Project report

Android Game (Dots Game)

Submitted By

Manik Kumer Kundu

ID: 2010-2-60-012

Chowdhury Rayhan Sahrear

ID: 2010-2-60-023

SUPERVISED BY

Dr. Shaikh Muhammad Allayear

Assistant Professor

Department of Computer Science and Engineering.

Department of Computer Science and Engineering

East West University, Dhaka, Bangladesh

May 2015

Declaration

This is to certify that this project is an original work and done by us under CSE 497 course and it has not been submitted elsewhere for the requirement of any degree for any other purpose except for publication. The materials that are obtained from other source are duly acknowledged in this project.

Student's Name & Signature:

_____

Manik Kumer Kundu

ID: 2010-2-60-012

_____

Chowdhury Rayhan Sahrear

ID: 2010-2-60-023

Letter of Acceptance

The project entitled "Android Game(Dots game)" submitted by Manik Kumer Kundu, ID: 2010-2-60-012 and Chowdhury Rayhan Sahrear, ID: 2010-2-60-023 to the Department of Computer Science and Engineering , East West University, Dhaka-1212, Bangladesh is accepted by the Department for the partial fulfillment of requirements for the degree of Bachelor of Science in Computer Science and Engineering on May 14, 2015.

_____

Dr. Shaikh Muhammad Allayear

Assistant Professor

Department of Computer Science and Engineering,

East West University, Dhaka-1212, Bangladesh

_____

Dr. Shamim Hasnat Ripon,

Chairperson and Associate Professor,

Department of Computer Science and Engineering,

East West University Dhaka-1212, Bangladesh

Acknowledgements

The project is about "Android Game(Dots game)" would never run successfully without the valuable encouragement and guidance from our supervisor Dr. Shaikh Muhammad Allayear Assistant Professor, Department of Computer Science and Engineering, East West University. He enlightened, encouraged and provided us with the ingenuity to transform our vision into reality. We are particularly grateful to Dr. Shamim Hasnat Ripon, Chairperson and Associate Professor, Department of Computer Science and Engineering, East West University, for his encouragement. We are also grateful to many of our friends who provided us with necessary hardware to test the software as it developed. We would like to thank Google for creating android platform and keeping it free.

ABSTRACT

Mobile application development is one of the recent trends in computing Industry.
Among several existing platforms for mobile, Android is one of the largest platforms in the world that run in several smart phones and tablets from various manufacturers like Google, Samsung, HTC etc. The project we have developed is a game for Android platform. This report contains the design and architecture of android, about SDK, version of android and implementation and details (feature, different states of the game, how to play, scoring) about the game which we have developed. Design and develop useful android applications with user interfaces by using, extending, and creating own layouts and views.
The project has been developed in Java Programming Language by using the Eclipse. We have used the Android software development applications on the Android platform. The most important of these are the android emulator and the android development tools plug-in for eclipse.

Table of Contents

List of Figure

Chapter 1

Introduction

1.1 Introduction

Android is one of the leading and fastest growing mobile platforms today. Android is an open source software and the Android SDK is available to developers for free. The minimal setup time and the number of tools available for android SDK to ease the development process allows the developers to concentrate in the design and implementation details of the application in the available timeframe. The main objective of this project is to experience developing an android mobile application which is one of the booming trends in computing industry. Our purpose is to make an android based application that would be an efficient app for smart Phone and also an entertaining app for user. I have implemented a mobile phone based version of the game. In this report, we will discuss about the game, the motivation for our game, its gameplay and implementation. We conclude with a description of the contribution of the game, as well as future work which can be done to take this version forward.

1.2 Motivations

The main motivation of this project is to explore the concepts of mobile application development in android. Game applications are the most liked and downloaded applications from android market. A game application that is simple and easy to be used in a mobile handset can become a hit with millions of mobile users. The motive of this application is to learn and experience developing a simple game application in android targeting a small set of users. The knowledge obtained in this process can be applied later in the career to develop any similar kind of application focusing large group of users.

1.3 Objectives

The main objective is to develop a game which can be run any android platform device. For developing this game we will be able to:

- Build our own Android application.
- Explain the differences between Android and other mobile development environment.
- Understand how Android applications work, their life cycle, manifest, intents, and using external resources.
- Design and develop useful Android applications with compelling user interfaces by using, extending, and creating our own layouts and Views.
- Discuss the process how to create an android operating system supporting game.
- Develop a user friendly game which will entertain user.

1.4 Contribution

To develop the game all the requirement are collected. Software, hardware and game framework requirements are also discussed in the chapter to initialize an android development environment. After that move into our project to discuss about it in details.

- We collected the necessary information about Android.
- We learned Android programming technique.
- We also collected requirements for our project.
- We tested our application and it passed in all the method we applied.

1.5 Organization of the Project Report

Chapter 2

In this chapter we will discuss about the basic information, structure, history and foundation of the Android platform. It also describes all the related literature and previous work we used in our application to develop our android application.

Chapter 3

This chapter describes the implementation procedure for example setting up android development environment, installing android development tools (ADT). We have used Eclipse IDE to create our application's internal coding and xml based user interface**.** Here also describes the entire game like how we create the game, useful description of elements that the game contains how to run this application and also testing techniques of application.

Chapter 4

The Chapter 4 Describes conclusion and future work of our application.

Chapter 2

Literature Review about Android

2.1 What is Android?

Android is the name of the mobile operating system made by American company Google. It most commonly comes installed on a variety of smartphones and tablets. This means you can easily look for information on the web, watch videos, search for directions and write emails on your phone, just as you would on your computer, but there's more to Android than these simple examples. Android phones are highly customizable and as such can be altered to suit your tastes and needs with wallpapers, themes and launchers which completely change the look of your device's interface.[1] You can download applications to do all sorts of things like check your Facebook and Twitter feeds, manage your bank account, order pizza and play games. You can plan events on from your phone's calendar and see them on your computer or browse websites on your desktop and pick them up on your phone.

Android is designed primarily for touchscreen mobile devices such as smartphones and tablet computers. Despite being primarily designed for touchscreen input, it also has been used in game consoles, digital cameras, and other electronics. As of 2011, Android has the largest installed base of any mobile OS and, as of 2013, Android devices also sell more than Windows, iOS, and Mac OS devices combined. As of July 2013 the Google Play store has had over 1 million Android apps published, and over 50 billion apps downloaded. A developer survey conducted in April–May 2013 found that 71% of mobile developers develop for Android. The Android Open Source Project (AOSP) is tasked with the maintenance and further development of Android. Google released most of the Android code under the Apache License, a free software license.

Android consists of a kernel based on the Linux kernel, with middleware, libraries and APIs written in C and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android has a large community of developers writing applications that extend the functionality of the devices. Developers write primarily in a customized version of Java. There are currently more than 520,000 apps available for Android. Apps can be downloaded from third-party sites or through online stores such as Android Market, the app store run by Google.

