data. A failed upload most often indicates user error, including but not limited to: illegal characters in the data set name, the fields and data lists are of different sizes, incorrect project id and contributor key combination, incorrect fields, too many items in the pending upload queue, and datatype mismatches (for example, a word in a number field). Figure 20 shows sample usage of all of these blocks to upload a simple dataset to an iSENSE project.



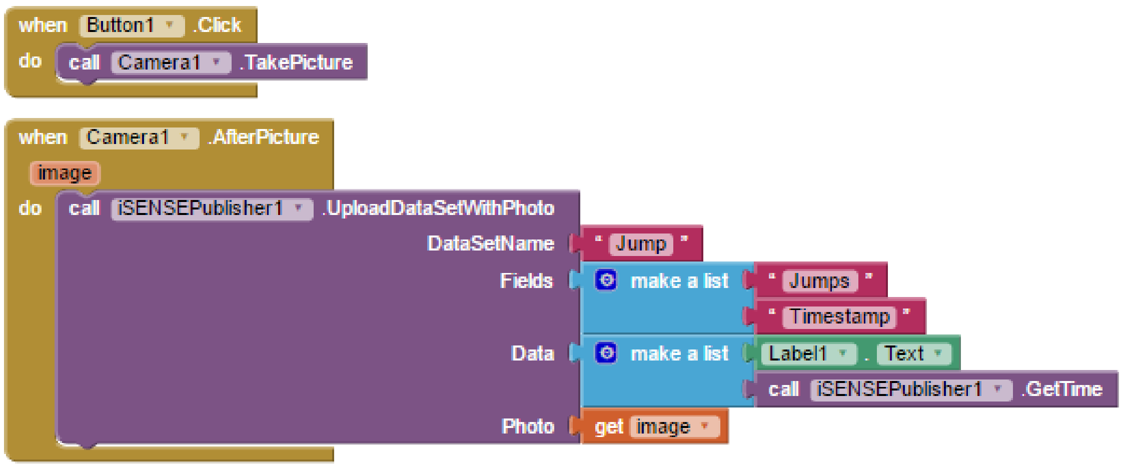
```
when AccelerometerSensor1 .Shaking
do  set JumpLabel . Text to   JumpLabel . Text + 1

when UploadButton .Click
do  call iSENSEPublisher1 .UploadDataSet
                    DataSetName  " Jump "
                    Fields       make a list  " Timestamp "
                                              " Jumps "
                    Data         make a list  call iSENSEPublisher1 .GetTime
                                              JumpLabel . Text
    call iSENSEViewer1 .Refresh

when iSENSEPublisher1 .UploadDataSetFailed
do  call Notifier1 .ShowAlert
                   notice  " Upload Failed! "

when iSENSEPublisher1 .UploadDataSetSucceeded
  datasetId
do  call Notifier1 .ShowAlert
                   notice  " Upload Succeeded! "
```

Figure 20: Sample Usage of AppVis Components

4.1.3 Photo Upload

Uploading photos to data sets was partially written in the iSENSE component, but did not work. Part of the issue is that the iSENSE API itself uses an older version of the httpmime library than App Inventor, and the backwards compatibility is not as good as it should be. Also, because photo uploading takes longer than data uploading, and photo uploading was also a blocking process, the function would appear as if it was crashing the app because the entire app would freeze for several full seconds. Finally, photo uploading in the iSENSE component was implemented as "UploadPhotoToDataSet", which required a user to know the specific ID of the dataset they wanted to upload to.

In the iSENSE Publisher component, photo uploading is implemented as "UploadDataSetWithPhoto". The method validates whether the photo exists, and if it is a photo from the App Inventor assets, or if it was a photo taken by the camera. After determining that the photo is valid, it puts an entry in the pending upload queue and the entry uploads just as the regular datasets upload. This function was removed in the summer camp version of AppVis due to instability, but was fixed and exists in the current builds of AppVis. Figure 21 shows taking a picture with a camera and attaching uploading a data set with a photo.

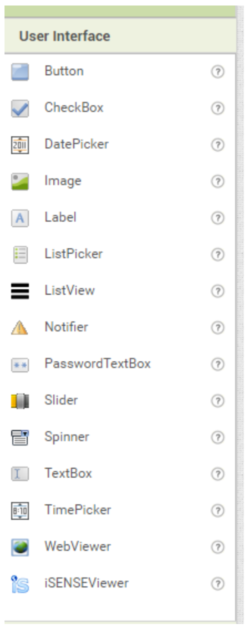


```
when Button1 .Click
do  call Camera1 .TakePicture

when Camera1 .AfterPicture
  image
do  call iSENSEPublisher1 .UploadDataSetWithPhoto
                      DataSetName  " Jump "
                      Fields       make a list  " Jumps "
                                                " Timestamp "
                      Data         make a list  Label1 . Text
                                                call iSENSEPublisher1 .GetTime
                      Photo        get image
```

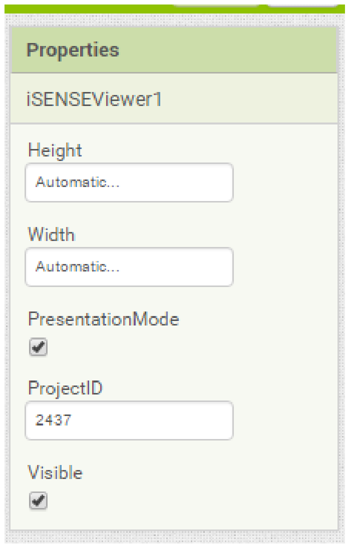
Figure 21: Sample Usage of AppVis Photo Upload

4.2 iSENSE Viewer

The iSENSE Viewer allows a user to easily display a project's default visualization within an app without having to type in a complex URL. Figures 22 and 23 show where the iSENSE Viewer lives in the Components palette, and the properties associated with the iSENSE Viewer. The iSENSE Viewer has two important fields in the properties menu: the PresentationMode option and the ProjectID. Presentation mode refers to whether or not the visualization should be displayed in full screen (presentation mode) or with a bar of controls on top (embedded mode). The ProjectID field corresponds to the iSENSE project ID that is being visualized.



User Interface
- Button
- CheckBox
- DatePicker
- Image
- Label
- ListPicker
- ListView
- Notifier
- PasswordTextBox
- Slider
- Spinner
- TextBox
- TimePicker
- WebViewer
- iSENSEViewer



Properties

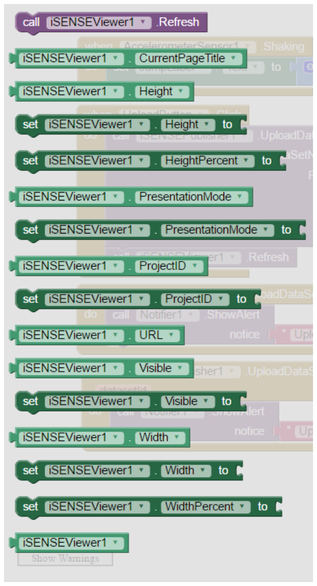
iSENSEViewer1

Height	Automatic...
Width	Automatic...
PresentationMode	✓
ProjectID	2437
Visible	✓

Figure 23: iSENSE Viewer Properties

Figure 22: iSENSE Viewer in Palette

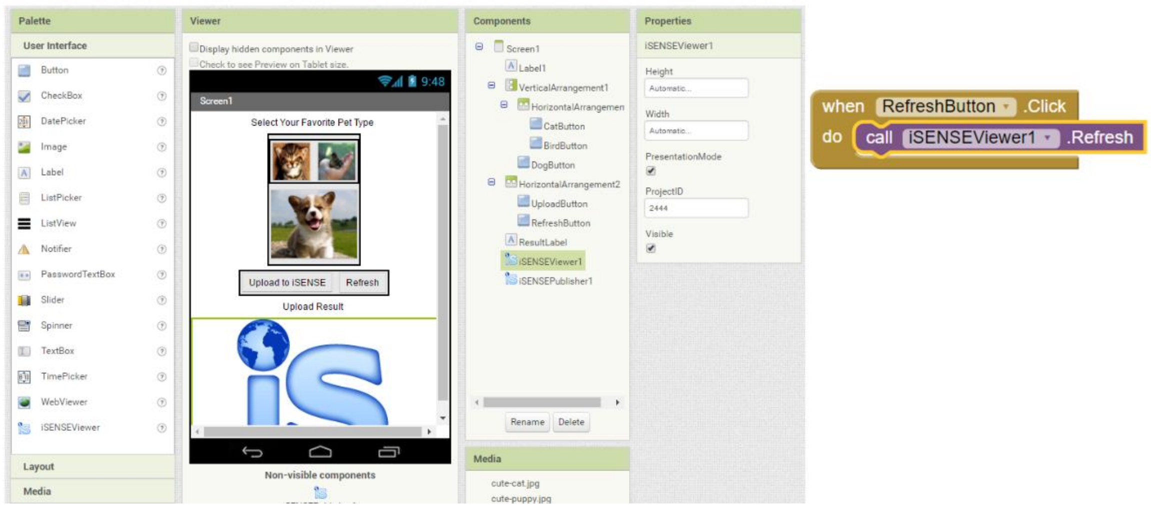
Figure 24 shows the available blocks for the iSENSE Viewer. The only function available is a call to refresh the page.



```
call iSENSEViewer1 . Refresh
iSENSEViewer1 . CurrentPageTitle
iSENSEViewer1 . Height
set iSENSEViewer1 . Height to
set iSENSEViewer1 . HeightPercent to
iSENSEViewer1 . PresentationMode
set iSENSEViewer1 . PresentationMode to
iSENSEViewer1 . ProjectID
set iSENSEViewer1 . ProjectID to
iSENSEViewer1 . URL
iSENSEViewer1 . Visible
set iSENSEViewer1 . Visible to
iSENSEViewer1 . Width
set iSENSEViewer1 . Width to
set iSENSEViewer1 . WidthPercent to
iSENSEViewer1
```

Figure 24: Available Blocks for iSENSE Viewer

The iSENSE Viewer was straightforward and simple to construct. It was built by cloning the existing WebViewer component and stripping out all unnecessary properties and functions, leaving behind only the core WebViewer capabilities. Figre 25 below shows the code and user interface layout for a sample app using the iSENSE Viewer.



iSENSEViewer1
Height: Automatic...
Width: Automatic...
PresentationMode: ✓
ProjectID: 2444
Visible: ✓

```
when RefreshButton .Click
do  call iSENSEViewer1 .Refresh
```

Figure 25: Sample App using iSENSE Viewer

Figure 26 below shows a screenshot of the above app in action, to show what the visualization actually looks like onscreen.

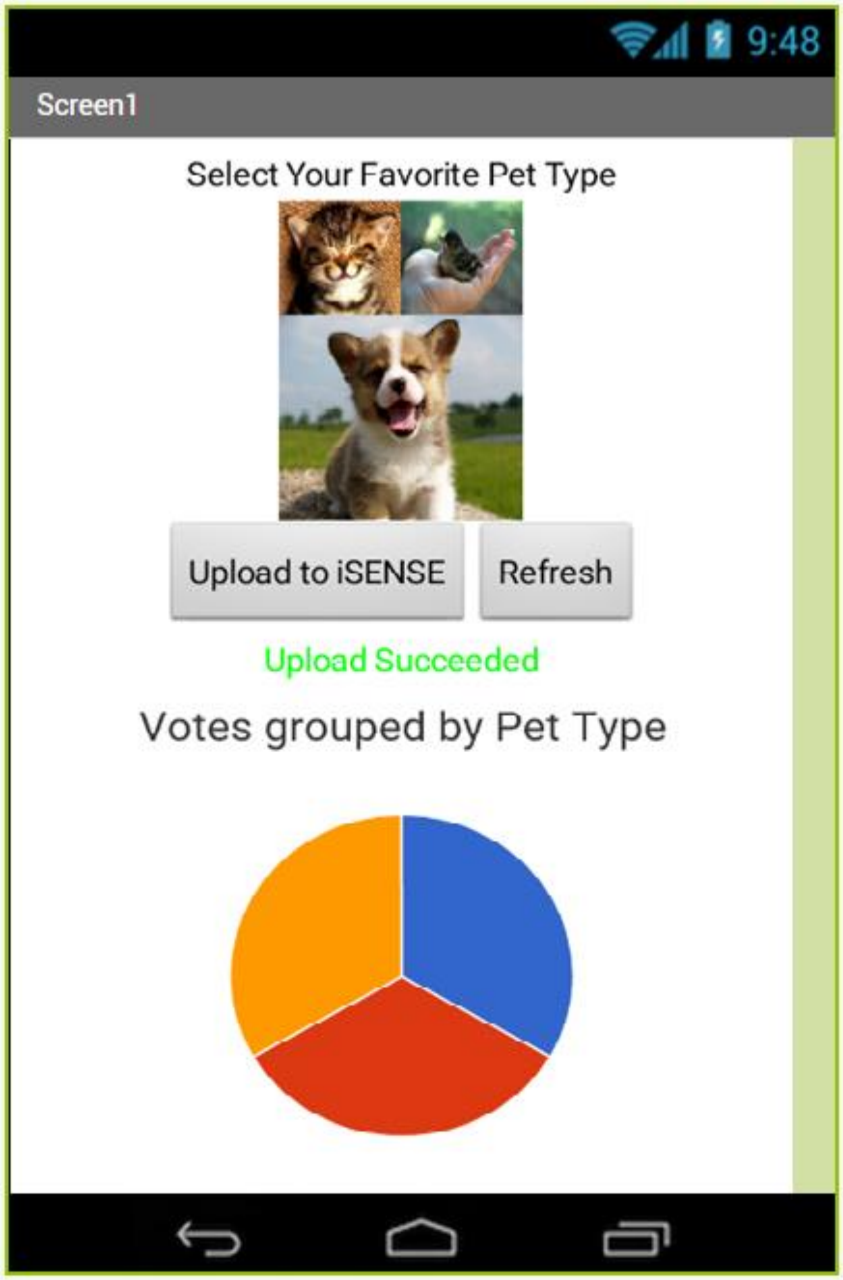


Figure 26: Screenshot of App using iSENSE Viewer

4.3 Extension

During the development of the iSENSE Publisher component, App Inventor released the public use of extensions. An extension for App Inventor is a component that can be imported into the main version of App Inventor, therefore not requiring a special version of the App Inventor environment and companion app. However, as of this writing, extensions are limited to non-visible components, so only an iSENSE Publisher component could be implemented as an extension. Figure 27 below shows the available blocks in the AppVis extension.

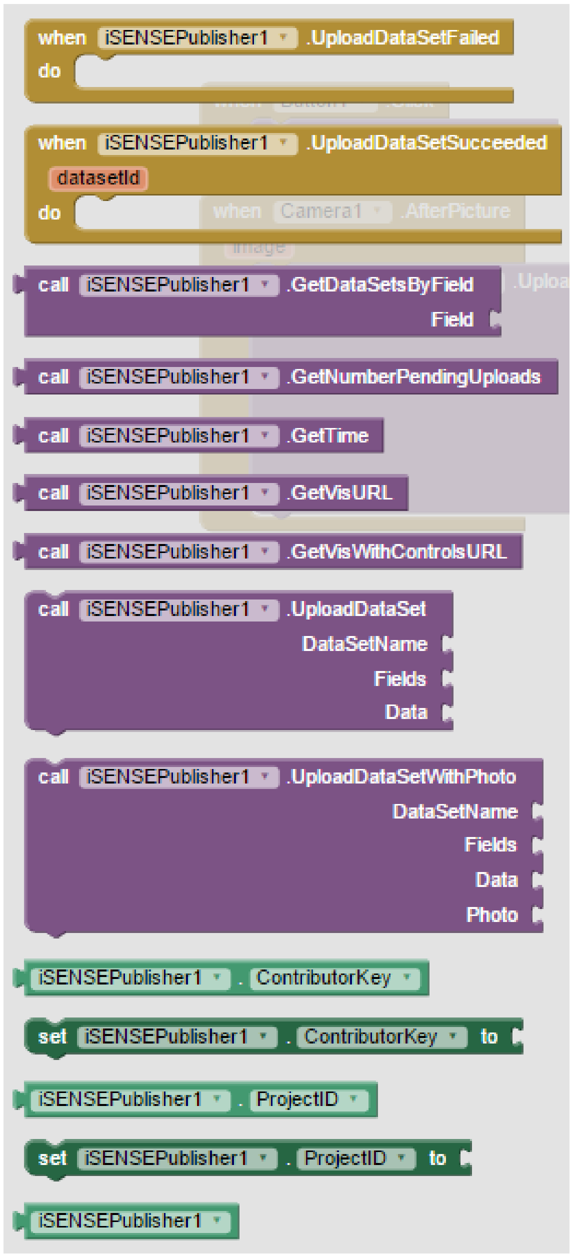
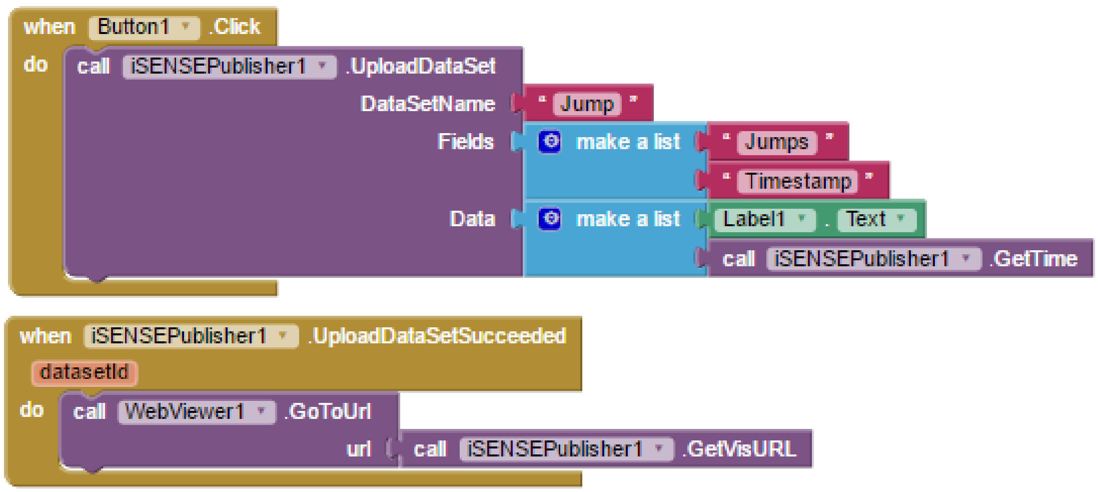

when iSENSEPublisher1 .UploadDataSetFailed
do
when iSENSEPublisher1 .UploadDataSetSucceeded
datasetId
do
call iSENSEPublisher1 .GetDataSetsByField
Field
call iSENSEPublisher1 .GetNumberPendingUploads
call iSENSEPublisher1 .GetTime
call iSENSEPublisher1 .GetVisURL
call iSENSEPublisher1 .GetVisWithControlsURL
call iSENSEPublisher1 .UploadDataSet
DataSetName
Fields
Data
call iSENSEPublisher1 .UploadDataSetWithPhoto
DataSetName
Fields
Data
Photo
iSENSEPublisher1 . ContributorKey
set iSENSEPublisher1 . ContributorKey to
iSENSEPublisher1 . ProjectID
set iSENSEPublisher1 . ProjectID to
iSENSEPublisher1

Figure 27: Available Blocks for AppVis Extension

In order to achieve the ease of visualization provided by the iSENSE Viewer, two new methods were added to the extension version of the iSENSE Publisher: GetVisURL and GetVisWithControlsURL. These URLs can be put directly into the WebViewer component in order to get a project's default visualization. These correspond to the presentation mode and embed mode described in the iSENSE Viewer component. Presentation mode, as described above, is fullscreen (GetVisURL) and embed mode has a bar of controls (GetVisWithControlsURL). Figure 28 shows example usage of GetVisURL and GetVisWithControlsURL



```
when Button1 .Click
do  call iSENSEPublisher1 .UploadDataSet
                 DataSetName  " Jump "
                 Fields       make a list  " Jumps "
                                           " Timestamp "
                 Data         make a list  Label1 . Text
                                           call iSENSEPublisher1 .GetTime

when iSENSEPublisher1 .UploadDataSetSucceeded
 datasetId
do  call WebViewer1 .GoToUrl
                 url  call iSENSEPublisher1 .GetVisURL
```

Figure 28: Example Usage of AppVis Extension

Chapter 4: Demonstration Study

We ran AppVis in two one-week summer camps for middle schoolers in two diverse school districts. This chapter presents more information on how the camp was conducted, and the campers themselves.

