LEARNING ASSESSMENT DATA COLLECTION FROM EDUCATIONAL GAME APPLICATIONS

A thesis submitted
to Kent State University in partial
fulfillment of the requirements for the
degree of Masters of Computer Science

by

Poonam Songar

December 2012
Thesis written by
Poonam Songar
B.E., Rajiv Gandhi Proudyogiki Vishwavidyalaya, 2008
M.S., Kent State University, 2012

Approved by

Arden Ruttan, Advisor
Javed I. Khan, Chair, Dept. of Computer Science
Timothy Chandler, Dean, College of Arts and Sciences
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>.......................................................................................................................... vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>.......................................................................................................................... viii</td>
</tr>
</tbody>
</table>

## CHAPTER 1  INTRODUCTION ......................................................................... 1

1.1 Educational Games .................................................................................. 1

1.2 Educational Games in STEM .................................................................... 2

1.3 STEM Educational Gaming Environment and the Design Principles ........ 3

1.4 Thesis Goal and Contributions – Learning Assessment Data Collection .... 5

1.4.1 “Packaged Solution” or Framework ..................................................... 6

1.4.2 Demo App & Assessment Data Collection, VL3.901_01 (Android) .......... 7

1.4.3 Why Mobile Educational Game app? ................................................... 7

1.5 Thesis Background .................................................................................. 8

1.5.1 Educational Mobile Game Apps in Material Science ......................... 9

1.5.2 Thesis Contribution to mobile game app based STEM education .......... 10

## CHAPTER 2  THE FRAMEWORK AND IT’S COMPONENTS .......................... 11

2.1 Data Collection Framework .................................................................... 11

2.1.1 Systematic Framework ....................................................................... 12

2.1.2 Framework Components .................................................................... 14

2.1.2.1 Educational Game Application .................................................... 14

2.1.2.2 Learning Assessment Data ............................................................ 16

2.1.2.3 Data Collection API .................................................................... 18
2.1.2.4 Remote Database ........................................................................................................ 19
2.1.2.5 Interface to the remote database ............................................................................... 19

2.2 Java API & The Database .................................................................................................... 20
2.2.1 Data collection API ......................................................................................................... 21
  2.2.1.1 Mutually Agreed Data Tags ..................................................................................... 22
  2.2.1.2 Functional Specification ......................................................................................... 22
2.2.2.1 Objective of the Remote Database ............................................................................. 28
2.2.2.2 Designing the Database – Design Principles ............................................................ 28
2.2.2.3 Database Tables ......................................................................................................... 29
2.2.2.4 Storing App Data – Server Side Operation (POST variables) ................................. 30

CHAPTER 3 DEMO APPLICATION ................................................................................................. 32

3.1 Background of the work ..................................................................................................... 32
  3.1.1 Virtual Lab 3.901 - Lesson 1 ....................................................................................... 33
    3.1.1.1 Simulation .............................................................................................................. 33

3.2 Walkthrough of Demo Game Application ............................................................................ 35

3.3 Systematic Application of the framework ............................................................................. 38
  3.3.1 Game Application Development ................................................................................. 38
  3.3.2 Learning Assessment Data Collection ........................................................................... 39
    3.3.2.1 Collect Data in a local file ...................................................................................... 39
    3.3.2.2 Post Data to the Remote Database Server ............................................................. 39
    3.3.2.3 Updating Game App User Interface .................................................................... 40
    3.3.2.4 Storing app data in remote database .................................................................. 40
LIST OF FIGURES

Figure 1: Systematic Framework for Learning Assessment Data Collection.

Figure 2: Learning Assessment Data Collection.

Figure 3: Transporting data, game app -> remote database.

Figure 4: Sample file for local storage on player's android device.

Figure 5: Posting game app data to remote database server.

Figure 6: Receiving data from remote database server.

Figure 7: MatDL Virtual Lab, Lesson 1 Simulation.

Figure 8: Screen 1 - Challenge Question Form.

Figure 9: Screen 2 - Feedback & Options Page.

Figure 10: Screen 3 - Simple Game Challenge, kicking the box.

Figure 11: Screen 4 - Final Feedback & Summary Page.

Figure 12: Steps to download android app from a web source.

Figure 13: Web based query interface to learning assessment database.
DEDICATION

To the dear God, my family and friends for the unconditional love and support.
ACKNOWLEDGEMENTS

I would like to express deepest appreciation for Dr. Arden Ruttan and Professor Laura Bartolo for their ideas, flexibility and guidance throughout the entire course of this Thesis work. It was through their support, feedback and flexibility that this work became possible. I would like to thank you both for making yourself available for our meetings over weekends. Our meetings not only gave me the opportunity to ask relevant questions but also helped me to stay motivated and focused.

I take this opportunity to thank the National Science Foundation (NSF) for support of the Materials Digital Library NSF (0532831). As a graduate student on this project team I have received tuition and stipend support from Center for Materials Informatics (CMI) Lab at Kent State University to complete my studies and thesis in Computer Science. This Thesis work has also benefited from contributions of other participating institutions, most notably Massachusetts Institute of Technology.

I would further like to thank Professor Donald Sadoway for his simple yet brilliant use of a rectangular cardboard box during his lectures at MIT, course 3.901, for introducing the concept of chemical states and energy barrier. The demo game application developed as part of this thesis is based on his idea of a rectangular box to illustrate the concepts of chemical states and energy barrier. Hence, his contribution is significant for this Thesis work.

Last but not the least I would like to thank almighty God for each opportunity he blessed me with to learn, to experiment and build upon my abilities.

Finally, I want to thank my mother for her unshakable faith in me and her guiding vision, which constantly encourages me to aim higher than ever.
DISCLAIMER

This work is supported by the National Science Foundation 0532831, 0817432. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.
CHAPTER 1

INTRODUCTION

1.1 Educational Games

Gaming industry is giant and mature. Based on industry facts reported by Entertainment Software Association [7] regarding America’s entertainment software industry, “Consumers spent $25.1 billion on video games, hardware and accessories in 2010.” Also, serious games and simulations have been an important part of instructional technology over past several years. “Major developments in educational software in the early and mid 1990s were made possible by advances in computer hardware,” [22] especially multimedia graphics and sounds further allowed for creation of rich multimedia educational games. Author Oblinger D. [23] writes “on a soccer field, at a table with a deck of cards, in front of a computer screen—games catalyze learning. Most games were not designed to be educational, yet they are immersive, experiential learning environments. Ignoring the educational power of games dismisses a potentially valuable learning tool. Digital games, in particular, carry enormous potential to draw students into a topic and help them learn information, skills, attitudes, and ways of thinking.”
1.2 Educational Games in STEM

The report published by National Academy Press [15] highlights “at a time when scientific and technological competence is vital to the nation’s future, the weak science achievement of U.S. students reflects the uneven quality of science education.” The report also identifies that “many experts call for a new approach to science education, based in cognitive research. In this approach, teachers spark students’ interest by engaging them in investigations, helping them to develop understanding of both science concepts and science processes while maintaining motivation for science learning.”

In order to explore Games and Simulation as a probable new approach to science education National Science Foundation and Flora Hewlett Foundation charged the National Research Council to form a committee to “plan and conduct a two-day workshop to explore the connections between what is known about science learning and computer gaming and simulations, the role computer gaming and simulations could play in assessing learning, and the pathways by which they could be used on a large scale” [15]

The final report written by the committee [15] states that “evidence for the effectiveness of games for supporting science learning is emerging but is currently inconclusive” (p. 54) however based on investigation of several successful educational games developed as research projects and industry products e.g. [6], [24] and others, the report committee “views simulations and games as worthy of future investment and investigation as a means to improve science learning.” (p. 2)
1.3 STEM Educational Gaming Environment and the Design Principles

The report [15] investigated serious games “designed to accurately model science or simulate scientific processes” (p. 10). One of the criterion satisfied by the investigated serious games is that the “interactions within the virtual world of the game are governed by established scientific principles” (p.10). Designing educational games qualifying these criterions is very challenging because educational game designers need to design such games to be fun & engaging while conforming to scientific principles. Additionally, educational games should have clear educational goals to be accomplished via gameplay. Further discussion of following relevant STEM game projects is helpful to gain an informed perspective on the design aspect of STEM educational games and its environment.

In the book [14] author provides a list of 11 principles for designing educational gaming environment:

1. It would have to be a real world environment
2. The learning environment should be as similar as possible to the environment in which people will use what they learn
3. It is important to provide unlimited numbers of learners with exactly the same learning environment
4. The learning environment should be designed to allow a learner to repeat an experience facilitating a post mortem analysis of what took place
5. The environment should be designed to facilitate variation along a single dimension or a given selection of dimensions
6. There should be some uniform means of assessing student’s performance
7. The environment should store and present the students with pre-planned learning experiences, some of them in a particular order
8. It should be possible for the student to explore new concepts and practice new techniques at his or her own pace
9. The student should be given immediate positive (and ideally public) feedback for any success that is commensurate with that student's current level of attainment.

10. There should be sufficient "cost" to getting something wrong to motivate correction, but not so great that it leads to the student losing heart and giving up.

11. The learning environment should, if possible, provide an enjoyable and stimulating experience.

For a reasonably good educational gaming environment most principles should apply. However, researchers can pick and group these design principles to experiment and to further their understanding of educational gaming environment. This thesis work recognizes that the ability to assess student learning in a gaming environment is vital to further the research in the field of educational gaming and finds motivation in Devlin’s educational gaming environment design Principle #6, “There should be some uniform means of assessing student’s performance.”

[12] suggests that:

Much more research is needed on how assessment can be built into appropriate serious games in science to demonstrate that learning can be assessed in reliable and valid ways. Until it has been established that assessment can be built into science games, the field cannot move on to address the question of how games can be used in formative assessments in the classroom or, perhaps, in accountability assessments to provide evidence of complex skills that cannot be assessed by existing formats. (p. Pg 19)

A demonstration of learning and assessment of student learning via a computer simulation is provided by The River City Project [6]. The River City Project “is an interactive computer simulation for middle grades science students to learn scientific inquiry and 21st century skills.”

This computer simulation goes beyond usual evaluation methods such as pre and post surveys, focus groups, interviews with students and questionnaires to assess student learning. River City project has an underlying database which records student utterances, behavior and movements when they are in River City. The idea is to collect effectiveness data from gameplay/simulation to assess student learning. However, River City platform is provided by ActiveWorld [13] which is
not an open source platform and hence the details of how data collection mechanism is embedded within the project are not published. Other popular games/simulations that deployed some data collection mechanism in an educational gaming environment are also built using non-open source commercial platforms such as [24] and [25]

1.4 Thesis Goal and Contributions – Learning Assessment Data Collection

Educational gaming is still an emerging field with potential for advancing STEM education. To harness this potential education technology community needs to conduct extensive research and experimentation. The work to be done is clearly interdisciplinary such that educators, game designers, technology enablers and students have to closely work together. To facilitate an accelerated growth of this field, computer professional need to provide the research community with flexible tools/technologies that can serve as research platforms for conducting relevant research and experimentation.

The main goal of the thesis is to propose and demonstrate an open source mechanism for collecting learning assessment data from an educational game.

The core idea of the thesis is to introduce a systematic framework and supporting tools for deploying a data collection mechanism in the educational game such that assessment data can be collected from the game and stored in a separate media for further analysis.

Hence, deliverable of this thesis is not a single tool or a list of research findings but a “packaged solution” for deploying assessment data collection mechanism in educational games. This “packaged solution” can be summarized as below individual components. Details of the individual components of this “packaged solution” are discussed in the following chapters.
1.4.1 “Packaged Solution” or Framework

This framework provides guidelines on how to go about deploying learning assessment data collection mechanism in an educational game app. Components that fit together to have such a data collection mechanism in place are listed below.

1. *Educational Game App:* An educational game app with clear educational goals. Also, need to determine the platform for app development: mobile, desktop, web based, etc. Finally, strategy to distribute the game app to players: app store, download via website, etc.

2. *Establishing Assessment Data:* Course instructor will need to devise a mechanism for assessing student learning for the educational goals established for the educational game app. This assessment mechanism can be set of challenge questions, game score, etc.

3. *JAVA API:* This API facilitates two communication of app and the remote database.


5. *Interface to Remote Database:* Once learning assessment data is being collected from the app, course instructor can analyze this data. To encourage analysis and use of the collected data it is recommended to provide an easy to use and non-technical interface to the database such as web based query selection screen, using reporting tools that sit on top of this database, etc.

To aid the understanding of this “packaged solution” a working example is also provided in form of a demo application with learning assessment data collection mechanism based on the framework. Review of the thesis along with demo app code is valuable to understand the framework concepts & components necessary to implement assessment data collection mechanism for any educational game app.
### 1.4.2 Demo App & Assessment Data Collection, VL3.901_01 (Android)

The demo educational game app is an Android game app called VL3.901_01. The app is based on MatDL MIT Virtual Lab 3.901 [http://matdl.org/jars/3.091/lesson01/index.html](http://matdl.org/jars/3.091/lesson01/index.html). Further details about the game app and its learning assessment data collection mechanism are discussed in Chapter 3.

### 1.4.3 Why Mobile Educational Game app?

In the past educational simulations and games were mostly developed for desktop and other bulky immobile consoles. However, mass popularity of highly capable & sophisticated Apple and Android mobile devices has invigorated efforts in mobile device industry to roll out commercially viable educational applications. Also, academic research focused on mobile educational game apps has gained momentum worldwide.

Several case studies of using mobile device as a platform for dissemination of educational resources have reported positive findings over the years such as [1], [2] & [3]. On per case basis these studies have shown that learning can occur via mobile platform. In the Augmented Learning book description [5] on the website [4] author’s argument regarding educational mobile games are summarized as below.

In the book Augmented Learning [5], Eric Klopfer describes the largely untapped potential of mobile learning games—games played on such handheld devices as cell phones, Game Boys, and Sony PSPs—to make a substantial impact on learning. Examining mobile games from both educational and gaming perspectives, Klopfer argues that the strengths of the mobile platform—its portability, context sensitivity, connectivity, and ubiquity—make it ideal for learning games in elementary, secondary, university, and lifelong education.
While simple mobile games such as Fire Rescue Math [3] can “facilitate numeric skill development” more complex simulations such as The River City Project [6] if extended to mobile platform can help learners grasp difficult to acquire skills such as “scientific inquiry, hypothesis formation and collaboration.”

Based on industry facts reported by Entertainment Software Association [7] on its website regarding America’s entertainment software industry “fifty-five percent of gamers play games on their phones or handheld device.” Also, based on reports of two significant independent research organizations, [8] and [9] worldwide mobile gaming industry is expected to boom and predicted to hit figures along the lines of $11 billion by 2014.

