1. File Management

1.1 File Component

File component is a non-visible component for storing and retrieving files. Use this component to write or read files on your device. The default behavior is to write files to the private data directory associated with your App. The Companion writes files to /sdcard/AppInventor/data for easy debugging. If the file path starts with a slash (/), then the file is created relative to /sdcard. For example, writing a file to /myFile.txt will write the file in /sdcard/myFile.txt.

1.1.1 Save File

Saves text to a file. If the filename begins with a slash (/) the file is written to the sdcard (for example, writing to /myFile.txt will write the file to /sdcard/myFile.txt). If the filename does not start with a slash, it will be written in the program's private data directory where it will not be accessible to other programs on the phone. There is a special exception for the AI Companion where these files are written to /sdcard/AppInventor/data to facilitate debugging. Note that this block will overwrite a file if it already exists. If you want to add content to a file use the append block.

1.1.2 Read File

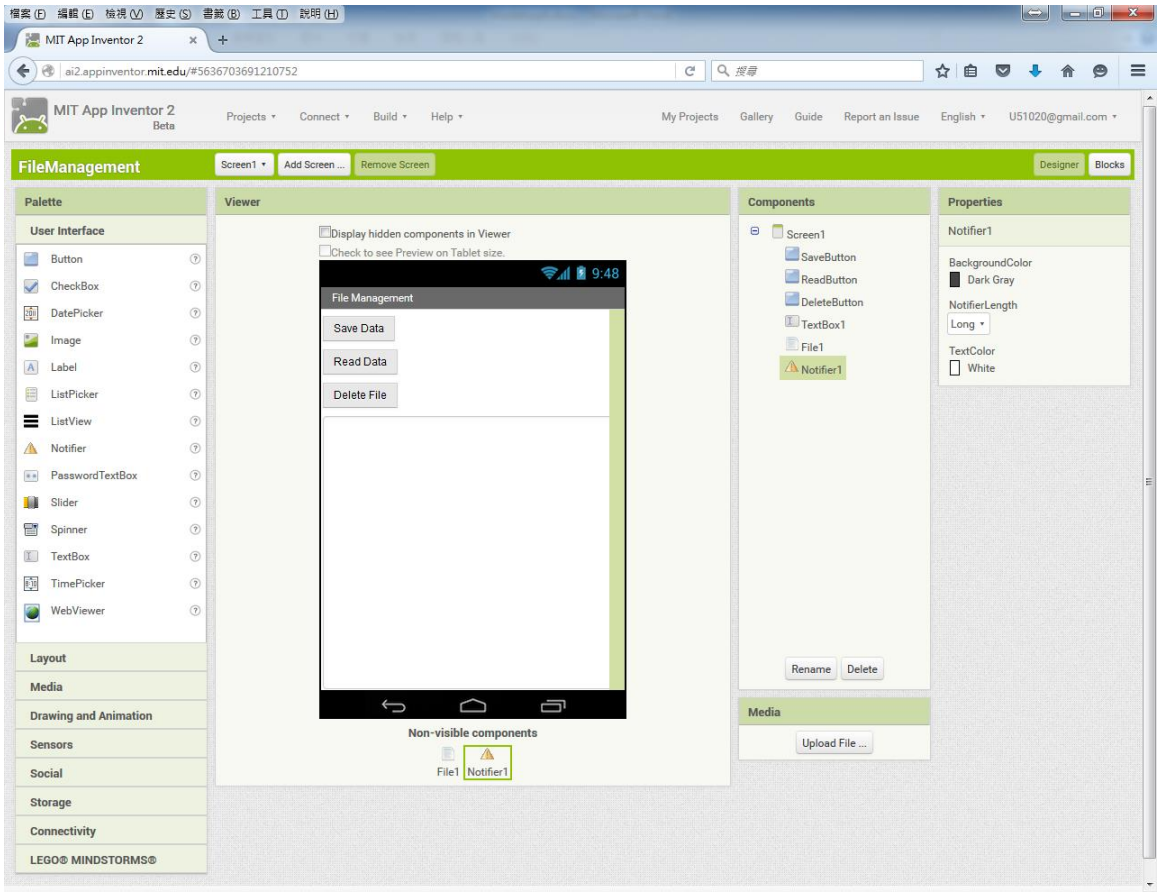
Reads text from a file in storage. Prefix the filename with / to read from a specific file on the SD card (for example, /myFile.txt will read the file /sdcard/myFile.txt). To read assets packaged with an application start the filename with //. If a filename does not start with a slash, it will be read from the application's private storage (for packaged apps) and from /sdcard/AppInventor/data for the Companion.

1.1.3 Delete File

Deletes a file from storage. Prefix the filename with / to delete a specific file in the SD card (for example, /myFile.txt will delete the file /sdcard/myFile.txt). If the filename does not begin with a /, then the file located in the program's private storage will be deleted. Starting the file with // is an error because asset files cannot be deleted.

1.2 Exercise: Text File Management

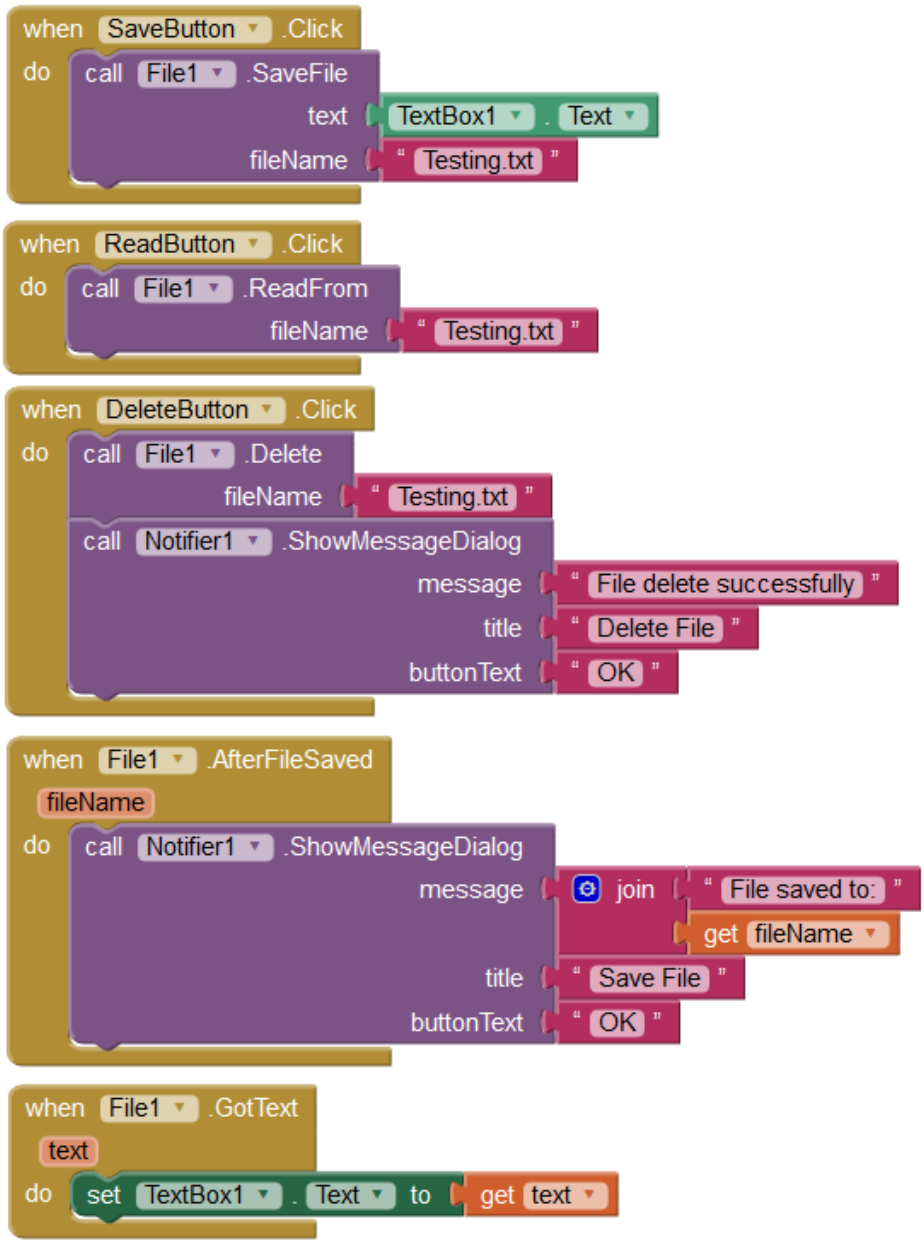
1.2.1 Designer View



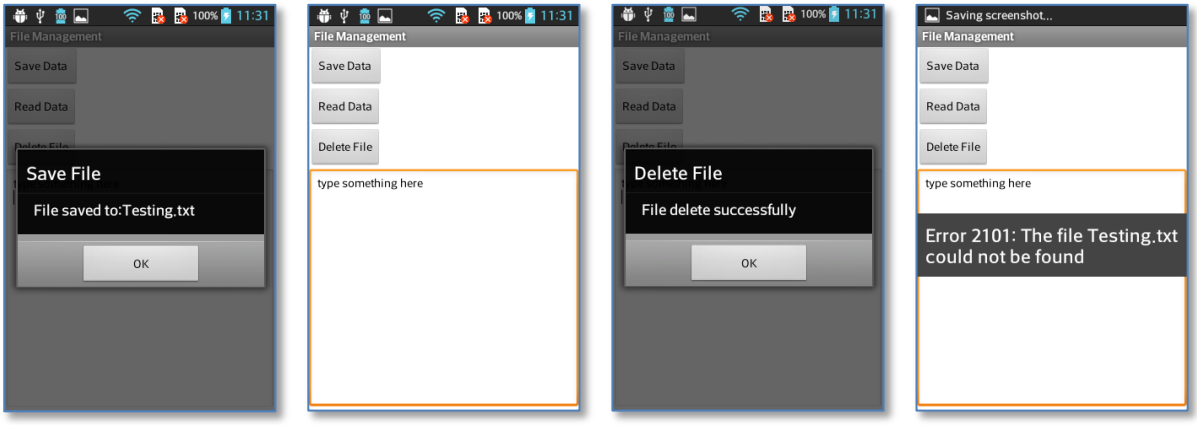
1.2.2 Components

Component	Name	Properties	Remark
Screen	Screen1	Title = "**File Management**"	
Button	SaveButton	Title = "**Save Data**"	
Button	ReadButton	Title = "**Read Data**"	
Button	DeleteButton	Title = "**Delete File**"	
TextBox	TextBox1	Height = "**Fill parent**" Width = "**Fill parent**" MultiLine = "**X**"	
File	File1		
Notifier	Notifier1		

1.2.3 Block Configuration



1.2.4 Sample Output



2. Local Database

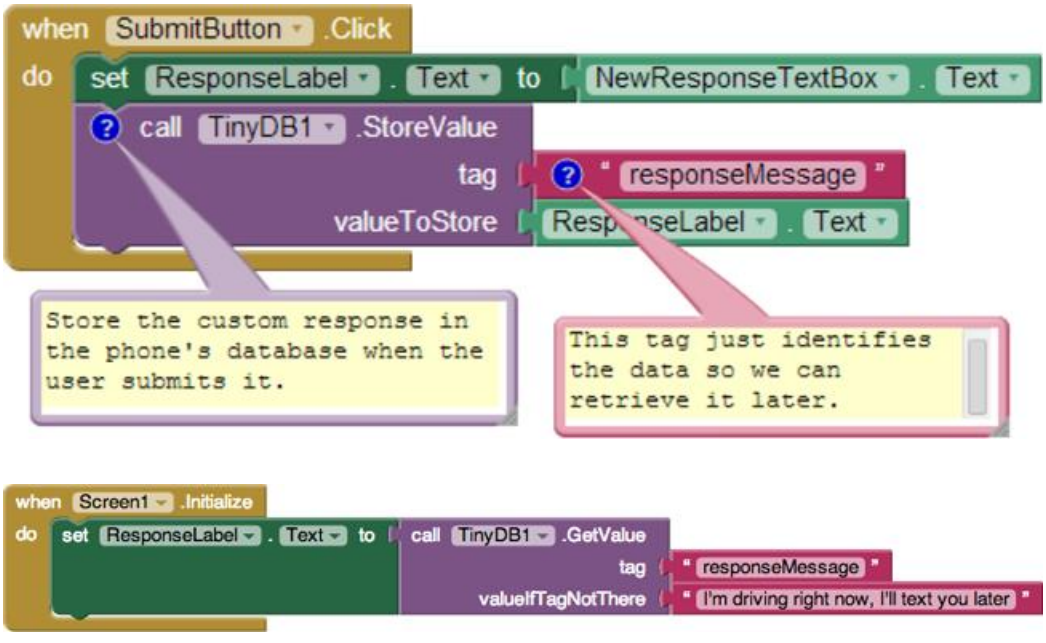
2.1 Tiny DB

TinyDB is a non-visible component that stores data for an app. Apps created with App Inventor are initialized each time they run. This means that if an app sets the value of a variable and the user then quits the app, the value of that variable will not be remembered the next time the app is run. In contrast, TinyDB is a persistent data store for the app. The data stored in a TinyDB will be available each time the app is run. An example might be a game that saves the high score and retrieves it each time the game is played.