2.2 History of Android

2.2.1 Foundation

Android, Inc. was founded in Palo Alto, California, United States in October 2003 by Andy Rubin (co-founder of Danger), Rich Miner (co-founder of Wildfire Communications, Inc.), Nick Sears (once VP at T-Mobile), and Chris White (headed design and interface development at WebTV). The early intentions of the company were to develop an advanced operating system for digital cameras, when it was realized that the market for the devices was not large enough, and diverted their efforts to producing a smartphone operating system to rival those of Symbian and Windows Mobile. Android Inc. operated secretly, revealing only that it was working on software for mobile phones. That same year, Rubin ran out of money. Steve Perlman, a close friend of Rubin, brought him $10,000 in cash in an envelope and refused a stake in the company.

2.2.2 Acquisition by Google

Google acquired Android Inc. on August 17, 2005 key employees of Android Inc. including Rubin, Miner, and White, stayed at the company after the acquisition. Not much was known about Android Inc. at the time, but many assumed that Google was planning to enter the mobile phone market with this move[2]. At Google, the team led by Rubin developed a mobile device platform powered by the Linux kernel. Google marketed the platform to handset makers and carriers on the promise of providing a flexible, upgradable system.

2.2.3. Post-acquisition by Google

Google marketed the platform to handset makers and carriers on the promise upgradable system. Google had lined up a series of hardware component and software partners and signaled to carriers that it was open to various degrees of cooperation on their part. Google's intention to enter the mobile communications market continued to build through December 2006. The earlier prototype codenamed "Sooner" had a closer resemblance to

a BlackBerry phone with no touchscreen, and a physical. Google wanted its search and applications on mobile phones and it was working hard to deliver that. Print and online media outlets soon reported rumors that Google was developing a Google-branded handset. Some speculated that as Google was defining technical specifications, it was showing prototypes to cell phone manufacturers and network operators. A news report InformationWeek covered a serve study reporting that Google had filed several patent applications in the area of mobile telephony.

2.2.4 Open Handset Alliance

The first commercially available smartphone running Android was the HTC Dream, released on October 22, 2008. But before one year on November 5, 2007, the Open Handset Alliance, a consortium of several companies which include Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile and Texas Instruments unveiled itself. The goal of the Open Handset Alliance is to develop open standards for mobile devices. On the same day, the Open Handset Alliance also unveiled their first product, Android, a mobile device platform built on the Linux kernel version 2.6.

That day, Android was unveiled as its first product, a mobile device platform built on the Linux kernel version 2.6.25. On December 9, 2008, 14 new members joined, including ARM Holdings, Atheros Communications, Asustek Computer Inc, Garmin Ltd, Huawei Technologies, Packet Video, Softbank, Sony Ericsson, Toshiba Corp, and Vodafone Group Plc.

2.2.5 Android Open Source Project

Android is an open-source software stack for a wide range of mobile devices and a corresponding open-source project led by Google. Here you can find the information and source code you need to learn more about the Android platform. From there you can create custom variants of the Android software stack, port devices and accessories to the Android platform, and ensure your devices are compatible with the Android compatibility definition [3]. The goal of the Android Open Source Project is to create a successful real-world product that improves the mobile experience for end users. AOSP also maintains the Android Compatibility Program, defining an Android compatible device as one that can run any application written by third-party developers using the Android SDK. The compatibility program is also optional and free of charge, with the Compatibility Test Suite also free and open-source

2.2.6 Version history

The version history of the Android mobile operating system began with the release of the Android beta in November 2007. The first commercial version, Android 1.0, was released in September 2008. Android is under ongoing development by Google and the Open Handset Alliance (OHA), and has seen a number of updates to its base operating system since its initial release[4].

List of Android version names:

- Alpha (1.0)
- Beta (1.1)
- Cupcake (1.5)
- Donut (1.6)
- Eclair (2.0–2.1)
- Froyo (2.2–2.2.3)
- Gingerbread (2.3–2.3.7)
- Honeycomb (3.0–3.2.6)
- Ice Cream Sandwich (4.0–4.0.4)
- Jelly Bean (4.1–4.3.1)
- KitKat (4.4–4.4.4)
- Lollipop (5.0-5.1.1)

Android 5.1-Lollipop

This version was released on March 9, 2015.Some new feature has been added for example high definition voice calls, ability to join Wi-Fi network, support for multiple sim card etc.

Android 5.1.1-Lollipop

This version was released on April 21 , 2015.Features of this version are various other bug fixes and Fixes a memory leak issue introduced with the Android 5.0 "Lollipop" release.

Android 4.4 - KitKat

Google announced that the next version of Android would be named for their favorite confectioneries — Kit Kat bars — on September 3, 2013. I'am not yet sure what manner of goodies i'll find in the next version of Android, because Google has been understandably with details [5].

Android 4.1-4.3 - Jelly Bean

Jelly Bean arrived at Google IO 2012, with the release of the ASUS Nexus 7, followed by a quick update for unlocked Galaxy Nexus phones. Later in the year, the release of the Nexus 10 and Nexus 4 updated things from 4.1 to 4.2 and on to 4.3, but the version remained Jelly Bean. The release polished the UI designstarted in Ice Cream Sandwich, and brought several great new features to the table.

Android 4.0 - Ice Cream Sandwich

The follow-up to Honeycomb was announced at Google IO in May 2011 and released in December 2011. Dubbed Ice Cream Sandwich and finally designated Android 4.0, Ice Cream Sandwich brings many of the design elements of Honeycomb to smartphones, while refining the Honeycomb experience.

Android 3.X - Honeycomb

Android 3.0 came out in February 2011 with the Motorola Xoom. It's the first version of Android specifically made for tablets, and brings a lot of new UI elements to the table. Things like a new System bar at the bottom of the screen to replace the Status bar I see on phones, and a new recent applications button are a great addition for the screen real estate offered by Android tablets.

Android 2.3 - Gingerbread

Android 2.3 came out of the oven in December 2010, and like Eclair, has a new "Google phone" to go along with the Nexus S. Gingerbread brings a few UI enhancements to Android, things like a more consistent feel across menus and dialogs, and a new black notification bar, but still looks and feels like the Android I're used to, with the addition of a slew of new language support.

Android 2.2 - Froyo

Android 2.2 was announced in May 2010 at the Google IO conference in San Francisco. The single largest change was the introduction of the Just-In-Time Compiler or JIT which significantly speeds up the phone's processing power. Along with the JIT, Android 2.2 also brings support for Adobe Flash 10.1. That means you can play your favorite Flash-based games in Android's web browser. Take that, iPhone!

2.3 Design and Architecture of Android

Android consists of a kernel based on the Linux kernel, with middleware, libraries and APIs written in C, C++ and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android uses the Dalvik virtual machine with just-in-time compilation to run Dalvikdex-code which is usually translated from Java byte code. The main hardware platform for Android is the ARM architecture. There is support for x86 from the Android x 86 projects and Google TV uses a special x86 version of Android.