Given the current and future prospects of mobile gaming industry and supporting evidence for effective use of mobile platform for dissemination of educational resources, educational mobile game apps are worth investigation.

1.5 Thesis Background

This work is part of NSF funded National Science Distributed Learning (NSDL) Materials Digital Library (MatDL) Pathway. As per the website of Materials Digital Library Pathway project [10]

The NSDL MatDL Pathway aims to provide stewardship for content and services needed across the MS community and in particular for its targeted audience of materials undergraduate and graduate students, educators, and researchers by offering various services such as Repository, MatForge, MatDL Wiki, Virtual Labs and other services.

One of the major goals of the Project MatDL is to increase contribution of resources to repository by academic groups, institutions & organizations that are actively pursuing research in Materials Sciences. To increase participation and contribution MatDL strives to provide Materials Science
community with infrastructure services such as Version Control System, Expert Wiki, Virtual Labs and other collaborative efforts so as to integrate repository submissions into the workflow of contributors and facilitate contributions. All collaborating institutions such as Massachusetts Institute of Technology (MIT), Carnegie Mellon University (CMU) and Kent State University (KSU) are prime contributors of repository resources. The demo mobile educational game app VL3.901_01 developed as part of this thesis work, will also be submitted as an educational resource to the MatDL repository.

1.5.1 Educational Mobile Game Apps in Material Science

Materials Science is an interdisciplinary field and often it is required of scientists from various disciplines working together to have a basic understanding of concepts in each other disciplines. Hence, having educational mobile gaming apps to teach materials science concepts is an interesting idea. The mobile game apps can provide target users such as materials scientists, undergraduate students and (or) graduate students with the ability to learn desirable concepts as and when needed.

Educational game app VL3.901_01 is an exciting step for MatDL because it takes the project from web to mobile platform. Successful rollout of VL3.901_01 will provide MatDL technical know-how of rolling out more educational mobile game apps in the future, opening a door to endless possibilities with what can be achieved in Materials Science education via mobile gaming apps. Also, assessment data to be collected from the published app will provide researchers an opportunity to investigate a case for systematic use of mobile games for learning in STEM disciplines more specifically Materials Science.
1.5.2 Thesis Contribution to mobile game app based STEM education

Implementation model of game app VL3.901_01 can be followed to rollout mobile educational game apps for other STEM disciplines with learning assessment data collection mechanism. Hence, author considers this work as an important contribution to advance mobile game apps based STEM education.

This chapter introduced the “packaged solution” or framework for collecting assessment data from educational game app and the demo application, which implements this framework. In the next chapter components of the “packaged solution” or the framework are discussed in detail.
CHAPTER 2

THE FRAMEWORK AND IT’S COMPONENTS

2.1 Data Collection Framework

As discussed in Chapter 1, Section 1.2 there is not sufficient evidence to conclude that educational games are effective, without such evidence it is difficult to build a strong case for using educational game apps in mainstream education or to advocate for huge investments in educational gaming research. In order to determine a game's educational effectiveness, data about player’s successes, failures, usage patterns—all of which is essentially learning assessment data—needs to be collected and analyzed.

In order to collect learning assessment data from an educational game, it is necessary set of learning assessment data criteria, a sufficiently large pool of game players, a remote database to collect this data and a set of easy to use functions that can collect appropriate data from the game and store it in a unified remote database. Also, it is nice to have a user-friendly interface to view data stored in the remote database. So the key requirements to build a data repository that can be analyzed to determine the effectiveness of an educational game are as below.

1. Educational Game App with sufficiently large player pool
2. Learning Assessment Data Criteria
3. A Remote Database
4. Data Collection Application Programming Interface (API)

5. Interface to Remote Database for Education Researcher

Based on number of required components and the interdependency of these components, collecting learning assessment data from the game can be a complex task. To simplify the task of implementing learning assessment data collection, a systematic framework encompassing these components is discussed in the following section.

2.1.1 Systematic Framework

An app creator should approach building a data collection mechanism for a game application in an orderly manner as discussed below.

1. **Brainstorming**: Educational game app creator chooses the STEM concepts which the app should teach and designs a learning activity to teach/reinforce these concepts. A game app idea can be based on this learning activity.

   To determine the educational effectiveness of the game app, app creator also needs to think about the learning assessment data that should be collected from the app.

2. **Implementation**: Once game app design is ready, actual development of the app can start.

   When developing such an app, creator may utilize the simple data collection JAVA API (discussed later in this section) to collect data from the app and store the collected data in a remote database.

3. **App Distribution, Data Collection & Analysis**: Once the app is ready for distribution and distributed to target players, players will play the app and the app will upload the game app learning assessment data in to the remote database. When substantial learning
assessment data is available for analysis, it can be analyzed to determine effectiveness of
the educational game app.

Below is an illustration of the framework showing relationship between various framework
components.

Figure 1: Systematic Framework for Learning Assessment Data Collection.
2.1.2 Framework Components

Each of the framework components (also marked above in the figure 1.3) are discussed in detail below.

2.1.2.1 Educational Game Application

A proposed model to teach related STEM concepts via educational game apps is to teach one concept per app such that higher level app teaches a higher level concept which builds upon more basic concepts taught by the basic game apps. So completing basic level game apps becomes a prerequisite to learn a higher-level concept via a higher-level game app. To ensure that such a learning model is feasible it is important that the educational game apps have a clearly defined educational goal or concept that app aims to teach. Without a clear educational goal it will be difficult to determine prerequisites for any higher-level educational game app.

1. Goal oriented educational Game Apps: While the model of multiple related educational game apps puts this thesis work in perspective, the focus of the thesis is solely on learning assessment data collection mechanism for a single goal oriented educational game app.

2. Game App Development: App creator designs the game app, implements the app and tests the app to ensure it is ready for distribution to the players. Implementation phase of the app also involves incorporating assessment data collection mechanism in the app. Details of how this mechanism can be incorporated in the app are discussed later in the following sections.

3. Game App Distribution: To facilitate rapid learning assessment data collection a good app distribution strategy is crucial to reach all potential players and to obtain large sets of
learning assessment data for these players. The report published by National Academy Press [15] identified:

two possible models for reaching scale in the use of simulations and games for science learning in formal education: (1) a traditional top-down market model, in which games or simulations are sold or distributed to universities, schools, and school districts, and (2) a market model in which widespread use of simulations and games for informal science learning by parents, students, and individuals could dramatically change how science is learned and taught in schools and colleges. (p. 126)

Alternatively, to motivate more users to play the game app and hence contribute valuable learning assessment data, the educational game app can be offered as a free download along with some incentives for playing the app.

4. Learning Assessment Data Collection: The player installs the educational game app, play the app and generate learning assessment data. The app locally collects and prepares this data to be sent to the remote database. The app checks for the connection to remote database and when the connection is found, the app uploads the learning assessment data for the player to the remote database for storage. Below is the step-by-step illustration of the learning assessment data collection from the game app.
Figure 2: Learning Assessment Data Collection.

2.1.2.2 Learning Assessment Data
The best way to measure educational effectiveness of a game app is to measure the learning that is occurring via playing the game app. And in order to measure this learning, app creator needs to devise metrics and collect values for it.

1. **Learning Assessment:** While different app creators may have different mechanisms for measuring this learning, this thesis work suggests a simple method called as “Challenge Questions.”

2. **Challenge Question for Learning Assessment:** To assess the learning that occurred via playing the game app, a player is asked to answer a set of challenge questions devised by the app creator. These challenge questions are specifically devised to measure player’s ability to understand/apply the concept(s) that the game app is built to teach.

   Prior to the game play, player is asked to answer these challenge questions and the responses are recorded in the remote database. Then the player is asked to play the game. Upon completing the game the player is asked to answer the same set of challenge questions and the responses post game play are also recorded in the remote database.

   These recorded responses to challenge questions can be used to determine player’s grasp of the concept(s) i.e. higher the number of correct responses to challenge questions higher is the player’s grasp of the concept(s). Further detailed analysis of these responses can be conducted to determine a player’s strong and weak areas based on the category of questions the player responded correctly/incorrectly.

*Alternate Strategy:* An alternate data collection strategy is to embed the questions into the gameplay so that the player must answer them correctly to continue. This strategy has the potential to be more engaging for the player, but requires considerably more development work than a pre and post challenge question strategy. Both strategies can be implemented using the tools developed below.
3. *Measuring game app educational effectiveness*: Analysis of challenge responses when conducted across several players can help determine effectiveness of game app in meeting educational goals set forth for the educational game app by its creator.

2.1.2.3 Data Collection API

To collect data from the game app, a pre-requisite is that the game app generates that data. Once the data is generated and ready for collection, it can be transported to the remote database.

1. *Generating Application Data*: Data to be collected from the educational game app includes user gameplay session identification information, learning assessment data and other supporting information as deemed necessary by the app creator. All the information that is to be collected from the game app needs to be generated by the game app in response to user activity and temporarily stored as data objects. Once all data to be collected from the educational game app is established in the game as data objects with final values, it is ready for transfer over to a remote database for storage. However, transferring data from the game app to an external medium such as a remote database requires network programming.

2. *JAVA API for two-way data transfer, App <=> Remote Database*: Author is of the belief that app creator should not need to know technical details of network programming to implement two way data transfer between the game app data and the remote database. Instead the app creator should be provided with an easy to use solution to perform any related tasks. This will allow the app creator to focus on educational aspect of the data to be transferred instead of the technical implementation details. Hence, author provides a generic flexible and extensible JAVA API for packing and unpacking game data for storage in and retrieval from a remote database. This JAVA API
can be used to collect data from any educational game and given the open source nature of API it can be further modified to add more functionality as necessary. Further details regarding the JAVA API and directions on how to use the API for data transfer are discussed in Chapter 2.2.

2.1.2.4 Remote Database

Another key consideration for collecting app learning assessment data is the design of the remote database that will house this data. Important things to consider here are: what tables will save what app data, what is the primary key for each of these database tables and how to ensure the design of the remote database is flexible enough to allow storage of additional data as and when necessary without having to a redesign the entire database. Details of remote database design for storing learning assessment data are discussed in Chapter 2.2.

2.1.2.5 Interface to the remote database

Merely collecting learning assessment data from educational game apps and storing it in a remote database is not very useful. This is because data retrieval is necessary before it can be analyzed. Hence, it is valuable to have a suitable query interface to this remote database.

1. Facilitating Analysis of Learning Assessment Data – Web Accessible Database: It is very helpful to make the collected learning assessment data accessible to the researchers that are capable of analyzing this data, via a user friendly web based query interface. This relieves the researchers from the burden of knowing the database query language to retrieve the data or reliance on a database query programmer to retrieve the data on their behalf.
Restricted Access to App Data: Ease of data retrieval is important; however, it is also necessary that the access to this data is restricted to authorized users only. This is necessary because the collected data may contain player sensitive information such as player identification number, email address, etc.

This chapter provided an overview of systematic framework for collecting learning assessment data from educational game apps and also discussed some of the framework components in detail. Each of these framework components plays an important role in enabling learning assessment data collection from the game app. Once a game app is distributed to a large player pool these players play the game app. Learning assessment data is generated in the game app, which the data collection API collects, and transfers over to the remote database. The remote database stores this data and makes it available for analysis via an easy to use web accessible query interface. All framework components have to coordinate with each other for successful learning assessment data collection from the game app.

The next chapter is a discussion of below topics.

1. Specification of the data collection API
2. Database Design of the remote database for storing learning assessment data &
3. Lower level details of the actual data transfer between the game app and the remote database based on this data collection API

### 2.2 Java API & The Database

This chapter is divided into two main sections as below.

1. *First Section – Data Collection API:* First section of the chapter is a discussion of JAVA API specification and how this API can be used for collection learning assessment data from an educational game app.
2. **Second Section – Remote Database Design**: Second section of the chapter is a discussion of a basic remote database design for storing learning assessment data that is collected from an educational game app.

### 2.2.1 Data collection API

Data collection Application Programming Interface (API) essentially provides a transparent mechanism for transporting data from the game app to the remote database and vice versa. This data has to transport over a communication network hence, at the sender site, the data has to be encapsulated in a format, such that the meaning of this data is also preserved at the receiving end.

An app creator can use the API to create a data carrier object called an “Element.” An element is composed of two parts, the data and a meaningful tag for the data to be carried in the element. App creator can create as many of these elements as necessary to transport full set of data entities from the app to the remote database and vice versa. Once, these data carrier objects or “elements” are ready for transfer, the API also provides functionality to transport these elements from the source (e.g. game app) to destination (e.g. remote database), on the internet via. Hypertext Transport Protocol (HTTP).

![Figure 3: Transporting data, game app -> remote database.](image-url)
2.2.1.1 Mutually Agreed Data Tags

Concept of an “element” is very useful for meaningful data transfer because the tag in an element can be used to identify meaning/context of the data it contains. An example of data object is a player high score equal to 35. The number 35 when transported from a sender to a receiver provides no meaningful information at the receiver end. However, transporting an element with data = 35 and a tag = high score provides meaningful information at the receiver end. However, this approach calls for a list of mutually agreed data tags such that the sender and receiver recognize and use these tags identically with absolutely no room for ambiguity. This is especially critical at the remote database server because the data tag is essentially used to determine the database field in which the data for the particular tag is stored. Please refer to APPENDIX B. for elementary list of data tags.

2.2.1.2 Functional Specification

The data collection API is written in JAVA, primarily to collect learning assessment data from an Android game app. The API has been tested with Android Platform 4.0, API Level 14. With minor modifications suggested in the API “read me file” – APPENDIX C., the API can be used to collect learning assessment data from any JAVA game written for a different platform. Following is a more detailed discussion of functionalities provided by the data collection API.

2.2.1.2.1 Flexible Data Collection

App creator cannot anticipate all the information that should be collected from a game app in order to analyze educational effectiveness of the game app. For example, in the initial setup, an app creator might only collect basic player information such as name & email, some game play information like high score and some minimal player feedback on the game app. But as app
creator analyses the collected data and (or) add more features in the game app, the app creator is likely to want to collect more app data to further explore and learn more about the player’s game based learning experiences. For example, app creator may want the players to answer challenge questions and receive more detailed app usage feedback from the players. Hence, one of the important characteristics of the data collection API is to support “flexible” and “unrestricted” collection of data in various formats with varied information structure (or) relationships. This is necessary to have the capability to collect any kind of information from the game app that can be generated in a game app.

The API achieves such a flexible data collection mechanism via Elements. For each data object to be collected from the game app, app creator simply creates an Element by defining a string tag and supplying string data for the tag. This string data may contain special characters, long paragraphs, single word responses, floating point numbers, and (or) a combination of several data formats. Current implementation of API supports single level tagging only i.e. one piece of data can have a single data tag. However, to transport more complicated information author proposes the concept of multiple level tagging, discussed in Chapter 4.

