Final Year Project Group Report

Smartphone Accessories Creation Platform

Using Bluetooth Low Energy Devices

Supervisor: Dr. Vincent Lau

Arpit Brij Gupta (UID: 2012525208)

Debopam Sengupta (UID: 2012580002)

Suyash Agarwal (UID: 2012529694)

Yatharth Sharma (UID: 2012515837)

Table of Contents

Section	Title	Page No.
1	Project Objective	5
1.1	Problem Statement	5
1.2	The Current Market Solution	6
1.3	Project Solutions	7
1.3.1	Suggested Solutions	8
1.3.2	Final Solution	8
1.4	Refined Objectives	9
2	Project Background	11
2.1	Bluetooth Smart	12
2.1.1	Introduction to Bluetooth 4.0	12
2.1.2	Development with Bluetooth Smart	12
2.1.3	Technical Aspects of Bluetooth Smart	12
2.2	Texas Instruments SensorTag	14
2.2.1	Sensors	14
2.2.2	Applications of SensorTag	15
2.3	MIDBot iCard	16
2.3.1	Services	17
2.3.2	Applications of iCard	17
2.4	Application Development Platform	17
2.4.1	Catrobat	18
2.4.2	Pocket Code (Catroid)	18
2.5	Event-Driven Programming	21
2.6	Parse Cloud Platform	22
2.7	GitHub	23
3	Project Methodology	25
3.1	The Process	25
3.1.1	Risk-Driven Development	25
3.2	Use Case Model	26
3.2.1	Primary Actor(s)	30
3.2.2	Secondary Actors	30
3.3	Primary Use Case Flowchart	31
3.3.1	Use Case Analysis	32
3.4	Proximity Devices Flowchart	33
3.4.1	Use Case Analysis	34
3.5	Planned Modification of Pocket Code	34

4	Project Schedule and Milestones	36
5	Mid-Term Progress	37
5.1	Analysis and Research	37
5.1.1	Catroid Source Code	37
5.1.2	Android APIs for Bluetooth 4.0	39
5.2	Initial Prototype	40
5.3	Modified Application	41
5.3.1	Front-end Development	41
5.3.2	Back-end Development	44
5.3.3	Sample/Example Apps	45
5.4	Risks Mitigated Until Mid-Term	48
5.5	Second Phase Plans	49
5.5.1	Road-map to Completion	49
5.5.2	Identified Drawbacks in Pocket Code	51
5.5.3	Risks to be Mitigated	52
5.5.4	Top Priority Tasks	54
6	Final Stage Development	55
6.1	Sharing Projects Using Parse	55
6.2	Multiple BLE Devices	58
6.3	Event-Driven Execution - the "*When*Brick"	61
6.4	Display Console Text	63
6.5	Interactions with MIDBot iCard	65
6.6	Finding BLE Devices in Proximity	67
6.7	Migration from Catroid to BLEnCode	69
7	User Guide	76
7.1	Creating Your First App!	77
7.2	Creating an App with BLE Sensors	82
7.3	Using the In-App Tutorials	85
7.4	Creating Applications which use Proximity Sensor	85
7.5	Sharing and Downloading Projects from Cloud	87

8	Pilot User Reviews	90
8.1	Familiarity with BLE	90
8.2	Usefulness of BLE to Apps	91
8.3	Most Useful Sensor on SensorTag	91
8.4	Least Useful Sensor on SensorTag	92
8.5	More Sensors?	92
8.6	MIDBot iCard Convenience	93
8.7	TI SensorTag Convenience	93
8.8	Ease of Use of Bricks	94
8.9	In-App Tutorials	94
8.10	Overall Lerning Experience	95
9	Potential Applications	96
10	Future Developments	98
10.1	How to Extend Our Project	98
11	BLEnCode - Legal	100
	References	107

1. Project Objective

1.1 Problem Statement

The task at hand is divided in to two broad spectrums which affect two different user groups. In one instance, we wish to make application development easier and readily available for younger audience (12-15 years). It is widely accepted that learning how to program a computer and learning how to break down a problem into smaller components is a highly valued skill in today's generation. Many times, we tend to ignore the importance of learning some basic skills from other domains that are beyond our studies that might help us gain a competitive edge over our peers. In words of Steve Jobs, "Everybody in this country should learn how to program a computer… because it teaches you how to think." And this is true indeed.

In essence, majority of computer programming is problem solving using novel techniques and tools. An early start in this field is vital for professional and personal growth. Main problem with such endeavors is that they aim to teach programming in a confined and conventional manner which includes highly technical software for teaching purposes (elaborate IDEs and high level languages). This might work for an older age group but for younger kids, this might be overbearing. We have analyzed this problem and through our final year project we aim to implement a viable solution.

On the other hand, another aspect of our project is to investigate the role of smartphone accessories in application development. In conventional use, smartphone accessories are either used to push data from the smartphone and simply give output (e.g. watch, bracelet etc.) or are used to input data and use them in a very restricted manner (e.g. pedometer for some calculations). There is not much flexibility in this area where much can be done by utilizing the data from smartphone accessories. Also, customizable application development is not convenient as of now and is not an easy task to undertake. We aim to extend/create such functionality in an application development platform that will allow for the inclusion of Bluetooth Sensor data in the development logic.

1.2 The Current Market Solution

The current visual programming tools to help extend application development to younger/aspiring developers are widely used in the industry today. Some of them are listed below:

- Scratch – Scratch is a free desktop and online multimedia authoring tool that can be used by students and teachers to easily create customized programs and provide a stepping stone to the more advanced world of computer programming. Scratch allows users to use event driven programming with multiple active objects called "sprites". Sprites can be drawn – as either vector or bitmap graphics – from scratch in a simple editor that is part of the Scratch, or can be imported from external sources, including webcam.

- MIT App Inventor – App Inventor for Android is an application originally provided by Google and now maintained by the Massachusetts Institute of Technology. It allows anyone, including people unfamiliar with computer programming, to create software applications for the Android operating system. It uses a graphical interface, very similar to Scratch and the StarLogo TNG user interface, that allowes users to drag-and-drop visual objects to create an application that can run on the Android system, which runs on many mobile devices.

- Alice – Alice is an innovative 3D programming environment that makes it easy to create an animation for telling a story, playing an interactive game, or a video to share on the web. Alice is a teaching tool for introductory computing. It uses 3D graphics and a drag-and-drop interface to facilitate a more engaging, less frustrating first programming experience.

- Stencyl – Stencyl is a game creation platform that allows users to create 2D video games for computers, mobile devices, and the web. Stencyl uses a highly intuitive block-snapping interface inspired by the popular MIT Scratch project, which has proven to be an effective teaching model with children as young as 6.

- Pocket Code – A visual programming language and app for Android smartphones and tablets. It is inspired by Scratch and developed by the Catrobat team as free open source software.

1.3 Project Solutions

During the Inception phase of the project, we brainstormed about possible ideas/implementations as solutions for the identified objectives, as well as the problem statement. We agreed that an optimal solution should include an intuitive user interface that reduces the difficulty and effort spent on understanding basic operations of the platform. It should also avoid very complicated syntax so that the development/programming/code is easy for younger users to grasp and understand – this can be done by using everyday usage terms, for example "Forever" instead of a "`while(true)`" for loops.

The toys and action figures of the previous generation have today been replaced by the smartphones of today – more and more kids are using smartphones to spend their free time. We want to take this as an opportunity (as have many mobile application developers) to create a programming platform for mobile devices (smartphones) rather than desktop computers. Moreover, we needed to solve the second aspect of the problem statement that to increase flexibility in using data provided by Bluetooth devices as smartphone accessories. The solution would involve integration of the programming platform with Bluetooth devices to enhance flexibility in manipulating the data sent to them, and that received from them.

These integrative solutions would also help supplement our initial objective of pitching the interest for application development among younger users. The younger generation is very proactive in picking up the nuances of modern technological gadgets, and since they are very easy to use, that is why the younger generation is very interested in learning more about them. The modern gadgets allow for a virtually infinite set of possibilities as far as the younger generation's creativity is concerned, and this only encourages them to delve deeper into the concept – like modern-day LEGO, kids are encouraged to experiment and understand practically the actual working of a product. These gadgets and devices involve a major degree of wireless

communication (mostly using Bluetooth), and if we could incorporate the learning of programming within the use of these interactive and fun accessories, it could be a breakthrough in terms of how to generate interest of young learners towards coding. To tap into the wireless communication aspect of modern gadgets, Bluetooth Low Energy (a.k.a. Bluetooth Smart) is the optimal choice to begin new development practices because it is the most power-efficient technology in the market and current as per modern day available resources.

1.3.1 Suggested Solutions

When it comes down to implementation, we had three main ideas to follow through with.

1. Build our own visual programming language – Create a simplistic visual programming language which can be utilized with a smartphone to allow simple programming environments which rely on an intuitive user Interface.

2. Migrate Scratch to mobile – Partner with Scratch team at MIT for us to develop a mobile version for their web software and then extend its capabilities to include easy interaction with Bluetooth devices.

3. Enhance Pocket Code – Leverage the existing open-source Pocket Code application which utilizes Catrobat visual programming language to create our own open-source project which will allow us to implement our project goals by modifying its source code.

1.3.2 Final Solution

After careful research and consultation with the professor, we decided to go with the third solution – to enhance Pocket Code.

The first idea of building our own visual programming language was eliminated because it would not be very productive on our end, since a lot of other visual programming platforms (such as Scratch) already exist across the industry. Examples of these are listed in Section 1.2

The second idea of migrating Scratch to mobile devices was eliminated primarily because it is not open-source, and significant effort would be needed to partner with MIT and entirely re-develop Scratch for mobile devices, whereas Pocket Code – an open source platform that is inspired from Scratch itself – already exists as a visual programming tool for Android devices.

1.4 Refined Objectives

The goal of this project is to extend the capabilities of Pocket Code into a customizable, generic, user-friendly Smartphone application that can interact with BLE devices. The extended version of Pocket Code will have support for all the sensors in Sensor Tag and will serve as a model for young and novice app developers (who are new to the concept) as a way of seamlessly integrating external sensors within their app with the help of Pocket Code. This gives birth to a new practice in software development where peripherals are used to receive meaningful data for a mobile application contrary to majority Bluetooth peripherals in the market today which are used primarily for data output (watches, speakers etc.)

The project will allow users to use drag-and-drop UI widgets and instructions to create their own custom scripts or programs that will be sent to the System-on-Chip on Sensor Tag as instruction sets to access and retrieve data from the selected sensors.

We understand that creativity is a major part of application development and younger age group (12-15 years) is better equipped with this skill. We want to extend ready help and support to beginners who want to learn how to build their own mobile application and implement their creativity. The existing Pocket Code application comes with online tutorials that may help a user understand in-app navigation but they give little explanation about logical components of the script itself. These tutorials are not available within the application and a user needs to change entire focus from development to online learning. We aim to incorporate the concept of “In-app help” for such users whereby they will have access to help materials which will introduce, explain and give examples about the logical components that are required to make an application. This convenient option will not require the user to go through the complex process of leaving the application and changing the focus from development to begin

an entire new learning process. The in-app help will supplement the development and learning of the user by giving succinct explanations and examples just a tap away, while retaining the user's focus on the ongoing development.

Also, users will be able to use the drag-and-drop interface to use the data from Sensor Tag in various ways to create their own applications. For example, a user might create a game that can utilize some particular sensors (say Accelerometer) and integrate them in their own UI for presentation purposes (say a car moves left on the screen when the Sensor Tag is moved left). Since the UI will be based on Pocket Code, all development-related functionalities of Pocket Code can be extended to the Smartphone application, integrating the interface from Sensor Tag within it.

As a further enhancement, since the project is open source and encourages collaborative creativity, we will offer users the ability to share their projects with others. For this purpose, a database on our own server will be provided. Users can choose to either share entire projects with particular users or with the whole Pocket Code BLE community by uploading their projects to our database.

2. Project Background

The motivation for this project is to allow any user to improve and expand the capability of their Smartphone using Bluetooth Low Energy (BLE a.k.a. Bluetooth Smart) technology and involving sensor chips to collect real time data for various intents and purposes. Customizability is our main vision for developing an easy-to-use platform to write data to and read data from BLE devices like the Sensor Tag from Texas Instruments.

Software related to our project, which are available in the market, includes the following:-

- Texas Instruments mobile applications on iOS and Android platforms to interact with and receive data from the BLE Sensor Tag, over Bluetooth.
- Pocket Code, based on Catrobat, is an on-device visual programming system for Android devices. Catrobat is a visual programming language and set of creativity tools for Smartphones, tablets, and mobile browsers. Catrobat programs can be written by using the Pocket Code programming system on Android phones and tablets.

This project will open up a new platform for people to utilize the Texas Instruments Sensor Tag, and moreover, utilize BLE technology in general. The project will enhance the utility of the TI Sensor Tag which can be used for personalized applications for dedicated purposes other than just as a general platform for sensor information.

The Smartphone platform that we will build as part of this project will involve enhancements to the currently existing Pocket Code and can act as a customizable monitoring device. Users can define the sensors involved as well as the application logic to read data from them over a specific, customizable time-period. Such an application would allow BLE devices to act as data reporting agents to any application made using the enhanced Pocket Code.

2.1 Bluetooth Smart

2.1.1 Introduction to Bluetooth 4.0

Bluetooth Smart (Bluetooth 4.0) is the latest and most power-efficient version of the Bluetooth wireless technology. While the power-efficiency of Bluetooth Smart is very useful for devices that do not have a very long lasting battery life, another very important feature is its ability to work with an application on the smartphone or tablet you already own. Classic Bluetooth provided effective means of communications between devices and this new version is a much larger step to ease that communication. Bluetooth technology is widely used in several industries like consumer electronics, PC peripherals and automotive. Many other industries also seek to integrate the new technology because of its power efficiency and ability to connect to smartphone apps — it is the perfect fit for a wide range of devices from heart-rate monitors to cycling computers, as well as BLE Sensors like the Sensor Tag by Texas Instruments.

2.1.2 Development with Bluetooth Smart

According to the official Bluetooth website, "Bluetooth Smart is an application-friendly technology supported by every major operating system. The technology costs less and offers flexible development architecture for creating applications to bring everyday objects like heart-rate monitors, toothbrushes, and shoes into the connected world and have them communicate with applications that reside on the Bluetooth Smart compatible smartphones, tablets, or similar devices those consumers already own. This means Bluetooth Smart developers are limited only by their imagination."

2.1.3 Technical Aspects of Bluetooth Smart

Bluetooth Smart extends the use of Bluetooth wireless technology to devices that are powered by small, coin-cell batteries much like the ones used in the SensorTag. Bluetooth's official website claims that in many cases, Bluetooth 4.0 makes it possible to operate these devices for more than a year without recharging. Key technical features of Bluetooth Smart are:

- Ultra-low peak, average and idle mode power consumption
- Ability to run for years on standard coin-cell batteries
- Lower implementation costs
- Multi-vendor interoperability
- Enhanced range (can be optimized to 200 feet)

Technical details from the official Bluetooth website, relevant to our project include the following:

- Data Transfers – Bluetooth Smart (low energy) supports very short data packets (8 octet minimum up to 27 octets maximum) that are transferred at 1 Mbps. All connections use advanced sniff-sub rating to achieve ultra low duty cycles
- Host Control – Bluetooth Smart (low energy) places a significant amount of intelligence in the controller, which allows the host to sleep for longer periods of time and be woken up by the controller only when the host needs to perform some action. This allows for the greatest current savings since the host is assumed to consume more power than the controller
- Latency - Bluetooth Smart (low energy) can support connection setup and data transfer as low as 3ms, allowing an application to form a connection and then transfer authenticated data in few milliseconds for a short communication burst before quickly tearing down the connection
- Range – Increased modulation index provides a possible range for Bluetooth Smart (low energy) of over 100 meters
- Strong Security – Full AES-128 encryption using CCM to provide strong encryption and authentication of data packets

2.2 Texas Instruments Sensor Tag

The award winning SimpleLink Bluetooth Smart SensorTag is designed to shorten the time and effort needed by developers to create Bluetooth apps from months to hours. The SensorTag is armed with six sensors which are controlled by a System-on-Chip, and these sensors provide several useful data to users and developers for use in their app.

2.2.1 Sensors

The sensors included are:

- IR temperature Sensor – uses Infra-Red radiation to monitor current temperature of the object it is pointed towards with typical +/- 1C accuracy.
- Humidity Sensor – The humidity sensor reports the relative humidity (%RH) and ambient temperature. This is an integrated 12-bit relative humidity sensor with a 14-bit temperature sensor.
- Pressure Sensor – The pressure sensor measures the barometric air pressure. This is a 16-bit pressure sensor with 16-bit temperature reading.
- Accelerometer – The accelerometer measures acceleration in 3 axes (X, Y, Z) with programmable resolution up to 14 bit. The accelerometer can also measure direction of gravity.
- Gyroscope – The gyroscope measures the rate of rotation in all 3 axes (X, Y, Z) with up to 16-bit resolution. Readings from the gyroscope can be combined with readings from the accelerometer to determine the orientation of the SensorTag in 3-D space.

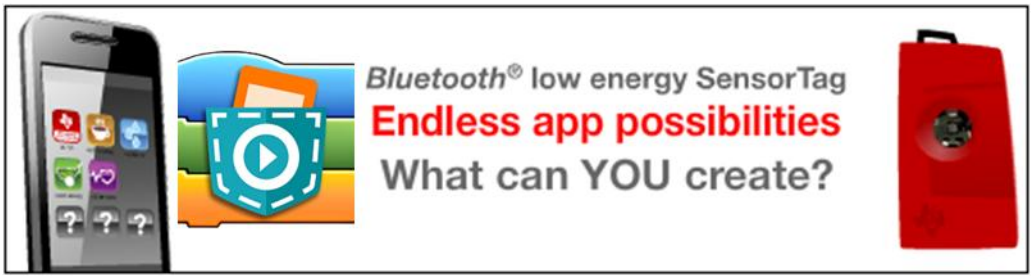
- Magnetometer – The Magnetometer measures the magnetic field in 3 axes (X, Y, Z). The data from the magnetometer is usually combined with an accelerometer to make a compass.

2.2.2 Applications of SensorTag

The versatility of the SensorTag means limitless app possibilities including those for health and fitness, medical, educational tools, toys, remote controls, mobile phone accessories, proximity and indoor locationing.

According to the Texas Instruments' website, users can now create "Endless 'appcessories'" with their SensorTag devices. The website also states "With a SensorTag App and no required hardware or software expertise, the kit removes the barriers to entry for smartphone app developers who want to take advantage of the growing number of Bluetooth Smart-enabled smartphones and tablets. The over-the-air download feature provides the ability to update the SensorTag firmware from a central device like a smartphone, tablet or PC. This simplifies the upgrade process and enables customers to develop new innovative features to existing Bluetooth Smart equipment to satisfy consumers' endless needs."

The SensorTag is the first Bluetooth Smart development kit focusing on wireless sensor applications and it is the only development kit targeting smartphone app developers.



2.3 MIDBot iCard

MIDbot™ iCard is a wide-range smart card. It can make a class to be more interactive with an app as there are buttons and LEDs that can feedback or be controlled during voting, selection, lucky draw or gaming.

2.3.1 Services

The iCard offers some interactive services which can be utilized by young developers and programmers:

- LED – The iCard has an LED whose color can be customized using RGB values. Also, the time period and flashing style of the LED can be customized. This is a 8-byte service.
- Buzzer – The iCard has a Buzzer service, which outputs sound at customizable frequency and loudness. This is a 6-byte service.

2.3.2 Applications of iCard

The MIDBot iCard can be used in various ways to create interactive apps. The Buzzer and LED can be easily used to create voting, lucky draw or quiz apps. Such interactive services allow for easy development for young developers.

The iCard can also be used in conjunction with other Bluetooth Smart devices like the SensorTag to move one step further towards an Internet of Things. In such cases apps can be built where the change in the value of a particular sensor of the SensorTag can sound the Buzzer or light up the LED of the iCard.

2.4 Application Development Platform

The Application Development platform for our project needed to satisfy to broad purposes. It should be easy to use for younger audience who want to understand and learn programming. It should also be open source and customizable so as to extend the existing capabilities to include BLE devices like TI SensorTag in the application development logic. After extensive research and discussions with supervisor, we chose to use Pocket Code for the above purposes. Brief descriptions of Pocket Code and its background are given in the following sections to introduce the application development platform.

2.4.1 Catrobat

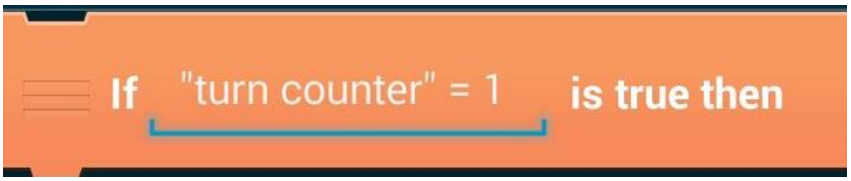
Catrobat is a visual programming language and utilizes simple drag-and-drop interface of Catroid to make short programs. This makes learning application development very intuitive, easy and interactive for a young user who is new to programming. Catrobat is open source and is available for developers to contribute and enhance/extend existing application capabilities to match other competitors including its inspiration project – Scratch. We have utilized the source code of Catroid available for developers on GitHub to extend the existing capabilities of Pocket Code to achieve our project objectives.

2.4.2 Pocket Code (Catroid)

The source code of Pocket Code (i.e. its project name) is known as Catroid. It is an on-device programming system for Android devices. It works as an IDE for Catrobat programming language. The user interface for Catroid is easy to use and uses intuitive actions to make an application. Existing features of Pocket Code include:

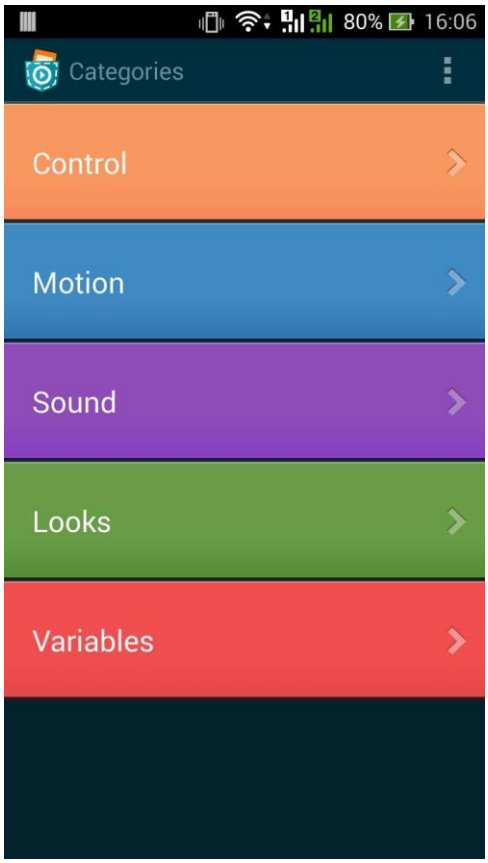
- Making projects which use images (from Camera or Pocket Paint) as objects.
- Basic motion control for objects.
- Basic programming paradigms like if-else, loops, conditionals etc. can easily be implemented in the script for any object for the program.
- Uploading a project to the developer community for discussion and sharing.

Pocket Code uses a Brick for forming the sequence of the user's application. The programming paradigms are provided to the user in the form of "Bricks", which the user can then drag-and-drop (one or more Bricks) to form the application logic. A combination of Bricks is executed sequentially to maintain the flow of the user's program. The following figure shows what a Brick looks like:



Pocket Code classifies the types of Bricks available into five main Brick Categories, and each category of Bricks has a different color. The categories are:

- Control: These Bricks manage the flow of control of the program, like “if-else” and loops.
- Motion: These Bricks determine where to place an object during the run-time of the program, like moving an object to a particular set of coordinates.
- Looks: These Bricks allow changing the look of an object during run-time of the program, like changing the background or hiding an existing image on the screen.
- Sounds: These Bricks manipulate the sound output from the device while the program is running, like saying a phrase such as “Hello World!”
- Variables: These Bricks maintain the values stored within program variables and also allow manipulations on them, like adding one to the value of a counter variable.



We also identified some limitations of Pocket Code which are being worked on in the Catrobat community:

- One major limitation is the inability to print text on screen. There is little support for making apps that may need to display some information as text.
- Any app made on Pocket Code does not generate an .apk file that can be universally run on any Android system. This problem is actively being pursued by the Catroid team and hopefully there should be a solution to this problem soon.
- The Help feature available for young developers using Pocket Code only redirects to the Pocket Code website using the mobile browser, making it ineffective if the user does not have an internet connection.

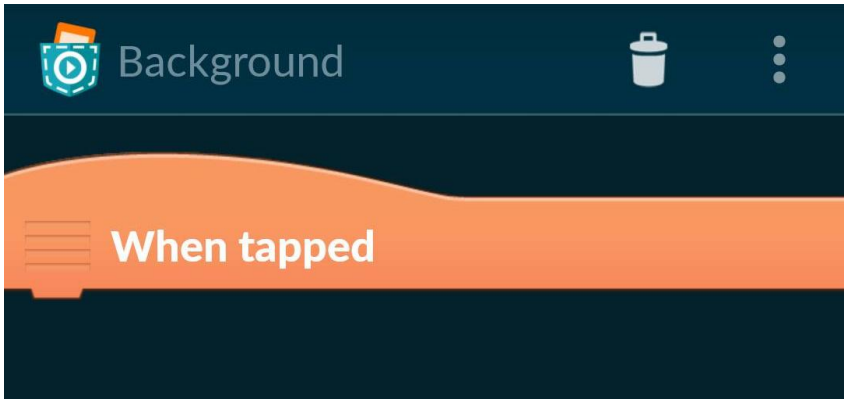
- Pocket Code is only compatible with Android operating system, which reduces its market potential. This is primarily because iOS does not support apps to make new apps.