2.3.1 Linux kernel

Android may be based on Linux, but it's not based on the type of Linux system you may have used on your PC. You can't run Android apps on typical Linux distributions and you can't run the Linux programs you're familiar with on Android. Linux makes up the core part of Android, but Google hasn't added all the typical software and libraries you'd find on a Linux distribution like Ubuntu. This makes all the difference.

Android's Linux kernel has further architectural changes that are implemented by Google outside the typical Linux kernel development cycle. Android's kernel is based on the Linux kernel and has further architecture changes by Google outside the typical Linux kernel development cycle. Android does not have a native X Window System nor does it support the full set of standard GNU libraries, and this makes it difficult to port existing Linux applications or libraries to Android. Certain features that Google contributed back to the Linux kernel, notably a power management feature called wake locks, were rejected by mainline kernel developers, partly because kernel maintainers felt that Google did not show any intent to maintain their own code. Even though Google announced in April 2010 that they would hire two employees to work with the Linux kernel community, Greg Kroah-Hartman, the current Linux kernel maintainer for the -stable branch, said in December 2010 that he was concerned that Google was no longer trying to get their code changes included in mainstream Linux. Some Google Android developers hinted that "the Android team was getting fed up with the process", because they were a small team and had more urgent work to do on Android. However, in September 2010, Linux kernel developer Rafael J. Wysocki added a patch that improved the mainline Linux wakeup events framework. He said that Android device drivers that use wake locks can now be easily merged into mainline Linux, but that Android's opportunistic suspend features should not be included in the mainline kernel. In August 2011, Linus Torvalds said that "eventually Android and Linux would come back to a common kernel, but it will probably not be for four to five years"[6].

2.3.2 Architecture of Android

Rather than running typical Linux applications, Android uses the Dalvik virtual machine to essentially run applications written in Java. These applications are targeted at Android devices and the application programming interfaces (APIs) Android provides rather than being targeted at Linux in general.

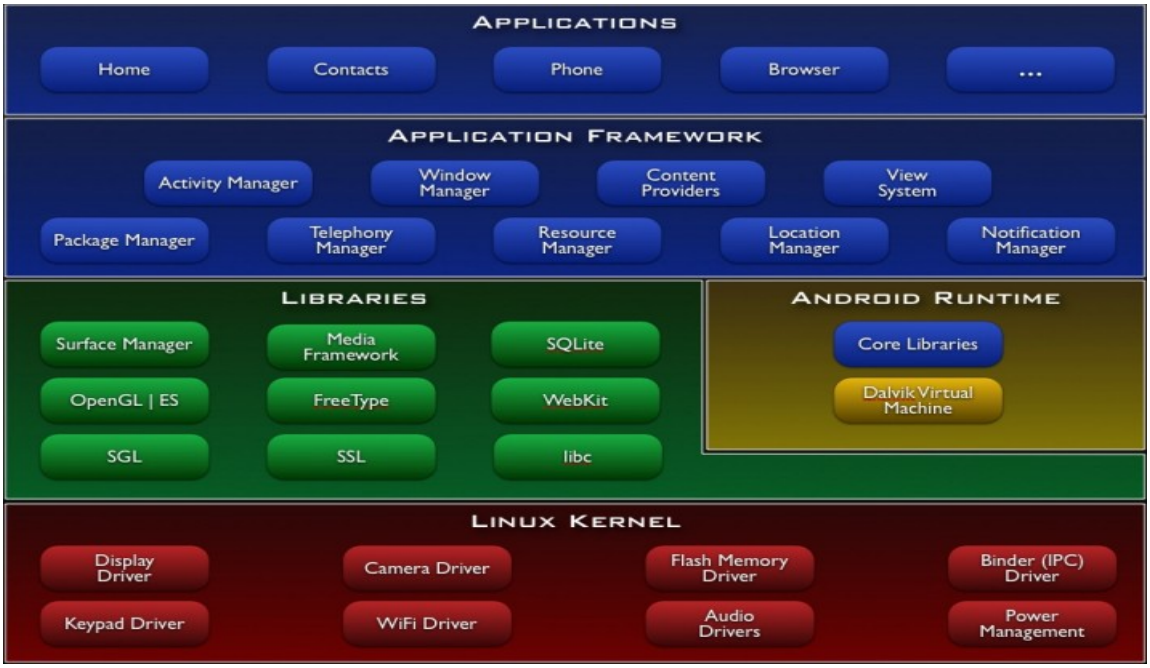

Figure 1: Android Architecture

2.3.3 Application

Android offers developers the ability to build extremely rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, and much, much more. Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language. Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities

and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

Underlying all applications is a set of services and systems, including:

- A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser.
- Content Providers that enable applications to access data from other applications or to share their own data.
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files.
- A Notification Manager that enables all applications to display custom alerts in the status bar.
- An Activity Manager that manages the lifecycle of applications and provides a common navigation back stack.

2.3.4 Libraries

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:
System C library – a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices.

Media Libraries – based on Packet Video's Open CORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG.

Surface Manager – manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications.

Lib Web Core – a modern web browser engine which powers both the Android browser and an embeddable web view.

SGL – the underlying 2D graphics engine.

3D libraries – an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer.

Free Type – bitmap and vector font rendering.

SQLite – a powerful and lightweight relational database engine available to all applications.

2.3.5 Features of Android

This is a list of features in the Android operating system

Screen capture

The android supports capturing a screenshot by pressing the power and volume-down buttons at the same time. Prior to Android 4.0, the only methods of capturing a screenshot were through manufacturer and third-party customizations or otherwise by using a PC connection (DDMS developer's tool). These alternative methods are still available with the latest Android [7].

Video calling

Android does not support native video calling, but some handsets have a customized version of the operating system that supports it, either via the UMTS network or over IP. Video calling through Google Talk is available in Android 2.3.4 and later. Gingerbread allows Nexus S to place Internet calls with a SIP account. This allows for enhanced VoIP dialing to other SIP accounts and even phone numbers. Skype 2.1 offers video calling in Android 2.3, including front camera support. Users with the Google+ Android app can video chat with other google + users through hangouts.

Voice based features

Google search through voice has been available since initial release. Voice actions for calling, texting, navigation, etc. are supported on Android 2.2 onwards. As of Android 4.1, Google has expanded Vce Actions with ability to talk back and read answers from Google's Knowledge Graph when queried with specific commands. The ability to control hardware has not yet been implemented.

Multi-touch

Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero. The feature was originally disabled at the kernel level possibly to avoid infringing Apple's patents on touch-screen technology at the time. Google has since released an update for the Nexus One and the Motorola Droid which enables multi-touch natively.