4.1 Student Demographics

Here we present the demographics of the students who participated in the 2016 AppVis summer camp. One of the goals of this project is to make Computer Science more accessible to all students, regardless of gender or race. Statistics are provided to show that we had a group of students that did not necessarily exactly reflect the districts, but still had a wide variety of diversity. Table 1 shows the racial composition of the students in the districts as well as in our camps.

Race	*District 1*	*District 2*	*District 1 Campers*	*District 2 Campers*
African American	14.50%	18.00%	18.75%	9.09%
Asian	8.70%	4.90%	18.75%	9.09%
White/Caucasian	62.60%	30.80%	37.50%	27.27%
Hispanic	9.80%	43.90%	6.25%	36.36%
Native American	0.30%	0.50%	6.25%	0.00%
Pacific Islander & Other	4.10%	2.00%	12.50%	36.36%

Table 1: Racial Composition of Districts and Camp 2016 Participants

We had equity of not only race, but gender. Figure 29 shows the gender composition of the campers who attended our AppVis camp.

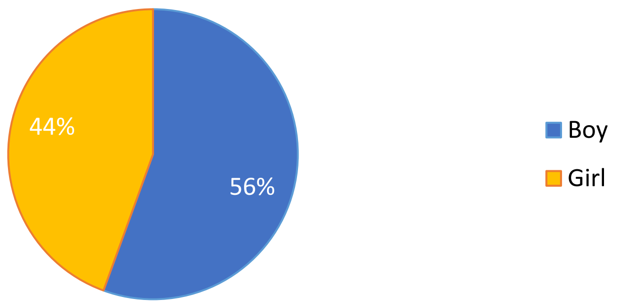


Figure 29: Gender Composition of Campers

When considering the students' languages, over half of students reported being multilingual (speaking a language other than English at home). Figure 30 shows the percentages of languages spoken at home other than English.

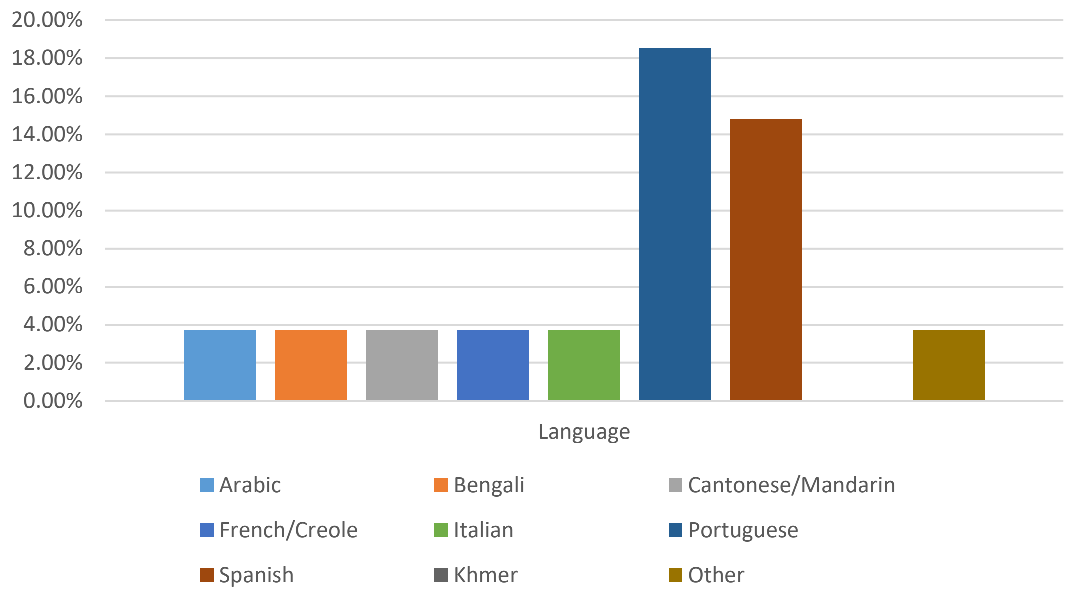


Figure 30: Languages Spoken at Home (Except English)

In order to gauge the relative experience of our campers, we asked for their grade level and prior programming experience. The results of these queries are displayed in the following figures. For grade level, the students that selected "Other" were in fifth grade going into sixth. The results are shown in Figure 31. Most campers were in 7th or 8th grade, which means our average camper was about thirteen years old.

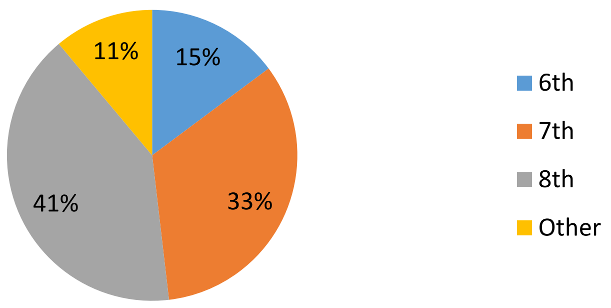


Figure 31: Grade Levels of Campers

Figure 32 below shows how much time students had identified as having programmed before they started the AppVis camp. Over half of the students had had less than one hour of prior programming experience, which means that most of our campers were novices, and the AppVis camp was their first proper introduction to programming or computing.

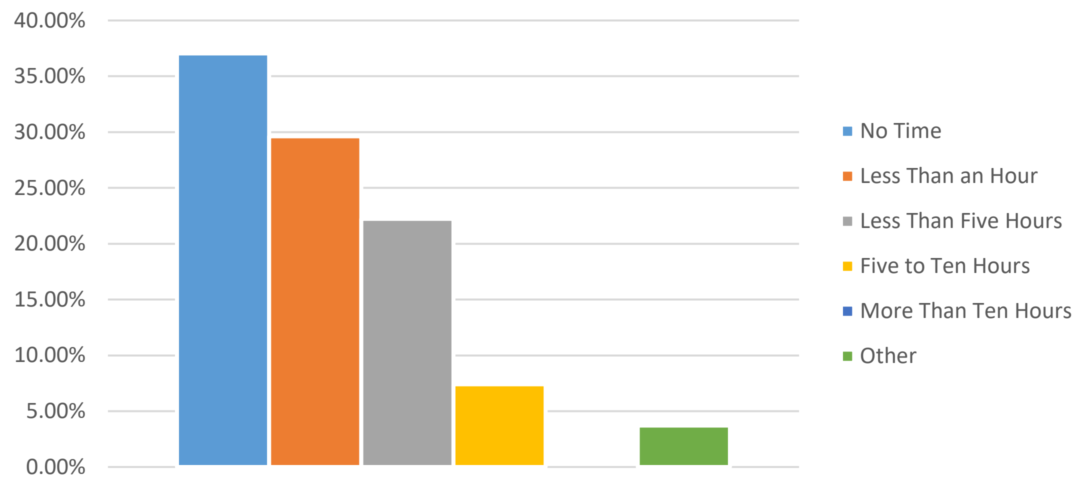


Figure 32: Time Spent Programming before Camp Start

To further gauge students' prior experience, we asked them which programming environments they had used before. Figure 33 shows the breakdown of which environments students had previously used. Half of the students reported as having used App Inventor before, which meant that AppVis would be easier for them to grasp. Students also reported having used Scratch and Code.org, which have similar block-based interfaces to App Inventor; again, this made App Inventor a less foreign environment. Several students indicated that they had used some different programming environments, but it should be noted that students were able to select multiple options—these options were not mutually exclusive.

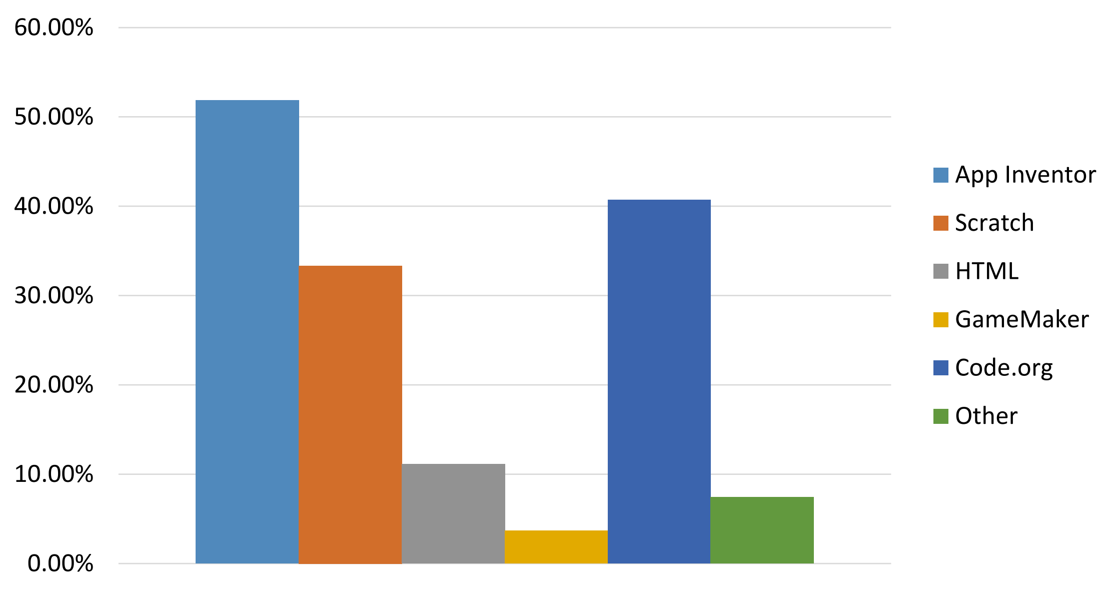


Figure 33: Types of Prior Experience

4.2 Curriculum

The initial thought for the summer camp curriculum was to use standard App Inventor with more utility-focused tutorials rather than game-based tutorials. However, when trying to design a tutorial to get a location and display it on a map, it ended up being far too difficult for the scope of an ideal tutorial. However, with the iSENSE Publisher and Viewer, uploading a location and displaying it on a map visualization is trivially simple. The ease of iSENSE was a driving force in using an iSENSE-augmented version of App Inventor rather than MIT's standard App Inventor.

In the first iteration of the camp, App Inventor was introduced to the students through the Maker Cards and through App Inventor's standard beginner tutorials, including Paint Pot and Mole Mash. Student apps at the end were, with a few exceptions, modifications of these tutorials. Over half of the student apps had versions of Paint Pot and/or Mole Mash. We felt that the Maker Cards were too simple, as each of them teaches one small concept, and that the tutorials themselves were too inflexible and potentially too complicated. It seemed that the students had trouble gleaning concepts from the tutorials and applying them to their own apps, and instead just made the same app as the tutorial again. These perceived issues, along with the fact that we were going to use AppVis instead of App Inventor, meant that we needed to create our own tutorials for this curriculum. Details about these tutorials are in the following section.

Additionally, at the end of the first iteration of the camp, many students reported in interviews that they had not built "real" apps, and the majority of the students said they did not want to publicly share their apps (Ni, Sherman, Schilder, & Martin, 2016). The idea of a "real" app can mean many things, and we were unsure as to how to rectify this issue. We determined that a "real" app would be something useful (a "utility" app) with a specific customer (a community partner), that would be made available to the public. Therefore, we took the approach of teaching students how to build utility apps rather than games, and allowed students to publish their apps on Google's Play Store if they so desired. However, because we would be potentially releasing the apps to the public, we needed to ensure that the students' apps had no copyright violations, and we needed to add an activity teaching them about Creative Commons.

Other design considerations for the curriculum included choosing activities, order and length of student activities as well as research activities, as well as whether or not there should be a visit from an industry professional. Two activities referred to as the "debugging activity" and the "temperature activity" were research components for another student's PhD work.

Our final version of the curriculum is shown in Table 2, and the following sections describe the activities in more detail. It should be noted that lunch and recess are not accounted for in these schedules, but were an hour break in the middle of each day. Research interviews were conducted all day Thursday and the first half the day on Friday for both camps.

M	T	W	TH	F
Introductions	Pair programming Intro	Survey Tutorial	Project work	Research surveys
Research surveys	Final app brainstorming	Project work		Finalize apps
Talk to Me Tutorial	GPS Tutorial	Debugging Activity		
Jump Count Tutorial		How to publish to Play Store	Temperature Activity	Present apps in “App Fair”
	Creative Commons & Copyright	Project work	Project Work	
Community Partner visit	“Storyboard” apps	Class-wide demos		

Table 2: Camp 2016 Schedule

4.2.1 Tutorials

Four tutorials were written for this camp: Talk to Me with Media Upload, Jump Count, GPS, and Survey. At the end of each tutorial, some extension ideas to make the apps more complex were added for students who finished before their peers. These tutorials are detailed below.

The first tutorial is a very simple app that is an extension of MIT's Talk to Me tutorial[4]. It shows students how to use the Layout and the Blocks tabs of App Inventor. It has a button and a Text-to-Speech component. It also guides students through uploading media to App Inventor and setting a photo as the background of the app. This tutorial is meant as a basic introduction to the App Vis camp for students that have never worked with App Inventor before, and to reacquaint students who had previously worked with App Inventor. It has no iSENSE-specific instruction.

Figure 34 shows the user interface for the end product, and Figure 35 shows the associated code.

[4] http://appinventor.mit.edu/explore/ai2/beginner-videos.html

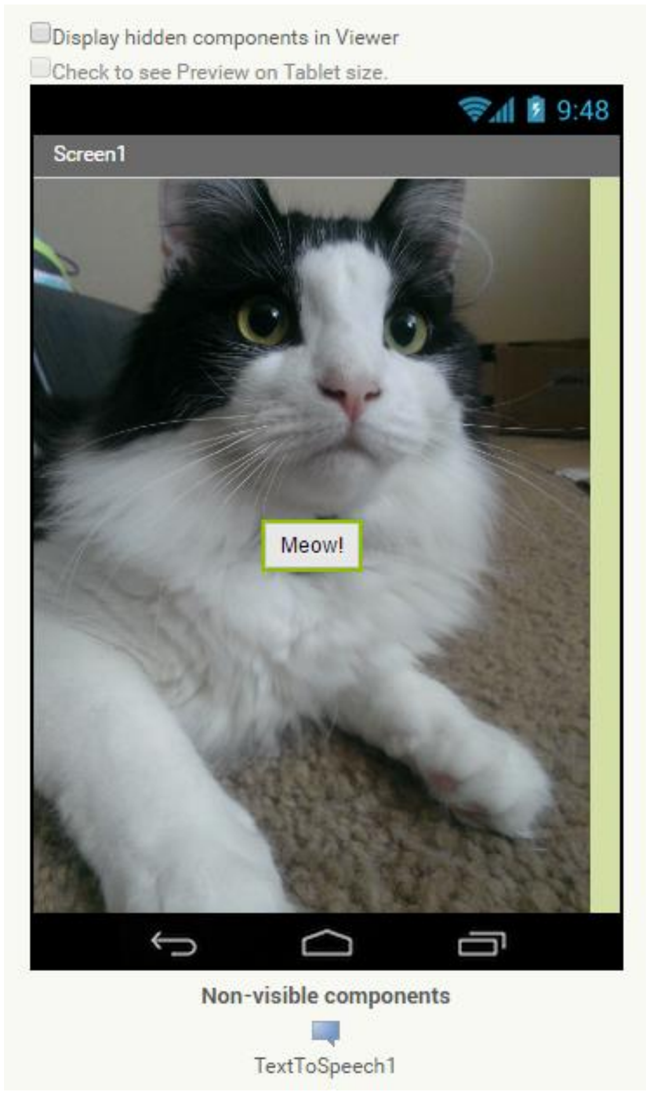
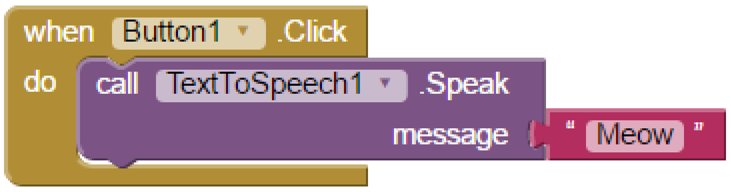

Display hidden components in Viewer
Check to see Preview on Tablet size.
9:48
Screen1
Meow!
Non-visible components
TextToSpeech1

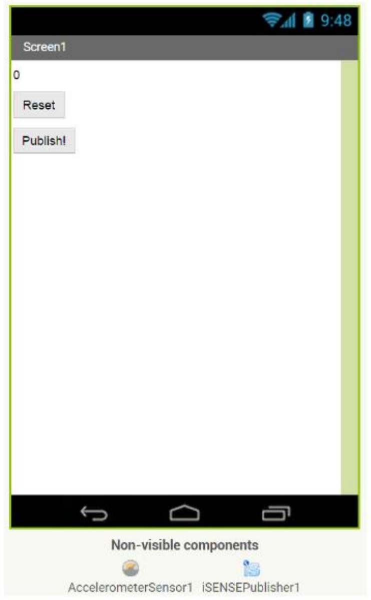
Figure 34: Talk to Me with Media Upload GUI



```
when Button1 .Click
do  call TextToSpeech1 .Speak
                      message " Meow "
```

Figure 35: Talk to Me with Media Upload Code

The second tutorial, Jump Count, introduces students to simple data gathering and collaboration. Students put an accelerometer into their app, and when they jump (or, more accurately, the accelerometer detects shaking), a counter is incremented onscreen. The second part of this tutorial guides students through uploading this data to an iSENSE Project, and then viewing all of the jumps as a histogram on the iSENSE website. As a secondary exercise, instructors in the camp took this opportunity to teach the students basic graph interpretation. The following figures show the user interface, code, and sample iSENSE visualization.