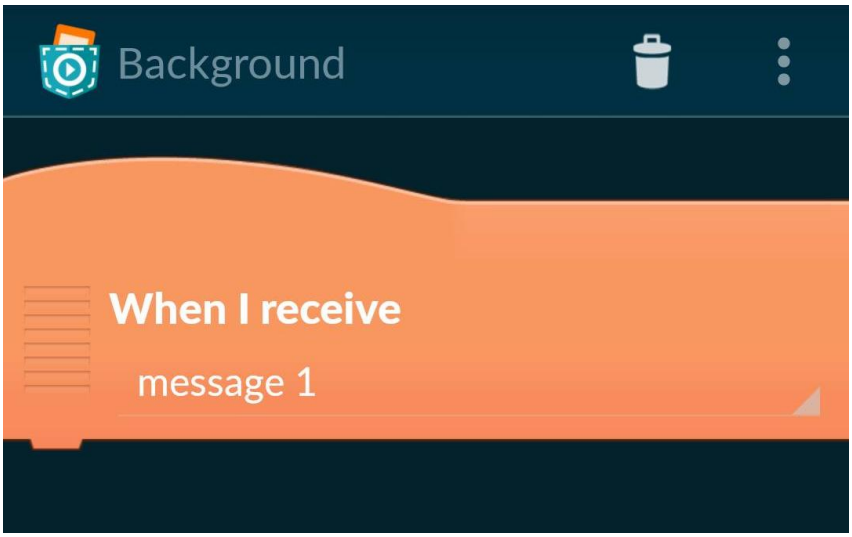
- There is currently no indentation to the Bricks added to a Script. As of now, all Bricks are placed one under the other. There is no indentation to highlight similar parts of the logic. So we will be modifying Pocket Code such that each If-else-if block is indented properly for easy debugging.

2.5 Event-driven programming

"*Event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (**mouse** clicks, key presses), sensor outputs, or messages from other programs/threads*." – Wikipedia

Currently event-driven programming in Pocket Code is minimal, only supporting a functionality that registers screen taps and particular message broadcast receivers within the app, as shown in the screenshots below.





We need to implement and register other events as callbacks to the application in BLEnCode for particular events coming in from the BLE devices that the app is connected to. Otherwise, to appropriately receive data from sensor devices as well as proximity data, the application will have to scan them continuously in a “forever” loop. This method of using callbacks for event-based execution avoids the need of such loops.

2.6 Parse Cloud Platform

To facilitate easy sharing of applications by Users, it is necessary to have a community where such creativity can be shared and other users can download and use shared apps. Pocket Code currently allows users to upload their apps to the Pocket Code community. However, for BLEnCode, a cloud-based platform was needed to store data about apps and users and make it available to other users.

Parse is a cloud-computing platform which powers apps across different platforms like iOS, Android, Windows Phone, etc. Parse provides the following functionalities which will be useful to create the BLEnCode community to start sharing apps:-

1. Save data in the cloud

   Parse handles everything needed to store data securely and efficiently in the cloud. Users can store basic data types, including locations and photos, and query across them without spinning up a single server.

2. Local Datastore

   Parse Local Datastore will allow BLEnCode to use all the querying and security features of Parse, even when there is no network connection on Android and iOS.

3. Built for Massive Scale

   Every aspect of Parse Core, from Data to Social and Cloud Code to Hosting, is designed with scalability in mind. Hundreds of thousands of apps access the Parse servers every minute, so as the BLEnCode community grows larger, storage and retrieval of large amounts of information will never be a hassle.

2.7 GitHub

"GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as wikis, task management, and bug tracking and feature requests for every project.

GitHub offers both paid plans for private repositories and free accounts, which are usually used to host open-source software projects. As of 2015, GitHub reports having over 9 million users and over 21.1 million repositories, making it the largest code hoster in the world." – Wikipedia

We plan to use the free services of GitHub for collaborative coding and as a Version Control System to manage our source code during the development of BLEnCode. GitHub also has various public repositories of developers using Java code to interact with the System-on-Chip of

the Texas Instruments SensorTag, which we can use in our project to integrate with Android APIs. The repository for Pocket Code (Catroid) is also publicly present on GitHub for developers to use and modify.

Another purpose for using GitHub is to promote further developments and enhancements to our own BLEnCode application. Developers can easily extend our project by following the guidelines and our source code – both of which are going to be available on our GitHub project repository.

3. Project Methodology

For the purposes of this project, we have made milestones for various parts of the project. In the given methodology, we have attempted to elaborate upon the following: use-case analysis; a detailed explanation of the functionalities to be extended in Pocket Code app, and some insight to the enhancement strategy for the same.

3.1 The Process

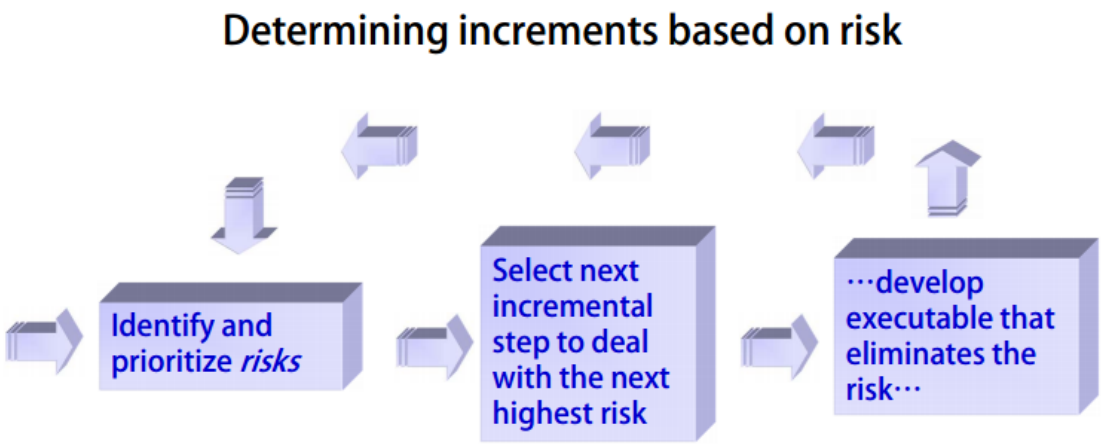
The initial guidelines/scope provided to us was very broad, including modification of Pocket Code to make it friendlier for young users and also modification of the TI SensorTag to introduce different features in the System-on-Chip like local storage of data.

Hence, the project started out rather cloudy until the scope was refined to focus on the modification of Pocket Code. After some important tasks and objectives of the project, such as integrating Pocket Code with TI SensorTag using Bluetooth Smart, were clearly understood, we did our initial research on the topics to determine the risks involved in the project.

From then on, the Risk-Driven Development process was followed.

3.1.1 Risk-Driven Development

Risk is a major driver in our project and we aim to mitigate the biggest risks earliest in our development cycle. The steps in this process are shown in this diagram:



The biggest risks identified were:

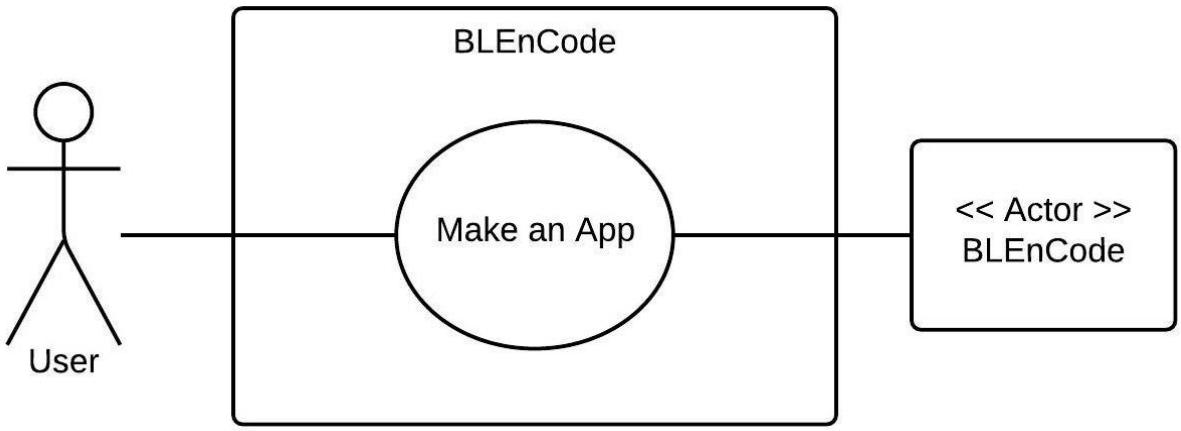
- Understanding Pocket Code source code (Catroid)
- Modifying Catroid to connect to, send instructions to and read data from SensorTag.
- Creating our own server and database to facilitate sharing of Pocket Code apps between users.
- Integrating the modified Pocket Code with the local offline storage system on TI SensorTag System-on-Chip implemented by Globalactive Technologies Ltd.

3.2 Use Case Model

Our use-case analysis conforms to the UML type representation. We have tried to represent the working of the application using a flowchart that details the various steps involved in the execution of the application. To incorporate data connections with BLE devices such as the Sensor Tag, we will modify the existing Pocket Code application to interact with the sensors, as detailed in this methodology.

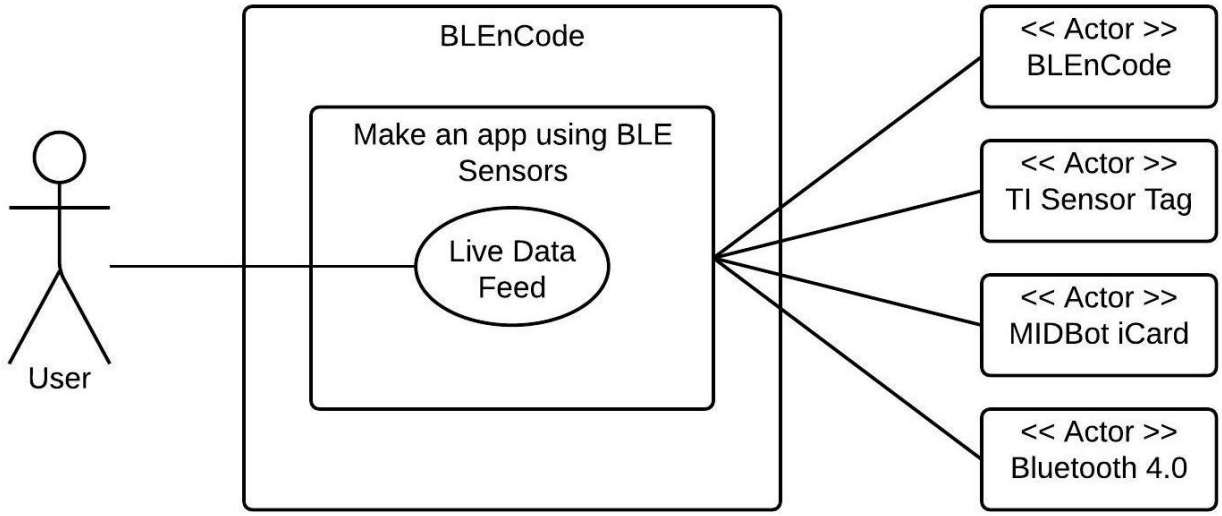
We have identified the following Use Cases for our application:

1. Make a new App using visual programming



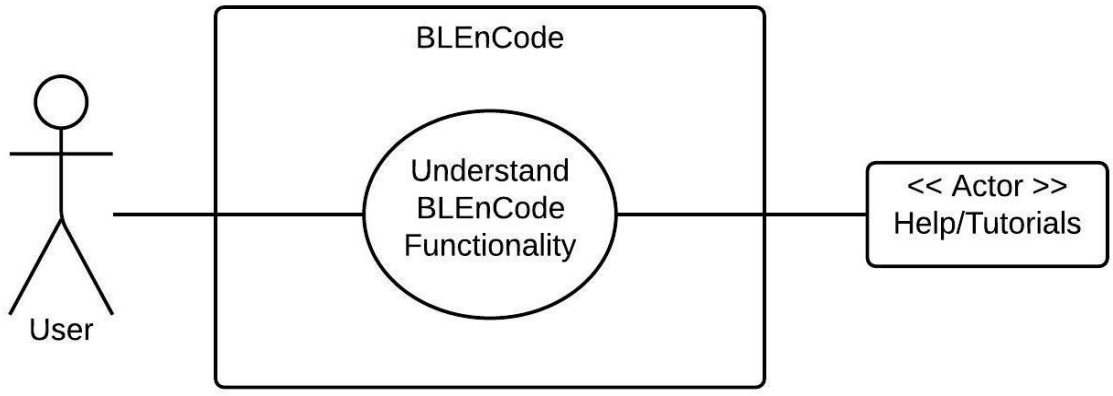
The first use case is to allow users to use existing Pocket Code functionality along with some of the enhancements we would introduce to create regular apps, which may not interact with any Bluetooth devices. One of the risks in this use case was identified to be to perform continuous regression testing to ensure that none of the new modifications would break existing Pocket Code features.

2. Make a new app using BLE Sensors to receive live data feeds



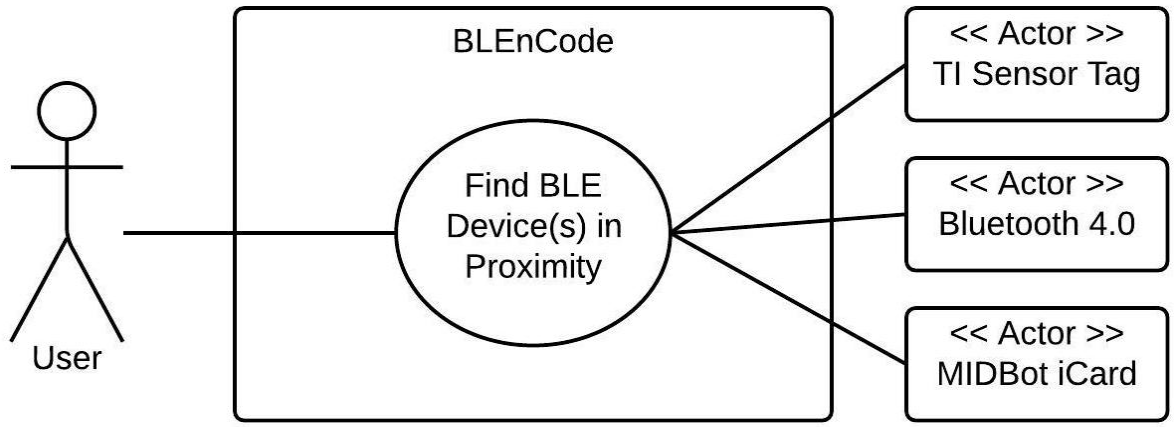
Our primary use case is to allow users to connect to Bluetooth devices through our app BLEnCode, to access live data from BLE devices and use it in their apps. The Bluetooth devices supported are TI SensorTag and MIDBot iCard. Bluetooth 4.0 APIs for Android are extensively used to support this use case.

3. Using tutorials to understand functionality



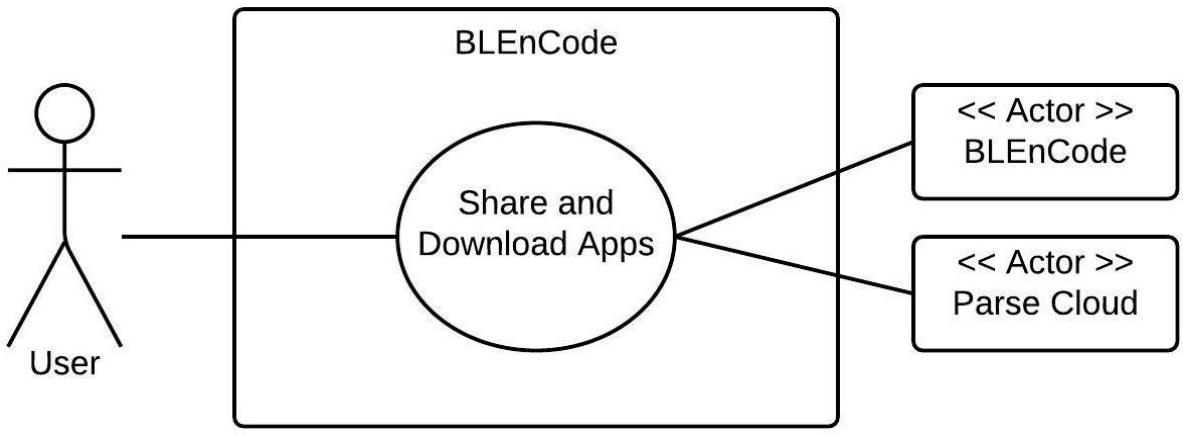
Another use case for our application is to allow young developers (aged 12-15 years) to learn programming in an easy format. This requires us to provide them with in-app tutorials throughout so that our audience can understand some of the basics of programming (like loops, conditionals, etc.) easily. Also, new technologies like Bluetooth and related functionality can be introduced through such tutorials.

4. Use RSSI to find BLE devices in proximity



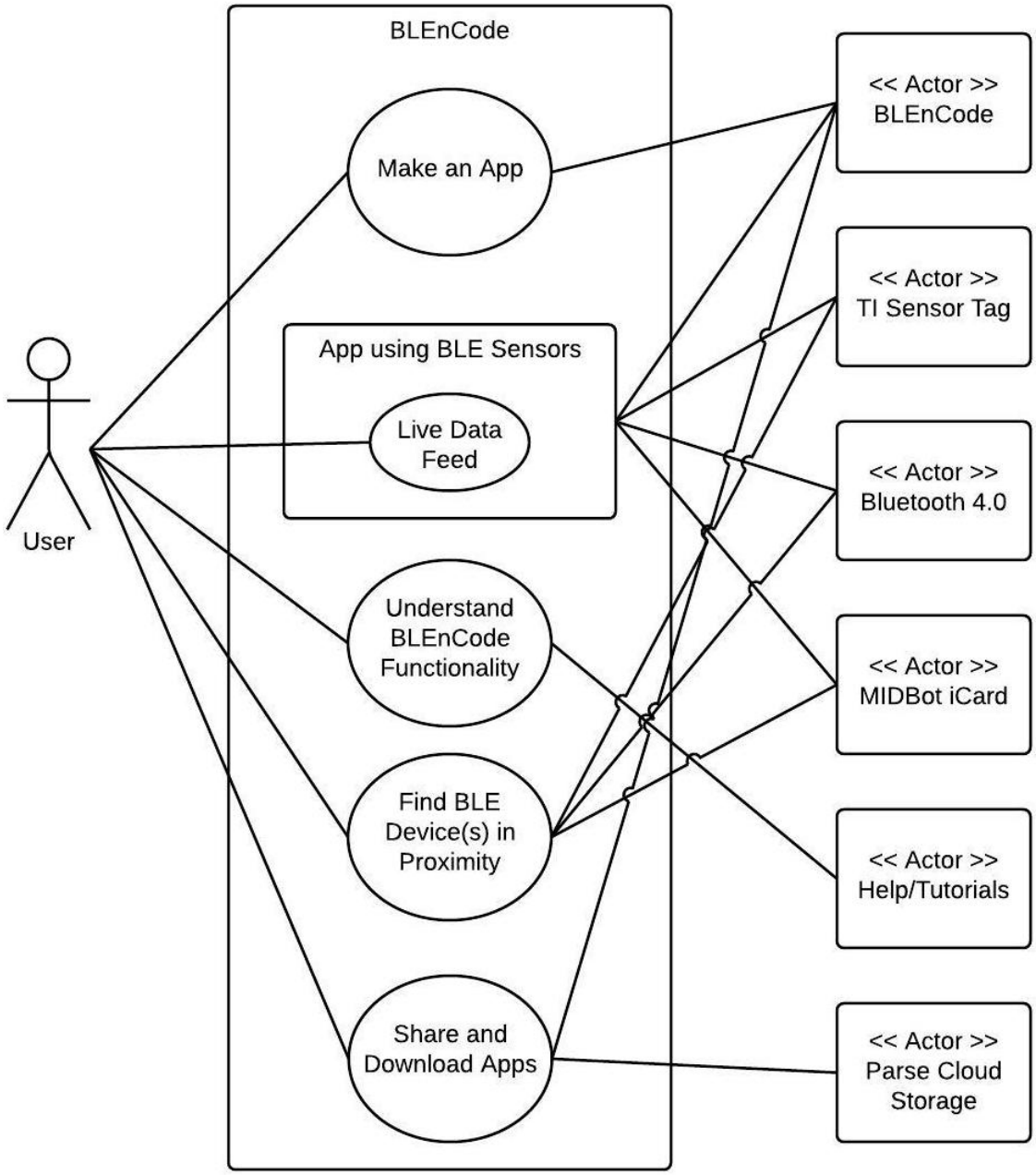
An important use case for our project was to make use of Bluetooth 4.0 technology to use the aforementioned Bluetooth devices as Proximity Sensors through signal strength. This would accomplish two goals. It would allow users extended freedom to use their BLE devices in a connectionless state through their application. Also, it would allow young developers to learn about new concepts like MAC Addresses and RSSI values of such devices.

5. Sharing your own and downloading other users' projects



The final use case for BLEnCode is to allow users to share their creativity by allowing them to share the apps that they create on a cloud platform, where all other users will be able to see and download their apps onto their local devices. The cloud platform is built using Parse cloud computing service.

The complete, comprehensive use-case model for our entire project is shown in the following diagram:



3.2.1 Primary Actor(s)

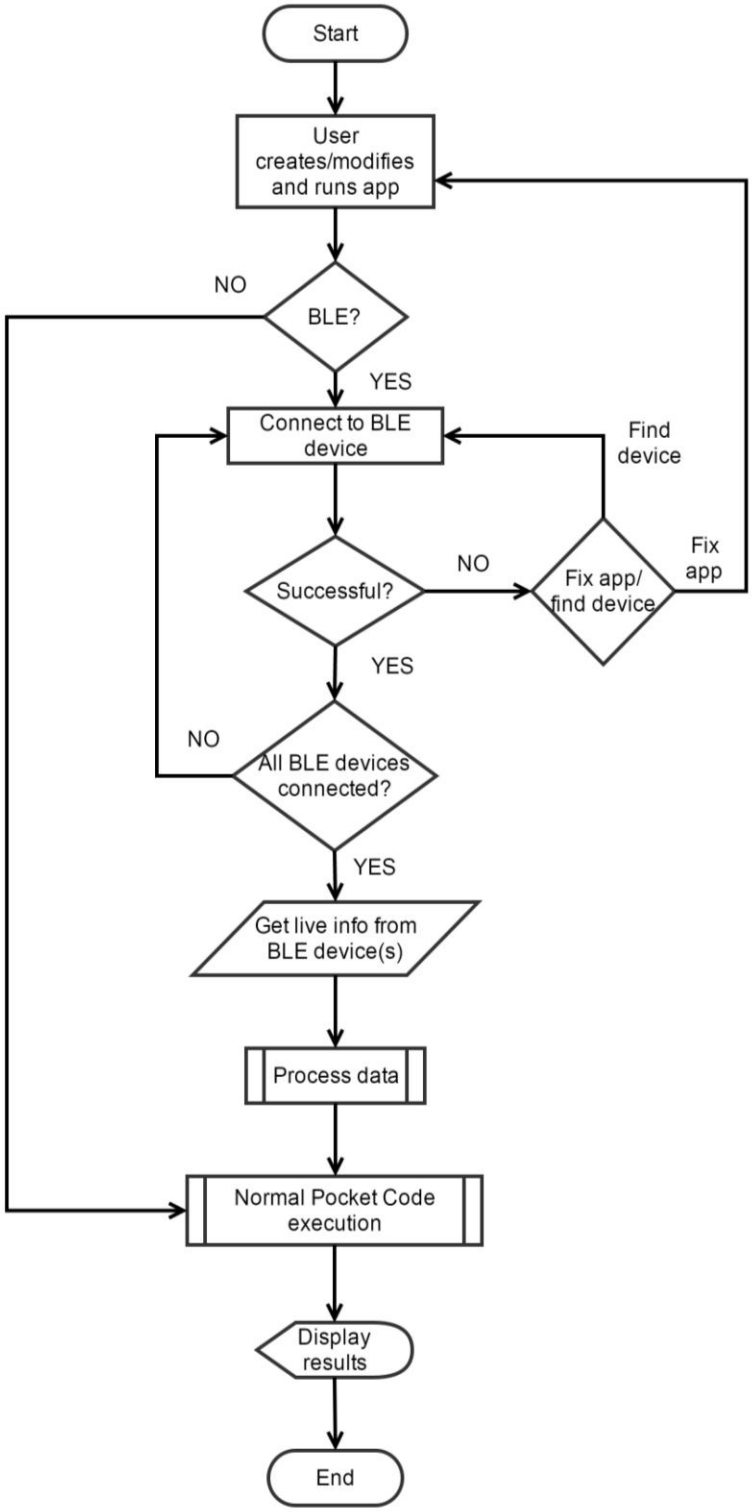
As shown in the UML diagram, the primary actor for our application is the Smartphone app user – including a wide range of audience, children, adults or developers. The User is our primary stakeholder, interested in making a customized application using the drag-and-drop interface on their Smartphones using BLEnCode. The User may also want to use data from BLE devices like the Sensor Tag and incorporate them in the application logic in their Pocket Code applications. This can be done seamlessly within the Pocket Code's original drag-and-drop user interface – as explained in the use-case descriptions above. The User wants to run their custom-built application (games, tools, etc.) on their smart device which utilize and/or display computations and/or animations based on the sensor data and app logic. Finally, the User can also share their own project/application with other users in the BLEnCode community.

3.2.2 Secondary Actors

The user will have ready access to help materials and tutorials from within the app that will explain basics like how to use a particular 'Brick' or 'Sensor'. This will be particularly useful when dealing with sensor bricks as their initialization and logic manipulation might not be very intuitive for all users. This help material will also be very useful for younger users who are new to programming and aspire to learn more about application development in their formative years.

The other secondary actors in our use case are the Pocket Code engine (Catrobat), Bluetooth 4.0 framework, the Texas Instruments Sensor Tag, the MIDBot iCard and the Parse cloud computing database framework. The Bluetooth 4.0 framework supports applications acting as a sender or receiver of data while communicating with classic Bluetooth or BLE devices. The Sensor Tag's System-on-Chip houses various sensors that can interact with the Smartphone (and by extension, BLEnCode) through the Bluetooth Smart framework. The interactions with the MIDBot iCard work similarly using Bluetooth Smart. Parse Cloud storage allows users to share their projects with the community.