Data items are strings stored under tags. To store a data item, you specify the tag it should be stored under. Subsequently, you can retrieve the data that was stored under a given tag.

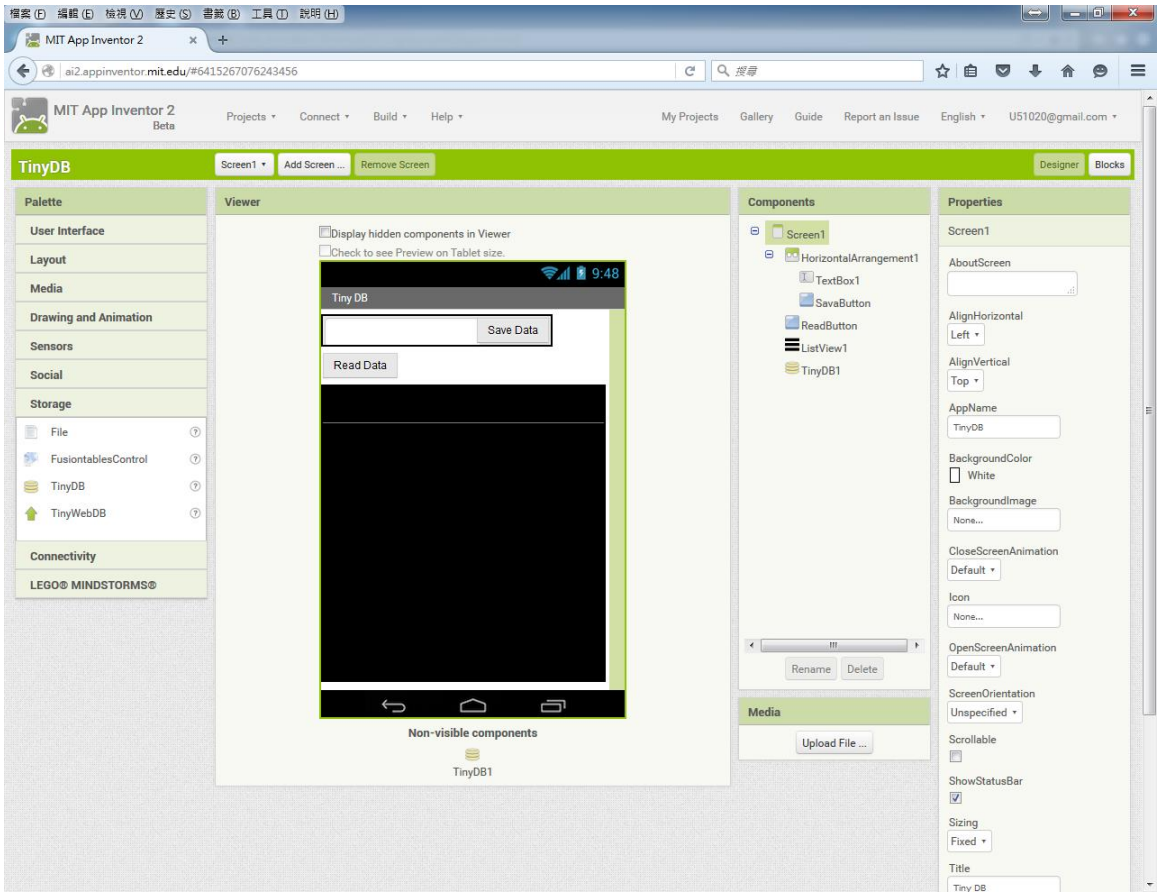
Each app has its own data store. There is only one data store per app. Even if you have multiple TinyDB components, they will use the same data store. To get the effect of separate stores, use different keys. You cannot use the TinyDB to pass data between two different apps on the phone, although you can use the TinyDB to share data between the different screens of a multi-screen app.

When you are developing apps using the AI Companion, all the apps using that Companion will share the same TinyDB. That sharing will disappear once the apps are packaged and installed on the phone. During development you should be careful to clear the Companion app's data each time you start working on a new app.



2.2 Exercise: Tiny Database

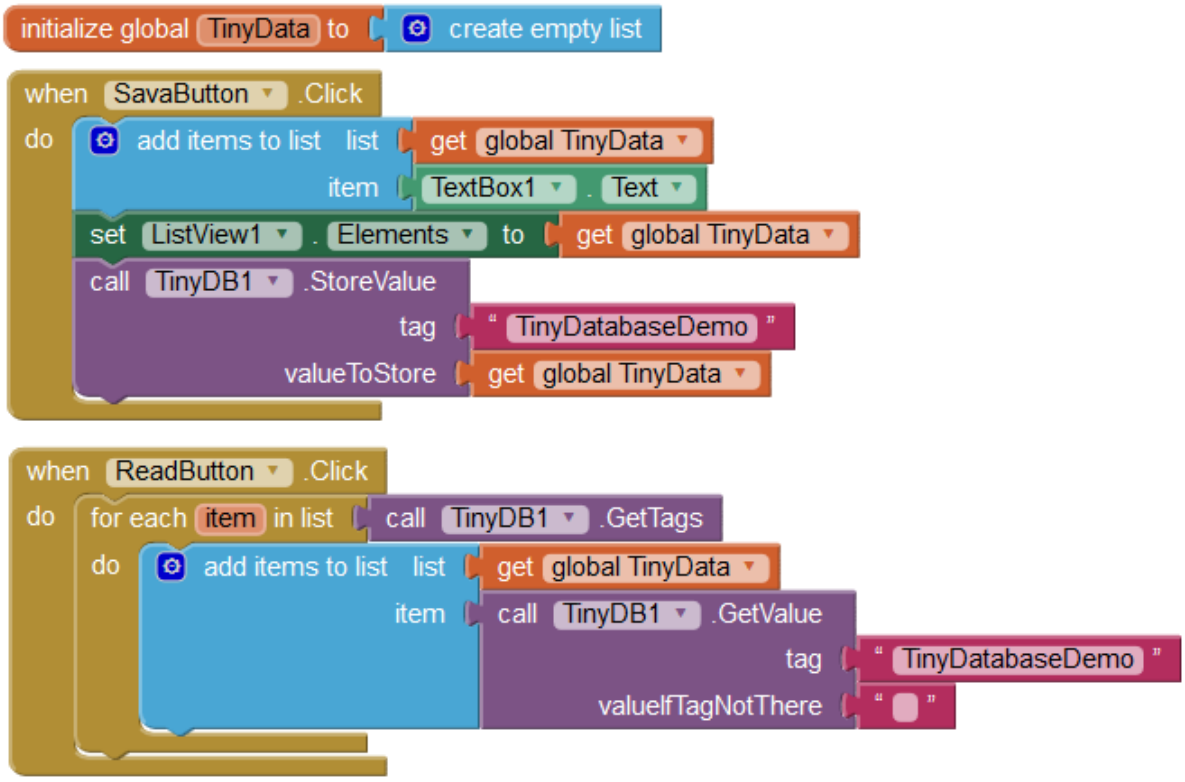
2.2.1 Designer View



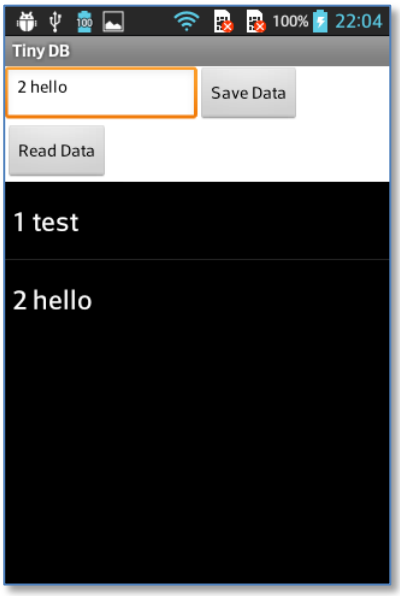
2.2.2 Components

Component	Name	Properties	Remark
Screen	Screen1	Text = "**Tiny DB**"	
HorizontalArrangement	HorizontalArrangement1		
TextBox	TextBox1		Inside HorizontalArrangement1
Button	SaveButton	Text = "**Save Data**"	Inside HorizontalArrangement1
Button	ReadButton	Text = "**Read Data**"	
ListView	ListView1	Height = "**Fill parent**" Width = "**Fill parent**"	
TinyDB	TinyDB1		

2.2.3 Block Configuration



2.2.4 Sample Output



3. Web Database

3.1 Tiny Web DB

Many apps have data that is stored in a web server and shared amongst users and devices. TinyWebDB is the key component you'll need. It is an App Inventor component that let your app store persistent data on the web and share data amongst phones and people. It has similar blocks as TinyDB, but the data is stored on the web instead of privately on the device.

- TinyWebDB is for talking to special App Inventor web services that store data in the same format as TinyDB.
- It is different than the Web component, which is for communicating with any kind of API and doesn't have the StoreValue/GetValue commands.

3.2 Where is the Data stored?

TinyWebDB has a property SourceURL. You can set it to any App Inventor compliant web service, that is, any site that has been setup especially for use with App Inventor and TinyWebDB. By default, TinyWebDB stores data at appinvtinywebdb.appspot.com. Be careful, though, as this web database is shared amongst all App Inventor programmers. It also has a limit of 100 total entries and thus is for testing purposes only.

