DESIGN AND CONSTRUCTION OF A LEARNING GAME ON ANDROID

MARC RUBIA CRIADO
DEFENDED ON APRIL 24TH, 2015
DIRECTED BY ALEJANDRO SCANDROLI
SUPERVISED BY PERE PAU VÁZQUEZ ALCOCER
COMPUTING ENGINEERING BSC, COMPUTER SCIENCE
FACULTAT D’INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BARCELONATECH
Chapter list

Chapter 1. Introduction, scope and goals ................................................................. 3
Chapter 2. Product design ..................................................................................... 23
Chapter 3. Application development .................................................................. 51
Chapter 4. The support libraries .......................................................................... 66
Chapter 5. Backend .............................................................................................. 90
Chapter 6. Results and conclusions ..................................................................... 94
Chapter 7. Bibliography and references ............................................................. 110

Abstract

English

Creating a game is a complex task where lots of roles are involved and requires a strong conviction and commitment to the project. If the game is oriented to the mobile voraciously competitive market, it also has to excel in several facets in order to achieve success.

In this project, the whole process of designing and creating an Android game is carried out: from the first steps of the idea generation and product design to the development of the game client and backend. It is created by and for Billy, the newest kid on the block between the mobile companies settled in Barcelona. The document also explores how different roles and teams take care of every one of these stages and how tasks are distributed.

From the technical side, the development of all the elements is explained, both from the implementation and performance points of view. The paper digs deep in how to achieve great results in the whole fragmented Android ecosystem and how can developers take advantage of the object-oriented mechanisms of Java to create high quality, reusable code for Android applications. To illustrate it, some parts of the application are abstracted and developed separately from the game client to be reused in other components and projects of the company.

Catalan

Crear un joc és una tasca complexa on molt rols es veuen involucrats i que requereix una forta convicció i compromís amb el projecte. Si a sobre el joc està orientat al voraç i competitiu mercat mòbil, ha d’excel·lir en moltes facetes per tal d’aconseguir ser exitós.
En aquest projecte es realitza tot el procés de disseny i creació d’un joc per Android, des dels primers passos de la generació de la idea i disseny del producte fins al desenvolupament del client i servidor del joc. És creat des de i per Billy, una de les noves empreses de l’entorn mòbil establertes a Barcelona. El document també explora com el diferents rols i equipos tenen cura de cadascuna de les parts i com es distribueixen les tasques.

Sobre la part tècnica, el desenvolupament de tots els elements és explicat tant des del punt de vista de la implementació com del rendiment. El projecte s’endinsa en com aconseguir grans resultats dins el fragmentadíssim ecosistema d’Android i com els desenvolupadors poden aprofitar-se dels mecanismes d’orientació a objectes de Java per tal de crear codi reutilitzable i de gran qualitat. Per il·lustrar-ho, algunes parts de l’aplicació s’abstraeixen i desenvolupen per separat del client del joc per tal de ser reutilitzats en altres components i projectes de l’empresa.

Spanish

Crear un juego es una tarea compleja donde muchos roles se ven involucrados y requiere una fuerte convicción y compromiso con el proyecto. Si además el juego está orientado al voraz y competitivo mercado móvil, este ha de ser excelente en muchas facetas para conseguir el éxito.

En este proyecto se realiza todo el proceso de diseño y creación de un juego para Android, desde los primeros pasos de la generación de la idea y diseño del producto hasta el desarrollo del cliente y servidor del juego. Es creado por y para Billy, una de las nuevas empresas del entorno móvil establecidas en Barcelona. El documento también recoge como los diferentes roles y equipos llevarán a cabo cada parte y como se distribuirán las tareas.

Acerca de la parte técnica, el desarrollo de todos los elementos se explica tanto desde el punto de vista de la implementación como del rendimiento. El proyecto ahonda en cómo conseguir grandes resultados dentro del fragmentadísimo ecosistema de Android y de cómo los desarrolladores pueden aprovecharse de los mecanismos de orientación a objetos de Java para crear código reutilizable y de gran calidad. Para ilustrarlo, algunas de las partes de la aplicación se abstraerán y desarrollarán por separado del cliente del juego con el objetivo de ser reutilizadas en otros componentes y proyectos de la empresa.
Chapter 1. Introduction, scope and goals

1.1 Project Relevance ................................................................. 4
1.2 Scope and goals .................................................................. 5
1.3 Project development Tools .................................................. 6
1.4 Project Methodology ............................................................ 7
1.5 Possible obstacles and solutions ......................................... 8
1.6 Context and references ...................................................... 9
1.7 Similar studies and state of the art ...................................... 11
1.8 Unresolved issues .............................................................. 13
1.9 Planning ............................................................................ 13
1.10 Budget and sustainability .................................................. 17
1.11 Social impact .................................................................... 21
1.12 Environmental impact ....................................................... 21
1.1 Project Relevance

Mobile devices are the most commercialized platforms in recent years, getting over traditional and long-established platforms like laptops or gaming devices. The rapid change experienced during the last years and the evolving mobile environment create an ecosystem where both users and developers have to constantly adapt to the different changes that occur: different versions of the operating system, changes in development tools, new standards, new software and hardware features...

The main feature of modern mobile devices are the thousands of applications available in their respective markets. Using Google Play stats as an example, there are more than 1.36 million apps available for Android users in September 2014. Developers have found their place in the mobile environment and today, application programmers are one of the most sought professionals in the software industry. However, the quality of the applications is not always guaranteed and all the app markets are flooded with low quality, which indicates that there is a long road of work and improvements ahead.

Going back to the figures, Appbrain states (AppBrain Stats, 2014) that 16% of the apps available in Google Play are low quality apps and 76% don’t meet the minimum quality requirements in order to be considered for Google as “Play Selection” apps. So it is all starting to be about quality instead of quantity. Mobile developers are required to focus on building better apps in all aspects: those related to the product design and those related to the technical capabilities of the platform. More and more profiles are getting involved in app development because the present and nearby future requires so: analysts, media buyers, designers or community managers are some of the increasing roles in the mobile apps environment.

In this project we will develop a new application from scratch which involves all these steps and roles in order to create a product that will become an important asset for the company.
1.2 Scope and goals

The initial scope for this project is to implement and release a first version of the game including its libraries and main features. The game will not be a final version and will be open to updates and more development stages, including new game mechanics and other features.

Taking in mind that the main goal of this project is to create an Android application with learning and innovation as their biggest and most important values, we have to consider that building this application will imply several and different processes which will be carried out in different areas of the company. These are the most important ones:

- Product design
- Graphic design
- Software architecture definition
- Backend construction
- Android app construction
- QA and performance tests

At the end of this academic project, users will be able to download and play with the game and the company will start receiving feedback, analytics and hopefully the first incomes. Besides this, we will develop some support libraries which will come bundled into the game and will be reusable in many other projects of the company.

The main requirement for the project is the correct operation of the application and its libraries, the integration within the mobile ecosystem as well as the satisfaction of various validation criteria that are defined in each step of its construction, including criteria related to the design and conceptualization of the product, performance criteria and criteria related to the use and adaptation by end users.

The technology part will be the most important as this project seeks among others to be able to create an application that can satisfy certain technological needs related to efficiency and app behavior among others.
1.3 Project development Tools

Three kind of tools will be used during the project: company tools, personal tools and development tools.

1.3.1 Company

Git
A code maintenance and version control tool used by the company. It will be used to store the code of the project and to keep track of every step in the code development.

Trello
Collaboration tool available in mobile and web used by Billy development team to assign and manage tasks. A new Trello board has been created for the project in order to keep track of the different tasks and to-dos.

Hipchat
Messaging mobile and desktop app used by every member of the company. It will be useful to chat with the different coworkers that will be involved in every step of the project.

Dropbox
Cloud storage used to share all the documentation and resources related to the project.

1.3.2 Personal

Sunrise
Calendar app that integrate all your Google accounts, Facebook and other calendars in a single place. Used to make reminders of meetings, deadlines and scheduling tasks or processes.

Evernote
Cloud rich-format notes app available in all platforms used to write down things at the time when inspiration strikes and sharing personal documents across all your devices.

1.3.3 Development

SimpleApi Framework
The PHP framework used by the company to create APIs, based on a decaf Symfony. We will use it to develop the server side of the application.
**PHPStorm**
A PHP IDE developed by JetBrains. We will use a student license, which provides a reduced-price year of the full licensed version.

**Android Studio**
The brand-new and only officially supported Android IDE developed by Google and based on JetBrains' IntelliJ IDEA.

### 1.4 Project Methodology
This project will involve several people from the company with different roles. The following steps are clear and defined:

- **Product phase**
  Define the specification of the application to be built, including among others its requirements, features and roadmap.
- **Development**
  Construct the software architecture of the application, choose the implementation tools and environments (IDEs, engines, frameworks...) and code the application and backend.
- **Postproduction**
  Execute QA processes and performance test. Analyze the impact and usefulness of the product within the product line of the company.

### 1.4.1 Validation Methodology and QA
Every advancement will be tracked and revised with the coworkers specialized in the corresponding area. At some concrete steps, we will ask for both external and internal opinions and feedback.

The product design and backend construction will be the most revised steps because they are the ones which are out of my work field in the company and will be the ones that require more implication from other coworkers.

The quality assurance process for each one of the phases related to the product design will be defined and carried out by the product team and the phases related to the technical implementation of the backend, the Android app and its performance will be in charge of the development team. For this purpose, the development team will use different benchmarks and stress tests.
When the application is finally build and ready to go live minor usability tests will be carried out. We will execute standard QA tests and processes and run through performance tests both for the application and its bundled libraries.

1.5 Possible obstacles and solutions

1.5.1 Deprecation

Changes occur within months and this directly affects projects and initiatives carried out in these platforms, which can see its features and procedures obsolete before going live. To avoid this problem, the development will be targeted to Android Lollipop 5.1, the latest available version.

Our first intention was to release the game for Android version 2.3 or superior. However, due to several factors such as compatibility libraries and market share, it doesn’t make sense to set 2.3 as the minimum version. We set up 4.0 as the minimum version, which covers more than the 95% of the Google Play market and allows us to use some APIs and features which are not available on 2.3.

1.5.2 Inhomogeneity of Android

Every manufacturer and brand produces their own hardware and software implementation and this affects any Android development. Lack of standards and peculiarities of every device are one of the most challenging things to overcome. We will use manual testing and external tools integrated in the application to check that the application works fine in every devices.

1.5.3 Company related issues

The project involves several people in the company and there are plenty of things to keep in mind when engaging people as personal contingencies that may arise due to lack of time or the dedication to other projects.
1.6 Context and references

1.6.1 The proposed problem

The problem we want to solve by building an application with a learning value has two aspects: product and technology.

The product side consists of building a high quality product adapted to today's standards. We want the application to use up to date systems and the latest trends, following the new design patterns like Material Design and the latest specifications included in the Android Developer guidelines.

The technological side faces the problem of giving the application nice capabilities in the artificial and emotional intelligence fields. We want the game to be able to know the user and adapt to any situation to obtain different objectives: to increase hours of playing, encourage friends and enhance the amount of in-app purchases and many other objectives that may arise.

1.6.2 Stakeholders

This section is dedicated to analyzing the context and project stakeholders. A stakeholder is a person, group or organization that has interest or concern in an organization or business opportunity.

Stakeholders can affect or be affected by the actions, objectives and policies. Some examples of key stakeholders are creditors, directors, employees, owners and the community from which the business draws its resources or focus its goals.

Taking in mind that this project is not a large scale oriented project, we have identified the stakeholders involved on it:

1.6.2.1 Company

The company is the main stakeholder of this project for several reasons. Firstly because even the company leaves the project in my hands and provides me with all the resources I need, this project is the first approach of the company to the massive in-app purchases business.

Business models based in mobile applications have grown a lot in the latest times, and the main reason for the increase in the amount of generated earnings is the boom of mobile gaming. Thousands of mobile games are downloaded every day in devices all around the world and their in-app purchases makes them highly profitable. ABI Research says
developers and other stakeholders generated $8.5 billion on mobile apps (Reisinger, 2014) last year, which is a pittance compared to where that figure is expected to go in the coming years.

This figure is expected to reach $46 billion in 2016 (Reisinger, 2014). Other researches are less optimistic and say that in-app purchase market will be worth $9 billion ($3 billion for tablets, $6 billion for smartphones) by 2016. However, 2015 will be a very exciting transition year so it is very important for the company to start developing this business line.

