School of Electronic and Computer Engineering

Technical University of Crete

Design and Implementation of a Game Theory Application Toolkit for Handheld Devices

Android Mobile Application Development

Undergraduate Diploma Thesis

Author:
Juliana Peres Hernandes Sanches

Committee:

Georgios Chalkiadakis, Assistant Professor-Supervisor

Michail G. Lagoudakis, Associate Professor

Katerina Mania, Associate Professor

Acknowledgment

The author would like to thank her supervisor Prof. Georgios Chalkiadakis for his great help with the idea of the thesis, for his continuous support and encouragement to this day.

The author wishes to thank the members of my thesis committee, Prof. Katerina Mania and Prof. Michael Lagoudakis, for their participation.

Special thanks to the people that have helped the author with their advices: To Dimitris Theodoropoulos, for his great Android Development advices. To Maria Kalenteri, for her great help. To all the generous people in The Intelligent Systems Laboratory (Students and Staff).

The author wishes to express her gratitude and love to her beloved family; for their understanding and patience, through the duration of her studies.

Table of Contents

Abstract 7

1)Introduction 10

1.1. Motivation and Goal 10

1.2 Tools and Requirements - Android Development Tool Intro 10

1.3 Our “Game Theoretic” Android Apps 13

2) Game Theory 17

2.1 Introduction 17

2.1.1 Definition 17

2.1.2 History 18

2.2 Normal Forms Games 19

2.3 Prisoner’s Dilemma 20

2.4 Battle of The Sexes (BoS) 23

3) Android Platform Fundamentals 28

3.1 Android APIs 28

3.2 Architecture 29

3.2.1 Android Layers 29

3.2.1 Android Class System - Views 31

3.3 Kernel 33

3.3.1 Kernel description 33

3.3.2 Kernel Booting 34

3.3.3 Kernel Versions 34

3.4 Android Runtime - Dalvik Virtual Machine 35

3.5 Application Framework 37

3.6 Application Layer 39

3.7 Installation and Setup 39

4) Android Application Fundamentals 45

4.1 Context 45
4.2 Components 46
4.2.1 Activity Definition 47
4.2.2 Services 48
4.2.3 Broadcast Receivers 48
4.2.4 Content Providers 50
4.2.5 Intents 53
4.3 Activities and Tasks 54
4.4 Threads 57
4.5 Google Cloud Messaging 58
5) “Confess or Not” Design 63
5.1 Description 63
5.1.1 Strategies in “Confess or Not” 64
5.2 Game Flow 69
5.3 Design 72
5.3.1 Server’s Side 74
5.3.2 Client’s Side 77
5.4 Implementation 81
6) “Lets Go Out” Design 85
6.1 Description 85
6.2 Game Flow 86
6.3 Design 88
6.3.1 Server’s Side 89
6.3.2 Client’s Side 90
6.4 Implementation 93
..... 95
7) Conclusion and future work 97
8) Related Work 101
9) Bibliography 103
Appendix 110
Structure of Web Service file 110
Structure of an example of a PHP file that applies the Webservices 112

Function *isRegistered()* 113

Table of Images

Figure 1 - “Confess or Not” payoff table 14
Figure 2 - Example of a “Let's go Out” payoff matrix 15
Figure 3 - General form of Prisoner's Dilemma payoff matrix [12] 20
Figure 4 - Example of general payoff matrix with dominant strategies marked with asterisk 21
Figure 5 - Example of Battle of The Sexes payoff matrix 23
Figure 6 - General form of payoff matrix in Battle of the Sexes [15] 23
Figure 7 - Battle of Sexes example [21] 24
Figure 8 - Battle of The Sexes example with probabilities 25
Figure 9 - Android Operating System Layers [29] 31
Figure 10 - Android Widget Framework [30] 32
Figure 11 - Example of View usage in “Confess or Not” game 32
Figure 12 - APIs and Kernel Versions [36] 34
Figure 13 - Dex file anatomy [37] 36
Figure 14 - Comparison between ART and Dalvik performance [39] 37
Figure 15 - Screenshot of Eclipse IDE with Android Developer Tools 40
Figure 16 - Screenshot of Android DDMS debugger 41
Figure 17 - XML editor built-in with graphical support 42
Figure 18 - Broadcast Receiver example indicating battery recharge [44] 50
Figure 19 - Content provider and URI relation [46] 52
Figure 20 - Content provider and URI relation [46] 52
Figure 21 - Activity Lifecycle [47] 55
Figure 22 - Activity States table [48] 56
Figure 23 - The GCM data flow 59
Figure 24 - “Confess or Not” payoff matrix 63
Figure 25 - Tit for Tat strategy possible moves 64
Figure 26 - Tit for Two Tats strategy possible moves 65
Figure 27 - Naive Peace Maker strategy possible moves 65
Figure 28 - Random strategy possible moves 66
Figure 29 - Always Defect strategy possible moves 66
Figure 30 - Always Cooperate strategy possible moves 66
Figure 31 - Grudger strategy possible moves 67
Figure 32 - Gradual strategy possible moves 67
Figure 33 - Suspicious Tit for Tat strategy possible moves 68
Figure 34 - Remorseful Prober strategy possible moves 68
Figure 35 - Screenshot of Game Request 70
Figure 37 - Settings screenshot 71
Figure 36 - Screenshot of receiving game request 71
Figure 36 - Screenshot of receiving game request 71
Figure 38 - Client-Server Interaction of Confess or Not Game of a requester 73

Figure 39 - Client-Server Interaction of Confess or Not Game of a requested player 74
Figure 40 - Interaction of files of Confess or Not for the requester 81
Figure 41 - Interaction of files of Confess or Not for the requested player................................ 82
Figure 42 - Let's go out example payoff table.. 85
Figure 43 - Screenshot of “Let's go Out” 87
Figure 44 - “Let's go Out” client server interaction 88
Figure 45 - “Let's go Out” file interaction - requester 94
Figure 46 - “Let's go Out” file interaction - requested player... 95

Design and Implementation of a Game Theory Application Toolkit for Handheld Devices

Abstract

In this thesis we developed two game-theoretic applications (applets) for Android-compatible handheld computer devices. Game Theory, a widely studied research domain, offers the formal tools for the analysis of the strategic interactions within a multiagent environment. Android is one of the most popular operating systems for handheld devices, and, with an 80% market share, it dominates the smartphones market worldwide. Moreover, it is an open platform, which means that it is constantly under improvement.

The first applet we developed is a variant of the well-known Prisoner's Dilemma game. This game contains the necessary features to play with another online player, or play against pre-devised strategies. These strategies mirror those that appear in Robert Axelrod's book "The evolution of cooperation". The second applet offers the ability to the users to invite their online friends to go out by choosing and rating the places to go. How can a user go to the place he likes but also certifies that he will have company to go out? This is interesting from a game-theoretic perspective, since all Nash equilibria of the game are problematic in some way. Moreover, the game requires player coordination, since they are aware that not choosing the same place with their opponent will result to mutual loss. Thus, it is not a foregone conclusion that a player should actually select her most preferred outcome. This applet was based on the famous "Battle of the Sexes" game.

Our work here resulted to a better understanding of important game theoretic concepts, and to the gaining of experience in developing fully-fledged Android applications. We believe that this thesis serves as a step towards popularizing game theory, and introducing it to a young generation that uses handheld computer devices and smartphones on a daily basis, for purposes ranging from recreation to business.

Σχεδίαση και Ανάπτυξη μιας Συλλογής Παιγνιοθεωρητικών Εφαρμογών για Φορητές Συσκευές

Περίληψη

Σε αυτή τη διπλωματική εργασία αναπτύχθηκαν δύο παιγνιοθεωρητικές εφαρμογές (applets) για υπολογιστικές συσκευές χειρός συμβατές με λειτουργικὀ Android. Η Θεωρία Παιγνίων, ένας ευρέως μελετώμενος ερευνητικός τομέας, προσφέρει τα τυπικά εργαλεία για την ανάλυση των στρατηγικών αλληλεπιδράσεων σε ένα περιβάλλον πολλαπλών πρακτόρων. Το Android έχει εδραιωθεί ως ένα από τα δημοφιλέστερα λειτουργικά συστήματα για συσκευές χειρός, και κυριαρχεί στην παγκόσμια αγορά των smartphones (με μερίδιο αγοράς 80%). Αποτελεί δε μια «ανοιχτή πλατφόρμα», κάτι το οποίο σημαίνει ότι υπόκειται διαρκώς σε βελτιώσεις.

Η πρώτη εφαρμογή που αναπτύχθηκε είναι μια παραλλαγή του δημοφιλούς παιχνιδιού "Prisoner's Dilemma". Το παιχνίδι περιέχει τις απαραίτητες λειτουργίες ώστε ένας χρήστης να παίξει εναντίον ενός άλλου διαδικτυακού παίκτη, ή εναντίον προσχεδιασμένων στρατηγικών που αντιστοιχούν σε στρατηγικές αυτές που παρουσιάζονται στο βιβλίο του Robert Axelrod “The evolution of cooperation”. Η δεύτερη εφαρμογή προσφέρει στους χρήστες τη δυνατότητα να προσκαλέσουν τους διαδικτυακούς φίλους τους για μια βόλτα διαλέγοντας και βαθμολογώντας τα διαθέσιμα μέρη για την έξοδό τους. Με ποιόν τρόπο μπορεί ένας χρήστης να βγει σε ένα μέρος της αρεσκείας του αλλά ταυτόχρονα να εξασφαλίσει ότι θα έχει και παρέα; Αυτό αποτελεί μια ενδιαφέρουσα περίπτωση για τη Θεωρία Παιγνίων, διότι όλα τα Nash Equilibria αυτού του ερωτήματος – το οποίο αντικατοπτρίζεται στο διάσημο παιχνίδι "Battle of the Sexes" - είναι «προβληματικά» σε κάποιο βαθμό. Επιπροσθέτως, το παιχνίδι απαιτεί το συγχρονισμό των παικτών, αφού είναι ενήμεροι ότι η μή επιλογή το ίδιου μέρους με τον αντίπαλό τους, θα οδηγήσει σε αμοιβαία ζημία. Έτσι, δεν είναι προδιαγεγραμμένο ότι ένας παίκτης θα επιλέξει την πλέον προτιμώμενη από τον ίδιο επιλογή του.

Η παρούσα εργασία μας οδήγησε σε μια βαθύτερη κατανόηση πολλών σημαντικών παιγνιοθεωρητικών εννοιών, και στην απόκτηση εμπειρίας στην ανάπτυξη ολοκληρωμένων εφαρμογών σε Android. Επιπροσθέτως, πιστεύουμε ότι αυτή η διπλωματική εργασία συμβάλει στη εκλαϊκευση της Θεωρίας Παιγνίων, καθώς την εισάγει σε μια νέα γενιά η οποία χρησιμοποιεί υπολογιστικές συσκευές χειρός και «έξυπνα τηλέφωνα» σε καθημερινή βάση, και για ένα ευρύ φάσμα δραστηριοτήτων – για την ψυχαγωγία της ως και την εργασία της.

1)Introduction

The following chapter provides an introduction to Android Tools and their basic structure. It also provides a general description of both applications presented in this document.

1.1. Motivation and Goal

In the latest years, millions of smartphones were sold. Smartphone usage is rising, and thousands of applications are being developed at this moment. [1] In 2008, Google made a great entrance in the market with Android open-source operating system. Many people use their Smartphone for gaming.

The idea of our applications was to popularize some Game Theory Games. To make those games accessible to users to play and understand the logic of theoretical games. Both games are simultaneous move games.

The first implementation is a game "Confess or Not" which is a game based on Prisoner's Dilemma game. The second implementation is a game called "Let's go out" that offers the user the ability to make an invitation to a friend to go out in the city of Chania by giving their preferences in the available places to go and ratings.

1.2 Tools and Requirements - Android Development Tool Intro

In the implementation we make use of the Android Development Tool (ADT) that is a platform to native Android Development. Android applications are written in the Java programming language. The graphical layout can be written either in Java or in XML. Android is based on a modified Linux Kernel. In a single package, the ADT Bundle includes everything required to development [2]:

- Android Platform-tools
- The latest Android platform
- The latest Android system image for the emulator
- Eclipse + ADT plugin

- Android SDK Tools (Software Development Kit)

ADT bundle requirements regarding the Operating Systems:

- Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)
- Mac OS X 10.5.8 or later (x86 only)
- Linux (tested on Ubuntu Linux, Lucid Lynx)
    - GNU C Library (glibc) 2.7 or later is required.
    - On Ubuntu Linux, version 8.04 or later is required.
    - 64-bit distributions must be capable of running 32-bit applications.

ADT bundle requirements regarding the Eclipse IDE(Integrated Development Environment) [3]:

- Eclipse 3.6.2 (Helios) or greater
- Eclipse JDT (Java Development Tools) plugin, that is included in most Eclipse IDE packages
- JDK 6 (Java Development Kit)
- Android Development Tools plugin
- **Not** compatible with GCJ (GNU Compiler for Java)

The Android SDK tools compile the code along with any data and resource files into an APK, and Android package. APK file contains all the contents of an Android App. It is the file used to install the app onto Android –powered devices. The Android project comes with a pack of files and folders necessary to development. The main folders and files in a project are:

- /src/- This folder contains the java files.
- /gen/- This folder contain the generated files such as the R file that links the java classes with XML.
- /assets/-A required folder where uncompiled file resources can be included in the project
- /bin/-Output directory of the build. Here is the final .apk file and other compiled resources.
- /libs/-Contains private libraries.
- /res/-Contains the resource files that will be explained in following paragraphs.
- default.properties—A generated build file used by Eclipse and the Android ADT plug-in.
- Android Manifest File-The main configuration file.

There is a resource file in the SDK project that is provided in the resource directory(res/). All the resources are accessed through subclasses of the R file. The R file is a system-made file that contains the addresses of each resource in memory. The resource files of an Android Project are:

- Animation Resources - Define the pre-determined animations.
- Color State List Resource- Define a color resource, it is saved in res/color/
- Drawable Resource-It stores anything that has to do with graphics such as bitmaps or XML. It is saved in /res/drawable/
- Layout Resource-Define the layout for the application UI. It is saved in res/layout/
- Menu Resource-Define tha content of application menus. It is saved in res/menu/
- String Resources-Define strings, string array. Include string formatting and styling. It is saved in res/values/
- Style Resources-Define the look and format for User Interface elements. Saved in /res/values/
- More Resources Types-Define values such as integers, Booleans , dimensions, colors .Saved in res/values/

Every application has an AndroidManifest.xml file in its root directory. This file gives important information about the application to the Android System and it consists of the central configuration file for the application. Among important information, this file names the Java packet with a unique identifier for the application. It describes all the components of the application such as activities, services, broadcast receivers and content providers. It also declares the permissions and the minimum level of Android API that the application requires. Additional information regarding the Application's components is given in chapter 4.

The SDK tool also provides a graphical interface to design the application layout. The developer can write in XML and then check the design in the Graphical Layout Section. Also the developer can drag and drop some features such as buttons, widgets and layouts.

Netbeans® [4] software was used for local server testing and implementation and Godaddy.com [5] for remote server testing. Regarding the local hosting test, XAMMP® [6] software served as the local database server while PHP [7] was the language used for coding.

1.3 Our “Game Theoretic” Android Apps

Our first implementation is a game entitled “Confess or Not”, which is a game app based on the (Iterated) Prisoner’s Dilemma (IPD) game. That means that the user can choose to confess a committed crime (defect) or not to confess (cooperate), perhaps also taking into account the past behavior of her[1] opponent. There is a random option for playing and also the user can choose to play a preselected strategy. The user can play with an auto-mated agent (i.e., against her own smartphone device), or with a friend. Though there exist many online implementations of the IPD game (for example, one can be found at [8]); up until now there was no android device app corresponding to this famous game.

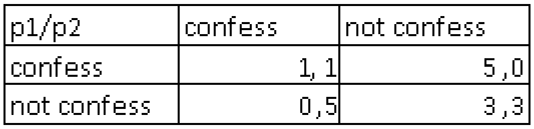
The first time a player chooses to play the game, she has to register in an online server. Online playing in the “Confess or Not” game is conducted between two registered users after a game request a player sends to another, which is received by the second player at his android device. After the notification of a game request is sent and acknowledged, the game starts. The game is played over a random number of rounds with the number lying within a pre-selected range of rounds. This range is specified by the user (who chooses among the ranges of “5-10”,“10-20”, “20-40” and “40-60”) in case that she is facing her device — or corresponds to the preset range 10 - 30 when two real users are facing each other. Note that the number of game turns “has to be” random, because, if the actual number of rounds to be played is known in advance, the rational equilibrium strategy prescribes to always defect (not confess).

The applet provides the following strategies that are popular IPD strategies that appeared in the famous Axelrod’s tournament [9], or were developed thereafter: Tit for Tat, Tit for Two Tats, Suspicious Tit for Tat, Naive Peace Maker, Random 50/50, Always Defect, Always Cooperate, Grudger, Gradual, Remorseful Prober Strategy, or Random. Tit for Tat always repeats the opponent’s last choice.

Tit for Two Tats defects if the opponent has defected two times in a row. Suspicious Tit for Tat is the same as Tit for Tat but opponent begins by playing “confess”. Naive Peace

[1] The female gender term is widely used among Game-theorists.

Maker is the same as Tit for Tat, but randomly chooses to cooperate. The Grudger strategy cooperates until the opponent defects. After this, the strategy will always defect. The Gradual strategy cooperates until the opponent defects. If the opponent defects, the strategy will defect the total number of times the opponent has defected during the game so far, and then cooperates twice. The Remorseful Prober Strategy, mimics Tit for Tat, but defects randomly. If the opponent defects after a random defection by the strategy, the strategy will cooperate once.



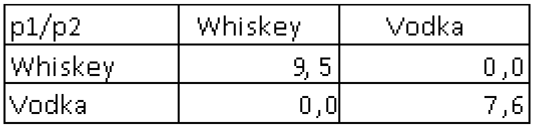
p1/p2	confess	not confess
confess	1, 1	5 ,0
not confess	0,5	3 ,3

Figure 1 - “Confess or Not” payoff table

The payoff matrix used in the application for a given stage-game is as in figure above. At the end of the IPD game, the total numbers of points accumulated by the agents over all rounds are counted. The score for each player is calculated as the number of total points scored by the player divided by the maximum possible player outcome.

Moreover, a “mutual outcome” rating corresponding to the combined scores achieved by both players as a percentage of the best possible combined score is also calculated and displayed, as done in [8].

The second game implementation, is a game called “Let’s go out” that simulates an invitation to a friend to go out tonight. Players indicate their preference ratings (in a range of 1- 10) regarding places to visit during a night out, and, if manage to coordinate on a place, they enjoy the corresponding benefits—otherwise they stay at home with a payment of zero. The game flow is as follows: at the moment user chooses and ranks, the values are stored in a server’s database. Also the values for the friend are stored after the friend makes her choices. The preferred choices of the two players are compared, and the common choices are displayed. These correspond to potential venues the friends might visit tonight—i.e., they are the options available to the players facing each other in a dynamically generated, 2-player, $n \times n$ matrix game (with n being the number of common choices between the player and his friend). The n × n table is displayed to each player. The preferences of each player are displayed in a payoff matrix with user preference ratings corresponding to player rewards for the “coordinated outcomes” matrix cells, while the appearance of zeroes in the cells corresponds to uncoordinated outcomes. For instance, say the row player tonight wants to go to a Whiskey bar with preference rating 9, a Vodka bar which she rates as a 7, or a Rum bar which she assigns 4 stars only; while the column player wants to go to either a Martini bar with preference rating 8, a Vodka bar with rating 6 or an Whiskey bar which she rates as a 5. The dynamically created game matrix will be as below:



p1/p2	Whiskey	Vodka
Whiskey	9, 5	0 ,0
Vodka	0,0	7,6

Figure 2 - Example of a “Let's go Out” payoff matrix

The game matrix is actually a generalization of the 2 × 2 matrix of a Battle of The Sexes [10] game. Battle of the Sexes is a 2-player coordination game, where players receive zero for not coordinating while having only two choices available. In our case, the number of options available to a player in the matrix game depends on the number of preferred venues the two players indicated they have actually in common (during this game’s instantiation).

Following the game’s matrix presentation, the player tries to guess which place his friend might choose. As stated, the goal of each player is to go out together; therefore, it is not a foregone conclusion that a player should actually select his most preferred outcome. If the players pick the same outcome, they enjoy a night out together (albeit to a different extent each); otherwise they stay home receiving zero reward and regretting their choices.

2) Game Theory

This chapter provides the theoretical game-theoretic background that motivated the creation of the applications. The chapter's section 2.1.1 begins by giving a definition for Game Theory, games, strategies and etc. In section 2.1.2, a brief historical background of Game Theory is introduced.   Normal form games are explained in section 2.2. The Prisoner's dilemma and The Battle of the sexes games are described in section 2.3 and 2.4 respectively.

2.1 Introduction

2.1.1 Definition

Game theory is a mathematical theory to model phenomena that can be observed when two or more decision agents have an interaction [11]. The theory provides the explanation of conscientious and objective decision processes involving more than a one single individual.

A game is a situation where the participants (players) make decisions in order to maximize their individual outcomes (payoffs). Every player has a set of strategies. A strategic game is composed by a set of players .The action profile of a player is formed after each player chooses his strategy. The game involves conflict between the participants.

More specifically, a pure strategy game has the following basic elements [12]. There is a finite set of players, represented by $G = \{g_{1,}g_{2,\ldots,}g_n\}$**.** Each player $g_1 \in G$ has a finite set $\boldsymbol{S_i} = \left\{s_{i1,}s_{i2,\ldots,}s_{im_i}\right\}$ of options, named as *pure strategy* of player $g_i(m_i \geq 2)$. A vector $= \{s_{1j_1}, s_{2j_2}, \ldots, s_{nj_n}\}$, which s_{ij_i} is a pure strategy for player $g_i \in G$ is a pure strategy profile. The set of all pure strategy profiles forms the Cartesian product:

$$S = \prod_{i=1}^{n} S_i = S_1 \times S_2 \times \cdots \times S_{n,}$$

And this is the pure strategy space of game. For the player $g_i \in G$, there is a utility function:

$$u_i : S \rightarrow \mathbb{R}$$

$$\boldsymbol{s} \longmapsto u_i(\boldsymbol{s})$$

That associates the payoff $u_i(\boldsymbol{s})$ of player g_i to each pure strategy profile $\boldsymbol{s} \in S$.

2.1.2 History

The earliest approaches to the study of game theory are traced in the 18th century [10]. In the 1920s, Emile Borel (1871–1956) and John von Neumann (1903–57) contributed with their work to the major development of the theory. A very important event in the development of the theory was the publication of the book *Theory of games and economic behavior (1944) by* John von Neumann and Oskar Morgenstern. The book contains the mathematical game theory based on strategic games and it established Game Theory as a field. A few years later (1950s), game-theoretic models were being used in economic theory and political science. During that period, psychologists began the study of human subjects' behavior in experimental games.

In the 1970s, Game Theory was first used as a tool in evolutionary biology. Consequently, game theoretic methods were beginning to predominate in microeconomic theory (and in many other fields of economics), being used also in a wide range of social and behavioral sciences.

A remarkable event in Game Theory history was the awarding of Nobel Prize in Economic Sciences to the game theorists John C. Harsanyi (1920–2000), John F. Nash (1928–), and Reinhard Selten (1930–). Specifically, they were awarded for their pioneering analysis of equilibria in the theory of non-cooperative games [13].

Game Theory is also used in the research of issues such as election, auctions, genetic evolution, etc. Recently game theory has been drawing attention in computer science and engineering [14].Some of the game-theoretic problems studied by computer scientists are:

- The computation of "price of anarchy", that reveals how the measurement of the efficiency of a system can be lowered due to selfish behavior of its agents.
- Algorithms and learning tools for computing or designing equilibria.
- Designing and management of recent network protocols.

2.2 Normal Forms Games

In Game Theory, normal form is a way to describe a game. The representation of a normal form game is a matrix known as payoff matrix. A payoff matrix is very useful to identify the strictly dominant strategies and the Nash Equilibria (N.E.) of the game. A strict dominant strategy is a strategy that provides a greater payoff regardless of what the other players do and provides the worse payoff regardless of what the other players do. An iterated elimination of strictly dominated strategies may lead the game to a single outcome. This single outcome is the Nash Equilibrium of the game [15].

A definition of Nash Equilibria from the view of the players, states that if all of them choose their strategies and no player can benefit by changing his/her strategy while the other players keep theirs unchanged, then the current set of strategy choices and the corresponding payoffs constitute Nash equilibrium [16].