For implementation details see definition of Element class in API Library, APPENDIX C.

2.2.1.2.2 Data Extraction from Game App

The API relies on the idea of collecting all the app data to be collected in a single local file on the player’s device. Having all the data in a single file simplifies the creation of data transport to a large extent.

Whenever a final value of a data object is available in the application code, an Element can be created for this data object and written to the local file on player’s device. When all of the app
data to be collected is written to the local file as Elements, the file is ready to be used for creating data transport request.

In order to assist app creator with performing above mentioned these file operations the API provides the functionality to create a temporary storage file on player’s game device and simple methods to read and write elements to this file. Also, to simplify creation of data transport request, API provides functionality to export all of the file data as an ArrayList. This is useful because the data transfer protocol used by the API utilizes an array list as an input variable.

For implementation details see definition of FileOperator class in API Library, APPENDIX C.

1. *Local Data Storage:* Below is an example of local file on player’s device. This file contains “email” and “challenge Question 1 response” data collected from the game app as two separate elements, with corresponding data tags “email” and “challengeQuestion1”.
2.2.1.2.3 Data Transfer to Remote Database – Data Send

Once the data collection is complete this data can be transferred to the remote database on suitable trigger, for example, when a player clicks on “Submit” button.

API provides functionality to transport data from the game app to the remote database using Hypertext Transfer Protocol (HTTP) network protocol. The array list created from the local file data is used to create a data POST request to send the data to the remote server. The API handles all of the details of the underlying network operations.
2.2.1.2.4 Data Transfer from Remote Database - Data Receive

A “cookie” can be used as a container to receive important information only available at the remote server but desirable in the game app. For example, link to download next level game app. These cookies can be received from the remote server as part of the response of the data POST request.

Figure 5: Posting game app data to remote database server.

Figure 6: Receiving data from remote database server.
1. **Mutually Agreed Cookie Names**: API provides functionality to read content of these cookies via the name of the cookie. Hence, this approach calls for a list of mutually agreed cookie names such that the sender and receiver recognize the meaning of these cookie names identically with absolutely no room for ambiguity. The reader can refer to APPENDIX B. for beginner’s list of cookie names. For implementation details see PostAppData class in API Library, APPENDIX C.

### 2.2.1.5 Data Uploads and Network Connection

There will be cases when the players will play the game app while offline and also generate data to be transferred to the remote database. Without a network connection this data transfer will fail and the data will be stored in a local file on the device. Author suggests checking for a network connection multiple times during an application’s lifecycle to maximize the opportunities for uploading the data to the remote database. Once the data is successfully uploaded from the local file on player’s device, the file can be deleted to free up storage space.

The discussed API is simple to use and provides a flexible data collection mechanism for collecting learning assessment data from educational game apps. The open source nature of the API also allows for further extending the API functionality. Hence, the author considers this API as a useful tool for app creators who want to build learning assessment data collection mechanism for an educational game app.

For further details regarding data collection API also see “read me” file included in the API Library, APPENDIX C.
2.2.2 Remote Database Design

In section 2.2.1 various functionalities provided by the data collection API to transport data between the game app and the remote database were discussed. In this section we discuss a basic remote database design to store learning assessment data received at remote server.

2.2.2.1 Objective of the Remote Database

Learning assessment data is collected from the game apps played by physically dispersed players in to a single remote database. The objective of collecting app data in a central unified database is to perform data mining on these large sets of data to assess student learning and (or) to determine educational effectiveness of the game app. For example, an app creator can run queries on the database to analyze a specific player’s performance in the game app over time and (or) run relational queries to analyze comparative performance of various players playing the same game app.

2.2.2.2 Designing the Database – Design Principles

A good database design is very important to run queries efficiently on large data sets. This implies that the database fields contain information that can minimize the processing necessary to generate query results. Author suggests below three kinds of fields to be considered for designing an efficient remote database for collecting learning assessment data:

1. **Core Fields**: Fields to store learning assessment data collected directly from the game app. For example, player device id, email, high-score, response to a challenge question.

2. **Pre-Computed Fields**: These fields store values that are derived from core fields and used computing frequently used query results. Pre-calculating frequently used values and
storing them in remote database eliminates the need for recalculation of these values at each query run and hence, significantly improves the query response time. For example,

(a) Core field: Player’s responses to challenge questions

(b) Frequently used Query: Comparative performance of all players

(c) Pre-computed field: Number of questions correctly answered by the player

Rationale: Calculating number of questions correctly answered by each player once and then storing the value in the remote database will save computational time for each query run that will use this particular information

3. Open Fields: In addition to above two kinds of fields a good database design should be flexible to store additional unforeseen information at a later time as necessary. These fields can be referred to as “Open Fields”

2.2.2.3 Database Tables

While the database design principles can guide the overall database design, the next step is to nail down the actual tables and the fields to store the collected app data. An app creator may want to store all the app data in a single table or choose to split the collected data across various tables. In the proposed database design author relies on two basic database tables for storing all relevant app data:

1. Table 1 – “app_data”: This table stores all data collected from the game app and includes core, pre-computed and open fields. Proposed table design allows storage of all data generated by many players playing multiple apps, multiple times into a single database table.

2. Table 2 – “app_info”: This table stores meta-data type information regarding multiple game apps and includes core and open fields.
3. **Other Tables:** App creators may begin with basic app data collection in place. However, after analyzing the collected data or adding more features to the game app, the app creator would like to collect even more data from the game app. Depending on the information to be collected above two tables may/may not be the best option to store this new information. Hence, app creators should consider using open fields in the above tables when appropriate but also be open to the idea of creating new tables to store the new information being collected from the game app. For example, if the app creator decides to keep track of how often a given player plays a given app, it would be best to create a new database table to store specifically this kind of information i.e. player statistics for a game app.

Refer to the detail list of fields proposed for each of these tables in APPENDIX B. Each of the tables discussed APPENDIX B provides details such as primary key for the table, list of proposed fields, field descriptions, data types and other supporting information.

The above database design is just a suggestion for basic database set up for storing learning assessment data. An app creator will probably need to modify and (or) extend the proposed design.

2.2.2.4 Storing App Data – Server Side Operation (POST variables)

Once data is posted to the remote server, all data on the server side is available in form of POST variables as shown below.

\[
\text{POST} [\text{<<element tag>>}] = \text{<<element data>>}
\]

App creator can write insert queries to store this data in designated database table fields based on the mapping of “element tags” and “table fields.” See APPENDIX B for a cross-reference of element tags and database fields for table “app_data.”
In Chapter 2 (Section I and Section II) author discussed the “packaged solution” for collecting learning assessment data from educational game apps. Chapter 2 – Section I is a high level discussion of the framework and its components and Chapter 2 – Section II specifically discussed details of data collection API and the remote database design. Hence, review of Chapter 2 is extremely helpful for the reader to understand the framework and its components. However, to further help the reader’s understanding of “how to apply” the framework, Chapter 3 is a discussion of “learning assessment data collection mechanism” for a demo app based on this framework.
CHAPTER 3

DEMO APPLICATION

This chapter illustrates how to leverage the proposed framework and related tools for collecting learning assessment data from any game app. The approach used to write the chapter is “learn by example” and hence the chapter is focused on the details of implementing learning assessment data collection for a demo app, VL3.901_01.

Note: This thesis work does not propose any best practices for assessment of student learning that occurred via playing the game app. However, author does provide a flexible framework and supporting tools to collect this learning assessment data from educational game apps. Refer to APPENDIX E for the demo of learning assessment data collection from demo app, VL3.901_01.

3.1 Background of the work

The demo app is a very simple educational android game based on lesson 1 of MatDL MIT Virtual Labs

[11] The lessons of this virtual lab are part of an introductory undergraduate chemistry class [21] at Massachusetts Institute of Technology (MIT). As per the website for MIT Virtual Labs

[11]

This set of lessons and applets is aimed at students in 3.091 Introduction to Solid State Chemistry. As each concept is introduced, an applet, accompanied by a series of exercises involving the applet, will be provided to aid the students' understanding of the concepts.
3.1.1 Virtual Lab 3.901 - Lesson 1

As per MIT Virtual Lab 3.901 [11]

The goal of this activity is to provide insight into the ways modern science views the effects of temperature on chemical reactions, particularly thermally activated processes. Our simulations will use a much simpler system, but one that retains the essentials of a thermally activated process. The system is that of the rectangular box demonstrated in lecture.

3.1.1.1 Simulation

The focus of this simulation is a rectangular box, similar to the box used by the course instructor, Dr. Donald Sadoway, in his classroom lecture. He uses the rectangular box to illustrate the concept of “states” of a thermally activated process.

This simulation consists of two kick buttons and corresponding slider bars to adjust kick magnitudes. One of these kick buttons is used to kick the rectangular box from stable state to metastable state and the other kick button is used to kick the box from metastable state to a stable state. It can also be observed in the simulation that the kick magnitude for kicking the box from a stable state to a metastable state is always more than kicking the box from a meta-stable to a stable state.

Below is an illustration of Lesson 1 simulation from 3.901 Virtual Lab [11]. Also, the live JAVA applet for lesson 1 can be viewed on any java enabled web browser at http://matdl.org/jars/3.091/lesson01/index.html
Figure 7: MatDL Virtual Lab, Lesson 1 Simulation.

3.1.2 Further Information

1. [11] for further information regarding Virtual Lab 3.901. This also includes a link to live java applet for lesson 1.

2. *Introduction to Solid State Chemistry:* More information specifically about the chemistry class can be found on [20] at [21].

3. *Project MatDL:* For more information regarding MatDL please refer to Chapter 1, Section 1.5.

4. *Framework Demo:* Working demo of learning assessment data collection from demo app, VL3.901_01 is included in the APPENDIX E.
3.2 Walkthrough of Demo Game Application

Demo game app based on virtual lab, 3.901, lesson 1 is an android game app developed and tested for Android 4.0, API Level 14. The game app is very simple and is discussed screen by screen below:

1. **Challenge Question Form:** Player is presented with a set of Challenge Questions, which the player needs to answer before playing the game. Responses to these challenge questions are recorded in the remote database.

![Figure 8: Screen 1 - Challenge Question Form.](image)

2. **Feedback Page and Options:** Player is then provided feedback on whether the player answered the questions correctly or not. If the player answered the question correctly, the player is presented with a choice to download the next level educational game app and to try kicking the rectangular box.
3. **Simple Game Challenge to kick the box**: If the player chooses to kick the box, player is presented with a simulation same as virtual lab 3.901, lesson 1 java applet. The simple game challenge for the player is to kick the box once from meta-stable state to stable state and vice versa using the magnitude sliders bars and kick button.
4. **Final Feedback & Summary Page:** If the player successfully kicks the box in both directions, player is provided feedback along with all possible options on a summary page i.e. answers challenge questions again, kick the box again, close the app or download the next level game app.
3.3 Systematic Application of the framework

This section is a discussion of learning assessment data collection mechanism in the demo game app.

1. **Learning Objective:** Previous section introduced the demo application VL3.901_01. The learning objective of this game app is same as that of the java applet i.e. to illustrate the concept of “states” of a thermally activated process.

2. **Learning Assessment:** In the context of game app VL3.901_01 player’s learning assessment will be done via evaluation of player responses for the challenge questions.

3. **Learning Assessment Data Collected:** The learning assessment data and supporting data collected from the demo game app is same as the data listed in beginner’s list of mutually agreed data tags – APPENDIX B. This includes data such as player’s email/device id, responses to challenge questions, etc. For more details regarding mutually agreed data tags, see Chapter 2 – Section I and Chapter 2 – Section II.

3.3.1 Game Application Development

Existing java applet code was adapted to build the logic to kick the rectangular box. However, all user interface programming was redone for android platform.

Android apps are developed using “activities.” Information regarding android activity lifecycle and concepts of android user interface programming can be found in Android Tutorial, APPENDIX A. Also, entire code for android demo app is also provided in APPENDIX E.
### 3.3.2 Learning Assessment Data Collection

Once educational game app is developed and tested for generating the data to be collected, data collection API can be used to collect and transport the data.

#### 3.3.2.1 Collect Data in a local file

Once a player answers the challenge questions and clicks on the form submit button, all the data to be collected is pulled from the game app into a local file on the player’s android device. All of this data is written to the local file as Elements. See example below for collecting player’s response to challenge questions.

```java
Element e = new Element("challengeQResponses", id_str1 + "," + id_str2);
```

Player’s responses to challenge questions are concatenated in a comma-separated string where `id_str1` is value for player’s response to ChallengeQuestion1 and `id_str2` is value for player’s response to ChallengeQuestion2.

#### 3.3.2.2 Post Data to the Remote Database Server

Once all the data is pulled into a single local file, this output file is used to generate an “array list” of the form of name-value pair. This list is then sent along with the POST data request to the remote server.
3.3.2.3 Updating Game App User Interface

Once the execution of data POST request is complete, game application user interface is updated as necessary. In case of game app VL3.901_01, player is provided with feedback, a link option to download next level game app (only if challenge questions were answered correctly) and a button to try kicking the rectangular box.

3.3.2.4 Storing app data in remote database

The “array list” sent to the remote database in POST data query is received by the remote server as “post variables” in the form as shown below.

`POST [ '[<<ELEMENT TAG>>'] = <<ELEMENT VALUE>>`

Based on the pre-determined mapping of an “element tag” to a “database table/field,” the collected data can be inserted in the database table via an INSERT SQL query. Refer to APPENDIX B for a reference of mapping of all “element tags” and corresponding “database table/fields”. Also, see below an example of an insert query to input game app data in the database table “app_data”.

```
mysql_query("INSERT INTO app_data (appCode, email, playCount , challengeResponses) VALUES($_POST[<<element tag for appCode>>]},
$_POST[<<element tag for email>>],$_POST[<<element tag for");
```
playCount>>, $_POST[<<element tag for challengeResponses>>]) or
die(mysql_error());

3.3.3 JAVA API Cheat Sheet

Refer to APPENDIX D. for a cheat sheet of data collection API syntax. This cheat sheet provides
a quick reference to the syntax for various API functionalities with example.

3.4 Game App Distribution

Demo game app, VL3.901_01 can be distributed via below distribution methods:

3.4.1 Android Market

Android market is the official platform for searching and downloading Android apps. However,
no pre-approval is necessary to put an app on the Android Market. Each app on the Android
Market has an app page featuring app details such as an App Image, App description, reviews and
permissions. A link to Android market is available on all Android devices home screen. After
game app is successfully uploaded to android market, player can search for the app on Android
market via name of the app, install it and play it.
3.4.2 Authorized Web Source

Game app can also be made available for download as .apk file (standard file format for android applications) via an authorized web service such as MatDL. A player can download .apk file from the website on to the android device and hit install button to install the app.