Moreover, the success of this project would create a precedent in mobile games for the company, implying the possibility of expanding the business area by growing in value and expanding areas of action. The company wants to stand out and gain prestige with a high-quality app, which makes use of the latest technologies and design guidelines so it can become a reference in the Play Store.

Finally, regarding all the libraries and systems developed for the application, they will be reusable and extensible so they can be used in many other projects carried out by the company which is very important since these libraries will be robust, fast and efficient and will take care of common actions present in most applications. Other external users will be able to use them as well because we will release them as open source code.

1.6.2.2 Project manager and involved employees
Other stakeholders are the employees who will be involved in this project.

Finally, I will be the person in charge of this project. I will take part in most decisions and will be present in every phase of the project. For me, this project is very important both personally and professionally.

1.6.2.3 End users
End users will be the beneficiaries of the product and will represent the final validation of the whole process, as this will be successful or not based on their acceptance. They are also the economic engine on which the project is based, for a large user base and a good acceptance and use by these will lead to higher profits. Even that product decisions may increase or decrease the performance, the volume of users will be crucial to scale the business.

Furthermore, they will receive the learning value included in the game itself, so they will not be only beneficiaries of the project product and validators but will receive an added value that involve knowledge and personal growth.
1.7 Similar studies and state of the art

1.7.1 Game and app Companies

The best implementations both technically and in terms of business strategy of Android applications are found in big game companies with many years of experience in the industry: King, Zynga, EA Sports, Rovio and more. They base their vision on a simple but profoundly successful strategy: games for all ages with a high level of addiction and an increasing difficulty.

Their games are fun, technically excellent and, even they have some objectionable aspects, they work perfectly in today's mobile environment. In fact, King is the leading company in this sector, generating with their "Saga" games more than $6 million a day (Takahashi, 2012) (Wauters, 2014) and placing four of their games among the 10 most profitable Play Store apps (AppAnnie, 2014).

Of course, we don’t want to compete with these companies. They all have more than 500 employees and years of experience in games and mobile. But they are an important reference for what’s trending, what works and how to be successful.

On the more technical and less business sense, there are some interesting approaches to what we want to create. One of the products with more "situational intelligence" that can be found in Play Store is Shifu (Patkar, 2013). Shifu is a smart task manager that reminds you when to do a task based on your location, time and your - calculated by the app - spare time.

Finally, Tasker is probably one of the most complex applications in Play Store (Wagoner, 2014). Tasker monitors everything that happens on your smartphone to perform actions based on the current environment. Thus, Tasker can put your phone on silent mode when you get to work or turn on airplane mode from 2 to 8 am. However, there is a big difference between Tasker and the system we want to implement: you have to tell Tasker what to do when certain conditions are met, because Tasker can’t think nor decide for itself.

1.7.2 Mvas and monetization platforms

While game companies are giants with great benefits, they still have a big problem: they don’t know how to optimize player behavior or make their applications intelligently respond to the behavior of these. That is why we have laid our eyes on mobile value-added services companies and ad monetization platforms. These companies, which often come from the desktop/web industry, also have years of experience and high-level optimization capabilities. They are able in a matter of tenths of a second to choose the best ad for a particular user based on information about his profile and information: country, language,
The whole sector of mobile advertising and companies running ad-servers rely on this. Halfway between a platform and an intelligent ad system, the ThinkGaming SDK (Think Gaming INC, 2014) provides an interesting tool for games and apps developers. It is a system that improves traffic acquisition of your game and optimizes the price and quantity of products that can be purchased through micro transactions within an application.

Thus, these two systems represent what we want to achieve. Firstly, to have a high quality application that is addictive and people like it. Secondly, implement an advanced technological environment to be able to know better the user and optimize actions and behaviors according to his profile, either to maximize purchases, hours of play, shares or to whatever we are interested in.

1.7.3 Other academic studies

The most academic part of this project lies in creating the algorithms and the artificial intelligence system implementation. Of course there are already thousands of games on Android and you can learn new things in many of them, but our innovation goes beyond that. This project is not just simply about making a top-notch game.

In the field of artificial intelligence, mobile devices and applications are still underdeveloped. Games of the companies listed above lack of adaptation to the situations and the user and this can be easily observed when for example they remind you to play while you’re at work. We want our application to be able to recognize the best moment for this type of call to actions and decide what to do and what not.

So, there are hundreds of projects related to application and games creation as well as social and/or payments integration, but it is really hard to find works related to intelligent apps or systems development especially on mobile platforms.

Even so, we have found a couple of university research papers that can be interesting. The first is titled “Microtransactions in an Android Game” and it is a complete overview on how to integrate inapp purchases in an Android game taking care of the gameplay and its side effects (Kokkonen, June 2014). However, it is not conclusive about the matter and it is more focused on the implementation process rather in the innovation.

The other interesting paper is titled “Quantifying User Satisfaction in Mobile Cloud Games” (Chun-Ying, Cheng-Hsin, De-Yu, & Kuan-Ta, 2014). It was published by a group of Taiwanese students and the paper discusses how several parameters such as the delay in network responses or the quality of graphics affects the user experience. These considerations are significant for our case because we must not ignore that carrying out certain actions,
complex and slow communications or huge calculations will lead to an unsatisfying user experience.

1.8 Unresolved issues

As we can see in the Play Store applications, many of the issues we want to solve in this project are not yet resolved. The main one is the lack of intelligence in most applications and how little these know about the end user. To fix this, we will create an entity that uses all the information that we can collect from the phone, user and environment to create a special behavior that decides what actions to perform depending on its state and input. This topic has been widely discussed and exemplified in the previous section, but in short we can say that this system includes aspects related to the game (AI, gameplay) and others more related to the mobile environment (reengagement, sharing, purchasing ...).

The other problem is present in all the Android platform and it’s about the small number of high quality applications, especially those with educational values. Android is in its worst moment in terms of the quality of their products and September 2014 was the first month in which applications considered low quality apps exceeded for the first time the volume of acceptable quality apps.

When this project comes to an end, we expect to have created an innovative application in terms of intelligence and technology, with a high quality product bundled with learning and design values. We hope that these features can convince the end users and the app becomes a first successful approach of the company to mobile games and microtransactions monetization models.

1.9 Planning

1.9.1 Estimated duration and special considerations

The estimated duration for this project is 4 months. The project starts in mid-December 2014 and the academic part will be finished before April 15th. After this time, project will be continued by the company depending on the acceptance and business expectations.
It is important to keep in mind that this is a project carried out in a company so there are a lot of things that can affect planning goals and deadlines. The project involves several people in the company and there are plenty of things to keep in mind when engaging people as personal contingencies that may arise due to lack of time or the dedication to other projects.

1.9.2 Project planning

In order to make a good project planning, it is important to clearly define the key points of the project. Breaking the project in parts based on its key points will make easier the whole planning development.

The key points of the project are listed below.

1.9.2.1 Project management
This phase divided into five stages:

- Project scope
- Project planning
- Project Budget and Sustainability
- Compliance of specialization conditions
- Documentation and oral presentation

The first four stages are part of GEP course and their deadlines are previously defined. The documentation and presentation stage will be carried out in parallel with the project.

1.9.2.2 Product phase
Divided into the following stages:
- Product definition
- Product image

In this phase we will define the specification of the product to be build, including among others conceptual parts such as its features, requirements, focusing and tangible parts such as its graphical design.

1.9.2.3 Development
Divided into two main stages:
- Application development
- Backend development
During this phase the software architecture as well as the engines and frameworks will be defined. Later on the application, its libraries and its backend will be coded and all its features will be implemented following good practices guidelines and keeping in mind future expansions, integrations and updates. We are including the development of the libraries and support systems inside the application development phase because they are directly bound and the libraries are giving an answer to the application requirements and features.

Development phase is bound to the product definition and it is impossible to start programming without having a specification document defining features and requirements. Because of this, development phase will start when the product phase is finished.

1.9.2.4 Postproduction
Mainly divided into:

- QA
- Performance evaluation

When the application and its backend are finished or significantly advanced, quality assurance processes and performance tests will be carried out in order to make sure that everything is working well. It will be very important to have some time for this phase because we will need time in case we have to make changes and fix things.

Even this phase is called postproduction, some of the processes will be executed as development is carried out. Moreover, by the nature of tests and QA processes it is important to be able to iterate over the development.

1.9.3 Resources
The collaborative and work-flow related resources that will be present are the ones that were specified in the scope section. Those mainly included messaging apps, collaborative tools and will not be listed here. However, there are other direct and indirect resources involved in this project.

1.9.3.1 Hardware

- Macbook Pro Retina, 13-inch, Late 2013
- Amazon Cloud Server

1.9.3.2 Software

- Android SDK (latest version, currently API level 22)
- PHPStorm IDE
- Android Studio
- SimpleApi Framework
1.9.4 Time division estimation

It is hard to make an estimation about the hours spent in each stage of the project because we don’t know exactly where the challenges and difficulties will arise. It is important to point out that no time has been kept apart for the academic parts like writing the project documentation or preparing the presentation because they are included in the time dedicated to every stage. This means that if we are planning to spend 80 hours in product definition, the hours dedicated to writing the documentation for this part are already included.

It is also important to differentiate between personal time and approximate company time. There are tasks where the personal time involved will be lesser than other coworkers, especially in those that are out of my working field.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Personal hours</th>
<th>Company hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project management</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Product phase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product definition</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Product image</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>Development</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application development</td>
<td>350</td>
<td>30</td>
</tr>
<tr>
<td>Backend development</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Postproduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QA</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Postproduction</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 1 Calculation and estimation of hours spent per task

1.9.5 Action plan and alternatives

The most blocking part of the project is the product phase. If the product phase does not suffer problems and delays the perspectives are good because the programming part will require less people involved and we can take time off from work hours and weekends.

It is important to keep in mind that technical features have not been specified yet so it is not known whether the implementation will take more time than estimated. If so, we would have to cut time in postproduction processes. The easiest part to cut is the related to the QA and performance tests which are not very relevant for this project and can be carried out later.
1.9.6 Planning overview

The total amount of personal time estimated is 630 hours. That is 158 hours/month, 40 hours/week which is the equivalent to a full time job.

However, taking in mind that I have other responsibilities and projects in the company, the average time per week I will spend in this project is 20 hours/week from work time. So other 20 hours per week will be taken off from spare time and weekends.

The first phases to be carried out will be the project management and the product design. After these two blocking stages, the development phase will take place and the postproduction will begin as the development is in an advanced state.

Finally, company’s own planning and high-priority tasks may affect the project planning.

1.10 Budget and sustainability

1.10.1 Considerations

Before going into depth with the budget, it is important to keep in mind that this project covers the product design and its development. In the life cycle of an application one of the most expensive stages is the traffic acquisition and marketing investment, but we will not cover it because it takes place after the application goes live.

It is also remarkable that there are a lot of considerations of whether an expense is related to the project. For example, both software and hardware is used during the project, but the company already owns most of it so it is not included in this budget.

1.10.2 Budget

1.10.2.1 Human resources
In the following table we can see the human resources expenses. Hourly rates listed below are the current hourly rates of the company. Hourly rates can be a bit out of the market (above or below) in some cases due to internal policy of the company and other
considerations that are not relevant for this project.

In my personal case, I’m earning 2000€/month and working 40 hours/week so that makes an hourly rate of 12.5€, taxes included.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Personal hours</th>
<th>Company hours</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>€/h</td>
<td>h</td>
<td>Role</td>
</tr>
<tr>
<td>Project management</td>
<td>12,50</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Product definition</td>
<td>12,50</td>
<td>15</td>
<td>Product developer</td>
</tr>
<tr>
<td>Product image</td>
<td>12,50</td>
<td>15</td>
<td>Product developer</td>
</tr>
<tr>
<td>Application development</td>
<td>12,50</td>
<td>350</td>
<td>Junior Android developer</td>
</tr>
<tr>
<td>Backend development</td>
<td>12,50</td>
<td>100</td>
<td>Senior software engineer</td>
</tr>
<tr>
<td>QA</td>
<td>12,50</td>
<td>10</td>
<td>Product developer</td>
</tr>
<tr>
<td>Postproduction</td>
<td>12,50</td>
<td>10</td>
<td>Frontend Engineer</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 Human resources expenses

### 1.10.2.2 Hardware

There is no traditional hardware involved in this project because the company already provides screens and laptops for each one of the employees so it is not necessary to buy any piece of hardware. However, in order to construct the backend and the website for the application, we will need a dedicated server. The dedicated server provider has not been decided yet so we have selected a standard high-end Amazon cloud services setup in order to have it present in the budget.