3.3 Primary Use Case Flowchart



3.3.1 Use Case Analysis

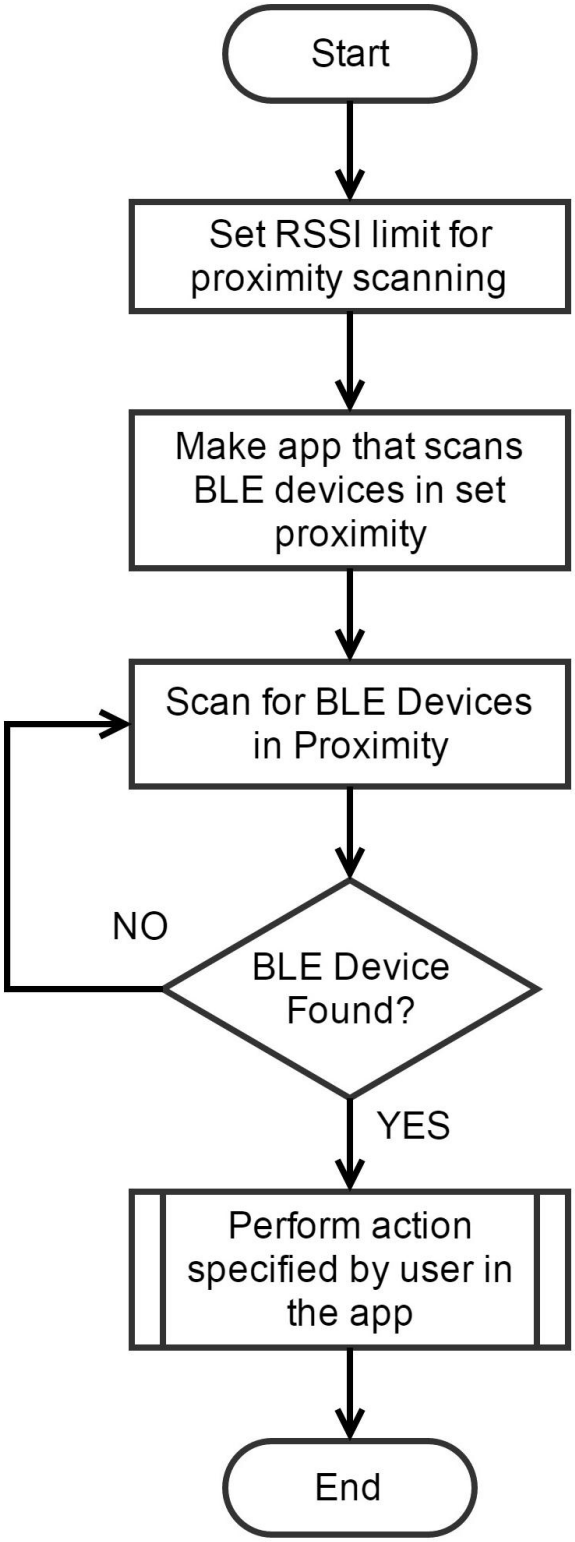
Once the user opens BLEnCode, he/she will have the following options:

- Create a new application
- Modify an existing app
- Run a previously created app
- Explore projects by others on the community
- Upload their application to the Pocket Code community

The flowchart above illustrates the primary backend/application process for creating/modifying an app on BLEnCode. When the user wants to create a new app or modify an existing one, he sees the drag-and-drop user interface. Here, he can combine widgets called “Bricks” with different functionalities like flow-of-control, inserting sound effects, controlling motion of objects, using sensors from SensorTag and/or outputting to MIDBot iCard to create applications.

If the application does not utilize BLE devices, then the BLEnCode engine will run the program. However, if the application uses the extended functionality by using BLE devices then the Smartphone tries to connect to all specified BLE devices, and once all the connections are successful the application activates the relevant sensors on the devices to get live feed from the connected SensorTags and/or MIDBot iCards or any other user-defined device if the user can register the appropriate callbacks within the application source code. During this time, the application either performs the user-specified actions, or else it stands-by to receive data that is pushed from the SensorTags in real-time.

3.4 Proximity Devices Flowchart



3.4.1 Use Case Analysis

To locate BLE devices in proximity, the User will need to set a limit for the RSSI value, which will indicate whether a device is “in range”. A default value will be set for new users who might be unfamiliar with the concept of RSSI. Either using event-driven programming, or a “forever” loop to continuously scan for devices, when a specified device is found by the application, it proceeds to execute the actions specified by the user in the application script.

A foreseen risk in this particular scenario is that RSSI varies logarithmically with distance. Also, the fact that RSSI values are unstable, often due to the surrounding environment, could prevent accurate distance calculations and locationing. Also, another major challenge could be to continuously receive RSSI data from all BLE devices within the specified proximity. Extensive research into Bluetooth 4.0 APIs for Android would be needed to fulfill this use case.

3.5 Planned modification of Pocket Code

The current Pocket Code version does not support any Bluetooth functionality. Thus the planned modifications of Pocket Code will include:

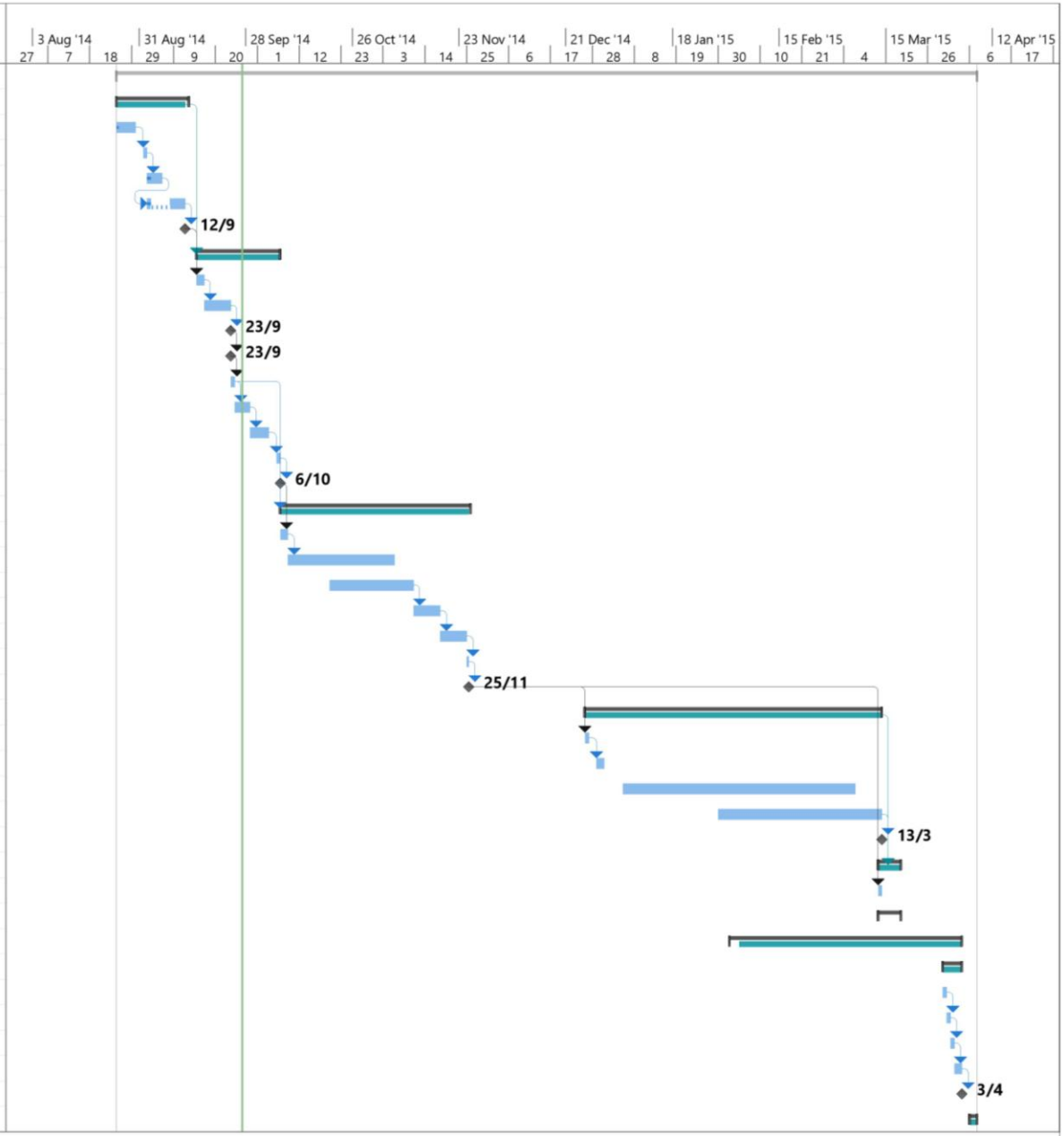
- Creating new Catrobat Bricks on Pocket Code to allow connection to SensorTag.
- Allowing users to monitor selected sensors from the SensorTag.
- Parsing raw data from SensorTag to provide easily-understandable data.
- Making sensor values easily available to users for use in their apps in various ways.
- In-app Tutorials to facilitate quick learning for young developers.
- Facilitating easy sharing of Pocket Code apps amongst users.
- Connecting multiple BLE devices to a single Pocket Code app, including a limited maximum of SensorTags and MIDBot iCards.

- Event-driven program execution style.
- Provision to be able to display console text (similar to “echo”).
- Locating BLE devices in proximity using RSSI.

The modifications on Pocket Code will be performed incrementally, and each modification will be extensively tested using prototype applications before the next improvements are made. This will include understanding and then changing the Java source code of Catroid to incorporate the planned changes.

4. Project Schedule and Milestones

ID	Task Mode	Task Name	Duration	Start	Finish
0		**Software Development**	**162 days**	**Mon 25/8/14**	**Tue 7/4/15**
1		**Scope**	**15 days**	**Mon 25/8/14**	**Fri 12/9/14**
2		Determine project scope	1 wk	Mon 25/8/14	Fri 29/8/14
3		Secure project sponsorship	1 day	Mon 1/9/14	Mon 1/9/14
4		Define preliminary	4 days	Tue 2/9/14	Fri 5/9/14
5		Secure core resources	1 wk	Tue 2/9/14	Thu 11/9/14
6		Scope complete	0 days	Fri 12/9/14	Fri 12/9/14
7		**Analysis/Software**	**16 days**	**Mon 15/9/14**	**Mon 6/10/14**
8		Conduct needs analysis	2 days	Mon 15/9/14	Tue 16/9/14
9		Draft preliminary software	5 days	Wed 17/9/14	Tue 23/9/14
10		Develop preliminary	0 days	Tue 23/9/14	Tue 23/9/14
11		Review software	0 days	Tue 23/9/14	Tue 23/9/14
12		Incorporate feedback on	1 day	Wed 24/9/14	Wed 24/9/14
13		Develop delivery timeline	2 days	Thu 25/9/14	Sun 28/9/14
14		Obtain approvals to	1 wk	Mon 29/9/14	Fri 3/10/14
15		Secure required resources	1 day	Mon 6/10/14	Mon 6/10/14
16		Analysis complete	0 days	Mon 6/10/14	Mon 6/10/14
17		**Design**	**36 days**	**Tue 7/10/14**	**Tue 25/11/14**
18		Review preliminary	2 days	Tue 7/10/14	Wed 8/10/14
19		Develop functional	20 days	Thu 9/10/14	Wed 5/11/14
20		Develop prototype based	16 days	Mon 20/10/14	Mon 10/11/14
21		Review functional	5 days	Tue 11/11/14	Mon 17/11/14
22		Incorporate feedback into	5 days	Tue 18/11/14	Mon 24/11/14
23		Obtain approval to proceed	4 hrs	Tue 25/11/14	Tue 25/11/14
24		Design complete	0 days	Tue 25/11/14	Tue 25/11/14
25		**Development**	**56 days**	**Fri 26/12/14**	**Fri 13/3/15**
26		Review functional	1 day	Fri 26/12/14	Fri 26/12/14
27		Identify modular/tiered	2 days	Mon 29/12/14	Tue 30/12/14
28		Develop code	45 days	Mon 5/1/15	Fri 6/3/15
29		Unit testing (debugging)	31 days	Fri 30/1/15	Fri 13/3/15
30		Development complete	0 days	Fri 13/3/15	Fri 13/3/15
31		**Testing**	**4 days**	**Fri 13/3/15**	**Wed 18/3/15**
32		Develop integration test	1 day	Fri 13/3/15	Fri 13/3/15
33		**Integration Testing**	**4 days**	**Fri 13/3/15**	**Wed 18/3/15**
38		**Documentation**	**45 days**	**Mon 2/2/15**	**Fri 3/4/15**
46		**Deployment**	**5 days**	**Mon 30/3/15**	**Fri 3/4/15**
47		Determine final	1 day	Mon 30/3/15	Mon 30/3/15
48		Develop deployment	1 day	Tue 31/3/15	Tue 31/3/15
49		Secure deployment	1 day	Wed 1/4/15	Wed 1/4/15
50		Deploy software	2 days	Thu 2/4/15	Fri 3/4/15
51		Deployment complete	0 days	Fri 3/4/15	Fri 3/4/15
52		**Final Report**	**2 days**	**Mon 6/4/15**	**Tue 7/4/15**



Project: Software Development
Date: Sat 27/9/14

Task	Project Summary	Manual Task	Start-only	Deadline
Split	Inactive Task	Duration-only	Finish-only	Progress
Milestone	Inactive Milestone	Manual Summary Rollup	External Tasks	Manual Progress
Summary	Inactive Summary	Manual Summary	External Milestone	

5. Mid-Term Progress

After considerable effort spent on reviewing and refining the scope of the project, the focus was on tackling the biggest risks identified – thoroughly understanding the source code of Pocket Code, and then being able to connect a Pocket Code app to the TI SensorTag to read and write data to and from the BLE device.

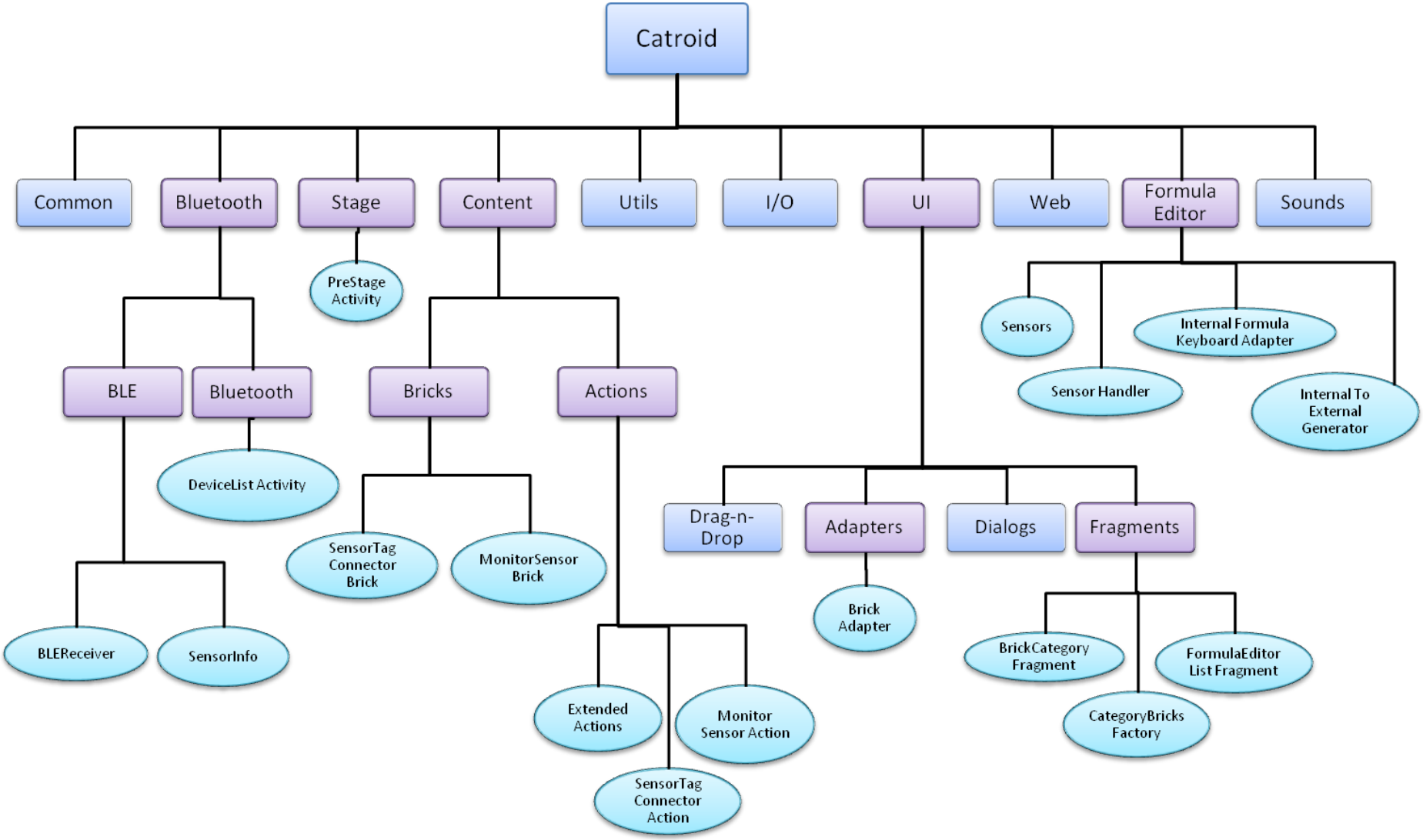
5.1 Analysis and Research

We first spent time devising a plan to conduct needs analysis about how we should make the interface and the Pocket Code "Bricks" as user-friendly as possible. We wanted to make the usage of all BLE technologies in Pocket Code appear at a high level of abstraction to the viewers. This was to assist younger programmers understand and utilize these features quickly.

5.1.1 Catroid Source Code

As part of the Risk-Driven Development strategy, we focused on tackling and solving the biggest risks of the project in the early stages as a reliable resiliency measure. We planned to create an initial vertical UI prototype to gain some feedback on our approach. Creating this prototype required us to research the Java source code of Catroid to see how the entire system was architected. Several modules were scanned to understand the UI of Pocket Code and make some additions to it.

The architecture of the entire Catroid application is shown in the diagram below. All packages are squares and the classes are circles. The packages that have been modified are highlighted in purple, while the classes that have been modified are highlighted in light blue.


Catroid
Common
Bluetooth
Stage
Content
Utils
I/O
UI
Web
Formula Editor
Sounds
PreStage Activity
BLE
Bluetooth
Bricks
Actions
DeviceList Activity
BLEReceiver
SensorInfo
SensorTag Connector Brick
MonitorSensor Brick
Extended Actions
Monitor Sensor Action
SensorTag Connector Action
Sensors
Sensor Handler
Internal Formula Keyboard Adapter
Internal To External Generator
Drag-n-Drop
Adapters
Dialogs
Fragments
Brick Adapter
BrickCategory Fragment
FormulaEditor List Fragment
CategoryBricks Factory

5.1.2 Android APIs for Bluetooth 4.0

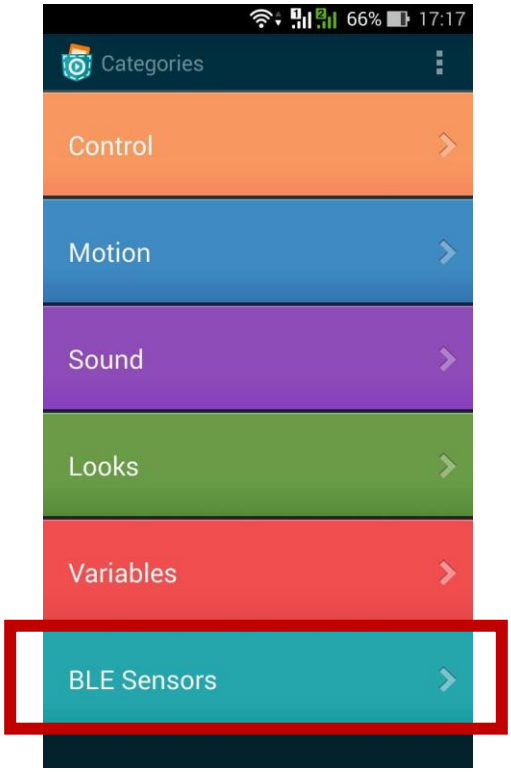
Also, for the purpose of this prototype, research was done on the Android APIs for Bluetooth 4.0 framework so as to understand how the Android OS works with Bluetooth 4.0 technology. This was done using informative YouTube channels and official Android documentation. According to the Android Developers website, “Android 4.3 (API Level 18) introduces built-in platform support for Bluetooth Low Energy in the *central role* and provides APIs that apps can use to discover devices, query for services, and read/write characteristics. In contrast to Classic Bluetooth, Bluetooth Low Energy (BLE) is designed to provide significantly lower power consumption. This allows Android apps to communicate with BLE devices that have low power requirements, such as proximity sensors, heart rate monitors, fitness devices, and so on.”

Key features and concepts involved in modifying Pocket Code are:

- Generic Attribute Profile (GATT) — The GATT profile is a general specification for sending and receiving short pieces of data known as "attributes" over a BLE link. All current Low Energy application profiles are based on GATT.
- Characteristic — A characteristic contains a single value and 0-n descriptors that describe the characteristic's value. A characteristic can be thought of as a type, analogous to a class.
- Descriptor — Descriptors are defined attributes that describe a characteristic value. For example, a descriptor might specify a human-readable description, an acceptable range for a characteristic's value, or a unit of measure that is specific to a characteristic's value.

5.2 Initial Prototype

The UI prototype that we created consisted of a new Brick Category called “BLE Sensors” (as shown in Figure 6). This category was added onto the existing categories of “Control”, “Motion”, “Looks”, “Sounds” and “Variables”. In the category of BLE Sensors, a new Brick was added, which allowed users to choose a sensor from the SensorTag via a drop-down list. The functionality of this Brick was hard-coded to just turn on Bluetooth, as part of the UI prototype. We gathered feedback on this prototype through a meeting with our supervisor, where he guided us as to how we could make the data from a SensorTag available to the user in a representation with a higher level of abstraction (decimal numbers instead of hexadecimal numbers) so that young users do not have to concern themselves with the math involved in handling raw data and understanding 3-dimensional data. After gaining some positive feedback on this approach, we proceeded onto developing some more functionality by adding features to new Bricks.



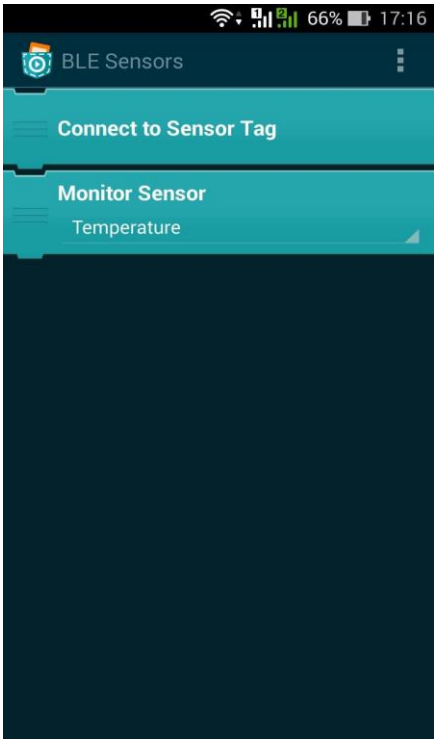
5.3 Modified Application

After the prototype was verified by our supervisor, we made several modifications to the existing Pocket Code application to reach our current stage. The modifications can be easily classified into two parts – front-end and back-end development.