Messaging

SMS and MMS are available forms of messaging, including threaded text messaging and Android Cloud To Device Messaging and now enhanced version of C2DM, Android Google Cloud Messaging is also a part of Android Push Messaging service.

Web browser

The web browser available in Android is based on the open-source Blink (previously WebKit) layout engine, coupled with Chrome's V8 JavaScript engine. The browser scores 100/100 on the Acid3 test on Android 4.0.

Java support

While most Android applications are written in Java, there is no Java Virtual Machine in the platform and Java byte code is not executed. Java classes are compiled into Dalvik executable and run on Dalvik, a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU. J2ME support can be provided via third-party applications.

Media support

Android supports the following audio, video, still media formats- Web M, H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, OggVorbis, FLAC, WAV, JPEG, PNG, GIF, BMP.

Streaming media support

RTP/RTSP streaming (3GPP PSS, ISMA), HTML progressive download (HTML5 <video> tag). Adobe Flash Streaming (RTMP) and HTTP Dynamic streaming are supported by the Flash plug-in. Apple HTTP Live Streaming is supported by RealPlayer for Mobile, and by the operating system in Android 3.0 (Honeycomb).

Connectivity

Android supports connectivity technologies including GSM/EDGE, Wi Fi, Bluetooth, LTE, CDMA, EV-DO, UMTS, NFC, IDEN and WiMAX.

Bluetooth

Supports voice dialing and sending contacts between phones, sending files (OPP), accessing the phone book (PBAP), A2DP and AVRCP. Keyboard, mouse and joystick (HID) support is available in Android 3.1+, and in earlier versions through manufacturer customizations and third-party applications.

Tethering

Android supports tethering, which allows a phone to be used as a wireless/wired Wi-Fi hotspot. Before Android 2.2 this was supported by third-party applications or manufacturer customizations.

2.3.6 Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently.The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool. The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

2.4 Applications

Android has a growing selection of third party applications, which can be acquired by users either through an app store such as Google Play or the Amazon App store, or by downloading and installing the application's APK file from a third-party site. Applications are usually developed in the Java language using the Android Software Development Kit, but other development tools are available, including a Native Development Kit for applications or extensions in C or C++, Google App Inventor, a visual environment for novice programmers and various cross platform mobile web applications frameworks. The client application filters the list of available applications down to those compatible with the user's device, and developers may restrict their applications to particular carriers or countries for business reasons.

2.4.1 Security of Application

The applications run in a sandbox, an isolated area of the system that does not have access to the rest of the system's resources, unless access permissions are explicitly granted by the user when the application is installed. A game may need to enable vibration, for example, but should not need to read messages or access the phonebook. After

reviewing these permissions, the user can decide whether to install the application. The sandboxing and permissions system weakens the impact of vulnerabilities and bugs in applications, but developer confusion and limited documentation has resulted in applications routinely requesting unnecessary permissions, reducing its effectiveness. The complexity of inter-application communication implies Android may have opportunities to run unauthorized code. Before installing an application, the Play Store displays all required permissions.

Google has now pushed an update to Android Verify Apps feature, which will now run in background to detect malicious processes and crack them down. Several security firms have released antivirus software for Android devices, in particular, Lookout Mobile Security, AVG Technologies, Avast!, F-Secure, Kaspersky, McAfee and Symantec. This software is ineffective as sandboxing also applies to such applications, limiting their ability to scan the deeper system for threats. Some type of security applications program and service, often described as Find My Phone, is available for Android, also for Microsoft Windows Phone and for Apple iPhone.

2.4.2 Privacy

The "App Ops" privacy and application permissions control system, used for internal development and testing by Google, was introduced in Google's Android 4.3 release for the Nexus devices. Initially hidden, the feature was discovered publicly. It allowed users to install a management application and approve or deny permission requests individually for each of the applications installed on a device. Access to the App Ops was later restricted by Google starting with Android 4.4.2 with an explanation that the feature was accidentally enabled and not intended for end-users for such a decision. Android smart phones have the ability to report the location of Wi-Fi access points, encountered as phone users move around, to build databases containing the physical locations of hundreds of millions of such access points. These databases form electronic maps to locate smart phones, allowing them to run apps like Foursquare, Latitude, Places, and to deliver location-based ads.

Third party monitoring software such as Taint Droid, an academic research-funded project in some cases, detect when personal information is being sent from applications to remote servers. In March 2012 it was revealed that Android Apps can copy photos without explicit user permission, Google responded they "originally designed the Android photos file system similar to those of other computing platforms like Windows and Mac OS I ‘am taking another look at this and considering adding permission for apps to access images. I’ve always had policies in place to remove any apps [on Google Play store] that improperly access your data."

2.4.3 Google Play

Google Play is an online software store developed by Google for Android devices. An application program called "Play Store" is preinstalled on most Android devices and allows users to browse and download apps published by third-party developers, hosted on Google Play. As of June 2012, there were more than 600,000 apps available for Android, and the estimated number of applications downloaded from the Play Store exceeded 20 billion. The operating system itself is installed on 400 million total devices.

Only devices that comply with Google's compatibility requirements are allowed to preinstall and access the Play Store. The app filters the list of available applications to those that are compatible with the user's device, and developers may restrict their applications to particular carriers or countries for business reasons.

Google offers many free applications in the Play Store including Google Voice, Google Goggles, Gesture Search, Google Translate, Google Shopper, Listen and My Tracks. In August 2010, Google launched "Voice Actions for Android", which allows users to search, write messages, and initiate calls by voice.

2.5 Software Development Tools

2.5.1 Android SDK

The Android SDK provides the API libraries and developer tools necessary to build, test, and debug apps for Android. Android software development is the process by which new applications are created for the Android operating system. Applications are usually developed in the Java programming language using the Android Software Development Kit, but other development tools are available. The Android software development kit (SDK) includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.5.8 or later, Windows XP or later. The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) Plug-in, though developers may use any text editor to edit Java and XML files then use command line tools (Java Development Kit and Apache Ant are required) to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely).
Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices.

Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing. Android applications are packaged in .apk format and stored under /data/app folder on the Android OS (the folder is accessible only to root user for security reasons). APK package contains .dexfiles (compiled byte code files called Dalvikexecutables), resource files, etc.

2.5.2 App Inventor for Android

App Inventor for Android is an open-source web application originally provided by Google, and now maintained by the Massachusetts Institute of Technology (MIT). On 12 July 2010, Google announced the availability of App Inventor for Android, a Web-based visual development environment for novice programmers, based on MIT's Open Blocks Java library and providing access to Android devices' GPS, accelerometer and orientation data, phone functions, text messaging, speech-to-text conversion, contact data, persistent storage, and Web services, initially including Amazon and Twitter. "I could only have done this because Android's architecture is so open," said the project director, MIT's Hal Abelson. Under development for over a year, the block-editing tool has been taught to non-majors in computer science at Harvard, MIT, Wellesley, Trinity College (Hartford,) and the University of San Francisco, where Professor David Wolber developed an introductory computer science course and tutorial book for non-computer science students based on App Inventor for Android.