<table>
<thead>
<tr>
<th>Product</th>
<th>Units</th>
<th>Lifespan</th>
<th>Amortization</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon AWS c3.2xlarge current generation dedicated instance</td>
<td>1</td>
<td>1 year</td>
<td>-</td>
<td>570,64 €</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td></td>
<td>570,64 €</td>
</tr>
</tbody>
</table>

Table 3 Hardware expenses
There is no amortization because we are not buying it physically and we can finish off the service anytime we want.

1.10.2.3 Software
Most software used in this project is Open-Source or free distributed software. Eclipse IDE and the Android SDK are open-source, all the communication and collaborative tools are free and the PHP framework will be an open source framework as well. So the only software cost will be PhpStorm, the chosen PHP IDE for the backend and web development. A special license for students is available so we will pay a reduced amount for it.

<table>
<thead>
<tr>
<th>Product</th>
<th>Units</th>
<th>Lifespan</th>
<th>Amortization</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhpStorm IDE</td>
<td>1</td>
<td>Lifetime</td>
<td>26% (6,76€)</td>
<td>26,00 €</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>26,00 €</strong></td>
</tr>
</tbody>
</table>

*Table 4 Software expenses*

In this case, the yearly amortization is ridiculous but it is stated that both software and hardware have an amortization of 26% during up to 10 years.

1.10.2.4 Other expenses
There are others uncategorized expenses. We have to buy at least one top-level domain for more than one year in order to help the search engine optimization. We have to buy a new developer account from Google in order to upload the application to the Play Store.

<table>
<thead>
<tr>
<th>Product</th>
<th>Units</th>
<th>Lifespan</th>
<th>Amortization</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>.com domain 3 years</td>
<td>1</td>
<td>3 years</td>
<td>-</td>
<td>27,90 €</td>
</tr>
<tr>
<td>Google developer account</td>
<td>1</td>
<td>Lifetime</td>
<td>-</td>
<td>25,00 €</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>52,90 €</strong></td>
</tr>
</tbody>
</table>

*Table 5 Other expenses*

Although there is a lot of debate about whether to amortize or not internet domains and accounts, we have decided to ignore it for this project because of the low amount that it represents.
### 1.10.2.5 Final budget

<table>
<thead>
<tr>
<th>Expense</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human resources</td>
<td>10,550,00€</td>
</tr>
<tr>
<td>Hardware</td>
<td>570,64€</td>
</tr>
<tr>
<td>Software</td>
<td>26,00€</td>
</tr>
<tr>
<td>Other</td>
<td>52,90€</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>11,199,54 €</strong></td>
</tr>
</tbody>
</table>

*Table 6 Final budget overview*

### 1.10.3 Budget follow-up

In order to check whether we are not out of budget we will need to have a record of the hours spent in each task, because there are only two main expenses: human resources and electricity. So if we want to check if we are not going out of budget at a certain point, we will need to check the hours spent in each task.

For that reason, the budget follow-up will take place at the end of each of the stages of the project. If we have spent more hours than planned we will have to calculate the extra costs by multiplying the hours for the hourly rates and adding the electricity cost. For example, if we spend 20 (10 personal and 10 from the Tech lead) extra hours in the backend development stage, we will have an extra cost of $10 \times 12.5 + 10 \times 40 + 20x$

In case we exceed the budget for a certain stage, we will notify the company in order to keep the financial and human resources departments informed. Then we will wait for feedback and further actions if required.

### 1.10.4 Economic viability

The company has not set any economic goals for this project. The budget is pretty small for a game-related project and at this moment the company sees it more as an experiment rather than a business opportunity. But of course there will be a business model behind the project, which will be defined in the product phase.

At this point, due to the current business models that are working better in Android, we have the strong feeling that the monetization will be linked to the amount of installs and
usage of the application, so unfortunately we will be more dependent of the volume of users rather than the quality and loyalty of a minor amount of them.

1.11 Social impact

One of the main values of the application we want to build is learning. Of course the application needs to be fun as well if we want people to play. As an educational application with a background that is undefined, we can say that will have a positive social impact as players learn new things or strengthen others they already known.

Similarly, being a first approach of the company in the gaming industry and in-app purchases, a good outcome of the project may lead to expand the company and make more products in the games line, create more jobs, etc.

1.12 Environmental impact

This is a project with a low load of electronics involved, since the only required elements will be the laptops of the people involved on it.

Laptops are MacBook Pro Retina 13-inch from late 2013. Apple reports in his website that this model meets the Energy Star 6.0 requirements and achieves a Gold rating from EPEAT\(^1\). Among many features designed to reduce environmental impact like arsenic-free display glass, mercury-free LED backlit display, brominated flame retardant-free, PVC-free and recyclable aluminum enclosure, the MacBook Pro is one of the most energy-efficient laptops in the market. Its power adapter efficiency is 89,7% and MacBook Pro uses around 12 watts when on but idle. It uses 16-18 watts when crunching something (opening a program, saving a file, etc.). Its power consumption peaks up to around 30 watts (when starting up).

Unfortunately, like most big companies Apple is setting its manufacturing plants in countries with child exploitation and without worker protection laws. A lot of labor abuses

and suicide cases are being investigated at the moment.

Given the amount of hours and assuming that all company employees use the same computer with an average power consumption of 20W, the consumption table is as follows:

<table>
<thead>
<tr>
<th>Component</th>
<th>Power consumption</th>
<th>Hours</th>
<th>Kw/h</th>
</tr>
</thead>
<tbody>
<tr>
<td>MacBook Pro</td>
<td>20 W (0.02 KW)</td>
<td>870h (total estimation)</td>
<td>17.4 KW/h</td>
</tr>
<tr>
<td>Asus VW225N support screen</td>
<td>50 W (0.05 KW)</td>
<td>870h (total estimation)</td>
<td>43.5 KW/h</td>
</tr>
</tbody>
</table>

Table 7 Electric consumption overview

Besides of the electric consumption, there will be other resources which use is closely related to the environment. We will use paper and notebooks for the follow-up meetings and note taking, as well as to print the final document for delivery.

All paper used in the office is separated for recycling.

### 1.12.1 Sustainability matrix

<table>
<thead>
<tr>
<th>Sustainable Planning</th>
<th>Economics</th>
<th>Social</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Economic viability</td>
<td>Improvement in life quality</td>
<td>Resources analysis</td>
</tr>
<tr>
<td>Score</td>
<td>9</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Results</td>
<td>Final expenses versus estimation</td>
<td>Social impact</td>
<td>Resources consumption</td>
</tr>
<tr>
<td>Score</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Risks</td>
<td>Adaptability to new scenarios</td>
<td>Social damages</td>
<td>Environmental damages</td>
</tr>
<tr>
<td>Score</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Global score 26

Table 8 Sustainability matrix of the project
Chapter 2. Product design

2.1 Idea generation

Our goal was to create an Android game with multiplayer features, high engagement and with learning as its biggest value.
When I started thinking about how to achieve this goal, I first thought in getting back to an original trivia game based only in images and with learning about all the aspects of the general and popular culture I had designed a year ago and fell on deaf ears.
I recovered this concept and made a first presentation to the product designers and business developers of our team explaining how the game worked and which where its main mechanics and strengths. Everyone liked the idea but when assessing it, in the first steps of the planning we quickly noticed we had a big obstacle in front of us which was the content creation. In order to make a good trivia game, we had to create many fresh and original content and we realized that this would cost us a considerable amount of time and resources.

So we changed our focus: we needed to design a game where content was already created or automatically generated because otherwise we would need to spend a lot of time creating content and this could have many implications such as outsourcing this phase or releasing a game with a low amount of content which could be finished too fast.
When thinking about this new terms, the idea of making a words-based game came up. We thought we could easily design a game concept that complies every one of our conditions.

First, the content of a word-based game could be auto generated by using a dictionary or a big sample of words. This solved the content creation issue and also guaranteed that users could play with auto generated content for as long as they want. Furthermore, it would make easier to translate it to other languages, which was another matter of concern.

Secondly, it was possible to set up funny and addictive multiplayer mechanics as it has been demonstrated by successful approaches made by other developers in many words-based
games such as Words with Friends, Scramble or Apalabrados. All these games use well-known mechanics from popular board games and have been huge hits in the Play Store due to their combination of simplicity and addictiveness.

Finally, we could use the game mechanics to introduce a learning value based on etymologic, meaning and grammar aspects of the words involved in the game. There were a lot of possibilities depending on the depth we wanted.

With all those things in mind, we had a brainstorming session to define the main game mechanic. After coming up with some ideas which were already present in other successful games and we wanted to avoid in order to have a differential factor, we decided to make an approach with the words ending game. The words ending game is the classic game we all have played when we were kids with our parents or siblings. It is based on a very simple mechanic: answering with a word that starts with the last two letters of the previous answer.

After this, we needed a name to refer to the project. We decided to name it Slang. In English language, slang is a very informal usage in vocabulary and idiom that is characteristically more metaphorical, playful, elliptical, vivid, and ephemeral than ordinary language. Slang can also be used to refer to the jargon of a particular class, profession, etc. There are two reason why we choose Slang as the name of the project. Firstly is because it is a metaword: a word used to describe words and language. Secondly because it is a short, cool, easy to pronounce and to remember word.

So, with this simple mechanic defined as the core of the game, we started building an entire product around it.
We had to think about several aspects of this product: its technical requirements, game rules and mechanics, the application design and screens, gamification aspects, game modes and many more. The whole design process of all these features is covered in the following sections.

2.2 The MDA framework

When we started designing the game and defining which were its basic mechanics and rules, we found out that there was a lack of formal methodologies in the decisions we were taking. At first we decided that the word’s ending game was our source of inspiration and that the game was going to use that mechanic as its core, but there was no process behind these design decisions. So we started looking for a new way to design the game which could give us a more formal, business oriented approach and methodology.
After reading about several game design frameworks and methodologies, we choose to stick to the MDA framework. The MDA framework (standing for Mechanics, Dynamics, and Aesthetics) is a formal approach to understanding games which attempts to bridge the gap between game design and development. It was developed and taught as part of the Game Design and Tuning Workshop at the Game Developers Conference at 2004 (Hunicke, LeBlanc, & Zubek, 2004). This is one of the few academic papers that achieved wide exposure within the game industry, probably because the authors are experienced game designers.

The MDA frameworks states that a game is defined in terms of its Mechanics, Dynamics, and Aesthetics:

- **Mechanics** are a synonym for the “rules” of the game. These are the constraints under which the game operates. How is the game set up? What actions can players take, and what effects do those actions have on the game state? When does the game end, and how is a resolution determined? These are defined by the mechanics.
- **Dynamics** describe the play of the game when the rules are set in motion. What strategies emerge from the rules? How do players interact with one another?
- **Aesthetics** (in the MDA sense) do not refer to the visual elements of the game, but rather the user experience (UX), the player experience of the game: the effect that the dynamics have on the players themselves. Is the game “fun”? Is play frustrating, or boring, or interesting? Is the play emotionally or intellectually engaging?

Before the MDA Framework was written, the terms “mechanics” and “dynamics” were already in common use among game designers. The term “aesthetics” in this sense had not, but has gained more use in recent years in modern game development.

![Diagram showing the relationship between Mechanics, Dynamics, Aesthetics, Rules, System, and “Fun”](image)

*Figure 1 Formalization of games in distinct components and their design counterparts*

With the definitions out of the way, why is this important? This is one of the key points of the MDA paper. The game designer only creates the Mechanics directly. The Dynamics emerge from the Mechanics, and the Aesthetics arise out of the Dynamics. As game
designers, we may want to design the play experience but we are stuck building the rules of the game and hoping that the desired experience emerges from our rules. This is why game design is sometimes referred to as a second-order design problem: because we do not define the solution, we define something that creates something else that creates the solution. This is why game design is hard, because unlike other entertainment products, the behavior of the users is unpredictable so the string of events that occur during gameplay and the outcome of those events are unknown at the time the product is finished.

Figure 2 The designer and player each have a different perspective

Design is not just a matter of coming up with a “great idea” for a game; it is about coming up with a set of rules that will implement that idea, when two-thirds of the final product (the Dynamics and Aesthetics) are not under our direct control. By moving between MDA three levels of abstraction, we can conceptualize the dynamic behavior of a game system. Understanding games as dynamic systems helps us develop methodologies for design allowing us to control for undesired and unexpected outcomes, and tune for desired behavior. In addition, by understanding how formal decisions about gameplay impact the end user experience, we are able to better decompose that experience, and use it to improve our game.