Figure 12: Steps to download android app from a web source.
3.5 Sample Query Interface to Learning Assessment Database

To analyze learning assessment data it is necessary to have a suitable query interface to the database. Below is the sample web interface to query collected learning assessment data stored in the remote database. Live query interface can be accessed at

http://devdesktop.cmi.kent.edu/choose_query.html

Figure 13: Web based query interface to learning assessment database.

Above queries in Figure 1.11, are a representative of the kinds of learning assessment queries that can be performed on the data collected from the game app. Each of these queries is briefly discussed below along with the inferences that can be drawn from the query results.

1. **Number of Attempts**: Query 1, Figure 1.11, can be used to determine if the level of difficulty of app is appropriate. If not then app can be modified to attain a suitable level of difficulty and hence making the app a more effective educational game app.

2. **Correctly Answering a Specific Question**: Challenge question specific queries such as query 2, Figure 1.11, are useful to understand how students are responding to a particular problem. Are they having difficulty in grasping a particular concept? If yes then either the app might not be very effective to teach that concept or needs to be improved to teach it.
3. *Comparative Performance:* Queries of the nature of query 3, Figure 1.11, helps in determining if a player understands a particular notion about a concept, is it easier for that player to grasp all notions related to that particular concept? If yes then why and if not then why not.

4. *Top Performers:* Queries of the nature of query 4, Figure 1.11 helps to determine top performers. This data may be compared against test results of an independent test to confirm that the students reported as top performers actually have a good grasp of the concepts being taught by the game app.

5. *Open Data Selection:* Finally, an open query can be run to view all data stored in the remote database or with restriction such as app specific data, player specific data, etc. Similar to the above queries several other queries can be written on the database to assess learning in different ways.

This chapter discussed the demo application and its learning assessment data collection mechanism based on the proposed framework. In the next chapter, author suggests further improvements/extensions to the work discussed in this Thesis.
CHAPTER 4

FUTURE WORK

Work completed as part of this thesis enables learning assessment data collection from any JAVA based educational game and this is only a step forward in the right direction. Further, a lot of work needs to be done to promote game based learning. Following is a discussion of future enhancements / project ideas that build upon the work accomplished in this thesis.

4.1 Building Evidence for Effective Learning via Educational Games

Building conclusive evidence regarding effectiveness of educational games is one of the most important research problems in educational gaming. This thesis provides a means to collect learning assessment data; analysis of this data can help researchers to draw inferences about effectiveness of learning via educational games.

The intent of this work is to promote learning assessment data collection by providing an out of the box solution and promote analysis of this learning assessment data. The data analysis if geared towards (a) identification of existing gaps in measuring effectiveness of educational games and (b) understanding how these gaps can be overcome, will help the researchers to make recommendations for further research in building conclusive evidence regarding educational game effectiveness.
4.2 Multi-level Data Tagging

Current implementation of API allows for single level data tagging i.e. one tag per data object. However, in the future the API can be extended to allow for multiple levels of tagging such that there can be any number of tag levels i.e. primary, secondary, tertiary and so on. Support for multi-level data tagging will provide flexibility in collection of related data items without loss of information regarding relationships/hierarchies among data objects.

Similarly, given the open source nature of the API it can be further enhanced to add other functionalities such as, extending the API to convert collected learning assessment data into various other standard formats such as JSON, XML, etc.

4.3 Rewriting data collection API in Objective C

Current implementation of the data collection API is JAVA based and it can be used with any educational game with JAVA interface for collecting learning assessment data. However, if the API is rewritten in Objective C, it will be ready to provide all API functionalities for any iPhone/iPad based educational game. Considering the number of iPhone/iPad users this project idea is worth consideration.

4.4 Multi-level Taxonomy for organizing storage of educational game apps

In this Thesis author’s focus is on learning assessment data collection from a single educational game app. However, as multiple apps are created to teach related concepts within an umbrella topic, there will be a need for better organization for storage and retrieval of these apps and the
data collected from them. One way to organize storage of these apps/app data is to use a meaningful identification code for each app such as a 12 digit code XXX-YYY-ZZZ-123 where:

1. **XXX**: These three alphabets represents the overall broad category of the app subject such as English/Science/Maths/..

2. **YYY**: These three alphabets represents the specific concentration within the broad subject such as Materials Science under Science

3. **ZZZ**: These three alphabets represents the actual topic of the app such as Solid State Chemistry is a topic within Materials Science

4. **123**: These three numbers represents a sequential app identification code

This taxonomy of organizing educational game apps (and their data) storage is further illustrated in the figure below:

![Taxonomy Diagram](image-url)

**Figure 14**: Taxonomy for organizing storage of educational game apps.
Based on the above figure code 003-999-003-001 uniquely identifies a “Science” game app, or more specifically a “Material Science” game app. The topic of this game app is “Solid State Chemistry” and the app is called VL3.901_01.

4.5 Review of Devlin’s Educational Gaming Environment Design Principles

This thesis work found its motivation in Devlin’s book [14] game environment design principles, specifically design principle #6 which states that, “there should be some uniform means of assessing student’s performance.” This Thesis work provided framework and related tools for collecting learning assessment data from educational game apps. Further, work can be done to fully realize Devlin’s principle #6 such as (a) best practices for student learning assessment in an educational game (b) best practices and guidelines for ensuring uniform learning assessment, etc.

Review of other Devlin’s game environment design principles can also help to identify and formalize other important educational game research problems.

Even though educational games are promising, until the evidence for effective learning via educational games is established we cannot expect to see educational games accepted as part of mainstream education technology or justify huge investments be made in to educational gaming research. But if we work towards finding evidence for effective learning via educational games, may be educational games will emerge as the next big step in improving our education technology.
REFERENCES


[18] Installing the Android SDK. http://developer.android.com/sdk/installing.html#Installing


APPENDIX A.

ANDROID TUTORIAL
Android Tutorial

This short tutorial is helpful to familiarize reader with Android Platform, setting up of development system for Android application development, application development in Android and some of the useful Android resources.

I. Why Android?

In order to provide a sample implementation of the framework, for collecting educational game data, several mobile platforms were considered. Apple iphone, Google Android & Windows Mobile platforms were primary candidates for model implementation and of these three options Android emerged as the preferred choice because of the merits listed below:

Android is an open source mobile platform and MatDL promotes use of open source tools. This helps cut costs and provides flexibility to adapt the tool as per requirements of MatDL user community.

Android Development is JAVA based and the MatDL Virtual Lab 3.901, lesson 1 is also implemented as Java Applet along with other MatDL Virtual Labs. This presented opportunity for code reuse when writing Android version of the game app based on MatDL Virtual Lab.

Android development can be done on Windows, Mac OS X or Linux whereas iPhone development can only take place on MAC OS X.

Low priced android enabled smart devices are an affordable option for the student community vs. high priced Apple smart devices such as iPhone and iPads.
II. Getting started with Android Development

As per (Android Developers) portal ("Installing the SDK")

“Android offers a custom plugin for the Eclipse IDE, called Android Development Tools (ADT), that is designed to give you a powerful, integrated environment in which to build Android applications. It extends the capabilities of Eclipse to let you quickly set up new Android projects, create an application UI, debug your applications using the Android SDK tools, and even export signed (or unsigned) APKs in order to distribute your application. In general, developing in Eclipse with ADT is a highly recommended approach and is the fastest way to get started with Android.”

Following are the steps to setup system for Android Development on Windows Platform (in order) within Eclipse Integrated Development Environment (IDE):

1. Download and install latest version of JDK
3. Download & install Eclipse
   Android Developers portal recommends "Eclipse Classic" version. Download the .zip file for the preferred version and unzip in desired location to complete Eclipse installation
5. Download & install Android SDK/recommended packages
   Android Developers portal recommends installer_r16-windows.exe for Windows Platform

III. Setting up Development Environment

Once SDK installation is complete follow the steps below.

Installing Packages: Double-click the SDK Manager.exe file in the root of the Android SDK directory to launch the Android SDK and AVD Manager. Android SDK and AVD Manager will open with all available components selected for installation. Either install all components checked (this may take a long time depending on network speed) or choose to install only required components by going to Available Packages > Android Repository and selecting components as below.

1. Android SDK Tools
2. Android SDK Platform-tools
3. SDK Platform Android, API

Select the most recent revision and API level as applicable during the time of installation

Samples for SDK API
Creating an Android Device Emulator: Android device emulator can be used to perform app testing without buying an actual android device. Also, emulator specifications can be edited to test app on various screen size, sd card size, resolution and other parameters.

Double-click the SDK Manager.exe file in the root of the Android SDK directory which will launch the Android SDK and AVD Manager. Then go to virtual devices and set up my_avd emulator as below. This will create an emulator named my_avd with target android platform, sd card size 1000 MiB and the skin as selected from dropdown menu.
Figure 2: Edit Android Virtual Device.

Downloading ADT plugin for Eclipse: Follow below steps as listed on Android Developers portal ("Downloading the ADT Plugin"). This will complete set up of development system for android development and testing on emulator using Eclipse IDE.

1. Start Eclipse, then select Help > Install New Software....
2. Click Add, in the top-right corner.
In the Add Repository dialog that appears, enter "ADT Plugin" for the Name and the following URL for the Location:
https://dl-ssl.google.com/android/eclipse/
Click OK
Note: If you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https" (https is preferred for security reasons).

3. In the Available Software dialog, select the checkbox next to Developer Tools and click Next.
4. In the next window, you'll see a list of the tools to be downloaded. Click Next.
5. Read and accept the license agreements, then click Finish.
If you get a security warning saying that the authenticity or validity of the software can't be established, click OK.
When the installation completes, restart Eclipse.

IV. Run an existing Android project

Open Eclipse and go to File > Import > General > Existing Projects into Workspace > Next >
Select root directory and select existing eclipse android project root folder. When done select
Copy Projects into workspace and hit Finish

Notes:

If experiencing any issues with the steps above, please verify the development machine meets system requirements for all new installations

For detailed notes please refer to (Android Developers) portal (“Installing the SDK”)

To uninstall eclipse/android simply delete the corresponding root folder and start over
V. Android Application Development

Android “Activity” Model

Android application development model involves creation of activities which a user sees/interacts with at any point of time. Each activity has a clear specific goal and a seamless user interface is achieved by switching between different activities that constitute a given android app. An Android activity has essentially four states, namely active/running, paused, stopped and killed/finished. This activity life-cycle is further illustrated in the below figure from (Android Developers) portal.
The first activity to start when the application is run can be specified in AndroidManifest.xml file and onCreate() method of this android activity is the entry point of the application. Activity lifecycle initiates from onCreate() method and proceeds as per the flow in Figure 1.11 based on the application code specified in the activity lifecycle state methods i.e. onCreate(), onStop(), onStart(), onPause(), onResume() and onDestroy().

User Interface Programming in Android

Android does not support commonly used User Interface programming Java packages such as JAVA Swing. In order to build user interface Android provides User Interface (UI) drawing classes that are optimized for the Android platform. Android View is a class which occupies a rectangular area on the screen and can be used to draw UI components on the screen. Also, View interface SurfaceHolder.Callback can be used to track changes to the view. Also, all UI resources of the android application such as image files, text strings, layouts, etc can be created and stored within “res” folder of the application.
**Game Development in Android**

2D-3D games. Library is pretty advanced now to support Android games. Android sample LunarLander is a good one to begin with. Game mechanics etc are same start with any good reference book on Game Development. Game Mechanics, Game Physics and Mathematics. 


**VI. App Distribution**

Android apps are distributed as .apk files which can be installed on Android devices. To prepare the app for distribution Android offers tools such as keytool(for generating key), jarsigner(for signing application) and zipalign (tool for automated optimization of android application). If using Eclipse with ADT plugin one can follow the steps below as listed on (Android Developers) portal to make the app distribution ready:

Select the project in the Package Explorer and select File > Export. 

Open the Android folder, select Export Android Application, and click Next.
The Export Android Application wizard now starts, which will guide you through the process of signing your application, including steps for selecting the private key with which to sign the .apk (or creating a new keystore and private key).

Complete the Export Wizard and your application will be compiled, signed, aligned, and ready for distribution.

Note: Minimum required Android Version for running the app can be specified in AndroidManifest.xml file. Android will check this version when installing the app for compatibility

**Android Market**

Android market is the official platform for searching and downloading Android apps. Each app available on Android Market had an app page App Page featuring the app providing details such as App Image, App description, reviews and permissions. Also, Android market is available from Android phone home screen. A player can download the app from Android app and install the app on their phone and play the app.
VII. Other helpful android notes

Below is a discussion of some Android topics that are useful when working with the platform. This section includes discussion of Android utilities and online resources.

Android Developer Portal

Android Developers portal is an indispensable resource for android developers. Portal is organized in sections namely SDK, Dev Guide, Reference, Resources, Videos and Blog. Dev Guide is an excellent starting point to build strong conceptual understanding of android platform.

As per (“The Developer’s Guide”) on the portal:
The Dev Guide provides a practical introduction to developing applications for Android and documentation about major platform features. It explores the concepts behind Android, the framework for constructing an application, and the tools for developing, testing, and publishing software for the platform.

**JAVA based Android Programming**

Android programming is JAVA based. Android SDK offers several android specific JAVA packages and additionally supports other open java packages such as java.io, java.math, java.net, java.util, org.apache.http. For a comprehensive list of packages available for android development refer to (“Package Index”) on (Android Developers) portal on Reference tab.

**Android Samples**

Sample applications code can be found in Resource section on (Android Developers) portal. In section 3.2 Getting started with Android Development installation of Samples for SDK API was discussed. If samples for SDK platform are installed they can be run via Eclipse IDE by going to File> New> Android Project> Create Project from existing sample(select a sample from samples dropdown)> Finish
Figure 5: Create project from existing sample.

Reading, running and experimenting with sample code helps to gain a quick insight on generic android app development model and various android features/capabilities.

**Android Logcat**

Android offers a great logging utility called logcat which can be used for logging messages from the android app for debugging purposes. Log messages can be of the type informational, error, warning, etc. Also, developer can use “log tag” to label the log message and describe its content.
Logged messages can be viewed via command line or in Eclipse IDE DDMS perspective by going to Window> Show View> LogCat.

![Android Logcat Utility](image)

Figure 6: Android Logcat Utility.

In addition to developer specified log messages android also provides default system messages in the log so sometimes it can be difficult to separate out application specific log messages in the big log displayed. However, these log messages can be filtered based on log type, tag, etc and can be conveniently viewed in a separate tab by defining a new filter in Logcat.
APPENDIX B.