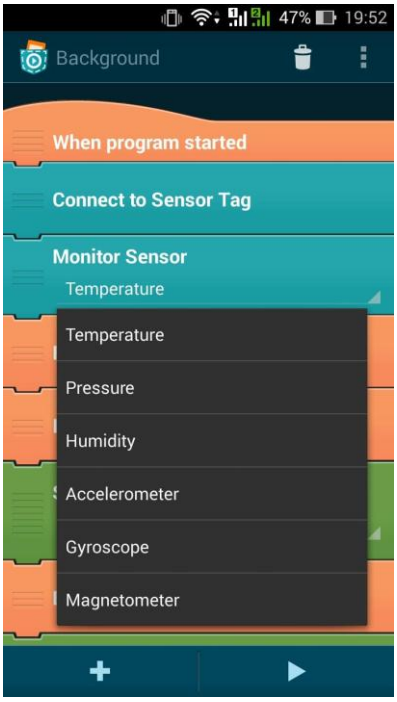
5.3.1 Front-end Development

Front-end development mostly involved understanding the UI of Catroid and adding features to it like new Bricks, new items in ListViews, creating drop-down lists to allow users to choose values and adding items to the Formula Editor UI. The Formula Editor is the part of Pocket Code that allows users to use numeric values as well as values from sensors, etc. to perform several tasks like create Boolean conditions, set value to variables, etc. The front-end development majorly includes:

- New Brick added to turn on Bluetooth, and then connect to SensorTag

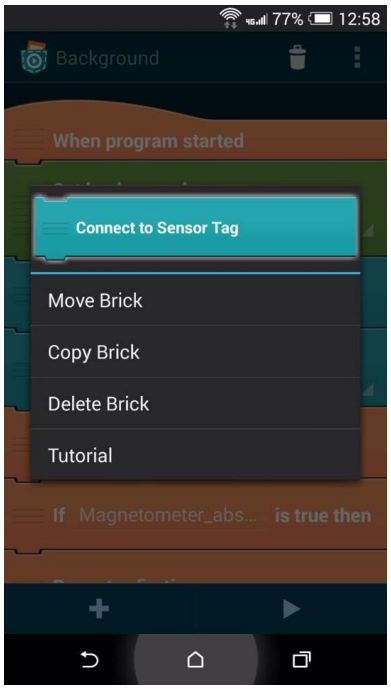


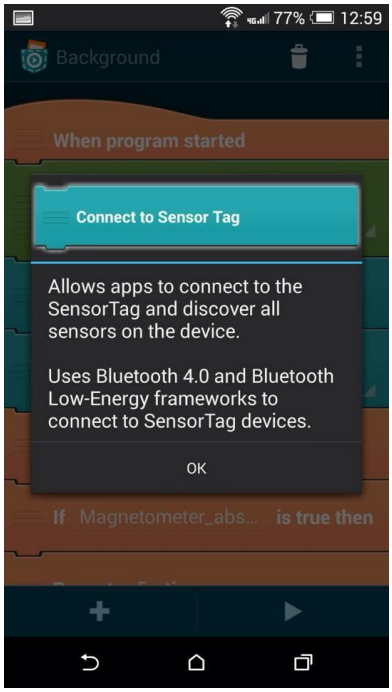
- New Brick to choose which sensor is to be accessed in the current Pocket Code application

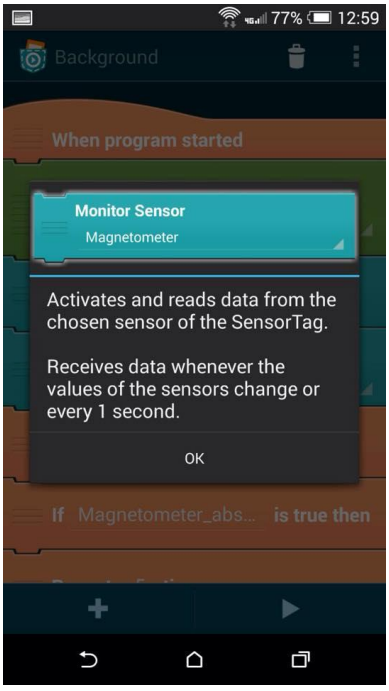


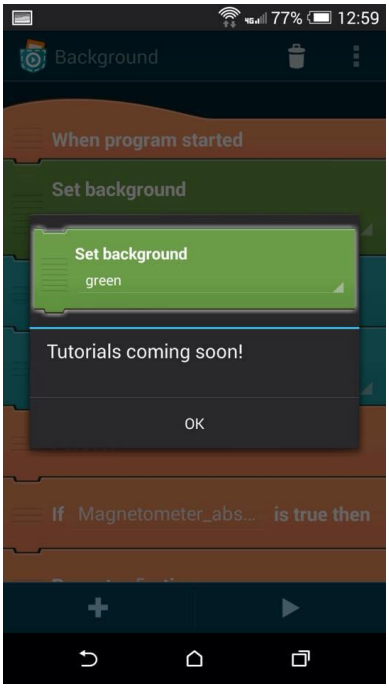
- Sensor values accessible by the Pocket Code app through 'Formula Editor', parsed to a certain level of abstraction for easy use by younger users

- Vertical prototype for short tutorial explaining the functioning of a brick that the user may be initially unfamiliar with.









Since we assume that the user has no prior knowledge of anything required to create apps on Pocket Code, the front-end needs to be as clean and user-friendly as possible. The heavy computations with values from TI SensorTag and other external operations like connecting to Bluetooth devices, registering callbacks, etc. are done in the back-end.

5.3.2 Back-end Development

The back-end of Pocket Code is highly complicated as Catroid actually runs user-made apps as a sequence of instructions using a customized interpreter. In this project, we added some functionality to integrate the front-end development with the core logic of the modifications we have made on Catroid.

- To connect to SensorTag, the app:-
    - Asks user permission to turn on Bluetooth
    - Scans for Bluetooth devices, and user chooses a SensorTag device from the list shown
    - Connects to the SensorTag using 'GATT profile' (a.k.a. GAP)
- To access a particular sensor, the app:-
    - Enables the selected sensor on SensorTag by writing a 'Characteristic' to it
    - Reads data from the selected sensor
    - Switches on 'live updates' to ensure that SensorTag will 'push' live data to the phone upon a change in the value of the sensor, or every second, by writing a 'Descriptor' to it
- Sensor values are accessed by the Pocket Code application using the procedure as follows:-
    - Formula Editor displays the various available sensor values, which the user can select

- The app parses boolean conditions involving the selected sensor values.
- The app parses raw hexadecimal data from the SensorTag into easily understood decimal values. The mathematical computations involved were achieved using codes already provided in Texas Instruments applications.
- The parsed values have a high level of abstraction – for example, data from x, y and z axes and absolute scalar values

- For users that are unfamiliar with the details of a particular brick, the user can press a brick to access a quick in-app help/tutorial explaining the functionality of the brick. This appears in a pop-up dialog box.
    - This has been implemented in such a way that if a developer ever attempts to introduce a new brick, they will be forced to override the text field containing a tutorial for that brick.

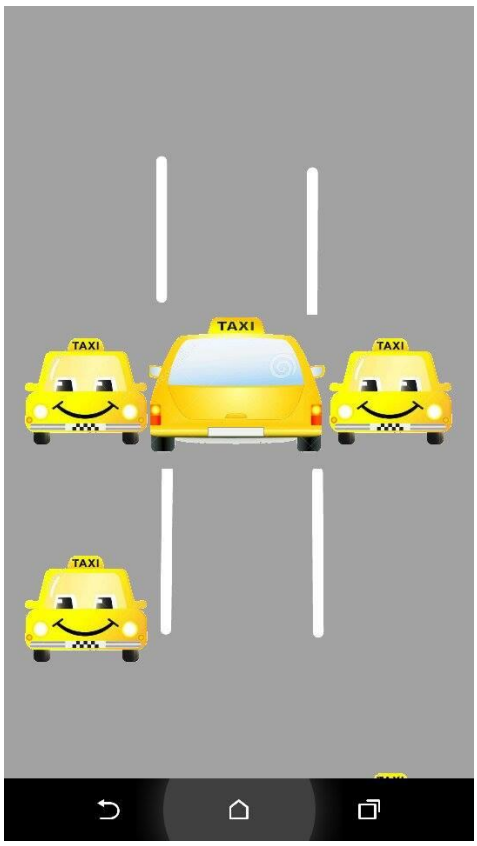
5.3.3 Sample/Example Apps

To demonstrate the functioning of the currently modified Pocket Code, we made three example applications on Pocket Code that utilize three different sensors in the SensorTag.

- Temperature Alarm: The Pocket Code app connects to the SensorTag and starts monitoring the temperature sensor continuously. If the temperature read by the sensor goes beyond a certain threshold (which can be determined by the user depending on the room temperature and the object whose temperature is to be measured) then the phone will output a beeping sound and the screen will change the color.
- Treasure Hunt: The Pocket Code app connects to the SensorTag and starts monitoring the magnetometer sensor continuously. If the magnetic field read by the magnetometer goes beyond a certain threshold (which can be determined by the user depending on the strength of the magnet involved) then the phone will output a beeping sound and the

screen will change the color. The thresholds, ranges, screen display upon entering a range as well as the sound output in a particular range can all be customized by the user while developing the application.

- Crazy Taxi: This is a short game that utilizes the gyroscope sensor in the SensorTag for the user to control the movement of a Taxi on the screen, trying to avoid obstacles on the way. The frequency of generating the obstacles as well as the number of obstacles can be easily customized by the user. If the player tilts the SensorTag to the left, the taxi goes one lane to the left and the taxi moves one lane to the right if the SensorTag is tilted to the right. If the taxi crashes into any obstacle, the game is over and the user's score is recorded. This score can be shared with an online community on a "Leader board" in a further enhancement to this application.



5.4 Risks Mitigated Until Mid-Term

By following the Risk-Driven Development process, we have so far identified and effectively overcome/mitigated a few of the biggest risks involved in the project, which we had identified earlier.

- We have created and defined the front end as well as back end logic for additional bricks in Pocket Code to communicate with SensorTag.
- Successfully connecting the SensorTag with Pocket Code within an application.
- A user-made Pocket Code application can receive and parse sensor data with back end mathematical calculations already implemented, so specific sensor data (like the degree of rotation in z-axis, from the gyroscope) is easily available in the 'Formula Editor'.

- Back end logic for helps and tutorials has been implemented and is easily accessible for new users that are unfamiliar with the functioning of a particular brick. It has also been made very easy for a developer who plans to modify Catroid source code at a later stage to add a quick tutorial when implementing the core logic for any new Brick.

5.5 Second Phase Plans

Having completed the prototype, we have touched the half way point of our project. Following our risk driven development approach, we have covered some major risks that we identified in the project plan which have been detailed in previous sections. Here we elaborate about our plans for the next phase of our project and the vision we have for the final deliverable. This is made after feedback gained from first phase presentation and further discussions with our supervisor.

5.5.1 Road-Map to Completion

- As of now, we have implemented the feature of integrating a few sensors of SensorTag with a restricted number of bricks. These bricks are the bricks that allow the user to connect to a specific sensor. During the next phase we will extend the implementation to integrate ALL sensors and bricks in SensorTag and Pocket Code respectively.

- Just the way we will integrate sensors and bricks, we will be providing tutorials for all the bricks available in Pocket Code. As of now, the tutorials are available for some of the bricks like "Connect to SensorTag" and for the "Monitor Sensor" brick. The tutorials will be extended to the bricks originally provided by Pocket Code such as the conditions and loops as well as for features such as bricks related to sensor accesses and all other enhancements done by us.

- A major task ahead is to explore the possibility of multiple devices connectivity across SensorTag and smartphone and vice versa. This task extends to integrating many more

types of BLE devices into one application so that more elaborate applications can be made. This will also include changing the user interface of the existing application. As of now, various sensors of the SensorTag are listed as multiple different vectors. For example, the magnetometer gets listed as four different sensors – one for each axis of movement, and one absolute value calculated as the scalar magnitude. This is certainly an inconvenience to navigate as multiple sensors will include many more such sensors and this will result in a very cumbersome list that is difficult to navigate. We plan to clean the user interface from such a list and display the sensors for each device separately as another drop-down list in the application. Moreover, we also plan to implement the integration such that more than one application developed on Pocket Code could be connected with one or more BLE devices.

Dependency: Modification of the firmware of System-on-Chip of SensorTag and other BLE devices.

- We plan to use the RSSI data provided by the sensors such that the user can be notified if the mobile device is moving away from the BLE device after a certain threshold. We also plan to provide an option to the user to use this data along with the 6 sensors provided by SensorTag and others provided by other BLE devices.

- As our modifications to Pocket Code are unique to our development and are not recognized by the Catrobat community as we include BLE sensor in our Scripts, sharing the apps made by users using our version of Pocket Code is not available yet. However, it is highly likely that our users will like to share their creativity with their friends and the larger community. So, we are looking forward to provide this option to our users to store their application on the internet or share their apps through the Pocket Code community. After some brainstorming, we came up with the following possible options:

    - Solution 1 – Provide an option to users in Pocket Code which will allow them to upload their applications to a database on a server owned by us, which will connect and allow sharing of apps developed on the enhanced Pocket Code provided by us.

Dependency: Availability of an appropriate online database and server.

- Solution 2 – Provide an option to users to email their application to themselves or to their friends.

- Solution 3 – Provide an option to users to store their application on the SD card of their phone from where they can share it with their friends.

Dependency: Ability to extract the whole user-made application including all the objects in the application, as well as their scripts.

- As of now, Pocket Code (as well as our enhancement to it) has the same User Interface for both mobile phones and Tablets. However, the screen sizes of tablets are considerably larger than that of mobile phones. Having realized that a lot of space is being wasted on tablets, Professor Lau gave us an amazing idea of utilizing this space by dividing the screen into 2 parts. We are planning to implement a feature on tablets such that one part of the screen shows the Scripts and allows the user to add bricks, while the second part provides visual changes at real-time to the User Interface so that the user gets an idea of how his or her changes affect the application developed by him or her. For example, if the user adds a brick for move object, the visual area can show how the object will move in the application.

- We have developed a few fun games and utility applications using Pocket Code and SensorTag. In future, we are planning to develop and provide a few more of such applications to the user with Pocket Code, so that they act as sample programs and guidance to users on what they can do with Pocket Code as well as how can they affectively use the bricks.

5.5.2 Identified Drawbacks in Pocket Code

While doing our research on Pocket Code, to understand the usage of Pocket Code and to get ourselves familiarized with the application we explored the app by writing some scripts and

developing experimental/probing applications. During this time, we noted some basic features that we think are important when we develop any application and found them missing in Pocket Code. So, during our next phase, we plan to upgrade Pocket Code further with the following improvements:

- Provide an option to print out values of the variables on the screen. As of now, the user can only use images or sound for output. So, we can implement a feature that will output user-defined text.
- Provide an option to write the script in an event-driven format. As of now, the user can only repeatedly check for an event to occur. So we will be extending the Catrobat language to allow certain scripts to be executed immediately when an event occurs, such as change in the value of a variable.
- Provide an option to break out of loops inside the script on change of conditions, as a user might want to execute a loop only until a certain condition is met.

5.5.3 Risks to be Mitigated

Following the Risk-Driven Development approach four our project, as mentioned before in Section 5.4 we have mitigated the major risks associated with the project's initial phase. Similarly, before going ahead with the implementation of further features, we have identified the major risks that still need to be mitigated.

- We have tried to connect many SensorTag devices to one application. However, we have not achieved much success in this area yet. As of now, our Pocket Code enhancement is not scalable enough to recognize multiple SensorTags. During the development, we will have to hardcode the number of SensorTag devices and the associated sensors with them. So, we need to do more research on this area and our past experiences with the implementation of this feature has taught us that it is a major risk that we would face during our second phase of development process. Moreover, we also plan to provide a feature so that the users can

connect to more than one type of BLE device to the same application, such as connect the application to a SensorTag and a BLE iCard. Integration of both devices to work together has been identified as a major risk. The option to connect to multiple devices also requires us to modify the User Interface. This is a challenge too as it will require us to refactor the source code associated with the UI for BLE enhancement.

- After the enhancements made to Pocket Code by us, the users can no more upload and share their applications with the Pocket Code community. So, as said earlier, we plan to provide this feature by either providing a feature to upload their work on a database on a server hosted by us or allow them to share it through email or store it on the SD card. However, the implementation of this feature is a challenge as we still need to find an appropriate online database and a server service, and a way to extract the whole user-made application including all the objects in the application, as well as their scripts.

- The User Interface of Pocket Code after our first phase of development is same for mobile devices and tablets. However, after the second phase of development, we plan to provide additional features on tablets due to availability of a lot more space. The feature of “live editing”, which means providing a real-time visual representation of changes made by the user on the script, seems to be a challenging task at hand.

- Before releasing the application to the market we need to assure that it is robust and suitable for users. So, thorough integration testing will be required for effective quality assurance. We assume that this will provide us with some insights into the gaps that we might have left open during our development process. As we do not know the size of these gaps, we take the worst case scenario and accept this as a reasonable risk.

- Today the app stores are filled with millions of applications. It is surely a challenge to stand out in the market – we need to come up with an effective marketing plan. Moreover, we are focusing our market audience to be 12-15 years old kids who are interested in how mobile applications are developed. This is an age group which is fickle in their loyalty and

likings, and therefore, getting their attraction will be a major challenge that we will be required to overcome.

5.5.4 Top Priority Tasks

1. Complete and more robust implementation of all sensors in SensorTag. A buggy application is its own enemy. The more the number of bugs, the less the people will like to use the application, no matter how great the idea behind it is. So, our top priority right now will be to complete the implementation of integration of all sensors in SensorTag and to all sensor bricks in Pocket Code. After completing the implementation, we will be doing thorough testing of the application. As parts of the project have been developed in groups and integrated together, thorough integration testing will be required. Along with testing, as the target users will mostly be beginners to the programming world, we will be extending effective and interesting tutorials to all the available bricks of Pocket Code.

2. Complete the implementation of connecting multiple sensors and modify the User Interface of Pocket Code accordingly. Such implementation has been one of our initial deliverables and a branch of our ongoing research focuses on this. It is of utmost importance to our project as integration of multiple sensors can increase the possibilities of the applications developed on Pocket Code manifold. We hope to achieve this and create more applications with such a feature.

3. Provide the feature to share and upload applications developed on enhanced Pocket Code. This is crucial for collaborative development and resource sharing between various developers who will use our enhanced Pocket Code to build their own apps and bring their ideas to live. We have reduced it down to 3 possible solutions as of now and in the coming few weeks we would start implementing our final solution after completing our research on this field.

6. Final Stage Development

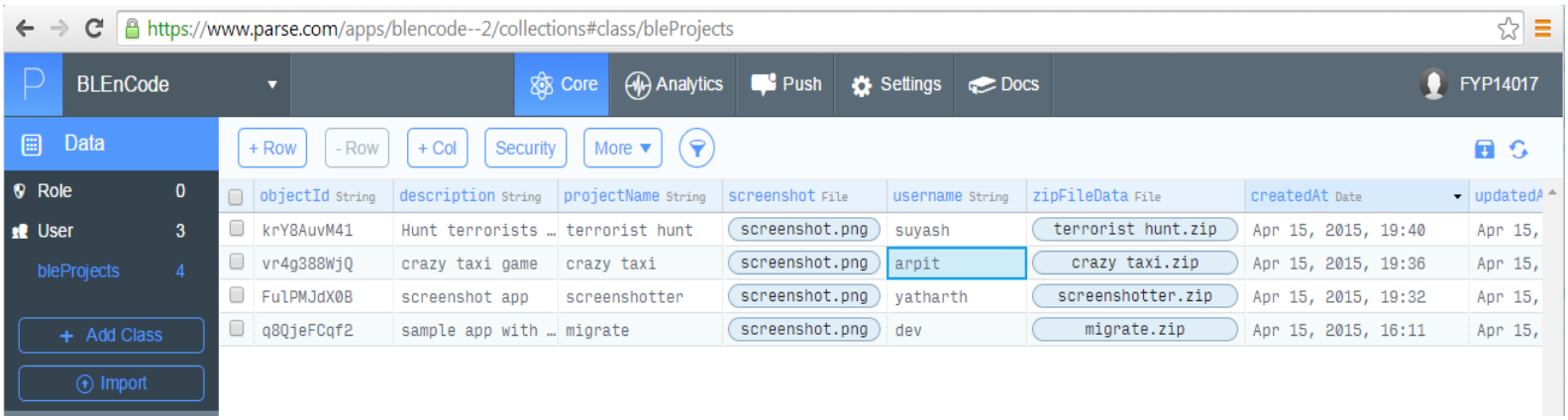
After the Mid-Term milestone of Elaboration phase, we presented our progress to Dr. Vincent Lau (Project Supervisor), Dr. T. W. Chim and Mr. George Mitcheson. After taking their feedback, we set out to develop on BLEnCode further with well-refined goals and plans. The following sub-sections describe our progress throughout the second semester – The Construction phase.

The Construction phase is where the majority of the system is built on the foundation that was laid during the Elaboration phase of the first semester. We implemented different features within different time-boxed iterations of roughly 10 days each. Detailed descriptions of the features of the final app are as follows:-

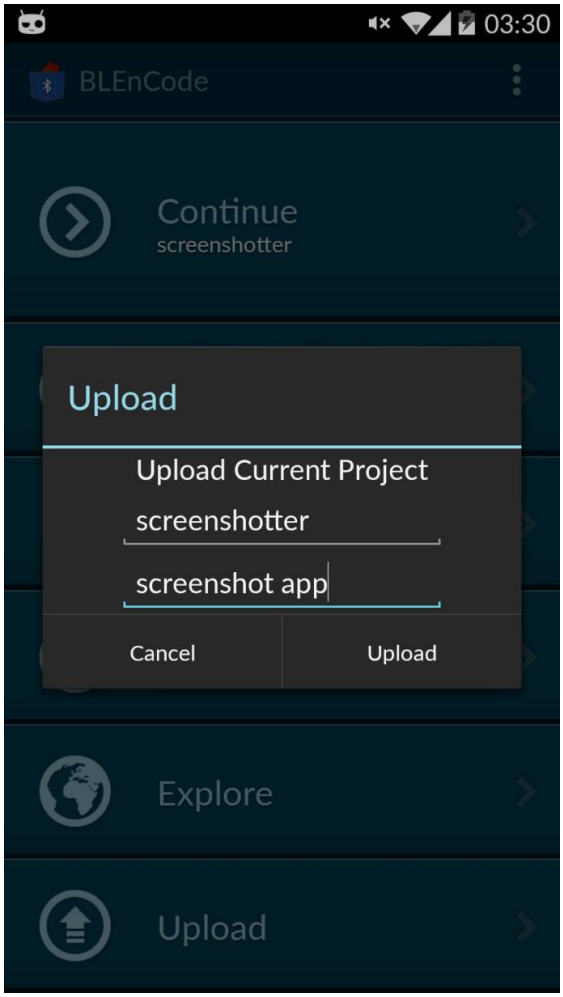
6.1 Sharing Projects Using Parse

Since BLEnCode projects/applications cannot be shared over Pocket Code servers, we needed a way to allow users to share their BLEnCode apps. As discussed before in Section 5.5.1, we had three possible solutions and the best one was having our own server on which users of the BLEnCode community could register, upload and download projects seamlessly without having to switch to emails or having to save on an external memory device. To achieve this we used the services of Parse cloud computing platform, which allows Android (among other Smartphone platforms) applications to interact with the cloud database for various CRUD (Create, Retrieve, Update, Delete) purposes.

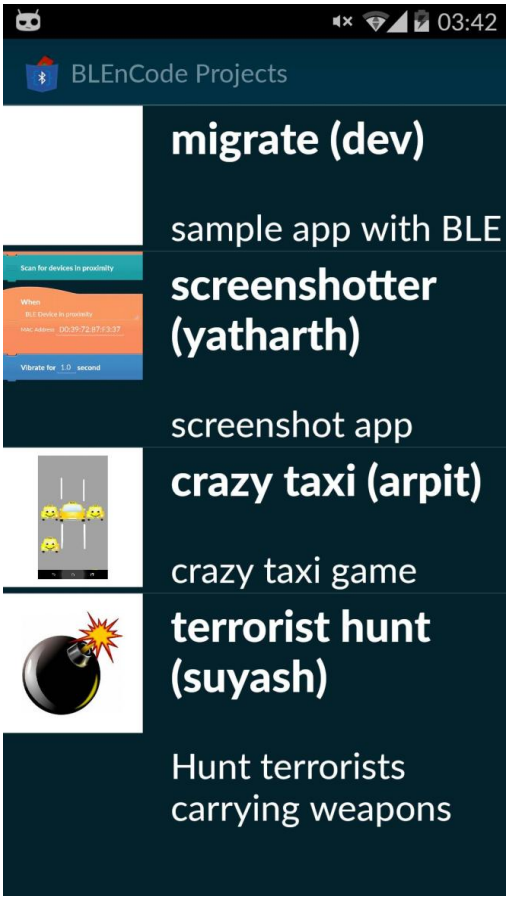
The 'Parse Core' back-end allows us to create an online database of BLEnCode projects and users' accounts. It contains information about the projects including the scripts within the project, a short description, the username of the uploader as well as a timestamp of the first and latest updates to the application. Following is a screenshot of the projects database table in the back-end:-



In the front end, there is an option for the user to share their project. The details of the application are consolidated into a .zip file and uploaded to the Parse server in the background if the user chooses to share the project.



To view other users' BLEnCode projects, the user can tap on "Explore" in the launch screen of the app. This opens a list view of all shared projects, with details such as project name as well as the username of the uploader.



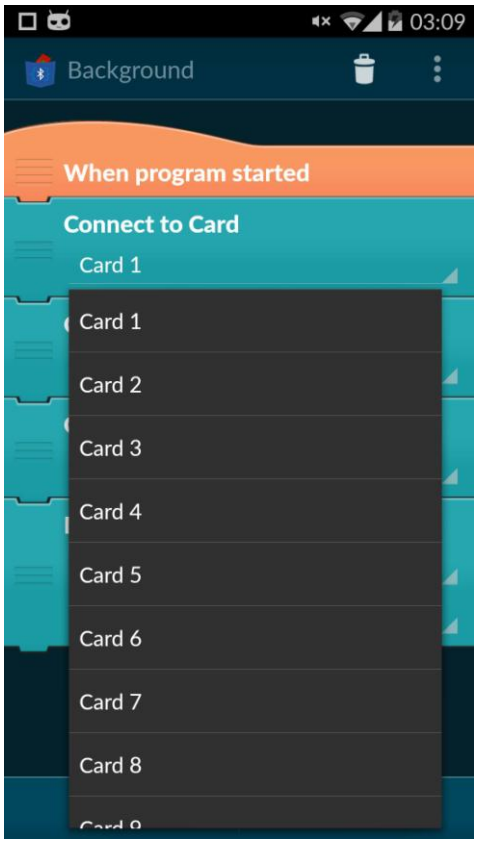
6.2 Multiple BLE Devices

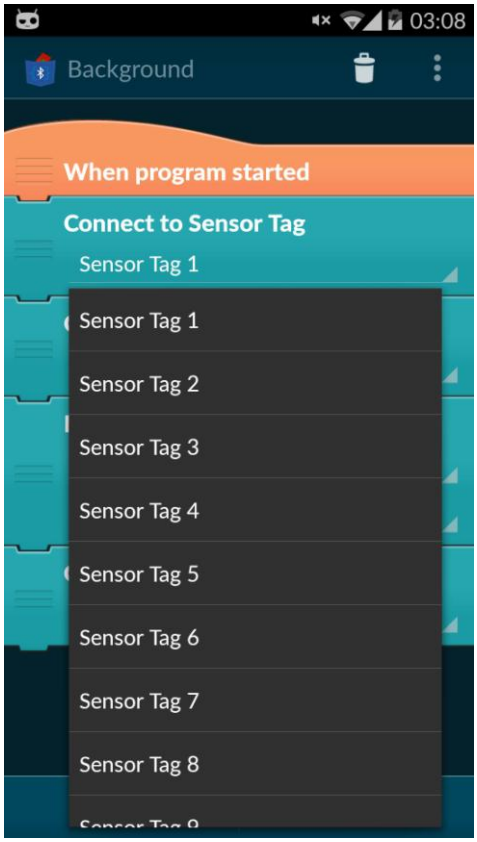
Up until the Mid-Term evaluation, the application that we had built was only functional with 1 SensorTag device. For the next phase, we set out to explore and push the boundaries of Bluetooth Low Energy by trying to simultaneously connect multiple SensorTag and MIDBot iCard devices to one BLEnCode application.