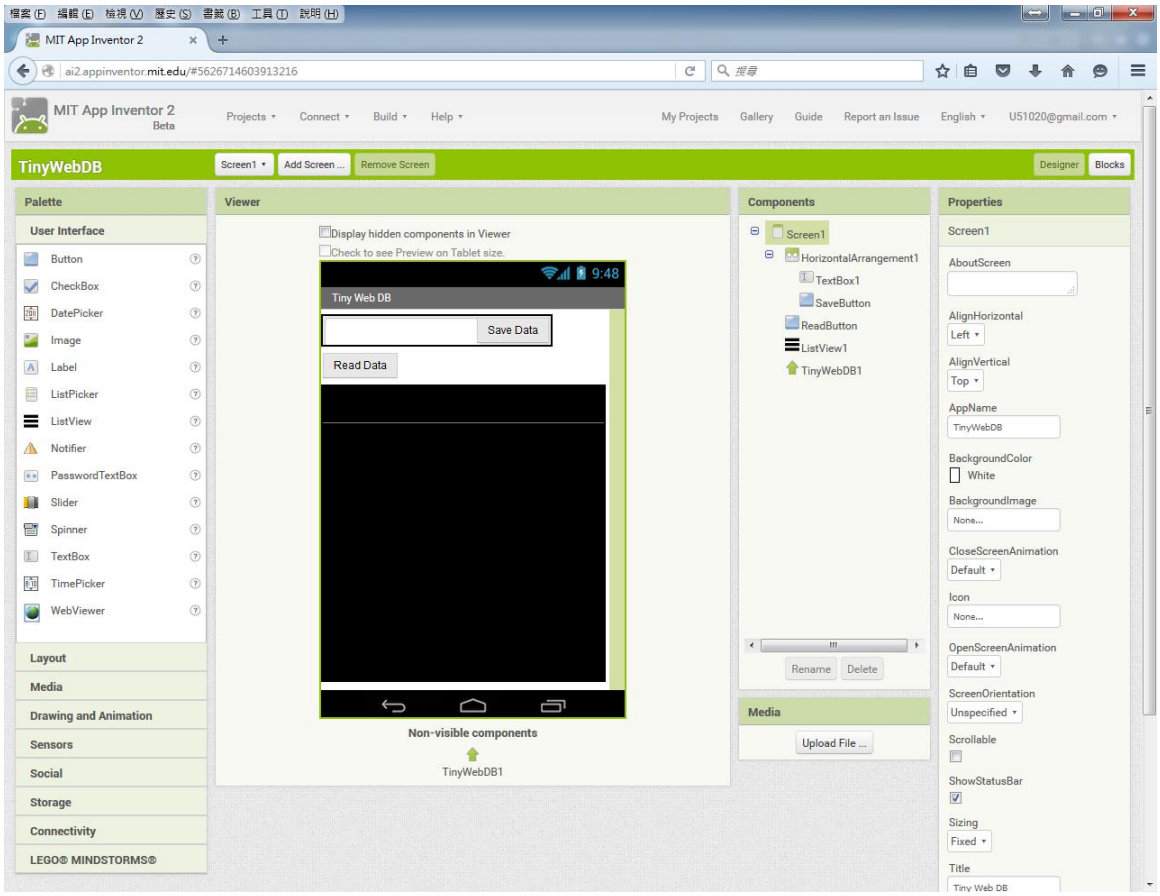
3.3 Comparison between TinyWebDB and TinyDB Blocks

TinyWebDB provides similar functions to TinyDB: StoreValue and GetValue. StoreValue works exactly the same, but GetValue works a bit differently. When you call GetValue, you're really just requesting the data and the data aren’t immediately returned. Instead, you must code the GotValue event-handler, which is triggered when data actually arrives from the web.

- In the TinyDB solution, the app calls GetValue and checks if the data is a list. If the data is a list, it is placed in the variable NoteList and the list is displayed.
- In the TinyWebDB solution, GetValue is again called from within Screen.Initialize, but note that it returns no data. The TinyWebDB.GetValue block doesn't have anything coming out the left that can be plugged in to a slot. Instead, the GetValue just sends a request to the web. Eventually, the data will arrive, and then the GotValue event-handler is triggered. When it is, the TagFromWebDB gives you the tag of the request and the valueFromWebDB is the data that was requested. As you can see, the data is processed in a similar manner to the TinyDB solution.

3.4 Exercise: Tiny Web Database

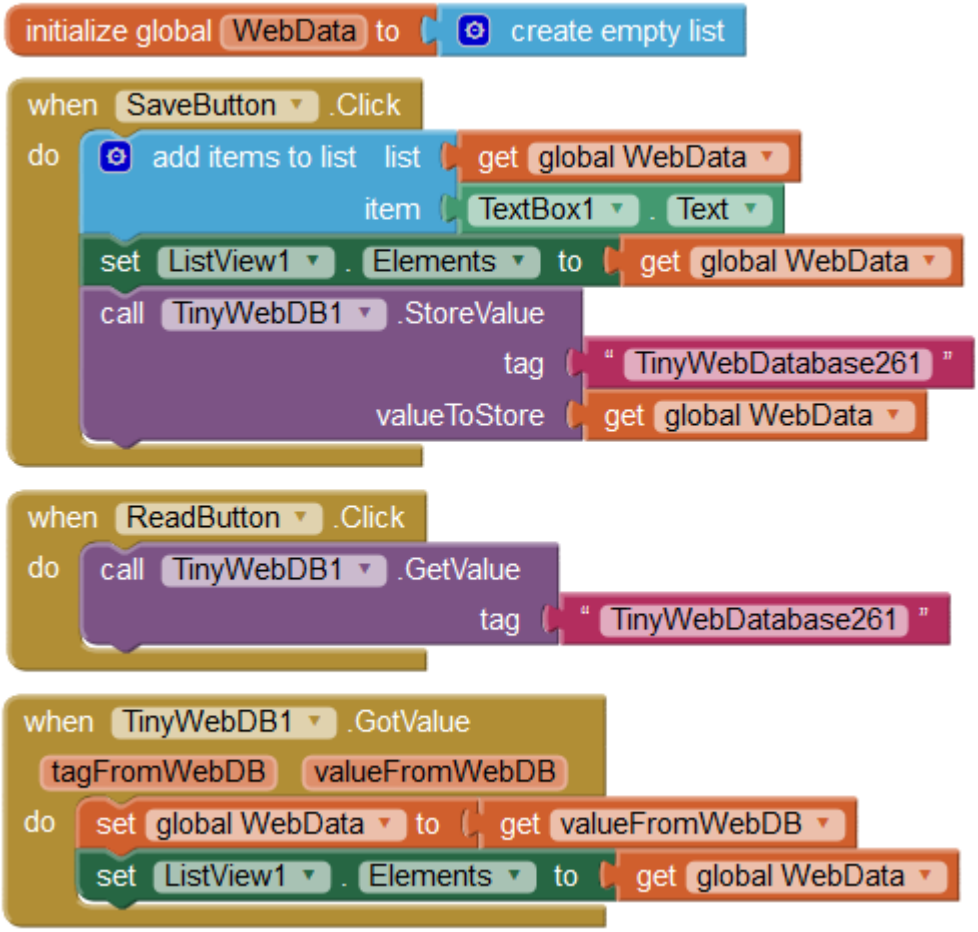
3.4.1 Designer View



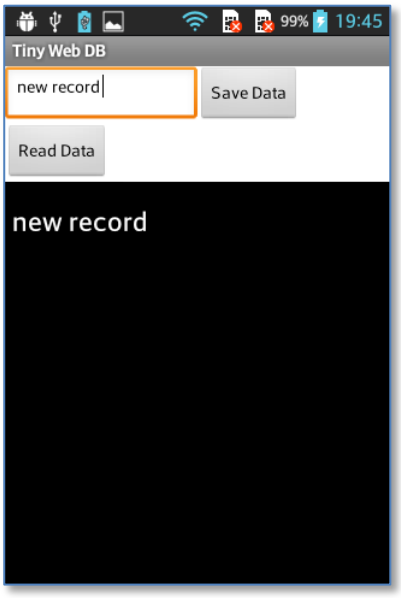
3.4.2 Components

Component	Name	Properties	Remark
Screen	Screen1	Text = "**Tiny Web DB**"	
HorizontalArrangement	HorizontalArrangement1		
TextBox	TextBox1		Inside HorizontalArrangement1
Button	SaveButton	Text = "**Save Data**"	Inside HorizontalArrangement1
Button	ReadButton	Text = "**Read Data**"	
ListView	ListView1	Height = "**Fill parent**" Width = "**Fill parent**"	
TinyWebDB	TinyWebDB1		

3.4.3 Block Configuration



3.4.4 Sample Output



4. Fusion Tables

4.1 Fusion Tables Control

A Fusion Table is a Google service to support the gathering, managing, sharing, and visualizing of data. Data is stored in Google's cloud. All of the data are stored in a public table that can be accessed via Google Drive, and allows different users to add information to the tables. Fusion Tables let you store, share, query and visualize data tables; this component lets you query, create, and modify these tables. This component uses the Fusion Tables API V1.0.

Applications using Fusion Tables must authenticate with Google's servers. There are two ways this can be done. The first way only uses an API Key which the developer obtains. With this approach end-users must also login to access a Fusion Table.

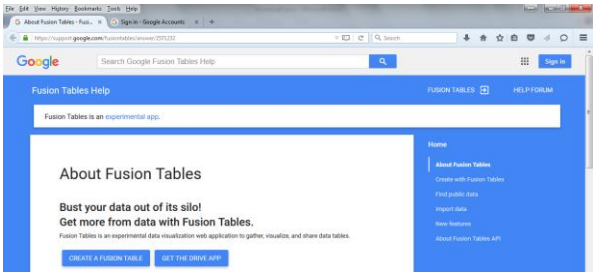
The second approach is to use Service Authentication. With this approach you create credentials and a special "Service Account Email Address" which allows end-users to use your Fusion Tables without logging in; your service account authenticates all access.

4.2 Create Fashion Table

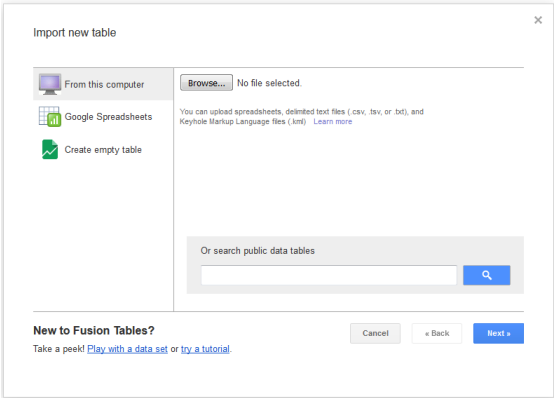
4.2.1 Creating Fusion Tables

You will probably want to create your own Fusion Tables to experiment with as you are developing your apps. This is as easy as creating a Google document. Here are the steps:

1. Go to https://www.google.com/fusiontables/, and select **[Create a Fusion Table]**. Then login with your Gmail account.