Other important concept regarding the outcomes is Pareto Efficiency. Pareto Optimality, its synonym, is a measure of efficiency. It was named after Vilfredo Pareto (1848-1923), an Italian economist. An outcome, by definition is Pareto efficient if it is not possible to improve the payoff of one player without lowering the payoff of another player. An outcome A Pareto dominates outcome B if the payoff of one or more player is higher and none are lower in outcome A. A Strategy profile s (i.e. a set of strategies for all players which fully specifies all actions in a game) is Pareto optimal, or strictly Pareto efficient, if there does not exist another strategy profile $s \in S$ that Pareto dominates s [17].

This definition can lead to some conclusion regarding the Pareto Efficient (Optimal) strategy profiles such as:

- Every game must have at least one such optimum.
- At least one Pareto optimum, in which all players adopt pure strategies, must always exist.

Also some games will have multiple optima. For example, in zero-sum games, all strategy profiles are strictly Pareto efficient and this definition is valid due to the fact that in zero-sum games the sum of both players' payoffs is zero in all strategies. Consequently, any change of strategy that benefits one player certainly involves other player's payoff decreasing [12]. In the case of pure coordination games (common-payoff games), the Pareto Optimal strategy profiles have the same payoff. Coordination games are games that all players have the same payoff for every action profile, meaning that players don’t have conflicting interests.

2.3 Prisoner’s Dilemma

One of the most popular games in game theory is Prisoner’s Dilemma (PD). PD is an example of non-zero sum game [18]. In a non-zero sum game there is no single optimal strategy that is preferable to all others, such as in the coin game, for example. A non-zero-sum game is a situation where on one hand, a player’s win doesn’t mean another’s loss, but on the other, a player's loss does not necessarily mean that the other player wins. Therefore, the players involved in a non-zero sum game have some complementary interests and but also some conflicting interests.

PD describes a situation in which two suspects are arrested for the same crime. Both are interrogated in separate rooms (simultaneous game) [10]. They have 2 choices (strategies), they can either confess the crime (defect) or stay quiet (cooperate). Player 1 (p1) has to build a belief about what choice Player 2 (p2) will make and, afterwards, choose the best strategy. A generalized form of a payoff matrix for the game [12] is represented as follow:

p1/p2	C	D
C	a,a	b,c
D	c,b	d,d

Figure 3 - General form of Prisoner's Dilemma payoff matrix [12]

The relation between the payoffs is c > a > d > b. The relations seen in the matrix show that mutual cooperation has the best payoff for the social welfare, whereas mutual defection is social welfare’s worst scenario. Regarding each player’s personal gain, the best payoff is attributed to the player who chooses defection when the other one has chosen cooperation. If there is a mutual defection, the payoff gives a poor social welfare for both but it is not the worst payoff a player can individually achieve. From a player’s individual view, it is impossible to achieve the worst payoff possible if he defects. Because of the symmetry of the matrix, this rule applies to both players. A strategy is dominant if the strategy earns a player a larger payoff than any other independently of what the other player chooses. Therefore, defection is the dominant strategy for both players. An example matrix of the above general form is presented in the following matrix:

p1/p2	Cooperate	Defect
Cooperate	3 ,3	0 ,5*
Defect	5* ,0	1* ,1*

Figure 4 - Example of general payoff matrix with dominant strategies marked with asterisk

When the dominant strategies of all players in a game matches the same row(s) in matrix as seen in figure 4, then in this/those row(s) is/are the Nash Equilibrium/a of the game. The asterisks in the table above are the dominant strategies. If both players have dominant strategies in the same row, this strategy's match is the Nash Equilibrium. This is a visual definition of Nash Equilibrium based on dominant strategies and applies for pure strategy games such as this one. A far more precise definition [19] of Nash Equilibria is the following. Let *(S, u)* be a game with *n* players. Where S_i is the strategy set for player *i*, $S = S_1 \times S_2 \times \ldots \times S_n$ is the set of strategy profiles and $u=(u_1(s),\ldots,u_n(s))$ is the payoff function for $s \in S$. A strategy vector $s \in S$ is said to be a Nash equilibrium if for all players *i* and each alternate strategy $s_i \in S_i$, we have that $u_i (s_i , s_{-i}) \geq u_i (s_i , s_{-i})$. Based on at least one of the definitions, it is conclusive that the Nash Equilibrium (NE) of the Prisoner's Dilemma (PD) is mutual defection.

Pareto Optimality (Pareto efficiency) implies that a change in strategy may improve a player's own payoff but at the same time the payoff of the opponent is not decreased. The Pareto efficiency assures the improvement of social welfare as a whole by bettering a group of players, without worsening the position of any other.

Which outcome is Pareto Optimal in PD? An outcome of cooperate-cooperate offers the social welfare as mentioned previously, a definition that matches the Pareto Optimality criterion. No player would change strategy if he/she could be able to guess that their opponent would certainly cooperate. But that is an altruistic choice. In Prisoner's Dilemma, two purely rational self-interested prisoners would betray the other one based on their own wellness.

The outcomes from defect-cooperate and cooperate-defect reveal that not only a change in strategy doesn't worsen the opponent's payoff but instead it increases the opponent's payoff. So these outcomes are also Pareto Optimal. The NE of the game is Pareto dominated by the mutual cooperation, since both players have their outcome decreased, which means, as mentioned, a worst social welfare scenario.

What if PD is played for more than one time? A player would "remember" a previous choice of the opponent. This would result in the Iterated Prisoner's Dilemma, whose strategies are formed from opponent´s previous choices. A repeated game can be analyzed in two different cases distinguished by the number of times the game is played, either finite or infinite.

If the game is played n times and the number of times is known by the players, then in round n, both players have a motivation to defect. Armed with this knowledge, the players will also have motivation to defect in n-1, and so on. Backward induction can be used to determine a sequence of optimal actions. In this case, defection is a rational choice.

In the next case, cooperation is a rational choice when the number of turns is unknown to the players. The rational choice theory says that the decision maker chooses the best action according to his preferences, i.e. the action chosen by a decision-maker is at least as good, according to her preferences, as every other available action. So, clearly, it is best for the whole group if all players cooperate. However, each player individually is better off by defecting in any given situation. Therein lies the paradox. [10]

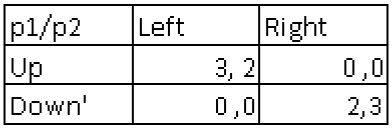
What if the number of players is more than two? Is there an optimal heuristic strategy? In 1984, Dr. Robert Axelrod, professor of political science at the University of Michigan, held a tournament of PD with various strategies. He invited many game theorists to provide their strategies, which were fed in a computer. During the tournament, programs played games against each other and themselves repeatedly for over 100 turns. The strategies were programmed algorithms based in previous self and opponent´s moves. All the entries and a totally random strategy were paired with each other in a round robin tournament. [9] [10]

The winner strategy was one of the simplest, Tit for tat. This strategy consists of repeating opponent's previous choices. In Axelrod's tournament, Tit for Tat usually did best because it could establish and maintain cooperations with many other players (agents). This strategy could prevent malicious players from taking advantage of it. [20] There are other possible strategies that can be found in the book "The selfish Gene" by Richard Dawkins that had not been submitted in the tournament but they are also interesting. Tit For Two Tats – it is Like Tit For Tat except that the opponent must make the same choice twice in a row before it is reciprocated. Suspicious Tit for Tat is the same as Tit for Tat but the opponent begins by defecting. Naive Peace Maker is the same as Tit for Tat, but randomly chooses to cooperate. The Grudger strategy cooperates until the opponent defects. After the first defection, the strategy will always defect. The Gradual strategy also cooperates until the opponent defects. If the opponent defects, the strategy will defect the total number of times the opponent has defected during the game so far, and then cooperates twice. The Remorseful Prober

Strategy, mimics Tit for Tat, but defects randomly. If the opponent defects after a random defection by the strategy, the strategy will cooperate once.

2.4 Battle of The Sexes (BoS)

The Battle of the Sexes game is a simultaneous game that describes a situation in which a husband and a wife want to go out together. The game is also called Bach or Stravinsky [10]. BoS is a two-player coordination game. Example of BoS game:



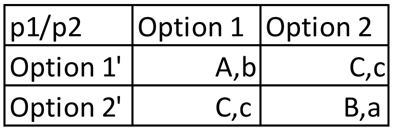
p1/p2	Left	Right
Up	3, 2	0 ,0
Down'	0 ,0	2,3

Figure 5 - Example of Battle of The Sexes payoff matrix

The non-zero payoffs are their preferences. The payoff for choosing different places is zero since they are both interested in going out together. The wife (p1) has a preference to go out to a place of her choice. The same applies to the husband (p2). BoS is an example of a game that combines elements of cooperation and competition.

In the Prisoner’s Dilemma the main issue is whether or not the players will cooperate. In the BoS, the players agree that it is better to cooperate, i.e. choose the same place to go out, than not to cooperate.

The general form of payoff matrix of BoS is:

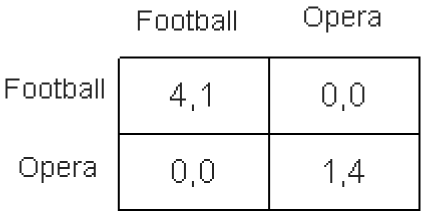


p1/p2	Option 1	Option 2
Option 1'	A,b	C,c
Option 2'	C,c	B,a

Figure 6 - General form of payoff matrix in Battle of the Sexes [15]

The relations between the values are A>B>C ≥ 0 and a>b>c ≥ 0.This relation leads to an affirmation that a player would prefer a place of his own preference. Regardless of the specific values, the players are trying to coordinate at the same location, though their individually most preferred outcome is different. The values in capital letters and the values in lower case can be either equal or different. In our application “Let’s go Out”, as shown in the following chapters, the common choices (Option 1-Option 1’, Option 2-Option 2’) are filled with players’ preferences while any other choice is zero.

Concerning the Pareto optimality in BoS, an example is given as follow:



	Football	Opera
Football	4,1	0,0
Opera	0,0	1,4

Figure 7 - Battle of Sexes example [21]

Let’s examine the pure strategies and the mixed strategies of the BoS. A pure strategy provides a complete definition of how a player will play a game. In particular, it determines the move a player will make for any situation he or she could face. In BoS, the pure strategies Nash Equilibria are all the non-zero values. If any of the players switch their options there will be the worst payoff for both of them since they want to go out together.

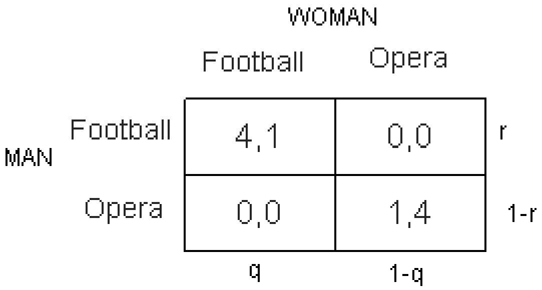
In the 2x2 BoS game in this example, there are two pure strategy Nash equilibria (4,1 and 1,4). A pure strategy, in this case, defines the nature of the choices of each player. In BoS each player has a place he/she would rather prefer. A different pure strategy equilibrium is preferred by each player. However, either equilibrium is preferred by both players to any of the non-equilibrium outcomes. In the case of each equilibrium, any deviation would imply in worse choice for both players. Even if there is the intention, for example, from one player to just not achieve the other player’s choice (“I am staying at home because I only want to go out to a place of my choice”), this player loses by staying at home. Thus, both equilibria are Pareto optimal since agreeing in going to one place Pareto-dominates the choice of staying at home. In the example, player 2 choosing left is a good result for both players.

A mixed strategy of a player in a strategic game is a probability distribution over the player's pure strategies [22]. This allows for a player to randomly select a pure strategy. To specify a mixed strategy of a player that prefers to go to the Opera, as example, we need to calculate the probability it assigns to each of this particular player´s actions.

There are some incentives to cooperation for both players in order to show up in the same place. However their preferences are different, so there is a motivation to compete against each other. Are they going to cooperate due to this mismatch in preferences? And further, which is the probability a player would change his preference since there is no clear equilibrium? This question can be answered after an analysis of a mixed strategy profile. Mixed strategies are helpful to understand how to solve this kind of games. BoS is a game where two NE appear, meaning that no real equilibrium can be found. In this case, knowing your opponent's strategy will not help you decide on your own course of action, and there is a chance that an equilibrium may not be reached. The way to solve this dilemma is through the use of mixed strategies, in which we look at the probability of our opponent choosing one or the other strategy and balance our pay off against it.

Let's suppose that the player that prefers the Opera (a woman) is inclined to choose Football with probability q and Opera with probability $(1-q)$. Similarly, the man is inclined to choose Football with a probability of r and Opera with a probability of $(1-r)$. In that case, the outcome results are:

- Football-Football: $q\ r$
- Opera-Football: $(1-r)\ \ q$
- Football-Opera: $r\ (1-q)$
- Opera-Opera: $(1-q)\ (1-r)$



		WOMAN		
		Football	Opera	
MAN	Football	4,1	0,0	r
	Opera	0,0	1,4	1-r
		q	1-q	

Figure 8 - Battle of The Sexes example with probabilities

The man's chances of going to a Football match, his expected utility (calculated by taking the weighted average of all possible outcomes), will be $4r$ (payoff*probability) and, of Opera, $1-r$ (because his utility from Opera is 1), therefore $4r = 1-r \Rightarrow r=1/5$.

Equivalently, for the woman, $q=4(1-q) \Rightarrow q= 4/5$. Now she must balance what q (the man's chances of valuing his own happiness over hers) really is. If $r>1/5$, they'll go to a Football match. If $r=1/5$, either could happen, and if $r<1/5$, the woman will get her own way and they'll go to the Opera. This is a simultaneous game as mentioned, which means that both have to balance their decisions carefully. If any side makes a mistake in evaluating the other's probability, both might achieve less utility. In this case, they could end spending the weekend apart.

To calculate the mixed strategy of the generalized form of BoS (figure 8) we have to find the expected utility for both option 1 and option 2 [15]. Let EU be the expected utility and *σ* to represent the probability that a player plays a particular pure strategy. So, for some percentage of the time, a player chooses c with σ probability and the rest of the time a player chooses b with *(1-σ)* probability. In such a case the relation between the EU and σ is

$EU_{Option\ 2} = f(\sigma_{Option\ 1'}) = \sigma_{Option\ 1'}\,(c)$ + (1-$\sigma_{Option\ 1'}$)(b)

$EU_{Option\ 1} = f(\sigma_{Option\ 1'}) = \sigma_{Option\ 1'}\,(a)$ + (1-$\sigma_{Option\ 1'}$)(c)

$EU_{Option\ 2} = EU_{Option\ 1}$

After the solution of those equations we have $\sigma_{Option\ 1'} = \frac{(b-c)}{(a+b-2c)}$ for a*+b-2c ≠ 0* and $\sigma_{Option\ 1'} > 0$. Because *a>b>c,* $0 < \sigma_{Option\ 1'} < 1$. The same applies to the opponent player.

Interestingly, all Nash equilibria of the game are problematic: The two pure strategy equilibria are unfair, since they allow one player to consistently do better than the other; while the mixed NE is inefficient. [23] The Nash Equilibria in BoS can’t be solved using the iterated deletion of strictly dominated strategies. There is no dominant (or dominated) strategy in BoS. If the woman chooses Opera, he would rather choose Opera. If she chooses Football, he would rather choose Football. The same applies to the man. So it is not possible to delete any strategies from the payoff table during an iterated deletion in such case.

3) Android Platform Fundamentals

This chapter introduces Android platform by offering an introduction to its fundamentals, as well as its basic and more advanced features. It begins by describing in section 3.1, the importance of Android APIs. In the next sections 3.2 and 3.3, the architecture of android OS and its kernel are explained. The Android Runtime, which forms an essential part of the Android platform, is explained in section 3.4. The Application Framework that provides the foundation to application building is explained in section 3.5. The Application Layer and Setup settings are explained in sections 3.6 and 3.7 respectively.

3.1 Android APIs

The Application Program Interface (API) can be defined as a set of protocols, routines and tools for building software applications. The APIs specify how software components should interact and they are used when programming graphical user interface (GUI) components [24].

The API level on Android is associated with a release version and indicates all the available capabilities of the specific version. Native and third-party applications are written with the same APIs and executed on the same run time. API levels, from a developer's point of view, means that it is possible to communicate with the devices' built-in functions and manage their functionality. As the API level increases, its functionality increments. Older APIs functions may become deprecated. [25]

The Android platform provides a framework API which applications can use to interact with the underlying Android system. [26] The framework API consists of:

- A set of XML elements and attributes for :
  - declaring a manifest file
  - declaring and accessing resources
- A core set of packages and classes
- A set of Intents (further explained in section 4.2.5)
- A set of permissions:
    - That can be requested by applications during the installation.

- That are permission’s enforcements included in the system i.e. high-level permissions that restricts access to entire components of the system or the application.

As the development of Android Operating System continues, new APIs are added and old APIs become obsolete and are eventually erased. Each Android version has its own API level. During the application development, the developer can define a minimum, maximum and preferred API level in their manifest file (described in section 1.2).

3.2 Architecture

The Android architecture provides the ability to publish and share Activities, Services, and data with other applications. It encourages the concept of component reuse. The access to those components are managed by security restrictions, in which (restrictions) are set during the application's development. This tool allows the developers to produce a replacement phone dialer or contact manager that can let them expose their application components. Subsequently, other developers can create new UI front ends and functionality extensions and also build on them [27]. In the following sections, each of the Android architecture layers will be described.

3.2.1 Android Layers

Android Operating System is a set of software components which is composed by five sections and four main layers of functionality as shown in figure 9 [28] .Those sections are:

- **Linux Kernel** - This section/layer provides basic system functionality such as memory management, process management and device management (ex. camera, keypad, bluetooth). The kernel also handles the networking.

- **Android Runtime** - This section, part of the second level, provides the Core Libraries and the Dalvik Virtual Machine.

- **Libraries** - This section, part of the second level, is a set of libraries written in C/C++ that includes:

    - WebKit - Provides tools for browsing the web i.e. to display html content.
    - SQLite - Contains the SQLite database management classes that an application would use to manage its own private database.
    - Libc - the library libc is a derivation of BSD's (Berkeley Software Distribution) standard C library code, originally developed by Google for Android OS.
    - SSL - The Security Socket Layer is a common building block for encrypted communication between clients and servers.
    - OpenGL - This library is a cross-platform graphics API that specifies a standard software interface for 3D graphics processing hardware.
    - Surface Manager - This library manages access to the display subsystem and composites 2D and 3D graphic layer for multiple applications.

- **Application Framework** - This framework provides the foundation for building applications. Developers have full access to the same framework APIs (ex. The Activity Manager) used by the core applications. This application architecture is designed to simplify the reuse of components and consists of classes for application creation, hardware access, as well as user interface and resource management. This set of capabilities also includes:
    - An extensive set of Views (explained in chapter 4) including lists, grids, text boxes and buttons.
    - Content providers that enable applications to share their own data and also access data from other applications.
    - A Notification Manager that enables applications to display custom alerts in the status bar.
    - An activity Manager that manages the lifecycle of applications.
    - An Android Resource Manager that manages the all the resource types of Android described in section 1.2.

- **Applications** - This is a set of the core applications that includes the SMS management, calendar, email client, browser, maps and others. Those applications are written in Java. It is important to note that a java application written for Android is not compatible with java programs written for the Java SE and Java ME platforms.

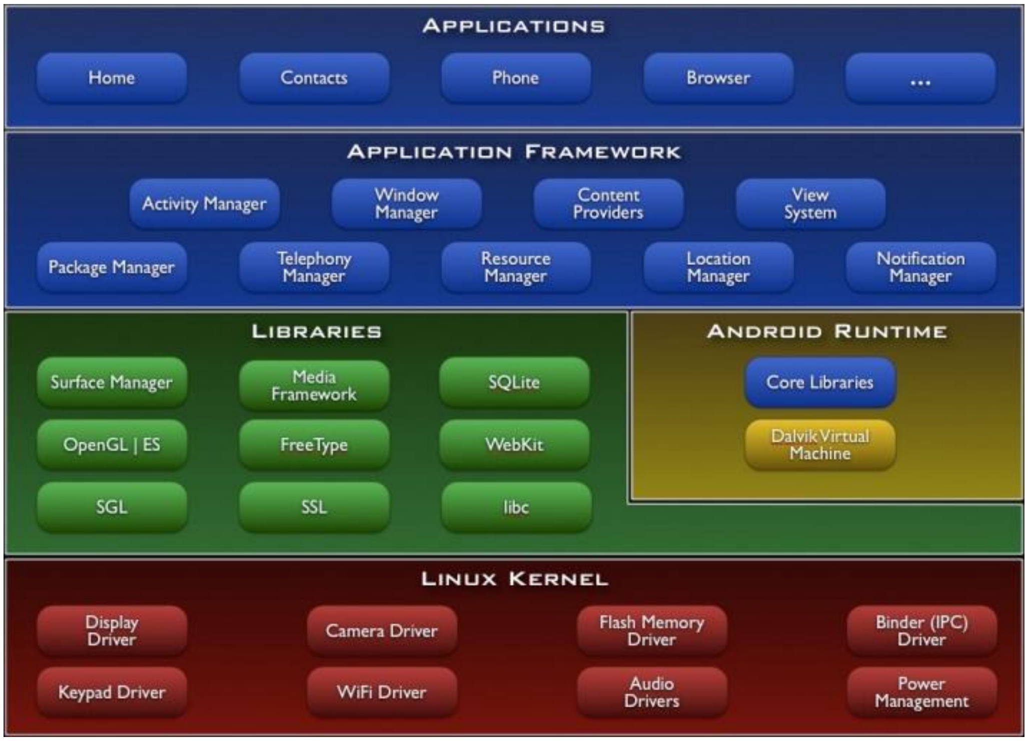


Figure 9 - Android Operating System Layers [29]

3.2.1 Android Class System - Views

This section describes a very important class group that is responsible for displaying onscreen features such as buttons, text, clock, progress bar, widgets and others. Widgets are used to create interactive UI components (buttons, text fields, etc) [30].

Additionally, this group contains classes that are responsible for the way all those contents will be displayed such as Frame Layout, Linear Layout and Relative Layout. The user interface of the Android application is composed by one or multiple forms such as windows or displays (ex. Dialog window). Each window is controlled by an Activity object that has a complex lifecycle explained in detail in section 4.6. The visual elements of the Activity instance are constructed using ViewGroup and View objects. The View class is the parent class of a large hierarchy of visual controls and widgets such as textboxes, checkboxes, buttons and others.

The ViewGroup class stands as the parent class for an hierarchy of layouts, used to manage collections of widgets and to define the layout architecture (linear, relative) of the window.

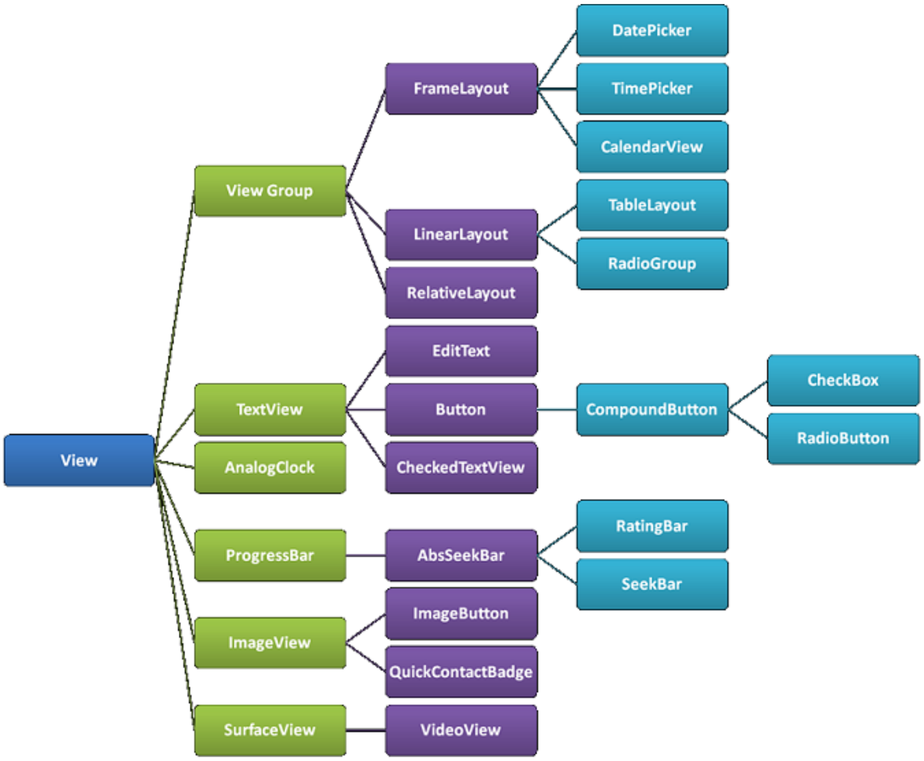


Figure 10 - Android Widget Framework [30]

The diagram above describes part of the Android widget framework that has at its root the View class. Below, there is an indicated usage of those classes in “Confess or Not”.