At first, we met with limited success when we tried to connect to multiple BLE devices all at once. We made the breakthrough when we had the user connect to the devices one-by-one. This allowed us to actively pursue the goal of allowing users to access different sensors of different SensorTag devices along with using different services of different iCards at the same time.

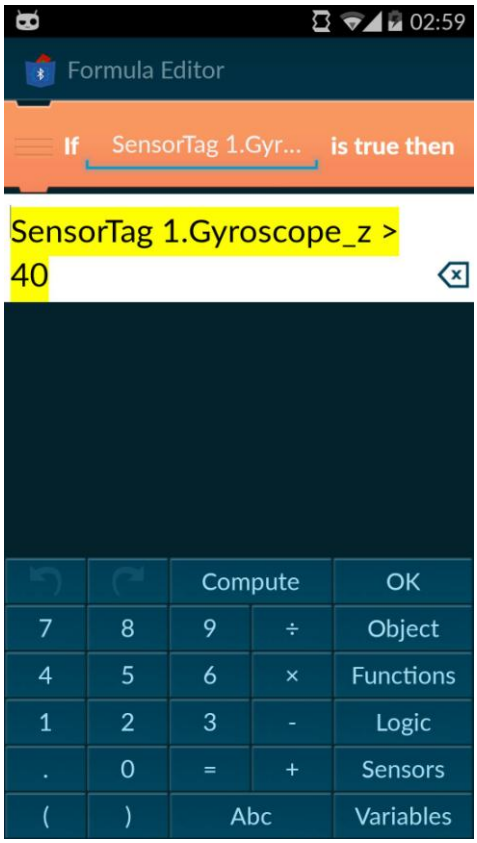
This was one of the most high-risk development tasks since most applications on Google Play currently just connect to at most 1 BLE device to act as a peripheral and even though it is theoretically possible to connect as many as 10 Bluetooth Low Energy devices at once, there are no commercially available apps which offer this functionality.

Now the user can choose to connect up to 10 SensorTag devices and 10 MIDBot iCard devices using a list view to choose which one to connect to. Also, the index/position of the device allows users to access data from and interact with the particular sensor of a particular device.


03:09
Background
When program started
Connect to Card
Card 1
Card 1
Card 2
Card 3
Card 4
Card 5
Card 6
Card 7
Card 8

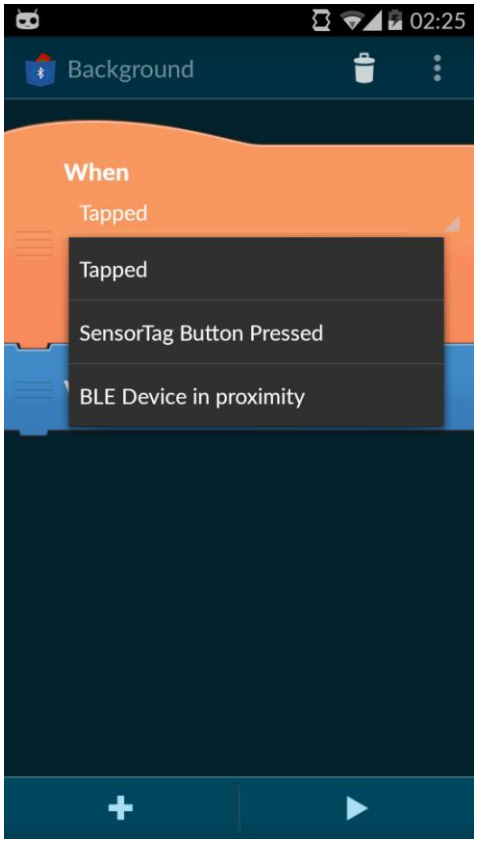

03:08
Background
When program started
Connect to Sensor Tag
Sensor Tag 1
Sensor Tag 1
Sensor Tag 2
Sensor Tag 3
Sensor Tag 4
Sensor Tag 5
Sensor Tag 6
Sensor Tag 7
Sensor Tag 8

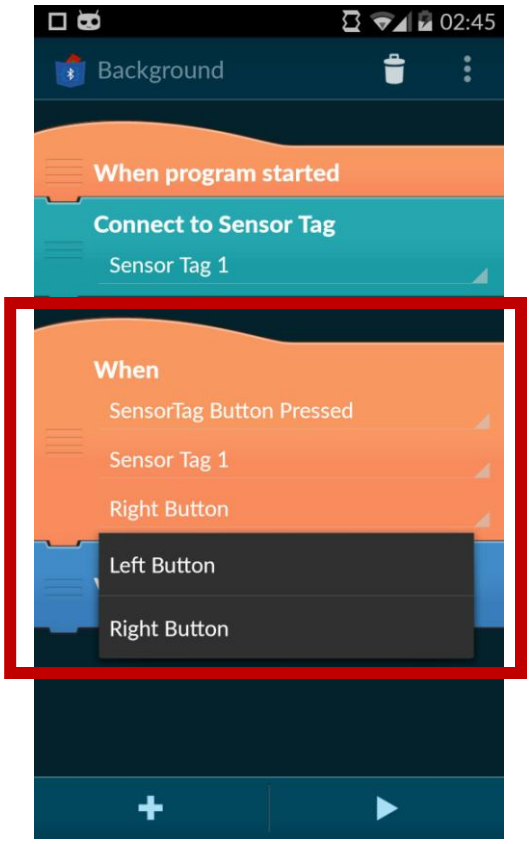

02:56
Sensors
face_y_position
Sensor Tag 1
Sensor Tag 2
Sensor Tag 3
Sensor Tag 4
Sensor Tag 5
Sensor Tag 6
Sensor Tag 7
Sensor Tag 8
Sensor Tag 9
Sensor Tag 10

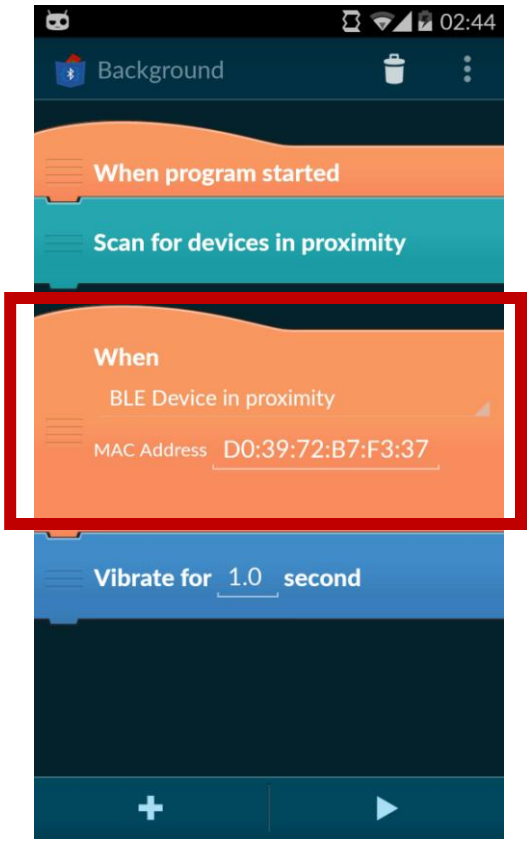

02:59
Formula Editor
If SensorTag 1.Gyr... is true then
SensorTag 1.Gyroscope_z > 40
Compute
OK
7
8
9
÷
Object
4
5
6
×
Functions
1
2
3
-
Logic
.
0
=
+
Sensors
(
)
Abc
Variables

6.3 Event-Driven Execution – the "*When*Brick"

To avoid always having to continuously scan-and-wait for certain events, we implemented our own callbacks to trigger when an event happens. The "*When*" brick was modified to achieve this. The user can decide from a list of possible events that can occur during the runtime of the application, and then customize the appropriate action to take place 'when' it happens.



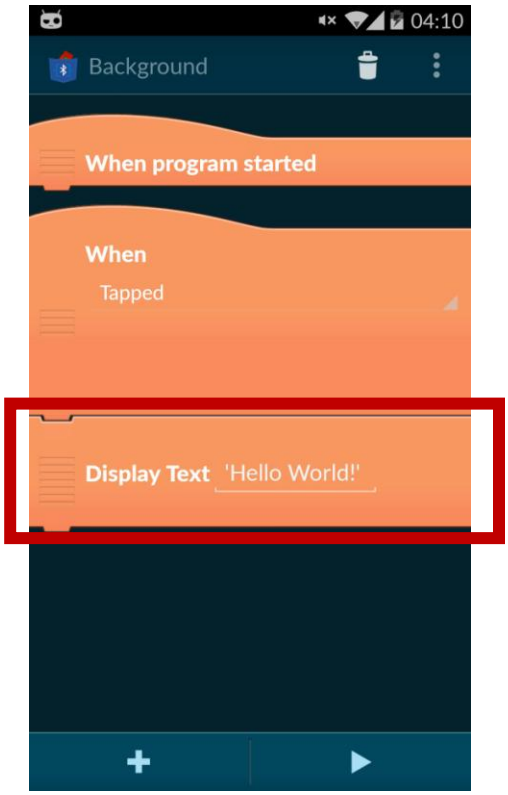

02:45
Background
When program started
Connect to Sensor Tag
Sensor Tag 1
When
SensorTag Button Pressed
Sensor Tag 1
Right Button
Left Button
Right Button


02:44
Background
When program started
Scan for devices in proximity
When
BLE Device in proximity
MAC Address D0:39:72:B7:F3:37
Vibrate for 1.0 second

6.4 Display Console Text

As programmers, we know that one of the most used function is to print text to the console to see output quickly. Pocket Code does not support such functionality, so we decided to add a "Brick" that can display text to the console during the execution of a BLEnCode app. This functionality would immensely help young developers who would like to see quick output on their screens to help them head in the right direction. These outputs would also enable easy debugging if needed so that users can follow the application logic.

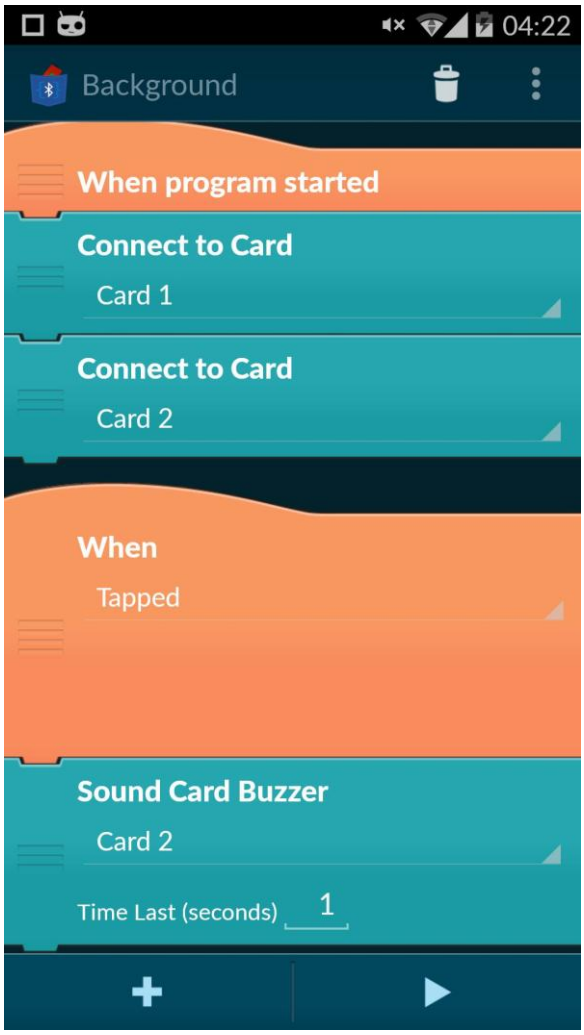
In the application script, the following screenshot shows how to add the console print:-

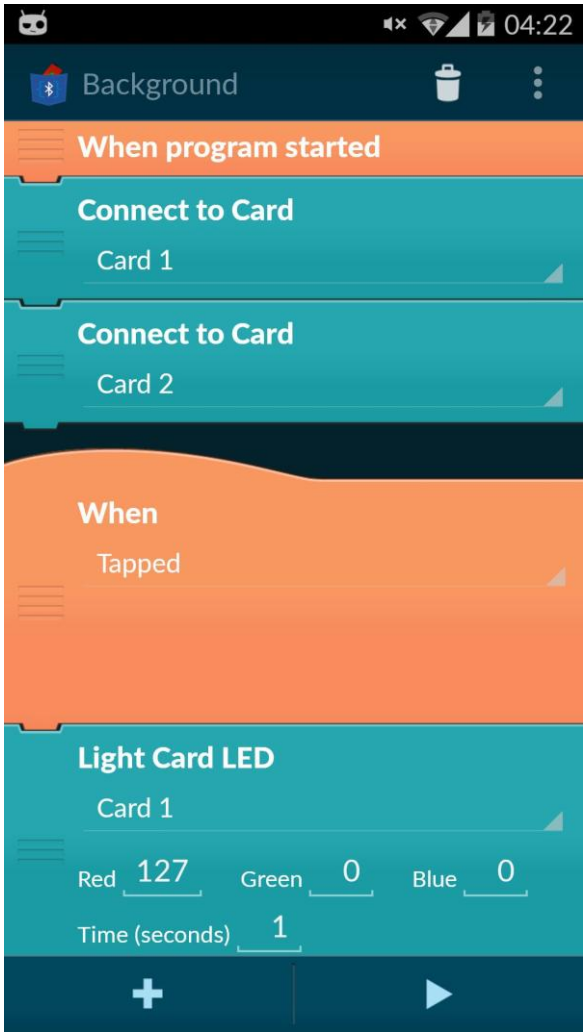


And this is the console display at runtime:-

6.5 Interactions with MIDBot iCard

The MIDBot iCard is a Bluetooth Smart device that interacts with BLEnCode, and the user can send instructions to multiple connected iCards, mainly to turn on the LED or sound the buzzer in the iCard. The user can easily customize the values for the color that the LED should display using specific Red/Green/Blue color values. The buzzer's sound output can also be manipulated to specify for how long the buzzer should sound.



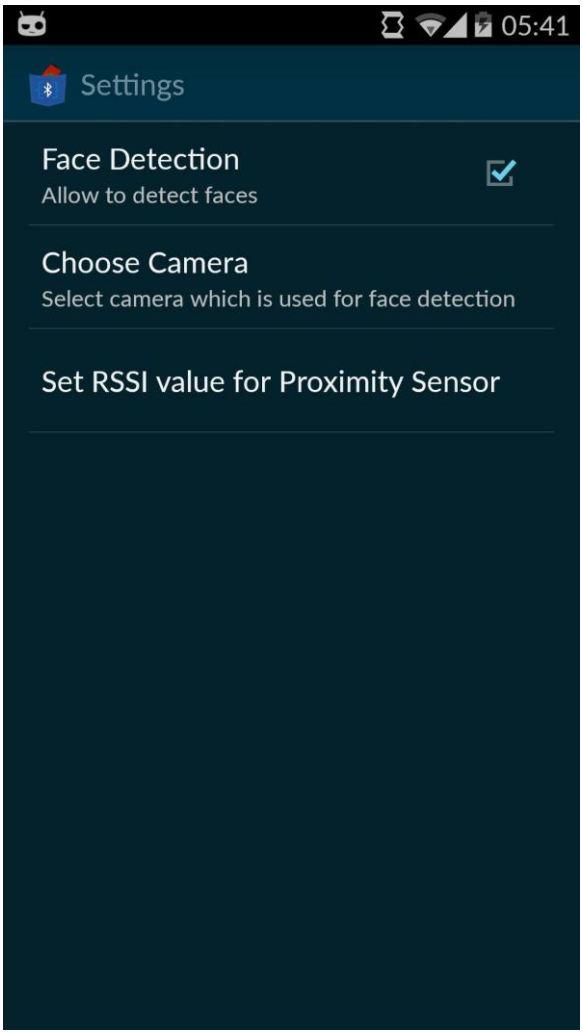


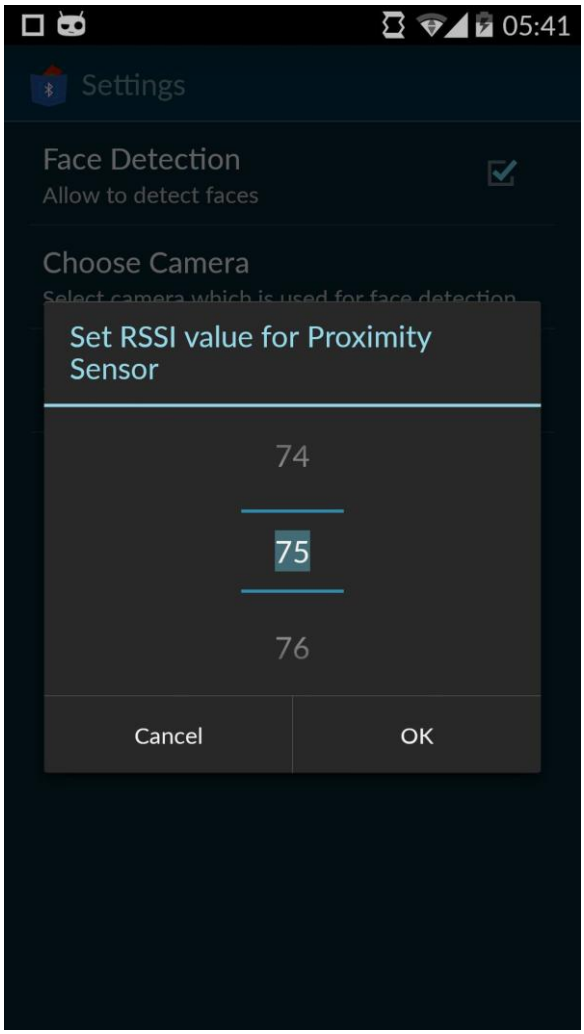

B0:39:72:B7:1D:77

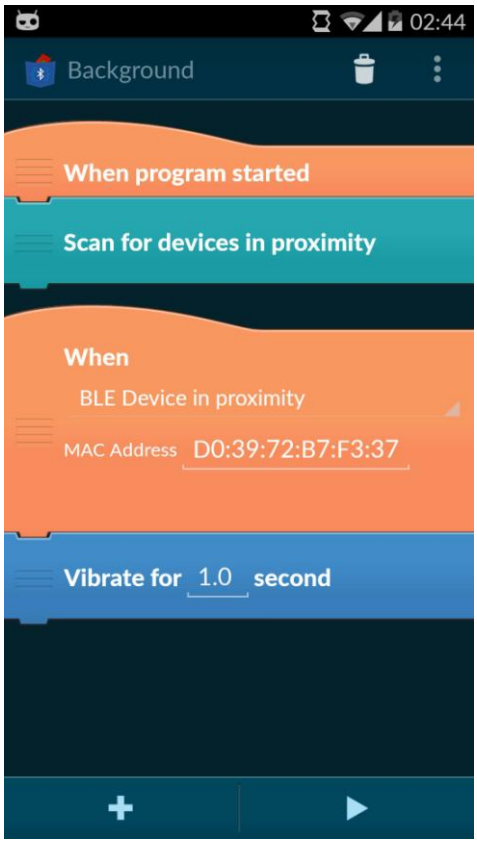
6.6 Finding BLE Devices in Proximity

After our Mid-Term presentation, there emerged a new idea to use the RSSI signal advertised by Bluetooth Low Energy devices to approximate their proximity and use these devices as Proximity Sensors. The initial approach was to try to determine distance of the BLE device using the RSSI value, but due to the unstable nature of RSSI value, this was deemed infeasible.

Hence, users were given the ability to set the RSSI value such that devices with signal strength stronger than the specified value would be considered “in proximity”. Also, the users would get to perform specified actions when a particular device (identified by its MAC Address) came into “proximity”.





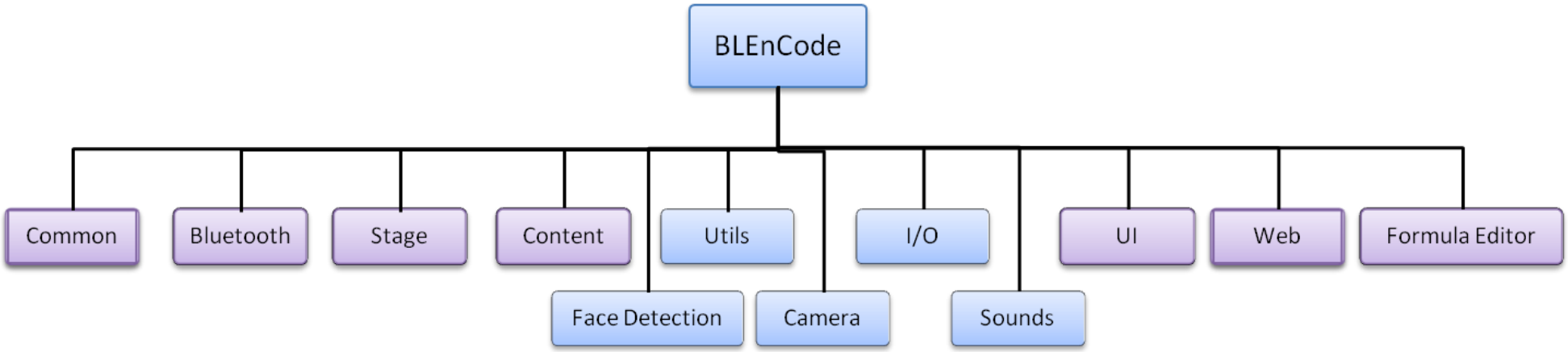

02:44
Background
When program started
Scan for devices in proximity
When
BLE Device in proximity
MAC Address D0:39:72:B7:F3:37
Vibrate for 1.0 second

6.7 Migration from Catroid to BLEnCode

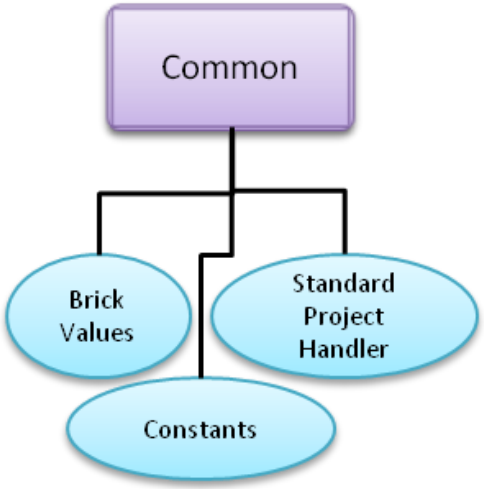
The original Pocket Code application's source code is called Catroid, and everything including the package names and even the finer details like strings are reflective of that. All our developments and enhancements were made on the Catroid source that we extracted from their public GitHub repository. Towards the completion of the project, we focused our efforts on making BLEnCode a totally different app on the Google Play Store, and this meant migrating all source code from the original "*org.catrobat.catroid*" to a different package name – we chose "*hku.fyp14017.blencode*". We also designed our own logo for the application, which is currently watermarked at the header in all pages of this report and serves as our default Android app icon.

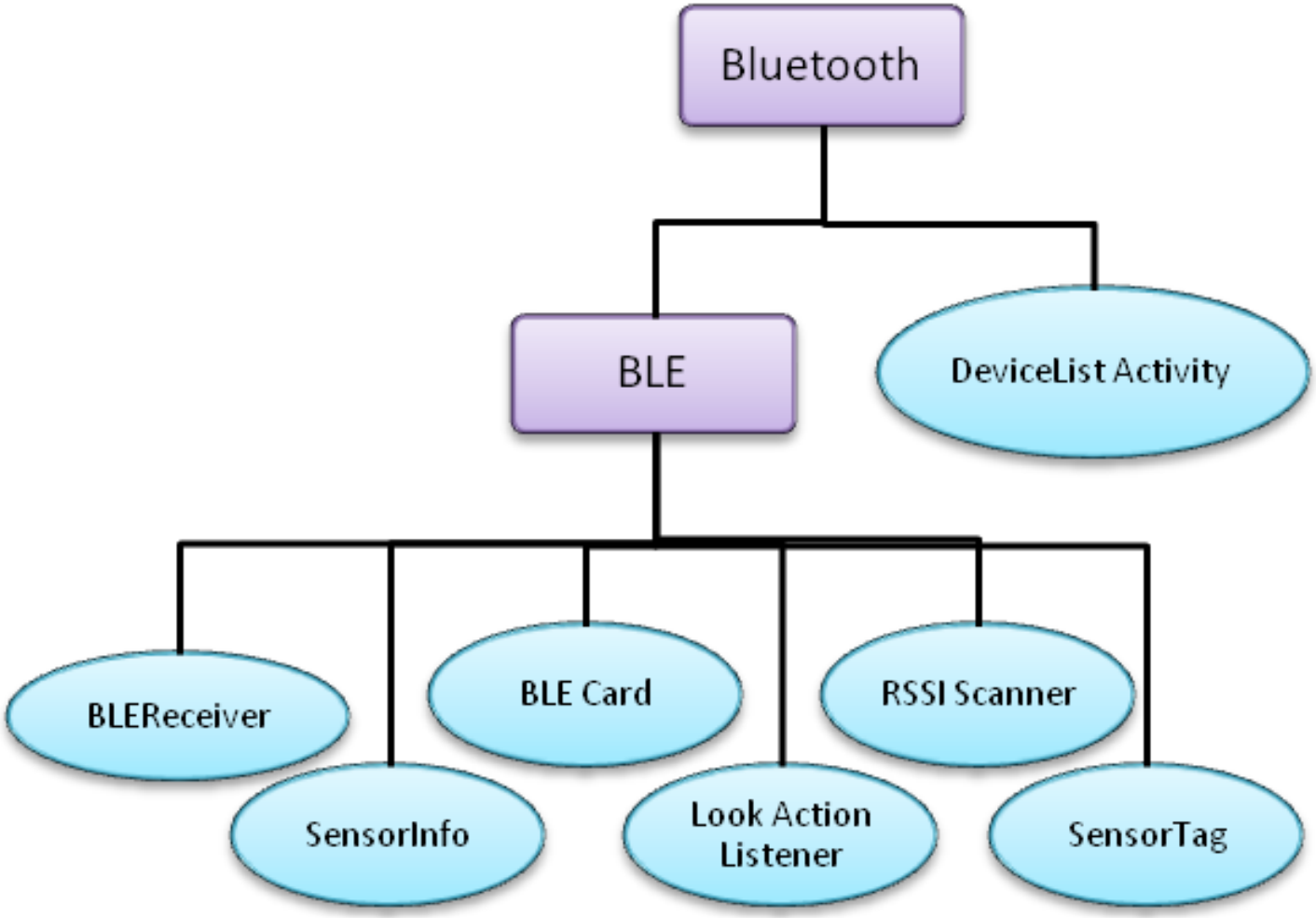
This involved thorough legal inspection of the licenses used by Catrobat for their development of Pocket Code, and we have generated our own license based on theirs, and ensured that all our developments and enhancements comply with the terms and conditions laid down by the aforementioned licenses. These legal aspects are further explained in section 11.