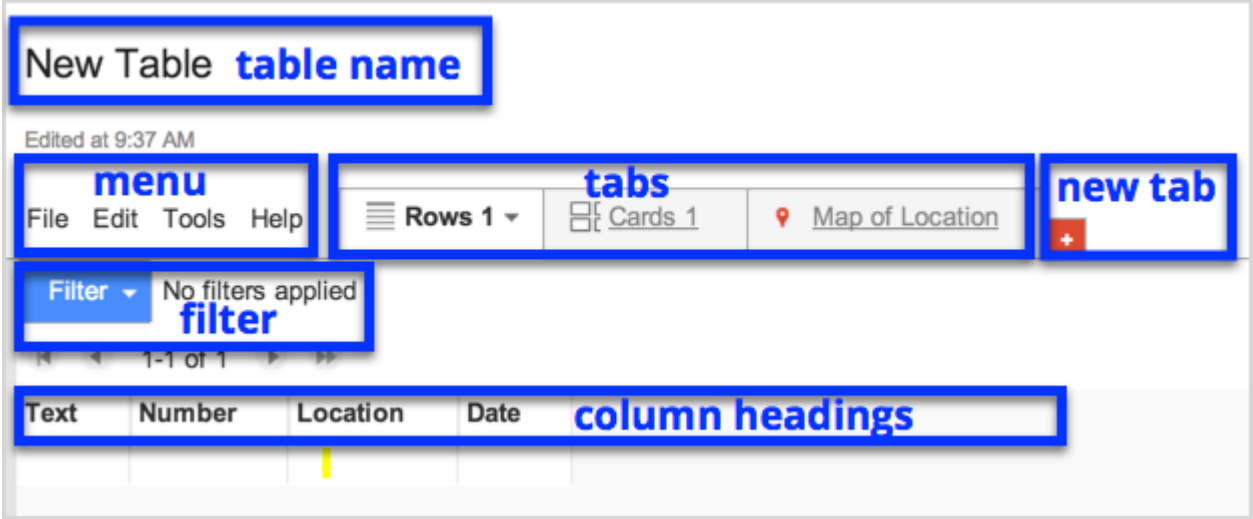


2. Select **[Create empty table],** and press **[Next]** to continue.



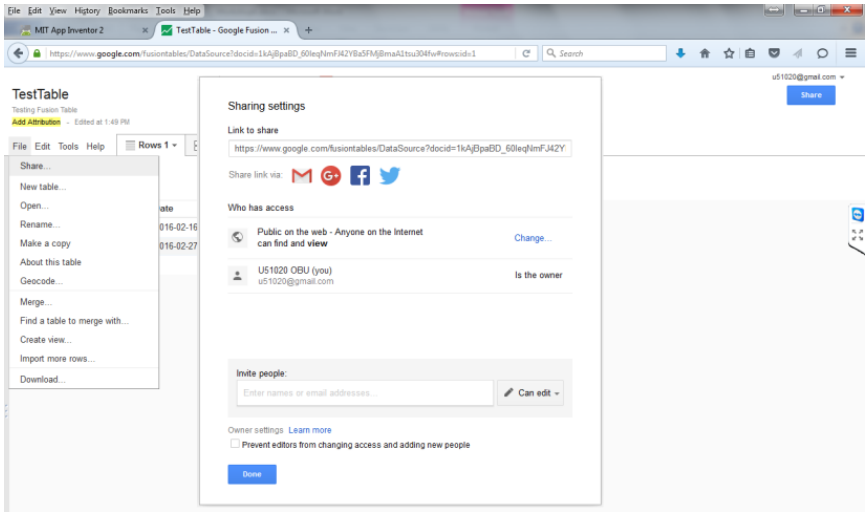
4.2.2 Understanding Fusion Table Interface

The current name of this table is New Table. This is not very descriptive. Change the name. Enter the new name in the Name field, and click [Save]. By default, others can download your data since Allow Downloads is checked. The Fusion Tables Interface includes the following components:

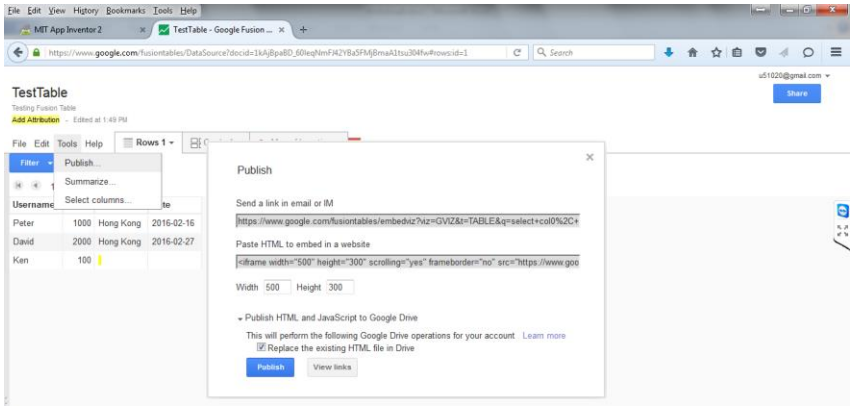


In order to publish your table, you need to:

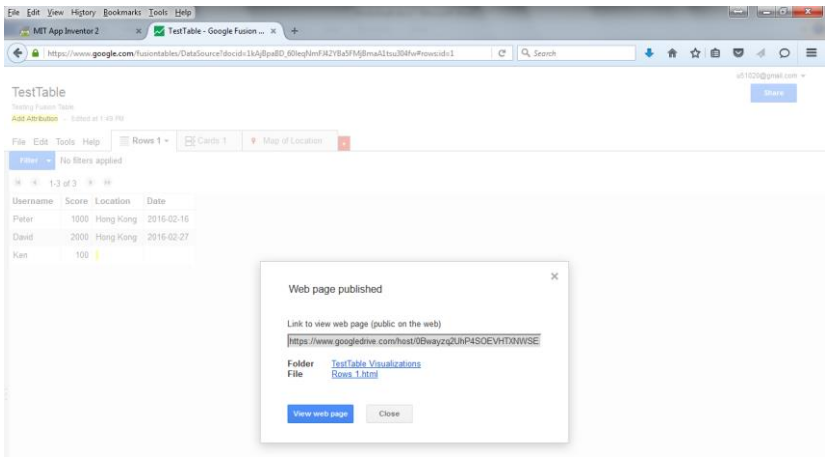
1. Select **File ➔ Share to** for accessing Sharing setting, and then change the access setting to "**Public on the web – Anyone on the Internet can find and view**". Finally, press **[Done]** button to confirm.



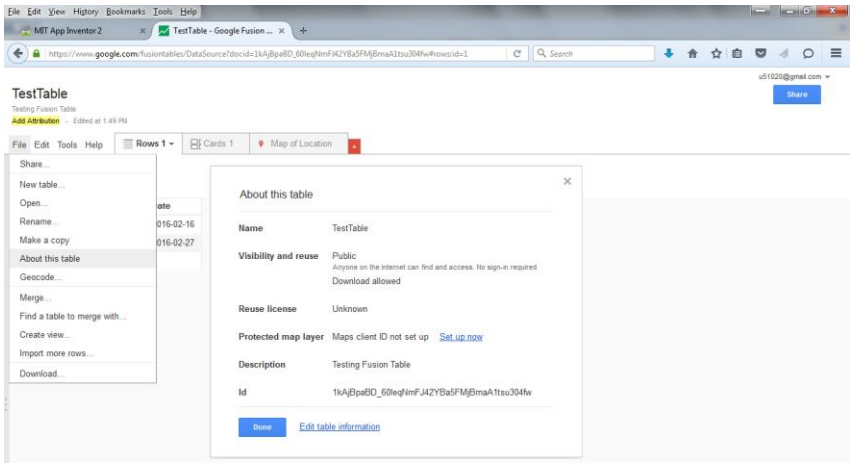
2. Select **Tools ➔ Publish** for publishing, and then press **[Publish]** to share.



3. Confirm the link for the web page, and press **[Close]** to continue.



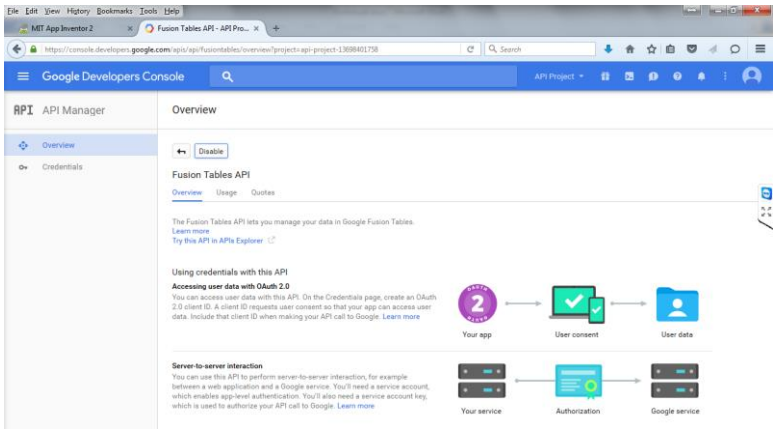
3. Finally, select **File ➔ About this table** to obtain the table ID.



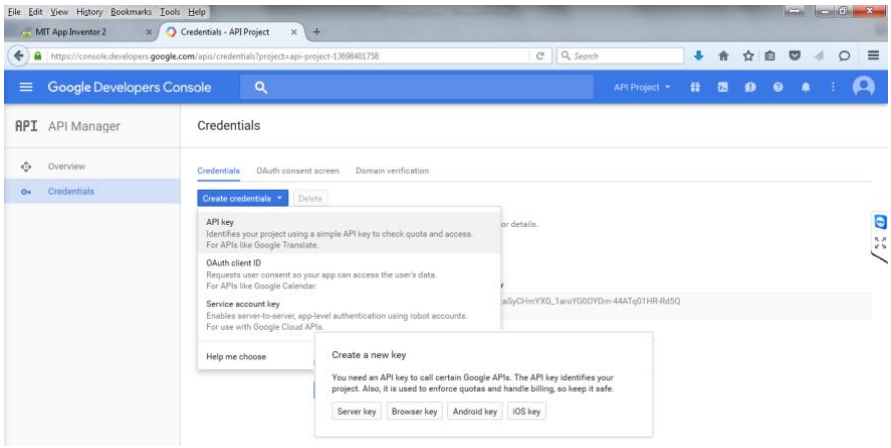
4.2.3 Register API Key

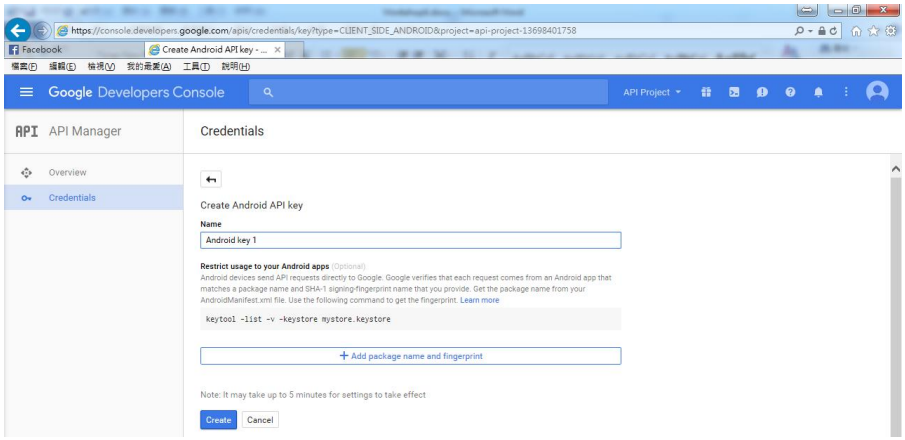
When you drag the FusiontablesControl component onto the Designer, don't forget to set its ApiKey property, which is initially blank. You should copy this from your Google Developers Console and paste it into the property field. To get an API key, follow these instructions.

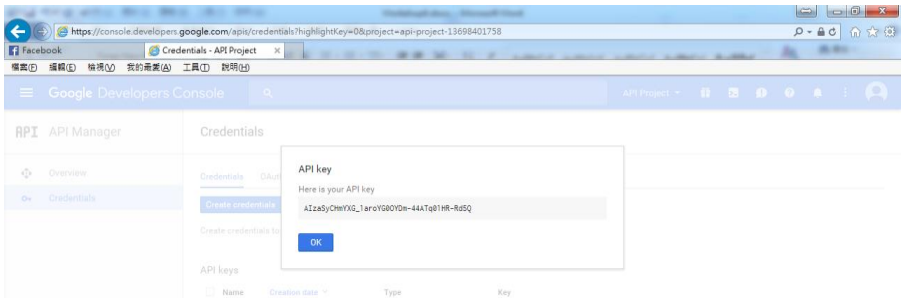
1. Go to and login your Google Developers Console – API Manager - Fusion Table Control (https://console.developers.google.com/apis/credentials/wizard?api=fusiontables).
2. Under APIs & auth select the APIs item from the menu on the left.
3. Choose the Fusion Tables API from the list provided and turn it on.



4. On the left bar, select the Credentials item. Under Public API access click Create new Key, choose Android key and click Create to generate an API key.