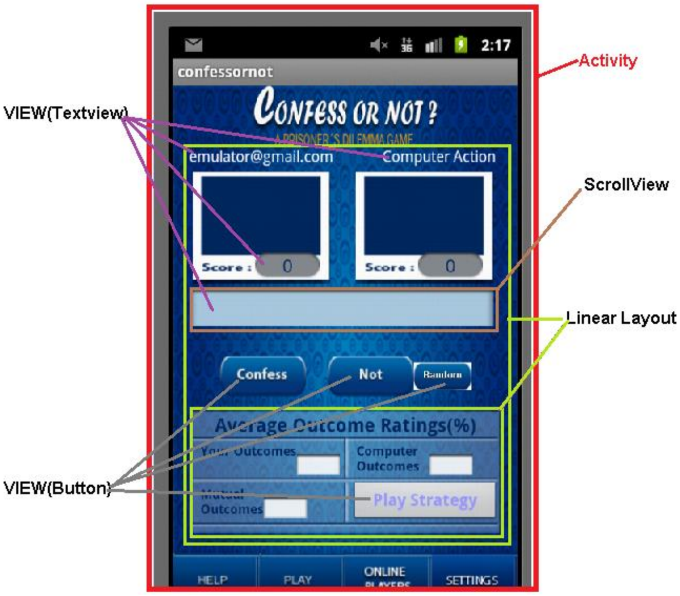


Figure 11 - Example of View usage in “Confess or Not” game

3.3 Kernel

3.3.1 Kernel description

A stripped down Linux Kernel lies at the base of the Android environment. This kernel is responsible for the process creation, device's drivers and file system management etc. The environment's current release uses Linux kernel version 2.6. Android OS does not utilize a standard Linux kernel and should not be named as a Linux solution. A Linux kernel is used for communication between the device's hardware and processors. Android OS supports multiple device processors. Some of the Android specific kernel enhancements include [31]:

- Android Binder - It is the core subsystem of Android OS and it is used instead of Inter-Process Communication (IPC- such as sockets and pipes[2] used in Linux kernels) [32]. With Binder, one Android process can call a routine in another Android process by identifying the method to invoke and pass the arguments between processes.
- Android Shared Memory Driver (ashmem) - It is a component of the Android OS that simplifies memory sharing and conservation [33].
- Alarm Driver – This driver provides timers to wakeup devices.
- Low Memory Killer (LMK) - Every application a user opens remains in the Random Access Memory (RAM) of the device. This mechanism decreases the time needed to reload an application. When the RAM overloads, the LMK deallocates some memory by shutting down the application(s) that are no longer needed.
- Android Power Management (PM) - Android OS requires that applications and services request CPU resources with "wake locks"(i.e. a way to keep the device working) through the Android application framework and native Linux libraries. If there are no active wake locks, Android will shut down the device's CPU. It is a more intrusive approach than the Linux's PM solution [34].
- Kernel Debugger and Logger.

[2] Inter-process communication (IPC) is the transfer of data among processes. Socket is a data stream sent over a network interface, either to a different process on the same computer or to another computer. Pipe is a two-way data stream interfaced through standard input and output and is read character by character. [65]

3.3.2 Kernel Booting

The Android Kernel boot is initialized after these two earlier general boot steps:

- Boot ROM – It starts when the power supplies are stable.
- Boot Loader – It is used to set up initial memories and load the kernel to RAM.

During the kernel boot, there is a core kernel initialization where memory and I/O areas are initialized. Interrupts are started, and the process table is initialized. Then the driver is initialized. Kernel daemons (threads) are started. The root file system is then mounted [35]. The kernel then runs */init* process which is the parent of all processes on the system and the first user-space process (a set of locations where normal user processes run) is started.

3.3.3 Kernel Versions

version	release date	API level	Android versions	Kernel
1.0	23/9/2008	1	-	-
1.1	9/2/2009	2	-	-
1.5	30/4/2009	3	Cupcake	2.6.27
1.6	15/9/2009	4	Donut	2.6.29
2.0	26/10/2009	5	Eclair	2.6.29
2.0.1	3/12/2009	6	Eclair	2.6.29
2.1	12/1/2010	7	Eclair	2.6.29
2.2-2.2.3	20/5/2010	8	Froyo	2.6.32
2.3-2.3.2	6/12/2010	9	Gingerbread	2.6.35
2.3.3-2.3.7	9/2/2011	10	Gingerbread	2.6.35
3.0	22/2/2011	11	Honeycomb	2.6.36
3.1	10/5/2011	12	Honeycomb	2.6.36
3.2	15/7/2011	13	Honeycomb	2.6.36
4.0-4.0.2	19/10/2011	14	Ice Cream Sandwich	3.0.1
4.0.3-4.0.4	16/12/2011	15	Ice Cream Sandwich	3.0.1
4.1	9/7/2012	16	Jelly Bean	3.0.31
4.2	13/11/2012	17	Jelly Bean	3.4.0
4.3	24/7/2013	18	Jelly Bean	3.4.39
4.4	31/10/2013	19	KitKat	3.4.39

Figure 12 - APIs and Kernel Versions [36]

Each version of the API contains its own kernel. Users can change the kernel but it implies some risks if it's not properly configured. Overclocking the kernel in order to get better performance is an example of a possible differentiation of the kernel's initial settings.

3.4 Android Runtime - Dalvik Virtual Machine

Android runtime is a defining component at the core of the Android software stack. It consists of the Dalvik core libraries and the Dalvik Virtual Machine (VM). Android runtime is what powers and handles the running of applications and provides the basis for the Application Framework. The Dalvik core libraries are a set of C/C++ code libraries that provide key functionality for applications. They implement general purpose[3] APIs used by code written in the Java programming language.

Android OS usually runs in devices that have some limitations such as limited memory/processor speed, OS without swap space and battery powered devices. Given those limitations, it was not possible to use a standard Java virtual machine. Google made the decision to create a new virtual machine Dalvik, the best answer to these limitations. Dalvik is essentially a Google's version of Java VM. In Android OS, each application runs in a separate process in its own Dalvik instance within a Linux process.

Dalvik virtual machine runs its own version of a compressed file. Instead of compressing and packaging the resulting class files into a .jar file, they are translated into *.dex* (Dalvik executable) files by the dx tool. The Dalvik VM executes *.dex* files. A *.dex* file is composed by taking the compiled Java *.class* or *.jar* files and consolidating all the constants and data within each *.class* file into a shared constant pool. After the conversion, the *.dex* file has a significantly smaller file size. This format is optimized to save memory [37]. A comparison between a .jar file and a .dex file after the translation is shown in figure 13. Heterogeneous constant pools are converted in separated types.

[3] General purpose APIs can be redefined during development.

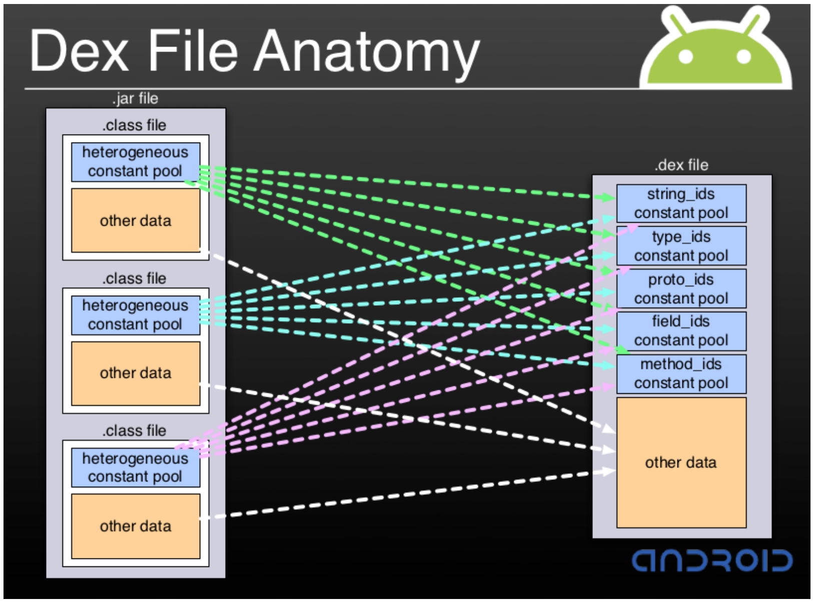


Figure 13 - Dex file anatomy [37]

Android applications use the Dalvik core libraries both directly and indirectly for data structures, networking, concurrency, I/O, and more. The Dalvik libraries break down into two categories:

- Dalvik VM-specific libraries: The VM specific libraries handle the VM specific information by enabling its request or modification. The code in those classes is only portable across Dalvik-based systems. The VM-specific Dalvik packages include the *dalvik.annotation*, *dalvik.bytecode* and *dalvik.system*.

- Java programming language interoperability libraries: This category of libraries provides a familiar environment for programmers writing code in the Java programming language. Much of the implementation of this code comes from Apache Harmony software, which is a modular Java runtime with class libraries and associated tools.

During compilation, Dalvik uses a just-in-time compiler (JIT). The JIT compilation is done during the execution of a program rather than prior to the execution. Suppose a user downloads and installs an app to his device. The app is allocated in the device until it is launched. During this process, Android pulls all the uncompiled data together, compiles them, and loads the application into RAM memory. During execution, all data is loaded to RAM. If the user uses a task killer, manually kill the app, or navigate away and load up other apps, the first app is unloaded, freeing up the RAM it had been using. When the user goes back to the initial application, it must go through the whole process all over again.

Recently, Android has added the ART ("Android Runtime")[4] to the version of Android 4.4 [38]. The ART uses ahead-of-time (AOT) compilation which pre-compiles an application during the installation. This pre-compilation allocates more memory space on a device and also initial installation takes longer time comparing to Dalvik. Meanwhile, applications launch and execute faster when using ART. As a result, because of the decreased use of the processor, battery life is increased. In Android 4.5, ART completely substitutes Dalvik.

In figure 14, a benchmark[5] of Quick Sort algorithm executed on an Android device with ART and Dalvik shows that ART is more time efficient than Dalvik.

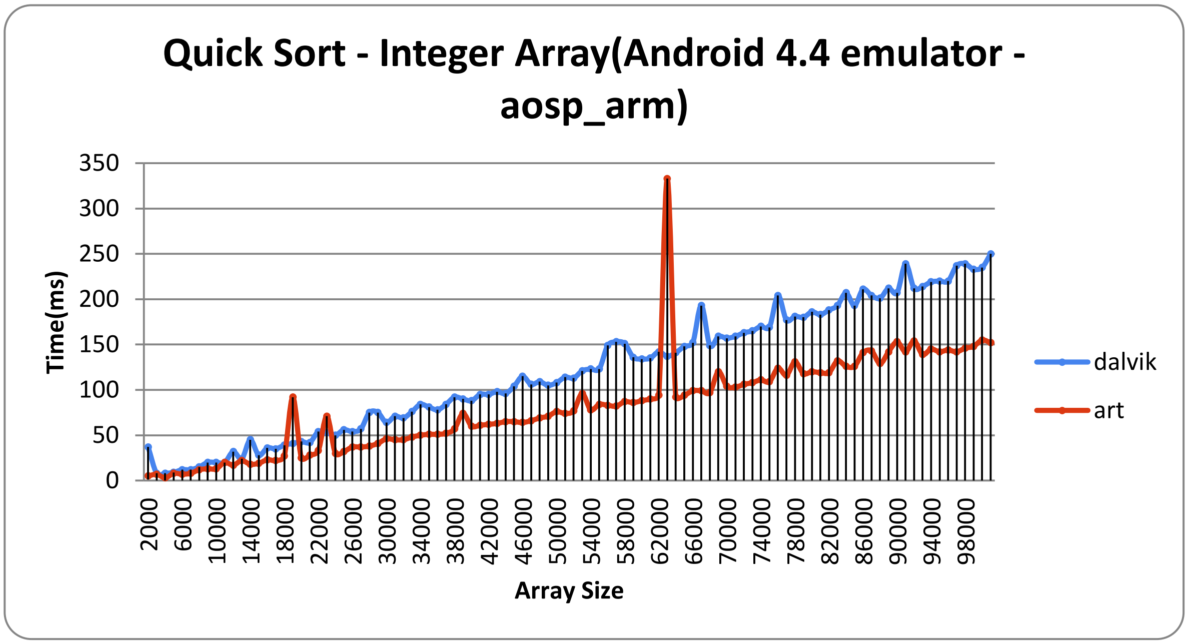


Figure 14 - Comparison between ART and Dalvik performance [39]

3.5 Application Framework

The Application Framework is a collection of services that incorporate the environment in which Android applications can be managed and run. This framework implements the concept that Android applications are composed of replaceable, interchangeable and reusable components [40].

[4] This is the name of the new Android runtime. It is different from the previously explained Android Runtime.

[5] The source code for this test is provided in the cited site.

This concept is taken a step further in that an application is also able to publish its capabilities along with any corresponding data so that they can be found and reused by other applications. The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

The Android framework includes the following key services:

- **Activity Manager** – Controls all aspects of the application lifecycle and activity stack. Interacts with the overall activities running in the system.
- **Content Providers –** In Android OS, there is no common shared space of data which any android package could access. Content providers store and recover data to make it accessible (by request) to another application. It is the interface that connects data in one process with code running in another process.
- **Resource Manager** – It is a class for accessing an application's resources. It provides access to non-code assets such as color settings, user interface layouts and strings.
- **Notifications Manager** – Allows applications to display alerts and notifications to the user. Programmatically, it is a class to notify the user of events that happen. This is how the user is informed of something happened in the background. A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area.
- **View System** –The View System contains the View and the ViewGroup subclass explained previously on section 3.2.1.
- **Package Manager** – It is a way the applications can find out information about other applications installed on the device.
- **Telephony Manager** – Provides information to the application about the telephony services available on the device such as status and subscriber information.
- **Location Manager** – Provides access to the location services allowing an application to receive updates about location changes.

3.6 Application Layer

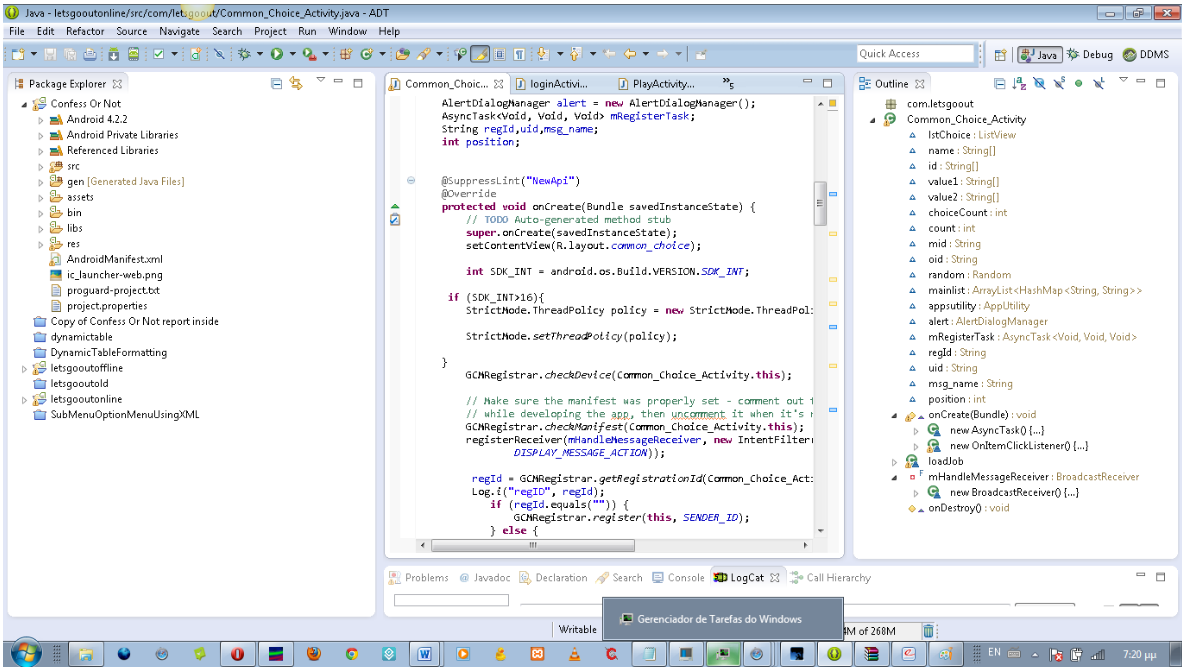
The applications are at the topmost layer of the Android stack. An average user of the Android device would mostly interact with this layer (for basic functions, such as making phone calls, accessing the Web browser etc.). Developed Applications can be installed on this layer exclusively [41].

Several standard applications are installed by default on any Android device:

- SMS client app
- Dialer
- Web browser
- Contact manager
- Any Application downloaded
- Any Application developed

3.7 Installation and Setup

The ADT Bundle provides everything needed to start developing apps, including a version of the Eclipse IDE with built-in ADT (Android Developer Tools) to streamline the Android app development [42]. It can be downloaded from Google's Android developer website. Setting up the bundle after downloading is quite simple. The user can unzip the file and start by clicking on Eclipse. The framework contains all the necessary plugins to start development.

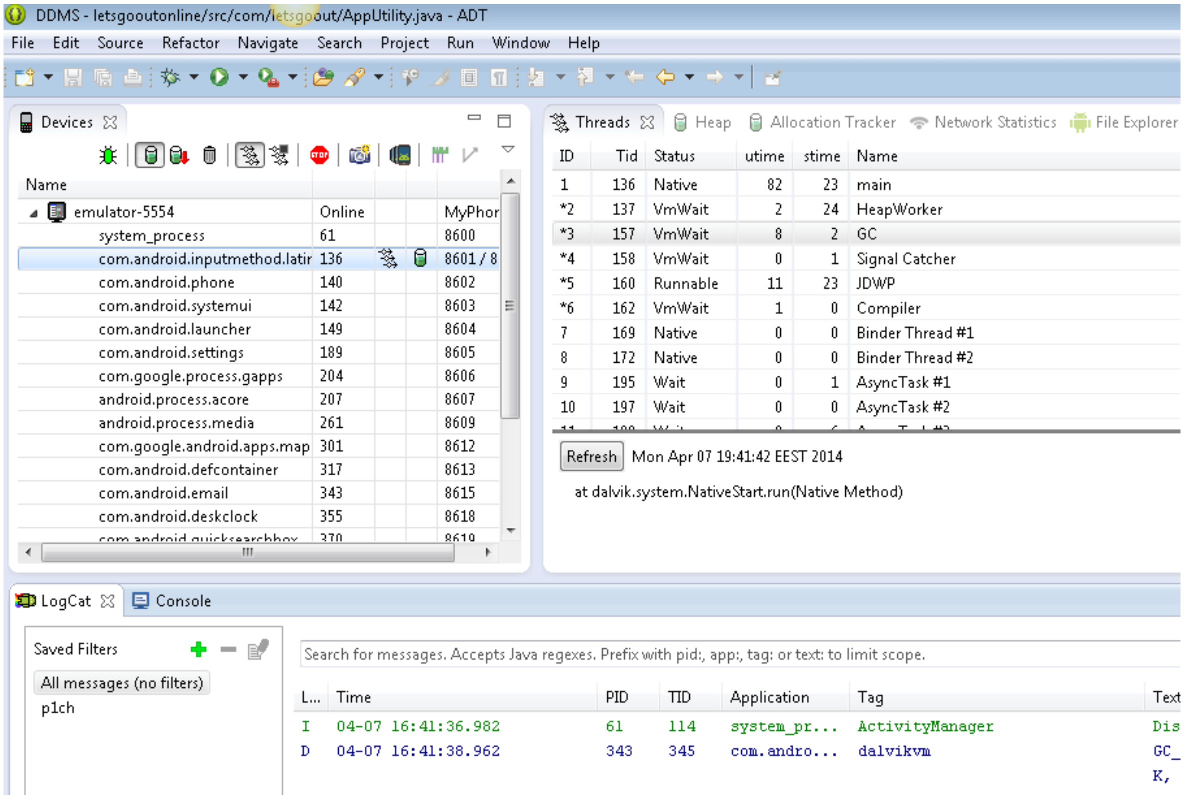


```
    AlertDialogManager alert = new AlertDialogManager();
    AsyncTask<Void, Void, Void> mRegisterTask;
    String regId,uid,msg_name;
    int position;


    @SuppressLint("NewApi")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.common_choice);

        int SDK_INT = android.os.Build.VERSION.SDK_INT;

     if (SDK_INT>16){
        StrictMode.ThreadPolicy policy = new StrictMode.ThreadPol

        StrictMode.setThreadPolicy(policy);

    }
        GCMRegistrar.checkDevice(Common_Choice_Activity.this);

        // Make sure the manifest was properly set - comment out t
        // while developing the app, then uncomment it when it's r
        GCMRegistrar.checkManifest(Common_Choice_Activity.this);
        registerReceiver(mHandleMessageReceiver, new IntentFilter
                DISPLAY_MESSAGE_ACTION));

         regId = GCMRegistrar.getRegistrationId(Common_Choice_Act
         Log.i("regID", regId);
            if (regId.equals("")) {
                GCMRegistrar.register(this, SENDER_ID);
            } else {
```

Figure 15 - Screenshot of Eclipse IDE with Android Developer Tools

The Eclipse environment contains:

- Java development where java code can be developed and built.
- Debugger where it's possible to run step-by-step Android Application.
- Dalvik Debug Monitor Server (DDMS), which provides port-forwarding services, screen capture on the device, thread and heap information on the device, *logcat* (a filtered system message error display), several processes (as seen in figure 16), radio state information (log radio data), incoming call and SMS spoofing, location data spoofing, and more.


DDMS - letsgoouto nline/src/com/letsgoout/AppUtility.java - ADT
File Edit Source Refactor Navigate Search Project Run Window Help
Devices
Threads
Heap
Allocation Tracker
Network Statistics
File Explorer
LogCat
Console
Saved Filters
All messages (no filters)
p1ch
Refresh

Name				
emulator-5554	Online			MyPhor
system_process	61			8600
com.android.inputmethod.latir	136			8601 / 8
com.android.phone	140			8602
com.android.systemui	142			8603
com.android.launcher	149			8604
com.android.settings	189			8605
com.google.process.gapps	204			8606
android.process.acore	207			8607
android.process.media	261			8609
com.google.android.apps.map	301			8612
com.android.defcontainer	317			8613
com.android.email	343			8615
com.android.deskclock	355			8618
com.android.quicksearchbox	370			8619

ID	Tid	Status	utime	stime	Name
1	136	Native	82	23	main
*2	137	VmWait	2	24	HeapWorker
*3	157	VmWait	8	2	GC
*4	158	VmWait	0	1	Signal Catcher
*5	160	Runnable	11	23	JDWP
*6	162	VmWait	1	0	Compiler
7	169	Native	0	0	Binder Thread #1
8	172	Native	0	0	Binder Thread #2
9	195	Wait	0	1	AsyncTask #1
10	197	Wait	0	0	AsyncTask #2

Mon Apr 07 19:41:42 EEST 2014

```
at dalvik.system.NativeStart.run(Native Method)
```

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.

```
L...  Time                  PID   TID   Application   Tag               Text
I     04-07 16:41:36.982    61    114   system_pr...  ActivityManager   Dis
D     04-07 16:41:38.962    343   345   com.andro...  dalvikvm          GC_
                                                                        K,
```

Figure 16 - Screenshot of Android DDMS debugger

The ADT Bundle contains also several features such as:

- An edition of user interface with graphical features including a drag and drop tool explained further in the following pages.
- Android Virtual Device Manager where a virtual device (emulator) can be configured from scratch or is possible to make use of existent device configurations. System configuration varies from SD card size to device CPU.
- Android SDK Manager where all updates are available for all APIs.
- Integrated documentation for Android framework APIs. Developers can access documentation by hovering over classes, methods, or variables.
- Updates for the ADT plugin.