2.5.3HyperNext Android Creator

HyperNext is a free visual software development system aimed at beginner programmers that runs on Macintosh and Windows computers. It was inspired by HyperCard and includes a GUI having controls such as buttons and listboxes, and an interpreted English-like programming language. HyperNext also includes a high-level object-oriented compiled BASIC. The HyperNext Studio package comprises three complementary applications that can help users create and run software under Windows and Mac OS Xand Mac OS 9 platforms. HyperNext Android Creator (HAC) is a software development system aimed at beginner programmers that can help them create their own Android apps without knowing Java and the Android SDK. It is based on HyperCard that treated software as a stack of cards with only one card being visible at any one time and so is well suited to mobile phone applications that have only one window visible at a time. HyperNext Android Creator's main programming language is simply called HyperNext and is loosely based on HyperCard's Hyper Talk language. HyperNext is an interpreted English-like language and has many features that allow creation of Android applications. It supports a growing subset of the Android SDK including its own versions of the GUI control types and automatically runs its own background service so apps can continue to run and process information while in the background.

2.5.4 Native Development Kit

The NDK is a toolset that allows you to implement parts of your app using native-code languages such as C and C++. For certain types of apps, this can be helpful so you can reuse existing code libraries written in these languages, but most apps do not need the Android NDK [8]. Libraries written in C and other languages can be compiled to ARM or x86 native code and installed using the Android Native Development Kit. Native classes can be called from Java code running under the Dalvik VM using the System. Load Library call, which is part of the standard Android Java classes [8].

Complete applications can be compiled and installed using traditional development tools. The ADB debugger gives a root shell under the Android Emulator which allows native ARM code or x 86 codes to be uploaded and executed. ARM or x 86 codes can be compiled using GCC on a standard PC. Running native code is complicated by the fact that Android uses a non-standard C library (libc, known as Bionic). The underlying graphics device is available as a frame buffer at /dev/graphics/fb0. The graphics library that Android uses to arbitrate and control access to this device is called the Skia Graphics Library (SGL), and it has been released under an open source license. Skia has backend for both win32 and UNIX, allowing the development of cross-platform applications, and it is the graphics engine underlying the Google Chrome web browser. Unlike Java App development based on the Eclipse IDE, the NDK is based on command-line tools and requires invoking them manually to build, deploy and debug the apps. Several third-party tools allow integrating the NDK into Eclipse and Visual Studio.

2.5.5 Android Open Accessory Development Kit

The Accessory Development Kit (ADK) is a reference implementation for hardware manufacturers and hobbyists to use as a starting point for building accessories for Android. Each ADK release is provided with source code and hardware specifications to make the process of developing your own accessories easier. Creating new and alternative hardware based on the ADK is encouraged. The Android 3.1 platform (also back ported to Android 2.3.4) introduces Android Open Accessory support, which allows external USB hardware (an Android USB accessory) to interact with an Android-powered device in a special "accessory" mode. When an Android-powered device is in accessory mode, the connected accessory acts as the USB host (powers the bus and enumerates devices) and the Android-powered device acts as the USB device. Android USB accessories are specifically designed to attach to Android-powered devices and adhere to a simple protocol (Android accessory protocol) that allows them to detect Android-powered devices that support accessory mode.

2.5.6 The Simple Project

The goal of Simple is to bring an easy-to-learn-and-use language to the Android platform. Simple is a BASIC dialect for developing Android applications. It targets professional and non-professional programmers alike in that it allows programmers to quickly write Android applications that use the Android runtime components. Similar to Microsoft Visual Basic 6, Simple programs are form definitions (which contain components) and code (which contains the program logic). The interaction between the components and the program logic happens through events triggered by the components. The program logic consists of event handlers which contain code reacting to the events. The Simple project is not very active, the last source code update being in August 2009.

2.6 App Components

2.6.1 Application Fundamentals

Android applications are written in the Java programming language. The Android SDK tools compile the code along with any data and resource files into an Android package, an archive file with an .apk suffix. All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application. Once installed on a device, each Android application lives in its own security sandbox:

- The Android operating system is a multi-user Linux system in which each application is a different user.
- By default, the system assigns each application a unique Linux user ID (the ID is used only by the system and is unknown to the application). The system sets permissions for all the files in an application so that only the user ID assigned to that application can access them.
- Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications.
- By default, every application runs in its own Linux process. Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other applications.

In this way, the Android system implements the principle of least privilege. That is, each application, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in

which an application cannot access parts of the system for which it is not given permission. However, there are ways for an application to share data with other applications and for an application to access system services:

- It's possible to arrange for two applications to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, applications with the same user ID can also arrange to run in the same Linux process and share the same VM (the applications must also be signed with the same certificate).
- An application can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.

That covers the basics regarding how an Android application exists within the system. The rest of this document introduces you to:

- The core framework components that define your application.
- The manifest file in which you declare components and required device features for your application.
- Resources that are separate from the application code and allow your application to gracefully optimize its behavior for a variety of device configurations.

2.6.2 Application Components

Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role each one is a unique building block that helps define your application's overall behavior. There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.
Here are the four types of application components:

Activities

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any one of these activities. For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture. An activity is implemented as a subclass of Activity and you can learn more about it in the Activities developer guide.

Services

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. A service is implemented as a subclass of Service and you can learn more about it in the Services developer guide.

Content providers

A content provider manages a shared set of application data. You can store the data in the file system, a SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider to read and write information about a particular person. Content providers are also useful for reading and writing data that is private to your application and not shared. For example, the Note Pad sample application uses a content provider to save notes. A content provider is implemented as a subclass of Content Provider and must implement a standard set of APIs that enable other applications to perform transactions. For more information, see the Content Providers developer guide.

Broadcast receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event. A

broadcast receiver is implemented as a subclass of Broadcast Receiver and each broadcast is delivered as an Intent object.

A unique aspect of the Android system design is that any application can start another application's component. For example, if you want the user to capture a photo with the device camera, there's probably another application that does that and your application can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera application. Instead, you can simply start the activity in the camera application that captures a photo. When complete, the photo is even returned to your application so you can use it. To the user, it seems as if the camera is actually a part of your application. When the system starts a component, it starts the process for that application and instantiates the classes needed for the component. For example, if your application starts the activity in the camera application that captures a photo, that activity runs in the process that belongs to the camera application, not in your application's process. Because the system runs each application in a separate process with file permissions that restrict access to other applications, your application cannot directly activate a component from another application. The Android system, however, can. So, to activate a component in another application, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

2.6.3 Activating Components

Three of the four component types' activities, services, and broadcast receivers are activated by an asynchronous message called intent. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your application or another. Intent is created with an Intent object, which defines a message to activate either a specific component or a specific type of component intent can be either explicit or implicit, respectively.