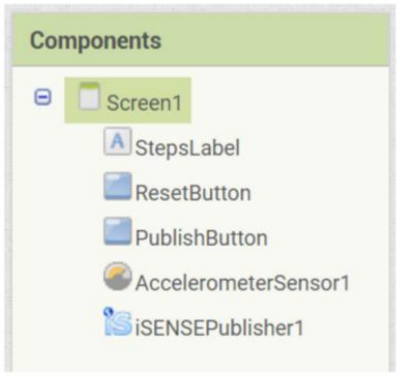
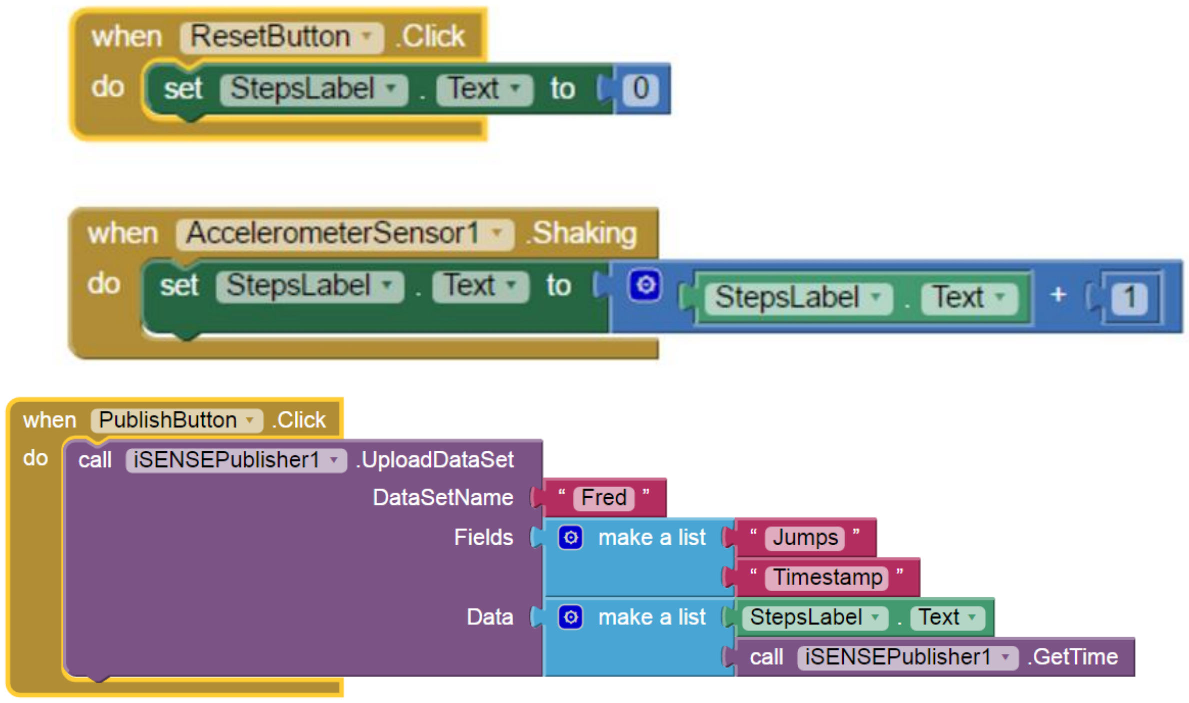


Figure 36: Jump Count User Interface



```
when ResetButton .Click
do  set StepsLabel . Text to 0

when AccelerometerSensor1 .Shaking
do  set StepsLabel . Text to StepsLabel . Text + 1

when PublishButton .Click
do  call iSENSEPublisher1 .UploadDataSet
                         DataSetName  "Fred"
                         Fields       make a list  "Jumps"
                                                   "Timestamp"
                         Data         make a list  StepsLabel . Text
                                                   call iSENSEPublisher1 .GetTime
```

Figure 37: Jump Count Code

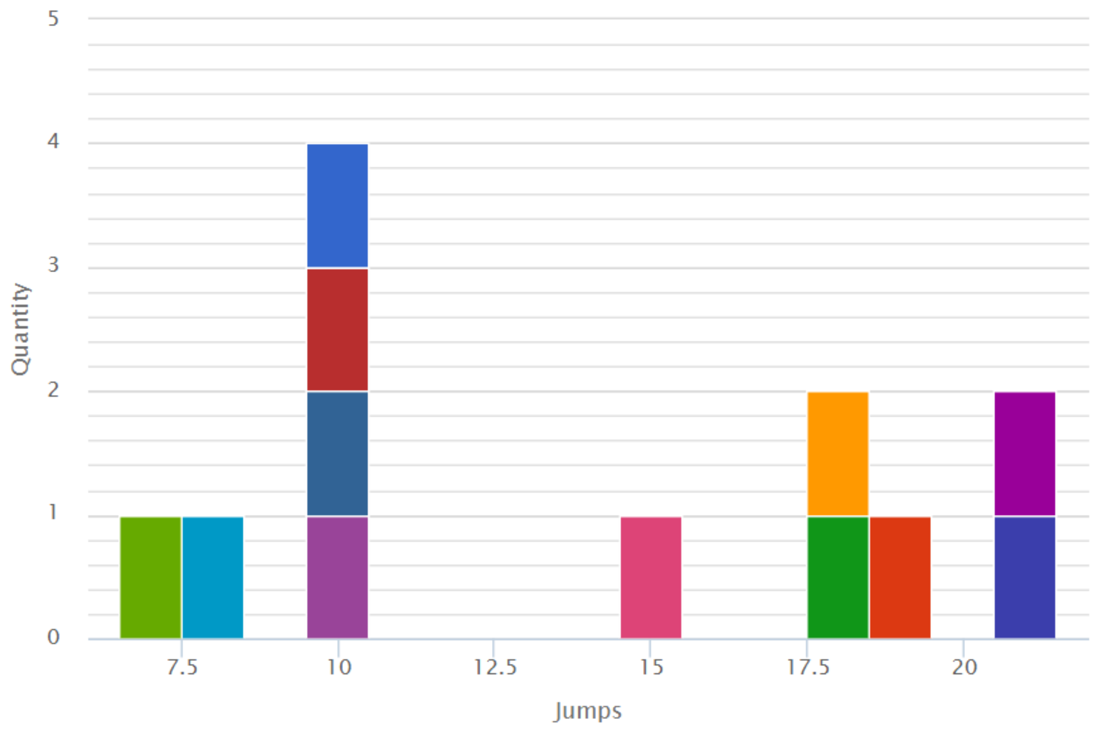


Figure 38: Jump Count Visualization

The GPS tutorial is similar to the accelerometer-based jump count survey, but uses the Location Sensor instead. It introduces the idea of latitude and longitude and the concept of GPS. It then walks students through uploading their location and viewing it on a map as an iSENSE visualization. The challenge of this tutorial is that the Location Sensor initializes with a (0,0) location, and students have to learn to see if their data is valid before uploading it. This also is a good place to introduce the Upload Failed and Upload Succeeded callback blocks of the iSENSE Publisher component. Because GPS signal quality is poor indoors, this tutorial served as a perfect opportunity for a guided, educational recess outside. The below figures demonstrate a sample user interface, code, and visualization.

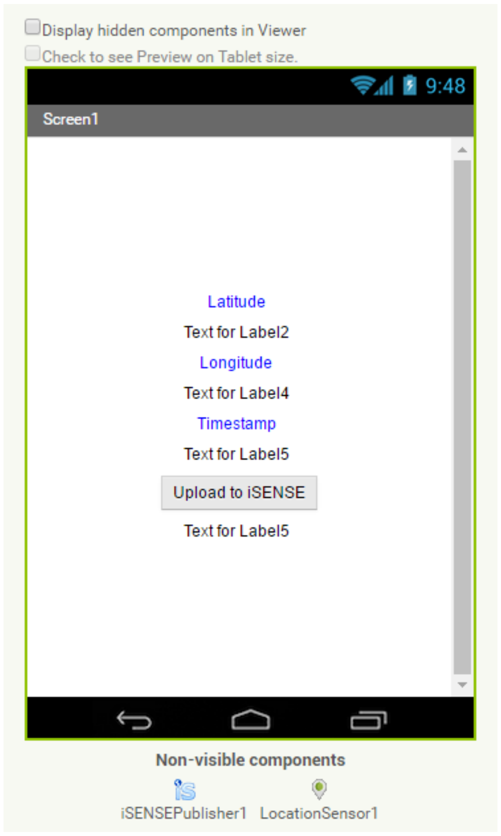
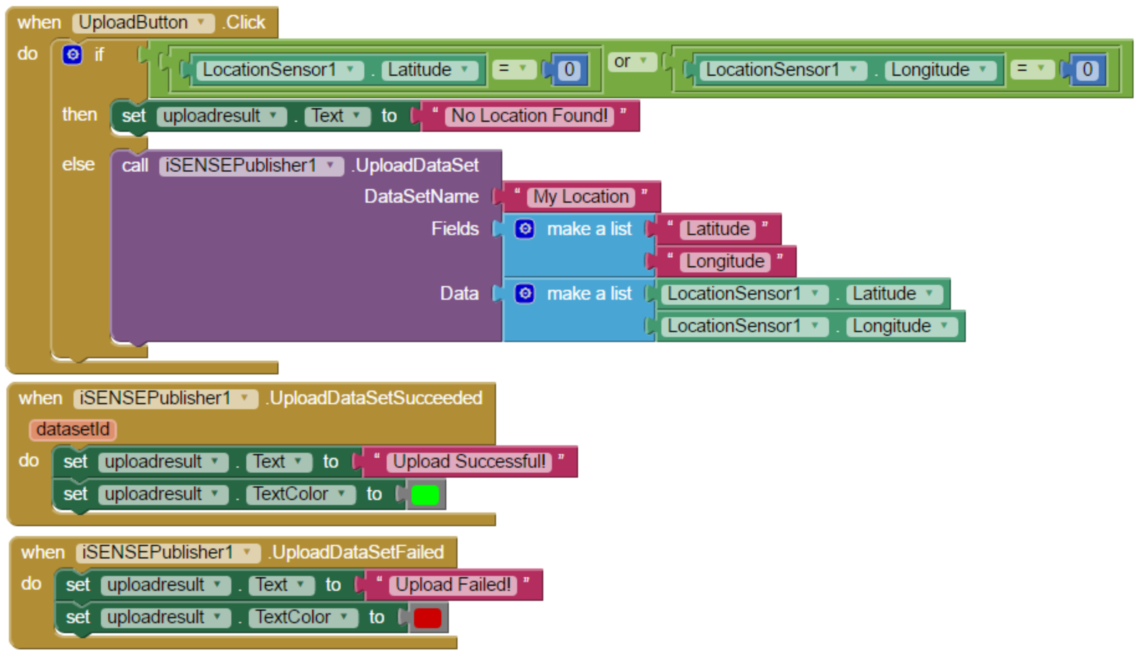


Figure 39: GPS User Interface



```
when UploadButton .Click
do  if   LocationSensor1 . Latitude = 0  or  LocationSensor1 . Longitude = 0
    then set uploadresult . Text to " No Location Found! "
    else call iSENSEPublisher1 .UploadDataSet
                   DataSetName " My Location "
                   Fields      make a list " Latitude "
                                           " Longitude "
                   Data        make a list LocationSensor1 . Latitude
                                           LocationSensor1 . Longitude

when iSENSEPublisher1 .UploadDataSetSucceeded
 datasetId
do  set uploadresult . Text to " Upload Successful! "
    set uploadresult . TextColor to 🟩

when iSENSEPublisher1 .UploadDataSetFailed
do  set uploadresult . Text to " Upload Failed! "
    set uploadresult . TextColor to 🟥
```

Figure 40: GPS Code

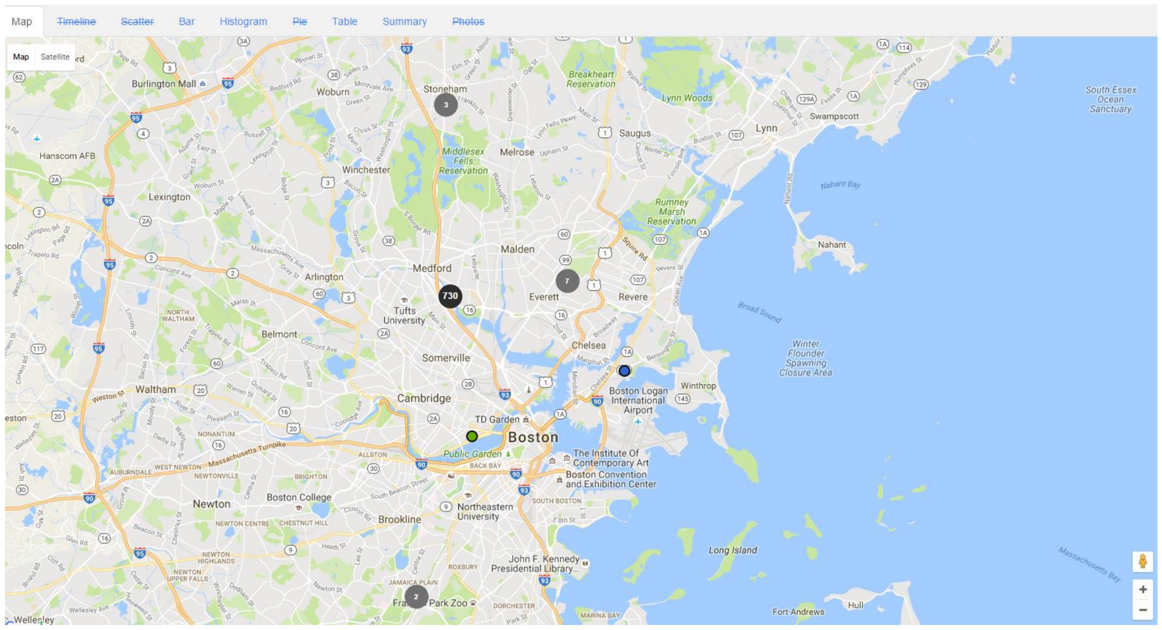
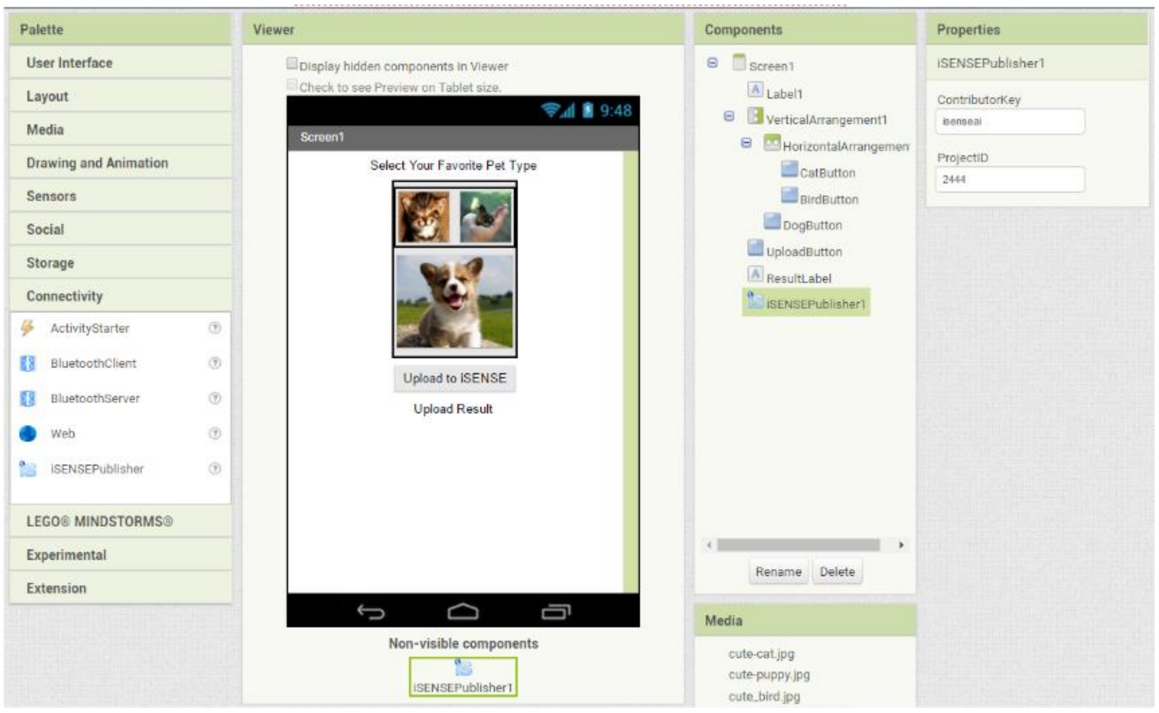


Figure 41: GPS Visualization

The survey is the final and most complex tutorial. It guides students through making a layout of buttons onscreen, and then programming each button to change a global variable. Then, it guides students through making their own iSENSE projects for their survey responses. Students then link their apps to their own iSENSE projects, and add an iSENSE Viewer to visualize their survey results within their apps. Teachers took this chance to encourage collaboration and have students take each other's surveys. Then, as a class, survey results were presented—another opportunity for a quick lesson on data interpretation.