The code editors available in ADT, in addition to Eclipse's standard editor features, are custom XML editors to help developers create and edit Android manifests, resources, menus, and layouts in a form-based or graphical mode already mentioned [42]. Double-clicking on an XML file in Eclipse's package explorer opens the appropriate XML editor. There is the option to edit the code in Graphical Layout or the in the XML markup code.

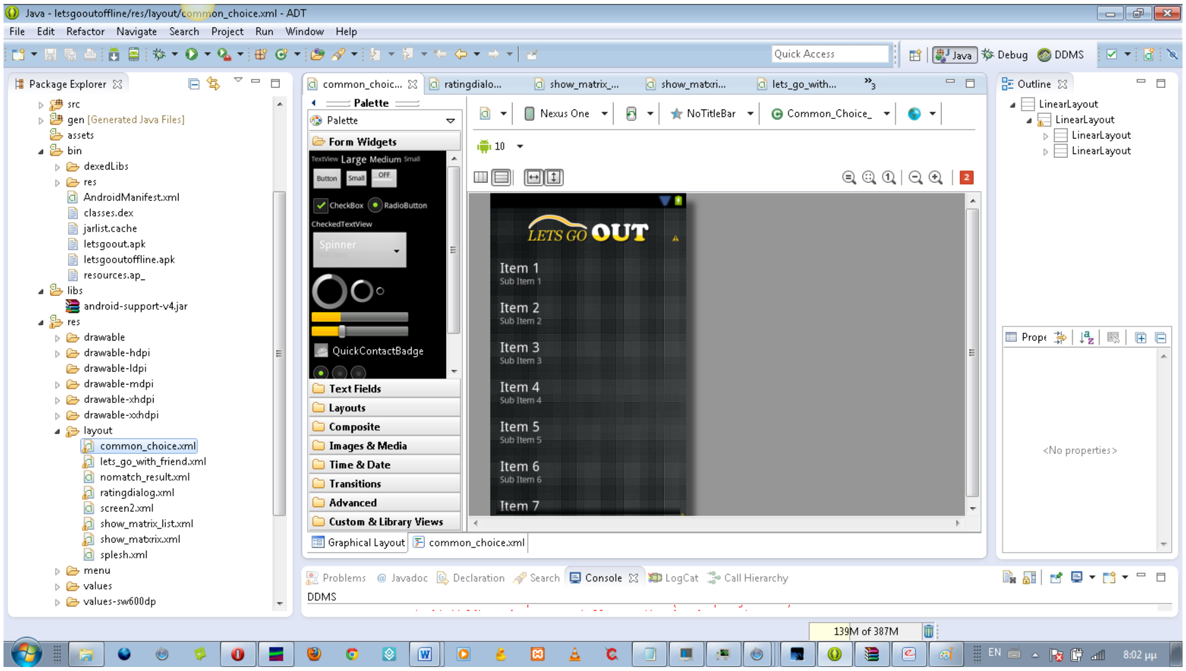


Figure 17 - XML editor built-in with graphical support

ADT provides the following custom, form-based XML editors:

- **Graphical Layout Editor** - In this editor developers can edit and design their XML layout files with a drag and drop interface. The layout editor renders the interface as well, offering developers a preview as they design their layouts. This editor is invoked when an XML file is opened (usually declared in res/layout).

- **Android Manifest Editor** - An editor for Android manifest file with a simple graphical interface. This editor is loaded when an AndroidManifest.xml file is opened.

- **Menu Editor** - When an XML file with a menu is opened , this editor is invoked. The menu feature is an auxiliary feature for the creation and editing of menus. This interface provides direct access to a previously created menu item.

- **Resources Editor** - Editor for the resources with a simple graphical interface. This editor is opened when an XML file with a *<resources>* tag is opened.

- **XML Resources Editor** - Editor for XML resources. This editor is opened when an XML file is opened.

In order to install an *.apk* , that is the executable of android, there are some ways such as:

- By publishing and downloading it from the Android Market.
- By copying it on the SD card and using another app to install it (many file managers offer this feature).
- By using the Android Debug Bridge (adb) that is a command line tool that allows the connection with an Android emulator or device.

4) Android Application Fundamentals

This chapter explains some important features of Android Application development's structure. It begins by explaining the concept of Context in section 4.1. In section 4.2, the main components are described. The lifecycle of an Android activity and those functions are described in section 4.3. A crucial element in Android development is the use of threads that are explained in section 4.4. And finally, an important service for message exchanging, Google Cloud Messaging is described in section 4.5.

4.1 Context

The Context class is an abstract class[6] provided by the Android system [2]. It allows access to local files, application-specific resources and classes, class loaders (associated to the environment), databases, and services (including system-level services).This class also allows upcalls for application framework's level operations such as launching activities, broadcasting and receiving intents (explained in 4.2.5), etc. An activity is a handle to the environment a certain application is currently running in. Further information on activities is given in sections 4.2.1 and 4.4.

From a developer's point of view, context is an entity that represents various environment data. It is an interface to global environment's information. In simple words, it is the context of an application's/object's current state. It lets newly created objects to understand the program's state at a given moment. Typically it is called to get information regarding another part of the program such as an activity, a package or an application.

Context is the base class for Activity, Service, Application and others. Developers can get the current context by invoking in the code *getApplicationContext()*, *getContext()*, *getBaseContext()* or *this* (when it is inside the activity class).

Typical uses of context [43]:

- **Creating New objects** such as views, adapters and listeners. Views are objects seen in graphical interface. The Adapter provides access to the data items. An event listener is an interface in the View class that contains a single callback

[6] An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed, as seen in [67].

method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI. It needs the current context as input. One example of an event listener is *onClick()* method that is required to Button View object.

- **Accessing Standard Common Resources between activities ,** such an example can be the "Shared Preferences", that are variables shared through all activities when invoked.
- **Accessing Components Implicitly,** regarding content providers, broadcasts and intents.
- **Activity Handling,** an Android application has activities. Activity is a Java code that supports a screen or a UI. In other words, the building block of the user interface is the activity. The activity object inherits the Context object. Context is used during launching of a new activity.
- **Obtaining system service,** such as security, memory management, process management and network stack.

A practical example of using the context class could be the case of creating a view dynamically in an activity class. Developers may want to dynamically create a TextView from Java code instead of creating it using XML layout. In such case, the TextView class can be instantiated. The constructor for the TextView class takes a context object, and because the Activity class is a subclass of context, the keyword "this" (as in conventional Java programming) can represent the context object. Each Activity is a context and each View needs a context so it can retrieve whatever resources it needs.
Context is very important because in Android programming, there is no static variable that would tell the developer the current global context of their application. The method *getApplicationContext()* is the closest way to get a "static" context of application.

4.2 Components

Android applications do not have a static entry point like a *main()* function. Android is designed in a component-based way. This means that an Android application can use another application or part of an Application. Components are the essential building blocks of an Android application. These components are declared by the application manifest file *AndroidManifest.xml* which describes each component of the application and how they interact among them.

When an application starts another application, or a component of it, a message from the requiring application is sent to the framework .This message is called Intent and will locate, instantiate, pass data and start a component based on some criteria. Those criteria can be packaged in the Intent. In order to start a component, it must be running and an instance of that component must be available.

There are four main components that can be used within an Android application:

Components	**Description**
Activities	They define the UI and handle the user interaction to the device screen
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.

4.2.1 Activity Definition

An Activity represents the presentation layer of an Android application, 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. An Android application can have several activities and it can switch between them during runtime of the application. An activity is implemented as a subclass of the Activity class [2].

Usually, a complete application consists of multiple activities that are bound to each other. In most cases, one activity in an application is specified as the "main" activity. The main activity is the first launch screen presented to the user. Each activity is able to start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack known as the "back stack". This name was taken from the back button, present in all Android devices that navigates the user to the last activity. When a new activity starts, it is pushed onto the back stack and takes user focus. The back stack abides to the basic LIFO stack mechanism, so, when the user is done with the current activity and presses the Back button, it is popped from the stack, it is destroyed and the previous activity resumes. An activity has a lifecycle that are state paths of the activity. More details concerning the activity's lifecycle can be found in section 4.6 of the present document.

4.2.2 Services

A service is a component that runs in the background to perform long-running operations without direct interaction with the user. A service has no user interface, that means that it is not bound to the lifecycle of an activity. 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.

Services, in general, are used for repetitive and potentially long running operations that are crucial to keep the user interface responsive. For example, service tasks as Internet downloads, data processing and checking for new data. They can also be used to update the content providers. A content provider manages access to a central repository of data and will be further explained in section 4.2.4.

In comparison with activities, a service can run with higher priority than invisible or inactive activities. It is also possible to assign the same priority to services as foreground activities, i.e. visible activities to the user. In such case, a visible notification to the user is required. For example, during a file download, a dialog containing the download progress is seen and the foreground activity waits for the download to end.

Another one of services' advantage is that they can also be configured to be restarted in case they get terminated by the Android System. The restart can be achieved once there will be sufficient system resources available.

4.2.3 Broadcast Receivers

Broadcast Receivers [44] are Android’s components that respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts, i.e. send messages to let other applications know that some data has been downloaded to the device and is available for them to use. This message is defined as broadcast and the component in charge of responding to this message is defined as Broadcast Receiver. In the example given, the broadcast receiver will intercept the communication between the sender and the receiver and will initiate an appropriate action.

A broadcast receiver is implemented as a subclass of the *BroadcastReceiver* class. Each message is broadcasted as an Intent object. An Intent is a messaging object you can use to request an action from another app component and will be further explained in section 4.2.5.

A practical example can be whenever the device`s battery is low, Android OS sends a message (broadcast) to the whole system informing that the battery charge is low. The applications that are interested in receiving this message, consequently, will be able to execute a determined action from this information. The Android system itself, has a component to respond to this message by displaying a dialog box on the screen informing the user that it is necessary to connect his device to the charger. The component that responds to this message is a Broadcast Receiver. In addition to the example cited above, there are several other events in Android that send a broadcast to the system, for example, when the device screen is off, when the charger is plugged into the device, when a headset is connected to the device, when a telephone call is received, when the system starts and several others.

A broadcast can be sent either by native Android applications either by the applications that are developed from the beginning. These broadcasts are sent via Intents, as mentioned before, in the same way as we do to invoke an Activity. So, the Intent Resolution is responsible for informing which Broadcast Receivers are interested in responding to the broadcast. The figure below illustrates an example of a broadcast sent by Android through intent, stating that the battery charge is low.

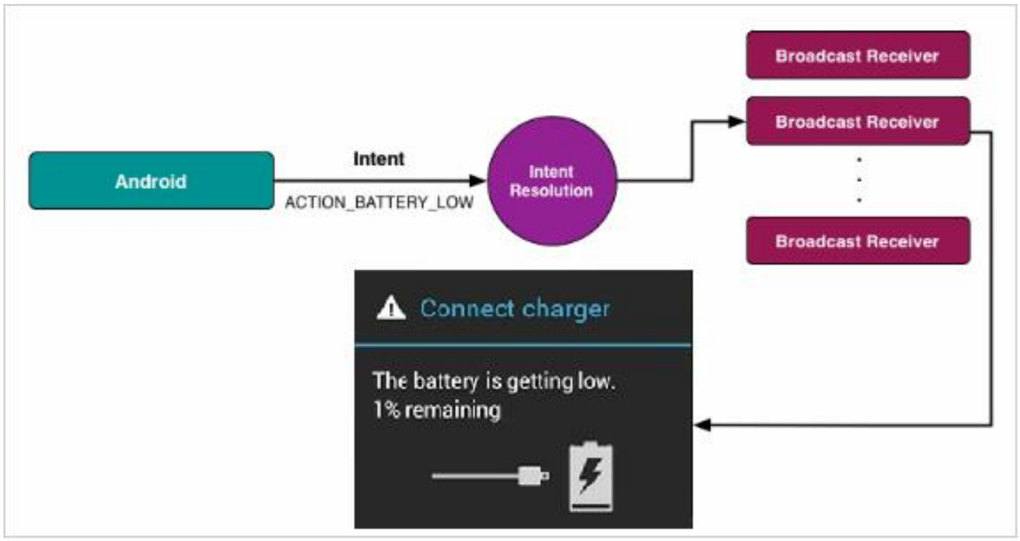


Figure 18 - Broadcast Receiver example indicating battery recharge [44]

In order to create a Broadcast Receiver, it is necessary to create a class that inherits the class *BroadcastReceiver* and implements the *onReceive()* function. Later in Activity Lifecycle section (4.6), it will become clear at which moment in an activity cycle the Broadcast Receiver is implemented.

The *OnReceive()* function is called by Android OS itself and takes as argument an object of type Context and an Intent object. This function enables the Broadcast Receiver to use the methods of these two classes and also allows the receiving of parameters through an Intent. Any code that a developer wishes to be executed in background by the Broadcast Receiver is implemented in *OnReceive()* function.

A Broadcast Receiver can be configured both statically and dynamically, i.e. editing the Android Manifest file or editing the code in Java inside an application's class. In static configuration, a Broadcast Receiver can be invoked even if the application is closed.

4.2.4 Content Providers

Content Providers store and recover data making them available to other applications. The data may be stored in the database, the file system or somewhere else entirely. Content Providers are needed because Android System doesn't have a shared location to store the applications' data and share this data among all Android packages (i.e. the Application). An example can be the shared contacts that can be used from both the phone dialer application and the calendar application. Content Providers can be developed for any application inside any application’s class in the source code. The class containing the Content Provider has to be implemented as a subclass of *ContentProvider* class and has to implement a standard set of APIs that enable other applications to perform transactions [45].

A content provider presents data to external applications as one or more tables that are similar to the tables found in a relational database. A row represents an instance of some type of data the provider collects, and each column in the row represents an individual piece of data collected for an instance.

The Android System has a lot of Content Providers as in the example above. A newly installed application can obtain the user´s contact list if the user allows this operation. In such case, the list can also be edited. A content provider is not needed for handling data in a unique application, but if the content is shared among other applications, a Content Provider is required.

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. This class provides applications’ access to the content model, i.e an interface to the content.

Also, the Content Resolver’s role is to receive requests from the clients and solve them, i.e., sends the request to another Content Provider. The *ContentResolver* has CRUD (create, read, update, delete) functions that correspond to the abstract functions of Content Provider such as insert, delete ,query, update. Each function takes as parameter a Uniform Resource Identifier (URI) [46] specifying the address of the Content Provider to interact with. In figure below, each application accesses the Content Provider by the matching URI.

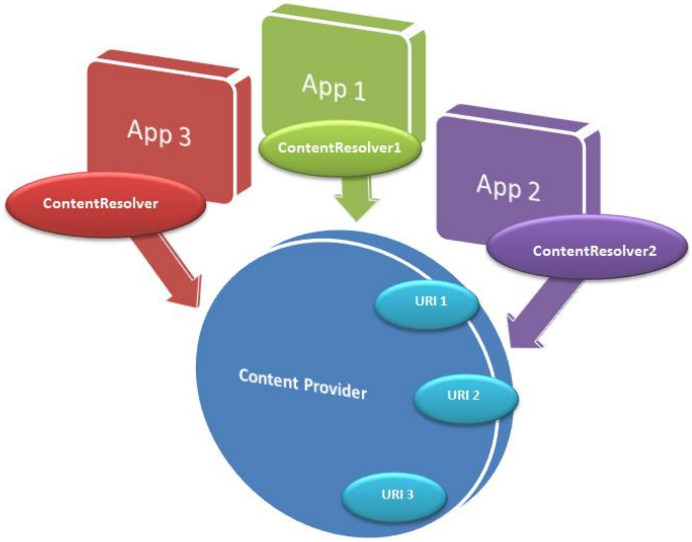


Figure 19 - Content provider and URI relation [46]

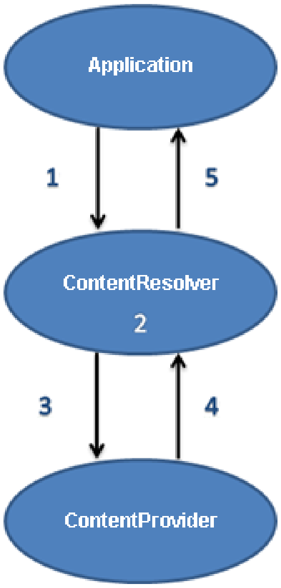


Figure 20 - Content provider and URI relation [46]

The lifecycle of a data re[illegible] in five steps:

1. An application calls function *getContentResolver().query(Uri, String, String, String, String*), this call executes the function "query" of Content Resolver.

2. When the call is made, the Content Resolver interprets the received URI (with the syntax starting with "content://"). Then the path of the requested Content Provider is extracted.

3. The Content Resolver redirects the request to the Content Provider (the path is known from the previous step). This is done when “query” function from Content Provider is called.

4. When the Content Provider executes the function “query”, a cursor is returned (or an exception) to the Content Resolver. A Cursor represents the result of the query and basically points to one row of the query result. The data model used by the Content Provider is a simple database table, where each row is a record and each column is data of a certain type.

5. The Content Resolver returns the result received from the Content Provider back to the Application.

One may wonder: Why don’t applications access data directly from a database instead of using the Content Provider? It is possible for the Applications to use a database, but it is even better to use the Content Provider because the way the data is recorded should be clear to the Application. In this case, the Application can focus on user interactivity instead of focusing on data retrieving. It is also possible to create Shared Content Providers that are “public” and seen from many applications such as the SMS/MMS Content Provider that allows any application to read received messages in a mobile device.

4.2.5 Intents

As previously explained on section 4.2.1, an Activity is related to a task that an application can perform. But how such a task is connected to a next possible task? They can be “connected” through messages called intents.

Intents are asynchronous messages which allow the application to request functionality from other services or activities. Intent is an abstract description of an operation to be executed. There are two main primary categories of intents:

- **Explicit Intents** are sent to a specified component. So the class to be run is provided as a parameter and the application calls directly a service or an activity. This is the way for an application to launch any activity that is visible and interacts with the user.
- **Implicit Intents** does not run on a specified component. The intent, in order to reach any destination (i.e. a component), must include enough information for the Android system to determine which of the available components is best to run for that intent.

For example the application could ask via intent for a contact application. Applications register themselves to intent via an *IntentFilter*.

A far more practical view of an Android **Intent** is that it is an object carrying an *intent* i.e. a message from one component to another component with-in the application or outside the application. The intents can be sent to any of the three core components of an application such as the activities, the services and the broadcast receivers .An example code of the use of Intent is presented as follows:

1- Call an application activity to be displayed on screen

```
Intent intent = new Intent(this,ClassActivity.class);
intent.putExtra("Message Name", "Mymessage");
startActivity(intent);
```

2-To open a browser

```
// Adress to be open
Uri uri = Uri.parse(http://www.tuc.gr);
// Creates the intent for the given address
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
// Send the intent to OS.The OS will handle the intent.
startActivity(intent);
```

4.3 Activities and Tasks

An activity is a visual component that you see on a screen. Each screen with some associated logic can manage its life cycle and navigation [2]. Each screen is a separated Java class. An application generally consists of several activities. When, for example, a user changes a screen by pressing a button, he is changing to the activity linked to that button. Each activity is defined as a subclass of *android.app.Activity* library, and must be declared in the manifest file. The activity also has a lifecycle, responding to the following state machine:

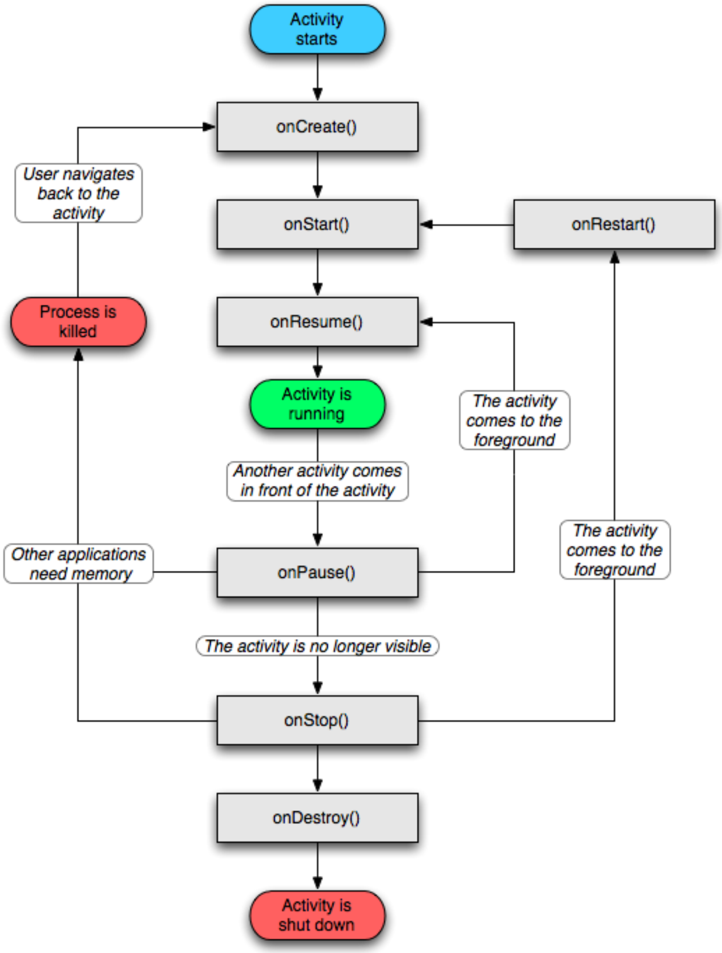

Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure 21 - Activity Lifecycle [47]

Each state's function can be edited and used during the development. For loading an activity and displaying a layout on screen, it is necessary to set the content View (load the xml layout) in *oncreate()* function. An example is as follows:

```
protected void onCreate(Bundle savedInstanceState) {
```

```
2.   super.onCreate(savedInstanceState);
3.   setContentView(R.layout.login);   }
```

At line 1, the on Create activity state is loaded. At line 2, this line is activating Dalvik VM to run the developed code in addition to the existing code in the *onCreate()*. At line 3, *setContentView* sets the activity content to an explicit view. So the *R.layout.login* screen is loaded.

Each state is described in the following table:

Callback	**Description**
onCreate()	This is the first callback and called when the activity is first created.
onStart()	This callback is called when the activity becomes visible to the user.
onResume()	This is called when the user starts interacting with the application.
onPause()	The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
onStop()	This callback is called when the activity is no longer visible.
onDestroy()	This callback is called before the activity is destroyed by the system.
onRestart()	This callback is called when the activity restarts after stopping it.

Figure 22 - Activity States table [48]

When an activity is on screen, it is in a running state (shown as green box in figure 21). When the Android switches between the activities, the current activity will be paused. Developers should take into account that their programs should be able to stop any CPU heavy processes that are running when the *onPause()* method is called.

Android may also request that an Activity should be stopped in order to recover memory. When this happens, the activity may be stopped. Also, after the *onStop()* method, the memory allocated for the application that has been stopped should be released. The *onCreate()*, *onStart()* and *onResume()* methods are used when the activity is brought to its running state.

Any activity may be started or restarted at any time. The information of a state of an activity can be saved at any moment. For example, when the screen rotates, the current activity is re-created so the activity can use a different layout and components. The user notices that the objects/Views are repositioned.

An example of the activity's state functions' usage can be seen in *loginActivity* in Confess or Not source code. The login activity displays an email and a password field for the users to fill the form in. Then it checks the correctness of the user's input format. A "Play" button is displayed under the login field. In the *onCreate()* function, the layout is displayed on screen with the login field and button. In the *OnResume()* function, the "Play " button is activated and has its own functionality, in this case, it switches to another activity. As mentioned in figure 22, the *OnResume()* state is called when the user interacts with the activity. In this example, the interactions are both the email/password entries and the pressing of the button.

4.4 Threads

Threads are fundamental for the multitasking operating systems [2]. They are mini-processes running simultaneously with a main process. The purpose of threads is to enable the appearance of parallel execution path within the applications.

When any Android application is first started, the runtime system creates a single thread in which all application components will run by default. This is the main thread and its role is to handle the user interface. Any additional components that are started within the application will, by default, also run on the main thread.

Any component within an application that performs a time consuming task using the main thread, may cause the stall of the entire application. The application will wait until the task is completed. This will typically result in the operating system displaying an "Application is unresponsive" warning to the user. This is a very unfortunate execution of an application. To avoid this kind of problem, it is recommended to launch a "heavy" task in a separated thread.