BEGINNER’S LIST OF DATA FIELDS
<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Description</th>
<th>Data Tag</th>
<th>Data Type</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>App Identification Code</td>
<td>“appCode”</td>
<td>String</td>
<td>This field uniquely identifies a game app. A sophisticated scheme to generate unique identification code for a game app within context of several game apps for various subjects and topics is discussed in Chapter 4, Section X.Y.Z</td>
</tr>
</tbody>
</table>
Each player will have a unique email address and hence this field can be used to uniquely identify the Player. Also, storing player's email address will enable the app creator to get in touch with the player, given that the necessary permissions to contact the player were previously obtained.

However, as the database grows in terms of records and several related tables it is more suitable to have a numeric player identification key instead of using an email.
address for player identification. While string primary key can be a drag to the performance of the system, a numeric key will aid in more efficient sorting and retrieval of records.

<table>
<thead>
<tr>
<th></th>
<th>Player’s play count for the app</th>
<th>&quot;playCount&quot;</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>A player will play the game app number of times until he/her successfully completes the app and the player can play the game app any number of times even after successfully completing the app. In both these cases it is valuable to collect app data because data collected for all cases</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
help answer important questions such as:

(i) How difficult it is for the player to complete the game app? More trials may suggest more difficult
(ii) Was the player able to answer the challenge questions all correct once the player successfully completed the app or the player faced additionally challenges in answering the questions? (iii) Is the player interested in playing the game app even after successfully completing the app?

<table>
<thead>
<tr>
<th></th>
<th>Total number of</th>
<th>&quot;totalNoOfQuestions&quot;</th>
<th>String</th>
<th>Total number of challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Challenge Questions</td>
<td>5 Responses to app challenge questions</td>
<td>6 String to denote if the responses to challenge questions was correct or not</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;challengeResponses&quot;</td>
<td>&quot;challengeResponsesEvaluation&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>String</td>
<td>String</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| questions asked by the app, See Chapter 2.1, Section X.Y.Z for details |

- Challenge question responses can be evaluated in the game app and a string of the form 0/1/TBD,#NA can be sent to represent if the challenge responses were:
  
  (i) correct = 0  
  (ii) Incorrect = 1  
  (iii) Response needs to be evaluated by a human e.g. free long text response = TBD  
  (iv) Not answered by the player:
<table>
<thead>
<tr>
<th></th>
<th>Total correct responses to challenge questions</th>
<th>&quot;totalCorrectResponses&quot;</th>
<th>String</th>
<th>#NA</th>
</tr>
</thead>
</table>

This field does not provide any new information but can aid in efficient calculation of certain query results which need this information.
<table>
<thead>
<tr>
<th>8</th>
<th>To store any feedback collected from the app</th>
<th>&quot;feedback&quot;</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Various kinds of feedback can be collected from the game app players to better understand how players feel about the app for example ratings can be collected on a scale of 1 -10 for below:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(i) How helpful was the app to help player understand the concept app aims to teach</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(ii) How interesting does the player find the game app to be</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(iii) Overall rating</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Also, players can be provided with an open comment field</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
to provide additional feedback they may have as players and users of the educational game app.

<table>
<thead>
<tr>
<th>9</th>
<th>Player high score for the app</th>
<th>&quot;highScore&quot;</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player high score can be used like a benchmark of their peak performance. This is a direct indicator of how players compare with each other and also to see if a player improves their performance in leaps and bounds or steadily with repetitive playing or in some cases players may not show any improvement</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
at all even with repetitive playing which may require a deeper investigation of the cause and possible solutions.

<table>
<thead>
<tr>
<th></th>
<th>Open Field 1</th>
<th>TBD</th>
<th>TBD</th>
<th>TBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Open Field 2</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>11</td>
<td>Open Field 3</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>12</td>
<td>Open Field 4</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>13</td>
<td>Open Field 5</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>Sr. No.</td>
<td>Description</td>
<td>Data Tag</td>
<td>Data Type</td>
<td>More Information</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------</td>
<td>----------------</td>
<td>-----------</td>
<td>------------------</td>
</tr>
<tr>
<td>1</td>
<td>App Identification Code</td>
<td>“appCode”</td>
<td>String</td>
<td>This field uniquely identifies a game app. A sophisticated scheme to generate unique identification code for a game app within context of several game apps for various subjects and topics is discussed in Chapter 4, Section X.Y.Z</td>
</tr>
<tr>
<td>2</td>
<td>Download URL for the app</td>
<td>&quot;downloadURL&quot;</td>
<td>String</td>
<td>Game app install file Download URL</td>
</tr>
<tr>
<td>3</td>
<td>Next App Identification Code</td>
<td>&quot;nextAppCode&quot;</td>
<td>String</td>
<td>App code for next higher level app, See Game app model, Chapter 2.1, Section X.Y.Z for details</td>
</tr>
<tr>
<td>4</td>
<td>Open Field 1</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>5</td>
<td>Open Field 2</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>6</td>
<td>Open Field 3</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>7</td>
<td>Open Field 4</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
<tr>
<td>8</td>
<td>Open Field 5</td>
<td>TBD</td>
<td>TBD</td>
<td>TBD</td>
</tr>
</tbody>
</table>
APPENDIX C.

DATA COLLECTION API
Game App data collection API

This API is a part of the Game App data collection framework as proposed by Master’s Thesis titled “Tools for measuring educational effectiveness of a Game App.” The data collection API is open source and can be used to collect game data from any Android Game App. The API is written and tested with Android Platform 3.2, API Level 13.

This API contains below classes and their respective definitions can be found in the accompanying API library.
• Element: Provides meaningful data container for transporting game app data
• FileOperator: Provides functionality to collect and prepare game app data before transporting it over to remote database
• PostAppData: Handles all necessary underlying network operations

Each of the above class file contains a brief overview of the class in the comments section at the top and additional explanatory comments are provided throughout the code. Each of these API classes logs all important activity/checkpoint in standard Android informational log under a log Tag called "MyApp."

Various values are hardcoded in the class files and can be edited as required. Example of some of these values hardcoded in the code includes log tag, element separator (separator between element data element tag), Line separator in the data file (file for temporary app data storage) and end of data file string.

Even though the API classes are written and tested specifically for Android Platform, API class files can be slightly modified and used for data collection from any JAVA based game. Some of the changes required in the API to collect game app data from a game app (if not Android based) include: (i) Obtaining file Input / Output stream in a manner as applicable to the platform, see class definition for FileOperator (ii) It may not be necessary to create an asynchronous task to carry out HTTP Post request as a background operation, see class definition for PostAppData for more details.
Below is the definition of Element class. This class is used to tag app data for collection and storage in remote database. Each element object consists of two parts: a tag and the data associated with the tag.

1. Tag: Meaningful label to describe the associated app data. Tag is a string type. Content of element tag is important as it is used to store the collected data in appropriate remote database fields by scripts on the database server.
2. Data: App data associated with the respective tag. Data is a string type.
3. Separator: Tag and Element data separator. Separator is string type.

```java
package hello.formstuff;

import android.util.Log;

public class Element {
    // App Data Tag
    private String tag;

    // App Data
    private String data;

    // Tag and Data Separator
    public String separator = ":";

    // App String Data
    private String data_string;

    // Android Log Tag
    private String log_tag = "Myapp";

    // Constructor
    /** To collect data, app creator can use Element constructor to specify both the element tag and element tag data as two separate String arguments to the constructor **/
    Element(String tag1, String data1) {
        tag = tag1;
    }
}```
data = data1;
Log.i(log_tag, "Element " + tag + ":" + data + " sucessfully created");

//Method to get app data with tag in string format
/** The method dataString returns the contents of an element object i.e. tag + seperator +
data in a concatenated string format. This string format of the element object can be
written to a file for complete element storage **/
public String dataString() {
    data_string = tag + separator + data;
    return data_string;
}

/**
Below is the definition of File Operator class. This class is used to encapsulate file
operations required to collect all app data as elements in one file before transferring it
over to the remote database. This class also contains a method to prep the data in form of
an Array List for the data post query.

Each File Operator object has a
1. File Name - Name of the file to read data from and write data to. File Name is a String
type.
2. Line Seperator - A seperator to distinguish between each new element written to the
file. Line seperator is a String type.
3. End of File Indicator - This indicator is written as the last line of the file when all data
to be collected is collected in the file. End of file indicator is a String type.
4. Data List - File operator class contains a method to prepare the data for upload in form
of an Array List. This Array List contains all app data as elements with name
corresponding to app data tag and vale corresponding to app data. This Array List is sent
to the remote database in post data query as Post Entity and recieved by the remote server

as post variables of the form POST['<<ELEMENT TAG>>'] = <<ELEMENT VALUE>>.

*/

package hello.formstuff;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import android.app.Activity;
import android.content.Context;
import android.util.Log;

public class FileOperator {

    // Name of the file in app directory for temporary app data storage
    private String file_name;

    // Line separator to distinguish between each new element written to the file.
    private String line_separator = "\r\n";

    // End of file indicator to mark file data as complete for transmission to remote database
    private String endof_file = "#!#";

    // Array List which is a collection of all elements to be sent to the remote database in post data query
    private List<NameValuePair> data_list = new ArrayList<NameValuePair>();

    // Android Activity. This is required to get the file output/input stream
    private Activity activity;

    // Read file data as String. This is to obtain file contents as a string
    private String file_string = "";
}
//FileOutputStream
private FileOutputStream output_stream = null;

//FileInputStream
private FileInputStream input_stream = null;

//Android Log Tag
private String log_tag = "Myapp";

//Constructor
/**
 * App creator creates an object of file operator by specifying name of the file on android device for temporary storage of app data & current android activity. The constructor sets the output stream & input stream for concerned file. These streams are then used by other class methods to read/write data from/to the file.
 **/
FileOperator(String file_name1, Activity activity1) {
    file_name = file_name1; //name of file for temporary data storage
    activity = activity1; //Android Activity

    //Get resulting output stream, also handles file creation if file does not exist
    try {
        output_stream = activity.openFileOutput(file_name, Context.MODE_PRIVATE);
        Log.i(log_tag, "Output Stream for file: " + file_name + " successfully created");
    }
    catch (Exception e) {
        e.printStackTrace();
        Log.i(log_tag, "Output Stream for file: " + file_name + " cannot be created");
    }

    //Get resulting input stream
    try {
        input_stream = activity.openFileInput(file_name);
        Log.i(log_tag, "Input Stream for file: " + file_name + " successfully created");
    }
catch (Exception e) {
    e.printStackTrace();
    Log.i(log_tag, "Input Stream for file: " + file_name + " cannot be created");
}

//Write element to file
/**
 * This method takes an element and writes the complete element to the output file. This
 * method returns 1 if the element was successfully written to the output file or a 0 if the
 * element cannot be written to the file
 ***/
public int writeDataToFile(Element element1) {

    //Check if the file is open
    if (output_stream != null)
    {
        try {
            //Write string format of element to file
            output_stream.write(element1.dataString().getBytes());
            //Add line seperator after each element to seperate different elements
            output_stream.write(line_separator.getBytes());
            Log.i(log_tag, "Element " + element1.dataString() + " successfully written to
            output file");
            //Element successfully written to file
            return 1;
        }
        catch (Exception e) {
            e.printStackTrace();
            Log.i(log_tag, "Element " + element1.dataString() + " cannot be written
to output file");
            //Element not written to file
            return 0;
        }
    }
    return 0;
}
// If file is not open
else {
    Log.i(log_tag, "Element " + element1.dataString() + " cannot be written to output file. File not open.");
    // Element not written to file
    return 0;
}

// Mark end of file
/**
 * Once all elements that constitute app data to be transferred to remote database are written to the output file, this method is used to mark the output file as complete. This method returns 1 if the file was successfully marked as complete or a 0 if the file cannot be marked as complete
 **/
public int markEndOfFile() {
    try {
        output_stream.write(endof_file.getBytes());
        output_stream.close();
        Log.i(log_tag, "End of file marked successfully");
        // End of file marked successfully
        return 1;
    }
    catch (Exception e) {
        e.printStackTrace();
        Log.i(log_tag, "Cannot mark end of file");
        // Cannot mark end of file
        return 0;
    }
}
//Read all app data from the output file into a string
/**
Method reads all app data collected in the output file into a string variable and returns the
file content as a string. The method returns an empty string if no output file data was
read.
**/
public void readDataFileString() {

    int ch;

    //Read file one character at a time and concatenate character in a string
    try {
        while( (ch = input_stream.read()) != -1)
        {
            file_string = file_string + (char)ch;
        }
        Log.i(log_tag, "File successfully read as the string below");
        Log.i(log_tag, file_string);
        input_stream.close();
    }
    catch (Exception e)
    {
        Log.i(log_tag, "File cannot be read as a string");
        e.printStackTrace();
    }
}

//Create an Array List from all the elements in the output file
/**
This method parses file content string to return an Array List containing all elements in
the output file as name and value pair. This Array List is sent to the remote database in
post data query as Post Entity and received by the remote server as post variables of the
form POST['<<ELEMENT TAG>>'] = <<ELEMENT VALUE>>. **/
public List<NameValuePair> createArrayList() {

}
int i = 0;
Element elem = new Element("donotsend", "donotsend");
String[] temp = null;
readDataFileString();

//Create a string array of individual elements
String[] separated = file_string.split(line_separator);

//Put all elements in the Array List except end of file character
while (i < separated.length - 1)
{
    //Break element string into constituent tag and data
    temp = separated[i].split(elem.separator);
    //For each element put in Array list tag as name and data as value
    data_list.add(new BasicNameValuePair(temp[0], temp[1]));
    Log.i(log_tag, "Added element to Array List - Tag: " + temp[0] + ", Data: " + temp[1]);
    i = i + 1;
}

return data_list;
}
}

/**
Below is the definition for class PostAppData. From Android version 3 onwards network operations cannot be performed within the main thread of an android application. To perform any time consuming operation such as network operations one of the below methods can be used:
- Multi-Threading
- Handlers
- Async Task
This class extends Class AsyncTask to perform post request in the background. Base class method onPostExecute() is Overriden in PostAppData class and is called automatically once the network operation (or the post request) is complete and the post response is received. This method can then be used to update Android Activity User Interface as necessary. Also, along with post response, cookies are received from the remote server. Class method getCookieValue() can be used to retrieve value of a given cookie name by suppling the method with the cookie name. This method searches for the particular cookie name amongst all the cookies received and returns the concerned cookie value.

In order to create PostAppData object use class constructor and supply Array List of data to be posted and android activity object which can be used to update activity User Interface(UI) as necessary