2.2.1 Mechanics

In the MDA framework, mechanics are the set of rules and constraints under which the game operates. Our first design goal was to define the set of mechanics in order to produce dynamics which at the end will result on aesthetics.

To create new mechanics, we had to take into account several factors. In order to succeed, mechanics had to be easy to understand for everyone, both experienced and new players. They should feel and look natural and fit into the game, and even though there is a lot of room for innovation in mobile environment, we thought that we should stick to the casual, well-known mechanics in a first version. Finally, when creating mechanics, as a designers we should keep in mind that we can introduce mechanics to control the state of the game as well as define how it can and cannot be played.
Our main mechanic was already defined: the player had to reply with a word starting with the end of his opponent’s answer. We extended it and made up some mechanics starting from the core which are listed and explained below.

**The word’s ending rule**
A correct reply to the opponent’s word has to start with the two last letters of the opponent’s word. For example, if the opponent has previously replied with the word “HELLO”, the player must reply with a word starting with “LO”. The word “LOLLIPOP” would be a correct answer, and then the opponent should reply with a word starting with “OP”.

**The first word rule**
When you start a new game, a random user who has the application installed will be chosen as your opponent. Then the application will pick up a random word and you will need to answer this word in your first round. For example, the user starts a new game and the application picks the random word “HARBINGER”. Then the user has to reply with a word starting with “ER”. If the user replies with a correct answer (e.g. “ERASE”) then his opponent will be notified and the game between them will start. So, in order to start a new game, the user needs to answer a random word generated by the application.

**The time rule**
The user has to answer a word in less than 30 seconds after seeing his opponent’s last word. If the user can’t reply in less than 30 seconds or leaves the game when the time is already running, he will lose.

**The no words rule**
When one of the player forces his opponent to reply with a word starting with a combination which cannot produce any English word (e.g. a word starting with “XZ”), he or she automatically wins the game.

**The points rule**
Players earn points by answering words. Every word will give a certain amount of points based on an algorithm which will take account of the length of the word, its amount of vowels and less common letters and its number of definitions.

**The items rule**
In the first version of the application, two items will be available to the user in order to add more gamification factors. These items can be used when the player is answering a new word.
The first item is the so called “Letter swap” and it swaps the two first letters. For example, if the player has to reply with a word starting with “LA”, he or she can use a Letter Swap item and the word will be starting with “AL”.

27
The second item is the “Extra Time”. By using it 30 extra seconds will be added to the timer. Users can buy items in the in-app store by using their earned points. A player can use as many items as he wants in a game or round.

These are the mechanics present in the game. We decided to keep it simple in the first version and expand them in next updates after receiving the first results, feedback and impressions.

2.2.2 Dynamics

Dynamics describes the runtime behavior of the mechanics acting on player inputs and interactions as well as each other’s outputs over time. Dynamics work to create aesthetic experiences which are the desirable emotional responses evoked in the user when he or she is playing. Our previously defined mechanics lead to game dynamics. This process can be explained as which flow and interactions are generated by the game rules. The dynamics generated by our mechanics are explained in this section. However, these are not fixed and are more a general approach rather than an undisputed truth.

**Dynamics generated by the word’s ending rule**
The user has to think of a word starting with the suitable letters and write it. When thinking which word could fit in, user will also keep in mind the termination of the word in the interest of making the opponent’s answer harder.

**Dynamics generated by the first word rule**
When a user starts a new game, he or she can play immediately the first round versus the computer to start the game. This generates a dynamic where the user can play, practice and earn points by starting new games and playing the first round versus the system. Furthermore, the opponent is chosen randomly so the user can start as many games as he or she wants with no need of inviting friends.

**Dynamics generated by the time rule**
When answering a new word the user is under pressure. This forces the user to think fast and use extra time items if required. Sometimes he or she won’t find any word that fits in and will lose the game, and in other occasions, if nothing better comes to mind, the answer might have a termination easy to reply for the opponent.

**Dynamics generated by the no words rule**
When assessing the quality of an answer, the user will keep in mind that the last two letters create a hard to reply combination. If this combination is strange or hard enough and no word starting with it exists in the dictionary, the user will automatically win. For example, if the opponent word was “ELBOW”, the user could reply with the word “OWL” and win.
because there is no word in English language that starts with “WL” or “LW” (due to the swap letter item).

**Dynamics generated by the points rule**
Similar to previous dynamics, the points rule forces the user to think twice and think better before answering to earn more points by using more complex words. It also adds a rewarding factor for playing and some depth to the game as well as a discovery dynamic because the player don’t know how many points will earn until he or she replies with a new word. Finally, it creates a challenge between players because every user will try to answer with words that give more points than his opponent's answers in order to spend them in buying more items.

**Dynamics generated by the items rule**
The possibility of using items while playing modifies the user flow in several ways. For example, by using the items he can extend the response time so he can modify the time rule and add extra time. The player can also change the word’s ending rule by using a “Swap Letter” item and swap the letters of the answer. So, using items is one of the most important dynamics because it modifies the game flow and introduce other implicit dynamics, such as decision dynamics when the player assess whether to use an item or not or game changing dynamics when the player is about to lose and uses an item to answer and avoid defeat. Finally, it adds other decision dynamics because the user can choose in which items spend their earned points.

### 2.2.3 Aesthetics

Aesthetics describes the desirable emotional responses evoked in the player when he or she interacts with the game. The dynamics present in our game define this responses and in every game we can find many aesthetics in varying degrees. Aesthetics are not absolute and depend on many factors, most of them based on the player experience and personality. A game can result challenging to someone and easy to someone other, but what we are actually exploring is what aesthetics will (or might) be evoked by our dynamics.

In describing the aesthetics of a game, we want to move away from words like fun and rewarding towards a more directed vocabulary. This includes but is not limited to the taxonomy listed here:
- Sensation
  Game as sense-pleasure
- Fantasy
  Game as make-believe
- Narrative
  Game as drama
- Challenge
  Game as obstacle course
- Fellowship
  Game as social framework
- Discovery
  Game as uncharted territory
- Expression
  Game as self-discovery
- Submission
  Game as pastime

Before thinking of what aesthetics are present in our game, we can easily see which are not. For example, there is clearly no fantasy, expression or narrative aesthetics. Instead, if we look at the dynamics we can see that some of them are focused on the player answering under pressure and with the best word he can think of. So, undoubtedly there is a challenge aesthetic. Answering with correct words, earning points and winning games are all rewarding dynamics that will lead to a pleasure sensation, so the game also has a sensation aesthetic.

Although the game doesn’t contain traditional exploring and discovery mechanics like travelling around a brand-new world or discovering new weapons or equipment, one of the most important features of the game is that players can see the definitions of each word involved in every game so here is our discovery aesthetic: the learning value found in discovering new words and their meanings.

All this gives us a general idea of the overall process of game design and, most importantly, how users will experience it. Although this is not an exact science, now we better understand the mechanics that must be implemented in the game as well as the dynamics generated by them which will define the game flow. By participating in these dynamics, players will experience different emotions that in the end are the goal of any game. In the next step, we are going to define the game flow resulting from all these decisions.
2.3 Game flow

After designing the mechanics and the dynamics they originated, we can determine the game flow and features.

The flow is the order in which dynamics are executed. Controlling the flow is very important because it helps us to avoid undesired behaviors, outputs, and player experience. In this first version of the product, the game flow is quite simple because most of the dynamics take place in the same moment: when the player is going to answer with a new word.

If we recap all the dynamics and mechanics seen in the previous sections, we can easily see that the most natural game flow is as follows:

- **User starts a new game**
- The application chooses a random word to start the game
- User has to answer the word following the established rules
- If the user answered correctly:
  - App chooses a random opponent

  While there is no winner:
  - Opponent answers the user's word
  - If answered correctly:
    - User answers the opponent's word
    - If not answered correctly:
      - Opponent wins
  - Else:
    - User wins

  EndWhile
  Else:
  - Game is not started and user is back to start

We can formalize this flow by using flowchart elements to obtain the following chart:
Chart 1 The game flowchart

User starts a new game

App chooses a random word

User has to answer App word

Back to main menu

User answered correctly?

No

User wins

Yes

App chooses a random opponent and sends him the user answer

User can use items

Yes

User can use items

No

User loses

Opponent has to answer word

Answered correctly?

No

Opponent can use items

Yes

Legend:

Decision

Process

Terminator

Manual Input
This flowchart shows the in-game paths that the user can follow when playing and will be useful in order to create and code the states of the game. Keep in mind that in the first version of the game, the player won’t be able to invite friends to play against. This will be one of the first new features to be implemented. So, at this points we have the rules and mechanics, the dynamics and aesthetics that result from them as well as the flowchart that defines the in-game flow.

What’s next is to start building the application: thinking about the application screens, drawing the wireframes and mockups and design them before moving to the first steps of the development process.
2.4 Application screens

When considering how we wanted the users to interact with the application before, while and after playing we decided that we should stick to the traditional hub-style main screen where the user could start any of the actions such as see the active games, start a new game or see important stats. After thinking about this main screen, we started designing all the other screens required in order to execute the previously explained game flow in the best possible way.

For drawing these mockups and create the final design we used a software called Sketch that allowed us to add basic forms, text and pictures to compose a screen and take advantage of extensive layer styles. Sketch’s fully vector-based workflow makes it easy to create beautiful, high-quality mockups and final designs from start to finish.

2.4.1 Mockups

Main Screen

This is the main screen of the game. In its top the player can see the profile icon and the three most important stats: the amount of points he or she has earned, the number of victories and the number of defeats.

Below these stats there is a feed containing the list of active games. The first positions of the feed are reserved for those games where it is the player’s turn to answer. Below there are the rest of games where the player is waiting for the opponent’s answer. In the bottom, there is a big button to start a new game.
When clicking on the profile icon in the welcome screen, a right navigation drawer will be displayed above the screen. It will show the avatar of the player on top. By clicking on the picture, the user will be able to change it.

Below the user’s picture there are two icons. The left one will open the in-app store where the user can buy the power-up items with his points. The right one is used to open the settings panel of the application.

Finally, the rest of the drawer is used to display the history of finished matches played by the user. It shows the picture of both players and the number of rounds the match lasted. By clicking on any match, it will be displayed following the match template feed which is explained below.
Game Feed Screen

This is the screen containing all the information related to a certain game. It is styled following a feed timeline layout similar to WhatsApp chats.

In this screen the user can see all the events that happened during a game, including his own and his opponent answers, the obtained points, the items used and the current state of the game: victory, defeat, waiting or answer.
By clicking on the answer button, the user will see his opponent’s word and will have to reply with a word starting with the last two letters of the answer. In the top the remaining time is displayed as well as a cross used to resign from the game. In the center of the screen, the player can see the first two letters of the word he has to reply. Below there are the items call-to-action buttons. By clicking them, the corresponding item will be used (letter swap or add time).

In the bottom there is a custom keyboard used to type in the answer.
2.4.2 Design

Main Screen

When thinking of how the main screen should look, we decided to use a light color palette based on the Blue Grey palette of the Material Design color spec. The design team started by boxing the elements and separating the stats header, the open games feed and the new game button. We used flat and colorful icons to show the points, victories and defeats and used the default Android user icon for opening the profile drawer on top of the screen.

The combination of grey and green gives visual information about the state of the games, and the blue used on the new game button highlights it and shows that it is an important action.

On the second iteration of the design, we improved the new game button by giving it some volume and removing the blue stroke. We also added some margins to the content box in order to adapt it to the Material Design guidelines.

Finally, in the last iteration we increased the contrast in some sections such as the top toolbar and the body containing the open games feed and we inverted the colors of the title and profile icon.
When clicking on the profile icon, the right navigation drawer is opened using the standard animation and several information displayed. The user’s profile picture, the amount of power-up items of each kind (letter swap and extra time) he or she has represented by an item icon and a blue-styled button used across all the application to open the power-ups store panel.

Below is the game’s feed that displays the list of finished games following the same style used in the main screen: the number of rounds and the total points obtained in the game. The information is displayed in different font sizes and colors to give more importance to the rounds rather than the points, but it could be changed in next updates and add other relevant information.