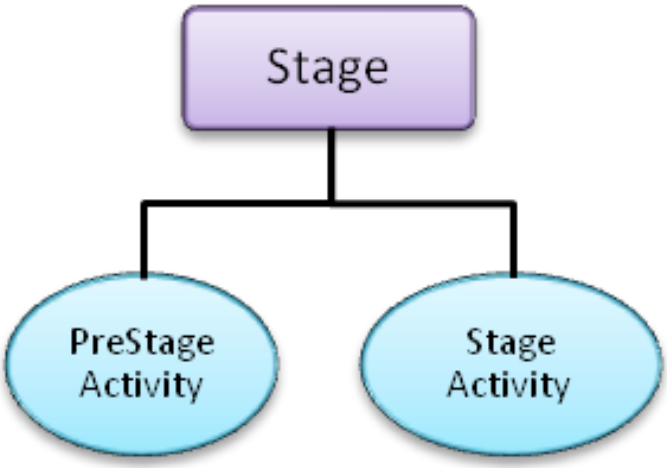
The development process spread around various modules present in the Catrobat system. Such modularity in the program structure led to very efficient development and helped in making an error-free application as various functionalities were restricted to their respective modules. Here, we elaborate on the said structure using a visual representation. The packages in purple are packages that have been modified or added, and the ones in blue are those left un-edited.

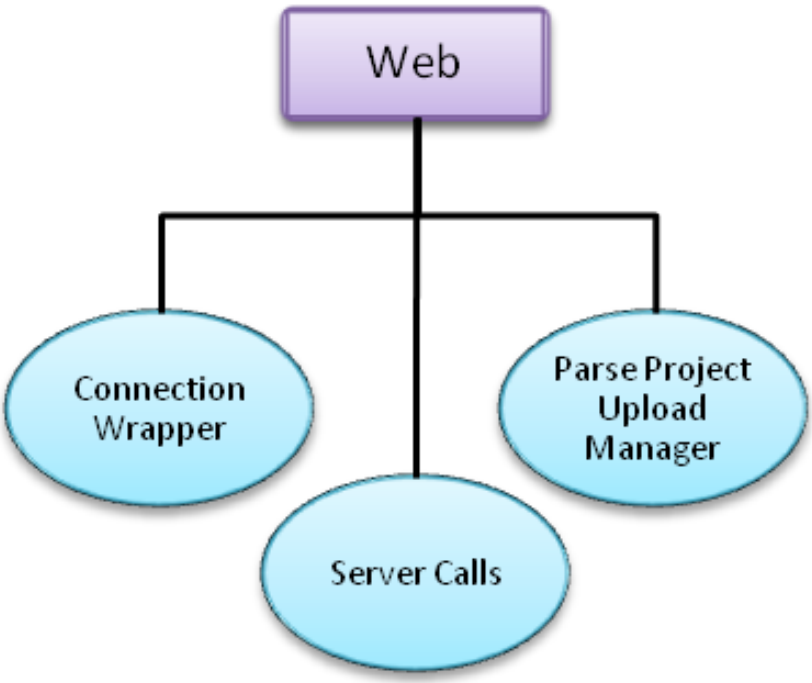


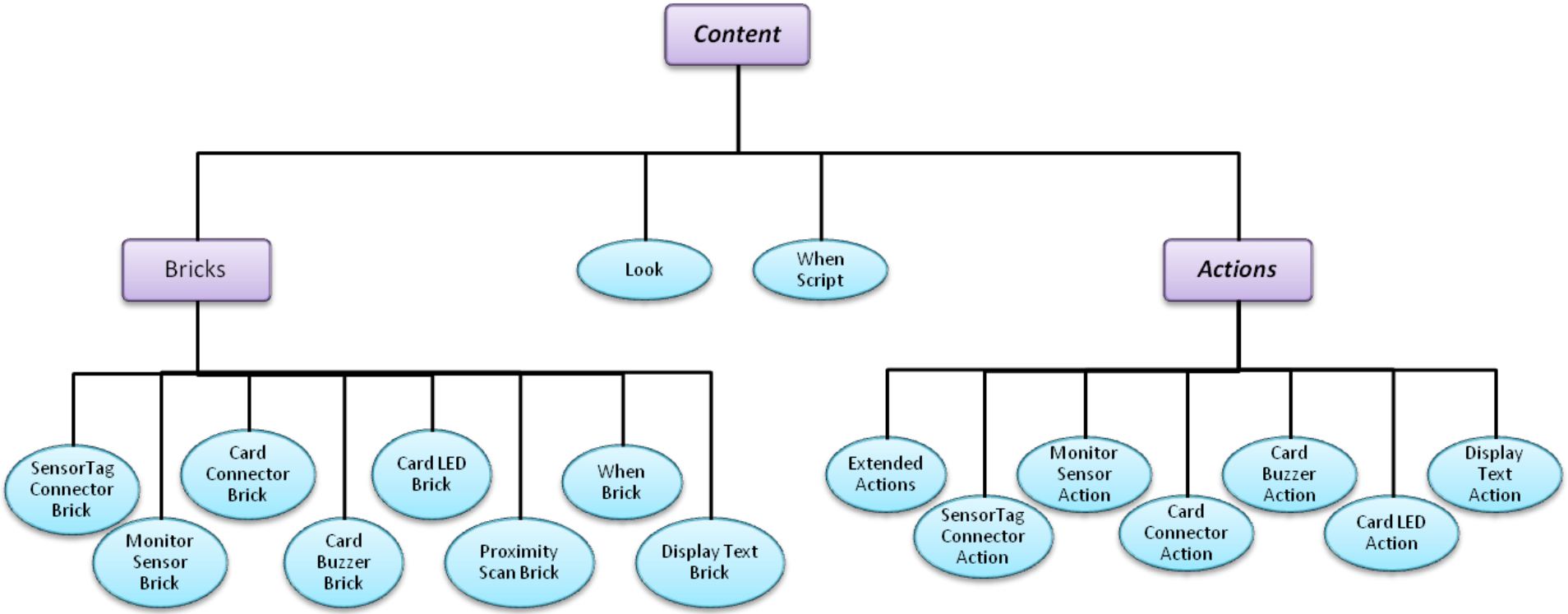
In further detail, the following are the package structures of the purple colored packages that we have modified/added:-

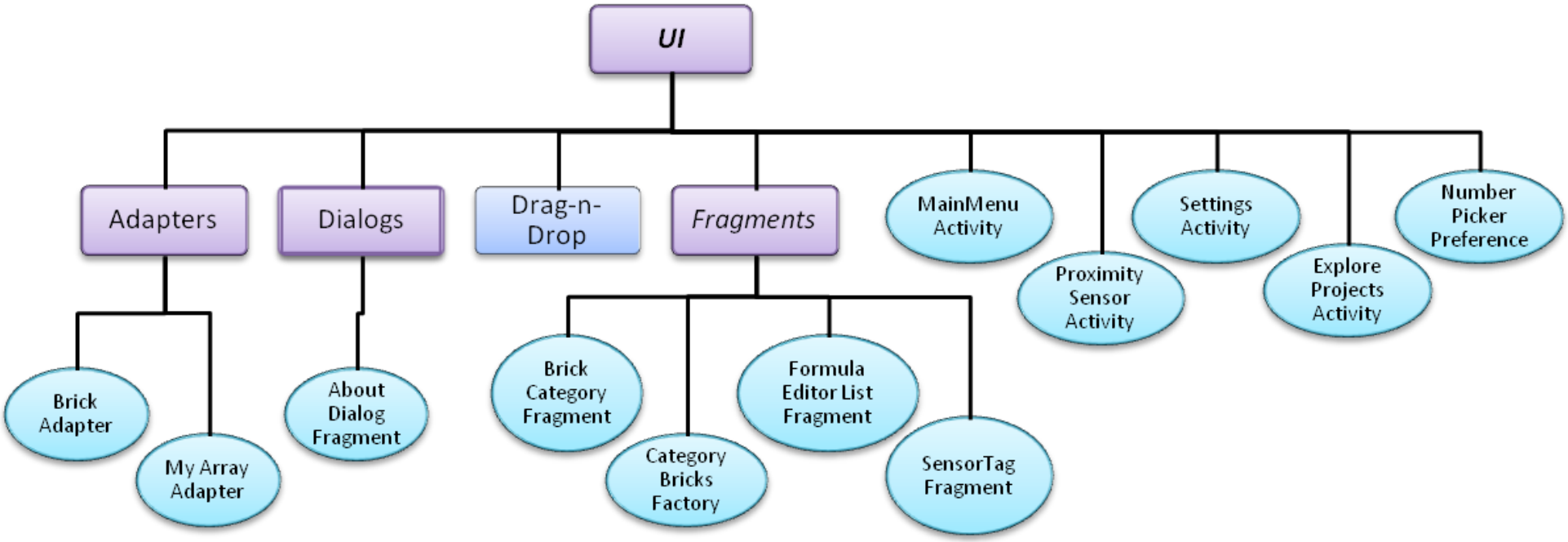


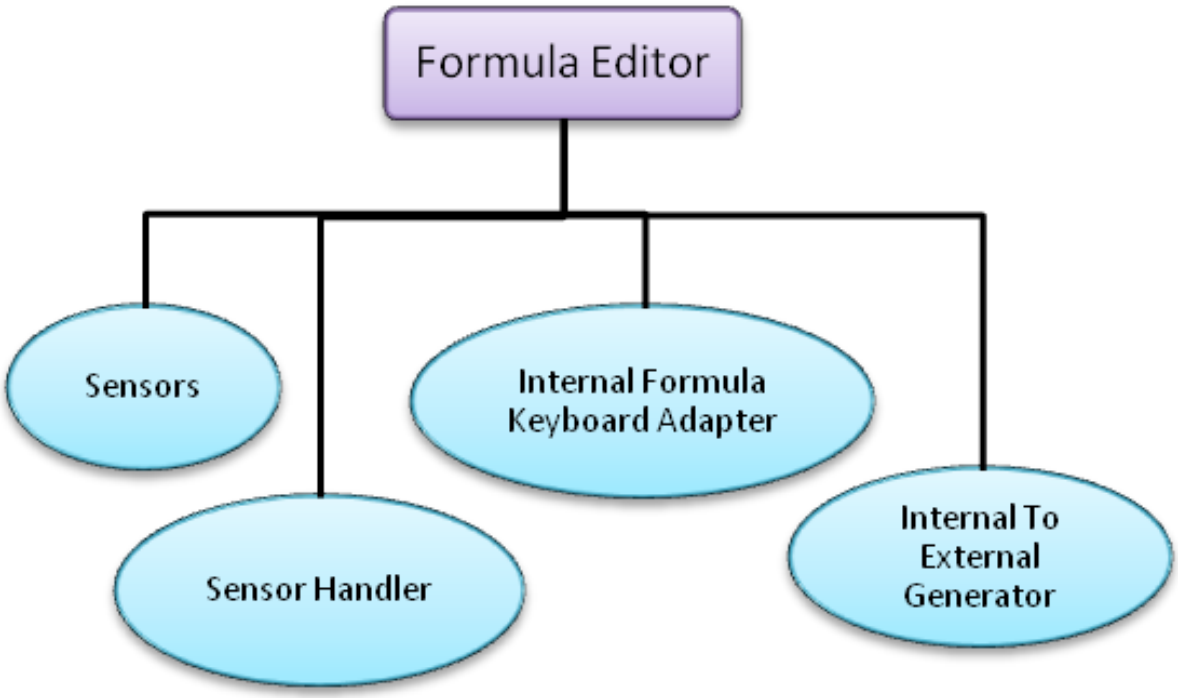

Bluetooth
BLE
DeviceList Activity
BLEReceiver
BLE Card
RSSI Scanner
SensorInfo
Look Action Listener
SensorTag


Stage
PreStage Activity
Stage Activity


Web
Connection Wrapper
Parse Project Upload Manager
Server Calls


Content
Bricks
Look
When Script
Actions
SensorTag Connector Brick
Monitor Sensor Brick
Card Connector Brick
Card Buzzer Brick
Card LED Brick
Proximity Scan Brick
When Brick
Display Text Brick
Extended Actions
SensorTag Connector Action
Monitor Sensor Action
Card Connector Action
Card Buzzer Action
Card LED Action
Display Text Action


UI
Adapters
Dialogs
Drag-n-Drop
Fragments
MainMenu Activity
Settings Activity
Number Picker Preference
Proximity Sensor Activity
Explore Projects Activity
Brick Adapter
My Array Adapter
About Dialog Fragment
Brick Category Fragment
Category Bricks Factory
Formula Editor List Fragment
SensorTag Fragment


Formula Editor
Sensors
Sensor Handler
Internal Formula Keyboard Adapter
Internal To External Generator

7. User Guide

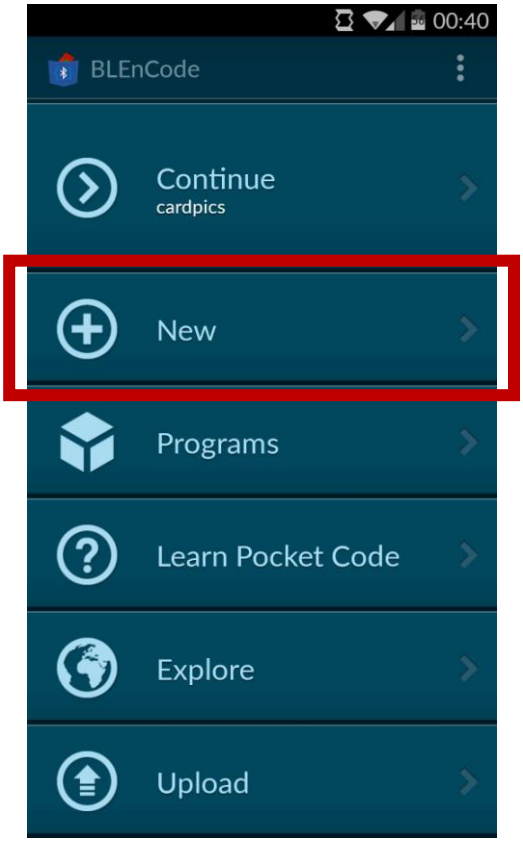
BLEnCode is a visual programming platform which heavily utilizes the Catrobat visual programming system (*Catrobat is developed by the Catrobat project http://developer.catrobat.org/*). You can now utilize BLE (Bluetooth Low Energy) sensors to create many more apps that utilize real-time data.

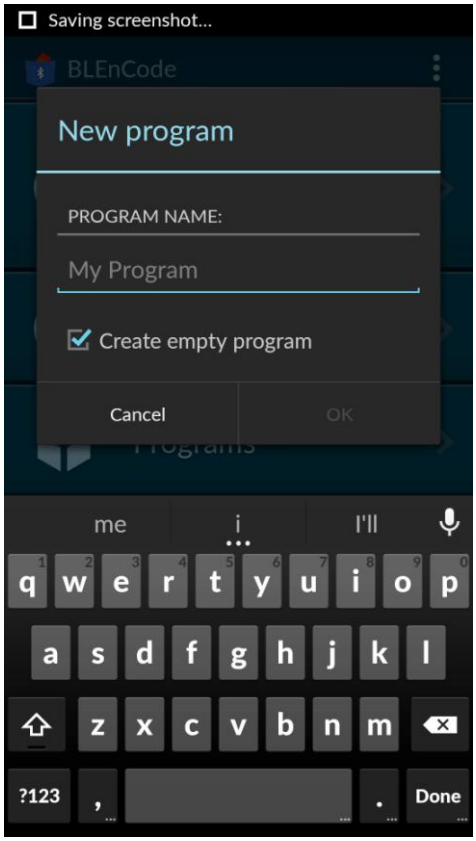
This short user guide will help you to get started and start making wonderful apps. This guide explains various new features of BLEnCode like using TI Sensor Tag, sharing your new apps in our community, utilizing tutorials to learn more about various functions supported by BLEnCode etc.

7.1 Creating Your First app!

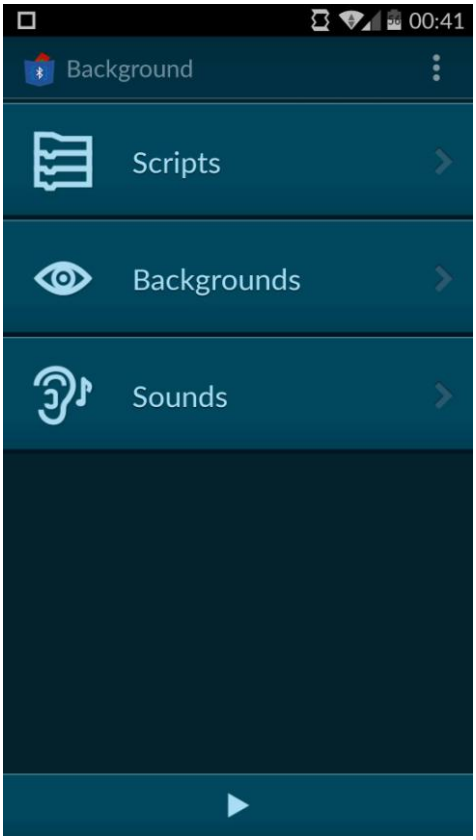
In this section we detail all the steps required to make your very first app on BLEnCode. This is a very simple app that changes the background of the screen when tapped on the screen.

1) Open BLEnCode. You will see various fields like “Continue”, “New”, etc. on the main menu screen. Click on “New” to make a new project. You will be asked to give your new program a name. Please input a name of your choice. PLEASE DO NOT FORGET to check the “Create empty program” option before clicking OK. Else, you will need to create another program.

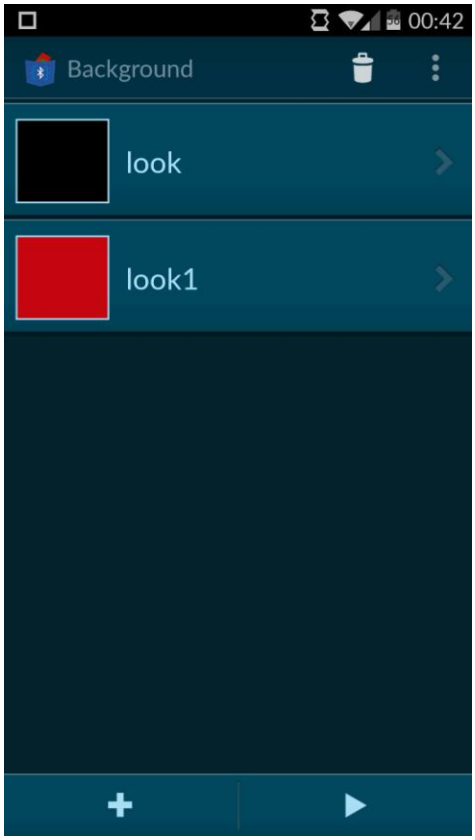




2) You have now entered your own new project! You can add backgrounds and objects to you application here. You can manipulate these later by writing scripts (explained later). For example, if your application needs a black background, please add one in Background.



3) Let's say for this application, you need a black background. Please click on Backgrounds -> Backgrounds -> "+" -> Draw image to choose your background color. Once you fill the background color, please return back to BLEnCode to resume the application. The black background will be named as "look" in the application screen.



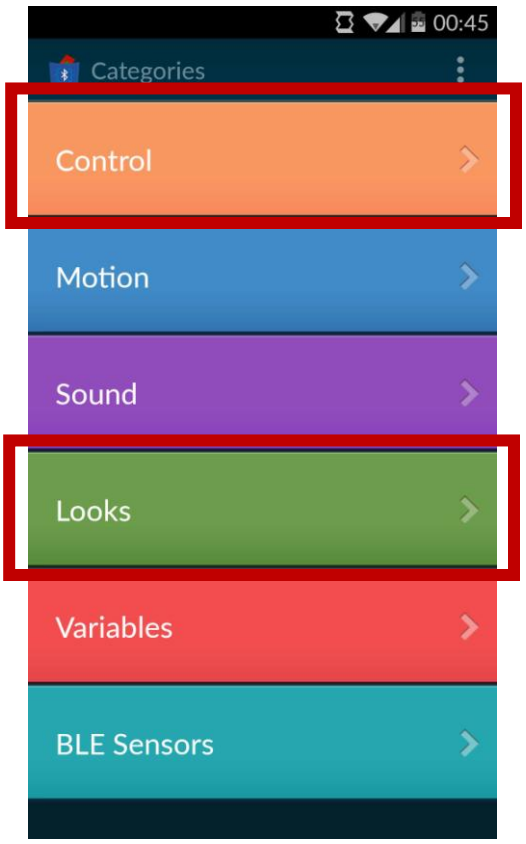
4) Return back to the project menu to add another Background as has been shown in the steps above. We will go ahead and add a simple script to the background which will make this background switch when the screen is tapped.

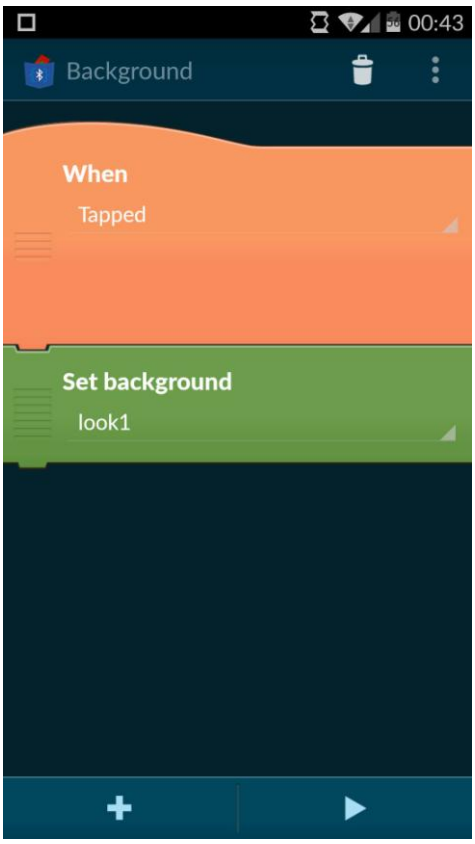
   a. Please navigate back to the project page and choose the background and choose "Scripts".

   b. Please press + to start adding a script. You need to add a "Control" brick. These bricks are how we will introduce various conditions into our programs. For example, Motion bricks add movement functionalities to our objects. Looks and Sound bricks help us manipulate visual and audio functions in our app. Here we need a condition that recognizes a screen tap. Hence, we go to Control section and choose the "When" brick. Whenever you are unsure about what a brick does, please click on the brick and choose the "Tutorial" option to learn more about it. As of now, we have a "When Tapped" brick (Tapped is the default option for this brick. We will learn more about other options in later examples).

   c. Now we want to add some action (being change in background) when the current black background is tapped. Hence, we add another brick in the same script to logically continue our implementation. Choose "Set background" in the "Looks" menu to change the background to another object. Hover this brick directly below the When brick to make logical sense. Choose the object from the dropdown in the "Set background" brick. Choose the background image which you want to switch to.

   d. And that's it! We have made our very first simple program. Click the play button to run the program, tap the screen and watch your background switch!


00:45
Categories
Control
Motion
Sound
Looks
Variables
BLE Sensors


00:43
Background
When
Tapped
Set background
look1

7.2 Creating an App with BLE Sensors

Now that we have seen how to make a small working application, let us venture into the realms of BLE Sensors and try to use some data from a TI SensorTag device in our application.

1) Go to the simple app created in the previous section. It can be accessed from the main menu screen through the "Continue" option to continue with the current application. Else, any project on the local storage can be accessed through the "Programs" option on the menu.

2) Go to the project menu to add an object. Click on + at the bottom of the screen to add an object. Let's say you choose an image from your own gallery here, please choose the appropriate option when prompted to go to your gallery to choose an image. Once you choose an image, you will be automatically taken to add a script to this object. Adding a script to any object/background helps you control its behavior. Script is where he programming g is involved and it is very easy to program in BLEnCode thanks to the Catrobat System. We will go ahead and add a simple script to the object.

    a. Please navigate back to the project page and choose the background and choose "Scripts".

    b. Please press + to start adding a script. You need to add a "Control" brick so we go to Control section and choose the "When Program Started" brick.

    c. Also, add a "Hide" brick from the Looks category under the "When Program Started" brick.

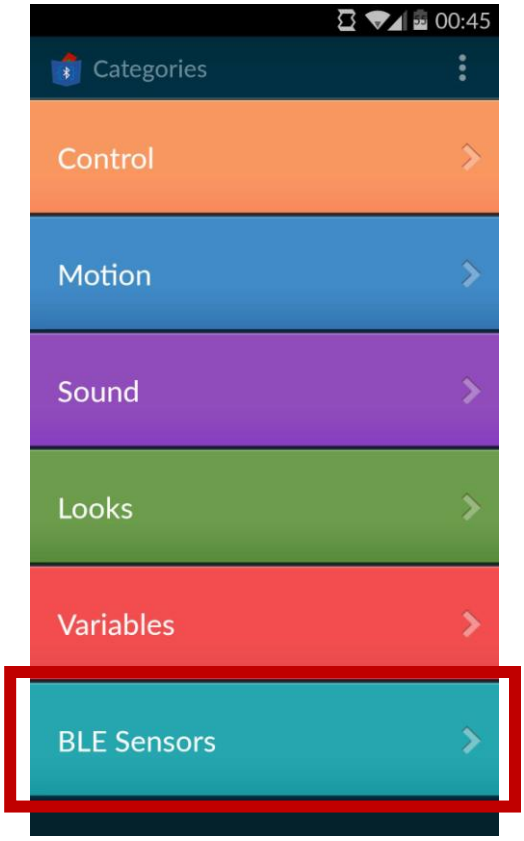
    d. Now, we can further add some actions to be performed when the program has started. Let us try to connect to a TI SensorTag. To do that, we will add a brick to the Script from the "BLE Sensors" category. Choose the "Connect to Sensor Tag" brick and add it to the script under the "When Program Started" brick. PLEASE ENSURE that whenever you connect BLE devices, please connect them in

numerical order. Choose “Sensor Tag 1” from the drop-down list to set this SensorTag to be the first one.