```java
package hello.formstuff;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.cookie.Cookie;
import org.apache.http.impl.client.BasicCookieStore;
```
import android.os.AsyncTask;
import android.util.Log;
import android.widget.Button;

public class PostAppData extends AsyncTask<String, Void, String> {

    private HelloFormStuffActivity activity1;
    private DefaultHttpClient client = new DefaultHttpClient();
    private CookieStore cookie_store = new BasicCookieStore();
    private List<Cookie> cookies;
    private HttpContext localContext = new BasicHttpContext();
    private List<NameValuePair> data_list1 = new ArrayList<NameValuePair>();

    //Android Log Tag
    private String log_tag = "Myapp";

    //Constructor
    /**
     * App creator can create a post request task by suppling arguments 1. Array list of data to
     * be posted and
     * 2. Android activity object to update activity User Interface as necessary.
     **/
    PostAppData(List<NameValuePair> data_list, HelloFormStuffActivity activity) {
        data_list1 = data_list;
        activity1 = activity;
        Log.i(log_tag, "Post request Task created");
    }

    //Execute Post request
    /**
     * Operation to be performed in the background. In this case perform a POST request
     * using Array List as data to be posted and also receive cookies along with the post
     * response
     **/
    @Override
    protected String doInBackground(String... urls) {

    }
String response = "";
String s = "";

for (String url : urls) {
    localContext.setAttribute(ClientContext.COOKIE_STORE, cookie_store);
    HttpPost httpPost = new HttpPost(url);

    try {
        //Execute post request with appropriate context
        httpPost.setEntity(new UrlEncodedFormEntity(data_list1));
        HttpResponse httpResponse = client.execute(httpPost, localContext);
        Log.i(log_tag, "Post request executed");

        //Retrieve cookies received
        cookies = cookie_store.getCookies();
        Log.i(log_tag, "Cookies retrieved");

        //Get Post Response as string
        InputStream content = httpResponse.getEntity().getContent();
        BufferedReader buffer = new BufferedReader(new InputStreamReader(content));
        while ((s = buffer.readLine()) != null) {
            response += s;
        }
    }
    catch (Exception e) {
        e.printStackTrace();
        Log.i(log_tag, "Post Response cannot be retrieved as a string");
    }
}

return response;
// Update Android Activity User Interface (UI) once post request execution is complete

@Override
/**
 * Overridden Base class method. This method is called automatically when the task is completed in the background.
 **/
protected void onPostExecute(String result) {
    // Update activity UI as necessary
    activity1.uiUpdateOnChallengeFormSubmit();
    Log.i(log_tag, "Below is the response received for post query");
    Log.i(log_tag, result);
}

// Get Cookie Value as a string for a given cookie name
/**
 * This method searches for the cookie with cookie name equal to cookie_name supplied as an argument to the method and returns the cookie value if a match is found else it returns an empty cookie value.
 **/
public String getCookieValue(String cookie_name) {
    int i;
    String cookie_value = "";
    // Cookie found flag
    int flag = 0;

    // Search for the cookie with the specified cookie name
    if (cookies.isEmpty()) {
        Log.i(log_tag, "No cookies received");
    }
    else {
        for (i = 0; i < cookies.size(); i++) {
            if (cookies.get(i).getName().contentEquals(cookie_name)) {
                cookie_value = cookies.get(i).getValue();
                flag = 1;
            }
        }
    }
}
if (flag == 1) {
    Log.i(log_tag, "Cookie: " + cookie_name + " = " + cookie_value);
}
else {
    Log.i(log_tag, "No cookie found with cookie name = " + cookie_name);
}
return cookie_value;
APPENDIX D.

CHEAT SHEET FOR API SYNTAX
1. **File Operator**
Create File Operator Object using constructor `FileOperator(String file_name1, Activity activity1)` and supply it name of the file to store app data and Android activity object. FileOperator uses android activity object to obtain file input and output streams.

Syntax
```
<FileOperator>> = new FileOperator(<name of file>, <android activity object>);
```

2. **Element**
Create an Element via Element constructor `Element(String tag1, String data1)` and supply data tag and data. Tag supplied can be used on remote server for store data in the appropriate field in the database.

Syntax
```
Element <<element>> = new Element(<tag>, <data>);
```

3. **Storing Elements in a local file**
Use FileOperator to store the element in a file on Android Phone using `writeDataToFile()` method. This methods returns an integer = 1 for success and 0 otherwise

Syntax
```
<<FileOperator>>.writeDataToFile(<<element>>);
```

Once all data encoded as Elements are written on the data output file on Android Phone, mark the file as complete using FileOperator method `markEndOfFile()`. This method has return type void and hence does not return any value.

Syntax
```
<<FileOperator>>.markEndOfFile();
```

4. **Creating arraylist with Post variables**
Once the file is marked complete use FileOperator method `createArrayList()` to generate an Array List containing all data to be posted in the form of name and value pairs such that each element to be posted Array name corresponds to element tag and Array value corresponds to element data. Return type for this method is `List<NameValuePair>`

Syntax
```
<<Array List>> = <<FileOperator>>.createArrayList();
```
5. Creating Data Transfer request
Create PostAppData object using constructor PostAppData (List<NameValuePair> data_list, HelloFormStuffActivity activity) and supply it ArrayList of data to post and Android Activity Object. You need to pass the Android activity object to update User Interface(UI) of activity once the post request is complete.

Syntax
$$\text{<<PostAppData>>} = \text{new PostAppData(<<Array List>>, <<Android Activity>>);}$$

6. Executing Data Transfer request
To execute post request use method execute() of the class PostAppData() and supply it with complete post url string

Syntax
$$\text{<<PostAppData>>.execute(new String[] { <<post_url_string>> });}$$

7. Receiving Data from Remote Server
Supply cookie name as string to PostAppData method getCookieValue() to receive cookie value

Syntax
$$\text{<<cookie value>>} = \text{<<PostAppData>>.getCookieValue(<<cookie name>>);}$$

8. Updating User Interface after data transfer
Define a public method in the activity called uiUpdateOnChallengeFormSubmit() with return type as void. Perform all User Interface updates to be executed on Post Request completion. This method is called by PostAppData class upon completion of Post query.

Syntax
```
//Update User Interface(UI)
public void uiUpdateOnChallengeFormSubmit() {
    //UI Update method definition goes here
}
```
APPENDIX E.

WORKING DEMO APPLICATION CODE
package hello.formstuff;

import java.io.File;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Pattern;
import org.apache.http.NameValuePair;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
import android.provider.Settings.Secure;
import android.text.Html;
import android.text.method.LinkMovementMethod;
import android.text.util.Linkify;
import android.util.Log;
import android.util.TypedValue;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;

public class HelloFormStuffActivity extends Activity {

    View vw;
    TextView tv;
    Button btn;
    RadioButton rb;
    RadioGroup rg;

    String feedback = "";
    String download_url = "error";
    Pattern pattern;
    Boolean answered_correctly = false;
}
//Unique Android Device Id
String device_id;

//Return List
String[] gather_data = new String[4];

String file_name = "hello_file";
Intent intent;

//String url_string = "http://app-test.matforge.org/";
//String post_to_url = "http://app-test.matforge.org/app_query.php";
String url_string = "http://devdesktop.cmi.kent.edu/";
String post_to_url = "http://devdesktop.cmi.kent.edu/app_query.php";
int i = -1;

List<NameValuePair> data_list = new ArrayList<NameValuePair>();
HelloFormStuffActivity activity1 = this;
FileOperator file_operator;
PostAppData task;
Boolean data_sent = false;

//Android Log Tag
private String log_tag = "Myapp";

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    attempt_data_load();
    //Get Device ID
device_id = Secure.getString(this.getContentResolver(), Secure.ANDROID_ID);
    //Assign Button Listeners
    btn = (Button) findViewById(R.id.btn_submit);
    btn.setOnClickListener(btn_submit_listener);
/btn = (Button) findViewById(R.id.btn_close);
//btn.setOnClickListener(btn_close_listener);
//btn = (Button) findViewById(R.id.btn_play_again);
//btn.setOnClickListener(btn_play_again_listener);
Log.i(log_tag, "All button Listeners assigned");

//Get get current activity intent for restart
intent = this.getIntent();

//Try Kick
btn = (Button) findViewById(R.id.btn_next_screen);
btn.setOnClickListener(btn_next_screen_listener);

/* **** Attempt to load data at beginning of the application run **** */
private void attempt_data_load() {
    File file = new File("data/data/hello.formstuff/files/" + activity1.file_name);
    if (file.exists()) {
        //Get App Data ready for post in an Array List
        FileOperator file_operator1 = new FileOperator(file_name, activity1);
        List<NameValuePair> data_list1 = new ArrayList<NameValuePair>;
        data_list1 = file_operator1.createArrayList();
        PostAppData task1;

        //Pass the array list to Async Task for sending the data
        task1 = new PostAppData(data_list1, activity1);

        //Post this array list to remote server
        task1.execute(new String[] { post_to_url });
    }
}

/* **** Combining the activities **** */
//Listening to button event
private OnClickListener btn_next_screen_listener = new OnClickListener() {
    public void onClick(View v) {
        //Starting a new Intent

Intent nextScreen = new Intent(getApplicationContext(),
VirtualLab.class);
// Sending data to another Activity
// nextScreen.putExtra("download_url", download_url);
nextScreen.putExtra("downloadurl", download_url);
startActivity(nextScreen);
finish();

} }

/* **** Other Form Button Listeners **** */

// Form Submit Button
private OnClickListener btn_submit_listener = new OnClickListener() {
    public void onClick(View v) {
        check_questions();
        collect_data();

        // Get Main View
        vw = (LinearLayout) findViewById(R.id.ll);

        // Do data post request in background and update UI accordingly
        // when the post data request is finished
        task = new PostAppData(data_list, activity1);
        task.execute(new String[] { post_to_url });
        uiUpdateOnChallengeFormSubmit();
    }
};

// Play Again Button
private OnClickListener btn_play_again_listener = new OnClickListener() {
    public void onClick(View v) {
        // restart gameplay activity
        finish();
        startActivity(intent);
    }
};
//Close Application Button
private OnClickListener btn_close_listener = new
OnClickListener()
{
    public void onClick(View v) {
        //Finish android activity
        finish();
    }
};

//Update User Interface(UI)
public void uiUpdateOnChallengeFormSubmit() {

    //Hide Form Title, tv_title
    tv = (TextView) vw.findViewById(R.id.tv_title);
    tv.setVisibility(vw.GONE);
    //Hide Challenge Question 1, tv_cq1
    tv = (TextView) vw.findViewById(R.id.tv_cq1);
    tv.setVisibility(vw.GONE);
    //Hide Challenge Question 1 Responses, rb_cq11, rb_cq12, rb_cq13, rb_cq14
    rb = (RadioButton) vw.findViewById(R.id.rb_cq11);
    rb.setVisibility(vw.GONE);
    rb = (RadioButton) vw.findViewById(R.id.rb_cq12);
    rb.setVisibility(vw.GONE);
    rb = (RadioButton) vw.findViewById(R.id.rb_cq13);
    rb.setVisibility(vw.GONE);
    rb = (RadioButton) vw.findViewById(R.id.rb_cq14);
    rb.setVisibility(vw.GONE);
    //Hide Challenge Question 2, tv_cq2
    tv = (TextView) vw.findViewById(R.id.tv_cq2);
    tv.setVisibility(vw.GONE);
    //Hide Challenge Question 2 Responses, rb_cq21, rb_cq22, rb_cq23, rb_cq24
    rb = (RadioButton) vw.findViewById(R.id.rb_cq21);
    rb.setVisibility(vw.GONE);
    rb = (RadioButton) vw.findViewById(R.id.rb_cq22);
    rb.setVisibility(vw.GONE);
    rb = (RadioButton) vw.findViewById(R.id.rb_cq23);
    rb.setVisibility(vw.GONE);
    rb = (RadioButton) vw.findViewById(R.id.rb_cq24);
    rb.setVisibility(vw.GONE);
    //Hide Submit Button, btn_submit
    btn = (Button) vw.findViewById(R.id.btn_submit);
    btn.setVisibility(vw.GONE);
//Show Close Button, btn_close
// btn = (Button) vw.findViewById(R.id.btn_close);
// btn.setVisibility(vw.VISIBLE);
//Show Play Again Button, btn_play_again
// btn = (Button) vw.findViewById(R.id.btn_play_again);
// btn.setVisibility(vw.VISIBLE);
//Show Try Kick Button, btn_next_screen
btn = (Button) vw.findViewById(R.id.btn_next_screen);
btn.setVisibility(vw.VISIBLE);
//Show feedback, tv_feedback
put_feedback();
Log.i(log_tag, "User Interface(UI) updated");
}

//Check Answers
public int check_questions() {
    int checked = 0;
    //Number of questions
    int no_of_questions = 0;
    int total_correct_responses = 0;
    String response_string = "";
    //To begin with assume all responses are correct i.e. success = 1
    String success = "true";
    //Check answer for challenge question 1
    no_of_questions++;
    rg = (RadioGroup) findViewById(R.id.rg_cq1);
    int id1 = rg.getCheckedRadioButtonId();
    rb = (RadioButton) findViewById(R.id.rb_cq11); //Correct answer
    int id2 = rb.getId();
    if (id1 == id2) {  //Correct response
        if (success != "false") {
            success = "true";
        }
        total_correct_responses++;
        response_string = response_string + "1,";
        Log.i(log_tag, "Responses to challenge question 1 is correct");
    }
    else {
        success = "false";
        if (id1 == -1) {

        }
    }

}
response_string = response_string + "NA,";
} else {
    response_string = response_string + "0,";
}
Log.i(log_tag, "Responses to challenge question 1 is found to be incorrect");

//Check answer for challenge question 2
no_of_questions++;
rg = (RadioGroup) findViewById(R.id.rg_cq2);
id1 = rg.getCheckedRadioButtonId();
rb = (RadioButton) findViewById(R.id.rb_cq22); //Correct answer
id2 = rb.getId();
if (id1 == id2) { //Correct response
    if (success != "false") {
        success = "true";
    }
    total_correct_responses++;
    response_string = response_string + "1";
    Log.i(log_tag, "Responses to challenge question 2 is correct");
} else {
    success = "false";
    if (id1 == -1) {