For activities and services, intent defines the action to perform and may specify the URI of the data to act on (among other things that the component being started might need to know). For example, intent might convey a request for an activity to show an image or to open a web page. In some cases, you can start an activity to receive a result, in which case, the activity also returns the result in an Intent (for example, you can issue an intent to let the user pick a personal contact and have it returned to you—the return intent includes a URI pointing to the chosen contact).
For broadcast receivers, the intent simply defines the announcement being broadcast (for example, a broadcast to indicate the device battery is low includes only a known action string that indicates "battery is low").The other component type, content provider, is not activated by intents. Rather, it is activated when targeted by a request from a Content Resolver. The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the Content

Resolver object. This leaves a layer of abstraction between the content provider and the component requesting information (for security).
There are separate methods for activating each type of component:

- You can start an activity (or give it something new to do) by passing an Intent to start Activity() or start Activity For Result() (when you want the activity to return a result).
- You can start a service (or give new instructions to an ongoing service) by passing an Intent to start Service(). Or you can bind to the service by passing an Intent to bind Service().
- You can initiate a broadcast by passing an Intent to methods like send Broadcast(), send Ordered Broadcast(), or send Sticky Broadcast().
- You can perform a query to a content provider by calling query() on a Content Resolver.

For more information about using intents, see the Intents and Intent Filters document. More information about activating specific components is also provided in the following documents: Activities, Services, Broadcast Receiver and Content Providers.

2.6.4 The Manifest File

Before the Android system can start an application component, the system must know that the component exists by reading the application's AndroidManifest.xml file (the "manifest" file). Your application must declare all its components in this file, which must be at the root of the application project directory.
The manifest does a number of things in addition to declaring the application's components, such as:

- Identify any user permissions the application requires, such as Internet access or read-access to the user's contacts.
- Declare the minimum API Level required by the application, based on which APIs the application uses.
- Declare hardware and software features used or required by the application, such as a camera, Bluetooth services, or a multi touch screen.
- API libraries the application needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