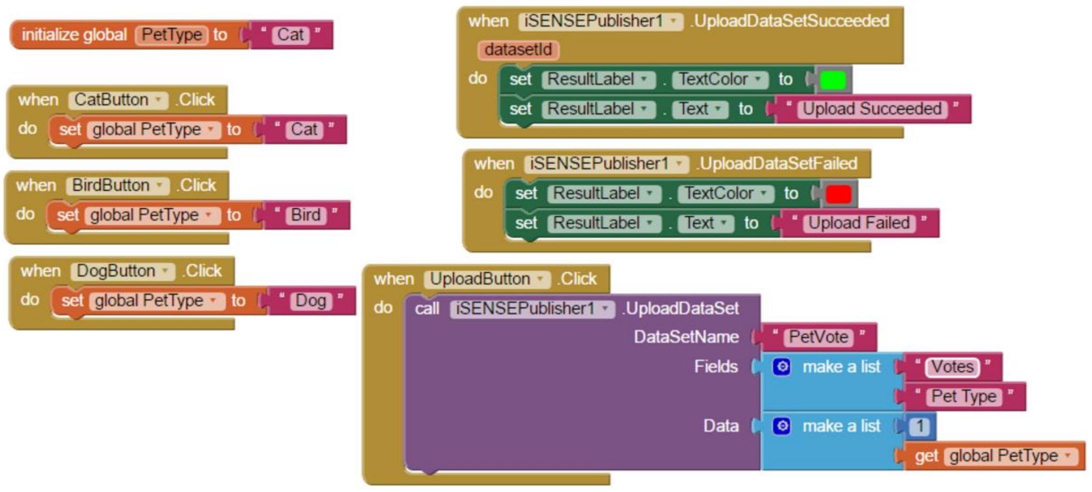
The following figures show a sample user interface and the corresponding code. A screenshot of the app in action is shown in Figure 42.



ContributorKey: isenseai

ProjectID: 2444

Figure 42: Survey App User Interface



```
initialize global PetType to "Cat"

when CatButton .Click
do  set global PetType to "Cat"

when BirdButton .Click
do  set global PetType to "Bird"

when DogButton .Click
do  set global PetType to "Dog"

when iSENSEPublisher1 .UploadDataSetSucceeded
  datasetId
do  set ResultLabel . TextColor to
    set ResultLabel . Text to "Upload Succeeded"

when iSENSEPublisher1 .UploadDataSetFailed
do  set ResultLabel . TextColor to
    set ResultLabel . Text to "Upload Failed"

when UploadButton .Click
do  call iSENSEPublisher1 .UploadDataSet
                 DataSetName  "PetVote"
                 Fields       make a list  "Votes"
                                           "Pet Type"
                 Data         make a list  1
                                           get global PetType
```

Figure 43: Survey App Code

4.2.2 Community Partner Visit

In order to engage students in building community-based social apps, some kind of motivation needed to be provided. To this end, we had several volunteers from local non-profits (our community partners) come in and discuss some issues that they faced that could be potentially alleviated with technology. They came in on the first day to present their topics, and on the last day to see what the students had built.

4.2.3 Play Store Publishing

In order to strengthen the idea that the students were building real apps, we built in an incentive: All of the apps that the students built would be published to the Google Play Store (should the students want their apps published). This motivated the students to build something that worked and that they would be proud to share with the world.

4.2.4 App Fair

The camps both culminated in what we referred to as an "App Fair", where the students presented their apps to each other, parents, and the community partners.

Each student app developer stood in front of the audience and presented their own apps, describing not only what they had done and what they were proud of, but also what they would have done if they had had more time. The app fairs themselves were evidence of success, as the students were incredibly proud of what they had done and felt that they had indeed built real apps.

The community partners themselves expressed gratitude and pride. Several of the partners approached the students to ask if their organizations could use these apps, as they were already available to download from the Play Store. One community partner asked a group of students to include her organization's official logo in the app. Another partner said, "If I could have all the students combine their individual apps into one app, then that would be the perfect app for our organization."

4.2.5 Order of Events

The camp ran for two weeks in the two school districts. While the activities themselves remained the same, some activities ran longer or shorter than planned between the two camps, and activities required reordering. The biggest difference was that the first camp began programming their final apps before having finished all the tutorials, while the second camp began their final apps after having finished the tutorials.

The first tutorial, acted as a "Level 0" App Inventor-only tutorial. Jump Count and GPS were both "Level 1" tutorials, with both App Inventor and AppVis, using an existing iSENSE project. Finally, the survey app served as a "Level 2" tutorial, as it used App Inventor, AppVis, and guided students through creating their own iSENSE projects.

The differences between the two camps are shown in Tables 3 & 4. In District 2's camp, one of the instructors set up Google Classroom to make turning in .AIA files (App Inventor's project files) easier. In District 1's camp, we had students upload their files to a shared Google Drive folder.

Interestingly enough, the complexity of the apps remained equivalent between the two camps, as did student sentiment (further detail is presented in the Analysis section). This leads us to believe that order of events does not matter, as long as the events themselves remain the same.

M	T	W	TH	F
Introductions	Pair programming Intro	Survey Tutorial	Project work	Temperature Activity
Research surveys	Final app brainstorming	Debugging Activity	Instructor-led careers discussion	Finalize apps
Talk to Me Tutorial	GPS Tutorial	How to publish to Play Store	Project Work	Post-Surveys
Jump Count Tutorial				Instructors review how to give presentations
	Creative Commons & Copyright	Project work		Present apps in "App Fair"
Community Partner visit	Project Work			
App Sketches	Class-wide Demos	Demos to Instructors	Students test each other's apps	

Table 3: District 1's Actual Schedule

M	T	W	TH	F
Introductions	Morning "Scrum"	Morning "Scrum"	Morning "Scrum"	Morning "Scrum"
Research surveys	Complete Jump Count	Finish Survey Tutorial	Project Work	How to publish to Play Store
Google Classroom Setup	Pair Programming Intro	Brainstorming		Finalize apps
Talk to Me Tutorial	GPS Tutorial	Storyboarding		
Community Partner visit				Instructors review how to give presentations
	Creative Commons & Copyright	Project work		Present apps in "App Fair"
Jump Count Tutorial	Survey Tutorial			
		Debugging Activity	Temperature Activity	
End of Day "Scrum"	End of Day "Scrum"	End of Day "Scrum"	End of Day "Scrum"	

Table 4: District 2's Actual Schedule

Chapter 5: Analysis of Student Work

In this section, we present the outcomes of the AppVis summer camp by providing several different formats of analysis of student work. First, we present several sample apps from both weeks of the summer camp to show some examples of what the students actually built. Then, a complexity and thematic analysis is presented to show the differences between the AppVis camp (Camp 2016) and the previous camp (Camp 2015). Finally, we show analyzed results of student interviews.

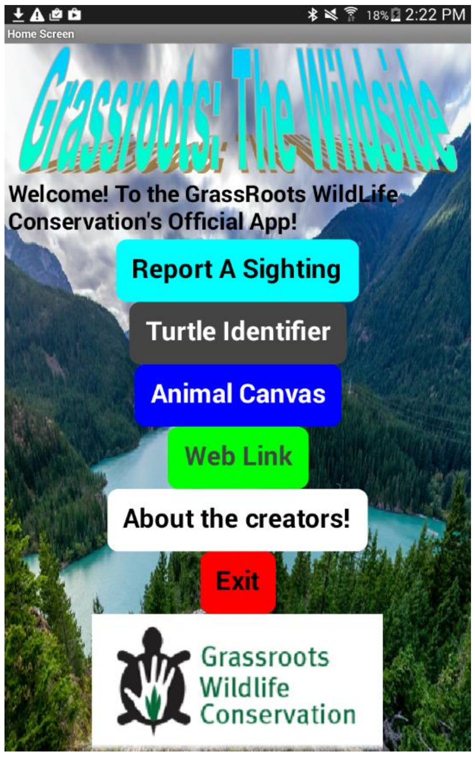
5.1 Sample Student Work

In this section, several student apps have been presented to show sample student work with AppVis. Sample screenshots, the AppVis code, and the corresponding iSENSE visualization are presented. For each app that uses AppVis, the students created both the App Inventor projects as well as corresponding iSENSE projects, then used the AppVis blocks to connect the two.

5.1.1 Grassroots: The Wildside

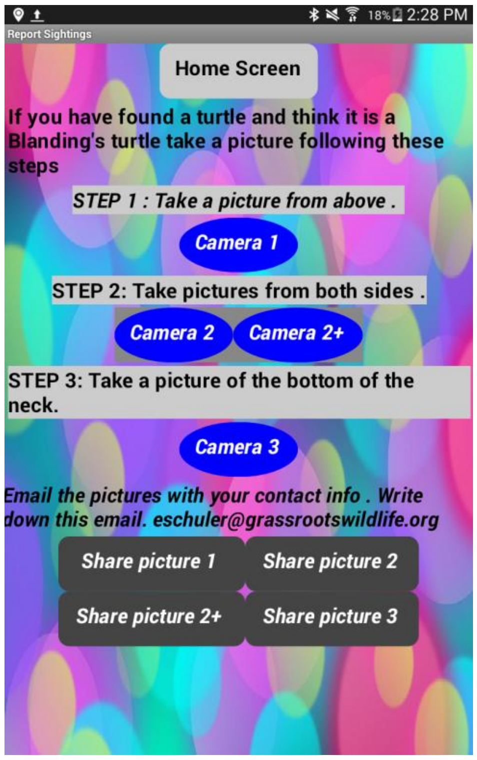
One of the community partners was with the Grassroots Wildlife Conservation, and she came in to discuss the plight of an endangered local species, the Blanding turtle. During her presentation, she described the "perfect" app for tracking these turtles. Ultimately, she wanted a system with which the community and the volunteers could see where these turtles are.

Two students decided separately that they wanted to create this app, and then decided to work together to build it. They used AppVis to report turtle sightings, but also added in a link to the Grassroots website and a PaintPot-like minigame. However, they also wanted to add in the ability to take four photos and email them to the community partner. This ended up being the most challenging part for them in creating their app.



Welcome! To the GrassRoots WildLife Conservation's Official App!

Figure 44: Grassroots Screenshot 1



If you have found a turtle and think it is a Blanding's turtle take a picture following these steps

STEP 1 : Take a picture from above .

STEP 2: Take pictures from both sides .

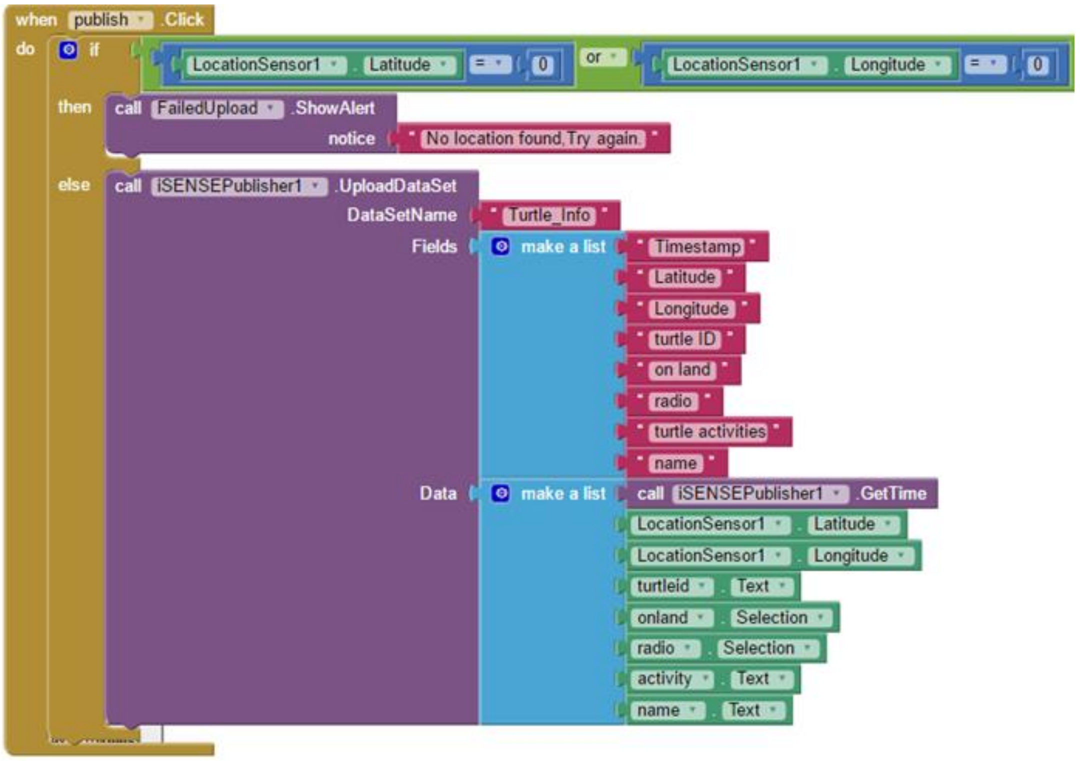
STEP 3: Take a picture of the bottom of the neck.

Email the pictures with your contact info . Write down this email. eschuler@grassrootswildlife.org

Figure 45: Grassroots Screenshot 2

During the App Fair at the end of the week, the community partner was so impressed with the students' work that she asked them to put the Grassroots logo on their app so that it could be marked as the organization's official app.

Figure 44 and 45 are selected screenshots from the Grassroots app. The students very much wanted their own logo of sorts, and settled on WordArt to accomplish this.



```
when publish .Click
do  if    LocationSensor1 . Latitude = 0   or   LocationSensor1 . Longitude = 0
    then  call FailedUpload .ShowAlert
                         notice  " No location found, Try again "
    else  call iSENSEPublisher1 .UploadDataSet
                     DataSetName  " Turtle_Info "
                          Fields  make a list  " Timestamp "
                                               " Latitude "
                                               " Longitude "
                                               " turtle ID "
                                               " on land "
                                               " radio "
                                               " turtle activities "
                                               " name "
                            Data  make a list  call iSENSEPublisher1 .GetTime
                                               LocationSensor1 . Latitude
                                               LocationSensor1 . Longitude
                                               turtleid . Text
                                               onland . Selection
                                               radio . Selection
                                               activity . Text
                                               name . Text
```

Figure 46: Grassroots AppVis Code

Figure 46 shows the publish code for the Grassroots app. It checks to see if there is a legitimate location detected by the location sensor, and, upon success, uploads all the appropriate information to the corresponding iSENSE project that they built.

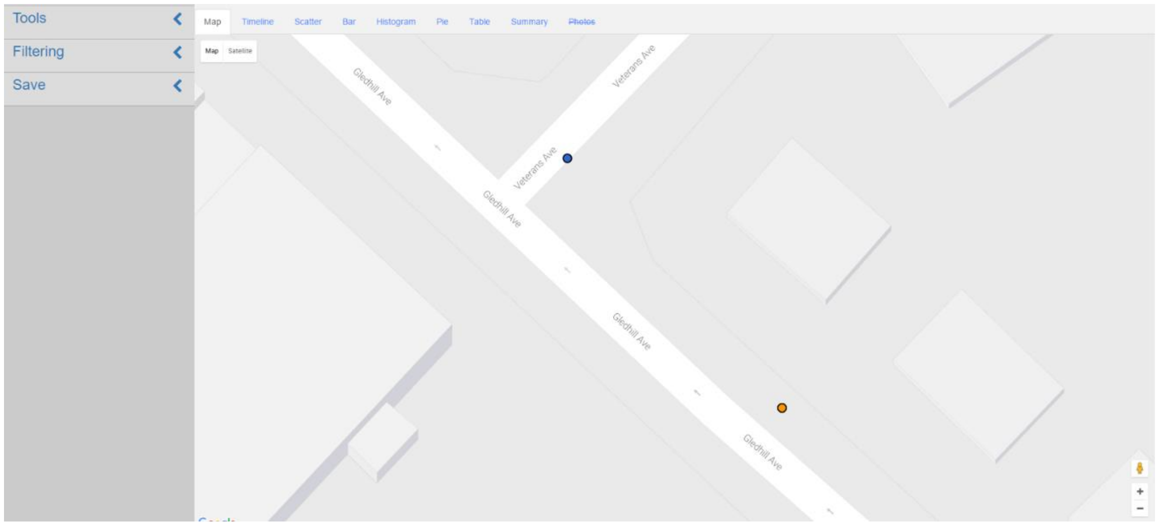
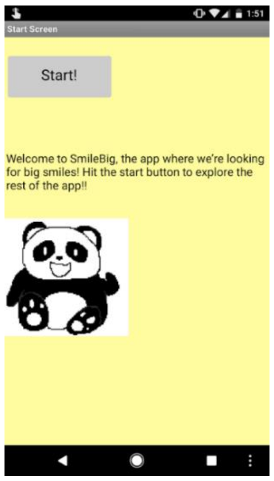


Figure 47: Grassroots Default iSENSE Visualization

Finally, Figure 47 shows the default visualization of the app on iSENSE. The students set the default to a map so that the turtle locations are the first thing people see. In the screenshot above, only test data is shown.

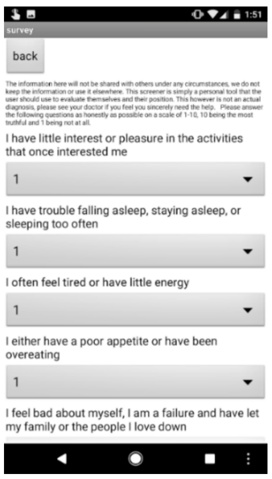
5.1.2 Smile Big

One of the community partners who came in was affiliated with public health, and she briefly mentioned that there are not enough resources for those suffering from mental health issues. Two students who had personal experience with depression took it upon themselves to create an app that would help users identify if they were depressed and to feel better if they were. The students did their own research to create a “screening” questionnaire to find if the user was depressed, and some information about depression. They also added several minigames to their app to add an element of fun. Their app also included a list of questions asking the user to input some of the things that make their own lives better. Finally, they used AppVis to have users upload self-compliments so that everyone could see these affirmations and feel better about themselves. They drew their own “mascots” (as they referred to them) so as to not infringe on copyright. The mascot and some selected screenshots from the app are shown in Figures 48, 49, and 50. These screenshots show the home screen, part of the screening, and the page that interfaces with iSENSE.



Welcome to SmileBig, the app where we're looking for big smiles! Hit the start button to explore the rest of the app!!

Figure 48: Smile Big Screenshot 1



The information here will not be shared with others under any circumstances, we do not keep the information or use it elsewhere. This screener is simply a personal tool that the user should use to evaluate themselves and their position. This however is not an actual diagnosis, please see your doctor if you feel you sincerely need the help. Please answer the following questions as honestly as possible on a scale of 1-10, 10 being the most truthful and 1 being not at all.