One of the key rules of application development is never to perform time-consuming operations on the main thread of an application. Another equally important rule is that the code within a separate thread should never directly update elements related to the user interface. All the changes in the user interface should be performed in the main thread. For example, the Services (explained in section 4.2.2) are started in the application main thread by default. Services perform background operations that may delay the main thread. The solution would be to assign the Service to another thread. With the correct use of threads, the user can execute background code without the danger of application's crashing.

4.5 Google Cloud Messaging

Google Cloud Messaging for Android (GCM) is a free service offered by Google that enables developers to send data from the servers to their Android applications on Android devices, and messages from the user's device back to the cloud [47]. This could be a lightweight message telling the Android application that there is new information to be brought from the server. In the projects presented in this document, GCM is used to register the user and also to send push notifications i.e. requests to other registered users. The GCM service handles all parts of the queueing of messages and their delivery to the target Android application running on the target device.

The GCM Cloud Connection Server (CCS) is a connection server based on Extensible Messaging and Presence Protocol (XMPP). The XMPP is an open, XML-based protocol for server-to-server near-real-time extensible instant messaging.

An installed Android application doesn't need to be running to receive messages. Google Cloud Messaging will deliver the message regardless of the application's status. The system will wake up the Android application via Intent broadcast when the message arrives. In the applications developed for this project, the broadcast receiver was set up so that the registered user is able to receive messages. In order to receive those messages, all proper permissions must be set in the manifest (configuration) file.

When the message is received by the device or the application on the device, it brings the system to [7]alert status. After GCM has completed the task by transmitting the "source data" to the application, the application will process the aforementioned data so that the end-user understands and assimilates them. Depending on the settings chosen by the user, the application will either display a notification symbol or synchronize the data received. In "Confess or Not", the registered device receives a notification request to play a game, various notifications during the game and a notification to end the game. In "Let's go Out", the registered device also receives a

[7] Through wake up locker file. This file is mentioned in chapter 5 and 6.

notification request as an invitation to go out, various notifications such as a notification containing the venues the other player had chosen and a notification with the final choices.

The GCM's device registration in both games is processed as follows:

- The Android application sends the sender's id and the application's id to the GCM server for registration.
- After registration, the GCM server issues a registration id to the Android device.
- After receiving the registration id, the Android application will send the registration id to his server.
- The server stores the registration id to the database.

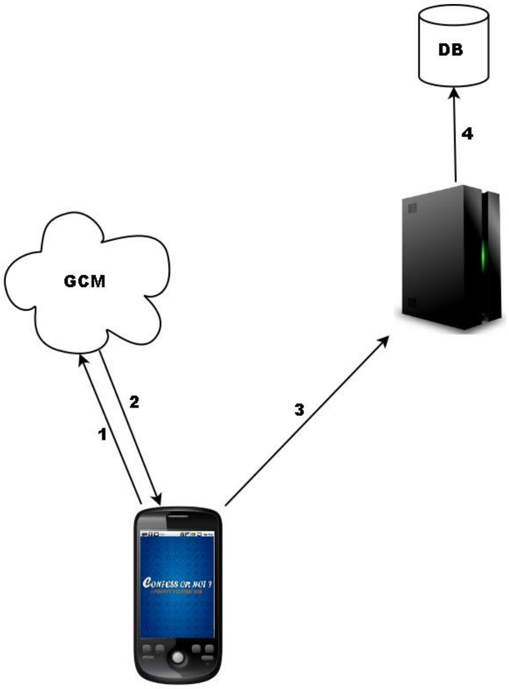


Figure 23 - The GCM data flow

After registration, all the registered users will be displayed in the player's list in the game, and whenever user A clicks on the player's displayed email, a notification message will be sent to user B. In this case, the server sends a message to the GCM server along with the device's registration id, which is already stored in the database. In both applications, "Confess or Not" and "Let's go out", the same steps are applied and each one has its own project number. The project number is obtained when the developer creates a project on a Google API console [48]. The GCM messaging has a lifecycle flow, comprised by the main processes involving the message exchange:

- **Enable GCM:** This can be achieved by an Android application running on a mobile device that registers to GCM in order to receive messages. When the Android application uses for the first time the messaging service, it calls the Google Cloud Messaging method *register()*. The *register()* method returns a registration ID. The Android application should store this ID for later use.
- **Send a message:** A sequence of events occurs when a 3rd-party application server sends messages to the device:
    - The application server sends a message to the GCM servers.
    - Google checks and stores the message in case the device is offline.
    - In case the device is online, Google sends the message to the device.
    - On the device, the system broadcasts the message to the specified Android application via Intent broadcast with proper permissions, this is necessary because the message should be received only by the targeted Android application. This broadcast wakes the Android application up. An android application can receive a message even when the target application is inactive.
    - The Android application processes the message. If the Android application is doing non-trivial processing, power manager .wakelock component can be added to the implementation in order to prioritize an application.
    - An Android application can unregister GCM if it no longer wants to receive messages
- **Receive a message:** Whenever an Android application receives a message from a GCM server:
    - The system receives the incoming message and extracts the raw key/value pairs from the message payload. Contingent upon the presented project, the message payload is in JSON format.
    - The system passes the key/value pairs to the targeted Android application in a com.google.android.c2dm.intent.RECEIVE Intent as a set of extras.
    - The Android application extracts the raw data from the com.google.android.c2dm.intent.RECEIVE Intent by key and processes the data.

In order to use the GCM library, the devices should be running Android 2.2 or higher. The devices also should have the Google Play Store application pre-installed, or an emulator running Android 2.2 with Google APIs. This application requires that the user has his device registered with a google email account (for pre-3.0 devices). A Google email account is not prerequisite for devices running Android 4.0.4 or higher.

5) “Confess or Not” Design

This chapter describes the “Confess or Not” game’s functionality. It begins with a general overview of the game. In section 5.1.1, the strategies used in the game are described using automata. The section 5.2. contains the flow of the game. The section 5.3 contains the description of the classes and methods used in both server’s and client’s side. The section 5.4 is composed of the implementation’s diagrams with the interaction of “Confess or not” game’s files.

5.1 Description

The idea of the Prisoner’s Dilemma, as mentioned in section 2.3, is that a player and his/her partner are caught by the police. Both are interrogated simultaneously. They have the choice to confess (defect) or not confess (cooperate). In “Confess or Not” game is played with a number of iterations. This number of iterations must be random as mentioned in section 1.3. In the game, the user can set a range of possible iterations when playing with the computer.

The payoff matrix with the points earned for each player for each iteration of “Confess or Not” is presented as follow:

p1/p2	confess	not confess
confess	1 ,1	5 ,0
not confess	0 ,5	3 ,3

Figure 24 - “Confess or Not” payoff matrix

For each iteration the sum of points is displayed as the score. In the final iteration, the sum is displayed. The values of Mutual Outcome and each user outcome ratings are also displayed. The Mutual Outcome rating represents the combined scores achieved by both players compared to the best possible combined score. The user outcome represents the percentage of points achieved divided by the maximum possible sum of points. The basic idea of the game, even the score system, is inspired from an online version of Prisoner’s Dilemma [49].

The user can set a specific strategy or even a random strategy. In case the user selects a random strategy, when pressing the button “play strategy “, the user can play an unknown random strategy. If the user chooses specific strategy, the button "play strategy" gives the result of the selected strategy.

In this game there are 10 different strategies as described in section 1.3: Tit for Tat, Tit for Two Tats, Suspicious Tit for Tat, Naive Peace Maker, Random 50/50, Always Defect, Always Cooperate, Grudger, Gradual and Remorseful Prober Strategy. All the possible player's moves are illustrated as State Diagrams for each strategy with the acronyms *C* and *N* meaning Confess and Not Confess analogously. For example, for the *CC* (Confess-Confess) acronym, the first *C* means the choice of a player playing a strategy, while the second C is the choice of the opponent playing a random one. In those State Diagrams, we show all possible moves of a player playing strategy against a player playing random. The arrows in some diagrams mean that the strategy has 2 possible initial states. Some strategies have an initial move such as Suspicious Tit for Tat, that is repeating the opponent's last choice but starting with defection (confess) [50].

5.1.1 Strategies in "Confess or Not"

Those strategies' automata are based in automata seen in [51]:

- Tit for Tat - Repeat opponent's last choice. Start by cooperation. Tit for tat was introduced by Robert Axelrod, who developed a strategy where each participant in an iterated prisoner's dilemma such as "Confess or Not" follows a course of action which is consistent with their opponent's previous turn. In such case, if provoked, a player will subsequently respond with retaliation, but if they are not provoked, the player will subsequently cooperate.

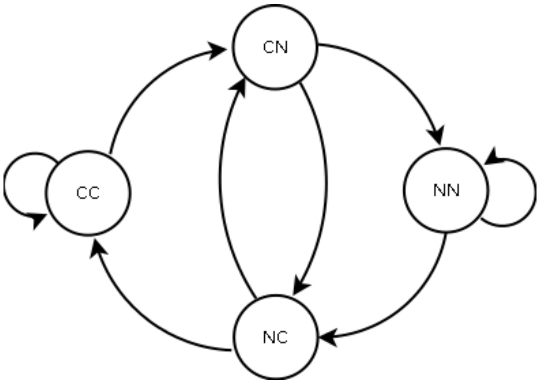


Figure 25 - Tit for Tat strategy possible moves

- Tit for Two Tats - If opponent defects two times in a sequence, then defect. Otherwise, cooperate. Start by cooperation.

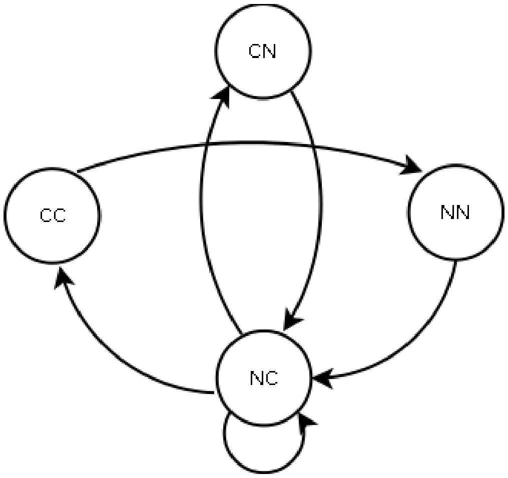


Figure 26 - Tit for Two Tats strategy possible moves

- Naive Peace Maker (Tit For Tat with Random Co-operation) - This strategy consists of repeating opponent's last choice, but sometimes settles with co-operating instead of defecting. Cooperation is randomly achieved.

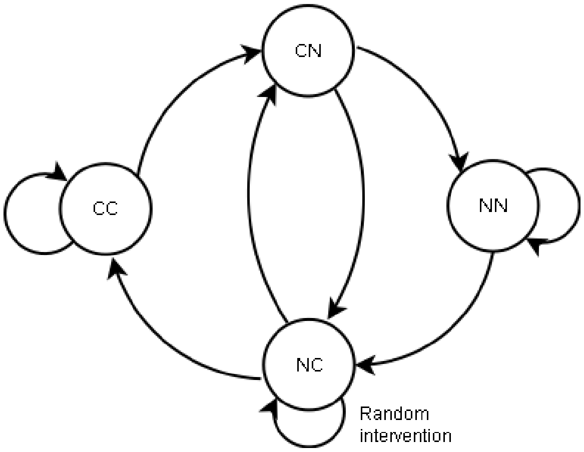


Figure 27 - Naive Peace Maker strategy possible moves

- Random 50/50 - Every turn is randomly played by a random generator with 50% probability. So any combination of subsequent plays, confess or not, is possible as seen in the diagram.

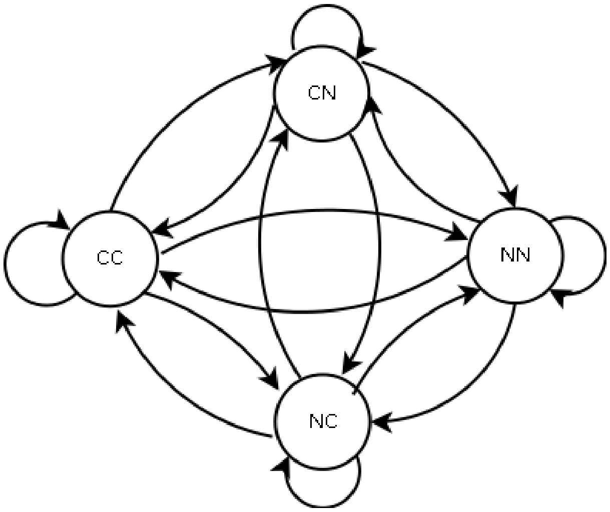


Figure 28 - Random strategy possible moves

- Always Defect - This strategy can achieve a Nash Equilibrium if the number of plays is known. In the case of Confess or Not, the user knows the range of the number of turns he/she is limited to. So if the user guesses the exact amount of turns, this is an optimal strategy.

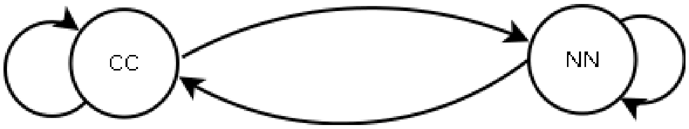


Figure 29 - Always Defect strategy possible moves

- Always Cooperate - In an iterated prisoners dilemma game, as Confess or Not, Cooperation can be a rational strategy (i.e., maximizes payoffs).

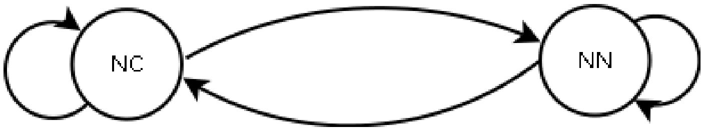


Figure 30 - Always Cooperate strategy possible moves

- Grudger - Begin by Cooperation. Cooperate until the opponent defects, and then defect unforgivingly.

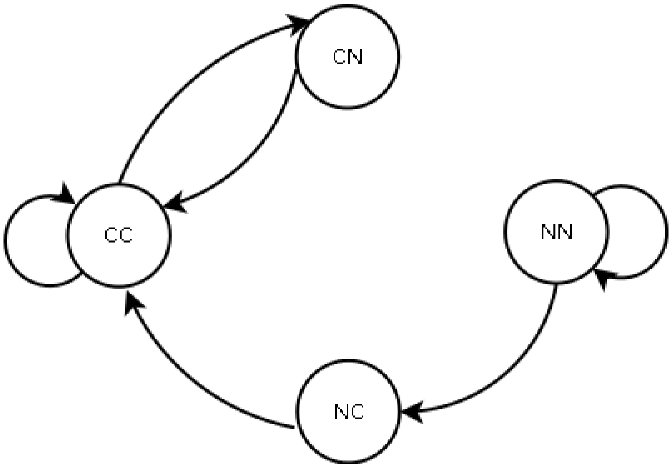


Figure 31 - Grudger strategy possible moves

- Gradual - Co-operates until the opponent defects. In such case, the player defects the total number of times the opponent has defected during the game. Followed up by two co-operations. Note that in this case, a pair of plays such as Not Confess-Confess cannot be repeated as seen in diagram.

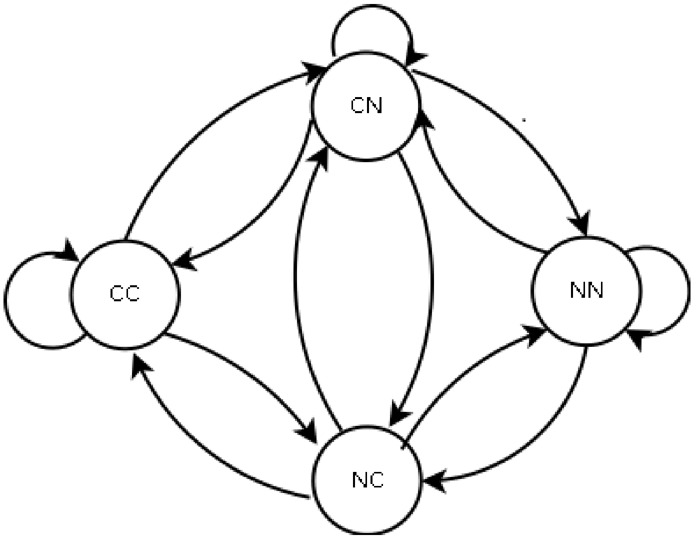


Figure 32 - Gradual strategy possible moves

- Suspicious Tit for Tat - It's the same strategy as Tit For Tat except that it begins by defecting. So it can start with a (player 1/player 2) Confess/Confess or Confess/Not Confess.

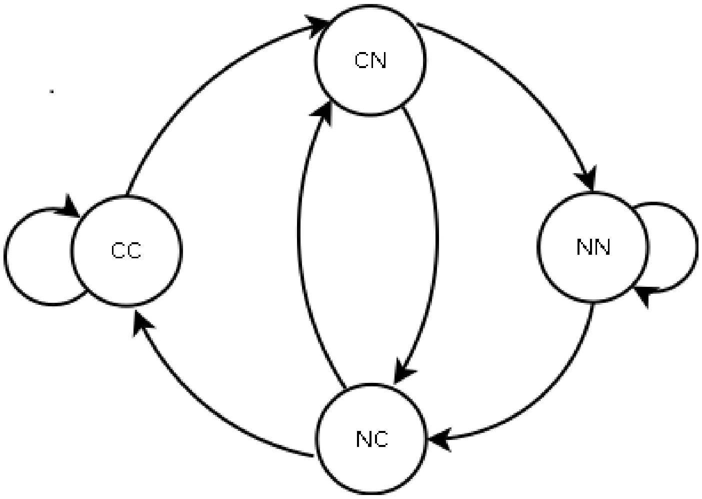


Figure 33 - Suspicious Tit for Tat strategy possible moves

- Remorseful Prober (Tit For Tat with Random Defection) - Repeat opponent's last choice but sometimes probe by defecting instead of co-operating. If the opponent defects in response to probing, then the player will co-operate once.

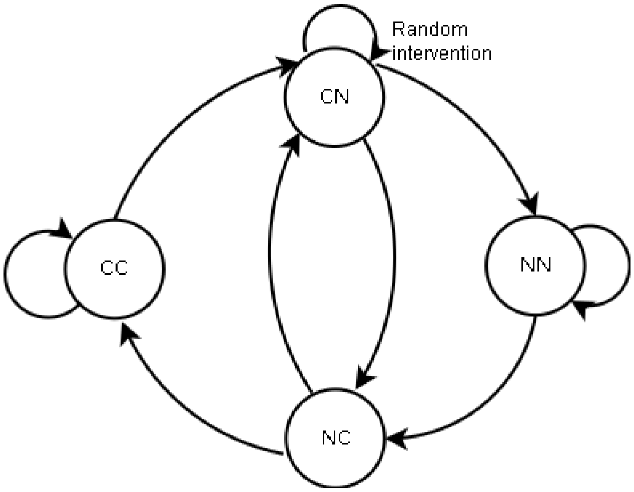
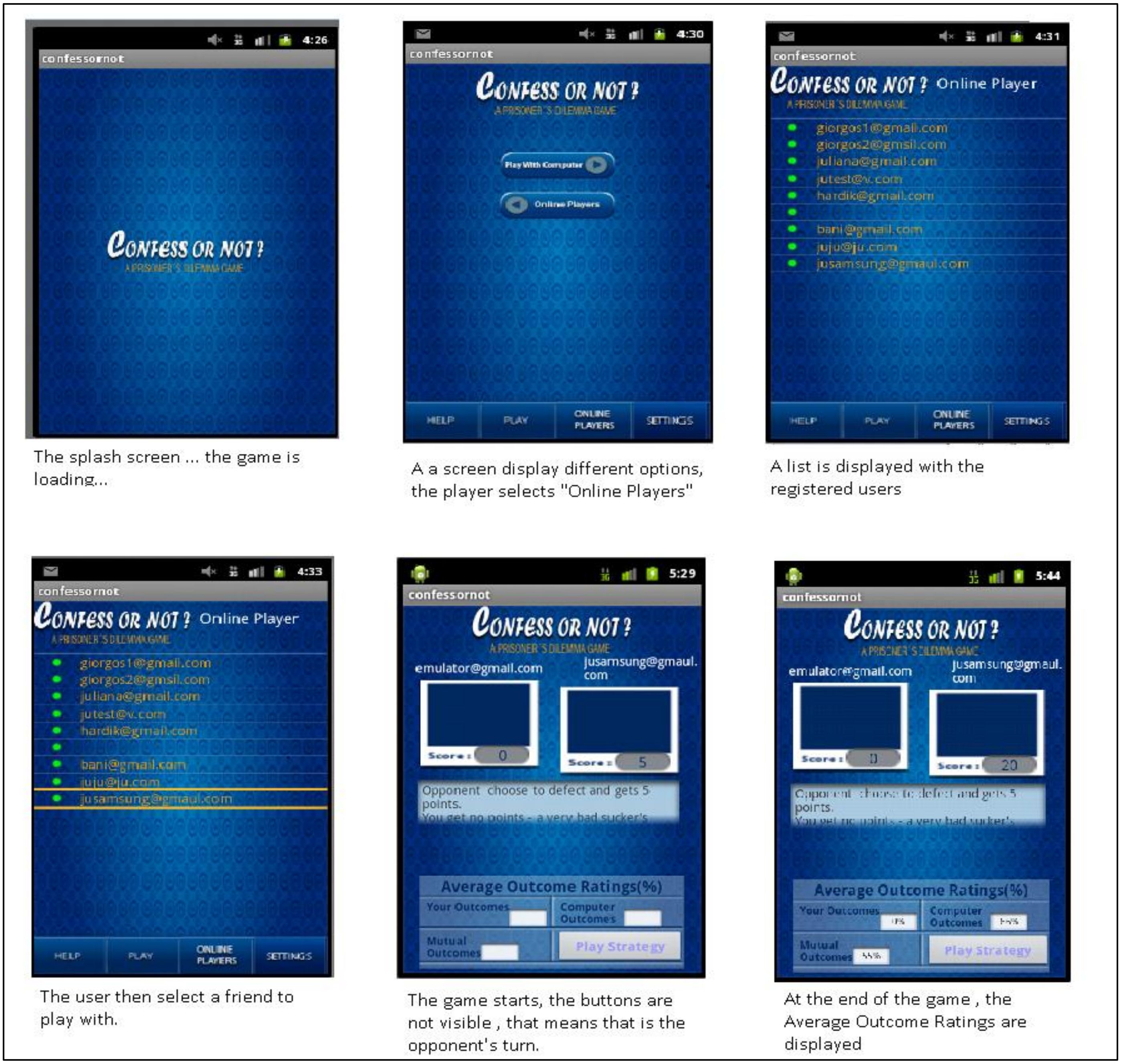


Figure 34 - Remorseful Prober strategy possible moves

5.2 Game Flow

The game flow of the game is illustrated with real game screenshots. In those following screenshots, a Virtual Device emulator (provided with ADT framework) is playing remotely against a phone device (Samsung® Galaxy).

During a game request, if the opponent's device is switched off, the notification will be sent whenever the opponent turns on his device. The bellow example in figure 35 represents the case of the player being the requester for online playing.



giorgos1@gmail.com
giorgos2@gmsil.com
juliana@gmail.com
jutest@v.com
hardik@gmail.com
bani@gmail.com
juju@ju.com
jusamsung@gmaul.com

emulator@gmail.com
jusamsung@gmaul.com

Opponent choose to defect and gets 5 points.
You get no points - a very bad sucker's

The splash screen ... the game is loading...

A a screen display different options, the player selects "Online Players"

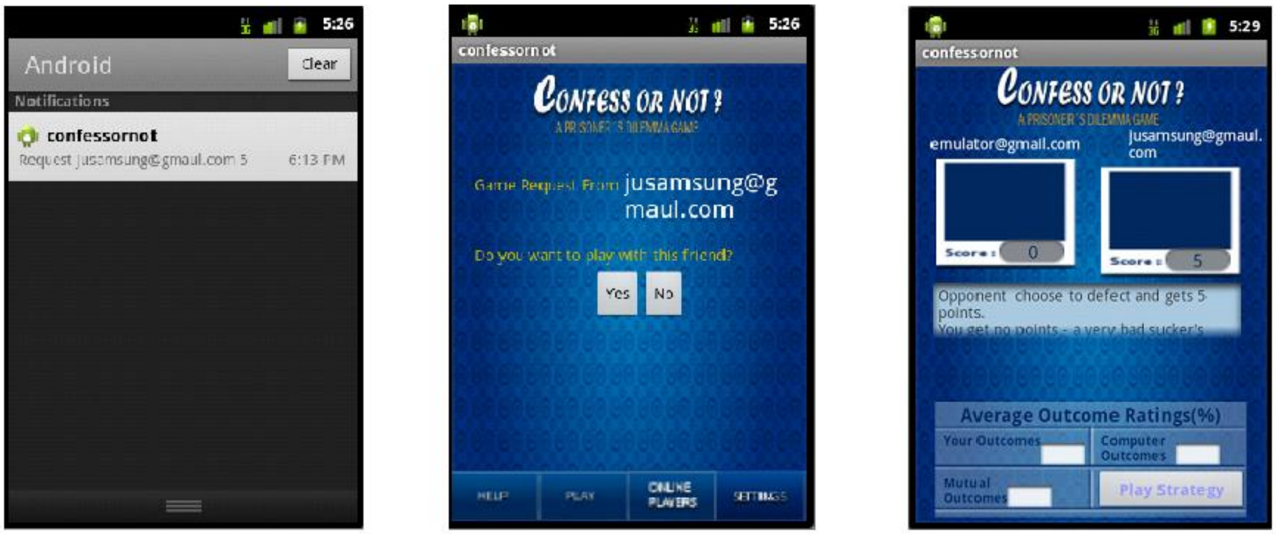
A list is displayed with the registered users

The user then select a friend to play with.