When the user clicks on the power-ups store button, a dialog is shown overlaying the main screen. On top the user can see the amount of points he or she has and the close cross. The body of the dialog contains the item icons, the amount of items selected and two side buttons used by increment or decrease the amount of each item which red and green colors extracted from the material design color spec associated. On bottom there is the common blue bezel button containing the price for the whole current item selection. By clicking on it the user will buy the selected items.
This screen shows a game history feed. It is shown every time the user clicks on a game and displays all the words answered, all the items used and all the points earned by every player. As in the main screen, the different information (points, new words, items) is displayed in different styles to give more contrast between them.

Below it is displayed the current state of the game which can be waiting, answer, victory or defeat. The button changes its color depending on the state. We decided to use a chat-styled feed where the newest actions are displayed on bottom because people is more used to it thanks to WhatsApp and these kind of apps.

On top, the time to answer and the surrender cross is displayed when the game is in Answer state, because when the user clicks on the Answer button the app will swipe down the feed and replace it with the answer screen in a fancy animation.
The play screen is designed based on the decision of creating our own keyboard. Since we didn’t want the user to input numbers or symbols, we decided to create a simple keyboard to limit the input to the character we want. By doing this, we were able to add a “Go” button on bottom that changes to “Correct” or “Incorrect” depending if the input word is correct or not.

The keyboard is simple and elegant and it is inspired by the latest version of Google’s Lollipop keyboard.

On top we have the remaining time and the surrender cross which were already present in the previous game feed screen.

In the middle of the screen there is the word input box that doesn’t behave like the regular input texts of Android because it ignores the user clicks and below it the two item buttons. Inside each button there is a number that displays the amount of items remaining. We used blue bezel buttons similar to the others used across all the application.
2.5 Application support: Libraries, APIs and SDKs

When writing Android applications, it is important to keep in mind that we should make a distinction between the different features and requirements of the application and distinguish between those who are related to the application itself such as the user interface or the data controllers and those who take care of common tasks present in many projects.

The first ones are written by the programmer and are very specific, impossible or very hard to reuse in other projects. But the latter are usually implemented by using third-party libraries, SDKs, APIs or open source codes that have been carefully developed and tested focused in carrying out certain actions.

But first let’s define these similar terms in order to understand better its distinctions and peculiarities.
A library is a chunk of code that you can call from your own code, to help you do things more quickly or easily. For example, a Bitmap Processing library will provide facilities for loading and manipulating bitmap images, saving you having to write all that code for yourself.
An SDK (software development kit) is a library (often with extra tool applications, data files and sample code) that aids in developing code that uses a particular system. For example, the Android SDK is an extension code for using features of the Android operating system.
An API (application programming interface) is a term meaning the functions/methods in a library that you can call to ask it to do things for you - the interface to the library. When a developer is using an API, he is not accessing the whole code but the public part, so he cannot modify any component or behavior.

Both libraries and SDKs can be bundled into applications but API connections should be coded by the developer.

Before focusing on which features and task will be coded apart from the application in form of libraries and SDKs, it would be helpful to have a general idea on how most complex Android applications are structured nowadays. The modern and most common approach of Android applications architecture is shown on the next figures.
This is the first level and corresponds to the distributed version. When the developer has finished the application and distributed it, the application is like a black box. The final users interact with it without knowing what is happening inside. They don’t care nor know if it is storing the data on a cloud server or dumping it into a local file, or if the images are loaded using a certain library or the developer has coded its own classes.

If we go deeper into the second level, we can see we have two clearly distinct parts, which are the client and the backend. The client is the package we are downloading via any app
store and installing on our device. It’s the part which will be used by the final user and contains all the required systems to provide the input, data management and output.

The backend is the server side of the application, and as in most embedded or web applications it is invisible to the final user and is only used by the client to manage the information systems. Although most backends are developed based on common frameworks oriented to construct server side systems, most times a company has to develop a new backend from scratch for every one of its applications. For example, the server side of a chat messaging application has nothing to do with a music streaming application. So when developing an application, we have to take care of both sides.

What’s more, in most modern platforms we have third-party players: libraries, APIs and SDKs.

If we look at the first image, we can see that external APIs are present across both client and backend. However, they are not bundled but connected with them. On the other hand,
if we look at the second image we can observe that libraries and SDKs are placed and packaged inside the client and server.

Well-known examples of Android APIs and SDKs are the Google Play Services, the Facebook SDK and Twitter’s Fabric. They allow your application to connect with these platforms, interact with them, send and retrieve data and many others only by implementing some snippets of code in your application which are usually already provided in the documentation of the API or SDK.

In this project, we are going to use external APIs in the backend and will construct some libraries to be bundled with the Android client. Since we will skip the social integration (login with social networks, invite friends, play against them, social ranks...) on the first version of the product, the only SDK we are going to use is the Android SDK itself, which is used in any Android application. We will see in details in the following sections.

2.5.1 Application Requirements

Below are listed the common application requirements that we are going to implement in separated support libraries. This is mainly because these requirements are common across many applications and we will reuse them in other company projects from now on.

Internet connection
The application needs to establish connection with a remote server which will act as backend and store information related to the users, games, stats, etc. It will also require a system to request information and handle the response.

Asynchronous execution
The application will execute several processes and tasks in asynchronous mode to improve the user experience and avoid waiting and loading states as much as possible. It will also need support to execute callbacks and handle the result of the asynchronous processes.

Heuristic system support
One of the main goals of the project was to give some kind of intelligence to the application. Although heuristic systems cannot be exactly considered intelligent and there is a lot of debate about them, it is true that they can be used to evaluate the quality of a certain decision.
We are going to use a heuristic system as an approach to problem solving and learning that employs a practical methodology not guaranteed to be optimal or perfect, but sufficient for the immediate goals because it is impossible to find an optimal response. This system will provide an interface to define actions that should be executed under certain circumstances, such as sending notifications, giving items away and displaying information related to
purchases, wins or victories to the user for example. We will see in later sections how it works in detail and which heuristic will we implement in the game.

2.5.1.1 Http Client

One of the most common requirements of any Android application is the ability to communicate with a backend. As it has been previously explained, most modern applications need to communicate with the information systems which are running in servers or cloud, out of the application.

The easiest and most common way to set up this communication channel is by making http requests from the client, handle the request in the backend and return a result to the client in order to process it.

When deciding which features will be available in our client first we have to keep in mind how the communication between the client and backend will work. We need to store information related to the matches and users and access it later so we will be making POST and GET requests. We also need to identify the user or match when fetching the data so we will need to add parameters to the requests. We need to handle the errors and format and display the obtained data so the client needs to be able to run callbacks and retry requests.

So, our Android client needs the following features:

- Create the Request object
- Define the HTTP method of the request (GET, POST, PUT, DELETE...)
- Set the url and endpoint of the request
- Add parameters in the headers and the post/get fields
- Retry policies and error handling
- A network cache
- Success and error callbacks to collect the server response and work with it

All these features are usually implemented in an HTTP client which encapsulates these functionalities and provides easy and fast access to them. If we look back at the application structure diagram and explore the first level of the client, we can see that the module in charge of the communication with the backend is the HTTP Client.
When speaking about the project, one of the principal requirements of the application is to create a client to make http requests in order to communicate with the backend. The main goal of this part of the project will be to design and develop this client with a sufficient level of abstraction so it can be used in other projects carried out by the company.

Because this is a common requirement there already exists many HTTP clients and libraries available to Android developers. Therefore our client will be based in one of those widely-used clients instead of on a completely new client. This way, we can extend it to add more features and also take advantage of a robust, tested core. So, with all its requirements stated, in the development stage we are going to create a stand-alone HTTP Client which could be used in many projects and covers all those requirements.

### 2.5.1.2 Asynchronous support

Every modern application needs to do work in the background in a way it doesn’t affect the user experience. These kind of task can range from image loading to network requests or data manipulation. In a game where those kind of processes take place frequently, it is important to control that they do not block any screen or affect the user in negative ways such as slowing down the device or displaying too many loading dialogs.

The best way to avoid this behaviors is to execute these tasks in separate threads. On Android, there are two distinct types of threads.

When an application is launched, the system creates a thread of execution for the application, called "main." This thread is very important because it is in charge of dispatching events to the appropriate user interface components. It is also the thread in which your application interacts with components from the Android UI toolkit. As such, the main thread is also called the UI thread.
The others are the worker threads. Because of the single thread model described above, it's vital to the responsiveness of an application's UI that it does not block the UI thread. If the application has operations to perform that are not instantaneous, it should be done in separate threads ("background" or "worker" threads). However, we have to be careful with the actions we carry out in worker threads, because they must not violate the first rule of Android's single-threaded model: do not access the Android UI toolkit from outside the UI thread.

To sum up, the features we need from the asynchronous support library are the following:

- Create worker threads and execute blocks of code on them
- Control the number of threads by setting a fixed thread pool to avoid overwork the device
- Log the state of the threads
- Run customized pre-execution methods before the thread’s main action
- Run post-execution based on the result of the thread main action
- Define error and success callbacks
- Define whether the callbacks should run on the UI thread in case they modify UI components
- Close the thread pool and decide whether to await the termination of the threads that are already being executed

Although these features can be implemented with the Java threading API, its usage is confusing and tedious. There is no encapsulation for fast access to create new threads with result collection and callbacks, and the amount of code required to make a UI callback from a worker thread in Android is just ridiculously high.

Our goal is to create an easy to use threading library that allows us to access the java threading API as well as the Android UI handlers without writing all the code required by the native solutions and focus only on the actions and callbacks to be executed.

The application will use these asynchronous features in several tasks: load the profile images of the rivals, execute heavy UI updates, load content from the internet, prefetch several contents to provide a fastest execution and many others which will be explained in detail in the development section.
2.5.1.3 Heuristic system

The most complex and hard to abstract feature we need is the heuristic system that allows us to carry out actions based on the available information in a certain point of the execution or the knowledge acquired by the system while it has been running on the device.

Given that one of our most criticized points of today’s applications is the lack of common sense and adaptability to situations, we want to demonstrate the point that we can create a set of rules that we can use to decide whether it is a good idea or not to carry out a certain action, for example sending a push notification to reengage an user or give away some items because the player loses most of the games and may uninstall the application.

When thinking about these situations, we identified a big problem: these rules are exclusively based on our own perceptions and experiences. Why shall a player prefer to be reengaged at home rather than at work? Well, actually we don’t know. We have the perception that exists certain behavior patterns and that there are situations where certain actions are more suitable than others, but that’s all. Heuristics are not an exact science, but it is out of question that exist certain patterns that can be easily generalized so we can use heuristics as a strategy to profiling. Profiling stands for the extrapolation of information about something, based on known qualities. So, the main task of our heuristic system will be execute actions when the profile of the device or the user meets certain conditions which will be supplied by the developer.

For every heuristic, we need to:

- Define the main action to be executed
- Supply the list of conditions, their weights and scores
- Set whether the heuristic requires or not to be learned
- If it has to be learned, add the learning threshold and learning function
- Define the range and/or minimum score to execute the action

By doing these actions, we can run the heuristic process we have in mind: check the preconditions, and if they are met and it’s the best action available, carry out the main action.

For example, in order to send a reengaging push notification we want to check if the phone is not in charging state and we have not send any reengaging push this week. We can see that in this case the second condition is stronger than the first, so we are going to put more weight on it. If both conditions met, we will surely send the push. If the first is met but not the second, we won’t send it and if the second is met and the first not we might send it anyways.
Later on we will dig deeper into the technological approach used to construct this system and which heuristic are we going to implement on the first version of the application.

2.6 Product recap

At the beginning of this chapter, we defined the idea of the game and used the MDA Framework as a formal approach to design the mechanics and the rules of the game. We listed all the mechanics and rules as well as the dynamics we hope they generate when the users play. We also listed a set of aesthetics that we expect to evoke on the user when playing and the dynamics that cause them.

Later on we formalized the mechanics and rules and created the first version of the game flow as well as a flowchart to help understand it. When the game flow was defined, we started drawing the first mockups of the application screens and then made the in-app design counterpart of each of them.

After the design stage was finished, we came back to the product to work on the most technical part and defined the support systems that we have to construct and bundle into the application as well as their features: the HTTP client, the asynchronous support system and the heuristic rules system.