2.6.5 Declaring components

The primary task of the manifest is to inform the system about the application's components. For example, a manifest file can declare an activity as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
<applicationandroid:icon="@drawable/app_icon.png" ... >
<activityandroid:name="com.example.project.ExampleActivity"
android:label="@string/example_label" ... >
</activity>
...
</application>
</manifest>
```

In the <application> element, the android: icon attribute points to resources for an icon that identifies the application. In the <activity> element, the android: name attribute specifies the fully qualified class name of the Activity subclass and the android: label attributes specifies a string to use as the user-visible label for the activity.

You must declare all application components this way:

- <activity> elements for activities
- <service> elements for services
- <receiver> elements for broadcast receivers
- <provider> elements for content providers

Activities, services, and content providers that you include in your source but do not declare in the manifest are not visible to the system and, consequently, can never run. However, broadcast receivers can be either declared in the manifest or created dynamically in code (as Broadcast Receiver objects) and registered with the system by calling register Receiver().

2.6.6 Declaring components capabilities

As discussed above, in Activating Components, you can use Intent to start activities, services, and broadcast receivers. You can do so by explicitly naming the target component (using the component class name) in the intent. However, the real power of intents lies in the concept of intent actions. With intent actions, you simply describe the type of action you want to perform (and optionally, the data upon which you'd like to perform the action) and allow the system to find a component on the device that can perform the action and start it. If there are multiple components that can perform the action described by the intent, then the user selects which one to use. The way the

system identifies the components that can respond to intent is by comparing the intent received to the intent filters provided in the manifest file of other applications on the device.

When you declare a component in your application's manifest, you can optionally include intent filters that declare the capabilities of the component so it can respond to intents from other applications. You can declare an intent filter for your component by adding an <intent-filter> element as a child of the component's declaration element. For example, an email application with an activity for composing a new email might declare an intent filter in its manifest entry to respond to "send" intents (in order to send email). An activity in your application can then create an intent with the “send” action (ACTION_SEND), which the system matches to the email application’s “send” activity and launches it when you invoke the intent with start Activity().

2.6.7 Declaring application requirements

There are a variety of devices powered by Android and not all of them provide the same features and capabilities. In order to prevent your application from being installed on devices that lack features needed by your application, it's important that you clearly define a profile for the types of devices your application supports by declaring device and software requirements in your manifest file. Most of these declarations are informational only and the system does not read them, but external services such as Google Play do read them in order to provide filtering for users when they search for applications from their device.

For example, if your application requires a camera and uses APIs introduced in Android 2.1 (API Level 7), you should declare these as requirements in your manifest file. That way, devices that do not have a camera and have an Android version lower than 2.1 cannot install your application from Google Play. However, you can also declare that your application uses the camera, but does not require it. In that case, your application must perform a check at runtime to determine if the device has a camera and disable any features that use the camera if one is not available. Here are some of the important device characteristics that you should consider as you design and develop your application:

Screen size and density

In order to categorize devices by their screen type, Android defines two characteristics for each device: screen size (the physical dimensions of the screen) and screen density (the physical density of the pixels on the screen, or dpi—dots per inch). To simplify all the different types of screen configurations, the Android system generalizes them into select groups that make them easier to target. The screen sizes are: small, normal, large, and extra large. The screen

densities are: low density, medium density, high density, and extra high density. By default, your application is compatible with all screen sizes and densities, because the Android system makes the appropriate adjustments to your UI layout and image resources.

Input configurations

Many devices provide a different type of user input mechanism, such as a hardware keyboard, a trackball, or a five-way navigation pad. If your application requires a particular kind of input hardware, then you should declare it in your manifest with the <uses-configuration> element. However, it is rare that an application should require a certain input configuration.

Device features

There are many hardware and software features that may or may not exist on a given Android-powered device, such as a camera, a light sensor, Bluetooth, a certain version of OpenGL, or the fidelity of the touch screen. You should never assume that a certain feature is available on all Android-powered devices (other than the availability of the standard Android library), so you should declare any features used by your application with the <uses-feature> element.

Platform Version

Different Android-powered devices often run different versions of the Android platform, such as Android 1.6 or Android 2.3. Each successive version often includes additional APIs not available in the previous version. In order to indicate which set of APIs are available, each platform version specifies an API Level (for example, Android 1.0 is API Level 1 and Android 2.3 is API Level 9). If you use any APIs that were added to the platform after version 1.0, you should declare the minimum API Level in which those APIs were introduced using the <uses-sdk> element. It's important that you declare all such requirements for your application, because, when you distribute your application on Google Play, the store uses these declarations to filter which applications are available on each device. As such, your application should be available only to devices that meet all your application requirements.

2.6.8 Application Resources

An Android application is composed of more than just code it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the application. For example, you should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using application resources makes it easy to update various characteristics of your application without modifying code and

by providing sets of alternative resources enables you to optimize your application for a variety of device configurations (such as different languages and screen sizes).

For every resource that you include in your Android project, the SDK build tools define a unique integer ID, which you can use to reference the resource from your application code or from other resources defined in XML. For example, if your application contains an image file named logo.png (saved in the res/drawable/ directory), the SDK tools generate a resource ID named R.drawable.logo, which you can use to reference the image and insert it in your user interface.