The game starts, the buttons are not visible , that means that is the opponent's turn.

At the end of the game , the Average Outcome Ratings are displayed

Figure 35 - Screenshot of Game Request



Opponent choose to defect and gets 5 points.
You get no points - a very bad sucker's

A game request notification is received on device indicationg that a friend wants to play.

After the user press the notification , a screen dialog is seen.

When the player accepts to play the game interface is seen and both see the same screen.

Figure 36 - Screenshot of receiving game request

In case the device is the request receiver, a notification is shown in the notification field of Android. If the notification message is selected, then the program loads. If the program has been loaded before, it emerges from the background to the front. So after the confirmation of a game request, the game commences.

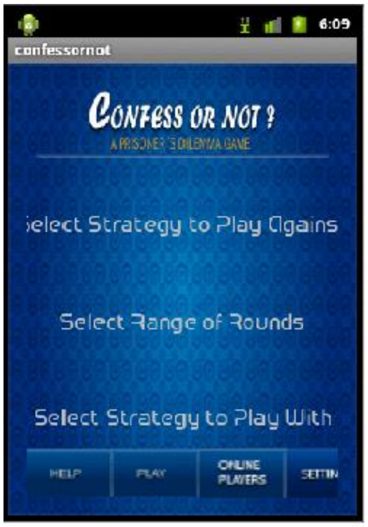

6:09
confessornot
CONFESS OR NOT ?
A PRISONER'S DILEMMA GAME
Select Strategy to Play Agains
Select Range of Rounds
Select Strategy to Play With
HELP
PLAY
ONLINE PLAYERS
SETTIN

Figure 37 - Settings screenshot

During offline gaming, the player selects the button PLAY and then “Play with Computer” and a game screen is displayed for playing as in online gaming. In Settings, the user can select the strategy to play against in offline playing, the range of turns, and the strategy to use when playing. A menu of options is displayed in all cases. When the user selects the strategy to play with, he can play the selected strategy by pressing the button "Play Strategy" in the game interface. The player selections are stored even when the application is switched off, either by killing the application or by turning off the device.

5.3 Design

This section analyzes the server and client interaction at the moment the application is installed and requires registration. In reality, the server files are accessed inside each client's file and they are not accessed during a screen transition.

Both figures show the data exchange between client and server in figure 38. In case the user is already registered, the game skips the login page to game mode. In case a user receives a game request, the client server interaction is depicted in figure 39.

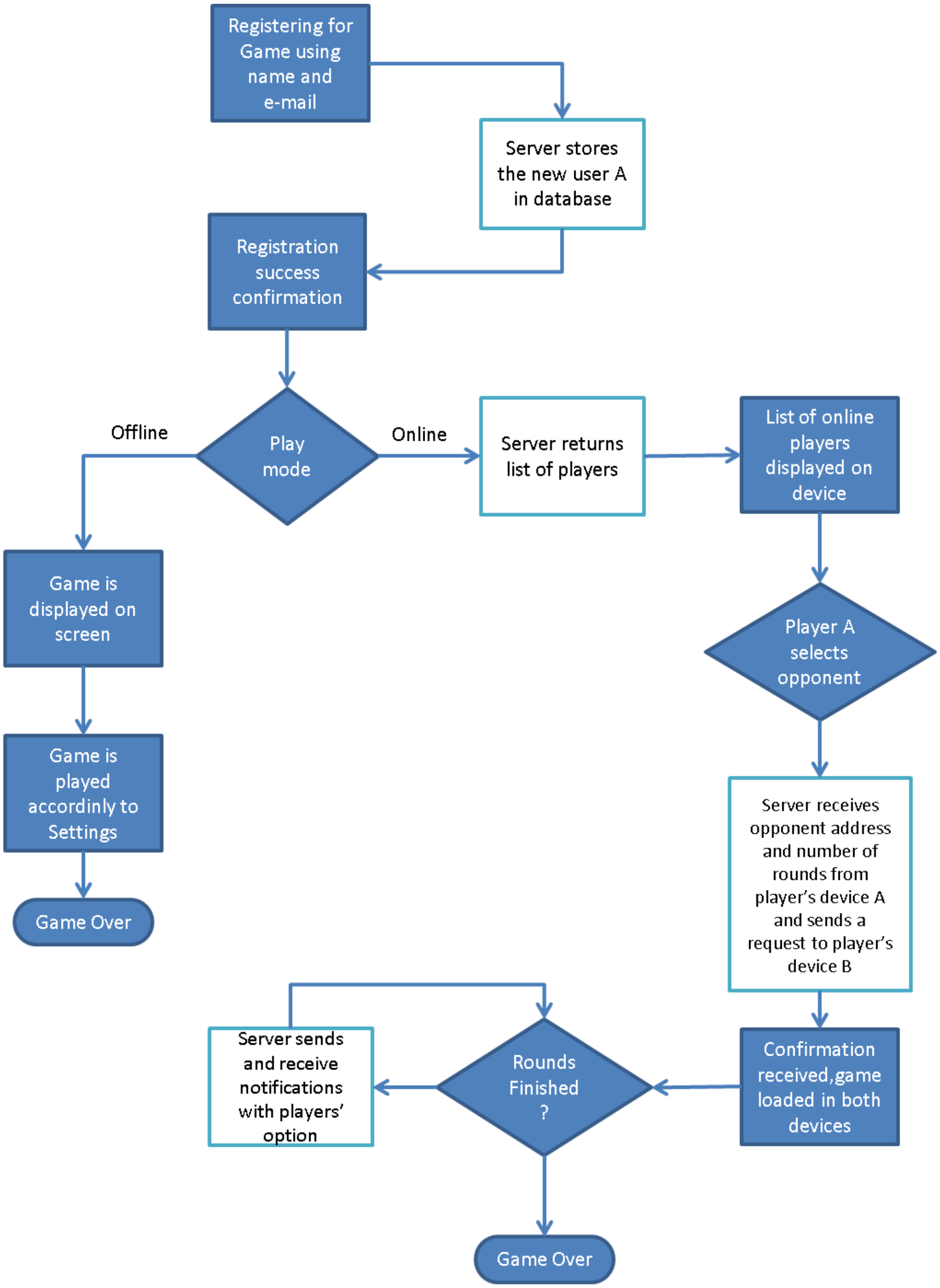

Registering for Game using name and e-mail
Server stores the new user A in database
Registration success confirmation
Play mode
Offline
Online
Server returns list of players
List of online players displayed on device
Game is displayed on screen
Player A selects opponent
Game is played accordinly to Settings
Game Over
Server receives opponent address and number of rounds from player's device A and sends a request to player's device B
Server sends and receive notifications with players' option
Rounds Finished ?
Confirmation received,game loaded in both devices
Game Over

Figure 38 - Client-Server Interaction of Confess or Not Game of a requester

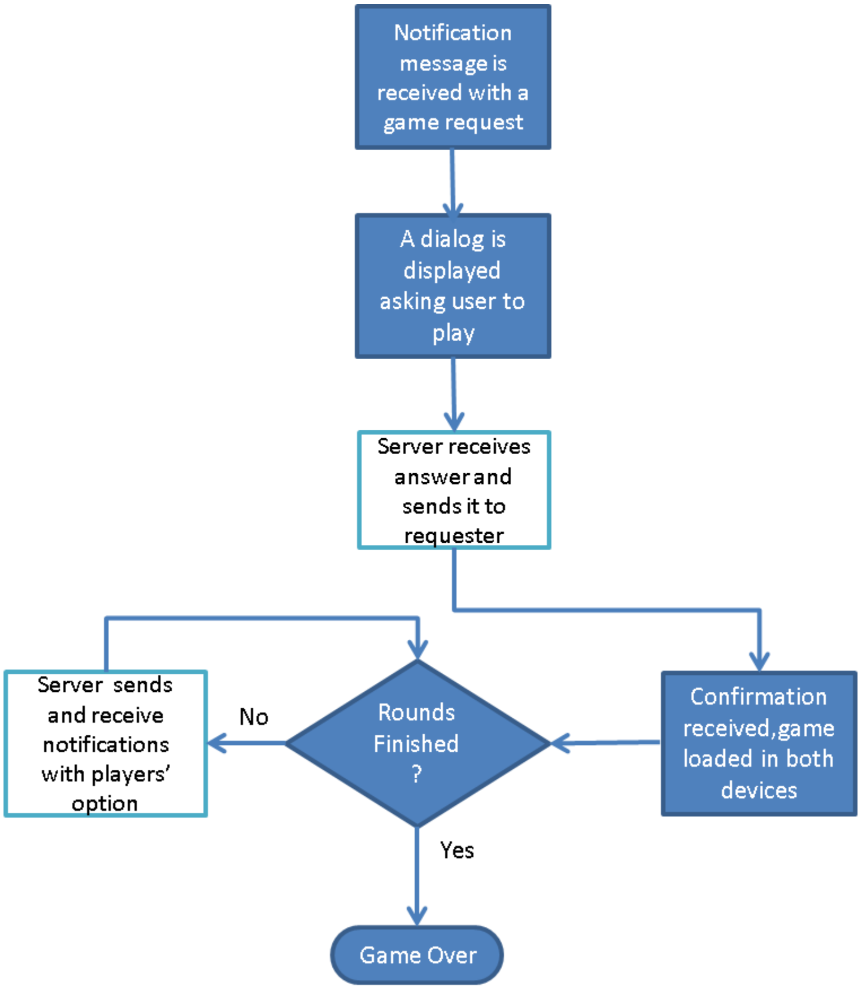

Notification message is received with a game request
A dialog is displayed asking user to play
Server receives answer and sends it to requester
Server sends and receive notifications with players' option
No
Rounds Finished ?
Confirmation received,game loaded in both devices
Yes
Game Over

Figure 39 - Client-Server Interaction of Confess or Not Game of a requested player

5.3.1 Server's Side

This section describes some of the functionalities of each file stored in server. Server files were developed in PHP ("PHP: Hypertext Preprocessor"). This language was originally designed for web development to generate dynamic content to the World Wide Web. A basic object oriented programming style is used for the server.

Posting in PHP is a way of transferring data to a server. For example, in a webpage if a user fills out a name form, the name will be stored in a variable. At the moment the user presses the “ok” button or “enters” in the keyboard, a PHP code reads that variable that the user filled and a “post” to the server is made. The post function is a part of the PHP library.

Also, the data is exchanged in JSON format. JSON is the acronym of JavaScript Object Notation. JSON is a light-weighted format widely used because of its simplicity. An example of a JSON format can be seen bellow:

```
{
 "employees": [
                  { "firstName":"John" , "lastName":"Doe" },
                  { "firstName":"Anna" , "lastName":"Smith" },
                  { "firstName":"Peter" , "lastName":"Jones" }
 ]    }
```

JSON uses JavaScript syntax for describing data objects. Meanwhile, JSON is independently used in any language or platform. JSON parsers and JSON libraries exists for many different programming languages.

This section can be divided into two main categories such as common files and interaction files. The common files are the files that are library files useful for execution of interaction files. The interaction files are responsible to send and receive information from each activity from the client's side. Each of the main files was tested for posting with Ajax JavaScript. An earlier approach for testing was writing some other PHP files and making posts by filling the forms, but Ajax JavaScript was a far better approach.

An important achievement of this part of development was the knowledge that things should be kept simple in development. It is a fact that the amount of online information could misguide a beginner into his journey of learning.

In the following paragraphs, a description of the most important files for the server will be provided. In the implementation section (5.4), a diagram of how all those files are connected will help the understanding of the whole design:

Interaction files:

- **Accept Request:** Confirms the availability of an opponent player during a game the receipt of a game request. Receives as parameters the response of the opponent that accepted to play the game, and his email. Sends through GCM the message to the opponent so that the game can be started. Returns the successful confirmation of the transaction.

- **Get Group List:** Makes an SQL request to display the list of registered users. It returns the list of online players with their credentials.

- **Get Register:** During registration, it checks in database if the user is already registered, if not a new entry is created. This file returns a confirmation of successful registering.

- **Game Platform:** This file is responsible for the message exchange during the game. The choice is sent to the server and the server sends the choice to the mobile through the notification sender function. At every game play for both player and opponent, the GCM registration id is retrieved from the database and a message to the device with the action is sent.

- **Send Request:** This file is accessed when the user picks a name from the list of players. It stores as parameters the game requester's email, the opponent's email and the number of turns calculated in the client. The file checks for opponent's availability and sends the number of turns to the opponent if there is an available one. The file returns a confirmation flag of a successful transaction.

Common files:

- **Webservice file:** The web service file contains the functions that create and manage the Webservice. This is a JSON/PHP web service. By the definition of a web service, the creation of a web service is a creation of a service for integrating web applications. In both "Confess or Not" and "Let's go out" games, the same web service is used. After a research of possible PHP web services servers online and because of the availability of JSON library on Android System, a model for initiating, serving and decoding JSON was needed. Mostly of the core design of the Webservice was based on an example [52].
  The web service object has its own parameters such as the function to be accessed and the JSON variable to be initialized. The main functions of the file are:
    - initialize(): After a Webservice object is created , it must be initialized, i.e. a JSON Webservice object must be created, setting it as a decoder for JSON.
    - serve(): This function decodes the given JSON parameter.
    - register(): Sets a given method to the object.

- **JSON file:** This file has the functions to convert to and from JSON format i.e. it is a JSON parser. This file is taken from the open source community [53]. The type of JSON encoding can be configured as desired. In this application, this file was used as a decoder. The functions are flexible enough to retrieve separated data from a JSON object or a bundle of JSON objects.

- **Configuration file:** Contains all the constants and the database connection links to facilitate the use of same files in another server.

- **GCM file:** It contains the necessary functions to send a notification to a device from the server as long as the registration id is provided as a parameter. It uses the cURL[8] Library that is a library supported by PHP widely used for accessing third party addresses such as the GCM server. Given an address (i.e. the GCM unique id) and a message field, the message can be sent to GCM online server with HTTP protocol.

5.3.2 Client's Side

This section describes the Android classes and it explores the each class' functionalities. Those classes are distinguished as the main activity's classes and library ones. The main activity's classes are classes that hold the code of every function of each activity possibly displayed on screen by the user. The library classes provide all the functions needed for the interaction between the client and the server, for checking connectivity and for other operations that will be described thoroughly. The main activity classes are:

- **Login Activity Class**: Checks the entry given by the user, i.e. the email and password. Also checks if the entry is provided in the right format, and if not provided at all prompts the user to make an entry. Stores email and password to the next activity.

- **Play activity Class:** In the *OnCreate* stage, i.e. the first action of an activity, the function *RegisterGCM()* is called. This function is responsible for the registration on the server. At first, it checks for internet connectivity. Then it retrieves the data from the previous activity, i.e. the email and password. Then it checks the device's compatibility with Google Cloud Messaging (GCM) and confirm the

[8] PHP supports libcurl, a library created by Daniel Stenberg, that allows users to connect and communicate to many different types of servers with many different types of protocols. Libcurl currently supports the http, https, ftp, gopher, telnet, dict, file, and ldap protocols. Libcurl also supports HTTPS certificates, HTTP POST, HTTP PUT, FTP uploading (this can also be done with PHP's ftp extension), HTTP form based upload, proxies, cookies, and user/password authentication, as seen in [66] .

manifest file's (configuration file) proper settings. A receiver is set so the device can receive the unique ID from GCM. The receiver checks if the device is already registered. If not, it receives the unique id. Also, this activity waits for a notification containing a flag from the opponent's device. If this notification comes from a game request, the game's display shifts to *GameRequestReceive* activity. If this notification is a response to a game request, the game's display shifts to *OnlineScoreActivity*.

- This class activates two buttons displayed on screen:
  - ➢ **Play with Computer**: When the button is pressed, the Shared Preferences (from Range Selection in Settings) are acquired, that means that the user's previous choice of the range of possible selections is stored and it is used to select the right range. It is important to note that Shared Preferences are stored variables and those can be retrieved by any activity on the application. For example, if the user selects a range of 5 to 10 rounds, a random number between 5 and 10 is generated. This concept is also applied to all possible ranges that are 5-10, 10-20, 20-40 and 40-60. This generated random number is stored and passed to the Score Activity, which means the game itself.
  - ➢ **Play Online**: When button is pressed it instantly redirects to *Player_Name* activity. This means that a list of existent players is displayed.

- **Player Name Activity Class:** At the moment a player is selected, the number of turns is retrieved randomly accordingly to a range pre-selected in Settings. Then, the user id of the player, the user id of the selected opponent and the number of turns are posted to a server file and a confirmation flag is received. On another thread, the list of possible opponents is displayed to the user before the selection. This list of players is retrieved, if available, and the registered users are requested from the database.

- **Flag Variables Class:** This is a helper class, used for Strategies' round history. Most of the strategies need a previous selection from both players so this class provides an object that stores the player's choice, the game' and a defect flag.

- **Score Activity Class:** This class is the game environment for offline playing, when a player chooses to play with the device. In the *onCreate* stage, *initUI* (initialize Interface) is called and the buttons are activated. The current score is seen at each round. The buttons displayed are:

- **Confess:** When the user selects the confess button, a round is added to a global counter and the player’s choice is stored in a Flag Variable (number 1 in confess case) to be used for strategies. The strategy of the opponent player is retrieved from the Settings.
- **Not Confess:** The same applies in this case, but the value is different for flag (number 2 in not confess case) variable.
- **Random:** The same as the previous buttons but with a random number stored in flag variables. The player can see the choice made by the “random generator” by checking the messages displayed on screen.
- **Play strategy:** The player can play a strategy from the ones already stored in the Settings.

The Score Activity class also contains all the strategies. With each button the player presses the function applies the strategy accordingly if any, stores the current choice for future playing and updates the count of rounds. The game ends when the counted number of rounds reaches the number retrieved from the previous intent from the class Play Activity. Afterwards, a dialog box is displayed with the total points. Under the dialog box the mutual score percentage, the player’s percentage on the game and the opponent’s percentage are displayed on a table.

- **Online Score Activity Class:** This class is the interface for the online playing of both player and opponent.

- **Game Request Receive Activity Class:** This class manages the notification received when a player chooses an opponent to play against. If an opponent is available, the opponent will receive a notification for a game request. There are two buttons displayed, yes or no. If the opponent chooses yes, then a flag that shows who is the receiver of the notification is sent to the next activity, i.e. the game (Online Score Activity). This flag comes from the server that identifies the sender and receiver.

- **GCMIntent Service Class:** This class holds the function that manages the notification service. This service delivers the messages necessary to login and to player’s selection, and any possible message between the devices and the server. An important function in this class is the function *isRegistered(),* that is called whenever a user needs to register their username on the as well as register their device on the GCM server. As explained previously in GCM functionality in section 4.7, a device must be checked if registered on GCM server in order to register any user of the game. This check is conducted within in this function.

- **Settings Activity Class:** This is the activity where the user can choose the desired range of the game's turns, the opponent's strategy in the case of offline playing and the player's own strategy to play against an opponent. All the settings are stored as Shared Preferences.

- **Application Utilities Class:** This is a configuration file that has the necessary URLs of files in the server. Additionally it holds some functions that are helpful for retrieving data and also some login support functions.

The library classes are used in both games; a description of those classes will be presented in the following paragraphs and the same applies for the "Let's go out" design. Some of those classes were inspired from a very helpful tutorial, especially in the beginning of the implementation [54]. This tutorial explains how to login in a server using Android. This tutorial proved to be very useful and offered some strong background. Each class is described as follows:

- **Connection Detector Class**: This class contains the functions that check for user's connectivity on the internet.

- **JSON Parser Class**: There are a lot of examples of JSON parser online, this JSON parser was based on an example seen on [54]. This class contains the functions necessary to send or receive data in JSON format. The POST functions are request methods supported by the HTTP protocol used by the World Wide Web.

- **Server Utilities Class**: Contains the basic necessary registering class to pair the device with the server information. Also this file contains an unregistering function and a post function to the server.

- **Wake Locker Class**: Contains the necessary functions to "wake up" a device when the device is idle. A power manager class is invoked.

5.4 Implementation

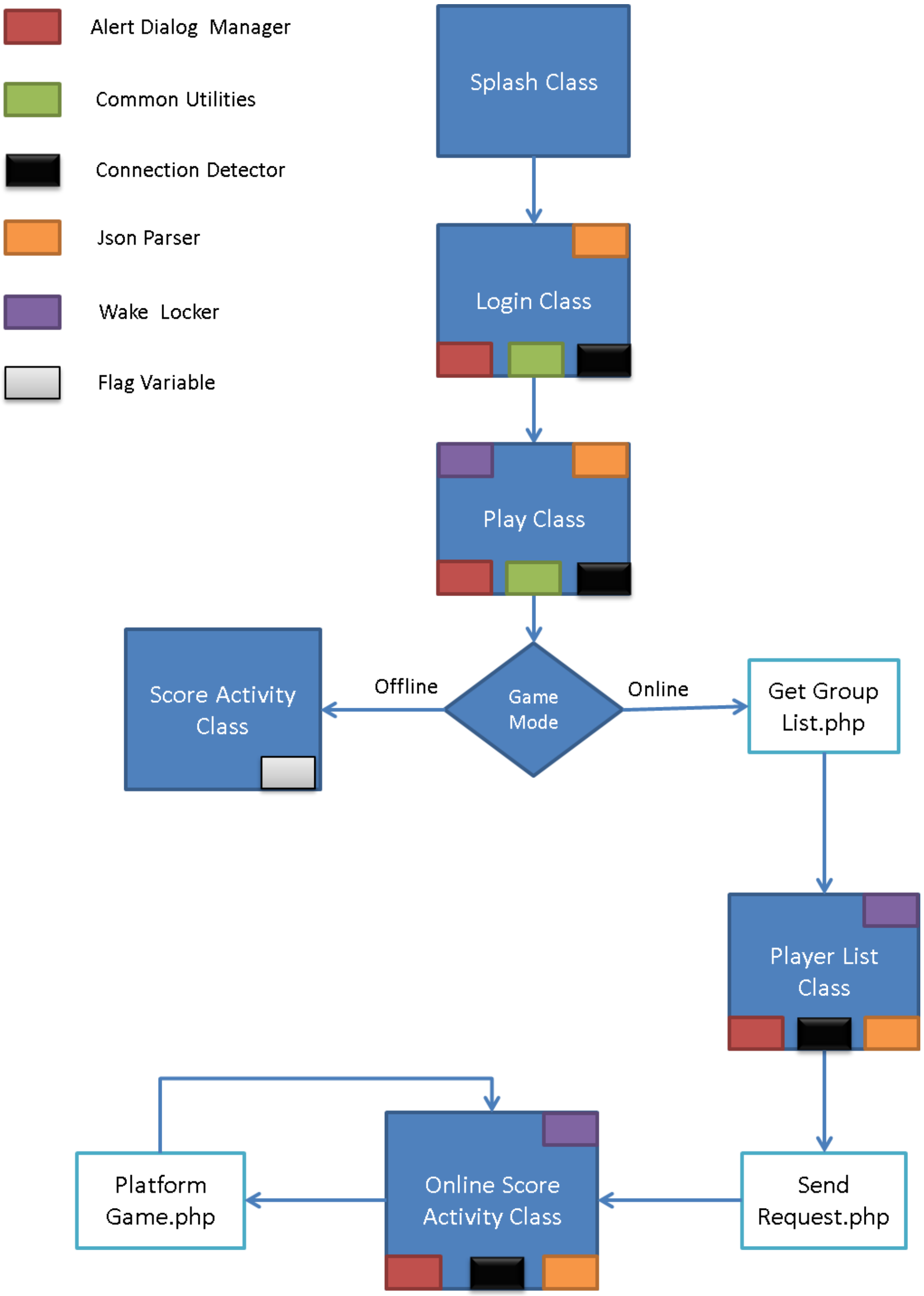


Figure 40 - Interaction of files of Confess or Not for the requester

The above diagram describes the interaction of the files with the helper classes. The blue boxes are client files, and the white boxes are server files. As mentioned before, in case the user is already registered, the Splash Screen skips registration. All necessary information for the registration is stored in Shared Preferences. The following diagram describes the file interaction from the requested player's side. The GCM Intent Service class manages the income messages, so that the device is ready to receive a Confess or Not notification. The Request and Receive Class is called after the notification message to the recipient player is displayed. After that, the game follows similar steps with the ones depicted in the previous diagram.