The next step is to start with the development stage by explaining the application layers and then move to the architecture and the coding.
Chapter 3. Application development

Chapter 3. Application development ................................................................................. 51
3.1 The clean architecture ................................................................................................. 52
3.2 Presentation layer ........................................................................................................ 55
3.3 Domain layer ................................................................................................................ 60
3.4 Data layer ..................................................................................................................... 64
3.1 The clean architecture

We know that writing quality software is hard and complex: it is not only about satisfying requirements, our code also should be robust, maintainable, testable, and flexible enough to adapt to new scenarios, new features and growth in general terms. This is where building our project under a clean architecture comes up and could be a good approach for using when developing any software application, mobile or not.

The idea is simple: clean architecture stands for a group of practices that produce systems that are:

- Independent of Frameworks
- Testable
- Independent of UI
- Independent of Database and Data Storage
- Independent of any external agent

![Clean architecture dependency model](image)

It is not a must to use only four circles as the picture shows, because they are only schematic but one should take into consideration the Dependency Rule: source code dependencies can only point inwards and nothing in an inner circle can know anything at all about something in an outer circle.
Here is some vocabulary that is relevant for getting familiar and understanding this approach in a better way:

**Entities**: These are the business objects of the application. In our application the entities are the game objects and game concepts representations such as the items, words, players, etc.

**Use Cases**: These use cases orchestrate the flow of data to and from the entities. Are also called Interactors. In our application are represented by the game flow and mechanics.

**Interface Adapters**: This set of adapters convert data from the format most convenient for the use cases and entities. Presenters and Controllers belong here.

**Frameworks and Drivers**: This is where all the visual go: UI, tools, frameworks, etc. In an Android application these are represented by the activities, fragments, views and XMLs among others.

The final goal is the separation of concerns by keeping the business rules not knowing anything at all about the outside world, thus, they can be tested without any dependency to any other external element.

To achieve this, our proposal is about breaking up the client into three different layers, in which each one has its own purpose and works separately from the others.

It is worth mentioning that each layer uses its own data model so this independence can be reached and we use data mappers and transformers between every layer. This is done to avoid the necessity of using the same data models over the entire application.

The following schema represents how our layers will be separated:

![Figure 4 Layer separation and bounding elements in clean architecture](image-url)
3.1.1 Presentation Layer

The presentation layer is where the logic related with views, user interface, graphics and animations happens. We used a Model View Presenter model. We will not get into details on it, but here fragments and activities are only views, there is no logic inside them other than UI logic, and this is where all the rendering stuff takes place. In our application, the Presenters in this layer are composed with interactors (use cases) that perform the job and come back with the data that will be rendered in the view.

The model

In an application with a good layered architecture, the model would only be the gateway to the domain layer or business logic. If we were using a clean architecture, the model would probably be an interactor that implements a use case. For now, it is enough to see it as the provider of the formatted data we want to display in the view.

The View

The UI, usually implemented by an Activity (it may be a Fragment, a View... depending on how the application is structured). In Android, the XML files that define how the activities and views are displayed can also be considered part of the UI. The view classes and objects will contain a reference to the corresponding presenters. The only thing that the view will do is calling a method from the presenter every time there is an interface action (a button click for example).

The presenter

The presenter is responsible to act as the middle man between view and model. It retrieves data from the model and returns it formatted to the view. But unlike the typical model-view-controllers, it also decides what happens when you interact with the view. In this application we will call those classes Presenter Controllers. Our Presenter Controllers will also contain the Interactor’s logic which is the element between the Presentation and the Domain layers.

3.1.2 Domain Layer

The domain layer is where the logic processes of the application happen. In this layer there is no Android components, only regular Java elements. The entities and the game objects
are part of the domain, so we must isolate these entities in order to work with them without Android dependencies.

The core mechanics of the game are programmed on those entities and their respective controllers, where the interactors are implemented.

For example, when answering with a new word, the corresponding instance of the word will be created and passed to the WordController that will check whether the word is correct or not and send the reply to the PresenterController at the UI who is waiting to show the red or green button.

We have to keep in mind that if the presentation layer is independent of the domain, the domain layer is also independent from the source and acquisition of the data, so the data layer is only accessed through the connection between the entities controllers and an interfaced RepositoryController.

3.1.3 Data Layer

The idea behind all this is that the data origin is transparent for the other layers and the client, which does not care if the data is coming from memory, disk or the cloud, the only truth is that the data will arrive and will be got.

In this case, since the data is retrieved from our cloud backend, we will need to work hard in make the data acquisition process as fast as possible so the user doesn’t feel stuck waiting. That’s why we will prefetch as many data as possible and use Android disk preferences when we could avoid remote calls among other performance improvements.

3.2 Presentation layer

If we dig deeper on how the presentation layer of the application is constructed, we will quickly notice about the strong separation of every element of the MVP model: the activities, the adapters and the controller.
The activities are quite clear in this first version. They take advantage of the hierarchy system and extends from ToolbarActivity or BaseActivity. ToolbarActivity is an abstract activity class that implements all the mechanisms be supplied and display a native Toolbar, the element that replaced the ActionBar in Android 5.0. Even though it is a new class we implemented it with Google’s official support library so it can be used in older versions of Android without any issue. The BaseActivity is a simple wrapping implementation of a blank Activity.

Furthermore, all our activities correspond with one of the game screens, so it is pretty easy to identify and understand them. As we explained in the clean architecture section, in the activities there is no logic implemented: only the methods related to the inflation of the views. All the operations related to the acquisition and accommodation of the data to be displayed is delegated to the PresenterController.

Similar happens with the adapters. Adapters in Android are objects that act as a bridge between an AdapterView and the underlying data for that view. The Adapter provides access to the data items and is also responsible for making a View for each item in the data set.

The scrollable views which require adapters in our application are all implemented by using the new RecyclerView class, therefore our adapters all extends the RecyclerView.Adapter inner class.

These adapters are very similar to the usual Android approach where the class that is extending the adapter interface or abstract class implements the methods that get the number of elements and return the view inflated in every position of the adapter. Even though this was the most easy to use and commonly seen in old Android applications, the regular implementation of Adapters is no longer in use because when Android-running devices grew in speed and memory a very useful pattern appeared and transformed the way developers worked with adapters.

It was the so-called ViewHolder pattern. The ViewHolder pattern works by using a static inner class inside the adapter to keep the already created views in memory and avoid creating them every time we had to re-render the position, which caused a faster scrolling.
with the deacceleration effect that happens when the user is scrolling fast and suddenly stops. It provided a much faster navigation in adapters and an easier implementation of the view’s inflation method. However, it caused troubles in devices that were not powerful enough to keep all the data in memory without causing memory heap exceptions, so the developer had more responsibilities when implementing this pattern because he had to deal with this kind of failures and keep the memory safe by recycling the old or unused views, a mission which was really hard to accomplish.

When Android 5.0 Lollipop was released, one of the most interesting new features was the RecyclerView class. It natively provides the behavior of the ViewHolder pattern in adapters and automatically takes care of the process of recycling the old, unused views, so it’s a win-win situation for the developer.

So, as we previously said all our adapters are implemented with the RecyclerView classes and so we can expect them to work fast and efficiently. As well as we did with the Toolbar, we implemented them extending from the support library instead of the official API so we can have backwards compatibility in older devices.

Later in the results section we can see a CPU and memory heap analysis which are much related to the use of RecyclerViews instead of the old classes.

The PresenterController is the class that connects the UI elements (Presentation layer) with the entities and its controllers (Domain layer). It acts as a simple proxy that collects data from certain entity controllers and format it to supply the adapters or the views. A simple example is how we display the points, victories and defeats in the main screen:

```java
public static class VHHeader extends RecyclerView.ViewHolder {
    protected ProximaNovaTextView mainPoints, mainVictories, mainDefeats;

    public VHHeader(View v) {
        super(v);
        mainPoints = (ProximaNovaTextView) v.findViewById(R.id.mainPoints);
        mainVictories = (ProximaNovaTextView) v.findViewById(R.id.mainVictories);
        mainDefeats = (ProximaNovaTextView) v.findViewById(R.id.mainDefeats);
    }
}
```

Snippet 1 RecyclerView TextView holder element with PresenterController supplying data

As we can see, the retrieval of the stats is invisible to the UI and delegated to the PresenterController. However, if we look at how these methods are implemented in the PresenterController, we can observe that its only task is retrieve it from the StatsController from the Domain layer and format it (transform the integer to a string in this case) so it can be used in the UI method:

```java
public static String getPoints() {
    return StatsController.getPoints() + "";
}
```
What we achieve by using this approach is the total independence of the UI, Domain and Data layers because if the UI changes, we only have to add code to the UI packages and same with the other layers. A change in the domain or the data storage (for example, changing from stubbed hardcoded data to cloud-stored data) should not affect the other layers at all.

Finally, another important matter when creating and programming a UI on Android is to consider the whole specter of screens and devices. Android is a very heterogeneous environment and if a developer creates an application that is only tested and constructed to work in a concrete device (for example, the device used for debugging the application), it is highly probably that the app doesn’t run smooth in most devices. We have to consider several factors such as the resolution of the screens, the density pixels or even the size of the fonts. To do this, we used a brand-new software called Zeplin that adapts a Sketch design in order the developer can extract and use all the drawable images and elements.

Furthermore, Zeplin creates the dimensions xml files for each screen density and size. This way, the developer can focus on construct the UI and test it on a single device and after it use the files provided by Zeplin in the application so it is automatically scaled to every screen. We followed this process and the result can be seen in the next pictures, which show the main screen in devices with lower resolutions than the used when developing.
Figure 5 Slang main screen in Nexus 4 (768x1280 px)

Figure 6 Slang main screen in Samsun S3 mini (480x800 px)
3.3 Domain layer

Game rules and mechanics are here: all the logic happens in this layer. The controllers of each one of the entities take care of the corresponding logic of the component as well as to obtain the data from the data layer.

This layer is a pure Java without any Android dependencies. All the UI components use the PresenterController to access the domain. However, when speaking about the domain of the game, we have to separate the game objects entities which control the flow (players, items, words...) from the app-related entities (stats and profile mainly).

![Diagram of Domain layer representation and bounding elements](image)

The game objects all implement certain conditions and methodologies that uniform the behavior of the objects across the backend and the application in order to help with the implementation and execution of the logic and rules, while the stats and profile are only Java objects connected to the data layer.

But how is the domain organized? How is the relationship of the different game objects?

In the following UML chart we can know more about how these entities are related to each other. These relationships try to be the more accurate and similar to the game flow as possible, because at the end they are the responsible of the logic and mechanics, which end up deriving in the game flow.
These relationships are implemented in the entity game objects. The most important about the entities is that they implement a couple of interfaces that, as explained above, uniform the behavior by making some common methods and allow the objects to be read, decoded and handled in a very convenient way.

**Textual Restrictions**
The player of a turn must be one of the players who are playing the match
3.3.1 Domain Interfaces

One of the most important traits of any object oriented language are inheritance and polymorphism. Even though Java is one of the least advanced languages on these topics because of the lack of multiple inheritance (it can be achieved by other ways but it is not natively supported), no data layer can be conceived without taking advantage of these two powerful mechanisms.

The Entity is the top element in the class hierarchy of every game object. It provides a lot of powerful features such as the already mentioned hierarchy, which is useful to add methods to every object and extend them. It is also very convenient when using common methods since we can use the Entity class as the return type when required. Since Entity is an abstract class it is not instantiable but we can use it to force the implementation of certain methods and therefore have some behaviors shared across all the game objects. Having a custom top hierarchy class is also very useful to test the application and work with other important methodologies such as dependency injection or library stubbing.

The only abstract method on the Entity class is the isValid method. It returns whether an Entity object has all its required fields and correct values assigned or not.

Finally, one of the handiest usages of the features provided by the Entity hierarchy is that we force the implementation of the Jsonable interface to every game class.

The Jsonable interface is present in all the entities of the application because the Entity class implements it and as an abstract class Entity doesn't actually need to implement it - but their subclasses do. The Jsonable interface allows an object to be encoded and decoded as a JSON Object. The class that implements the Jsonable interface has to implement the following method signatures:

```java
JSONObject toJSONObject() throws Exception;
void restoreFromJSONObject(JSONObject jsonObject) throws Exception;
```

The first one should return a JSON Object representing all the fields and important data of the object. The keys of each field are presented in the class as public final static Strings because they are associated with the class itself rather than the instance of the class so we can assure that every object that belongs to a certain class can be encoded and decoded using the same functions.
The second one does the inverse work and decodes the object recovering its fields from the saved JSONObject. In this function, the most natural and aesthetic behaviour would be to return the object itself or the superior class which should be the Entity, but to avoid casting and templating we decided to use it as a non-return method which could inflate the object retrieving the fields and using the class setters. By doing this we avoid large amounts of code to determine the class of each object and we can still easily decode them and have fast access by using the following methodology:

```java
Player p = new Player(); //empty constructor
p.restoreFromJSONObject(playerJSONObject);
//we pass the JSONObject containing all the player fields, the object is now usable
```

Snippet 4 Example of PlayerEntity create and restore methods

To be sure we are never using entities which are not initialized or restored we use the Entity’s interface isValid method. We will only set an entity as valid at the end of the parametrized creator or the restore method. This way we can ensure that no invalid or corrupted object is used during the execution.