        response_string = response_string + "NA";
    } else {
        response_string = response_string + "0";
    }
    Log.i(log_tag, "Responses to challenge question 2 is found to be incorrect");
}

//All questions checked, now log whether all responses were correct or not
if (success == "false") {
    Log.i(log_tag, "Responses to challenge questions checked as not all correct");
} else if (success == "true") {
    Log.i(log_tag, "Responses to challenge questions checked as all correct");
}
gather_data[0] = success; //check answers
gather_data[1] = String.valueOf(total_correct_responses); //Total number of questions
gather_data[3] = String.valueOf(no_of_questions); //Total correct responses
checked = 1;
return checked;
}

//Get feedback
public void put_feedback() {

    tv = (TextView) vw.findViewById(R.id.tv_feedback);

    //Check if the Challenge answer is correct. If yes
    //Also provide download next app link
    if ( gather_data[0] == "true") {
        feedback = "You have correctly answered all challenge questions!";
    }
    else if ( gather_data[0] == "false" ) {
        feedback = "You have failed to answer challenge questions correctly!";
    }

    tv.setText(feedback);
    tv.setTextSize(TypedValue.COMPLEX_UNIT_DIP,20);
    tv.setVisibility(vw.VISIBLE);
    Log.i(log_tag, "feedback provided");
}

//Get feedback
public void put_download_url() {

    //Check if all the Challenge answers are answered correctly. If yes
    //Provide download next app link
    if ( gather_data[0] == "true") {
        answered_correctly = true;
    }
tv = (TextView) vw.findViewById(R.id.tv_download_url);

//Check if the Challenge answer is correct. If yes
//Also provide download next app link
download_url = task.getCookieValue("next_app_download_url");
feedback = "Download next app here <a href='" + url_string +
download_url + ".html'>Next App</a>";
//Activate HTML Links in Feedback TextView
pattern = Pattern.compile(url_string + download_url + ".html");
tv.setMovementMethod(LinkMovementMethod.getInstance());
Linkify.addLinks(tv, pattern, "http://");
tv.setLinkTextColor(Color.rgb(238,135,0));
tv.setText(Html.fromHtml(feedback));
tv.setTextSize(TypedValue.COMPLEX_UNIT_DIP,20);
tv.setVisibility(vw.VISIBLE);
Log.i(log_tag, "Download URL Posted");
}

//Collect all game app data
public void collect_data() {

String id_str1 = "#NA#";
String id_str2 = "#NA#";

//Get App Data ready for post in an Array List
file_operator = new FileOperator(file_name, activity1);
Element element1 = new Element("username", "poonam");
i = file_operator.writeDataToFile(element1);
Element element2 = new Element("password", "password");
i = file_operator.writeDataToFile(element2);

//Get response for challenge question 1
rg = (RadioGroup) findViewById(R.id.rg_cq1);
int id1 = rg.getCheckedRadioButtonId();
if (id1 != -1) {
    rb = (RadioButton) findViewById(id1);
    id_str1 = rb.getText().toString();
}

//Get response for challenge question 2
rg = (RadioGroup) findViewById(R.id.rg_cq2);
int id2 = rg.getCheckedRadioButtonId();
if (id2 != -1) {
    rb = (RadioButton) findViewById(id2);
    id_str2 = rb.getText().toString();
}

Element element3 = new Element("challengeQuestionResponses",
    id_str1 + "," + id_str2);
i = file_operator.writeDataToFile(element3);

Log.i(log_tag, "Device ID being sent is: " + device_id);

Element element4 = new Element("email", "" + device_id);
i = file_operator.writeDataToFile(element4);

Element element5 = new Element("appCode", "VL3.901_01");
i = file_operator.writeDataToFile(element5);

Element element6 = new Element("challengeResponsesEvaluation",
    gather_data[2]);
i = file_operator.writeDataToFile(element6);

Element element7 = new Element("totalCorrectResponses",
    gather_data[1]);
i = file_operator.writeDataToFile(element7);

Element element8 = new Element("totalNoOfQuestions",
    gather_data[3]);
i = file_operator.writeDataToFile(element8);

file_operator.markEndOfFile();
data_list = file_operator.createArrayList();
}

package hello.formstuff;
import android.util.Log;

public class mybox {
    float ar = (float) 1.5;
float[] theta; //this array holds the values of theta that describe each box's angle with the floor
float[] W; //?
float[] comx; //?
float[] comy; //?
float[] X0; //holds the initial x positions of the corners of the box
float[] Y0; //holds the initial y positions of the corners of the box
float[] ths0; //?
float[] ybot; //?? presumably the location of the floor that the box rests on
float[][] X,Y,ths; //X and Y represent the positions of the corners of the box(es)
float sinth; //?? presumable sine of the angle between the side of the box and the floor
float costh; //?? cosine of the angle between the side of the box and the floor
float windoww; //width of the display window
float windowh; //height of the display window
float r; //?
float old; //?
float scale=1; //this is a scaling factor used when multiple boxes are displayed
float w; //this is the width of the boxes to be displayed
float h; //this is the height of the boxes to be displayed
int N; //this is the total number of boxes displayed in the window
int n; //this is the number of columns of boxes displayed in the window
int m; //this is the number of rows of boxes displayed in the window
int s; //I don't know what this is
int[] sold;
int[] atrest;
float[] center;
float centerX;
float centerY;
int degrees;

public mybox(float w,float h,int n,int m) {
    this.N=n*m;
    this.n=n;
    this.m=m;
    theta=new float[N];
    W=new float[N];
    comx=new float[N];
    comy=new float[N];
    sold=new int[N];
    ybot=new float[N];
X=new float[4][N];
Y=new float[4][N];
ths=new float[4][N];
X0=new float[4];
Y0=new float[4];
ths0=new float[4];
center=new float[2];
deegres=180;
X0[0]=w/2;
Y0[0]=h/2;
X0[1]=w/2;
Y0[1]=-h/2;
X0[2]=-w/2;
Y0[2]=-h/2;
X0[3]=-w/2;
Y0[3]=h/2;

for(int i=0;i<4;i++) {
    ths0[i]=(float)Math.atan2(Y0[i],X0[i]);
}

r=(float)Math.sqrt(w*w+h*h)/2;
this.w=w;
this.h=h;
atrest = new int[N];
atrest[0] = 1;
center[0]=(X0[0]+X0[3])/2;
center[1]=(Y0[0]+Y0[3])/2;
centerX=center[0];
centerY=center[1];

return;
}

void refreshbox(int i){
sinth=(float)Math.sin(theta[i]);
costh=(float)Math.cos(theta[i]);

for(int j=0;j<4;j++) {
    ths[j][i]=ths0[j]+theta[i];
}

for(int j=0;j<4;j++) {
    X[j][i]=X0[j]*costh*scale-Y0[j]*sinth*scale+comx[i];
    Y[j][i]=Y0[j]*costh*scale+X0[j]*sinth*scale+ybot[i]-
    (float)Math.sin(ths[s][i])*r*scale;
}
center[0]=(X[0][0]+X[3][0])/2;
center[1]=(Y[0][0]+Y[3][0])/2;
degrees = Math.round((float)(theta[0]*180/Math.PI));
}

void refreshbox(){
    sinh=(float)Math.sin(theta[0]);
    costh=(float)Math.cos(theta[0]);
    for(int j=0;j<4;j++) {
        ths[j][0]=ths0[j]+theta[0];
    }
    for(int j=0;j<4;j++) {
        X[j][0]=X0[j]*costh*scale-Y0[j]*sinth*scale+comx[0];
        Y[j][0]=Y0[j]*costh*scale+X0[j]*sinth*scale+ybot[0]-
             (float)Math.sin(ths[s][0])*r*scale;
    }
    center[0]=(X[0][0]+X[3][0])/2;
    center[1]=(Y[0][0]+Y[3][0])/2;
    degrees = Math.round((float)(theta[0]*180/Math.PI));
}

void setposition(){
    float dx,dy;
    dx=windoww/2;
    dy=windowh/2;
    comx[0]=dx;
    ybot[0]=dy-Math.max(h/2,w/2)*scale;
}

package hello.formstuff;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.Rect;
import android.view.View;

class MyDrawing extends View
{
    private Rect myRect;
    private Point myCenter;
    private Paint rectFillPaint = new Paint();
    private Paint pointPaint = new Paint();
    Paint rectOutlinePaint = new Paint();
    public int degree;

    private Matrix transform = new Matrix();
    private mybox b;
    private boolean kick=false;
    private boolean residual=false;
    private float kickmag;
    private int dcount = -1;

    public MyDrawing(Context context, Rect rect)
    {
        super(context);

        //Get rectangular box
        myRect = rect;
        //Calculate center of the rectangular box
        myCenter = new Point( (int) myRect.exactCenterX(), (int) myRect.exactCenterY() );

        //Creating mybox object
        float ar = (float)1.5;
        int boxwidth = 100;
        int boxheight = (int)(((float)boxwidth)*ar);
b=new mybox(boxwidth,boxheight,1,1);
b.windowh=400;
b.windoww=400;
b.theta[0]=(float)Math.PI;
b.scale=(float)1.35/b.n;
b.setPosition();
b.refreshBox();
}

public void setDegree(int ndegree){
    degree=ndegree;
}

public void setKick(boolean truefalse){
    kick = truefalse;
}

public void getRotationMatrix()
{
    transform.setRotate(b.degrees, myRect.exactCenterX(),
    myRect.exactCenterY());
}

public void rotateBoxPoints() {
    //Calculate degree of rotation
    kickupdate();

    //Get Rotation matrix
    getRotationMatrix();

    //centerTranslate();

    // Call onDraw method of the view
    invalidate();
}

public void centerTranslate(){
    float Tx=-b.centerX+b.centerX;
    float Ty=-b.centerY+b.centerY;
    transform.setTranslate(Tx,Ty);
}

public boolean isCalculating(){
return (kick || residual);
}

public void setKickMag(float nkickmag){
    kickmag=(float)(nkickmag/1.0e5);
}

@Override
public void onDraw(Canvas canvas){
    if(myRect != null && myCenter != null){
        rectFillPaint.setColor(Color.RED);
        pointPaint.setColor(Color.YELLOW);
        rectOutlinePaint.setColor(Color.BLACK);
        canvas.setMatrix(null);
        //Draw Surface
        //canvas.drawLine(300, 623 , 660, 623, rectOutlinePaint);
        canvas.setMatrix(transform);
        //Draw rectangular box outline
        Rect myRectOutline = new Rect(397,397,563,623);
        canvas.drawRect(myRectOutline, rectOutlinePaint);
        //Draw Rectangle Box on rotated canvas
        canvas.drawRect(myRect, rectFillPaint);
        //Draw center point
        //canvas.drawCircle(myCenter.x, myCenter.y, 5, pointPaint);
        //Draw rectangular box diagonal lines
        canvas.drawLine(397, 397 , 563, 623, rectOutlinePaint);
        canvas.drawLine( 563, 397, 397, 623, rectOutlinePaint);
    }
}

/**
 *
* Function module which implements the logic to calculate degree rotation
*
*/

public void kickupdate()
{
    if (kick || residual)
    {

        //Section 1
        if (dcount > -1)
        {
            dcount++;
            if (dcount % 75 == 0)
            {
                dcount=0;
            }
        }

        //Section 2
        b.theta[0]+=b.W[0];
        b.old=b.Y[0][0];
        b.s=0;

        //Section 3
        for(int j=0;j<4;j++)
        {
            if(b.Y[j][0]<b.old)
            {
                b.old=b.Y[j][0]; b.s=j;
            }
        }

        //Section 4
        if(Math.abs(b.theta[0] - Math.round(b.theta[0]*2/Math.PI)*Math.PI/2) > 1.0e-7)
        {
            b.W[0]+=(float)(.0001*Math.cos(b.ths[b.s][0]));
        }

        //Section 5
        if(b.sold[0]!=b.s)
        {
            residual=false;
            b.W[0]=0;
            dcount=-1;
        }
    }
}
b.theta[0]=(float)(Math.round(b.theta[0]*2/Math.PI)*Math.PI/2);
    setdegree((int)Math.round(b.theta[0]*180/Math.PI));
}

//Section 6
if(kick)
{
    //if the box is standing up, kick it over
    if( (b.theta[0]>=Math.PI-5*1.0e-8) &&
        (b.theta[0]<=Math.PI+1.5*1.0e-7) )
    {
        dcount=0;
        b.W[0]=kickmag;
        kick = false;
        residual=true;
    }
    //if the box is lying down, kick it up
    else if ( (b.theta[0]>3*Math.PI/2-5*1.0e-8) &&
        (b.theta[0]<=3*Math.PI/2+5*1.0e-8) )
    {
        b.W[0]=-1 * kickmag;
        dcount=0;
        kick = false;
        residual=true;
    }
}

//Section 7
b.comy[0]=b.ybot[0]-(float)Math.sin(b.ths[b.s][0])*b.r*b.scale;
    b.sold[0]=b.s;
    b.refreshbox(0);
}
package hello.formstuff;

import java.math.BigDecimal;
import java.util.Timer;
import java.util.TimerTask;
import java.util.regex.Pattern;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Color;
import android.graphics.Rect;
import android.os.Bundle;
import android.os.Message;
import android.text.Html;
import android.text.method.LinkMovementMethod;
import android.text.util.Linkify;
import android.util.Log;
import android.util.TypedValue;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.FrameLayout;
import android.widget.LinearLayout;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;

public class VirtualLab extends Activity implements OnSeekBarChangeListener{
    //View to draw rectangular box and its components
    MyDrawing myDrawing;
    Rect myRect = new Rect(400,400,560,620);

    //Record Success
    int success = 0;

    //Kick 1
    SeekBar seekbar_kickmag1;
    int kick1mag = 0;
    Button btn_kick1;

    //Kick 2
    SeekBar seekbar_kickmag2;
int kick2mag = 0;
Button btn_kick2;

//TextView to get value of timer counter
TextView mTextView;

//Timer variables
Timer mTimer = null;
TimerTask mTimerTask = null;
static int delay = 10;  //1s
static int period = 10;  //1s
static int timer_call_count = 0;

//Message Handler for Virtual Lab
Handler mHandler = null;

//Other Variables
boolean isNotKicked = true;
static final int UPDATE_TEXTVIEW = 0;
float ar = (float) 1.5;
double progress_value = 0;

Intent intent;
Button btn;
View vw;
SeekBar sb;
TextView tv;
    View custom_vw;
    String feedback = "";
    String download_url1 = "error";
    Pattern pattern;
    String url_string = "http://devdesktop.cmi.kent.edu/";

    //Activity_string
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_screen2);

        //Get get current activity intent for restart
        intent = this.getIntent();
        download_url1 = intent.getStringExtra("downloadurl");
/* 
 * CUSTOM VIEW
 */

//Create custom view to draw the rectangle box with clearly marked center
myDrawing = new MyDrawing(this, myRect);
//Add view to placeholder
((ViewGroup) findViewById(R.id.myDrawing_placeholder)).addView(myDrawing);

/* 
 * SEEKBARS
 */

//Get TextView to display kick magnitude
mTextView = (TextView) findViewById(R.id.mytextview);

//Get seekbar_kickmag1
seekbar_kickmag1 = (SeekBar) findViewById(R.id.seekbar_kickmag1);
seekbar_kickmag1.setOnSeekBarChangeListener(this);

//Get seekbar_kickmag2
seekbar_kickmag2 = (SeekBar) findViewById(R.id.seekbar_kickmag2);
seekbar_kickmag2.setOnSeekBarChangeListener(this);

/* 
 * BUTTONS
 */

//Get Kick1 Button and assign listener
btn_kick1 = (Button) findViewById(R.id.btn_kick1);
btn_kick1.setOnClickListener(btn_kick1_lstnr);

//Get Kick2 Button and assign listener
btn_kick2 = (Button) findViewById(R.id.btn_kick2);
btn_kick2.setOnClickListener(btn_kick2_lstnr);

/* 
 * MESSAGE HANDLER
 */

mHandler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case UPDATE_TEXTVIEW:
            // Handle message
        }
    }
}
updateView();
break;
default:
    break;
}
}

//Update View
private void updateView()
{
    if (myDrawing.isCalculating()){
        //mTextView.setText(String.valueOf(timer_call_count));