I have little interest or pleasure in the activities that once interested me

1

I have trouble falling asleep, staying asleep, or sleeping too often

1

I often feel tired or have little energy

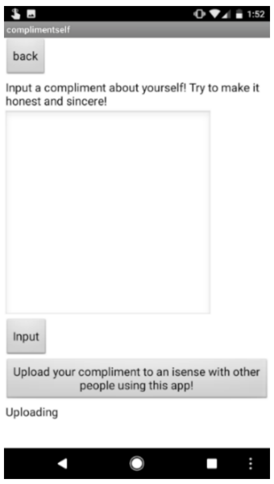
1

I either have a poor appetite or have been overeating

1

I feel bad about myself, I am a failure and have let my family or the people I love down

Figure 49: Smile Big Screenshot 2

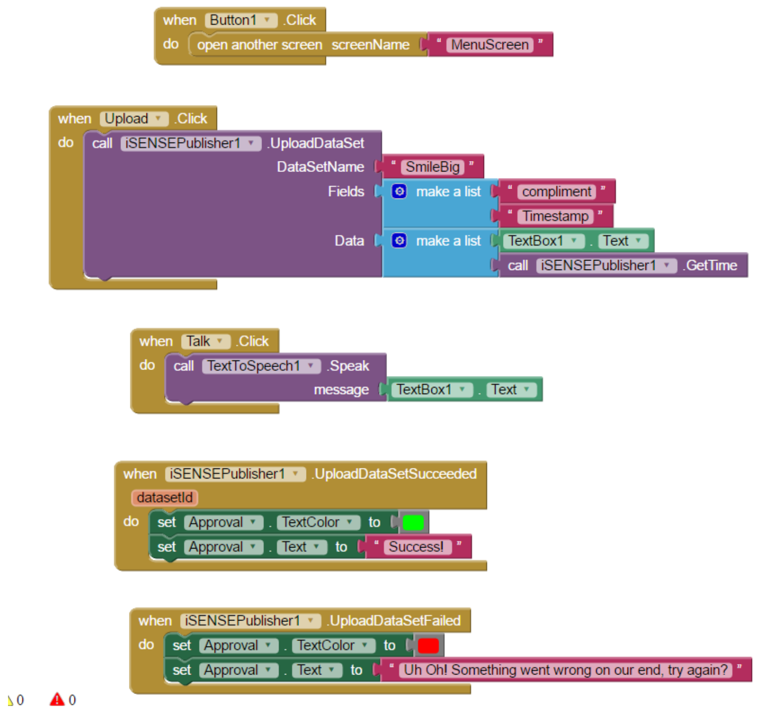


Input a compliment about yourself! Try to make it honest and sincere!

Upload your compliment to an isense with other people using this app!

Uploading

Figure 50: Smile Big Screenshot 3



```
when Button1 .Click
do  open another screen  screenName  " MenuScreen "

when Upload .Click
do  call iSENSEPublisher1 .UploadDataSet
                     DataSetName  " SmileBig "
                     Fields  make a list  " compliment "
                                          " Timestamp "
                     Data    make a list  TextBox1 . Text
                                          call iSENSEPublisher1 .GetTime

when Talk .Click
do  call TextToSpeech1 .Speak
                 message  TextBox1 . Text

when iSENSEPublisher1 .UploadDataSetSucceeded
 datasetId
do  set Approval . TextColor to 🟩
    set Approval . Text to " Success! "

when iSENSEPublisher1 .UploadDataSetFailed
do  set Approval . TextColor to 🟥
    set Approval . Text to " Uh Oh! Something went wrong on our end, try again? "
```

Figure 51: Smile Big AppVis Code

Figure 51 shows part of the code for the screen seen in Figure 50. The app has the user input a self-compliment, then uploads it to an iSENSE project created by the students. The students wanted this to be used so that people with depression can see that others suffer the same, and learn to value themselves.

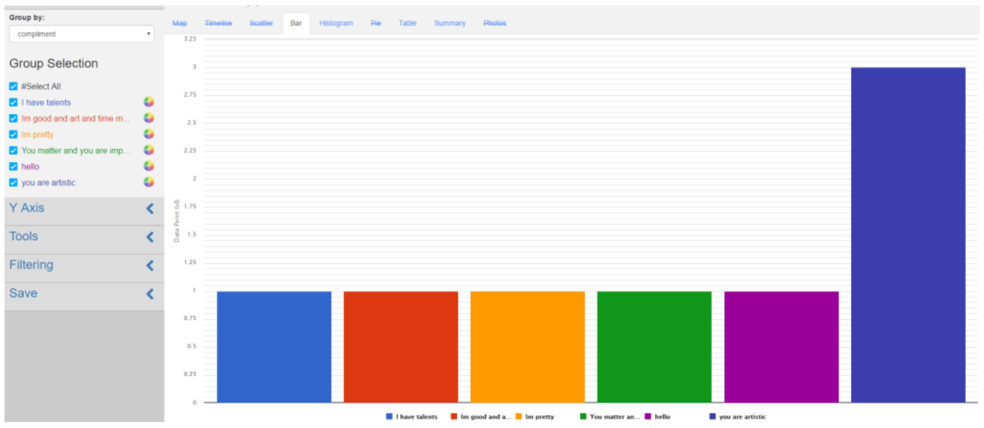


Figure 52: Smile Big Default iSENSE Visualization

Figure 52 shows the default visualization for this app on iSENSE. The students struggled with which visualization would work best to show people’s compliments. Their problem was interesting because they wanted to work with data, but their data were user-designated free responses instead of numeric data, making their data more difficult to visualize.

5.1.3 Head Count

A community partner came in from the park district, and she discussed how her organization is trying to increase the number of people who utilize public parks. One of her duties in her current position is to actually go to the parks and fill out a worksheet of sorts that counts the number of people at the park and what activities they are partaking in. Two students realized that this could easily be a crowdsourced activity built into an app so that one person would not have to do this job.

Because there were so many fields to fill out on the form, the students decided to break up the form over several screens. However, then the students ran into the problem of preserving user-entered data across screens. To rectify this, their app utilizes a TinyDB component to store information locally and uses AppVis to upload the information from the TinyDB.

When one of the students was asked what motivated her to build this app, she replied:

> "[The community partner name] came in and said it was her job to make parks better. She goes to parks and records how many people she sees biking, walking, etc., to see if it's getting good use. We made an app to make that easier."

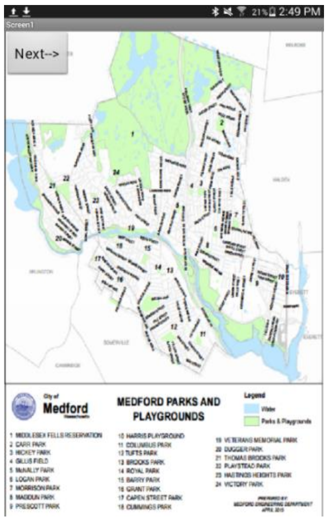
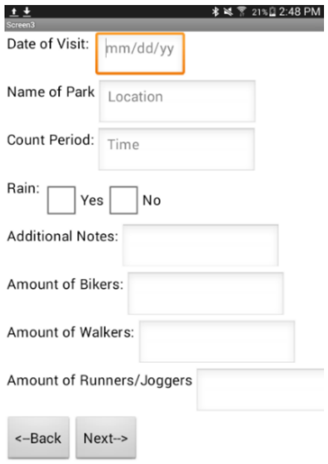


Figure 53: Head Count Screenshot 1



Date of Visit: mm/dd/yy

Name of Park: Location

Count Period: Time

Rain: ☐ Yes ☐ No

Additional Notes:

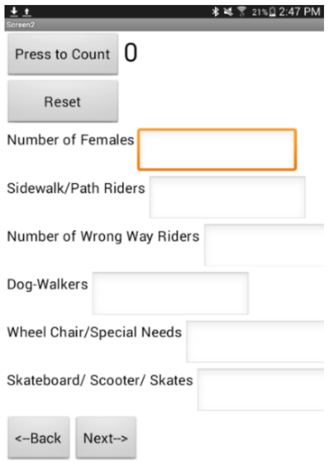
Amount of Bikers:

Amount of Walkers:

Amount of Runners/Joggers

<--Back Next-->

Figure 54: Head Count Screenshot 2



Press to Count 0

Reset

Number of Females

Sidewalk/Path Riders

Number of Wrong Way Riders

Dog-Walkers

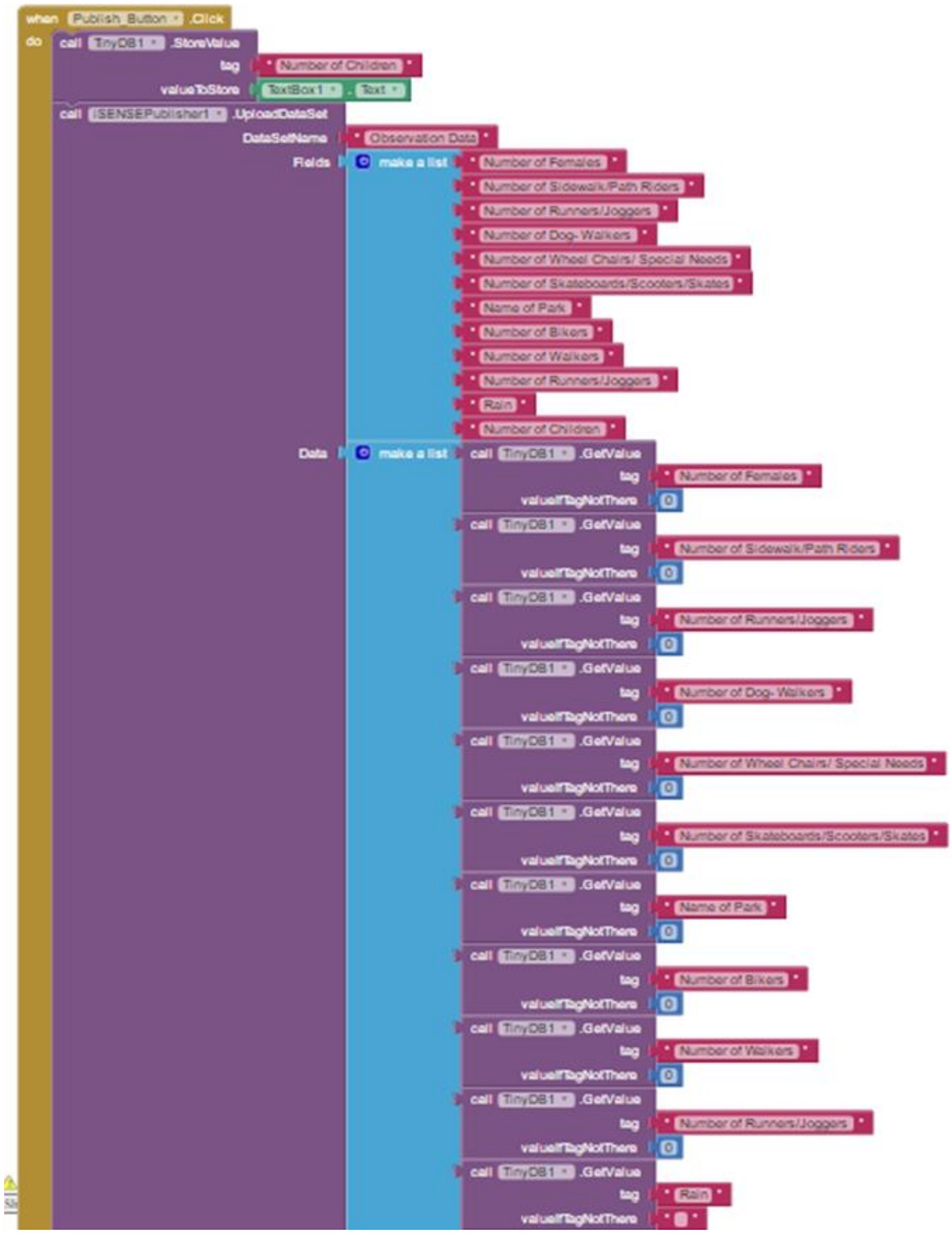
Wheel Chair/Special Needs

Skateboard/ Scooter/ Skates

<--Back Next-->

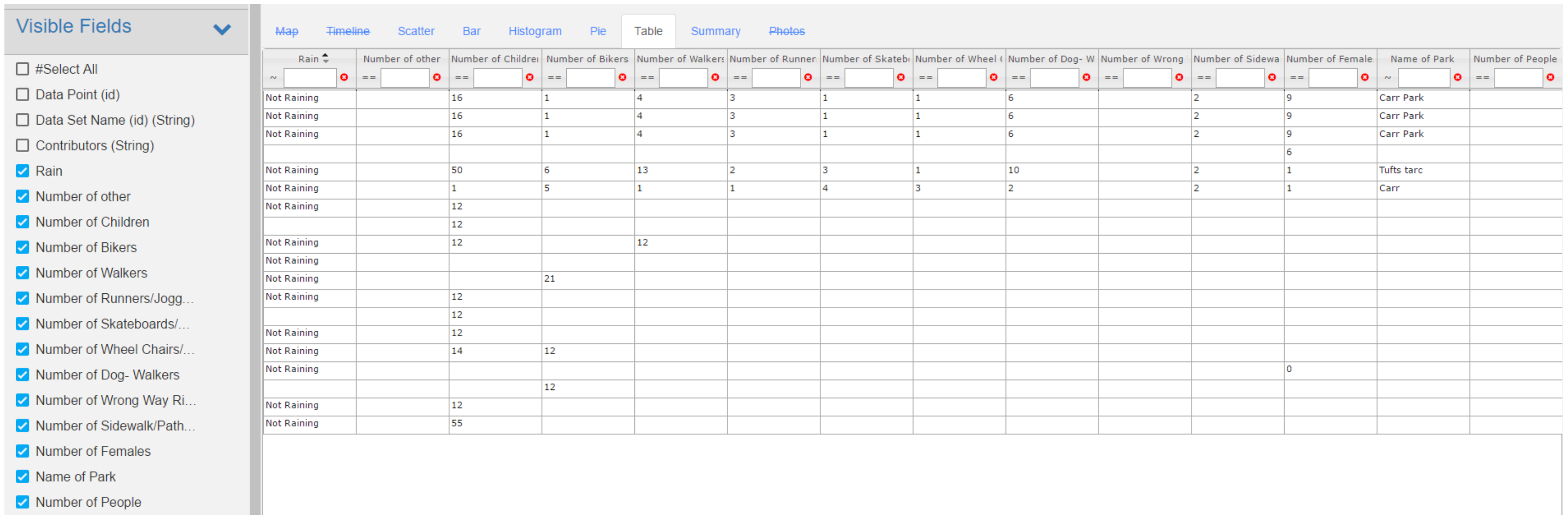
Figure 55: Head Count Screenshot 3

Figures 53, 54, and 55 show the home screen and the two of the user input screens. Figure 56 below shows the upload code for this app. It builds the data list by retrieving all values from the TinyDB component. It also reports zero values if the user did not fill out a field. Finally, Figure 57 shows the default iSENSE visualization for this app with test data filled in. The students chose to visualize as a table so that all data can be seen easily on one screen—had they chose to visualize as a graph (for instance, a histogram), then they would have had to choose which features to graph by, and the students did not want to emphasize any one field of data above another.



```
when Publish_Button .Click
do  call TinyDB1 .StoreValue
                  tag  "Number of Children"
         valueToStore  TextBox1 . Text
    call iSENSEPublisher1 .UploadDataSet
                  DataSetName  "Observation Data"
                  Fields  make a list  "Number of Females"
                                       "Number of Sidewalk/Path Riders"
                                       "Number of Runners/Joggers"
                                       "Number of Dog- Walkers"
                                       "Number of Wheel Chairs/ Special Needs"
                                       "Number of Skateboards/Scooters/Skates"
                                       "Name of Park"
                                       "Number of Bikers"
                                       "Number of Walkers"
                                       "Number of Runners/Joggers"
                                       "Rain"
                                       "Number of Children"
                  Data  make a list  call TinyDB1 .GetValue
                                                        tag  "Number of Females"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Number of Sidewalk/Path Riders"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Number of Runners/Joggers"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Number of Dog- Walkers"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Number of Wheel Chairs/ Special Needs"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Number of Skateboards/Scooters/Skates"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Name of Park"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Number of Bikers"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Number of Walkers"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Number of Runners/Joggers"
                                              valueIfTagNotThere  0
                                     call TinyDB1 .GetValue
                                                        tag  "Rain"
                                              valueIfTagNotThere  " "
```

Figure 56: Head Count AppVis Code



Rain	Number of other	Number of Childre	Number of Bikers	Number of Walkers	Number of Runner	Number of Skateb	Number of Wheel (	Number of Dog- W	Number of Wrong	Number of Sidewa	Number of Female	Name of Park	Number of People
Not Raining		16	1	4	3	1	1	6		2	9	Carr Park	
Not Raining		16	1	4	3	1	1	6		2	9	Carr Park	
Not Raining		16	1	4	3	1	1	6		2	9	Carr Park	
											6		
Not Raining		50	6	13	2	3	1	10		2	1	Tufts tarc	
Not Raining		1	5	1	1	4	3	2		2	1	Carr	
Not Raining		12											
		12											
Not Raining		12		12									
Not Raining													
Not Raining			21										
Not Raining		12											
		12											
Not Raining		12											
Not Raining		14	12										
Not Raining											0		
			12										
Not Raining		12											
Not Raining		55											

Figure 57: Head Count Default iSENSE Visualization

5.2 Overall App Analysis

We analyzed all the apps from Camp 2015 and Camp 2016, and categorized them by themes, intended usage, and complexity.

5.2.1 App Themes

Overall, student apps remained socially-oriented both last year and this year. Out of 38 apps from Camp 2015, three were not community-focused; out of 18 apps from Camp 2016, only one was not community-focused.

Out of the 17 community-focused apps from Camp 2016, eight addressed personal health, six addressed local park issues, 2 addressed local endangered wildlife, and one addressed renewable power. All of these apps directly addressed a specific issue brought up by one or more of the community partners, implying that the students really did see the community partners as the intended customers for their apps.

5.2.2 App Classification

We classified all student apps as Informational, Utility, or Game, where an informational app is aimed at providing the user with information, a game is an app aimed at entertaining the user, and a utility app is designed to be a tool for the user. Figure 58 shows these classifications. Apps may have multiple modalities, so one app may be classified in multiple columns.