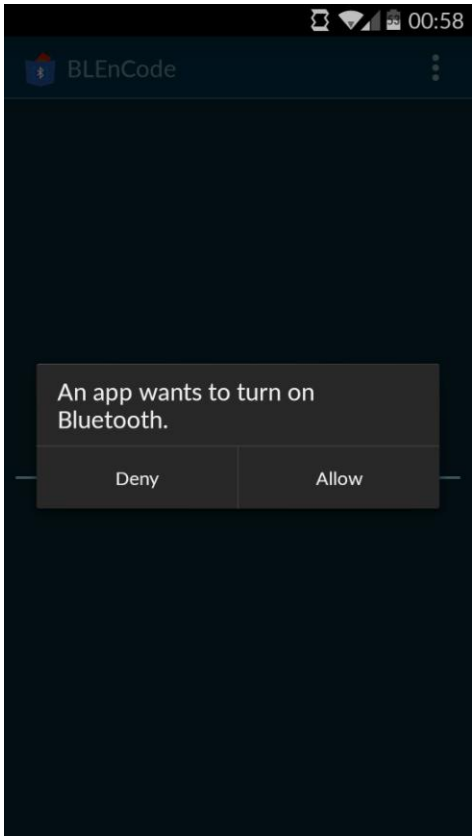
e. Now, let us change the “When” condition on the previous “When Tapped” brick. You can use the drop-down list of options to set the condition to “Sensor Tag Button Pressed”. Further options will allow you to choose which SensorTag (choose “Sensor Tag 1”) and which button (choose any of the two buttons).

f. Under the “When Sensor Tag Button Pressed” brick, add a brick “Show” from the Looks category to ensure the object appears on the screen when the SensorTag buttons are pressed.

g. Now we are done adding scripts to the app.



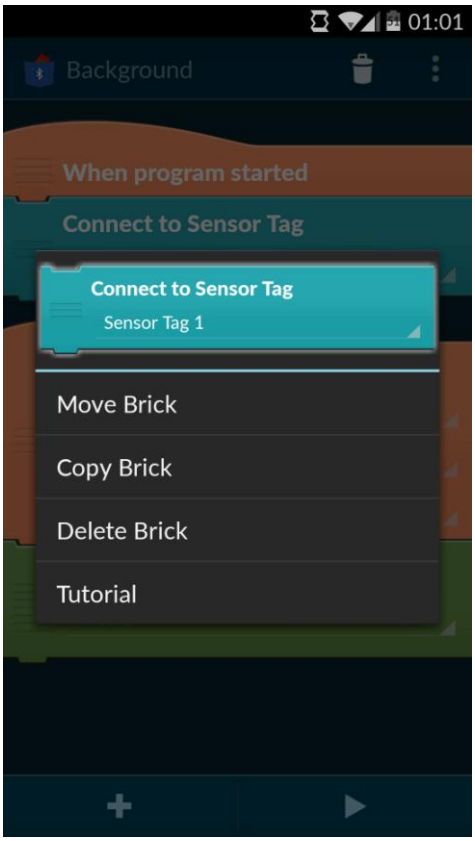
3) Click on the play button to run the app. Since, we have used a BLE Sensor brick, the app will prompt you to turn on Bluetooth. Enable Bluetooth, “Search for devices” to find the desired SensorTag and click to connect to it. The app will start once it has successfully connected to the device.

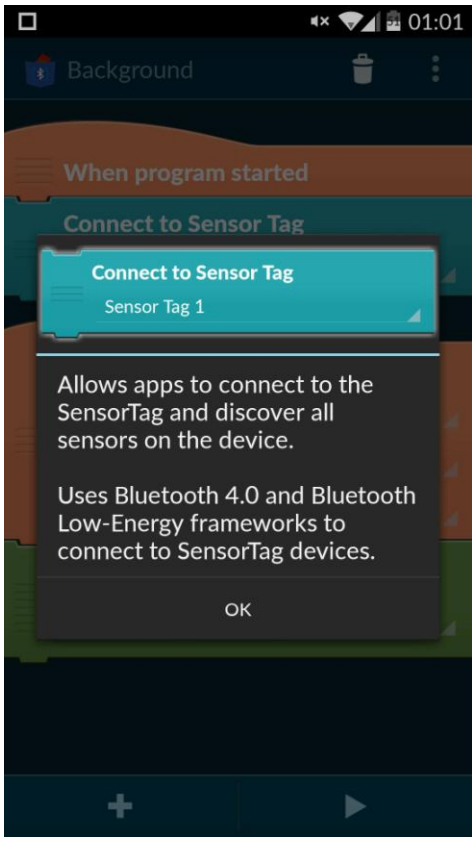


4) That’s it! Now try clicking the buttons on the SensorTag to see your image appear on the screen.

7.3 Using the in-app Tutorials

Whenever you feel unsure about how any of the bricks behave, click on it to see the relevant options associated with it. Choose “Tutorials” to know more about the usage of the brick.

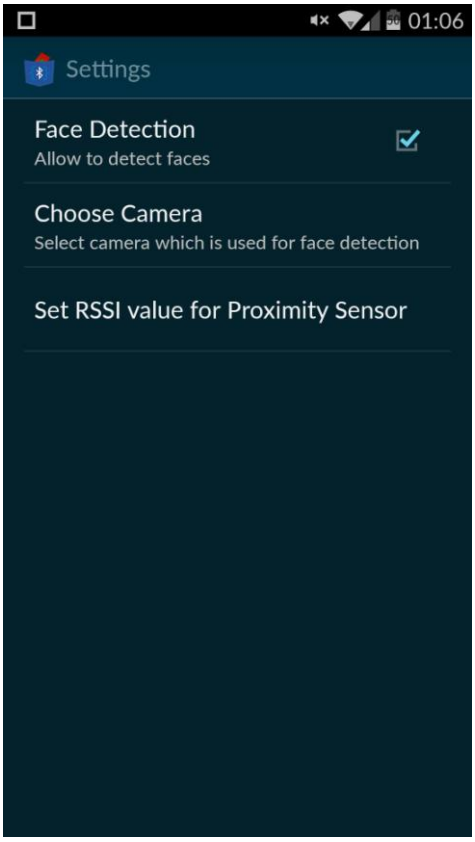


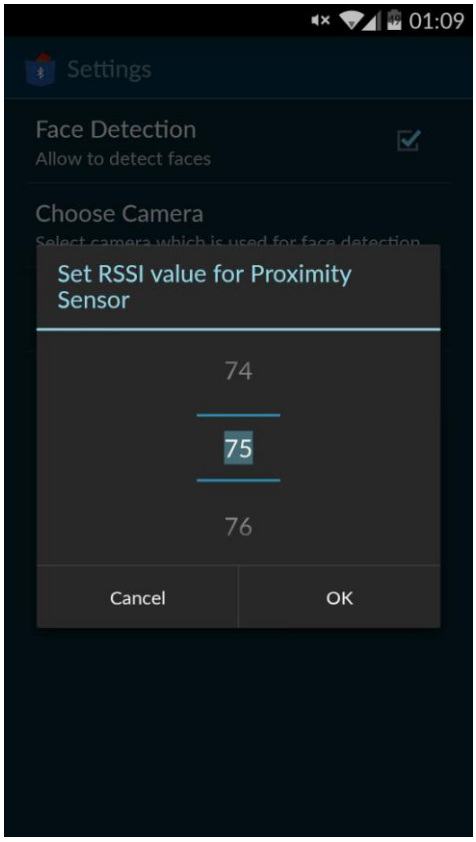


7.4 Creating Applications which use Proximity sensor

Once we have become a little more familiar with BLE bricks and functionality, let us try to create an app which uses the Proximity sensor.

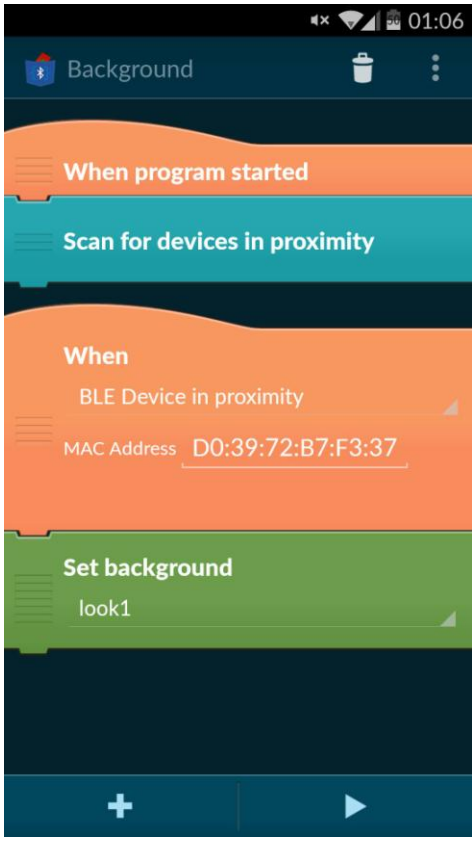
1) To do this, you will need to go the Settings Menu and choose “Set RSSI value for Proximity” from the settings. Here, you can pick the RSSI value for which the Bluetooth device will be considered “in proximity”. The RSSI value is the signal strength of the Bluetooth device. Larger values indicate a larger search radius to find devices.





2) Next, go to the app that we had previously created to add new functionality to it. Go to the “Background” and then to its “Scripts”. Here, add a “Scan for devices in proximity” brick from the BLE Sensors category, under “When Program Started” section. This will allow the program to run Bluetooth scans in the background to search for devices. Remove the previous “Connect to Sensor Tag” brick for this app.

3) Let us again change the “When” condition on the previous “When Sensor Tag Button Pressed” brick. You can use the drop-down list of options to set the condition to “BLE device in Proximity”. A further option will allow you to enter the MAC Address of the desired BLE device which you want to have in your proximity. Keep the rest of the script same.



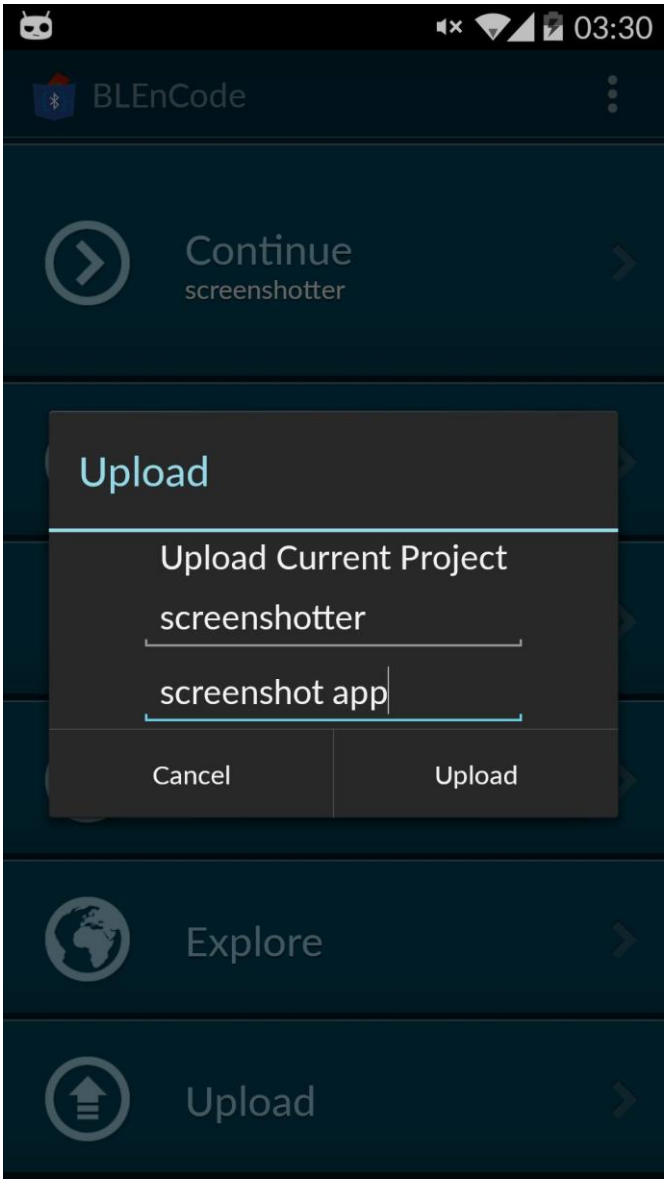
4) Again, click play to run the app. The app will prompt you to enable Bluetooth and will start as soon as Bluetooth is enabled. Now, bring your desired device into proximity of your smartphone and see your background switch!

7.5 Sharing and downloading projects from the cloud

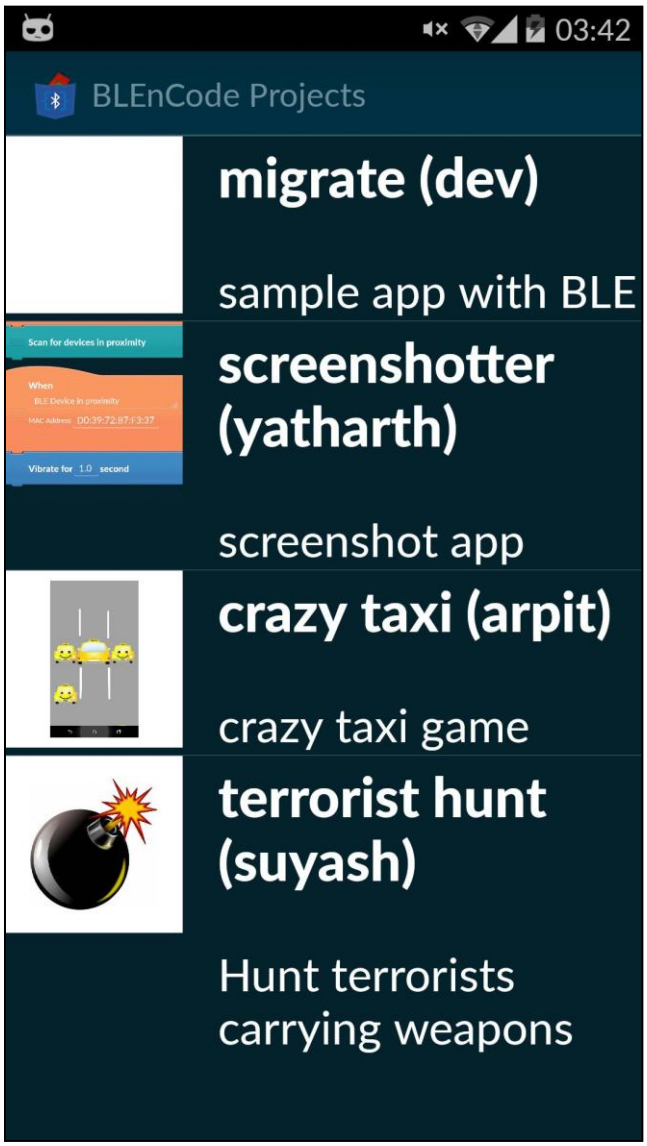
Now that we have learnt how to make various kinds of apps through BLEnCode, let us now see how to share our apps.

1) From the main menu screen, click the “Upload” option to share the current project. The first time, the app will prompt you to log into the cloud server. Enter your credentials if you already have an account with BLEnCode, otherwise click “Sign Up” to create a new account.

2) Once you have signed up to BLEnCode servers, click “Upload” again to enter the name and description for your current project. Once you have entered these details, click “Upload” and wait a few seconds while the necessary data is uploaded to our servers.



3) If you would like to explore projects created by other users n BLEnCode, click the "Explore" option on the main menu to view all the projects currently on our servers. Clicking on any interesting project will allow you to download it locally onto your device. Once you have downloaded new projects, you can check them out through the "Programs" option on the main menu.



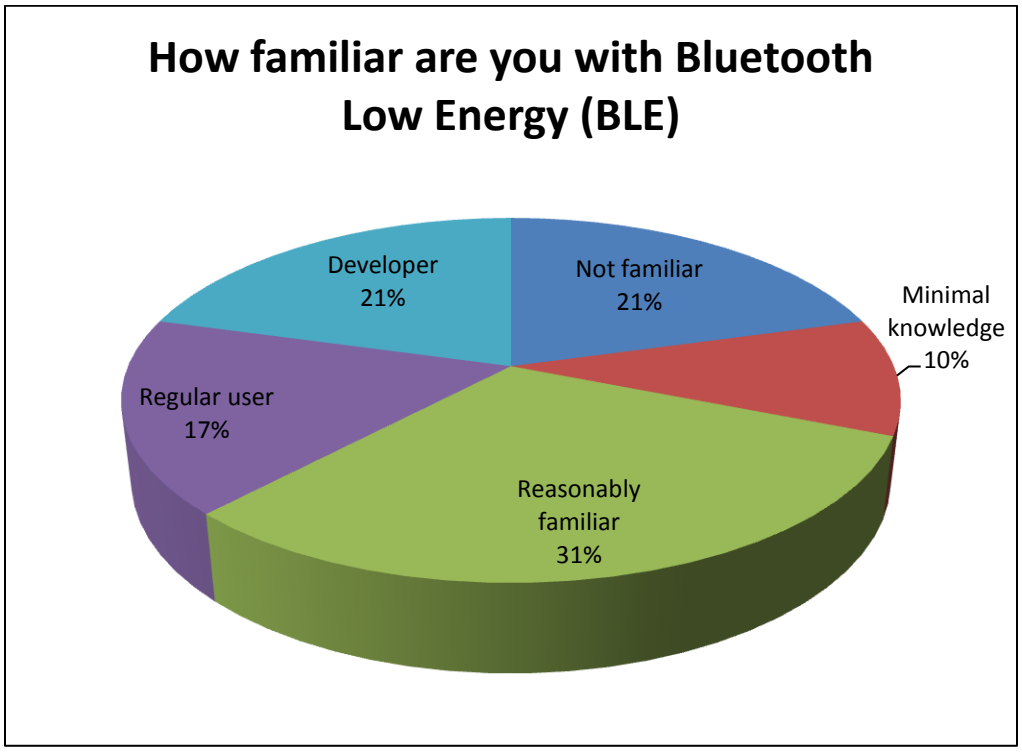
There it is. That's all we can teach you. Now, armed with this knowledge and your limitless creativity, the sky is the limit for creative innovative and interactive apps through BLEnCode.

Happy BLEnCoding!

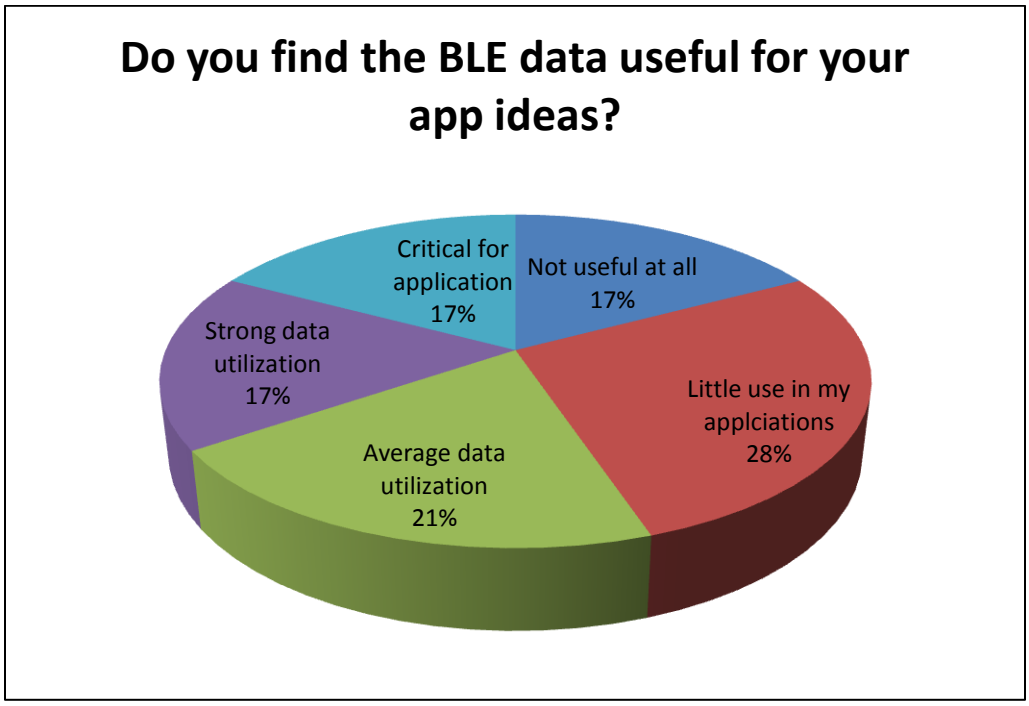
8. Pilot Users Reviews

To test the user-friendliness and potential of our app with prospective users, we invited some of our batch-mates and friends who are familiar with programming and some who are also novices in programming. We asked them to use our application and provide answers to some feedback questions compiled in a Google Form. The questions varied from general awareness of BLE and sensors to general application development and some questions were related to application development using visual programming. The answers we received have been compiled here for analysis and subsequent understanding of the benefits and deficiencies of this approach to extend programming and data utilization from BLE sensors.