        myDrawing.rotateBoxPoints();
    }
    else {
        stopTimer();
        //Also, check if success
        if (success == 2) {
            btn = (Button) findViewById(R.id.btn_close);
            btn.setOnClickListener(btn_close_listener);
            btn = (Button) findViewById(R.id.btn_play_again);
            btn.setOnClickListener(btn_play_again_listener);

            //Get Main View
            vw = (LinearLayout) findViewById(R.id.container);

            //Hide Try Kick components
            //android:id="@+id/mytextview"
            tv = (TextView) vw.findViewById(R.id.mytextview);
            tv.setVisibility(vw.GONE);
            //android:id="@+id/seekbar_kickmag1"
            sb = (SeekBar) vw.findViewById(R.id.seekbar_kickmag1);
            sb.setVisibility(vw.GONE);
            //android:id="@+id/btn_kick1"
            btn = (Button) vw.findViewById(R.id.btn_kick1);
            btn.setVisibility(vw.GONE);
            //android:id="@+id/seekbar_kickmag2"
            sb = (SeekBar) vw.findViewById(R.id.seekbar_kickmag2);
            sb.setVisibility(vw.GONE);
            //android:id="@+id/btn_kick2"
            btn = (Button) vw.findViewById(R.id.btn_kick2);
            btn.setVisibility(vw.GONE);
            //android:id="@+id/myDrawing_placeholder"
custom_vw = (FrameLayout) findViewById(R.id.myDrawing_placeholder);
custom_vw.setVisibility(vw.GONE);

//android:id="@+id/kick_success_msg"
tv = (TextView) vw.findViewById(R.id.kick_success_msg);
tv.setVisibility(vw.VISIBLE);

Boolean temp_bool = download_url1.equals("error");
if (!temp_bool ) {
    put_download_url();
}

//Try Kick
btn = (Button) findViewById(R.id.btn_next_screen);
btn.setOnClickListener(btn_next_screen_listener);

//Show Close Button, btn_close
btn = (Button) vw.findViewById(R.id.btn_close);
btn.setVisibility(vw.VISIBLE);

//Show Play Again Button, btn_play_again
btn = (Button) vw.findViewById(R.id.btn_play_again);
btn.setVisibility(vw.VISIBLE);

//Show Try Kick Button, btn_next_screen
btn = (Button) vw.findViewById(R.id.btn_next_screen);
btn.setVisibility(vw.VISIBLE);

//Get feedback
public void put_download_url() {

    //Check if all the Challenge answers are answered correctly. If yes
    //Provide download next app link
    tv = (TextView) vw.findViewById(R.id.tv_download_url);

    //Check if the Challenge answer is correct. If yes
    //Also provide download next app link
// Receiving the Data

        feedback = "Download next app here <a href="" + url_string +
download_url1 + ".html'>Next App</a>";
        //Activate HTML Links in Feedback TextView
        pattern = Pattern.compile(url_string + download_url1 +
        ".html");
        tv.setMovementMethod(LinkMovementMethod.getInstance());
        Linkify.addLinks(tv, pattern, "http://");
        tv.setTextColor(Color.rgb(238,135,0));
        tv.setText(Html.fromHtml(feedback));
        tv.setTextSize(TypedValue.COMPLEX_UNIT_DIP,20);
        tv.setVisibility(vw.VISIBLE);
    }

    //Listeners

    //Next Screen Button
    private OnClickListener btn_next_screen_listener = new
    OnClickListener() {
        public void onClick(View v) {

            //restart gameplay activity
            finish();
            startActivity(intent);
        }
    }

    //Play Again Button
    private OnClickListener btn_play_again_listener = new OnClickListener() {
        public void onClick(View v) {

            //Starting a new Intent
            Intent nextScreen = new Intent(getApplicationContext(),
            HelloFormStuffActivity.class);
            startActivity(nextScreen);
            finish();
        }
    };
//Close Application Button
private OnClickListener btn_close_listener = new OnClickListener() {
    public void onClick(View v) {
        //Finish android activity
        //av.getParent().finish();
        finish();
    }
};

//OnClick Listener for "btn_kick1" button
private OnClickListener btn_kick1_lstnr = new OnClickListener() {
    public void onClick(View v) {
        myDrawing.setKickMag(seekbar_kickmag1.getProgress());
        myDrawing.setKick(true);
        startTimer();
    }
};

//OnClick Listener for "btn_kick2" button
private OnClickListener btn_kick2_lstnr = new OnClickListener() {
    public void onClick(View v) {
        myDrawing.setKickMag(seekbar_kickmag2.getProgress());
        myDrawing.setKick(true);
        startTimer();
    }
};

//onProgressChanged listener for "seekbar_kickmag1"
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    switch (seekBar.getId()) {
        case R.id.seekbar_kickmag1:
            progress_value = ((float) ((float) progress/100000.0) /
(0.0000239996 * Math.log(-0.291078 + (double)ar) + 0.000377782));
            progress_value = round(progress_value, 6,
            BigDecimal.ROUND_HALF_UP);
            mTextView.setText(" " + progress_value);
    }
}
break;

    case R.id.seekbar_kickmag2:
        progress_value = ((float) ( (float) progress/100000.0)/(0.000394011/(-0.0127876 + (double)ar)) - 0.0000290327));
        progress_value = round(progress_value, 6, BigDecimal.ROUND_HALF_UP);
        mTextView.setText(" " + progress_value);
        break;
    }

    //onStartTrackingTouch listener for "seekbar_kickmag1"
    public void onStartTrackingTouch(SeekBar seekBar) {} 

    //onStopTrackingTouch listener for "seekbar_kickmag1"
    public void onStopTrackingTouch(SeekBar seekBar) {} 

    /**
     * Function to send message to message handler
     */
    public void sendMessage(int id){
        if (mHandler != null) {
            Message message = Message.obtain(mHandler, id);
            mHandler.sendMessage(message);
        }
    }

    /**
     * Function to start timer
     */
    private void startTimer(){
        isnotKicked = false;

        //Create Timer
        if (mTimer == null) {
            mTimer = new Timer();
        }

        //Create TimerTask
        if (mTimerTask == null) {
            mTimerTask = new TimerTask() {
                @Override
                public void run() {

                }
            }
        }

    }
sendMessage(UPDATE_TEXTVIEW);
timer_call_count ++;
};
};//Schedule Timer using TimerTask
if(mTimer != null && mTimerTask != null ) {
   mTimer.schedule(mTimerTask, delay, period);
}
}

/**
 * Function to Stop Timer
 */
private void stopTimer(){
    isNotKicked = true;

    ///** Logic for enabling and disabling Kick1 and Kick2 **//

    //If box is horizontal then disable kick1 and enable kick2
    if (myDrawing.degree == 270)
    {
        success = 1;
        btn_kick1.setEnabled(false);
        btn_kick2.setEnabled(true);
        progress_value = (double) seekbar_kickmag2.getProgress();
        progress_value = ((float) (progress_value/100000.0)/((0.000394011/(-0.0127876 + (double)ar)) - 0.0000290327));
        progress_value = round(progress_value, 6, BigDecimal.ROUND_HALF_UP);
        mTextView.setText(" " + progress_value);
    }

    //If box is vertical then disable kick 2 and enable kick1
    if (myDrawing.degree == 180 )
    {
        if (success == 1)
        {
            success = 2; //Kicked successfully
        }
        btn_kick1.setEnabled(true);
        btn_kick2.setEnabled(false);
        progress_value = (double) seekbar_kickmag1.getProgress();
    }
progress_value = ((float) (progress_value/100000.0) / (0.0000239996 * Math.log(-0.291078 + (double)ar) + 0.000377782));
progress_value = round(progress_value, 6, BigDecimal.ROUND_HALF_UP);
mTextView.setText("" + progress_value);
}

//Cancel Timer
if (mTimer != null) {
    mTimer.cancel();
    mTimer = null;
}

//Cancel TimerTask
if (mTimerTask != null) {
    mTimerTask.cancel();
    mTimerTask = null;
}
timer_call_count = 0;

//Method to round float values
public static double round(double unrounded, int precision, int roundingMode)
{
    BigDecimal bd = new BigDecimal(unrounded);
    BigDecimal rounded = bd.setScale(precision, roundingMode);
    return rounded.doubleValue();
}

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="tv_title"></string>
    <string name="tv_cq1">Question 1: Minimum approximate magnitude of Kick 1 necessary to produce a transition</string>
    <string name="rb_cq11">14-16</string>
    <string name="rb_cq12">22-26</string>
    <string name="rb_cq13">17-19</string>
    <string name="rb_cq14">18-20</string>
</resources>
Question 2: Minimum approximate magnitude of Kick 2 necessary to produce a transition

- rb_cq21: 42-24
- rb_cq22: 39-41
- rb_cq23: 23-25
- rb_cq24: 37-39

Try Kick!
android:id="@+id/ll1"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:isScrollContainer="true">

<!-- Form Title -->
<
    <View
        android:id="@+id/tv_title"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="30px"
        android:text="@string/tv_title"
    />

<!-- Challenge Question 1 -->
<
    <View
        android:id="@+id/tv_cq1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="30px"
        android:text="@string/tv_cq1"
        android:textSize="20.0px"/>

<!-- Challenge Question 1 Responses -->
<
    <View
        android:id="@+id/rg_cq1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="30px"
        android:orientation="vertical">
        <
            <View
                android:id="@+id/rb_cq11"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/rb_cq11" />
        <
            <View
                android:id="@+id/rb_cq12"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/rb_cq12" />
        <
            <View
                android:id="@+id/rb_cq13"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/rb_cq13" />
        <
            <View
                android:id="@+id/rb_cq14"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/rb_cq14" />
    </View>
</View>
</View>
<!-- Challenge Question 2 -->
< TextView
android:id="@+id/tv_cq2"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:paddingBottom ="30px"
android:text="@string/tv_cq2"
android:textSize="20.0px"/>

<!-- Challenge Question 2 Responses -->
<RadioGroup
android:id="@+id/rg_cq2"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:paddingBottom ="30px"
android:orientation="vertical">
<RadioButton android:id="@+id/rb_cq21"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/rb_cq21" />
<RadioButton android:id="@+id/rb_cq22"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/rb_cq22" />
<RadioButton android:id="@+id/rb_cq23"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/rb_cq23" />
<RadioButton android:id="@+id/rb_cq24"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/rb_cq24" />
</RadioGroup>

<!-- Submit Button -->
<Button
android:id="@+id/btn_submit"
android:layout_width="100px"
android:layout_height="50px"
android:padding="10px"
android:text="@string/btn_submit" />

<!-- On Submit Challenge Questions Form -->

<!-- Feedback TextView -->
<TextView
android:id="@+id/tv_feedback"
<?xml version="1.0" encoding="utf-8"?>

<ScrollView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:fillViewport="true"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <LinearLayout
        android:id="@+id/container"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="30px"
        android:visibility="gone" />

    <!-- Next Screen Button -->
    <Button
        android:id="@+id/btn_next_screen"
        android:layout_width="120px"
        android:layout_height="50px"
        android:padding="10px"
        android:paddingBottom="30px"
        android:text="@string/btn_next_screen"
        android:visibility="gone" />

</LinearLayout>
</ScrollView>
</LinearLayout>

<?xml version="1.0" encoding="utf-8"?>

<ScrollView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:fillViewport="true"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <LinearLayout
        android:id="@+id/container"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="30px"
        android:visibility="gone" />

    <!-- Next Screen Button -->
    <Button
        android:id="@+id/btn_next_screen"
        android:layout_width="120px"
        android:layout_height="50px"
        android:padding="10px"
        android:paddingBottom="30px"
        android:text="@string/btn_next_screen"
        android:visibility="gone" />

</LinearLayout>
</ScrollView>
</LinearLayout>
android:layout_height="match_parent">

<TextView
    android:id="@+id/mytextview"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:textSize="60dp"
    android:layout_marginTop="40dp"
    android:text="@string/number" />

<!-- Seekbar for Kick1 Magnitude -->
<SeekBar
    android:id="@+id/seekbar_kickmag1"
    android:layout_width="600px"
    android:layout_height="wrap_content"
    android:max="1500"
    android:progress="350"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true" />

<!-- Kick1 Button -->
<Button
    android:id="@+id/btn_kick1"
    android:layout_width="300px"
    android:layout_height="50px"
    android:padding="10px"
    android:text="@string/btn_kick1" />

<!-- Seekbar for Kick2 Magnitude -->
<SeekBar
    android:id="@+id/seekbar_kickmag2"
    android:layout_width="600px"
    android:layout_height="wrap_content"
    android:max="1500"
    android:progress="350"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true" />

<!-- Kick2 Button -->
<Button
    android:id="@+id/btn_kick2"
    android:layout_width="300px"
    android:layout_height="50px"
    android:padding="10px"
android:enabled="false"
android:text="@string/btn_kick2" />

<TextView
android:id="@[id/kick_success_msg"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="30dp"
android:layout_marginTop="40dp"
android:text="@string/kick_success_msg"
android:paddingBottom ="50px"
android:visibility="gone" />

<TextView
android:id="@[id/tv_download_url"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:paddingBottom ="30px"
android:visibility="gone"
/>  

<!-- Play Again Button -->
<Button
android:id="@[id/btn_play_again"
android:layout_width="160px"
android:layout_height="50px"
android:padding="10px"
android:paddingBottom ="30px"
android:text="@string/btn_play_again"
android:visibility="gone" />

<!-- Next Screen Button -->
<Button
android:id="@[id/btn_next_screen"
android:layout_width="120px"
android:layout_height="50px"
android:padding="10px"
android:paddingBottom ="30px"
android:text="@string/btn_next_screen"
android:visibility="gone" />

<!-- Close Button -->
<Button
android:id="@[id/btn_close"
android:layout_width="120px"
android:layout_height="50px"
android:padding="10px"
android:paddingBottom ="30px"
android:text="@string/btn_close"
android:visibility="gone" />

<!-- Placeholder to hold myDrawing -->
<FrameLayout
    android:id="@+id/myDrawing_placeholder"
    android:orientation="vertical"
    android:layout_width="1000px"
    android:layout_height="1000px"
    android:paddingTop ="10px"
>
</FrameLayout>

</LinearLayout>
</ScrollView>