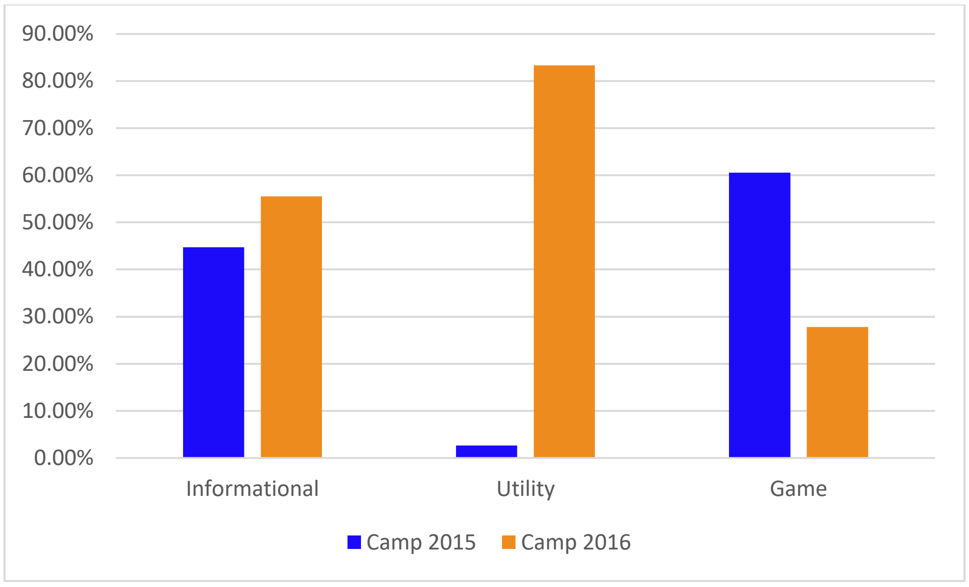


Figure 58: App Classifications - Fewer Games, More Utility

As seen above, the proportion of students who wanted to build informational apps remained similar. However, the proportion of students who built utility apps drastically changed. In the tutorials, we introduced the students to the AppVis components, which we classify as utility components, rather than showing them games as we did last year. This all shows that teaching students with games will lead them to make games, but teaching them with utility concepts will encourage them to make utility-based apps.

Also, Camp 2016 students embraced the idea of multi-modal apps. For example, one app was separated into a kids section and an adults section. The kids section had games, but the adults section had recipes for healthy meals. In Camp 2015, 7.89% of the apps were multi-modal; in Camp 2016, 50% were multi-modal. The students who built games, for the most part, built mini-games into their utility apps.

This supports the research that students will lean towards being interested in using technology to solve problems in their local communities. However, the correct tools need to be provided. Using game-based tutorials is not sufficient to enable the students to think about using App Inventor for making socially relevant utility apps, but maybe for socially themed games. For example, last year, several students made recycling themed games (e.g., tap on the items that are trash), whereas this year, several students made fitness apps that aimed to provide tips for how to lead a healthier lifestyle. These apps were informational but also utility-oriented, as several included daily calorie intake calculators based on a user's weight, gender, and height.

The main takeaway is that it very much matters how a tool is introduced to the students. If it is introduced as a tool for building games, then it will remain as such. However, if it is introduced as a tool for helping the community, then the students will treat it as such. With the two students who built a game, they had already had significant coding experience before the camp, so this was not an introduction for them. They already had an idea of what they wanted to do, and what they thought computer science is about, so our approach was least effective with them.

5.2.3 Sophistication of App Code

In order to measure the sophistication of the students' final apps to see if the students had made comparable apps with and without AppVis, we turned to Sherman and Martin's mobile computational thinking assessment rubric (Sherman & Martin, 2015). We used the rubric to score all of the apps from Camp 2015 (without AppVis) and Camp 2016 (with AppVis). Figure 59 shows the app sophistication scores based on the rubric.

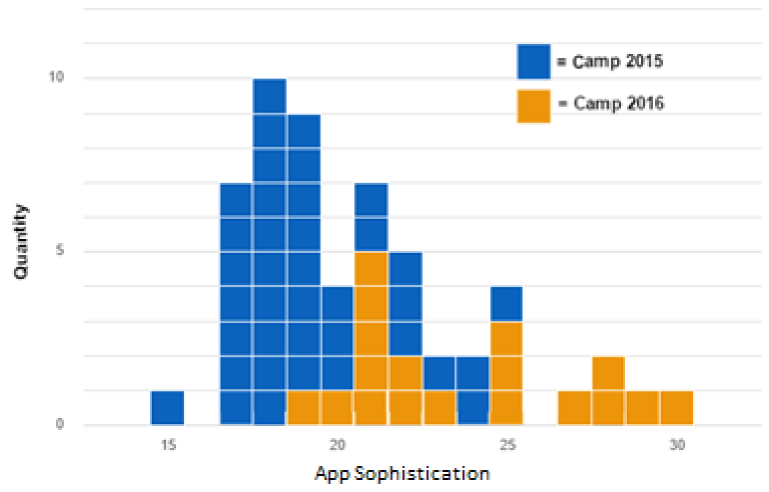


Figure 59: Distribution of App Sophistication by Camp Year

Apps from Camp 2015 had an average score of 19.21, and apps from Camp 2016 scored 23.38. This was largely attributable to students' use of the AppVis components. The iSENSE Publisher component yielded four points in the "data sharing" category according to the rubric, and the iSENSE Viewer yielded two points in "public web services." The average "data sharing" score without AppVis was 1.0, but the average score for the same category is 2.22 with AppVis. Similarly, the average "public web services" score without AppVis was 1.24, but the average score with AppVis is 2.11.

Students in Camp 2016, in general, incorporated data-sharing into their work, while retaining the level of sophistication demonstrated students of Camp 2015. This result shows that the AppVis material was readily accessible to the campers

5.3 Interview Analysis

We conducted semi-structured interviews with 25 students with consent on the end of Day 4 and Day 5, during the stage of finishing final projects. 14 of the 25 students were female. 9 out of the 25 students reported prior experiences with App

Inventor including four alumni of our 2015 summer camp. Another five students learned App Inventor through a school-year class. Two students had some programming experience with Scratch.

We used the same interview protocol as last year's to understand participants' perspectives, experiences and programming practices through engaging them in conversations about their final projects and the design and developing processes (Ni, Sherman, Schilder, & Martin, 2016). The interviews were coded using thematic analysis. Starting with a code list created from prior analysis of interviews with Camp 2015 participants (Ni, Sherman, Schilder, & Martin, 2016), the researchers identified common ideas clustered into major themes on students' attitudes and experiences including addressing community needs, positive App Inventor experiences, and interest in learning more about computer science. Meanwhile, new codes were created and new themes were identified in the analysis.

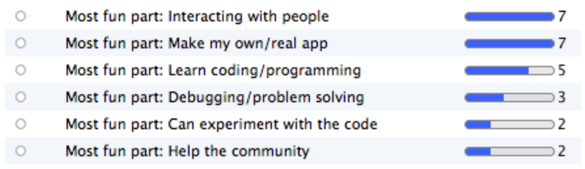
5.3.1 Camp 2016 Findings

Similar to last year's findings, students were able to connect the app ideas with the community partners' visits; they enjoyed the camp and expressed that they would like to learn more about creating apps and more about computer science. Compared to last year's results, we also found that students in Camp 2016 were more excited about sharing and publishing their apps. They saw App Inventor as a vehicle to help community and people. These findings are also consistent with the hypothesis that, overall, students in Camp 2016 felt that they had built "real" apps, as opposed to the students in Camp 2015.

Students also expressed that they enjoyed working with friends, partners, and teachers, and report this aspect as one of the most fun aspects of the camp. When asked about issues in the camp, Camp 2016 interviewees reported more programming-specific challenges or difficulties they encountered in creating apps; Camp 2015 interviewees had reported challenges more focused on technical issues and screens design. These findings were consistent across gender and race, showing that we achieved equivalent results across minority and majority groups.

Per this project's focus, students were guided to create their final apps addressing a specific community need. When asked where their app ideas came from, most students (except 3 out of 25) explicitly connected their ideas with the community partners' visit or identified the needs from their communities. The other three students reported their apps were linked with their personal interests. This supports the app artifacts themselves, all of which explicitly linked to the community partner needs.

Overall, students reported favorable experiences with the summer camp. They saw App Inventor was "fun and cool" and felt proud of the products they created through the camp. All 25 interviewees reported that their camp experience was a positive one. Figure 60 shows what students described as the most fun part of the camp.



○	Most fun part: Interacting with people	7
○	Most fun part: Make my own/real app	7
○	Most fun part: Learn coding/programming	5
○	Most fun part: Debugging/problem solving	3
○	Most fun part: Can experiment with the code	2
○	Most fun part: Help the community	2

Figure 60: Most "Fun" Parts of Camp

For the most part, students' favorite aspects of the camp were the interactions with friends, partners, and teachers, and the ability to make their own apps (or, to make "real" apps). Below are several specific quotes from students explaining what made the camp fun for them:

"I enjoyed most interacting with people and learning it from their point of view and how they use it. The students and the teachers, mostly the teachers."

"[The most fun part is] working as a group, trying to solve the bug of the app and the teamwork of my friends and the teachers. There were things I didn't understand and they helped me a lot. I know a lot more now about App Inventor and apps."

"The most fun part was making your own app, choosing something you want to do, helping the community by doing something they want to do but making things easier for them."

In addition to asking them what the most fun part of the camp was, we also asked them what they were most proud of. The majority of students stated that they were most proud of being able to have a final app completed by the end of the week. One student reported:

"I feel most proud of the app… Even though it's gonna be a prototype, you get that feeling that you're so happy about it and so excited that you want to jump around and scream. The proud part is like, you're making a thing that mostly people don't know. [This is] a new experience in your life."

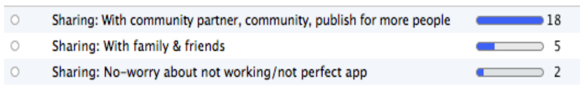
We also asked the students how they would describe App Inventor to their friends. The majority of the students reported that they would recommend it to their friends, and even expressed a desire to teach their friends, as seen in the following quote:

"I would tell them it's about coding and creating apps in your own way. Sometimes when they think of App Inventor they think of apps but not creating them in your own way. I would like to recommend this so they can learn it because most of my friends don't know how to code and I can teach them."

Six students explicitly described App Inventor as a tool to help the community, as seen in the following quote:

"I would recommend it because some people would like to make the world a better place. They can use App Inventor. It's an easier way to show your ideas."

When asked about their plans for sharing final apps, students were excited about publishing their apps to the Google Play Store. This was a new incentive provided in Camp 2016 that was not an option in Camp 2015.



○	Sharing: With community partner, community, publish for more people	18
○	Sharing: With family & friends	5
○	Sharing: No-worry about not working/not perfect app	2

Figure 61: Plans to Share Apps

18 of the 25 students explicitly mentioned they were interested in sharing the app with the community or publishing to the Google Play Store to benefit more people. For example, one student expressed her willingness of showing the app to a community partner and other communities:

> "I want to show the [community partner name]. I want to show her because it's for her basically, but also for other parks and recreation things in other communities too."

The two students who were nervous about sharing the app were at stage of still working on completing the app at the time of interview.

When it came to learning more about computer science, all but one student expressed a desire to continue learning and had future plans within computer science. The one student said:

> "I don't want to be the only one there without a friend and I'll probably get confused a lot and not know what to do."

When specifically asked about their future plans, the other 24 out of 25 students talked about learning more about creating apps, computers, debugging, and other programming languages. Several students mentioned a desire to join a coding club or take computer science courses at school.

We also asked students about the most difficult part of the camp. Students reported varied challenges they encountered in creating apps. Most of the difficulties were programming-specific, such as a misconfiguration of the AppVis components, trouble with TinyDB, or struggles with getting a global variable to update. One student used her trouble with AppVis to explain a problem many beginner programmers face—the need to be specific when writing code (Robins, Rountree, & Rountree, 2003):

> “[The most difficult part is] figuring out what to put in the programming part to send everything to iSENSE. If you don't have the correct labels on iSENSE or this, the whole thing doesn’t work.”

Only four students mentioned non-programming related challenges (organizing pictures/screen layout, finding information from multiple websites, getting non-copyright pictures/resources).

In contrast, the majority of Camp 2015’s participants complained about simple technical problems, such as screen design and WiFi connectivity (Ni, Sherman, Schilder, & Martin, 2016).

5.3.1.6 Girls vs. Boys

When we disaggregated the interview data by gender (14 females and 11 male students interviewed), no significant difference was identified related to the findings presented above. Overall, we saw similar points districted across the two groups showing how they enjoyed the camp, felt proud of their work, and were interested in learning more. We saw a few slightly different points. First, two boys reported that the most enjoyable part was being able to help the community, while two girls reported that the most fun part was that they could “experiment with code”. In terms of interest in computer science, one girl expressed her need of friends’ company in taking more computing courses. Although the same number of boys and girls expressed their interest in publishing the apps, four girls and one boy wanted to share their final apps with their friends and family. When explaining the difficulties they encountered in creating apps, three girls and one boy mentioned non-programming related challenges.

Chapter 6: Conclusions and Future Directions

In Camp 2015, we experienced a similar situation to the one Blikstein describes (Blikstein, 2013). Blikstein's students were engaged and excited by fabricating keychains. Similarly, the Camp 2015 students were engaged and excited by building simple games. These games were enough to impress the students' friends and families, so students never thought to use App Inventor for building apps that were not games. In other words, as has been remarked by a developer in the App Inventor community: "If you show the students Mole Mash, they will build Mole Mash."

We consider Camp 2016 to have been successful, not just in a pedagogical sense, but also in a research sense. This year, during our App Fair, all the students presented their apps, not only to their family and friends, but also to the community partners who had come on the first day to talk about community issues. One community partner this year said, "I wish we could merge all these apps together. That would be the perfect app for our project." Another community partner asked two students if they minded putting the community partner's logo on their app so that it could be identified as being the official app of the partner's initiative. We had no comparable responses last year, before introducing AppVis.

Overall, Camp 2016 had more advanced content available to the students who were interested in going above and beyond, as we wanted to challenge alums returning from last year's camp. We also had a higher instructor-student ratio to better assist all students and alleviate logistical issues (i.e., internet problems). The students had an overwhelmingly positive reaction to the camp—when asked what could be improved, many could not think of any suggestions. The community partners (the end "customers" of the apps) had only positive reactions to the students' work.

Another conclusion we came to is that students will rise to the expectations placed on them. When provided an environment for building tools for social good, students as young as ten years old strive to achieve. Providing a motivation also helps, and we had two motivations: Google Play Store publishing, and the App Fair at which students would present their work to their "customers". Additionally, because we showed a variety of abstract concepts throughout our tutorials, the diversity of student-built apps increased. Nevertheless, because the students are young, it is important to take them outside of the classroom and allow them small breaks, or "recess".

This dissertation has introduced a new tool, AppVis, for engaging students in computer science learning through a data-rich App Inventor environment, with the specific goal of encouraging them to build apps for social good. After using it in a summer camp with middle schoolers and seeing the positive results in both their work as well as their excitement, we could see that it successfully engaged students across minority and majority groups. The students felt as if they had done something constructive that benefited their own communities, and the community partners agreed. We feel confident that AppVis is a tool that can be used more widely to engage students in building apps that gather and share data.

References

A-Ghamdi, S., Al-Rajhi, N., Al-Onaizy, N., & Al-Khalifa, H. (2016). Using App Inventor 2 in a Summer Programming Workshop: Improvements Over Previous Years. *IEEE Global Engineering Education Conference.*

Barba, E., & Chancellor, S. (2015). Tangible Media Approaches to Introductory Computer Science. *ITICSE '15.*

Batsleer, J. (2008). *Informal Learning in Youth Work.* London: Sage Publications Ltd.

Blikstein, P. (2013). Digital Fabrication and 'Making' in Education: The Democratization of Invention. *FabLabs: Of Machines, Makers, and Inventors*.

Finzer, W. (2013). The Data Science Education Dilemma. *Technology Innovations in Statistics Education*.

Grover, S., & Pea, R. (2013). Using a Discourse-Intensive Pedagogy and Android's App Inventor for Introducing Computational Concepts to Middle School Students. *SIGCSE '13.*

Ito, M., Horst, H., Bittanti, M., Boyd, D., Herr Stephenson, B., Lange, P., . . . Robinson, L. (2009). *Living and Learning with New Media: Summary of Findings from the Digital Youth Project.* The John D. and Catherine T. MacArthur Foundation.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., . . . Werner, L. (2011). Computational Thinking for Youth in Practice. *ACM Inroads*, 32-37.

Martin, F., Scriber-MacLean, M., Christy, S., Rudnicki, I., Londhe, R., Manning, C., & Goodman, I. (2011). Reflections on iCODE: Using Web Technology and Hands-On Projects to Engage Urban Youth in Computer Science and Engineering. *Auton Robot*, 265-280.

McCallie, E., Bell, L., & Lohwater, T. (2009). *Many Experts, Many Audiences: Public Engagement with Science and Informal Education.* Washington, DC: Center for Advancement of Informal Science Education.

Michalka, S., Dalphond, J., & Martin, F. (2016). Inquiry Learning with Data and Visualization in the STEM Classroom. *Society for Information Technology & Teacher Education International Conference*, (pp. 2204-2211).

MIT Media Lab. (2016, December). *Seymour A. Papert*. Retrieved from MIT Media Lab: http://media.mit.edu/people/papert

Morelli, R., Limardo, N., de Lanerolle, T., Tamotsu, E., Lake, P., & Uche, C. (2011). Can Android App Inventor Bring Computational Thinking to K-12? *SIGCSE '11.*

Morelli, R., Wolber, D., Rosato, J., Uche, C., & Lake, P. (2014). Mobile Computer Science Principles: A Professional Development Sampler for Teachers. *SIGCSE '14.*

Ni, L., Sherman, M., Schilder, D., & Martin, F. (2016). Computing with a Community Focus: An App Inventor Summer Camp for Middle School Students. *SIGCSE '16.*

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 137-172.

Sherman, M., & Martin, F. (2015). The Assessment of Mobile Computational Thinking. *Consortium for Computing Sciences in Colleges.*

Soares, A., & Martin, N. L. (2014). Teaching Non-Beginner Programmers with App Inventor: Survey Results and Implications. *2014 Proceedings of the Information Systems Educators Conference.* Baltimore, Maryland USA.

Stern, T. (2007). *NetScratch: a networked programming environment for children.* Cambridge: Massachusetts Institute of Technology.

Wolber, D., Abelson, H., & Friedman, M. (2014). Democratizing Computing with App Inventor. *GetMobile*, 53-58.

Xie, B., Shabir, I., & Abelson, H. (2015). Measuring the Usability and Capability of App Inventor to Create Mobile Applications. *SIGPLAN '15.*

Appendix A: IRB Form

IRB AMENDMENT FORM

For IRB Use only: IRB Protocol No.:

Amendment No.:

This form is to be used ONLY for an amendment to a previously approved research project. Complete all sections. If any questions are not applicable, indicate by 'N/A'. Attach a detailed explanation of the reason(s) you are seeking to modify your previously approved research project. Also attach any revised instruments, questionnaires, letters of cooperation, Informed Consent Forms (with current dates and signatures), etc., together with the originally approved version. If submitting a signed form, send by intercampus mail to OIC – Wannalancit 2nd floor, scan & e-mail to irb@uml.edu, or fax to 978-934-6012.