One of the most important aspects of providing resources separate from your source code is the ability for you to provide alternative resources for different device configurations. For example, by defining UI strings in XML, you can translate the strings into other languages and save those strings in separate files. Then, based on a language qualifier that you append to the resource directory's name (such as res/values-fr/ for French string values) and the user's language setting, the Android system applies the appropriate language strings to your UI.

Chapter 3

Implementation

3.1 Design

Our application is a game for those devices which are run by android Operating System. The full project contains:

- A game which is played human to human
- Have 5x5 dots
- Click on the line between dots (horizontally or vertically) to connect the dots
- Score when you manage to join the two dots together which form the last side of a four-sided box.
- Player 1 : Blue and Player 2: Red
- Winner will be that player who manage to fill up the maximum square box

3.2 Implementation Procedure

Creating a fair playing field for android development:

Here are some simple pre-requisites one must have to develop an android app.

Developer Requirement:

- Advanced knowledge of java.
- Basic knowledge of XML

3.2.1 Hardware Requirement*:*

Android emulator and Android tablet or android phone is needed. To develop an android application a good configuration machine is also needed. We have used a machine which is core i3 (clock speed 2.42 GHz) with 2 GB ram.

3.2.2 Android Development Environment

For developing application we had to create android development environment. Google basically supports the "Eclipse Classic" version. But there are also other IDEs. We have used Eclipse for our development. There are other IDEs like Intel lite Idea. But we choose Eclipse because we wanted such IDE in which we could write java code and at the same time using the same IDE we could work on GUI for android. Eclipse provides both of these features. So we used Eclipse for our project development. We also used libgdx framework for the game.

3.3. Android Game

3.3.1 Feature

The features of this game are:

- Multi player game(human to human)
- Difficulty level is easy
- Board size 5x5 dots
- Fixed color for two players(Player 1 is blue and player 2 is red)
- No game speed

3.3.2 How to play

- Connect any two dots together which are next to each other, either horizontally or vertically
- Touch on line between dots (horizontally or vertically) to put your line that connect the two dots

3.3.3 Scoring

- Have to manage join two dots which form the last side of the four-sided box, your color will fill the box, indicating that it belongs to you.
- If you have created a box, you will get an extra free move before the other player takes its turn.
- If you manage to create another box in any free move, then you can keep going until you are unable to create a further box
- The player who manages to create maximum box that player will win.
- If any player fails to touch in a committed place to make a score on it will be a penalty for the player and score will go up for the other one.

3.3.4 Game Screen

The very first screen views after the game has been started. Where we can see 5x5 dots.

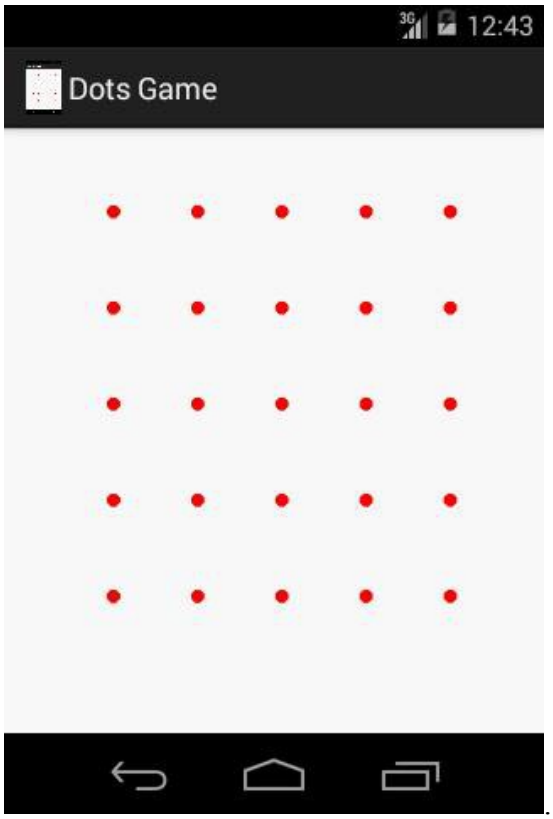

Figure 2: Game Screen

3.3.5 Connecting Dots

The two players started connecting dots with respective colors. (player 1: blue and player 2: red).

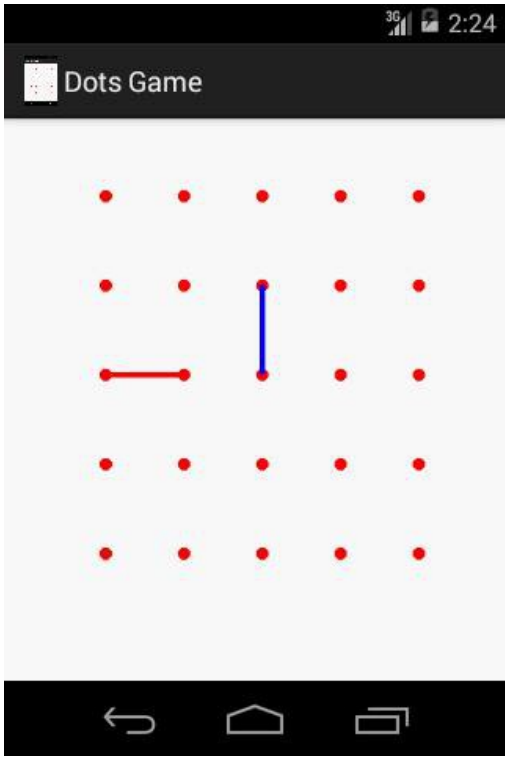

Figure 3: Connecting Dots

3.3.6 Player 2 Scores

The red box denotes player 2 owns a score because player 2 joins two dots which form the last side of the four-sided box.

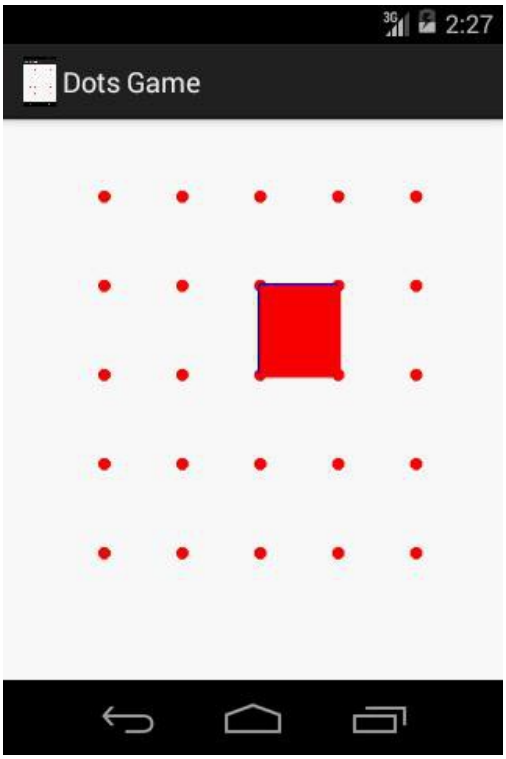

Figure 4: Player 2 Scores

3.3.7 Player 1 Scores

The blue box denotes player one owns a score. Because player 1 joins two dots which form the last side of the four-sided box.

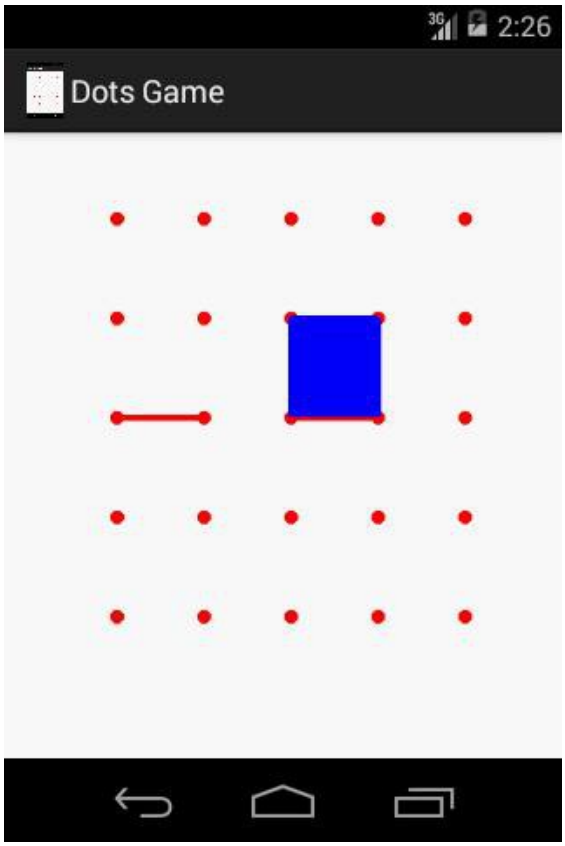

Figure 5: Player 1 Scores

3.3.8 Final Moment

The very last moment of the game where we can see one side of the four-sided box is empty and also we can see blue box is more than red box that means player 1 is going to win.

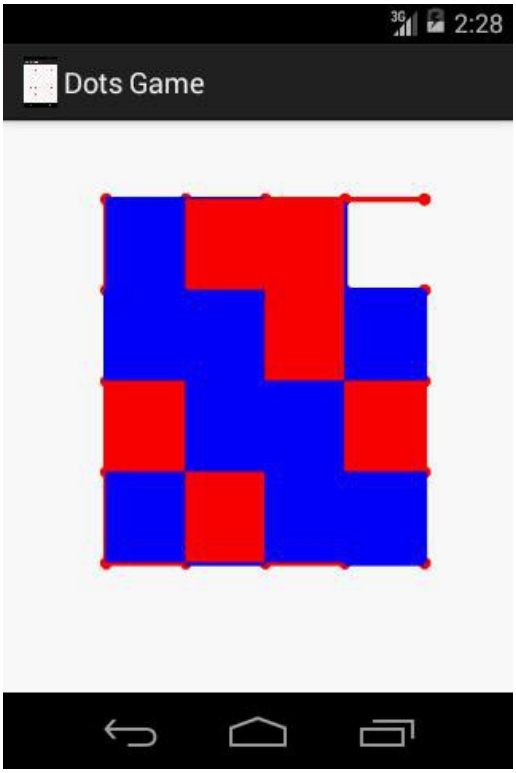

Figure 6: Final Moment

3.3.9 Game Result

After counting blue boxes if it becomes more than red ones ‘Player 1 Won’ massage will be shown and vice versa.

Figure 7: Game Result

3.3.10 Tie Moment

If both the boxes are equal then ‘Match Drawn’ massage will be shown

Figure 8: Tie Moment

3.4 Hardware Specification

We have used following devices for development, debugging and testing purpose:

- Acer Iconia
- Walton Primo GH3
- Samsung Galaxy S-3
- Symphony W91
- Nokia XL

Chapter 4

Conclusion and Future Work

4.1 Conclusion

This project successfully demonstrated a mobile based application. Android games are developed in Java, but Android is not a Complete Java implementation. Many of the packages that we have used for OpenGL and other graphic are included in the Android software development kit. Some very helpful packages for game developers, especially 3-D game developers, are not included. But most of the packages are available in Android for this game, because we developed a 2-D game. With each release of new Android SDK, more and more packages become available, and older once may be removed. We will need to be aware of which packages we have to work with.

For this project we choose Android OS because it is the most popular, user-friendly mobile operating system of the world. I think the application will be able to generate the outcome that we wanted from the very beginning of our development process. In the process of developing the application we learned many things about the android operating system and the development related works. We also learn easier and fresh ways that would help us in future for development of the game. On the other hand it will help other android developers to develop it and modify it according to their way to make this application more updated and global. We know there are some mistakes in our project. But I think it will help us to develop an application and learn a lot about android. We have made our application keeping some space for further development. But before all, we need to know how users react to our application. We are eagerly waiting for the response.

4.2 Future Work

Android is an open source operating system. So android application is built to be modified as time goes. This game is also built keeping this fact in mind. Some future modifications we want to make on this application are given below:

- We want to modify the game as the user can play with the computer.

- We want to work for the game to make it on air so that the user from all over the world could play the game easily.

- We want to add many levels in this game (Easy, Medium, and Hard. Like, board size would be 8x8 or larger etc.).

- We want to make an iOS version of this application. Because besides android, iPhone is also widely used smart phone across the world.

References

1. "What is Android"
   http://recombu.com/mobile/news/what-is-android-and-what-is-an-android-phone_M12615.html [29-03-2015]

2. "Acquisition by Google"
   http://en.wikipedia.org/wiki/Android_(operating_system)
   [05-04-2015]

3. "Android Open Source Project"
   https://source.android.com/
   [07-04-2015]

4. "Version History"
   http://en.wikipedia.org/wiki/Android_version_history
   [09-04-2015]

5. "Android 4.4-KitKat"
   http://www.androidcentral.com/android-versions
   [12-04-2015]

6. "Linux Kernel"
   http://www.howtogeek.com/189036/android-is-based-on-linux-but-what-does-that-mean/
   [13-04-2015]

7. "Screen Capture"
   http://en.wikipedia.org/wiki/List_of_features_in_Android
   [18-04-2015]

8. "Native Development Kit"
   https://developer.android.com/tools/sdk/ndk/index.html
   [25-04-2015]