Your API key(s) will appear in the pane next to "Public API access". You must provide that key as the value for the ApiKey property in your Fusion Tables app. Once you have an API key, set the value of the Query property to a valid Fusiontables SQL query and call SendQuery to execute the query. App Inventor will send the query to the Fusion Tables server and the GotResult block will fire when a result is returned from the server. Query results will be returned in CSV format, and can be converted to list format using the "list from csv table" or "list from csv row" blocks.

Note that you do not need to worry about UTF-encoding the query. But you do need to make sure the query follows the syntax described in the reference manual, which means that things like capitalization for names of columns matters, and that single quotes must be used around column names if there are spaces in them.

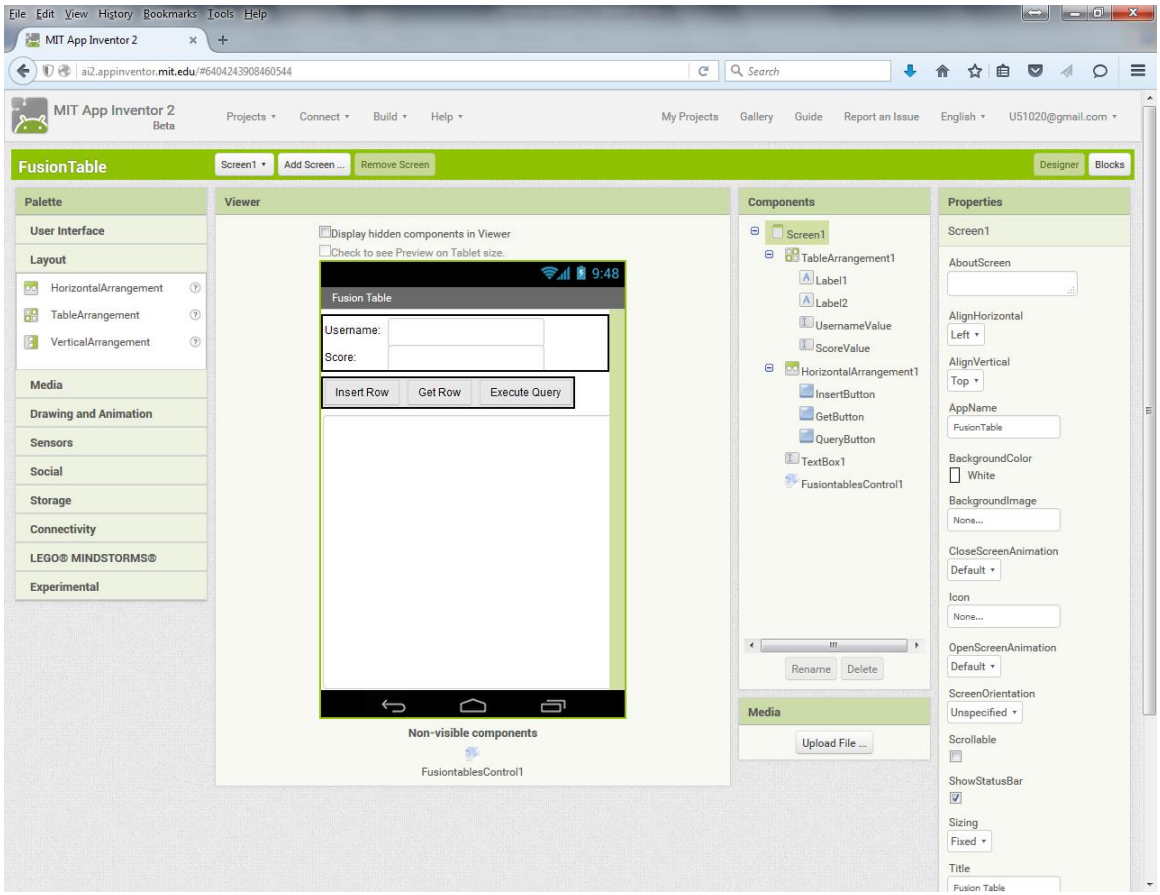
4.2.4 Service Authentication

To set up Service Authentication for your Fusion Table, follow these additional steps:

1. In the Google APIs Console under APIs & auth select the APIs item from the menu on the left.
2. Click the Create new Client ID button. Select the Service account option, and click Create Client ID.
3. A file called the KeyFile will automatically download onto your computer. Save it in a place you will remember. Once the creation is complete, you will get a table with your Service Account information.
4. In the designer window of App Inventor, select the FusionTablesControl. In the properties pane, add the ServiceAccountEmail, upload the KeyFile, and check the UseServiceAuthentication box.
5. Share the Fusion Table with your ServiceAccountEmail, and give it editing permissions, just like you would share any other Google Doc with an email address.

4.3 Exercise: Fusion Table

4.3.1 Designer View

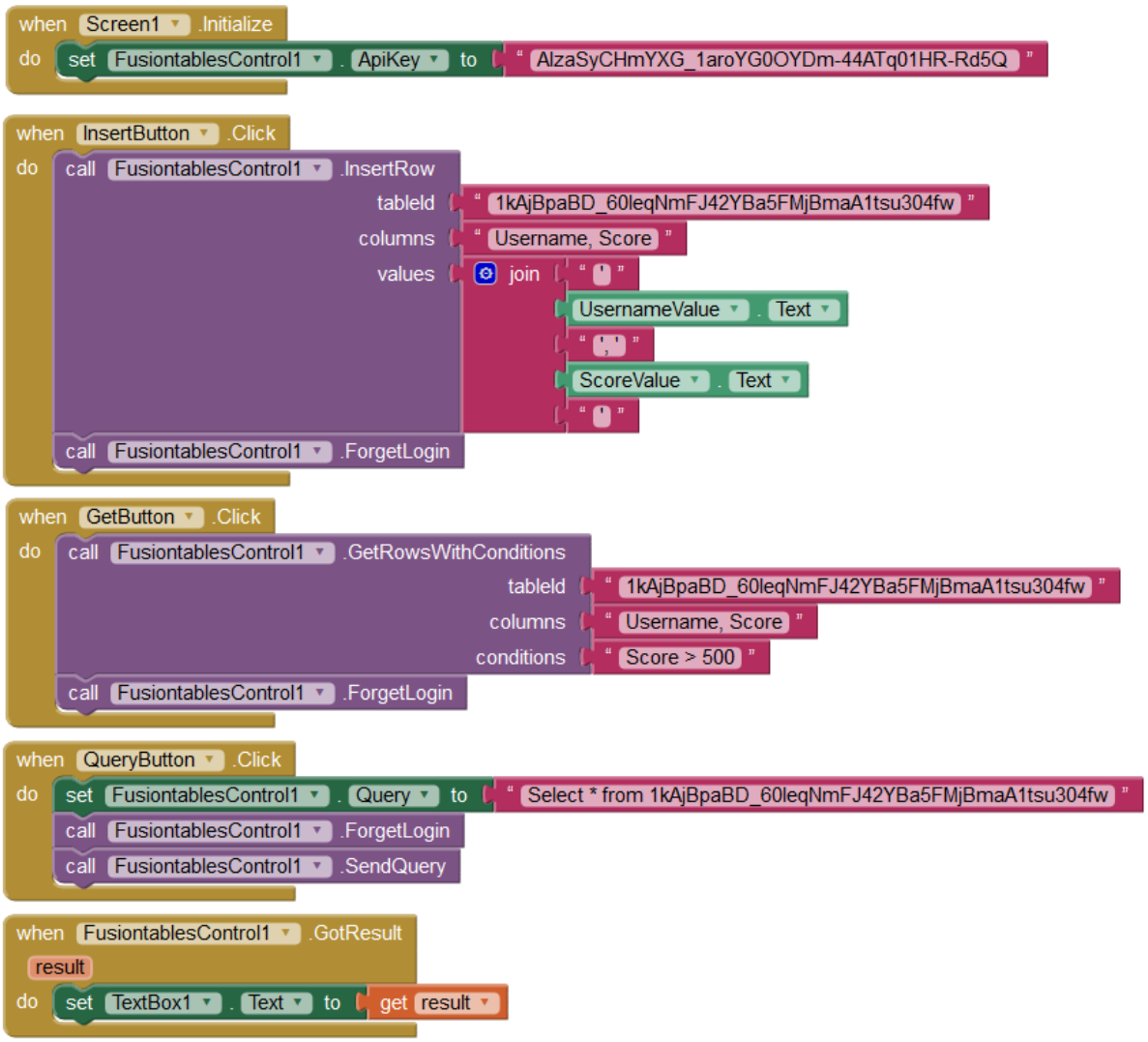


4.3.2 Components

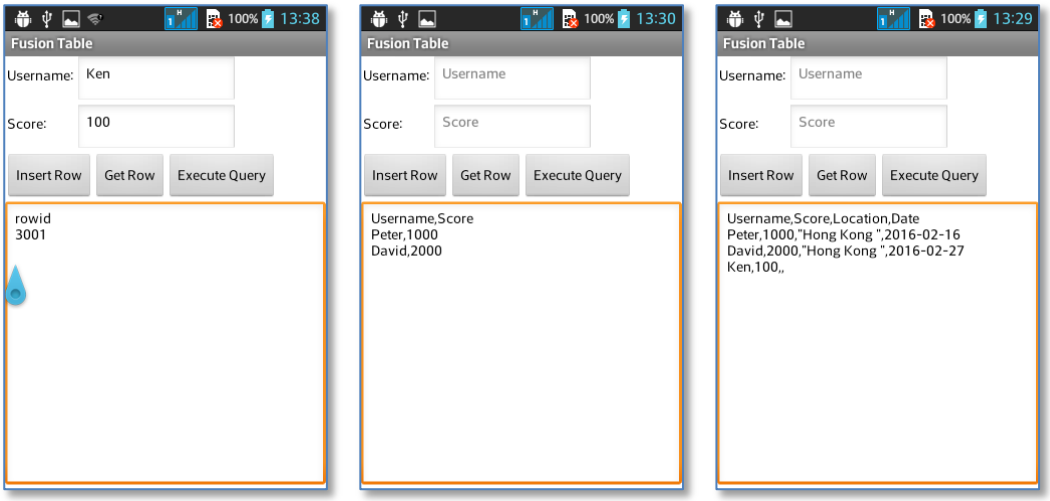
Component	Name	Properties	Remark
Screen	Screen1	Text = "**Fusion Table**"	
TableArrangement	TableArrangement1	Width = "**Fill parent**"	

Label	Label1	Text = "**Username:**"	Inside TableArrangement1
Label	Label2	Text = "**Score:**"	Inside TableArrangement1
TextBox	UsernameValue	Hint = "**Username**"	Inside TableArrangement1
TextBox	ScoreValue	Hint = "**Score**"	Inside TableArrangement1
HorizontalArrangement	HorizontalArrangement1		
Button	InsertButton	Text = "**Insert Row**"	Inside HorizontalArrangement
Button	GetButton	Text = "**Get Row**"	Inside HorizontalArrangement
Button	QueryButton	Text = "**Execute Query**"	Inside HorizontalArrangement
TextBox	TextBox1	Height = "**Fill parent**" Width = "**Fill parent**"	
FusionTablesControl	FusionTablesControl1		

4.3.3 Block Configuration



4.3.4 Sample Output



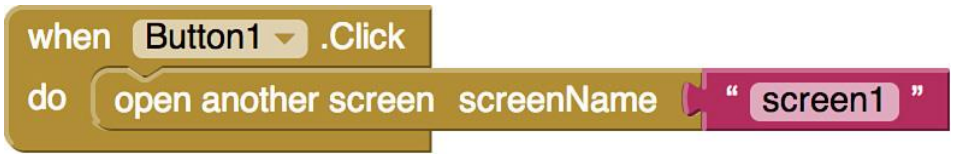
5. Create App with Multiple Screen

5.1 Handle Multiple Screens

In App Inventor, you can have one screen open a second screen. Later, the second screen can return to the screen that opened it. You can have as many screens as you like, but each screen closes by returning to the screen that opened it. The screens can share information by passing and returning values when they open and close. Creating such apps is not very different from creating apps with only a single screen. In fact, you should think of this process as creating several stand-alone apps which communicate by sending messages to each other.