Project Title (of previously approved project): Middle School Pathways in Computer Science

Reason for amendment request: add researcher to project; update student summer camp surveys; add teacher structured interview protocol; update teacher ICF.

Date of initial approval by the IRB:7/7/14

Date of last continuing review approval by the IRB: 7/7/15

The following change is proposed for the above-titled research project:

( ) Principal Investigator has changed. From: to .
( ) Co-PIs involved in project have been changed, added to, or removed
From: to .
(X) Student Investigators or change in Research Personnel have been added (attach training
certification)
Name(s):Farzeen Haranuni, doctoral student. Completed CITI training on 2015-08-22.
( ) Addition or deletion of funding source:
Added:
Deleted:
(X) Addition or deletion of survey instruments, etc. Describe: We have updated pre- and post- instruments for this summer's camp with students, `Student Summer Camp Pre Survey 2016.pdf` and `Student Summer Camp Post Survey 2016.pdf`. The following changes were made: language on technology was clarified (items 2, 3, 4, 8, and 9); options for race/ethnicity were expanded (items 16 and 17). ICFs for this research activity were previously approved January 4, 2016.

We also have a new structured interview protocol for project teachers, `INTERVIEWING TEACHERS ABOUT THEIR CURRICULUM IMPLEMENTATION.pdf`. Teacher interviews will be conducted by telephone.

( ) Addition or deletion of segments of project. Describe:
( ) Change in Title:
New Title:
(All forms that require the modification must also be submitted with the amendment.)
( ) Change in Methodology. Describe:
In some cases, the original protocol must also be submitted (using the track changes tool) to show the proposed change in relation to the original protocol.
( ) Change in number of participants. From: to .
( ) Addition or deletion of cooperating institutions.
Added:
Deleted:
(X) Revised Informed Consent Form, indicate the revision(s), reason for changes and include date of revisions on form (Revised form(s) must also be submitted with the amendment). We updated our teacher informed consent form to include a statement about participation in structured interviews, and to revise the date.

( ) Translated Materials, list all documents translated. (Certificate of Translation must also be signed and submitted).
( ) Other, explain:

Name of PI:Fred Martin	Date: 6/24/2016
PI Signature: **OR** (x) Check here if submitted electronically from the PI's email account.	

Appendix B: AppVis Technical Details

This appendix contains the code for the iSENSE Publisher and Viewer components that were implemented for AppVis and used in the summer camp. Footnotes have been added to provide more information on design decisions. The extension version of the iSENSE Publisher has also been included, and that is the most recent version of the code as of the time of this writing. Because visible components are not able to be implemented as extensions, there is no comparable version of iSENSEViewer.java.

The original AppVis files exist in:

`appinventor/components/src/com/google/appinventor/components/runtime`.

In addition to the files that have been put in full here, there were several other additions:

- Added a .jar library file for the iSENSE API[5]
- Added an iSENSE logo as an image (one small and one large)

[5] http://isenseproject.org/api/v1/docs

- `appinventor/appengine/src/com/google/appinventor/client/Images.java`
    - To include the iSENSE logo into App Inventor
- `appinventor/appengine/src/com/google/appinventor/client/OdeMessages.java`
    - All methods and properties of new components need to have entries put into OdeMessages
- `appinventor/appengine/src/com/google/appinventor/client/TranslationDesignerPallete.java`
    - For adding help strings
- `appinventor/appengine/src/com/google/appinventor/client/editor/simple/components/MockiSENSEViewer.java`
    - Because the iSENSE Viewer is a visible component, there needs to be a corresponding mock component
- `appinventor/appengine/src/com/google/appinventor/client/editor/simple/palette/SimpleComponentDescriptor.java`
    - To translate the mock viewer into the real viewer
- `appinventor/appengine/src/com/google/appinventor/client/youngandroid/YoungAndroidFormUpgrader.java`
    - All components need to have an upgrade path built in for future revisions
- `appinventor/blocklyeditor/src/versioning.js`
    - To guide the version upgrade, if necessary

- `appinventor/components/src/com/google/appinventor/components/common/YaVersion.java`
    - Added version numbers
- Build.xml files
    - App Inventor builds using ant, and these build files were changed to take the iSENSE embedded library into account

This appendix aims to provide some more technical detail and background on the design decisions that went into designing the AppVis components. In each of the following sections, details are provided with line numbers as references. These line numbers will correspond to the full code listings beneath the details.

iSENSEPublisher.java

Note: this is the version of the code that was used in Camp 2016. Because of the issues with instability in the photo uploading, it has been commented out of this version.

Line	**Description**
46	Versioning in App Inventor is traditionally dictated by YaVersion.java and versioning.js. While it is possible to keep track of the version within the component itself (see the Extension version), it is not the standard format.
53	While “isense.jar” is not provided within this dissertation, it is a compiled version of the iSENSE Java API, as mentioned above.
57	ProjectID refers to the numeric ID of the corresponding iSENSE project
58/59	dataSetID and mediaID are the return values for UploadDataSet and UploadDataSetWithPhoto
60	The ContributorKey is the user-designated iSENSE project password
61	In order to restrict users from inundating a project with data, a pending upload queue has been implemented, with the length limit determined in line 67. It is a Linked List of DataObject objects, which are described below.
62	An instantiation of the iSENSE Java API
63	A reference to the handler threat that controls the user interface
82	A “getter” for the iSENSE project ID
87	A “setter” for the iSENSE project ID
94	A “getter” for the contributor key
99	A “setter” for the contributor key

107	UploadDataSet function for uploading to iSENSE. The parameters are the name of the dataset and YailList objects, which are the internal data structures for the lists seen within AI's programming environment.
110	If the fields and data list sizes are not the same, then the data are already wrong, so we can call failure from here.
115	Ensure that our pending queue still has space
121	Create a new "DataObject". This object simply takes the provided parameters, packs them into an object, and puts it in the queue, then kicks off a new instance of an UploadTask (described below).
126	UploadDataSetWithPhoto was removed for the summer camp.
162	Defines a DataObject. There are two constructors, based on whether or not this will be a photo upload.
187	This is beginning of the UploadTask private class, which is an instantiation of an AsyncTask. This allows us to run on the handler thread upon upload completion, but run the upload on a background thread.
190	Beginning of background thread code
193	We check for internet connectivity (a mobile or WiFi connection)
205	Sleep for a second if no internet connection; check for interrupt exceptions.
210	If there is no mobile data option, trying to get information about the mobile connection status will return null.
216	Look at the top of the pending queue to see our next upload
220	Use the API to get the project fields, confirm that they match the fields that were given as a parameter, then put the data into a JSONObject.

241 Make the API call to upload data with the current time, data, project ID, contributor key, and the default contributor name ("AppVis"). The upload call returns an UploadInfo object (defined by the API).

243 Get the dataset id. Check if it was a failed upload or successful.

250 If we have a photo to upload, ensure that it is a valid photo. This code did not apply in this version of the camp.

286 This function does run on the UI handler thread, which means that it does have access to the callback functions.

297 GetDataSetsByField makes a simple API call that gets the datasets by field and returns them as a correctly formatted YailList.

304 GetTime returns a timestamp that is formatted the way iSENSE expects.

312 This returns the number of DataObjects currently in the pending queue.

317 Callback function for successful data upload

322 Callback function for failed data upload

```
1 package com.google.appinventor.components.runtime;
2
3 import java.text.SimpleDateFormat;
4 import java.util.ArrayList;
5 import java.util.Calendar;
6 import java.util.LinkedList;
7 import java.io.File;
8 import java.net.URL;
9
10 import org.json.JSONArray;
11 import org.json.JSONException;
12 import org.json.JSONObject;
13
14 import android.app.Activity;
15 import android.util.Log;
16 import android.os.Handler;
17 import android.content.Context;
18 import android.net.ConnectivityManager;
19 import android.net.NetworkInfo;
```

```java
import android.os.AsyncTask;
import android.net.Uri;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.annotations.UsesLibraries;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.util.AsynchUtil;
import com.google.appinventor.components.runtime.util.YailList;
import com.google.appinventor.components.runtime.util.MediaUtil;

import edu.uml.cs.isense.api.API;
import edu.uml.cs.isense.api.UploadInfo;
import edu.uml.cs.isense.objects.RDataSet;
import edu.uml.cs.isense.objects.RPerson;
import edu.uml.cs.isense.objects.RProjectField;


@DesignerComponent(version = YaVersion.ISENSEPUBLISHER_COMPONENT_VERSION,
    description = "A component that provides a high-level interface to iSENSEProject.org",
    category = ComponentCategory.CONNECTIVITY,
    nonVisible = true,
    iconName = "images/isense.png")
@SimpleObject
@UsesPermissions(permissionNames = "android.permission.INTERNET,android.permission.ACCESS_NETWORK_STATE")
@UsesLibraries(libraries = "isense.jar")

public final class iSENSEPublisher extends AndroidNonvisibleComponent implements Component {

  private int ProjectID;
  private int dataSetID = -1;
  private int mediaID = -1;
  private String ContributorKey;
  private LinkedList<DataObject> pending;
  private final API api;
  private final Handler androidUIHandler;
  private static Activity activity;

  private final String CONTRIBUTORNAME = "AppVis";
  private final int QUEUEDEPTH = 50;

  public iSENSEPublisher(ComponentContainer container) {
    super(container.$form());
    Log.i("iSENSE", "Starting? " + container.toString());
    api = API.getInstance();
    ProjectID(-1);
    ContributorKey("");
    pending = new LinkedList<DataObject>();
    androidUIHandler = new Handler();
    activity = container.$context();
  }

  // Block Properties
  // ProjectID
  @SimpleProperty(description = "iSENSE Project ID", category = PropertyCategory.BEHAVIOR)
    public int ProjectID() {
      return ProjectID;
    }

  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue = "")
    @SimpleProperty(description = "iSENSE Project ID", category = PropertyCategory.BEHAVIOR)
    public void ProjectID(int ProjectID) {
      this.ProjectID = ProjectID;
    }

  // Contributor Key
  @SimpleProperty(description = "iSENSE Contributor Key", category = PropertyCategory.BEHAVIOR)
    public String ContributorKey() {
      return ContributorKey;
    }

  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue = "")
    @SimpleProperty(description = "iSENSE Contributor Key", category = PropertyCategory.BEHAVIOR)
    public void ContributorKey(String ContributorKey) {
      this.ContributorKey = ContributorKey;
    }

  // Block Functions
  // Upload Data Set in Background
  @SimpleFunction(description = "Upload Data Set to iSENSE")
    public void UploadDataSet(final String DataSetName, final YailList Fields, final YailList Data) {
      // ensure that the lists are the same size
      if (Fields.size() != Data.size()) {
        UploadDataSetFailed();
        return;
      }
      // A simple throttle if too much data is being thrown at the upload queue
      if (pending.size() > QUEUEDEPTH) {
        Log.i("iSENSE", "Too many items in upload queue!");
        UploadDataSetFailed();
        return;
      }
      // Create new "DataObject" and add to upload queue
      DataObject dob = new DataObject(DataSetName, Fields, Data);
      pending.add(dob);
      new UploadTask().execute();
    }

  // Upload Dataset With Photo
 /* @SimpleFunction(description = "Uploads a dataset and a photo")
    public void UploadDataSetWithPhoto(final String DataSetName, final YailList Fields, final YailList Data, final String Photo) {
      // ensure that the lists are the same size
      if (Fields.size() != Data.size()) {
        UploadDataSetFailed();
        return;
      }
      // A simple throttle if too much data is being thrown at the upload queue
      if (pending.size() > QUEUEDEPTH) {
        Log.i("iSENSE", "Too many items in upload queue!");
        UploadDataSetFailed();
        return;
      }
      // Validate photo
      String path = "";
      String[] pathtokens = Photo.split("/");
      // If camera photo
      if (pathtokens[0].equals("file:")) {
        try {
          path = new File(new URL(Photo).toURI()).getAbsolutePath();
        } catch (Exception e) {
          Log.e("iSENSE", "Malformed URL or URI!");
        }
      } else { // Assets photo
        path = "/sdcard/AppInventor/assets/" + Photo;
      }

      // Create new "DataObject" and add it to the upload queue
      DataObject dob = new DataObject(DataSetName, Fields, Data, path);
      pending.add(dob);
      new UploadTask().execute();

    }
*/
  // Private class that gives us a data structure with info for uploading a dataset
  class DataObject {

    String name;
    YailList fields;
    YailList data;
    String path;

```

```java
    // Normal dataset
    DataObject(String name, YailList fields, YailList data) {
      this.name = name;
      this.fields = fields;
      this.data = data;
      this.path = "";
    }

    // Dataset with photo
    DataObject(String name, YailList fields, YailList data, String path) {
      this.name = name;
      this.fields = fields;
      this.data = data;
      this.path = path;
    }
  }

  // Private asynchronous task class that allows background uploads
  private class UploadTask extends AsyncTask<Void, Void, Integer> {

    // This is what actually runs in the background thread, so it's safe to block
    protected Integer doInBackground(Void... v) {

      // Sleep while we don't have a wifi connection or a mobile connection
      ConnectivityManager cm = (ConnectivityManager) activity.getSystemService(Context.CONNECTIVITY_SERVICE);

      boolean wifi = cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI).isConnected();
      boolean mobi = false;

      if (cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE) != null) {
        mobi = cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE).isConnected();
      }

      while (!(wifi||mobi)) {
        try {
          Log.i("iSENSE", "No internet connection; sleeping for one second");
          Thread.sleep(1000);
        } catch (InterruptedException e) {
          Log.e("iSENSE", "Thread Interrupted!");
        }
        wifi = cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI).isConnected();
        if (cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE) != null) {
          mobi = cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE).isConnected();
        }
      }

      // Active internet connection detected; proceed with upload
      DataObject dob = pending.peek();
      UploadInfo uInfo = new UploadInfo();

      // Get fields from project
      ArrayList<RProjectField> projectFields =
api.getProjectFields(ProjectID);
      JSONObject jData = new JSONObject();
      for (int i = 0; i < dob.fields.size(); i++) {
        for (int j = 0; j < projectFields.size(); j++) {
          if (dob.fields.get(i +
1).equals(projectFields.get(j).name)) {
            try {
              String sdata = dob.data.get(i + 1).toString();
              jData.put("" + projectFields.get(j).field_id, new
JSONArray().put(sdata));
            } catch (JSONException e) {
              UploadDataSetFailed();
              e.printStackTrace();
              return -1;
            }
          }
        }
      }

      // login with contributor key
      Calendar cal = Calendar.getInstance();
      SimpleDateFormat sdf = new SimpleDateFormat("MMMM dd, yyyy
HH:mm:ss aaa");
      String date = " - " + sdf.format(cal.getTime()).toString();
      uInfo = api.uploadDataSet(ProjectID, jData, dob.name + date,
ContributorKey, CONTRIBUTORNAME);

      int dataSetId = uInfo.dataSetId;
      Log.i("iSENSE", "JSON Upload: " + jData.toString());
      Log.i("iSENSE", "Dataset ID: " + dataSetId);
      if (dataSetId == -1) {
        return -1;
      }

      // do we have a photo to upload?
      if (!dob.path.equals("")) {
        File pic = new File(dob.path);
        if (!pic.exists()) {
          Log.e("iSENSE", "picture does not exist!");
          return -1;
        }
        String[] splt = pic.getName().split("[.]");
        if (splt[0].length() > 20) {
          String newname = splt[0].substring(0,20) + "." + splt[1];
          String[] pathsplit = dob.path.split("/");
          String newpath = "";
          for (int i = 0; i < pathsplit.length - 1; i++) {
            newpath += pathsplit[i] + "/";
          }
          File temp = new File(newpath + newname);
          dob.path = newpath + newname;
          if (!pic.renameTo(temp)) {
            Log.i("iSENSE", "Filename too long: " + pic.getName());
            return -1;
          }
          pic = temp;
        }
        pic.setReadable(true);
        Log.i("iSENSE", "Trying to upload: " + dob.path);
        uInfo = api.uploadMedia(ProjectID, pic, API.TargetType.DATA_SET, ContributorKey, CONTRIBUTORNAME);
        int mediaID = uInfo.mediaId;
        Log.i("iSENSE", "MediaID: " + mediaID);
        if (mediaID == -1) {
          return -1;
        }
      }
      return dataSetId;
    }

    // After background thread execution, UI handler runs this
    protected void onPostExecute(Integer result) {
      DataObject dob = pending.remove();
      if (result == -1) {
        UploadDataSetFailed();
      } else {
        UploadDataSetSucceeded(result);
      }
    }
  }

  // Get Dataset By Field
  @SimpleFunction(description = "Get the Data Sets for the current project")
    public YailList GetDataSetsByField(final String Field) {
      ArrayList<String> result = api.getDataSetsByField(ProjectID, Field);
      return YailList.makeList(result);
    }

  // Get Time (formatted for iSENSE Upload)
  @SimpleFunction(description = "Gets the current time. It is formatted correctly for iSENSE")
    public String GetTime() {
      Calendar cal = Calendar.getInstance();
      SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZ");
      return sdf.format(cal.getTime()).toString();
    }

  // Get Number of Pending Uploads (Advanced Feature)
  @SimpleFunction(description = "Gets number of pending background uploads. Advanced feature.")
    public int GetNumberPendingUploads() {
      return pending.size();
    }

  @SimpleEvent(description = "iSENSE Upload Data Set Succeeded")
```

```
    public void UploadDataSetSucceeded(int datasetId) {
      EventDispatcher.dispatchEvent(this, "UploadDataSetSucceeded",
datasetId);
    }

  @SimpleEvent(description = "iSENSE Upload Data Set Failed")
    public void UploadDataSetFailed() {
      EventDispatcher.dispatchEvent(this, "UploadDataSetFailed");
    }

}
```