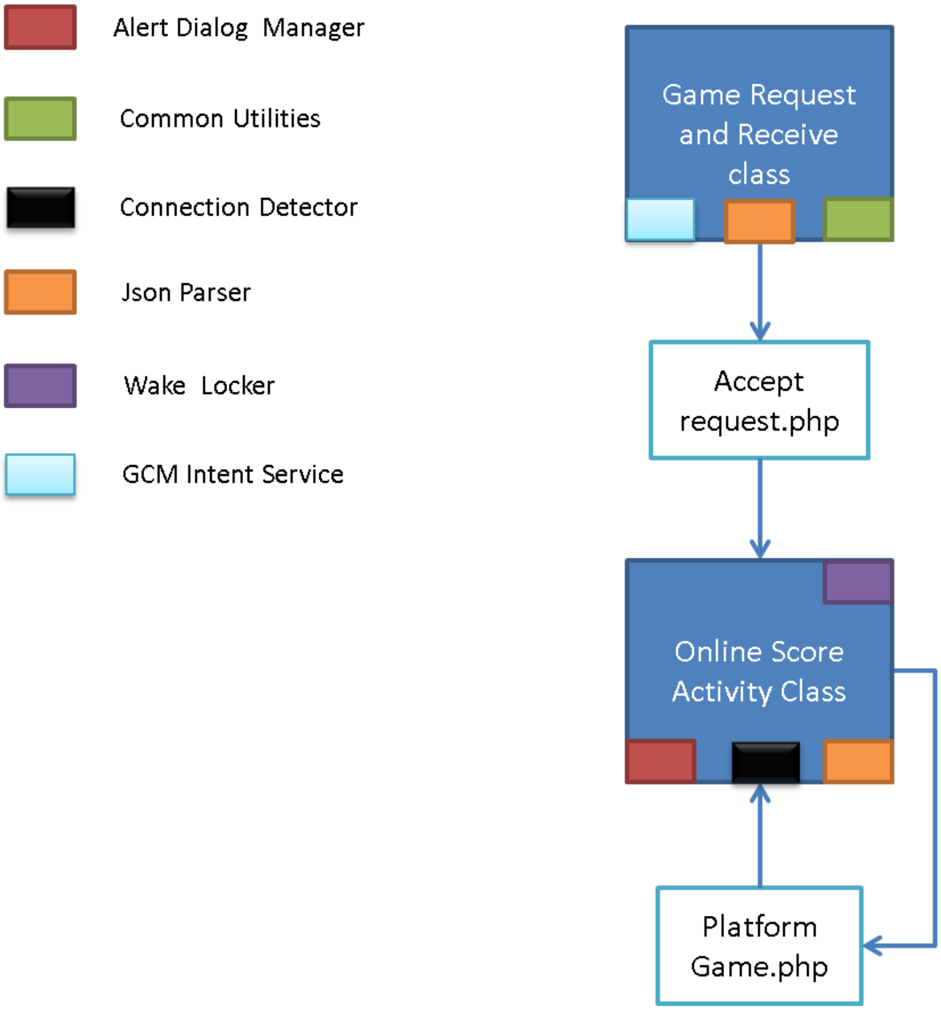


Figure 41 - Interaction of files of Confess or Not for the requested player

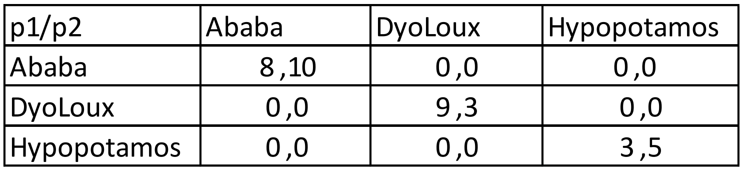
Each server file was tested separately with Java Script. An initial approach from the project's client side was to design a complete module with all the necessary logic for the game. This proved to be inefficient because the handling of separated threads was impossible. This might could have been achieved, if it was possible to have a serial execution of Android client, but that is not the case. Also, in Android OS there are no global variables, so data has to be transferred from an activity to subsequent one. Therefore, the game logic is split through the files and data is passed through intents.

6) “Lets Go Out” Design

This chapter describes the “Let’s go Out” game’s functionality. It begins with a general overview of the game. In the section 6.2., the flow of the game is described. The section 6.3 contains the description of the classes and methods used in both server’s and client’s side. The section 6.4 is composed of the implementation’s diagrams with the interaction of “Let’s go Out” game’s files.

6.1 Description

The goal of “Let’s go Out” application is to simulate an invitation to a friend to go out to a place of their common choice, based on their evaluation on those places. After the first login, the player can invite a registered friend to go out. Online friends are displayed in a list. After both players have chosen the desired places to go out, a payoff table is displayed with both preferences. A brief description of “Let’s go Out” was already provided in section 1.3.



p1/p2	Ababa	DyoLoux	Hypopotamos
Ababa	8 ,10	0 ,0	0 ,0
DyoLoux	0 ,0	9 ,3	0 ,0
Hypopotamos	0 ,0	0 ,0	3 ,5

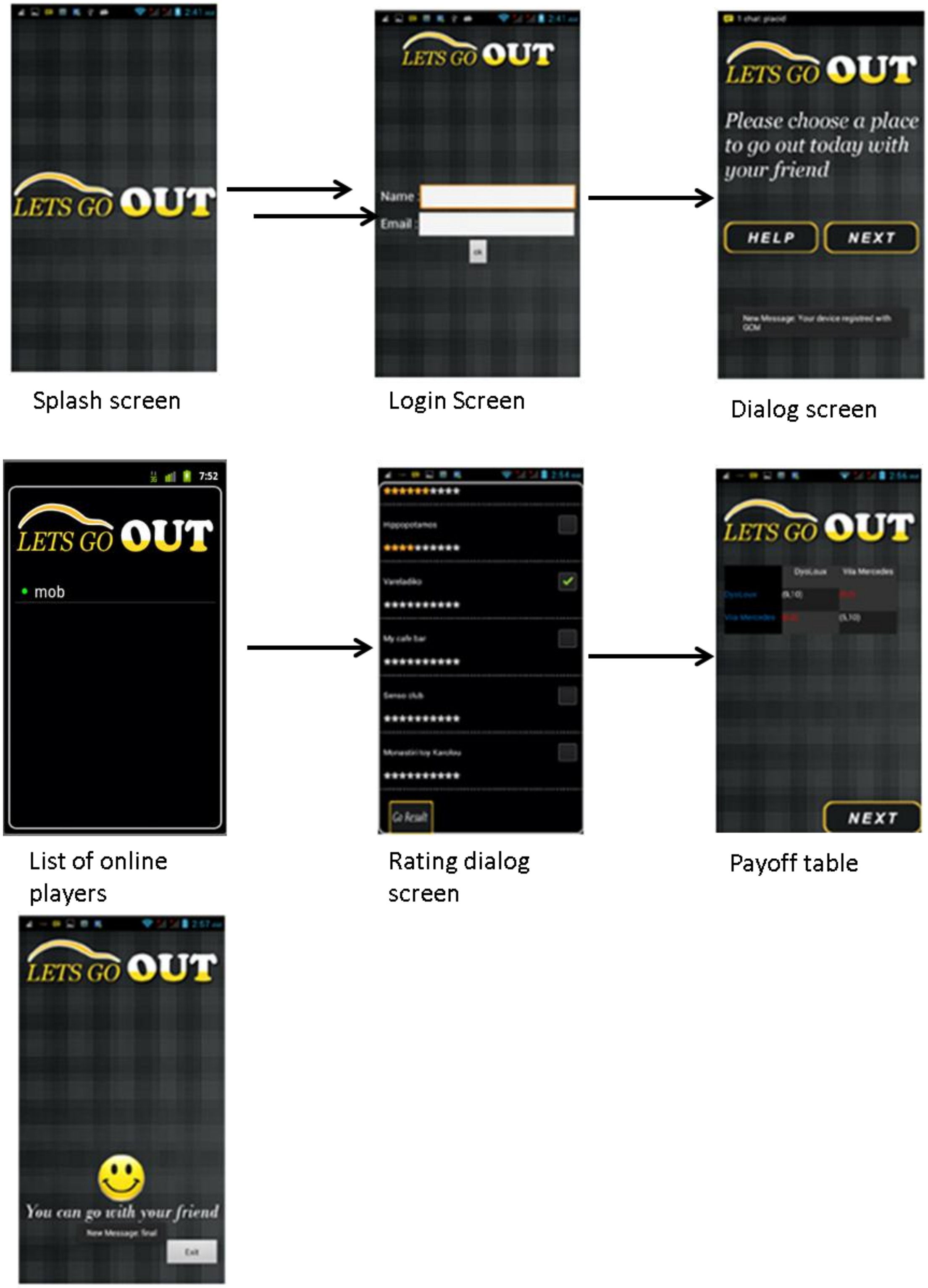
Figure 42 - Let's go out example payoff table

This table is based on the Battle of The Sexes game that is already described in section 2.4. A similarity is that both games present each player’s preferences in a payoff table. In the payoff table, the preferences are displayed diagonally because the intention is to go out together. That means that for non-matching choices, the values are zeroes. The difference from a typical Battle of The Sexes game is that there can be more than two places to go in “Let’s go out” i.e there are more than two strategies in the game. A user may select a place that combines their best possible own preference and their friend’s preference, because probably it would be a common choice in the end. After the payoff table is displayed for both players, they will choose the place that they would go out based on their preferences. If the place is a match, the result of the preferences is displayed for both players.

6.2 Game Flow

The game flow in this section is illustrated with game screen shots. After installing the application, the registration is made by filling a form, just like in “Confess or Not” game. Similarly to “Confess or Not game”, there are two possible types of players, the requested player and the requester. Let’s say that a player A requests player B to go out. There are three possible situations involving the player’s choices:

- Both players match all the places to go out together. After the matrix appearance, if their final choices are common, they will go out.
- Players match some places, so the intersection of their choices will appear on screen.
- Players don’t match any place to go out after the ratings (figure 43 Rating Dialog Screen) and a “You won’t go out today” dialog will appear.



Please choose a place to go out today with your friend

New Message: Your device registered with GCM

	DyoLoux	Vila Mercedes
DyoLoux	(5,10)	[illegible]
Vila Mercedes	[illegible]	(5,10)

You can go with your friend

New Message: final

Splash screen

Login Screen

Dialog screen

List of online players

Rating dialog screen

Payoff table

Choices Matched

Figure 43 - Screenshot of “Let's go Out”

6.3 Design

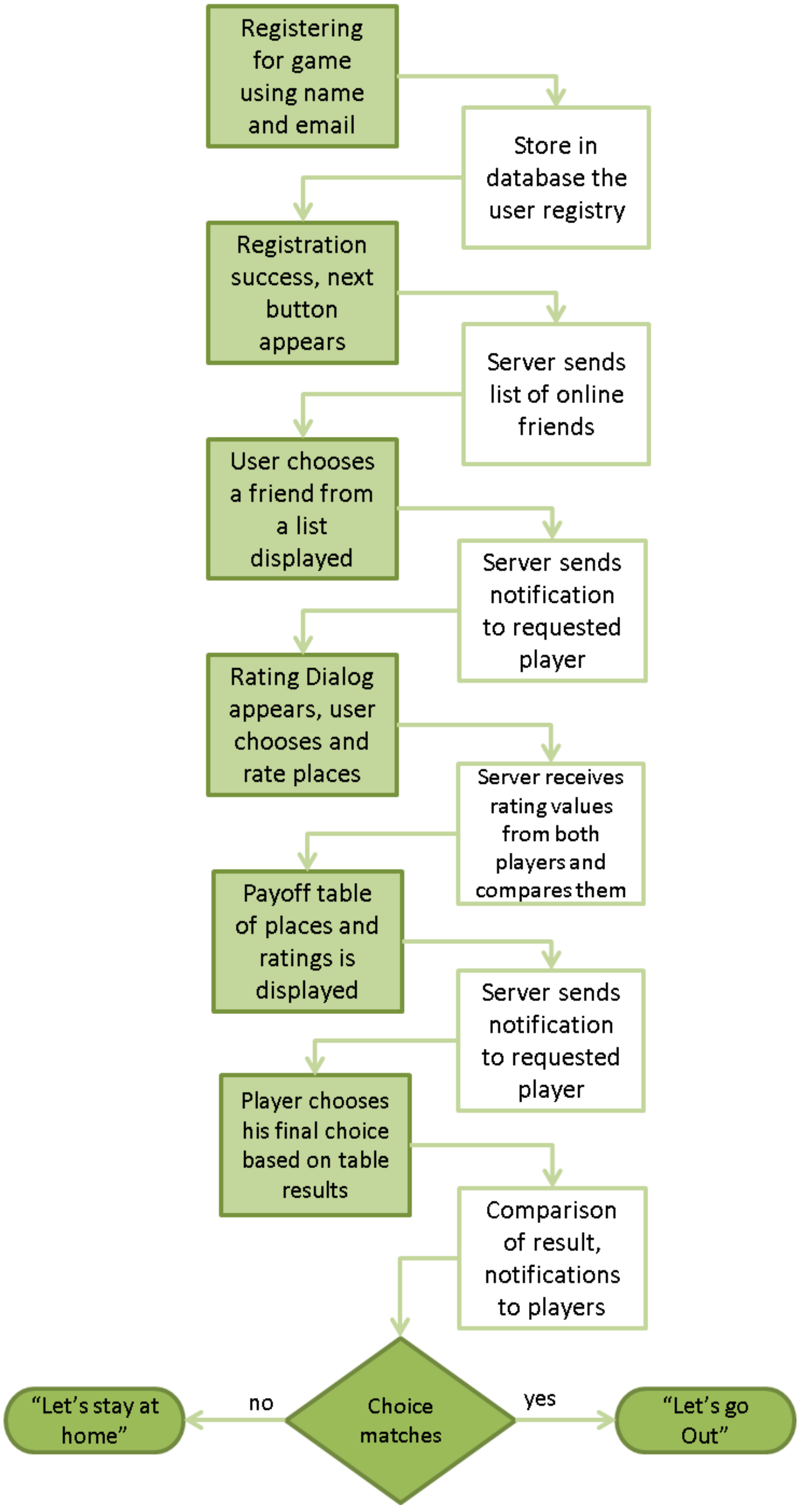

Registering for game using name and email
Store in database the user registry
Registration success, next button appears
Server sends list of online friends
User chooses a friend from a list displayed
Server sends notification to requested player
Rating Dialog appears, user chooses and rate places
Server receives rating values from both players and compares them
Payoff table of places and ratings is displayed
Server sends notification to requested player
Player chooses his final choice based on table results
Comparison of result, notifications to players
Choice matches
no
yes
"Let's stay at home"
"Let's go Out"

Figure 44 - "Let's go Out" client server interaction

This section describes the file server interaction and contains a description of each of the most important files in the Application. Figure 44 provides a server and client interaction outline for the moment a user first installs the application and registration is required. The darker green boxes are client's actions. The lighter boxes are the server's actions. As mentioned in "Confess or Not" (section 5.3), the server files are accessed inside each client's file and they are not accessed during a screen transition.

6.3.1 Server's Side

This section describes some of the functionalities of each file stored in the server. The architecture is similar to "Confess or Not" Game. Server files were developed in PHP as described in section 5.3.1.

In the case of "Let's go Out", all data is exchanged in JSON format as well as in "Confess or Not". Following the example of "Confess or Not" client-server communication, the pack of files is also divided in two main categories: common files and interaction files. The common files are the files that are library files useful for the execution of the interaction files. The interaction files are responsible to send and receive information from each activity from the client's side. Each of the main files was tested for posting with Ajax JavaScript.

In the following paragraphs, a description of the server's most important files is provided. In the implementation section (6.4), a diagram of how all those files are connected is helpful to the understanding of the whole design.

Interaction files:

- **Delete Record:** This file erases all previous entries in the database that have been used during a game session.

- **Registration:** During the registration, checks in database if the user is already registered, if not a new entry is done. This file returns a confirmation of successful registering.

- **Select User:** After the player selects the name and send notification request to the friend. This file receives as parameters the user's id and name and return the GCM number of the friend's device.

- **User list:** This file retrieves the online users from the database, stores them to a user's list and sends this list through a notification to the requester.

- **Send Choice:** This file takes the rating choices of the requester, decodes each result from JSON format and inserts it into the database. The requester's client fills the table of the player's ratings. Then it sends a notification to the opponent (the requested) as a confirmation.

- **Send Choice 2:** This file has similar function to the "Send choice" file but handles the ratings of the requested player. The requested player's client updates the table of player's rating choices.

- **Show Table:** This file returns the list of common choices and ratings.

- **Send final choice:** This file receives the requester's final choice and checks for a match with the friend's option.

- **Send final choice 2:** This file receives the requested player's final choice and checks for a match with friend's option.

- **Request Response:** This file receives the friend's id from client and sends a confirmation to the device using the GCM id retrieved from the database.

The Common files in "Let's go Out" game, i.e. the files that are necessary for the Web services, are the same files used in "Confess or Not" game. Those files were described in section 5.3.1 of this document.

6.3.2 Client's Side

This section describes the Android classes and functionalities of each class .Those classes can be classified as the main activities classes, helper classes and library classes. The main activities classes are classes that have the code for all functions of each activity possible seen on screen by user. The library classes provides all the functions needed for the interaction between client and server, checking connectivity and other functions that will be described thoroughly.

- **Login Activity Class:** This class checks for internet connection and take the login details from user. It checks for entry in right format and stores email and password to next activity.

- **Screen 2 Activity Class:** This class displays a dialog, a next button on the foreground and also displays a progress bar in case of delay. In the background, it registers the device to GCM server, and stores the unique ID and clear previous game entries in database.

- **User list Activity Class:** This class displays the friend's list in the foreground. In different thread, it runs the player's friend selection. When the player selects a friend to play, the request is sent and the activity waits to the friend's response. While the player waits for the confirmation, a progress bar is also displayed.

- **Rating Dialog Activity Class:** This class displays the rating dialog and stores the user preferences and ratings. When the user presses the "Go toResult" button, the choices are stored in a *hashMap*[9] instance variable so then can be sent as a POST to server. The server then stores the results for both users in database. The next activity starts when a confirmation (notification) that the user's friend has finished his choices is received.

- **Show Table Activity Class:** The stored user's and friend's preferences and ratings are retrieved from database and encoded to a hashMap instance. This instance's data is used to allocate the rows dynamically in a payoff table. This is executed on postExecute activity cycle because, at that moment, the data for allocating the rows would be available. In case of delay, a progress dialog is displayed.

- **Common Choice Activity Class:** This class displays the common choices of players, i.e. the same places that are displayed in the payoff table. At the moment a user chooses the place to go, by pressing the place's name, the choice is sent to server for comparison with the friend's choice. If the player is the requester, he waits for the opponent's final choice. For the final result, both players wait for confirmation (notification).

[9] A collection class that maps Java Object's indexes.

- **Request Confirmation Activity:** This class is loaded when the player is the requested, at the moment he presses the notification request. The message in notification contains the requester's id**.** It loads a dialog for the player to choose between accepting and rejecting his friend's request. After the acceptance, the client sends his own id and the requester friend's id to server and sends confirmation (notification) back to his friend.

- **GCMIntent Service Class:** This class has the function that manages the notification service. This service delivers the messages necessary to login, select player and any possible message between the devices and server. An important function in this class is the function isRegistered() that is called whenever a user needs to register his username on server as well as register his device on GCM server. As explained previously in GCM functioning in section 4.7,a device must be checked if registered on GCM server in order to register any user of game. This check is made in this function.
- **Application Utilities class:** This is a configuration file that has the necessary URLs of files in server .And also have some functions that are helpful for retrieving data and login support functions.

The library classes are used in both games, a description of those classes can be found in section 5.3.1.

6.4 Implementation

Figure 41 describes the file interaction of the requester, and figure 43 describes the file interaction of the requested player. The green boxes are client’s files, and the white boxes are server’s files. In reality, the PHP files were accessed inside the class and not during the transition between the activities. However, the diagrams show the PHP files being accessed during the transition of the files to represent the data exchange between them.