Every screen that you create has its own components in the Designer window. In the Blocks Editor, you will be able to see only these components and none of the components from the other screens in your app. Similarly, blocks of code related to a screen cannot refer to blocks of code in another screen. Please note that none of the components, variable definitions, and procedures that you define in one screen will be accessible from a second screen.

To open another screen, you use the block under the Control palette called open another screen. This block requires one input, which must be the name of the screen you want to open, in a text block.



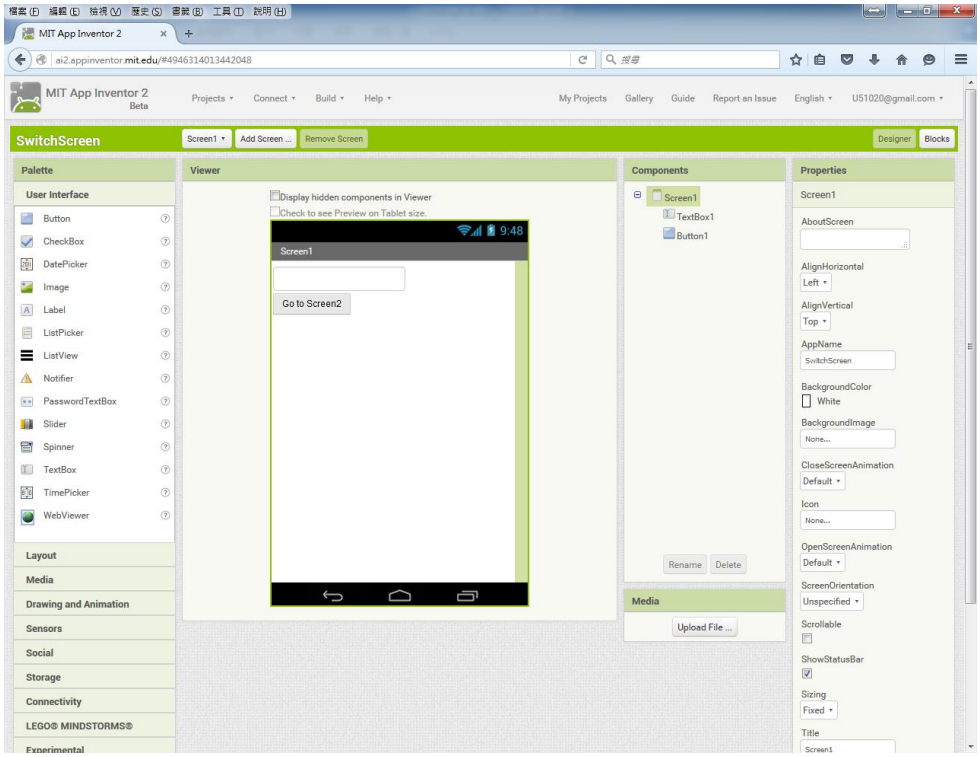
Current limitations of the AI Companion app:

- The close screen block triggers the Initialize event instead of the OtherScreenClosed event.
- The close screen with value block triggers both the Initialize and OtherScreenClosed events instead of only the OtherScreenClosed event.
- The close application block does not work, a message 'Closing forms is not currently supported during development' will be displayed instead.

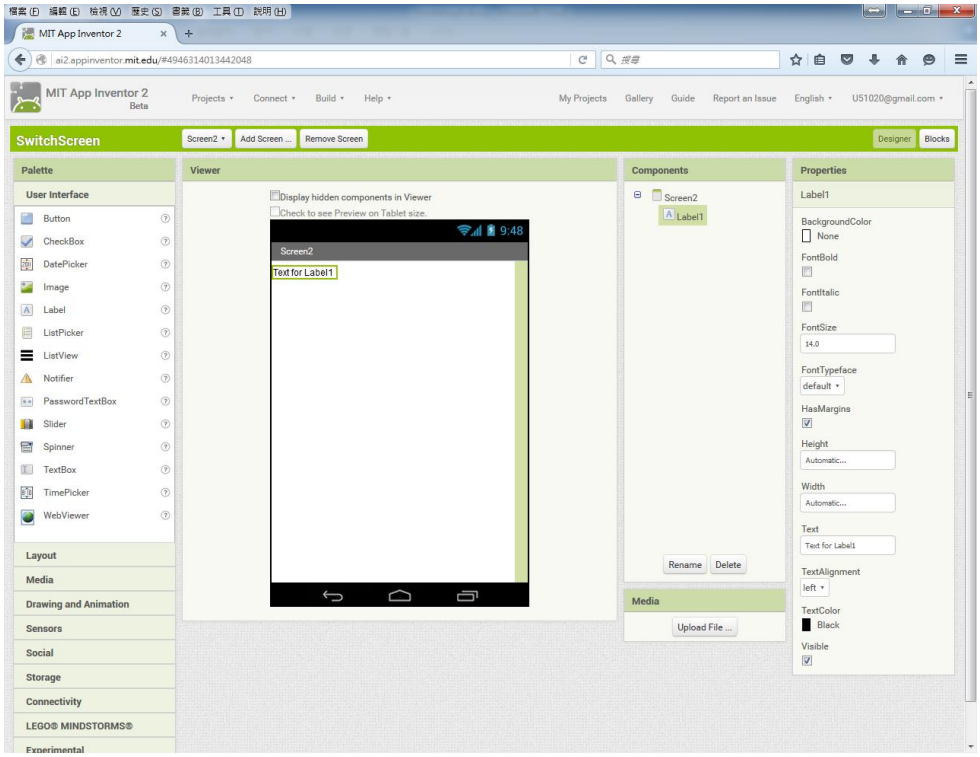
5.2 Exercise: Switch Screen

5.2.1 Designer View

5.2.1.1 Screen1



5.2.1.2 Screen2



5.2.2 Components

5.2.2.1 Screen1

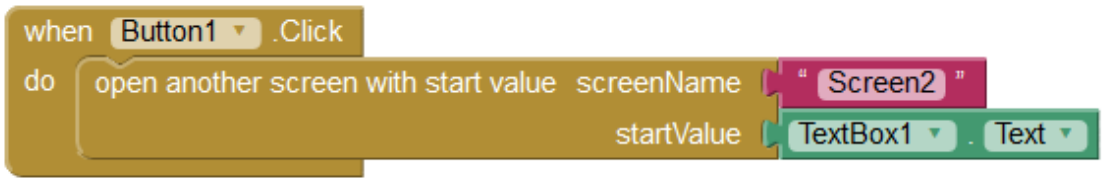
Component	Name	Properties	Remark
Screen	Screen1		
TextBox	TextBox1		
Button	Button1	Text = "**Go to Screen2**"	

5.2.2.2 Screen2

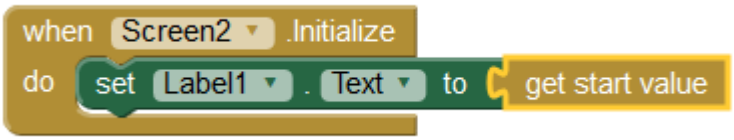
Component	Name	Properties	Remark
Screen	Screen2		
Label	Label1		