An example that sums up the behavior of the Jsonable interface in a game object is shown below:

```java
PlayerEntity p1 = new PlayerEntity(id1, "Marc");
PlayerEntity p2 = new PlayerEntity(id2, "David");

List<PlayerEntity> players = new ArrayList<PlayerEntity>();
players.add(p1);
players.add(p2);

WordEntity word1 = new WordEntity("hello");
WordEntity word2 = new WordEntity("lobster");
WordEntity word3 = new WordEntity("eradicate");
WordEntity word4 = new WordEntity("tea");

WordEventEntity move1 = new WordEventEntity(word1);
WordEventEntity move2 = new WordEventEntity(word2);
WordEventEntity move3 = new WordEventEntity(word3);
WordEventEntity move4 = new WordEventEntity(word4);

TurnEntity turn1 = new TurnEntity(p1, move1);
TurnEntity turn2 = new TurnEntity(p2, move2);
TurnEntity turn3 = new TurnEntity(p1, move3);
TurnEntity turn4 = new TurnEntity(p2, move4);

List<TurnEntity> turns = new ArrayList<TurnEntity>();
```
turns.add(turn1);
turns.add(turn2);
turns.add(turn3);
turns.add(turn4);

MatchEntity match = new MatchEntity(match_id, turns, players);

This simple data stub creates all the required entities to simulate a match between two players who answered two times every one of them. As we can see, the creation of the entities is very simple and don’t require parameters that are (or can be) calculated or depend from other sources (the correspondent controller, for example).

### 3.4 Data layer

The data layer is still to be fully implemented. It will be the last element of the game to work as intended in the first version, mainly because we have been using a fake data layer with data stub along all the development process.

However, the behavior is defined and the tasks are very clear. The data layer will use the HTTP client to connect with the backend and retrieve the required information, then notify the controllers which are waiting.

This information will be always formatted as a JSONObject so we can take advantage of the Jsonable interface and avoid the step of translating the data from the backend structure to the concrete entity structure.
So, our data layer will mainly consist of a set of network calls and their respective callbacks which will save it to disk or cache and notify the controllers that the data is finally available. In case we are not able to retrieve the data and it’s a critical part, such as the active games, we will notify the player properly and ask them to try again later.

The connection between the domain and the data layer is made through the RepositoryController, which is where the data retrieval is ordered and served when available.

As we explained before, we take performance very seriously and that’s one of the reasons we are still working on the data layer, both from the client and server side. We want to be able to give automatic response in certain points of the flow in where a fast server response is critical.

For example, when we are verifying if an answer submitted by the player is correct or not, we don’t want to make a network call to check it but immediately accept or reject the answer. So, we have to prefetch all the valid words at the same time we set the game in “Answer” state. If one of the active games of the player is answered by his or her opponent with the word “hello”, when the player opens the app and the data layer retrieves all the active games including the one we are using for this example, it will also retrieve all the words starting by “lo” and “ol” (because of the swap item). Therefore when answering to the word “hello” sent by his opponent the application will be able to determine whether a word is valid or not in runtime.

This is just one of the examples but there are more. Implementing a cron task in the client that takes care of the profile-related data and sends it to the server, fetch all the information when the user opens the app instead of lazy loading from the backend and load the profile images and other resources asynchronously by using our library are other performance improvements that are on the to-do development list before the official Play Store release.
Chapter 4. The support libraries

Library and SDK design overview ........................................................................................................... 66
The Slang Libraries .................................................................................................................................. 68
Skynet ...................................................................................................................................................... 71
Hydra ...................................................................................................................................................... 77
Hal .......................................................................................................................................................... 82

4.1 Library and SDK design overview

There are some common traits in every great library or SDK. First of all, we have to keep in mind that the main reason to create an SDK or a library is to separate independent code and functionalities from concrete product features and classes. Besides this, as a developer, I would say that another important reason for creating an SDK is giving other developers the opportunity of using your robust and tested code instead of creating their own implementations and solutions. To sum up, a state of the art SDK should have the following traits:

Ease of integration, easy to use, hard to misuse

If an SDK or library is hard to integrate and use, you either won’t save time by using it or don’t want to add to your application. Having functionalities in an SDK or library should always lead to an easier use or implementation of these functionalities. If using the library is harder than creating your own implementation, it is just useless. Also, it should give little space to misuse and misunderstandings. It should be clearly documented, with an explanations and solutions to frequent problems.

Extensible and consistent

The user should be able to interact with it in certain ways. For example, it is a good practice to provide a default implementation of callbacks and exception/error handling but also giving the opportunity of override those callbacks to the user. Same thing happens with default configurations, debug modes, etc. This directly leads to the consistence property. Your SDK or library should include all support classes and methods in order to let the user use all its features and none related to external or non-used features. So, if you are creating a new REST client, you should include all the necessary code to make requests,
include parameters, and implement response and error callbacks and everything in order to fulfil all the user's requirements. And of course, don’t include a Facebook login feature because nobody is expecting that. This may sound common sense but you can see many examples of these kind of bad practices out of there.

**Useful and real time saving**

If some feature or action can be carried out with less than 100 lines of code, it’s probably better to give a code snippet or an implementation guide rather than creating an entire SDK for it. Before creating a new SDK, it should be considered whether it is necessary or not. Can you achieve the same result by using a single method or class? Are there any other SDK or library that already does the same thing? Will this feature be used in more than one application? These are some questions that should be answered before creating a new project of this kind.

**Lightweight and robust**

Keep things light. Don’t include other libraries or resources such as images or sample xml files. In case they are required, try to compress the images and make the final project as light as possible. And of course, make it fast and robust. Nobody wants their application or project crashing because of an external SDK or library is not working as intended. If my application is crashing due to an SDK I’m using, I would probably want to slap the guy who made it. Catch exceptions at every non-trivial piece of code and call hierarchies. Isolate functionalities in a way that if you can’t do your work because there is no network connection or no disk space available, the developer and/or the user receive appropriate feedback without suffering from crashes. Last but not least, keep in mind your initialization and performing times and thread them if necessary. Remember that if you block the UI thread for more than five seconds in Android, the SO will show the infamous “App not responding” dialog and we will be back with the “I want to slap the guy who made this”.

4.2 The Slang Libraries

Slang has 3 libraries bundled with it as we have explained in the product design chapter. All these libraries use Android classes so they cannot be used in non-Android Java projects.

The first library is called Skynet in allusion to the global cyberspace software from Terminator. It is a RESTful client used to make HTTP requests and underlies on Google’s Volley project.

The second library is Hydra, named after the many-headed serpent in Greek mythology. It is a thread manipulation/encapsulation library which provides easy and fast access to create and execute asynchronous blocks of code.

Finally, Hal is a simple approach of a heuristics system which can be used to calculate the likelihood and correctness of a certain decision or action. The name Hal comes from HAL 9000 (Heuristically programmed ALgorithmic computer), the AI that controls the spacecraft in the Space Odyssey series.

We will see every one of them a little bit further down the road.

4.2.1 Libraries design pattern

All Slang support libraries are implemented using the singleton pattern. This kind of pattern is used by most of the Android SDKs and libraries mainly because of two important properties. First is that the singleton is always reachable from all the parts of the application without the need of taking care of the app lifecycle. Secondly, it guarantees that there will be only one instance of particular object in the application.

So, in order to implement the singleton classes for every one of our Java libraries, we used the enum approach, which is the best way to implement an on-demand serializable singleton. Although enumerations in Java have many preconceived expectations, they are really much more than a list of symbolic constants.
public class MySingleton {
    private static class SingletonHolder {
        public static MySingleton instance = new MySingleton();
    }

    public static MySingleton getInstance() {
        return SingletonHolder.instance;
    }

    public void foo() {
    }
}

MySingleton.getInstance().foo();

public enum MySingleton {
    INSTANCE;

    public void foo() {
    }
}

MySingleton.INSTANCE.foo();

Snippet 6  Old public field singleton and new enum approach with access example

The enum approach is functionally equivalent to the public field commonly accepted approach, except that it is more concise, provides the serialization machinery for free, and provides an ironclad guarantee against multiple instantiation, even in the face of sophisticated serialization or reflection attacks and is also more memory-efficient that the holder singleton pattern which is important in the Android environment. While this approach has yet to be widely adopted, a single-element enum type is the best way to implement a singleton.
4.2.2 Special Considerations

First, it is important to take into account that the Android minimum SDK of the libraries and the game are not the same. This is because the libraries do not use modern Android APIs as the game does, so it is not necessary to set up a more modern minimum Android version. We want to release these libraries and make them available for everyone who wants to use them, so we will set up a real minimum version and not the same used in the game.

Secondly, as we said we want to use these libraries in other projects and make them available for everyone, so we will compile them into a jar and upload both the jar and the entire project on Github and Bitbucket. This is possible because even they are Android libraries they do not use any XML resource such as layouts or drawables, so its content can be bundled and distributed into a jar.

Finally, we have to keep in mind that there are several improvements, methods and features that could fit more in the libraries rather than in the game implementation of their interfaces, but since we want to publish and make them open we will keep those features in the game, separated from the libraries.
4.3 Skynet

4.3.1 Overview

Skynet is an HTTP RESTful client that underlies in Volley, a Google's library for network communication. It encapsulates Volley methods by using a simple request creator where only the method, endpoint and parameters have to be specified. Skynet is able to make GET, POST, PUT and DELETE requests and collect a JSONArray, JSONObject or plain text response. It also provides a request timeout implementation which the user can modify in order to define his own timeout requirements by modifying the default timeout, the number of retries and the backoff multiplier. User can also define a map of keys-values to be sent as post parameters for every request. Finally, the user is allowed to set a fixed domain that will be used as the base URL for every request.

As it has been previously explained, Skynet acts as a wrapper of Google’s Volley Library. Volley offers the following benefits:

- Automatic scheduling of network requests.
- Multiple concurrent network connections.
- Transparent disk and memory response caching with standard HTTP cache coherence.
- Support for request prioritization.
- Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.
- Ease of customization, for example, for retry and backoff.
- Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
- Debugging and tracing tools.

As we can see, Volley itself provides all we need to work with HTTP requests and handle their results and we could do everything that the application requires by using only the Volley’s jar.
But Skynet is very important and handy because it provides a fast, easy to use and flexible access and usage of Volley features as we are going to see in the next section.
4.3.2 Usage

If we take a look at how Google explains in their website how to use Volley, the first recommendation is that the developer should create a Singleton class to hold the RequestQueue object and create the getters and initialization methods. Instead of implementing an entire Singleton for it, Skynet encapsulates the RequestQueue and other support objects in its own enum element, which as previously stated it is the best and most efficient implementation of the singleton pattern in modern java. So, we can access our RequestQueue at any time just by using the accessor Skynet.INSTANCE.

Besides this, there is no way to fix a main URL for the requests in Volley. So, if we want our requests to attack a fixed base domain, which is the usual behavior if we are connecting to our own API or Backend, we have to write the entire URL every time we declare a new request. Skynet offers an easy mechanism that adds a base URL which can be configured just by calling one of the overloaded init methods, for example:

```
Skynet.INSTANCE.init("http://httpbin.org")
```

```plaintext
Snippet 7 Skynet initialization method to set a base URL
```

This way our RequestQueue always uses httpbin.org as the base URL and we only have to define the concrete endpoint at every request. Also, as we said the init method is overloaded and can be used to define a global request timeout and retry configurations. Volley lets the user to specify custom timeout and retry policies but it forces to implement it on every request. Skynet allows the user to set a new global timeout and retry configuration. The init method signatures are listed below:

```
init(String baseURL)
init(String baseURL, RequestTimeoutConfig timeoutRequestConfig)
init(RequestTimeoutConfig timeoutRequestConfig)
```

```plaintext
Snippet 8 init method signature overloading
```

So, as we can see there are some differences between using Volley natively versus using Skynet. But the most important difference is on the request creation and definition process. Volley allows the user to make String, JSONObject and JSONArray requests by GET, POST, PUT and DELETE HTTP methods. A common example of a Volley’s String request object creation looks like:
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            // Do something with the response
        },
    },
    new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            // Handle error
        }
    });

Snippet 9 Volley's simple StringRequest creation

We use the StringRequest class to create a new object and in the creator we define the url, the HTTP method and override the two ResponseListener callbacks. This creator is not overloaded but we can put null callbacks if we don’t need any. What is inexplicable in Volley is that headers and request parameters are only present at certain request types. For example, the String Request do not allows any parameters at all, however JSON requests do accept them.