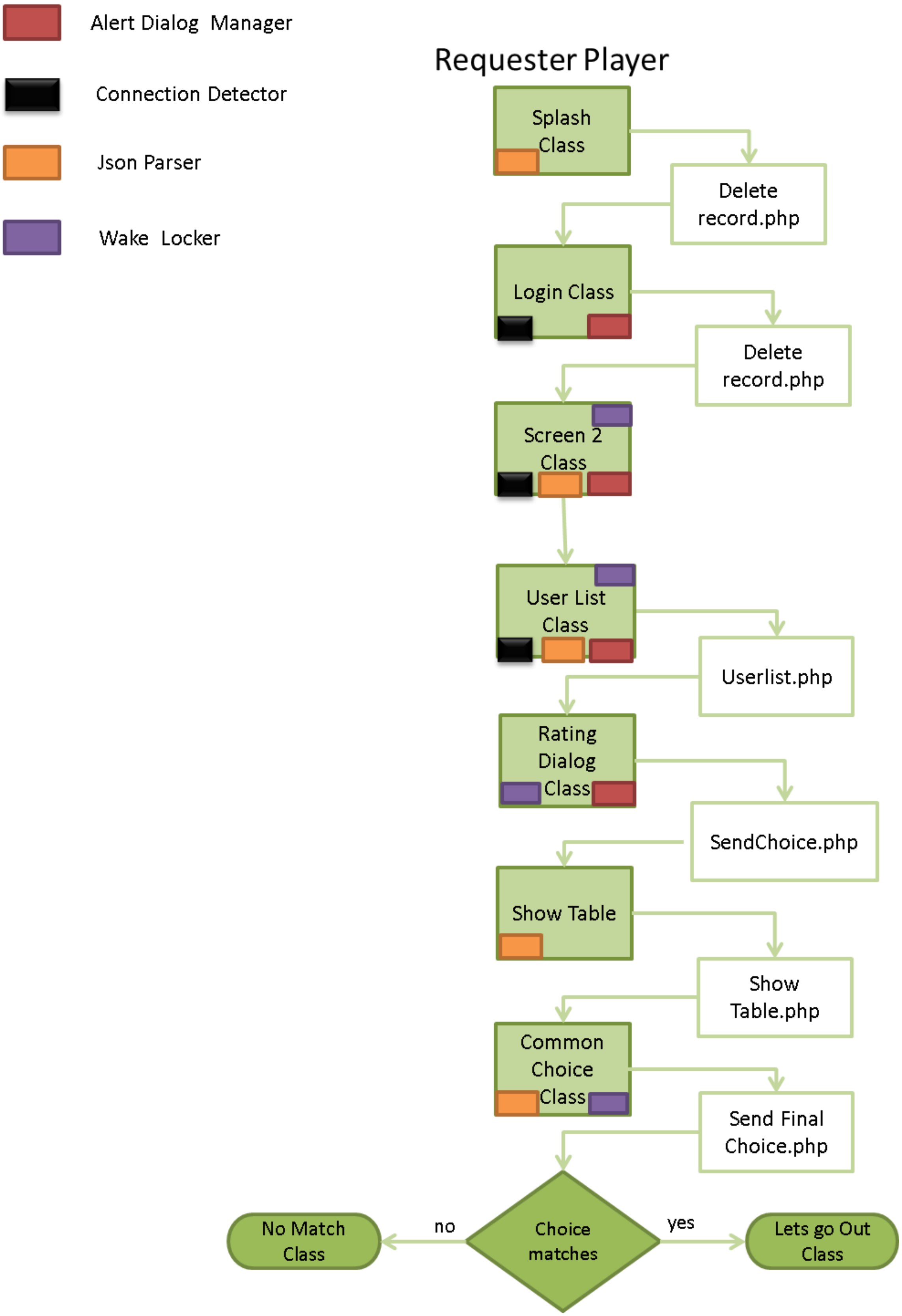


Figure 45 - “Let's go Out” file interaction - requester

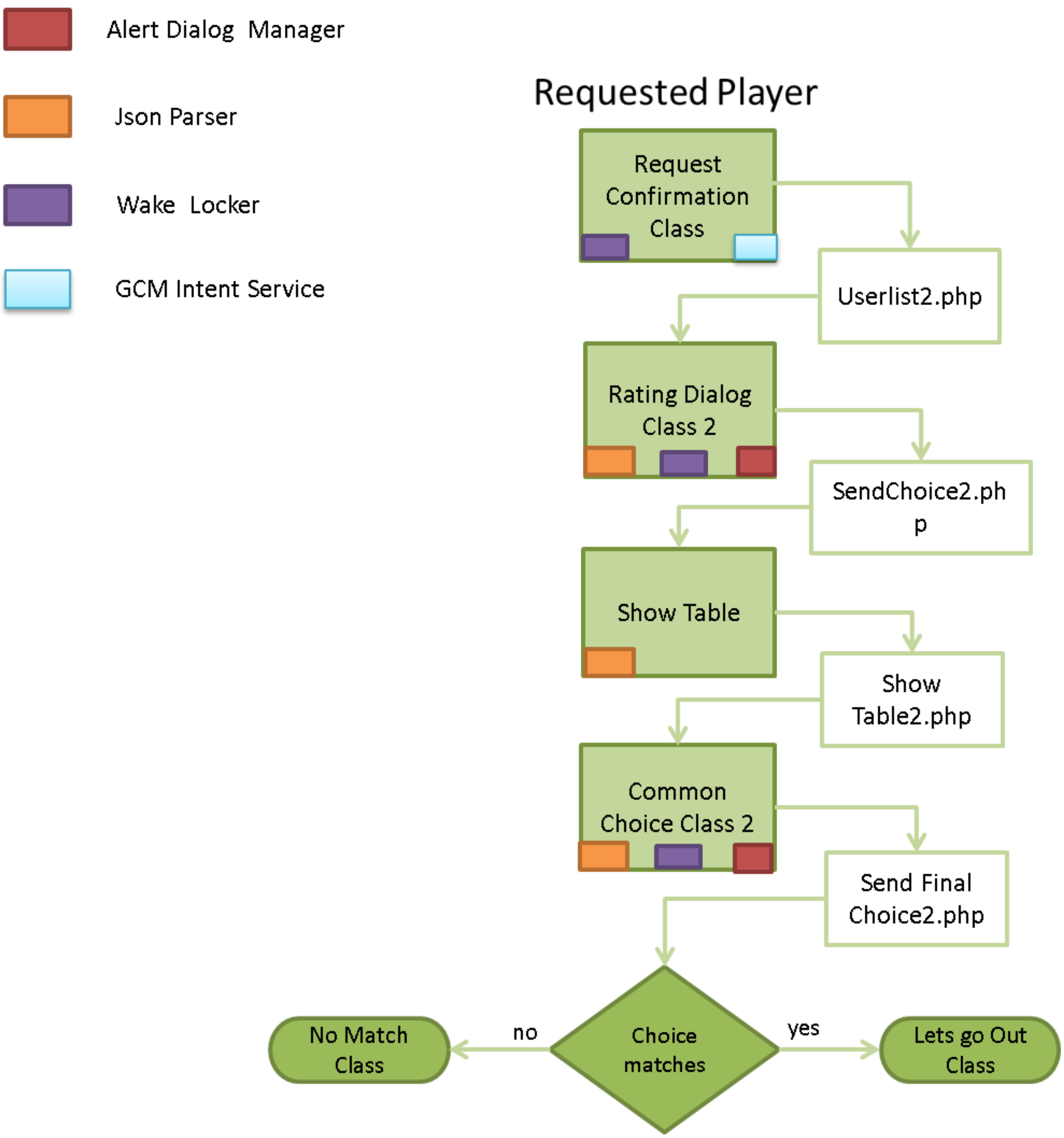


Figure 46 - “Let's go Out” file interaction - requested player

7) Conclusion and future work

Android Development is a rapidly evolving field. During the projects' development, more features were added to Android OS by its providers. The Android community is a dynamic one and because it is an open source platform, there are many volunteers reporting their problems and development ideas. This is a great hand for every newcomer to practically apply any of their idea. Google also provides a great resource of information on the official Android site.

On the journey of learning, a beginner developer on this field can have plenty of experiences; some of them may prove to be encouraging while others disappointing. The key is to understand the functionality of Android OS and its structure from the beginning. A mistake a beginner usually makes is start programming using fragments of knowledge without the deep understanding it requires. The experience of programming on Android showed that an efficient design is valuable at the very beginning of development. An efficient design in Android, first of all, is completely directed towards usability. A developer has to be aware that his application can be accessed through various devices of different sizes, so the size of a button and/or distance between buttons or widgets play a significant role.

We are still in an intermediate era of mobile usage and development, if we make a comparison of how many possibilities lie ahead. Developing of this project led to the creation of a successful client-server communication with the assistance of a PHP server and Goggle Cloud Messaging. The result achieved was a simple and stable communication between two devices, so important to game playability.

The methods and procedures used for client-server communication, stand as a solid example for other games in Game Theory, with two players and two strategies such as in the case of "Confess or Not" game, or involving more strategies such as in the "Let's go out" game. The structure of an Android application, such as the fact that each screen corresponds to an independent activity was both an advantage and a disadvantage in some aspects. The fact that each screen can be developed as a different module facilitates interface's editing, message sending to the server and activity debugging. But this also means that data exchange between modules (activities) have some special procedures, such as for sending data inside intents, using a singleton[10] class, using application singleton, static fields and hash Map. Also data can

[10] In software engineering, the singleton pattern is a design pattern that restricts the instantiation of a class to one object.

be shared through persist objects such as SQLite, Shared Preferences, file, etc. In the projects, some of those methods were used accordingly to necessity.

Regarding Android Applications’ interface, it is important that some of its aspects adapt to the Android needs. Android has a limited processor capability but much more demanding users. Mobile application's users prioritize the quickness of an application’s execution. This fact, looking back after concluding both projects, was quite a challenge. Reminding that the slightest change in the structure of an application can be quite costly if not properly designed. Every application's activity can be a set of asynchronous tasks, background threads and services. All those must operate in agreement with the Android activity cycle.

Those games are almost ready to be uploaded to Google Play Store. However there are some security issues that must be evaluated prior to uploading to the online community. For example, all users have access to any online user and this implies that loads of notifications can be sent to anyone. A future idea that could be a solution for this kind of problem is dividing the users in close groups of friends. The access to the group would be restricted to unknown users. A search option for finding friends via email or user name could be added. Another option for having only the desired friends in the list of friends to play, could be implemented with the program searching for Contacts stored in the device and checking if those contacts have installed the application. Many widely known chat applications, such as WhatsApp© uses the contacts already stored in the device. Those differentiations certainly require a more profound server implementation and several changes in the client interface. These could be applied for both games.

For future work ideas on "Confess or Not", it would be very useful to keep a log of the played games and maybe store them to study the tendency of human player's choices and winning strategies. The inclusion of more strategies of Axelrod Tournament would give the user an even better experience in playing. Also, it would be very interesting if a user could edit his strategy by changing some parameters such as the probability of a strategy[11] repetition, the number of consecutives repetitions of a strategy or the duration of a strategy.

Regarding the "Let’s go Out" application it would be useful if the friends were loaded from the user’s Facebook© contacts. This can be achieved with a new tool offered by Facebook [55]. Also login could be made using a Facebook account. A statistics screen could be added with the friend’s previous choices. A user could also add a place of his

[11] Strategy in this case means as defined in game theory, i.e. any of the options a player can choose.

choice. The information about the places on a user’s list of places could be loaded from third part website or application, such as Google© Maps [56].

8) Related Work

There are some related work concerning Game Theory and the possible applications involving this field:

- Game Theory for education purpose:
    - ○ **Math Jump, from concept to a published title:** This work presents an overview of game theory, gamification, and business opportunity as the reasoning for the development of Math Jump, an educational mobile game **[57]**.
    - ○ **Mobile Applications: Games that Transform Education:** A mobile app designed for the iPhone and Android operating systems that focuses on teaching players SAT I math concepts using Nintendo's wildly popular Pokemon game design model [58].

- Prisoner's Dilemma application:
    - ○ **A Large Scale, Distributed, Iterated Prisoner's Dilemma Simulation:** This work is concerned with implementing a large simulation of mobile IPD players, across a network of machines **[59]**.
    - ○ **An online applet that plays Prisoner's Dilemma** as seen in [60]**.**

- Online software tools for Game Theory:
    - ○ **Gambit:** It is an open-source collection of tools for doing computation in game theory. With Gambit, a user can build, analyze, and explore game models [61].
    - ○ **Gamut:** It is a suite of game generators designated for testing game-theoretic algorithms [62].
    - ○ **Game Theory Solving Applet:** This applet is designed to find Nash Equilibria in a simple game theory problem [63].

- Online downloadable mobile applications:
    - ○ **Graphical simulation of the spatial iterated prisoner's dilemma:** This mobile application demonstrates the spread of 'altruism' and 'exploitation for personal gain' in an interacting population of individuals learning from each other by experience.

- **Youprisoner:** A simple application that plays prisoner’s dilemma against a Grim strategy.

There aren’t any mobile applications in any of the biggest online stores such as Google Play, Amazon, Windows Mobile Store and even iTunes (iPhone applications) that have a similar educational game-theoretic game as the projects described in this document.

The use of the combination of webservices and Android development can be found in the essay **“RESTful Mobile Application for Android : Mobile Version of Inspectera Online”**. This master’s thesis designs and develops a Web service-based mobile application for Android platform following the constraints of REST architectural style [64].

9) Bibliography

[1] D. Hardawar, "The magic moment: Smartphones now," [Online]. Available: http://venturebeat.com. [Accessed june 2014].

[2] Google, "Android Developer," google, [Online]. Available: http://developer.android.com. [Accessed june 2014].

[3] Google, "Google ADT," [Online]. Available: http://developer.android.com/tools/help/adt.html. [Accessed june 2014].

[4] Netbeans, "Download NetBeans IDE 8.0," Oracle, [Online]. Available: https://netbeans.org/downloads/. [Accessed 2014 june].

[5] Go Daddy , "Go Daddy Webhost," Go Daddy, [Online]. Available: www.godaddy.com. [Accessed june 2014].

[6] Apache, "XAMMP Download," [Online]. Available: https://www.apachefriends.org/download_success.html. [Accessed june 2014].

[7] The PHP Group, "Home Page," The PHP Group, [Online]. Available: http://www.php.net/. [Accessed june 2014].

[8] W. Davis, "Iterared Prisoners Dilemma Application," 22 March 2007. [Online]. Available: http://www.iterated-prisoners-dilemma.net/. [Accessed June 2014].

[9] R. Axelrod, The Evolution of Cooperation, NY: Basic Books, 1984.

[10] M. J. Osborne, An Introduction to Game Theory, Toronto: University of Toronto, 2000.

[11] M. J. Osborne and R. Rubinstein, A Course in Game Theory, USA: MIT Press, 1994.

[12] K. L. Brown and Y. Shoham, Essentials of Game Theory: A Concise, Multidisciplinary Introduction, Oregon: Ronald J. Brachman, Yahoo Research,Tom Dietterich, Oregon State University, 2008.

[13] Nobel Media AB, "MLA style: "The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel 1994"," nobelprize.org, [Online]. Available: http://www.nobelprize.org/nobel_prizes/economic-sciences/laureates/1994/. [Accessed june 2014].

[14] H. Tembine, "Game Theory Meets Computer Science and Engineering," King Abdullah University of Science and Technology, [Online]. Available: http://sri-

uq.kaust.edu.sa/Pages/Seminar-33.aspx. [Accessed june 2014].

[15] W. Spaniel, Game Theory 101: The Complete Textbook, NY: Amazon Kindle, 2011.

[16] G. Chalkiadakis, "Mutiagent Reinforcement Learning: Stochastic Games with Multiple Learning Players," University of Toronto, Canada, 2003.

[17] F. Carmichael, A Guide to Game Theory, UK: Pearson Education, 2006.

[18] E. Roberts, "Non-zero Sum Games," Stanford University, 2006. [Online]. Available: http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/nonzero.html. [Accessed june 2014].

[19] N. Nisan, T. Roughgarden, E. Tardos and V. V. Vazirani, Algorithmic Game Theory, UK: Cambridge Press, 2007.

[20] D. S.Nau, "Game Theory," [Online]. Available: http://www.cs.umd.edu/~nau/cmsc421/game-theory.pdf. [Accessed june 2014].

[21] Wikipedia, "Battle of the Sexes," [Online]. Available: http://en.wikipedia.org/wiki/Battle_of_the_sexes_(game_theory). [Accessed 2014 june].

[22] T. P. team, "Mixed Strategies," [Online]. Available: http://www.policonomics.com/mixed-strategy/. [Accessed june 2014].

[23] S. A. Schropp, Trade Policy Flexibility and Enforcement in the WTO:A Law and Economics Analysis,page 55 notation 5, UK: Cambridge University Press, 2009.

[24] Webopedia, "API - application program interface," Quinstreet Enterprise, [Online]. Available: http://www.webopedia.com/TERM/A/API.html. [Accessed june 2014].

[25] R. Kannan, "Android Applications," [Online]. Available: http://androidappsdevelopmentss.blogspot.gr/2014/04/what-does-api-level-mean.html. [Accessed june 2014].

[26] Google Android, "SDK elements," [Online]. Available: http://developer.android.com/guide/topics/manifest/uses-sdk-element.html. [Accessed june 2014].

[27] R. Meier, Professional Android Application Development, John Wiley & Sons, 2009.

[28] A. Misra and A. Dubey, Android Security: Attacks and Defenses, CRC Press, 2013.

[29] Mediawiki, "eLinux," [Online]. Available: http://elinux.org/Android_Architecture. [Accessed june 2014].

[30] IT&C Solutions, "Android Tutorial (4) – Procedural vs. Declarative Design of User Interfaces," [Online]. Available: http://www.itcsolutions.eu/2011/08/27/android-tutorial-4-procedural-vs-declarative-design-of-user-interfaces/. [Accessed june 2014].

[31] D. A. Heger, "Android Internals," [Online]. Available: http://www.dhtusa.com/media/AndroidInternals.pdf. [Accessed june 2014].

[32] J. Ahn, "Binder: Communication Mechanism of Android Processes," NHN Corporation, [Online]. Available: http://www.cubrid.org/blog/dev-platform/binder-communication-mechanism-of-android-processes/. [Accessed june 2014].

[33] R. Martin, "An Introduction to Android Shared Memory," [Online]. Available: http://notjustburritos.tumblr.com/post/21442138796/an-introduction-to-android-shared-memory. [Accessed june 2014].

[34] Netmite, "Android Power Managment," Netmite, [Online]. Available: http://www.netmite.com/android/mydroid/development/pdk/docs/power_management.html. [Accessed june 2014].

[35] eLinux, "Android Booting," eLinux, [Online]. Available: http://elinux.org/Android_Booting. [Accessed june 2014].

[36] Wikipedia, "Android Version History," [Online]. Available: http://en.wikipedia.org/wiki/Android_version_history. [Accessed june 2014].

[37] D. Bornstein, "Google I/O Session Videos and Slides- Dalvik VM Internals," 2008. [Online]. Available: https://sites.google.com/site/io/dalvik-vm-internals/. [Accessed june 10'4].

[38] J. Levi, "Dalvik vs. ART: Android virtual machines and the battle for better performance," Pocket Now, 13 november 2013. [Online]. Available: http://pocketnow.com/2013/11/13/dalvik-vs-art. [Accessed june 2014].

[39] "Android Benchmark," [Online]. Available: https://code.google.com/p/android-benchmarks/. [Accessed june 2014].

[40] N. Smith, Android 4.4 App Development Essentials-First Edition, eBookFrenzy, 2014.

[41] Edureka, "The Beginner's Guide to Android: Android Architecture," 2 january 2013. [Online]. Available: http://www.edureka.in/blog/beginners-guide-android-architecture/. [Accessed june 2014].

[42] Android Google, "Android Developer Tools," [Online]. Available: http://developer.android.com/tools/help/adt.html. [Accessed june 2014].

[43] D. Smith, "Context,what context?," Double Encore, [Online]. Available: http://www.doubleencore.com/2013/06/context/. [Accessed june 2014].

[44] Hachi Tecnologia, "Broadcast Receivers no Android," [Online]. Available: http://blog.hachitecnologia.com.br/mobile/trabalhando-com-broadcast-receivers-no-android. [Accessed june 2014].

[45] Tutorial Point, "Android Application Components," [Online]. Available: http://www.tutorialspoint.com/android/android_application_components.htm.

[46] F. Silveira, "Content Providers," [Online]. Available: http://www.felipesilveira.com.br/2010/05/content-providers/. [Accessed june 2014].

[47] Google Developers, "Google Cloud Messaging for Android," [Online]. Available: http://developer.android.com/google/gcm/index.html. [Accessed june 2014].

[48] Google developers, "Google Api Console," [Online]. Available: https://code.google.com/apis/console/. [Accessed june 2014].

[49] W. Danis, "Iterated Prisoners Dilemma," [Online]. Available: http://www.iterated-prisoners-dilemma.net/ . [Accessed april 2014].

[50] L. Tesfatsion, "Notes on Axelrod’s Iterated Prisoner’s Dilemma (IPD) Tournaments," [Online]. Available: http://www2.econ.iastate.edu/classes/econ308/tesfatsion/axeltmts.pdf. [Accessed june 2014].

[51] M. Wooldridge, "Computation and the Prisoner’s Dilemma," [Online]. Available: http://commonsenseatheism.com/wp-content/uploads/2013/04/Woolridge-Computation-and-the-Prisoners-Dilemma.pdf. [Accessed june 2014].

[52] S. Shulte, "The Sean Code," 19 May 2006. [Online]. Available: http://seancode.blogspot.gr/2006/05/my-own-jsonphp-web-services.html. [Accessed june 2014].

[53] M. Migurski, "Package Information: Services_JSON," [Online]. Available: http://pear.php.net/package/Services_JSON/download. [Accessed june 2014].

[54] R. Tamada, "Android Login and Registration with PHP, MySQL and SQLite," 31 january 2012. [Online]. Available: http://www.androidhive.info/2012/01/android-login-and-registration-with-php-mysql-and-sqlite/. [Accessed june 2014].

[55] Facebook Developers, "Facebook SDK for Android," [Online]. Available: https://developers.facebook.com/docs/android/. [Accessed june 2014].

[56] Google Developers, "Google Maps Android API v2," [Online]. Available: https://developers.google.com/maps/documentation/android/. [Accessed june 2014].

[57] L. Tatu, "Math Jump, from concept to a published title-Bachelor's Thesis," march 2013. [Online]. Available: http://www.theseus.fi/bitstream/handle/10024/55158/Laine_Tatu.pdf. [Accessed june 2014].

[58] L. Loeb and E. Y. Zhang, "Mobile Applications: Games that Transform Education-Dartmouth Computer Science Technical Report," may 2013. [Online]. Available: http://www.cs.dartmouth.edu/reports/TR2013-737.pdf. [Accessed june 2014].

[59] M. Townsley, M. Weeks, R. Ragade and A. Kumar, "A Large Scale, Distributed, Iterated Prisoner’s Dilemma Simulation," p. 6, 2006.

[60] M. Shor, "Java applets, online simulations, and game theory demonstrations.," 2006. [Online]. Available: http://www.gametheory.net/applets/prisoners.html. [Accessed june 2014].

[61] T. Turocy, "The Gambit Project," University of East Anglia, [Online]. Available: http://www.gambit-project.org/. [Accessed june 2014].

[62] Stanford University, DARPA's Transfer Learning Program, [Online]. Available: http://gamut.stanford.edu/. [Accessed june 2014].

[63] J. Kaplan, “Game Theory Solving Applet,” University of Colorado, [Online]. Available: http://spot.colorado.edu/~kaplan/econ4838/Projects/GameTheoryFinal.html. [Acesso em june 2014].

[64] S. A. Arman, "RESTful Mobile Application for Android : Mobile Version of Inspectera Online," Linköpings universitet-University Essay, [Online]. Available: http://www.essays.se/essay/9af51767d7/.

[65] Wikipedia, "Inter-process communication," [Online]. Available: http://en.wikipedia.org/wiki/Inter-process_communication. [Accessed june 2014].

[66] The PHP Group, "The PHP Manual," [Online]. Available: http://www.php.net/manual/en/intro.curl.php. [Accessed june 2014].

[67] Oracle, "Abstract Class," [Online]. Available: http://docs.oracle.com/javase/tutorial/java/IandI/abstract.html. [Accessed june 2014].

Appendix

Structure of Web Service file

```
<?php

/**
 *   Include the JSON encoder file.
 */

require_once("JSON.php");

/**
 *   Connects to the Server
 */

$con = mysql_connect("localhost", "myName", "myCode") or die('Server
connection failed');

/**
 *   Connects to the Database
 */
$dbCon = mysql_select_db('mydatabase') or die('Database connection
failed');

class SSWebService
{
    var $methods;
    var $json;

    /**
     * This function creates a JSON instance from JSON encoding file.
     *
     */
    function initialize()
    {
        $this->json = new Services_JSON(SERVICES_JSON_LOOSE_TYPE);
    }

    /**
     * This function adds a $name function to the server, so it can be
     * accessed.
     *
     */
    function register($name)
    {
        $this->methods[$name] = true;
    }

    /**
     * This function sets a registered $name function to be not
     * accessed.
     */
```

```php
    function deregister($name)
    {
        $this->methods[$name] = false;
    }

    /**
     * This function executes the given method $name, passing its
     *  single parameter($param).
     * Then JSON encodes the returned value.
     *
     */
    function call($name, $param)
    {
        if ($this->methods[$name] == true)
        {
            $evalstring = $name."(\$param);";
            eval("\$rval=".$evalstring.";");
            return $this->json->encode($rval);
        }
    }

    /**
     * This function decodes the JSON parameter into a native object,
     *  and calls the given method($method).It returns s JSON object.
     */
    function serve($method, $param)
    {
        $obj = $this->json->decode(stripslashes($param));

        if ($this->methods[$method] == true)
        {
            $res = $this->call($method, $obj);
        }
        else
        {
            $res = $this->json->encode("Not a registered function.");
        }

        echo $res;
    }
}
?>
```

Structure of an example of a PHP file that applies the Webservices

```php
<?php
/**
 * Include the web service´file
 */
require_once('webservice.php');

/**
 * Create a web service's instance
 */
$server = new WebService;

/**
 * Call method to create a JSON instance.
 *
 */
$server->initialize();

/**
 * Define the function(in this file) to call.
 */
$method = 'MyFunction';

/**
 * Set the parameter of the POST action to make.
 */
$param = $_POST['CommonVariable'];

/**
 * Registration of the method via Webservice.
 */
$server->register($method);

/**
 * Call the registered function (MyFunction)
 */
$server->serve($method, $param);

/**
 * Inside this function, any action can be made.
 *
 */
function MyFunction($par) {


  /**
 * Do anything….
 *
 */


          return $Something;
}
?>
```

Function *isRegistered()*

This function is an example of how the data in parameters is encoded in JSON format and sent to server. The data in the example is the email of the player and the action (confess or not).This function returns the confirmation of a successful POST in PHP file.

```
static String getUrl=serverAddress+"CommonVariable.php";

public boolean isRegistered(Context context, String oponentemail,String
action) {


             boolean isRegisteredSuccessfully = false;
             JSONObject RegistrationResultJSONObject = null;
             ArrayList<NameValuePair> postParameters = null;
             try {
                    postParameters = new ArrayList<NameValuePair>();
                    postParameters.add(new BasicNameValuePair("opponentemail",
                    opponentemail));
                    postParameters.add(new BasicNameValuePair("action",
                     actionData));

                    RegistrationResultJSONObject = new JSONObject(
                    com.confess.utility.JsonParser.doFetchDataFromWebService(
                                                                 AppUtility.getUrl,
                                                                 "CommonVariable",
                                                                 postParameters));


RegistrationResultJSONObject
=RegistrationResultJSONObject.getJSONObject("getGroupList");

if (RegistrationResultJSONObject.getString("Status").equals("Yes")) {
                          isRegisteredSuccessfully = true;

                    }

             } catch (Exception e) {
                    e.printStackTrace();
             }
             return isRegisteredSuccessfully;
      }
```