5.2.3 Block Configuration

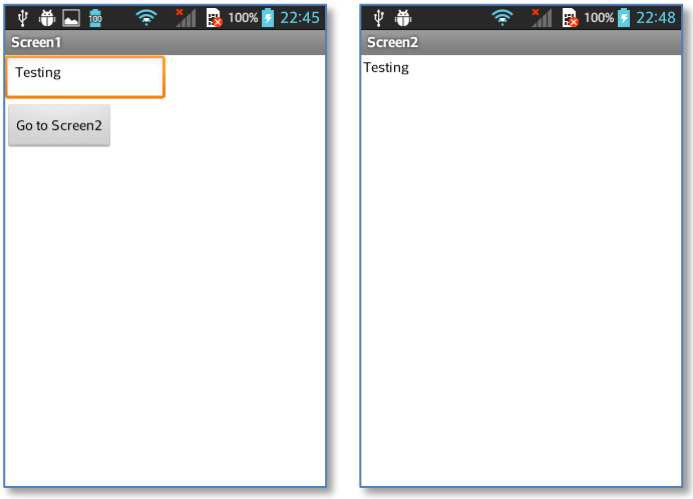
5.2.3.1 Screen1



5.2.3.2 Screen2



5.2.4 Sample Output



6. Interactive Multimedia Game

6.1 Exercise: Crazy Bird

6.1.1 Media Files

6.1.1.1 Image Files

background.png

Gameover.png

Mainmenu.png

bird0.png

bird1.png

bird2.png

ghost.png

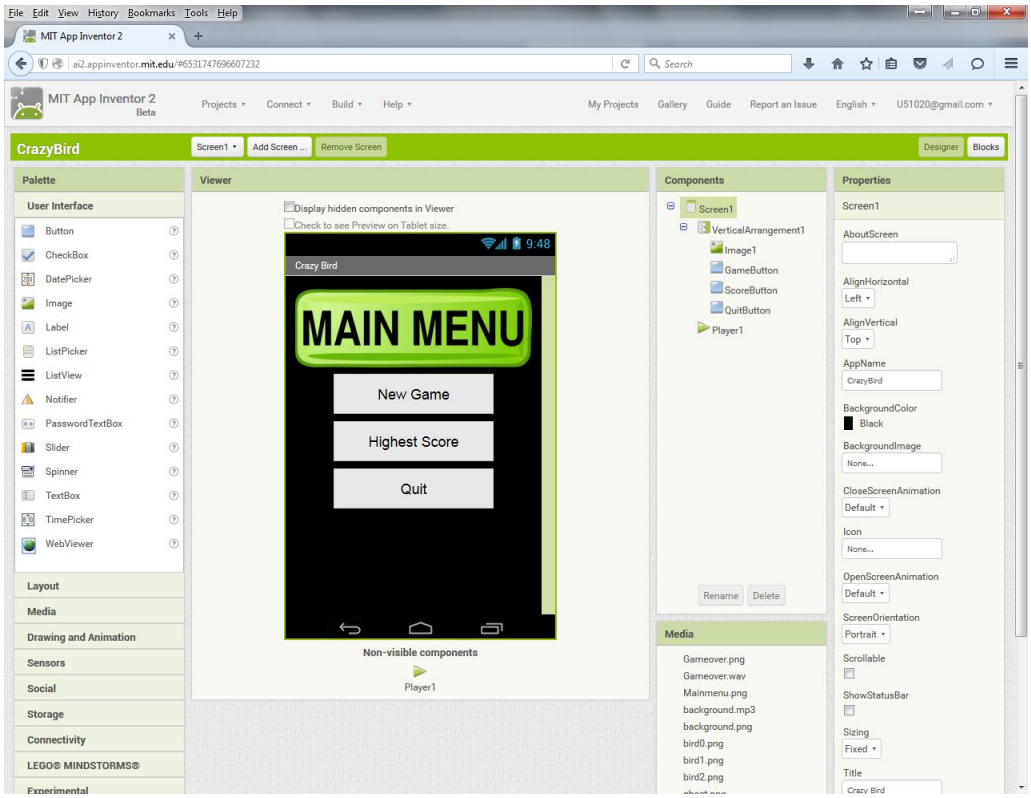
6.1.1.2 Sound Files

background.mp3

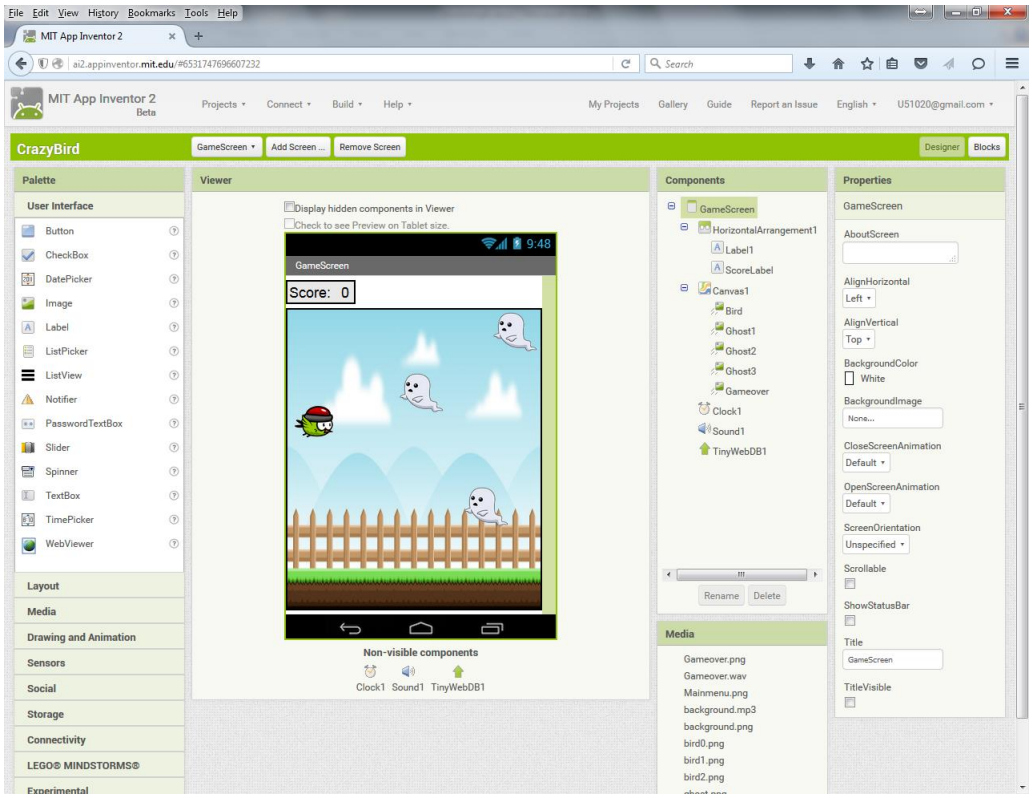
Gameover.wav

6.1.2 Designer View

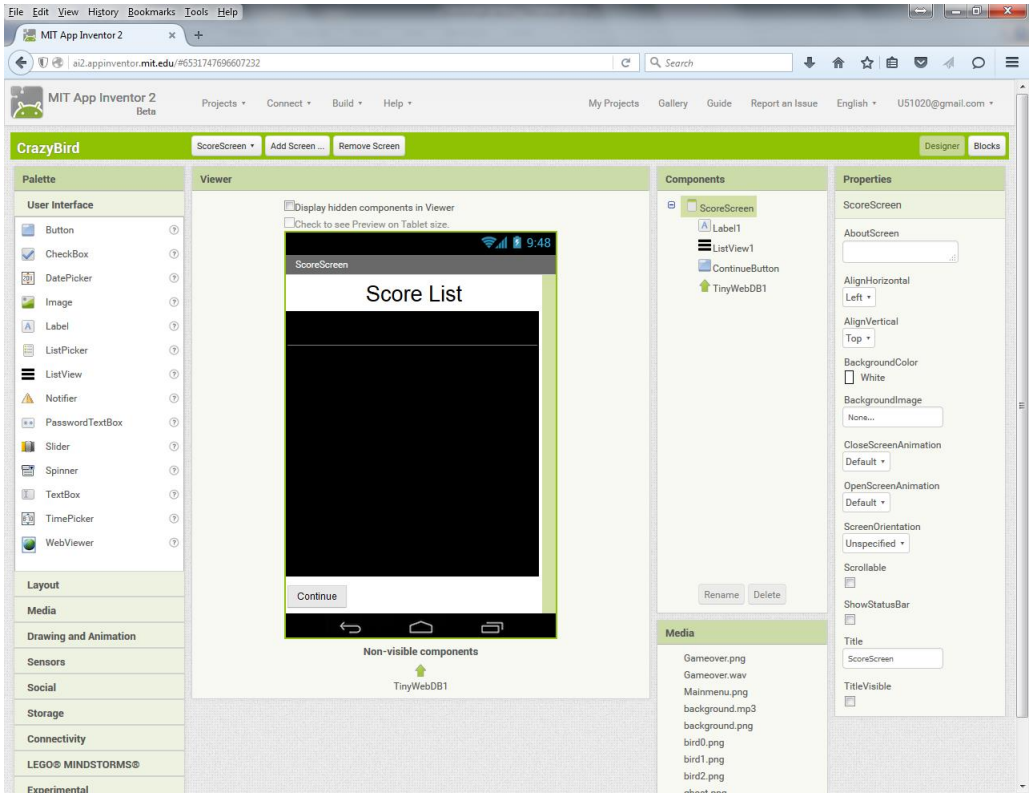
6.1.2.1 Screen1



6.1.2.2 GameScreen



6.1.2.3 ScoreScreen



6.1.3 Components

6.1.3.1 Screen1

Component	Name	Properties	Remark
Screen	Screen1	BackgroundColor = "**Black**" ScreenOrientation = "**Portrait**" ShowStatusBar = *[Blank]* TitleVisible = *[Blank]*	
VerticalArrangement	VerticalArrangement1	AlignHorizontal = "**Center**" Width = "**Fill parent**"	
Image	Image1	Picture = "**Mainmenu.png**"	Inside VerticalArrangement1
Button	GameButton	FontSize = "**20**" Height = "**50 pixels**" Width = "**200 pixels**" Text = "**New Game**" TextAlignment = "**center**"	Inside VerticalArrangement1
Button	ScoreButton	FontSize = "**20**" Height = "**50 pixels**" Width = "**200 pixels**" Text = "**Highest Score**" TextAlignment = "**center**"	Inside VerticalArrangement1
Button	QuitButton	FontSize = "**20**" Height = "**50 pixels**" Width = "**200 pixels**" Text = "**Quit**" TextAlignment = "**center**"	Inside VerticalArrangement1
Player	Player1	Source = "**background.mp3**"	

6.1.3.2 GameScreen

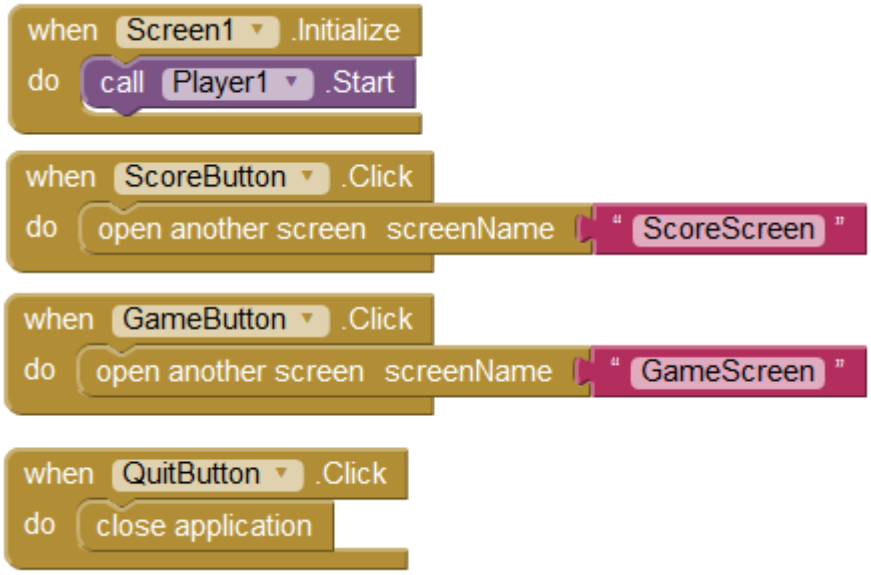
Component	Name	Properties	Remark
Screen	GameScreen	ScreenOrientation = "**Portrait**" ShowStatusBar = *[Blank]* TitleVisible = *[Blank]*	
HorizontalArrangment	HorizontalArrangment1		
Label	Label1	FontSize = "**20**" Text = "**Score:**"	Inside HorizontalArrangment1
Label	ScoreLabel	FontSize = "**20**"	Inside HorizontalArrangment1
Canvas	Canvas1	Height = "**Fill parent**" Width = "**Fill parent**"	
ImageSprite	Bird	Heading = "**270**" Picture = "**bird1.png**" Speed = "**5**" X = "**6**"	Inside Canvas1
ImageSprite	Ghost1	Height = "**60 pixels**" Width = "**60 pixels**" Heading = "**180**" Picture = "**ghost.png**"	Inside Canvas1
ImageSprite	Ghost2	Height = "**60 pixels**" Width = "**60 pixels**" Heading = "**180**" Picture = "**ghost.png**"	Inside Canvas1
ImageSprite	Ghost3	Height = "**60 pixels**" Width = "**60 pixels**" Heading = "**180**" Picture = "**ghost.png**"	Inside Canvas1
ImageSprite	Gameover	Picture = "**Gameover.png**" X = "**5**" Y = "**75**"	Inside Canvas1
Clock	Clock1	TimerAlwaysFires = "**X**" TimerEnabled = "**X**" TimerInterval = "**100**"	
Sound	Sound1	Source = "**Gameover.wav**"	
TinyWebDB	TinyWebDB1		