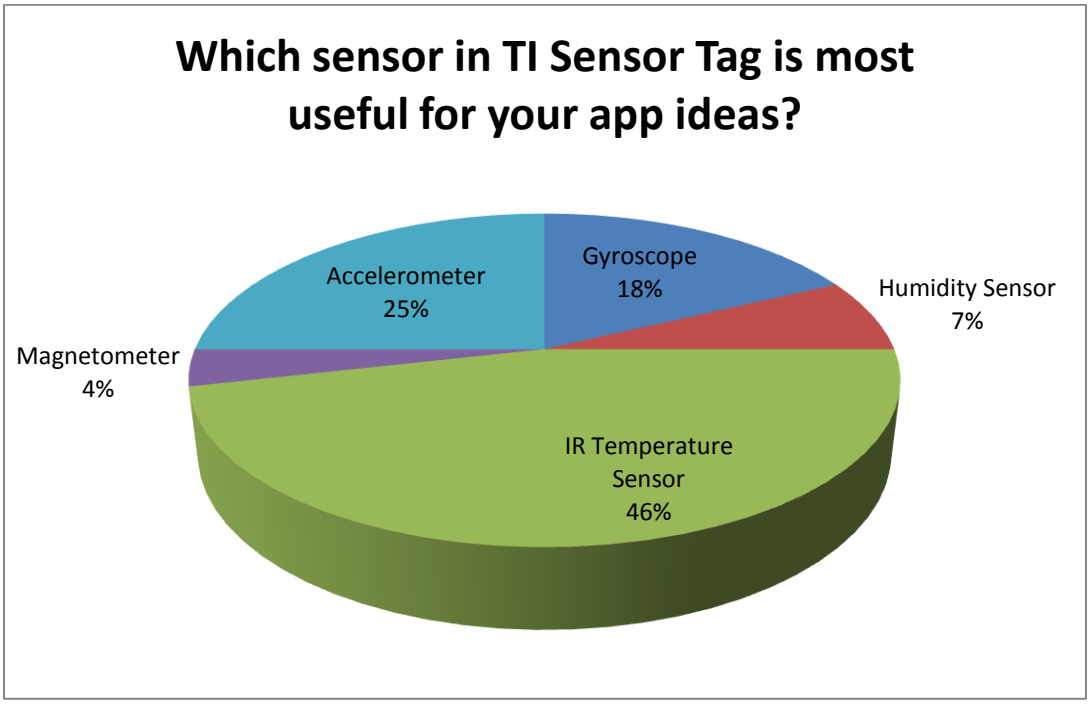
8.1 Familiarity with BLE



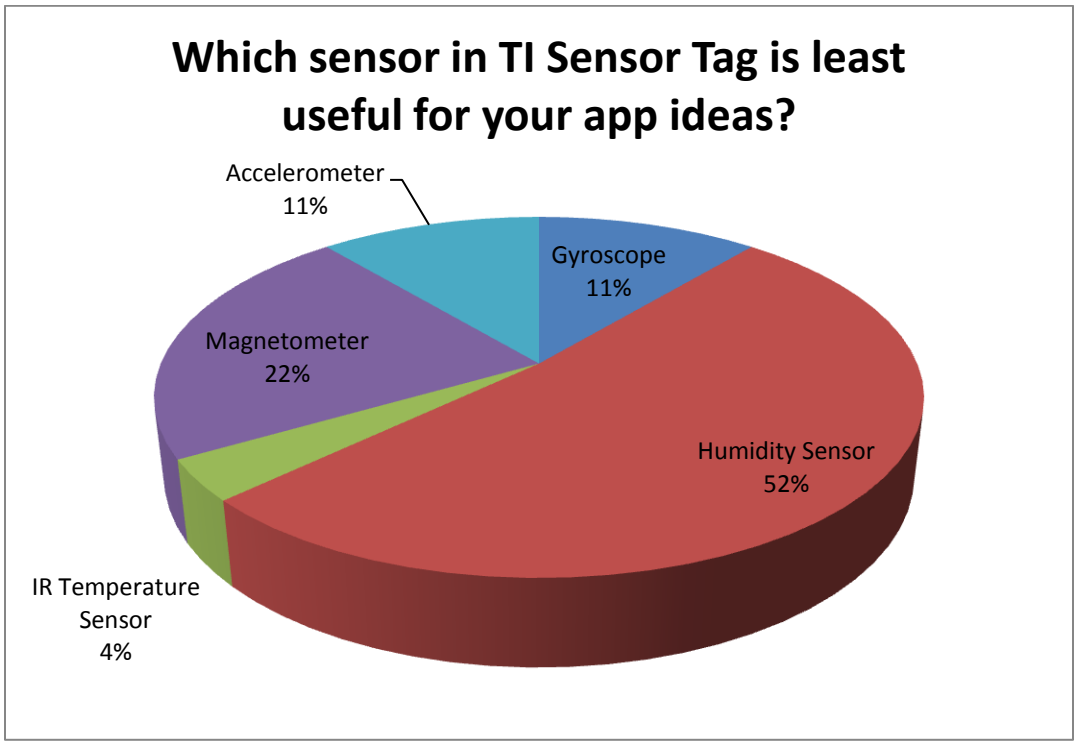
8.2 Usefulness of BLE to apps



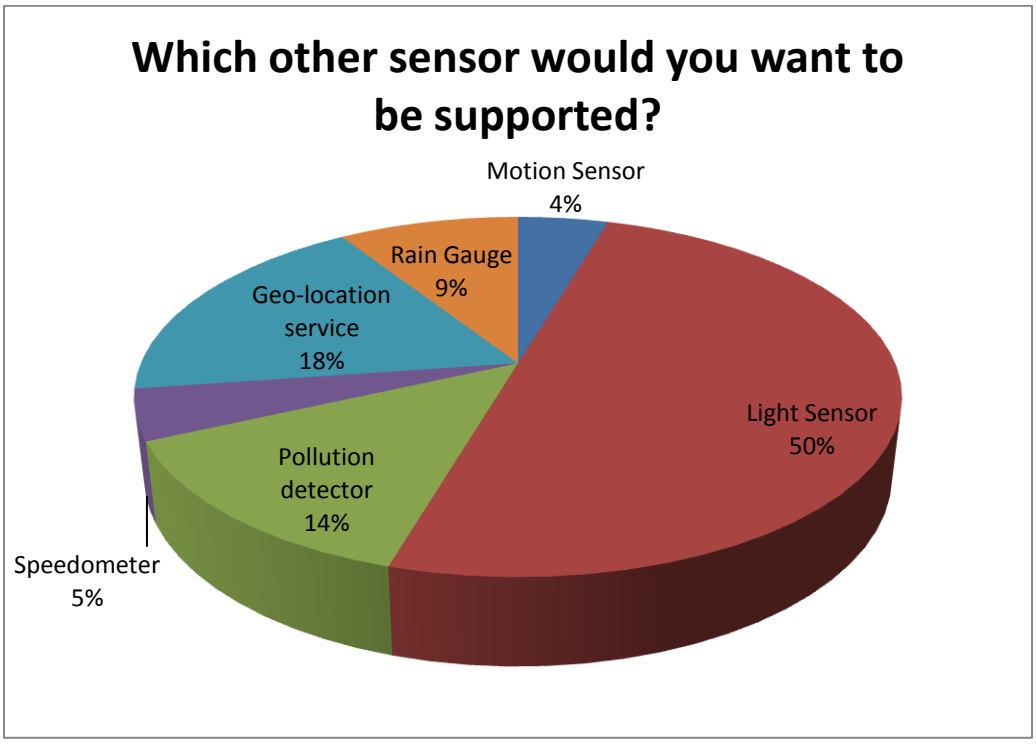
8.3 Most useful sensor on TI SensorTag



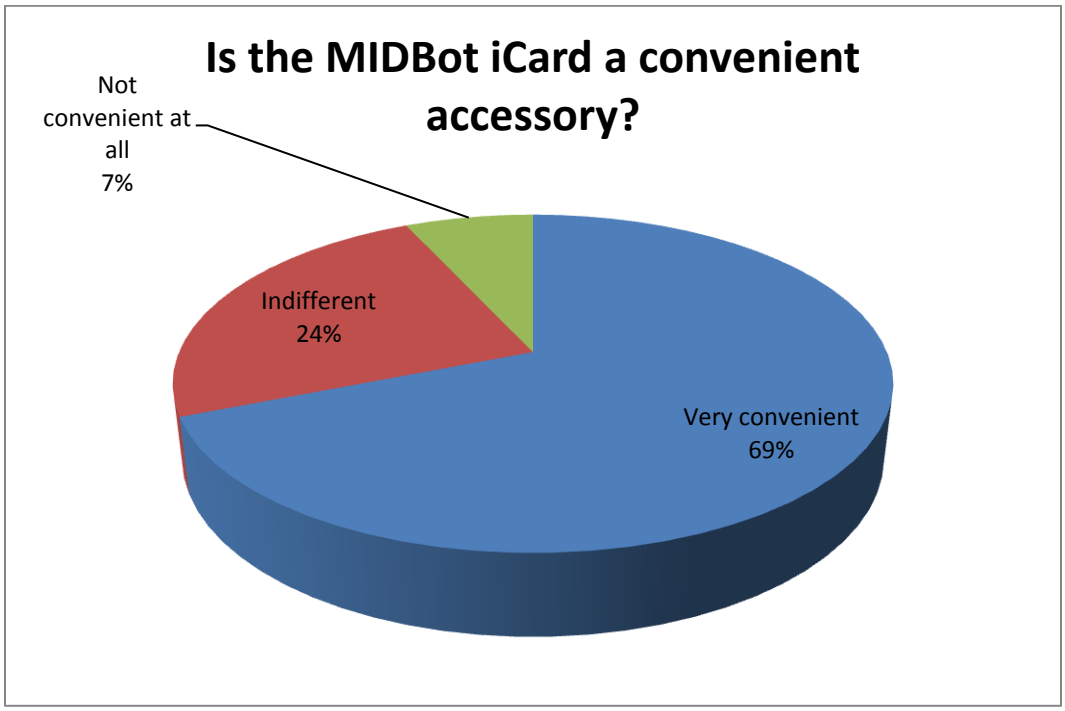
8.4 Least useful sensor on TI SensorTag



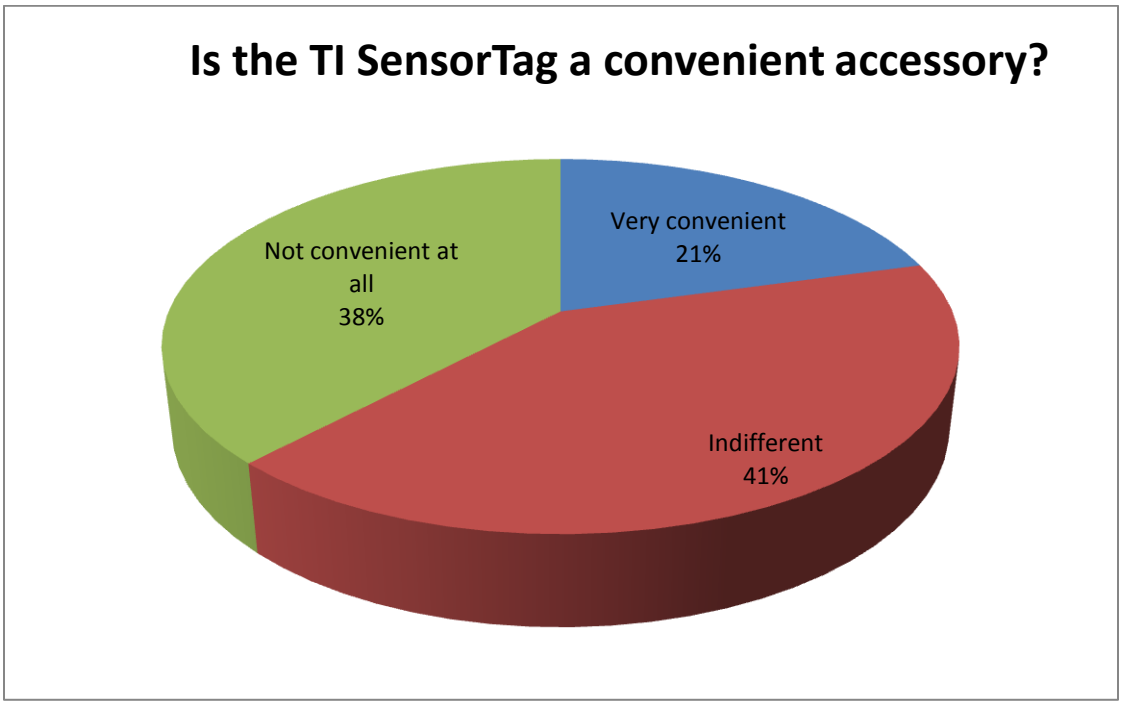
8.5 More Sensors?



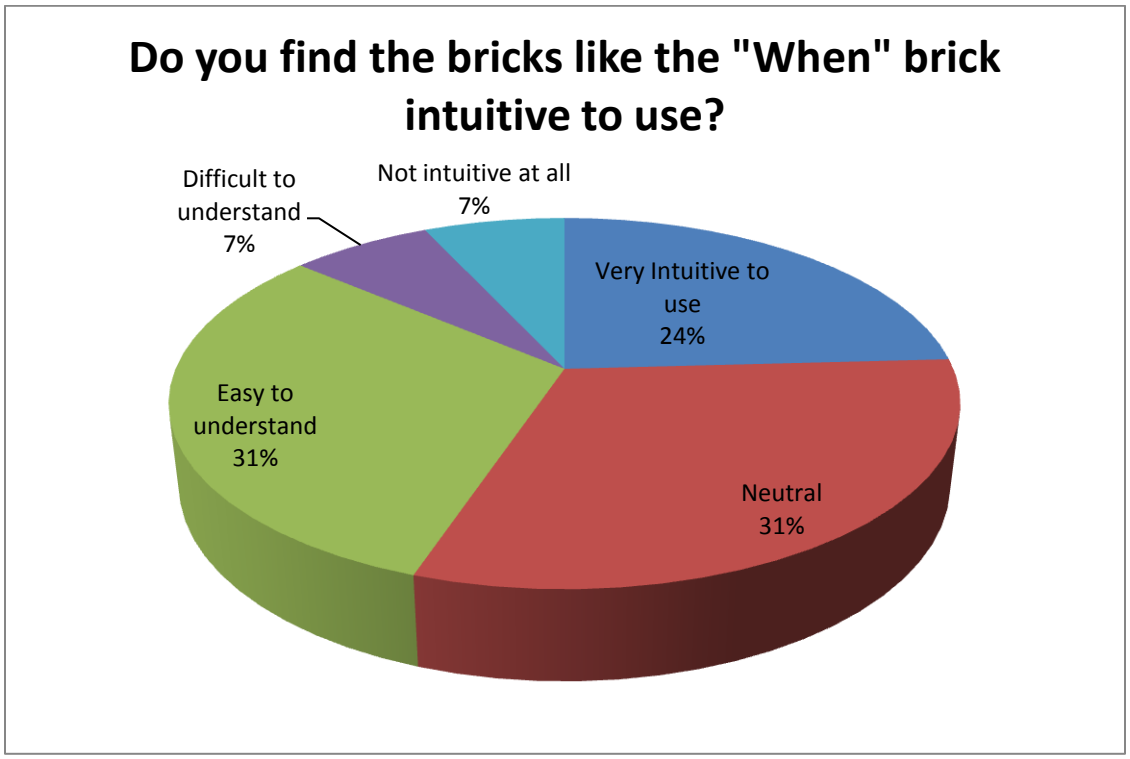
8.6 MIDBot iCard Convenience



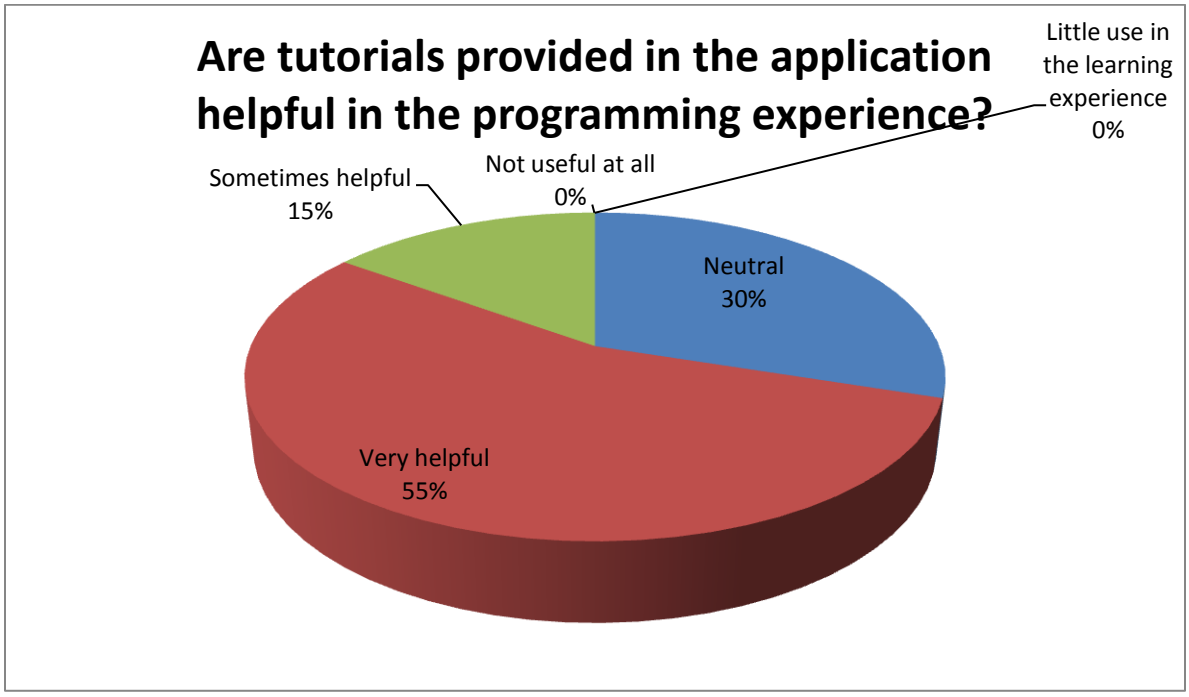
8.7 TI SensorTag Convenience



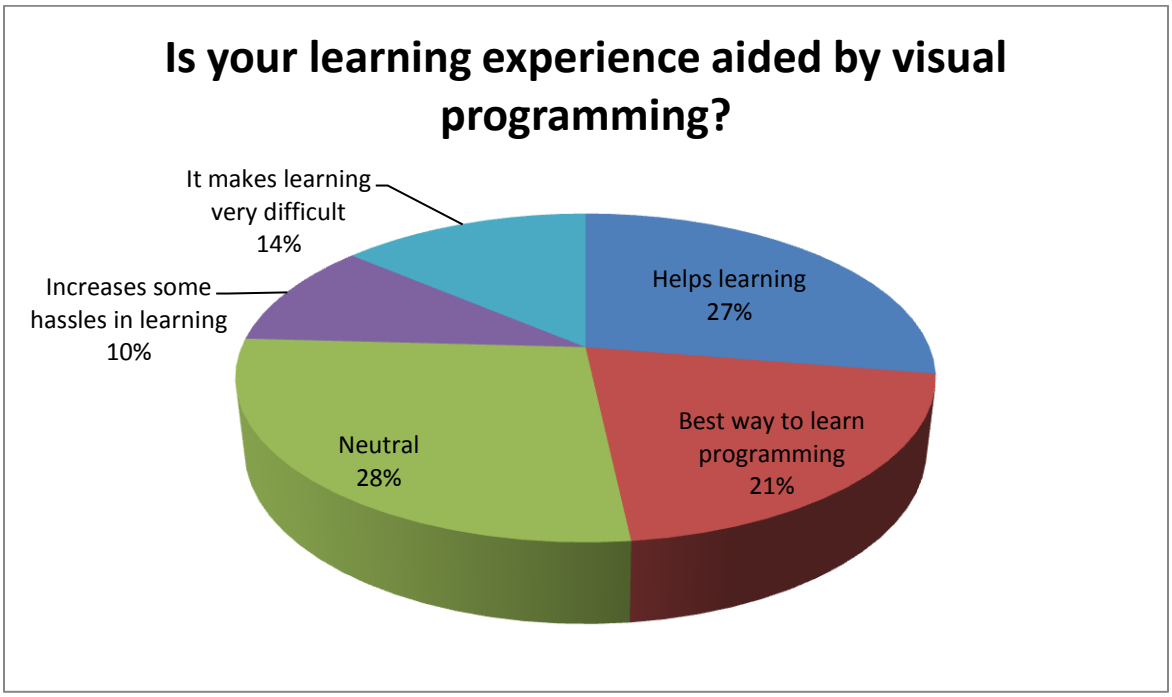
8.8 Ease-of-Use of Bricks



8.9 In-app Tutorials



8.10 Overall Learning Experience



9. Potential Applications

Presently Pocket Code alone can be used to develop utility tools and games with hardware functionalities limited to the phones' capabilities. Similarly, SensorTag also has its limitations as it can only send data to the Texas Instruments BLE applications that can only display a live feed from the sensor. Our enhanced version of Pocket Code, BLEnCode, is aimed at utilizing the capabilities of the two platforms into one that is more powerful, customizable and intuitive for users to develop applications according to their imagination. Some potential areas of focus and further implementation could be as follows:

- Personalized Monitoring System that can monitor and keep track of ambient room temperature, proximate air pressure, humidity for research and analysis in places like households, greenhouses, laboratories.

- Security Systems – gyroscope and accelerometer can be used for theft protection systems. Fire detection can be aided through an application, which sets off an alarm and/or a mass SOS if it detects temperature over a certain value.

- Lost and Found System – the RSSI data from the BLE sensor can be used to notify the user through a warning sound or LED blinking if his or her smartphone moves away from the BLE device by a certain threshold.

- Single player and Multiplayer games can be developed where the users chose their own rules to interact and compete. For example, a quiz game can be developed. Multiple SensorTag devices can be used as a buzzer and the Quiz Master can be notified through an LED blink on the BLE iCard when a team presses a button on the SensorTag as the buzzer.

- Medical Application – we can utilize sensor data like from the accelerometer and gyroscope to make applications that can aid monitoring of elderly persons and alert the relevant caretakers via a warning system defined by the user.

- Robot Control – The project deliverables can also be integrated with a mechatronic robot (for instance MIDBot iDroid) using the same smartphone application to control the robot as well as monitor the information from the BLE sensor.

- … and many more!

10. Future Developments

The application can also be rolled out to other mobile/tablet platforms such as iOS and Windows Phone. This will help us to cater to a much wider user group by removing OS-based restrictions. Moreover, this expansion of market can be further boosted by modifying the application to recognize and interact with more such BLE sensors. We have agreed that the said enhancements are out of scope as of now due to time and resource constraints.

10.1 How to extend our project

Since we have created an open-source project based on the existing Pocket Code project available to the market, we would like other developers to contribute to our project and leverage it to add further enhancements to it. We would like other developers to add more BLE Sensors into the Pocket Code framework to make it more universally accessible to a wide variety of Bluetooth Smart devices.

Our source code will be regularly published on GitHub (https://github.com/fyp14017/Pocket-Code-BLE). This will allow for the code to be accessed by developers worldwide. As developers maybe new to development on Pocket Code or using BLE devices, here is a list of general instructions to follow while making modifications to the current project:

1. Make a new Brick for the BLE device
    a. Add a new Brick to the category of "BLE Sensors" in the file CategoryBricksFactory.java
    b. Create the file containing UI logic for Brick in the Bricks package of Catroid.
2. Add actions to new Brick
    a. In the Actions package, create the file holding back-end logic of the new Brick.
    b. Add the action in the class ExtendedActions.java in the Actions package and link the Action file to the file of the Brick.
3. Register callbacks

a. The callbacks of the BLE devices are managed through the class PreStageActivity.java in the Stage package. The callbacks allow us to create a global reference for the connected BLE devices.

4. Parse data from BLE devices

    a. The parsing and manipulation of data received from the BLE Sensors is done in the SensorInfo.java class in the BLE package.

    b. The values from the sensors are held in static variables accessible through the "Formulas" if necessary.

5. Make the sensor data available to users in Scripts

    a. This can be done in various ways, for example using the Formula Editor.

    b. Add a new Sensor in the Sensors.java in the FormulaEditor package.

    c. Create the instance of the sensor through the InternFormulaKeyboardAdapter.java file.

    d. Retrieve the value of the sensor through SensorHandler.java, which links to the SensorInfo.java class in the BLE package.

    e. Add an entry into a HashMap in the InternToExternGenerator.java file to map the Sensor to the string displayed in the Formula Editor UI.

    f. The values of the Sensors can also be made available to the users through other ways like creating another new Brick.

11. BLEnCode – Legal

Licenses of the BLEnCode system

The outline and language of this license has been taken from the following site: http://developer.catrobat.org/licenses to conform to the original license requirements of the Catrobat System.

All source code of the software constituting the BLEnCode system, including but not limited to, our own code that incorporates Bluetooth Low Energy peripherals in the program and the other enhancements, interpreters and compilers made by the Catrobat team, are free software and you are free to modify/redistribute them under the terms of the GNU Affero General Public License as published by the Free Software Foundation (version 3 or above).

The names and logos of the BLEnCode project and subprojects are copyrighted by the BLEnCode project and cannot be used in derivative work. You are however allowed to use the names and logos created by the FYP14017 team in non-derivative work such as, e.g., books, brochures, or descriptive web pages. Please include a note that says "BLEnCode is developed by FYP Group 14017. Please visit http://i.cs.hku.hk/fyp/2014/fyp14017/ for more information on the relevant licenses and project information. ".

In addition to conforming to the above conditions, you must also conform to all conditions mentioned by the Catrobat System team (more information below) and conform to all the license conditions mentioned at their licenses page (http://developer.catrobat.org/licenses).

Disclaimers:

We hereby acknowledge that BLEnCode is an independent derivative work originating from the Catrobat project and utilizes their source code for the enhancement made in the direction of inclusion of Bluetooth Low Energy devices. We give full credit to the development team of the Catrobat System and following is our conformance to their license condition.

“Catrobat makes it easy to program your own interactive stories, games, and animations --- and share your creations on the web. As you create and share Catrobat programs, you learn to think creatively, reason systematically, and work collaboratively. All parts of the Catrobat system are developed by the Catrobat project. They are available as free open source software from http://developer.catrobat.org/"

"This work was created using programs developed by the Catrobat project. Catrobat makes it easy to program your own interactive stories, games, and animations --- and share your creations on the web. As you create and share programs with Catrobat, you learn to think creatively, reason systematically, and work collaboratively. Catrobat is available as free open source software from http://developer.catrobat.org/ and is distributed by the Catrobat project under the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version when it becomes available. For the parts of the Catrobat system necessary for running a Catrobat program, e.g., the Catrobat Interpreter, the GNU Affero General Public License is supplemented by an additional permission under section 7 of version 3 of the License. This exception allows to convey a runtime version of a Catrobat program, for which you otherwise legally hold the necessary rights, in combination with the parts of the Catrobat system necessary for running that Catrobat program, under terms of your choice, consistent with the licensing of the independent parts of the corresponding Catrobat program."

By means of inheritance for the parent project being Catrobat, our source code also includes third party code and resources as follows:

libGDX: <http://libgdx.badlogicgames.com>

License information: Apache 2

License <https://raw.github.com/libgdx/libgdx/master/gdx/LICENSE>

Copyright © 2011-2012 See <https://github.com/libgdx/libgdx/tree/master/gdx>

XStream: <http://xstream.codehaus.org>

License information: <http://xstream.codehaus.org/license.html>

Toolbar custom icons 1.0 facilities from Gentleface custom icon design:

<http://gentleface.com/free_icon_set.html>

License information: Creative Commons Attribution-NonCommercial 3.0: <https://creativecommons.org/licenses/by-nc-nd/3.0/>

Copyright © 2006-2011 Gentleface.

MINDdroid: <https://github.com/NXT/LEGO-MINDSTORMS-MINDdroid>

Copyright © 2010/11 Guenther Hoelzl, Shawn Brown

Copyright © 2010/11 LEGO System A/S, Aastvej 1, DK-7190 Billund, Denmark

MINDdroid is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MINDdroid is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

See the GNU General Public License for more details <http://www.gnu.org/licenses/>

Sparrow: <https://github.com/Gamua/Sparrow-Framework>

Simplified BSD License

Copyright 2013 Gamua. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY GAMUA "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL GAMUA OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of Gamua.

Java HTTP Request Library:

<http://kevinsawicki.github.io/http-request/>

Copyright (c) 2011 Kevin Sawicki.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

FlowLayout (part 1):

<https://code.google.com/p/devoxx-schedule/source/.../FlowLayout.java>

FlowLayout (part 2):

<https://github.com/ApmeM/android-flowlayout>

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ARTEM VOTINCEV BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Parrot AR.Drone Open API: <https://projects.ardrone.org/>

Copyright (C) 2007-2011, PARROT SA, all rights reserved.

Licensed under Parrot AR.Drone Development License v2.1

DISCLAIMER

The APIs is provided by PARROT and contributors "AS IS" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall PARROT and contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

References

- Catrobat: http://developer.catrobat.org/
- Pocket Code: https://pocketcode.org/
- Pocket Code Source Code on GitHub: https://github.com/Catrobat/Catroid
- Texas Instruments Sensor Tag: http://www.ti.com/ww/en/wireless_connectivity/sensortag/index.shtml?INTC=SensorTag&HQS=sensortag-bt
- Bluetooth Smart: http://www.bluetooth.com/Pages/Bluetooth-Smart.aspx
- Technology behind Bluetooth Smart: http://www.bluetooth.com/Pages/low-energy-tech-info.aspx
- TI CC2541: http://www.ti.com/tool/cc2541dk-sensor
- SensorTag User Guide: http://processors.wiki.ti.com/index.php/SensorTag_User_Guide
- Android API for Bluetooth Low Energy: https://developer.android.com/guide/topics/connectivity/bluetooth-le.html
- Event-Driven Programming: http://en.wikipedia.org/wiki/Event-driven_programming
- Parse: https://www.parse.com/products/core
- GitHub: http://en.wikipedia.org/wiki/GitHub

We would like to take this opportunity to thank Dr. Vincent Lau for his continuous guidance.