iSENSEViewer.java

Line	**Description**
33	Versioning in App Inventor is traditionally dictated by YaVersion.java and versioning.js.
44	A WebView is a built in Android object that provides some of the core functionality
47	String constants are defined here to build up the visualization URL
48	Presentation mode is for full-screen view
49	Embedded mode is for a control taskbar to appear above the visualization
69	Defines the onTouch event handler
116	Project ID “getter”
121	Project ID “setter”
129	Presentation Mode “getter”
133	Presentation Mode “setter”; reloads viewer in case mode has changed
145	Gets current visualization URL
162	Gets title of current page
170	Refreshes the page
175	An internal function for resetting the webView client

```
1 package com.google.appinventor.components.runtime;
2
3 import android.content.Context;
4 import android.webkit.JavascriptInterface;
5 import
com.google.appinventor.components.annotations.DesignerComponent;
6 import
com.google.appinventor.components.annotations.DesignerProperty;
7 import
com.google.appinventor.components.annotations.PropertyCategory;
```

```java
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;

import com.google.appinventor.components.runtime.util.FroyoUtil;
import com.google.appinventor.components.runtime.util.SdkLevel;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;

import android.view.MotionEvent;
import android.view.View;
import android.webkit.WebView;
import android.webkit.WebViewClient;

/**
 * Component for displaying iSENSE Visualizations
 *
 */

@DesignerComponent(version = YaVersion.ISENSEVIEWER_COMPONENT_VERSION,
    category = ComponentCategory.USERINTERFACE,
    description = "Component for viewing iSENSE visualizations. " +
    "The user provides a project ID and whether the vis should appear " +
    "in presentation mode (or embedded mode). This is intended to work " +
    "only with isenseproject.org.")

@SimpleObject
@UsesPermissions(permissionNames = "android.permission.INTERNET")
public final class iSENSEViewer extends AndroidViewComponent {

  private final WebView webview;

  // To build iSENSE URL
  private String baseUrl = "https://isenseproject.org/projects/";
  private String preSuffix = "/data_sets?presentation=true";
  private String embSuffix = "/data_sets?embed=true";
  private int projectID = 5;
  private boolean presentationMode = true;

  /**
   * Creates a new iSENSEViewer component.
   *
   * @param container  container the component will be placed in
   */
  public iSENSEViewer(ComponentContainer container) {
    super(container);

    webview = new WebView(container.$context());
    resetWebViewClient();
    webview.getSettings().setJavaScriptEnabled(true);
    webview.setFocusable(true);
    webview.getSettings().setBuiltInZoomControls(true);

    container.$add(this);

    webview.setOnTouchListener(new View.OnTouchListener() {
      @Override
      public boolean onTouch(View v, MotionEvent event) {
        switch (event.getAction()) {
          case MotionEvent.ACTION_DOWN:
          case MotionEvent.ACTION_UP:
            if (!v.hasFocus()) {
              v.requestFocus();
            }
            break;
        }
        return false;
      }
    });

    // set the initial default properties.  Height and Width
    // will be fill-parent, which will be the default for the web viewer.

    Width(LENGTH_FILL_PARENT);
    Height(LENGTH_FILL_PARENT);
  }

  @Override
    public View getView() {
      return webview;
    }

  // Components don't normally override Width and Height, but we do it here so that
  // the automatic width and height will be fill parent.
  @Override
    @SimpleProperty()
     public void Width(int width) {
       if (width == LENGTH_PREFERRED) {
         width = LENGTH_FILL_PARENT;
       }
       super.Width(width);
     }

   @Override
     @SimpleProperty()
     public void Height(int height) {
       if (height == LENGTH_PREFERRED) {
         height = LENGTH_FILL_PARENT;
       }
       super.Height(height);
     }

   @SimpleProperty(description = "iSENSE Project ID", category =
 PropertyCategory.BEHAVIOR)
     public int ProjectID() {
       return projectID;
     }

   @DesignerProperty(editorType =
 PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue = "")
     @SimpleProperty(description = "iSENSE Project ID", category =
 PropertyCategory.BEHAVIOR)
     public void ProjectID(int ProjectID) {
       this.projectID = ProjectID;
       webview.loadUrl(URL());
     }

   @SimpleProperty(description = "Presentation mode", category =
 PropertyCategory.BEHAVIOR)
     public boolean PresentationMode() {
       return presentationMode;
     }

   @DesignerProperty(editorType =
 PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defaultValue = "True")
     @SimpleProperty(description = "Set presentation mode on or
 off", category = PropertyCategory.BEHAVIOR)
     public void PresentationMode(boolean status) {
       this.presentationMode = status;
       webview.loadUrl(URL());
     }

   /**
    * Returns the URL of the page the iSENSEViewer should load
    *
    * @return URL of the page the iSENSEViewer should load
    */
   @SimpleProperty(description="Current URL", category =
 PropertyCategory.BEHAVIOR)
     public String URL() {
       if (presentationMode) {
         return baseUrl + projectID + preSuffix;
       } else {
         return baseUrl + projectID + embSuffix;
       }
     }

   /**
    * Returns the title of the page currently being viewed
    *
    * @return title of the page being viewed
    */
   @SimpleProperty(
       description = "Title of the page currently viewed",
       category = PropertyCategory.BEHAVIOR)
     public String CurrentPageTitle() {
       return (webview.getTitle() == null) ? "" :
 webview.getTitle();
```

```
    }


  /**
   * Refresh the current visualization
   */
  @SimpleFunction(description = "Refresh the visualization")
    public void Refresh() {
      webview.loadUrl(URL());
    }

  private void resetWebViewClient() {
    boolean ignoreSslErrors = false;
    boolean followLinks = false;
    if (SdkLevel.getLevel() >= SdkLevel.LEVEL_FROYO) {
webview.setWebViewClient(FroyoUtil.getWebViewClient(ignoreSslErrors,
followLinks, container.$form(), this));
    } else {
      webview.setWebViewClient(new WebViewClient());
    }
  }
}
```

iSENSEPublisher.java (Extension Version)

Note: Many of these notes are very similar to the above iSENSEPublisher notes, so these notes are not as fine-grained.

Line	Description
1	In an AI Extension, components need to have a custom package that is different from the standard component package.
23	Because this component is not in the components package, more imports are required.
47	Because in an extension, we no longer have access to changing files such as YaVersion.java, the extension version is a variable within the component itself.
52	Because we can't have a photo within the images assets of App Inventor for an extension, there is a link to an icon.
55	As of the writing of this dissertation, compiling embedded jar files into an extension is not implemented correctly in App Inventor, and, while the extension works, there are issues on the build server side of App Inventor and building apk files within App Inventor fails.
116	UploadDataSetWithPhoto is stable and fully implemented in this version. This function first validates the photo as either an assets photo or a camera photo (the two possibilities), creates a DataObject, puts it in the pending queue, then kicks off an UploadTask thread.

252 If the photo path field is not empty, then there is a photo to upload. This uses the uploadMedia call of the API to upload the photo to the dataset we just uploaded (using the dataset ID we just received).

300 Because the iSENSE Viewer is not available as an extension (visible components cannot currently be implemented as extensions), GetVisURL provides the URL of the default visualization in presentation mode.

306 Because the iSENSE Viewer is not available as an extension (visible components cannot currently be implemented as extensions), GetVisWithControlsURL provides the URL of the default visualization in embedded mode.

```
1 package edu.uml.cs.isense;
2
3 import java.text.SimpleDateFormat;
4 import java.util.ArrayList;
5 import java.util.Calendar;
6 import java.util.LinkedList;
7 import java.io.File;
8 import java.net.URL;
9
10 import org.json.JSONArray;
11 import org.json.JSONException;
12 import org.json.JSONObject;
13
14 import android.app.Activity;
15 import android.util.Log;
16 import android.os.Handler;
17 import android.content.Context;
18 import android.net.ConnectivityManager;
19 import android.net.NetworkInfo;
20 import android.os.AsyncTask;
21 import android.net.Uri;
22
23 import
com.google.appinventor.components.annotations.DesignerComponent;
24 import
com.google.appinventor.components.annotations.DesignerProperty;
25 import
com.google.appinventor.components.annotations.PropertyCategory;
26 import com.google.appinventor.components.annotations.SimpleEvent;
```

```
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.annotations.UsesLibraries;
import
com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import
com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.runtime.util.YailList;
import com.google.appinventor.components.runtime.Component;
import
com.google.appinventor.components.runtime.AndroidNonvisibleComponent;
import com.google.appinventor.components.runtime.ComponentContainer;
import com.google.appinventor.components.runtime.EventDispatcher;

import edu.uml.cs.isense.api.API;
import edu.uml.cs.isense.api.UploadInfo;
import edu.uml.cs.isense.objects.RDataSet;
import edu.uml.cs.isense.objects.RPerson;
import edu.uml.cs.isense.objects.RProjectField;


@DesignerComponent(version = iSENSEPublisher.VERSION,
    description = "A component that provides a high-level interface
to iSENSEProject.org",
    category = ComponentCategory.EXTENSION,
    nonVisible = true,
    //iconName = "images/extension.png")
    iconName =
"https://raw.githubusercontent.com/farxinu/appinventor-
sources/master/appinventor/appengine/src/com/google/appinventor/images/
isense.png")
@SimpleObject(external = true)
@UsesPermissions(permissionNames =
"android.permission.INTERNET,android.permission.ACCESS_NETWORK_STATE")
@UsesLibraries(libraries = "isense.jar")

public final class iSENSEPublisher extends
AndroidNonvisibleComponent implements Component {

  public static final int VERSION = 1;
  private static final String CONTRIBUTORNAME = "AppVis";
  private static final int QUEUEDEPTH = 50;

  private int ProjectID;
  private String ContributorKey;
  private LinkedList<DataObject> pending;
  private final API api;
  private static Activity activity;


  public iSENSEPublisher(ComponentContainer container) {
    super(container.$form());
    Log.i("iSENSE", "Starting? " + container.toString());
    api = API.getInstance();
    ProjectID(-1);
```

```java
    ContributorKey("");
    pending = new LinkedList<DataObject>();
    activity = container.$context();
  }

  // Block Properties
  // ProjectID
  @SimpleProperty(description = "iSENSE Project ID", category = PropertyCategory.BEHAVIOR)
    public int ProjectID() {
      return ProjectID;
    }

  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue = "")
    @SimpleProperty(description = "iSENSE Project ID", category = PropertyCategory.BEHAVIOR)
    public void ProjectID(int ProjectID) {
      this.ProjectID = ProjectID;
    }

  // Contributor Key
  @SimpleProperty(description = "iSENSE Contributor Key", category = PropertyCategory.BEHAVIOR)
    public String ContributorKey() {
      return ContributorKey;
    }

  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING, defaultValue = "")
    @SimpleProperty(description = "iSENSE Contributor Key", category = PropertyCategory.BEHAVIOR)
    public void ContributorKey(String ContributorKey) {
      this.ContributorKey = ContributorKey;
    }

  // Block Functions
  // Upload Data Set in Background
  @SimpleFunction(description = "Upload Data Set to iSENSE")
    public void UploadDataSet(final String DataSetName, final YailList Fields, final YailList Data) {
      // Create new "DataObject" and add to upload queue
      DataObject dob = new DataObject(DataSetName, Fields, Data);
      pending.add(dob);
      new UploadTask().execute();
    }

  // Upload Dataset With Photo
  @SimpleFunction(description = "Uploads a dataset and a photo")
    public void UploadDataSetWithPhoto(final String DataSetName, final YailList Fields, final YailList Data, final String Photo) {

      // Validate photo
      String path = "";
      String[] pathtokens = Photo.split("/");
      // If camera photo
      if (pathtokens[0].equals("file:")) {
        try {
          path = new File(new URL(Photo).toURI()).getAbsolutePath();
        } catch (Exception e) {
          Log.e("iSENSE", "Malformed URL or URI!");
          UploadDataSetFailed();
          return;
        }
      } else { // Assets photo
        path = "/sdcard/AppInventor/assets/" + Photo;
      }

      // Ensure photo exists
      File pic = new File(path);
      if (!pic.exists()) {
        Log.e("iSENSE", "picture does not exist!");
        UploadDataSetFailed();
        return;
      }

      // Create new "DataObject" and add it to the upload queue
      DataObject dob = new DataObject(DataSetName, Fields, Data, path);
      pending.add(dob);
      new UploadTask().execute();
    }

  // Private class that gives us a data structure with info for uploading a dataset
  class DataObject {

    String name;
    YailList fields;
    YailList data;
    String path;

    // Normal dataset
    DataObject(String name, YailList fields, YailList data) {
      this.name = name;
      this.fields = fields;
      this.data = data;
      this.path = "";
    }

    // Dataset with photo
    DataObject(String name, YailList fields, YailList data, String path) {
      this.name = name;
      this.fields = fields;
      this.data = data;
      this.path = path;
    }
  }

  // Private asynchronous task class that allows background uploads
  private class UploadTask extends AsyncTask<Void, Void, Integer> {

```

```
  // This is what actually runs in the background thread, so it's safe to block
  protected Integer doInBackground(Void... v) {

    DataObject dob = pending.peek();
    // ensure that the lists are the same size
    if (dob.fields.size() != dob.data.size()) {
      Log.e("iSENSE", "Input lists are not the same size!");
      return -1;
    }

    // A simple throttle if too much data is being thrown at the upload queue
    if (pending.size() > QUEUEDEPTH) {
      Log.e("iSENSE", "Too many items in upload queue!");
      return -1;
    }

    // Sleep while we don't have a wifi connection or a mobile connection
    ConnectivityManager cm = (ConnectivityManager) activity.getSystemService(Context.CONNECTIVITY_SERVICE);

    boolean wifi = cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI).isConnected();
    boolean mobi = false;

    if (cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE) != null) {
      mobi = cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE).isConnected();
    }

    while (!(wifi||mobi)) {
      try {
        Log.i("iSENSE", "No internet connection; sleeping for one second");
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        Log.e("iSENSE", "Thread Interrupted!");
      }
      wifi = cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI).isConnected();
      if (cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE) != null) {
        mobi = cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE).isConnected();
      }
    }

    // Active internet connection detected; proceed with upload
    UploadInfo uInfo = new UploadInfo();

    // Get fields from project
    ArrayList<RProjectField> projectFields = api.getProjectFields(ProjectID);
    JSONObject jData = new JSONObject();
```

```
      for (int i = 0; i < dob.fields.size(); i++) {
        for (int j = 0; j < projectFields.size(); j++) {
          if (dob.fields.get(i + 1).equals(projectFields.get(j).name)) {
            try {
              String sdata = dob.data.get(i + 1).toString();
              jData.put("" + projectFields.get(j).field_id, new JSONArray().put(sdata));
            } catch (JSONException e) {
              UploadDataSetFailed();
              e.printStackTrace();
              return -1;
            }
          }
        }
      }

      // login with contributor key
      Calendar cal = Calendar.getInstance();
      SimpleDateFormat sdf = new SimpleDateFormat("MMMM dd, yyyy HH:mm:ss aaa");
      String date = " - " + sdf.format(cal.getTime()).toString();
      uInfo = api.uploadDataSet(ProjectID, jData, dob.name + date, ContributorKey, CONTRIBUTORNAME);

      int dataSetId = uInfo.dataSetId;
      Log.i("iSENSE", "JSON Upload: " + jData.toString());
      Log.i("iSENSE", "Dataset ID: " + dataSetId);
      if (dataSetId == -1) {
        Log.e("iSENSE", "Upload failed! Check your contributor key and project ID.");
        return -1;
      }

      // do we have a photo to upload?
      if (!dob.path.equals("")) {
        File pic = new File(dob.path);
        pic.setReadable(true);
        Log.i("iSENSE", "Trying to upload: " + dob.path);
        uInfo = api.uploadMedia(dataSetId, pic, API.TargetType.DATA_SET, ContributorKey, CONTRIBUTORNAME);
        int mediaID = uInfo.mediaId;
        Log.i("iSENSE", "MediaID: " + mediaID);
        if (mediaID == -1) {
          Log.e("iSENSE", "Media upload failed. Is it a valid picture?");
          return -1;
        }
      }
      return dataSetId;
    }

    // After background thread execution, UI handler runs this
    protected void onPostExecute(Integer result) {
      DataObject dob = pending.remove();
      if (result == -1) {
        UploadDataSetFailed();
```

```java
      } else {
        UploadDataSetSucceeded(result);
      }
    }
  }

  // Get Dataset By Field
  @SimpleFunction(description = "Get the Data Sets for the current project")
    public YailList GetDataSetsByField(final String Field) {
      ArrayList<String> result = api.getDataSetsByField(ProjectID, Field);
      return YailList.makeList(result);
    }

  // Get Time (formatted for iSENSE Upload)
  @SimpleFunction(description = "Gets the current time. It is formatted correctly for iSENSE")
    public String GetTime() {
      Calendar cal = Calendar.getInstance();
      SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZ");
      return sdf.format(cal.getTime()).toString();
    }

  // Get Number of Pending Uploads (Advanced Feature)
  @SimpleFunction(description = "Gets number of pending background uploads. Advanced feature.")
    public int GetNumberPendingUploads() {
      return pending.size();
    }

  // Get visualization url for this project
  @SimpleFunction(description = "Gets URL for project visualization in simple fullscreen format.")
    public String GetVisURL() {
      return "https://isenseproject.org/projects/" + ProjectID + "/data_sets?presentation=true";
    }

  // Get visualization url with controls for this project
  @SimpleFunction(description = "Gets URL for project visualization with controls onscreen.")
    public String GetVisWithControlsURL() {
      return "https://isenseproject.org/projects/" + ProjectID + "/data_sets?embed=true";
    }

  @SimpleEvent(description = "iSENSE Upload Data Set Succeeded")
    public void UploadDataSetSucceeded(int datasetId) {
      EventDispatcher.dispatchEvent(this, "UploadDataSetSucceeded", datasetId);
    }

  @SimpleEvent(description = "iSENSE Upload Data Set Failed")
    public void UploadDataSetFailed() {
      EventDispatcher.dispatchEvent(this, "UploadDataSetFailed");
    }

}
```

Biographical Sketch of Author

Farzeen Harunani received her Bachelors of Science from Marquette University (Milwaukee, Wisconsin) in May 2015, with a primary degree in Computer Engineering, a secondary degree in Computer Science, and a minor in Mathematics. She was an intern and then a co-op at UTC Aerospace Systems (formerly Hamilton Sundstrand, in Rockford, Illinois) during her undergraduate career, and worked as a student research assistant under Dr. Dennis Brylow. As a student research assistant, her role involved developing educational tools on embedded platforms. Her senior project involved developing a less costly alternative for the Exploring Computer Science course's final module. She began at University of Massachusetts Lowell in August 2016 to continue researching computer science education under Dr. Fred Martin.

During her undergraduate and graduate career, Farzeen has volunteered for several university outreach programs, Girls Who Code, the Girl Scouts' Code Girl, and Compassus Hospice. She co-organized several programming competitions for high schoolers with Marquette University’s Association for Computing Machinery chapter, and started a tutoring program for computing classes as president of the Marquette chapter of Upsilon Pi Epsilon.