6.1.3.3 ScoreScreen

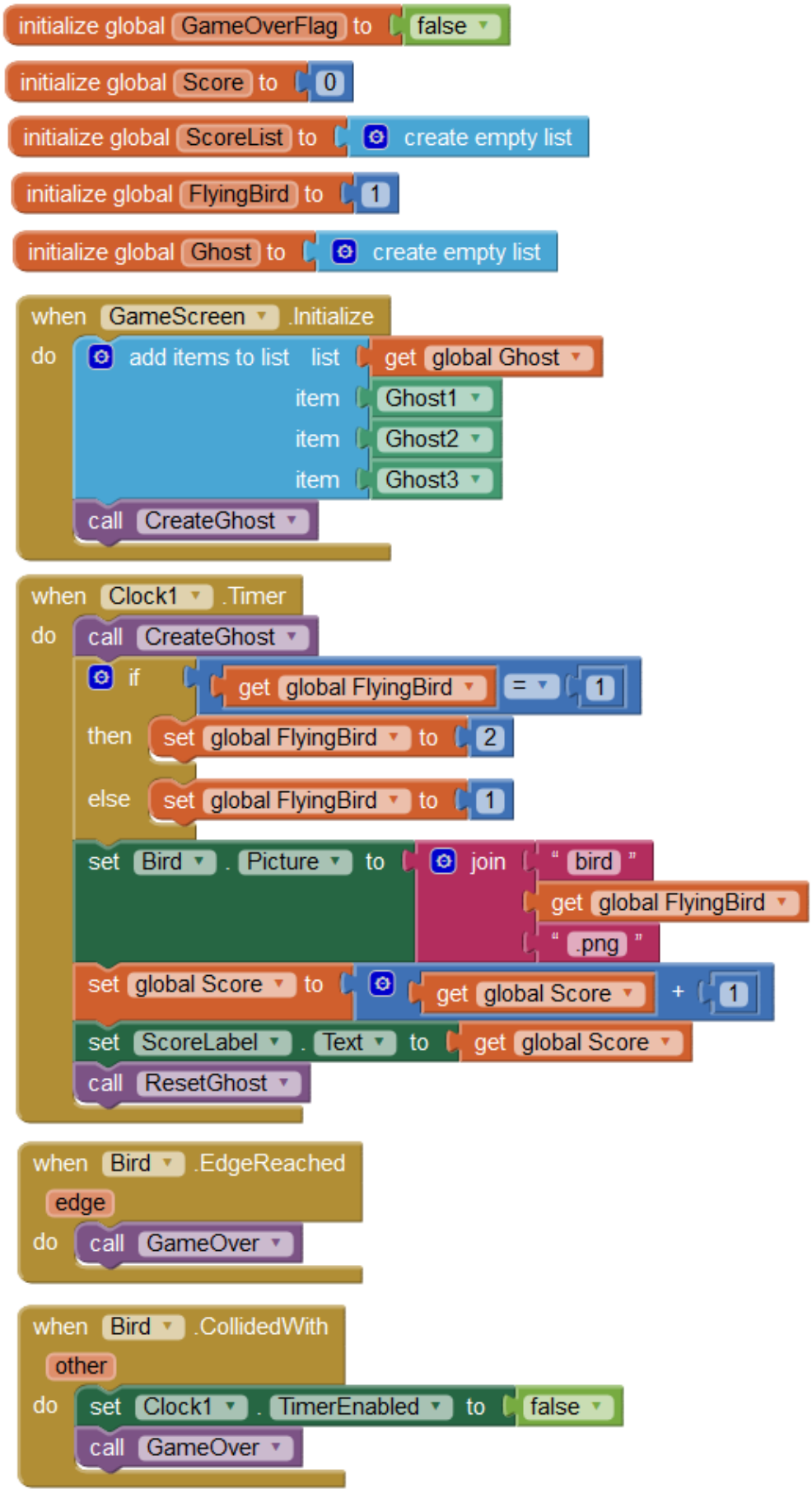
Component	Name	Properties	Remark
Screen	ScoreScreen	ScreenOrientation = "**Portrait**" ShowStatusBar = *[Blank]* TitleVisible = *[Blank]*	
Label	Label1	FontSize = "**30**" Width = "**Fill parent**" Text = "**Score List**" TextAlignment = "**center**"	
ListView	ListView1	Height = "**Fill parent**" Width = "**Fill parent**" TextSize = "**22**"	
Button	ContinueButton	Text = "**Continue**" TextAlignment = "**center**"	
TinyWebDB	TinyWebDB1		

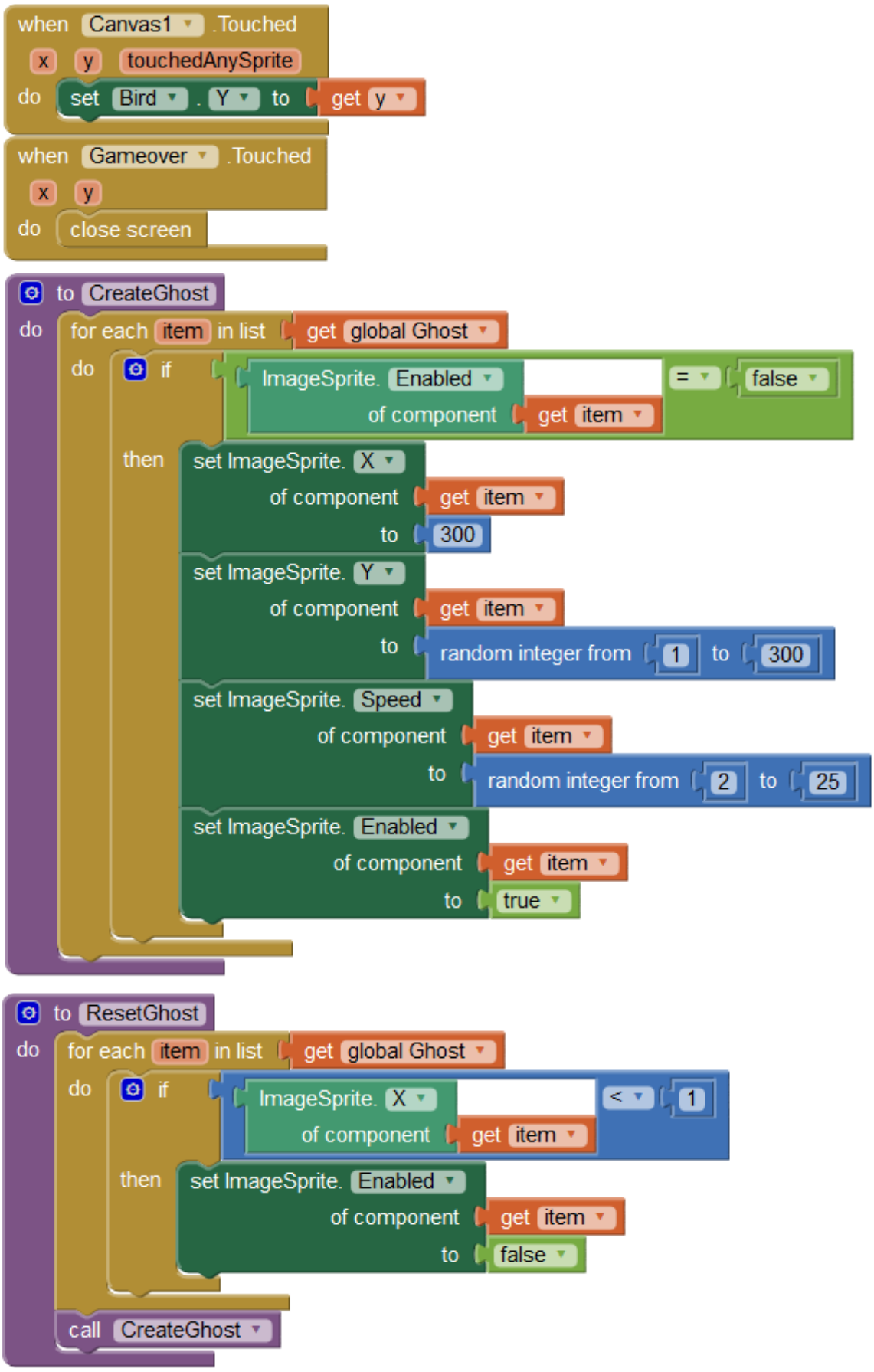
6.1.4 Block Configuration

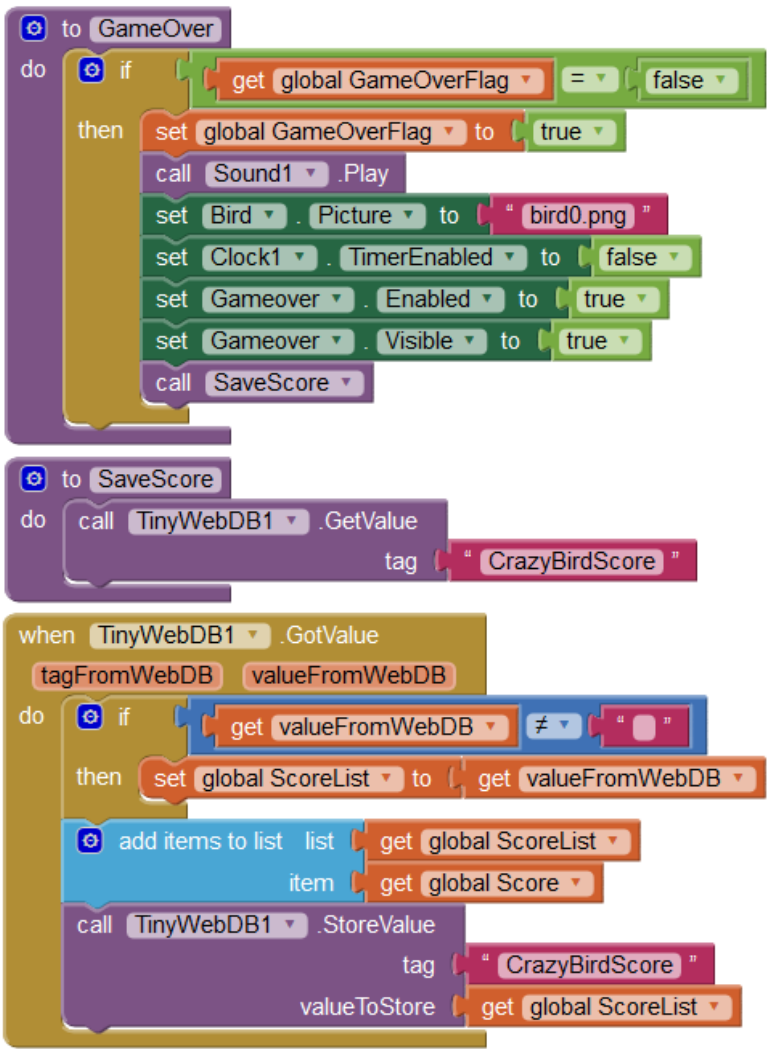
6.1.4.1 Screen1



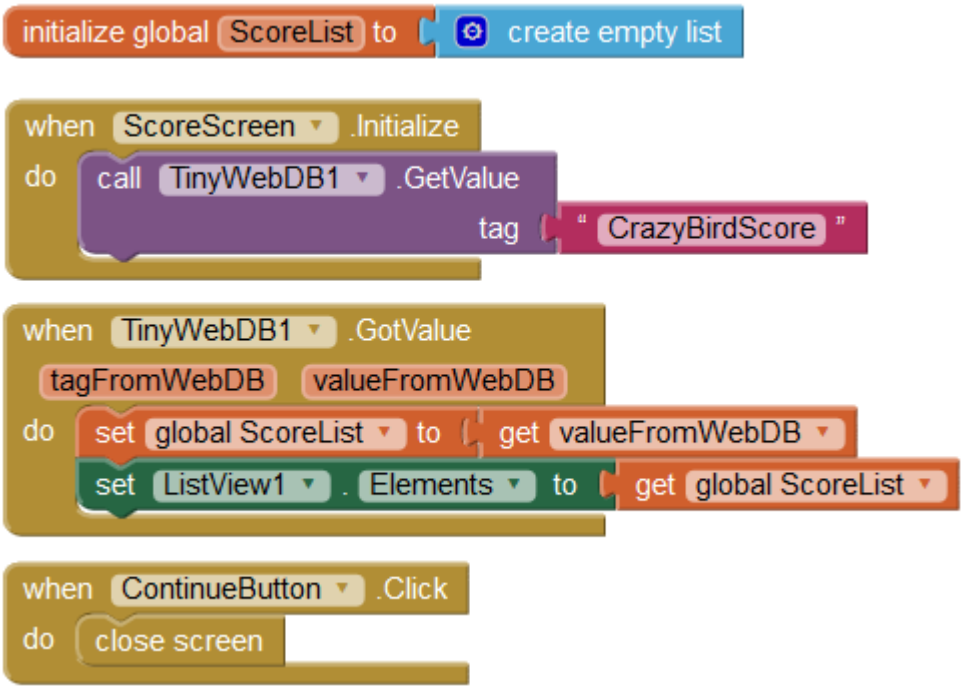
6.1.4.2 GameScreen




when Canvas1 .Touched
x y touchedAnySprite
do set Bird . Y to get y
when Gameover .Touched
x y
do close screen
to CreateGhost
do for each item in list get global Ghost
do if ImageSprite. Enabled = false
of component get item
then set ImageSprite. X
of component get item
to 300
set ImageSprite. Y
of component get item
to random integer from 1 to 300
set ImageSprite. Speed
of component get item
to random integer from 2 to 25
set ImageSprite. Enabled
of component get item
to true
to ResetGhost
do for each item in list get global Ghost
do if ImageSprite. X < 1
of component get item
then set ImageSprite. Enabled
of component get item
to false
call CreateGhost



6.1.4.3 ScoreScreen



6.1.5 Sample Output