Skynet’s mission is to make the request creation easier and more consistent. To do that, Skynet provides the RequestCreator static class that acts as a request creator handler that wraps and overloads all the requests types and methods present in Volley. For example, the next snippet shows how to make the same request object as shown above in Skynet:

RequestCreator.getRequest(ResponseType.STRING, url,
    RequestMethod.GET, new SkynetRequestCallback<String>() {
        @Override
        public void onResponse(String response) {
            // Do something with the response
        }
        @Override
        public void onError(VolleyError error) {
            // Handle error
        }
    });

Snippet 10 Skynet’s simple StringRequest creation

The code is pretty similar except that if you don’t care about type check warning you don’t need to define the template type and you can avoid the cast (although it is a bad practise).
The advantage of Skynet is that it overloads the methods and provides a common implementation for every request type which makes it more versatile and hard to misuse. These are the available methods to create a Request in the RequestCreator class:

```java
getRequest(ResponseType type, String endpoint, RequestMethod method)
ggetRequest(ResponseType type, String endpoint, RequestMethod method,
            Map<String, String> postParams)
ggetRequest(ResponseType type, String endpoint, RequestMethod method,
            SkynetRequestCallback<?> callback)
ggetRequest(ResponseType type, String endpoint, RequestMethod method,
            SkynetRequestCallback<?> callback, Map<String, String> postParams)
```

Snippet 11 getRequest method signature overloading

We can decide whether to implement the callbacks or use the default ones that only logs the error and response and whether to add parameters to the request. Finally, we have to keep in mind that if we defined a custom timeout and retry policy in the instance, it will be applied to every newly created request.

4.3.3 Performance Tests

To test if our HTTP client was fast and reliable enough to be used in both our game and other systems of the company, we run a sample of tests which were mainly pointed to check the requests do not fail and to test the speed of the client.

First we faced Skynet against the cURL UNIX command in order to see its speed and reliability against one of the most widespread methods to make HTTP requests in Unix systems. We were connected to the same Wi-Fi network (50 MBPS) and made the GET and POST (with two parameters) calls to the httpbin’s test service by using the following commands:

```bash

time curl --data "param1=value1&param2=value2" -i http://httpbin.org/post --no-sessionid --no-keepalive
```

Snippet 12 cURL commands used to make the HTTP requests

As it can be seen, in the command we added options to avoid caches and session ids, so every time we execute the call is like the first time.
In Skynet, we created the equivalent requests and made a simple application that fired them when a button was clicked. The Skynet test ran in a Nexus 5 running under Cyanogen Mod 12 and the cURL test ran in a MacBook Pro 2013 running under Yosemite OSX. Every experiment consisted of ten rounds. A round consisted of five repetitions and the value selected as the time of a round was the mean of the five repetitions. In total, we made 5 repetitions x 10 rounds = 50 calls. This methodology was applied for every device (the laptop and the smartphone) and for every method (GET and POST). We obtained the following results:

<table>
<thead>
<tr>
<th>ROUND</th>
<th>DEVICE</th>
<th>NETWORK</th>
<th>ELAPSED CPU TIME (MS) WITH GET</th>
<th>ELAPSED CPU TIME (MS) WITH POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>271</td>
<td>282</td>
</tr>
<tr>
<td>2</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>260</td>
<td>238</td>
</tr>
<tr>
<td>3</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>232</td>
<td>243</td>
</tr>
<tr>
<td>4</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>233</td>
<td>260</td>
</tr>
<tr>
<td>5</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>232</td>
<td>228</td>
</tr>
<tr>
<td>6</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>235</td>
<td>230</td>
</tr>
<tr>
<td>7</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>254</td>
<td>239</td>
</tr>
<tr>
<td>8</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>248</td>
<td>236</td>
</tr>
<tr>
<td>9</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>246</td>
<td>267</td>
</tr>
<tr>
<td>10</td>
<td>LAPTOP</td>
<td>WIFI</td>
<td>243</td>
<td>243</td>
</tr>
<tr>
<td>MEAN</td>
<td></td>
<td></td>
<td>245.4</td>
<td>246.6</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td>13.23</td>
<td>17.45</td>
</tr>
</tbody>
</table>

Table 9 MacBook HTTP requests performance results with cURL

<table>
<thead>
<tr>
<th>ROUND</th>
<th>DEVICE</th>
<th>NETWORK</th>
<th>ELAPSED CPU TIME (MS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PHONE</td>
<td>WIFI</td>
<td>205</td>
</tr>
<tr>
<td>2</td>
<td>PHONE</td>
<td>WIFI</td>
<td>251</td>
</tr>
<tr>
<td>3</td>
<td>PHONE</td>
<td>WIFI</td>
<td>220</td>
</tr>
<tr>
<td>4</td>
<td>PHONE</td>
<td>WIFI</td>
<td>196</td>
</tr>
<tr>
<td>5</td>
<td>PHONE</td>
<td>WIFI</td>
<td>176</td>
</tr>
<tr>
<td>6</td>
<td>PHONE</td>
<td>WIFI</td>
<td>185</td>
</tr>
<tr>
<td>7</td>
<td>PHONE</td>
<td>WIFI</td>
<td>169</td>
</tr>
<tr>
<td>8</td>
<td>PHONE</td>
<td>WIFI</td>
<td>200</td>
</tr>
<tr>
<td>9</td>
<td>PHONE</td>
<td>WIFI</td>
<td>184</td>
</tr>
<tr>
<td>10</td>
<td>PHONE</td>
<td>WIFI</td>
<td>204</td>
</tr>
<tr>
<td>MEAN</td>
<td></td>
<td></td>
<td>199</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td>23.72</td>
</tr>
</tbody>
</table>

Table 10 Nexus 5 HTTP requests performance results with Skynet
As we can see, we obtain similar times when running POST and GET in the same device. The difference of performance (CPU time) between the cURL and Skynet is not significant under Wi-Fi but we can see that Skynet is slightly better. This is probably because the laptop is running more network processes than the smartphone. However, if we look at the standard deviation we can see that cURL is more stable than Skynet, but as happens with the CPU time it makes no big difference.

Regarding the stability, both systems worked perfectly at the first try on every request, so we could say that in this test running on Wi-Fi our request success rate is 100%.

After this first test, we wanted to repeat the smartphone process while being connected to mobile networks. Tests were run connected to 4G, 3G and 2G networks. The speed of 4G in the region is about 80 MBPS, the 3G is about 10MBPS and 2G is about 0.5MBPS. The methodology used and the number of requests was the same that with the Wi-Fi connection, but we are showing the means directly instead of all the data to make the table more concise and easy to read.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>4G NETWORK</th>
<th>3G NETWORK</th>
<th>2G NETWORK</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELIABILITY</td>
<td>GET 100%</td>
<td>POST 100%</td>
<td>GET 96%</td>
</tr>
<tr>
<td>ELAPSED CPU TIME (MS) MEAN</td>
<td>233.9</td>
<td>418.4</td>
<td>511.6</td>
</tr>
<tr>
<td>STANDARD DEVIATION</td>
<td>39.76</td>
<td>85.05</td>
<td>299.74</td>
</tr>
</tbody>
</table>

Table 11 Nexus 5 HTTP requests performance results with Skynet under 4G, 3G and 2G

As the results show, when connected to mobile networks the performance is far less stable than when connected via Wi-Fi. The deviation values are high which means that in every request the response time can vary a lot. As we could expect, with slower connections this deviation grows, and in the 2G network it reaches a considerable deviation of one second, which would be reasonably higher if we were in movement or with less coverage.

It is also observable that the POST requests were way slower than the GET ones with 4G network (even slower than the same 3G POST request which were made some minutes later), although there is no reason to think it is not circumstantial.

Although all these data is significant, the most important value is the reliability, which stands for which percentage of the requests finished successfully. We can observe that 4G, like Wi-Fi, has a 100% success rate. However, when running under 3G network we had a timeout on a GET request, and some more in both methods when running under 2G. Anyways, those failures are not fatal because Skynet has a configurable request retry mechanism that by default repeats every request up to three times. So, even though we had some timeouts, in the second and third retries they were completed successfully and therefore no request was lost, which is very important because translated to the game it means that information related to the answers, player status or matches could be missed.
4.4 Hydra

4.4.1 Overview

Hydra is a simple thread manipulation library. It provides fast access and encapsulation of Java’s and Android threading APIs by using a custom ExecutorService and some support classes that extends the native runnables implementations. It has a fixed thread pool to reduce resources consumption and implements a system of success and error callbacks that allows the user the execution of code both on the UI thread and the worker threads started to do the background task. User has to specify a certain code to run in this thread by implementing the action method. This action method should return a boolean value or a data bundle, which specifies whether to run the post-run method (so this post-run method can be used to execute code only if the thread complies certain requirements defined by the action method). Pre-run and post-run methods are also implementable.

Hydra uses an AtomicInteger to identify the threads so the user is able to keep track of every one of the threads added to the pool.

Finally it also includes a special mechanism that allows the user to close the thread pool in two ways, the first blocks the current thread until the tasks are finished and the second just kills it and returns the list of tasks that have not been executed so the developer can take care of them.

4.4.2 Usage

One of the main reasons we developed Hydra is we need to access the Java and Android threading APIs a lot and they are tedious and hard to use because we need to write a lot of code to carry out simple out simple actions, especially those who should run callbacks and communicate with the UI thread. And everything gets worse if we want to work with the result of the thread or want execution callbacks.
In the following code snippet we can see the code to run a threaded method in Java:

```java
Thread t = new Thread(new Runnable() {
    public void run() {
        foo();
    }
    void foo() {
        try {
            Thread.sleep(5000);  
        } catch (Exception e) {
        }
    }
});
t.start();
```

This snippet will execute the foo method that blocks the thread for 5 seconds before finishing in a separate thread.

It doesn’t provide any callbacks or execution control mechanism, does not provide serialization and does not provide any ordering or pool mechanism if we do not run it through an ExecutionService that has been previously set up.

Instead, if we look at the next HydraRunnable example we can see that even though they look pretty similar in terms of amount of code, the features that it provides differ:

```java
HydraRunnable worker = new HydraRunnable("thread_name") {
    @Override
    public boolean action() {
        try {
            Thread.sleep(5000);  
        } catch (Exception e) {
            return false;
        }
        return true;
    }
};
Hydra.INSTANCE.getHydraExecutor().dispatch(worker);
```

It executes the same code but in a very different way. The action method is similar to the Java Runnable’s run method but it returns a boolean to control whether the action has been correctly executed or not. Depending on that, the thread will execute the success or the
error callback. These callbacks can be easily overriden when creating a new HydraRunnable, for example:

```java
@Override
public void finishCallback() {
    Hydra.INSTANCE.log("I HAVE BEEN CORRECTLY EXECUTED :)");
}
@Override
public void errorCallback() {
    Hydra.INSTANCE.log("I HAVE BEEN NOT :( ");
}
```

*Snippet 15 Implementable thread completion callbacks in Hydra*

And we can also run and override a pre-execution callback before long tasks so we can see when we are starting them:

```java
@Override
public void beforeCallback() {
    Hydra.INSTANCE.log("THIS WILL TAKE SOME TIME");
}
```

*Snippet 16 Before execution callback in Hydra*

What is more, HydraRunnable is serialized so it is less memory expensive and can be passed in Android data bundles. Finally, HydraRunnables can be run in a newly created thread such as in the first common Runnable example because the class implement the Runnable interface, but it is far more interesting to use the library’s own implementation of the ExecutorService to control the number of active threads and run them in a queue-style execution. The access and usage of this Executor is as simple as shown in the last line of the HydraRunnable example.

Although this is quite useful, the most powerful features of Hydra is the ability to create UI callbacks, which by default are not allowed on Android because as explained in earlier sections the UI is not accessible from any thread except the main UI thread. The most accepted and common way to communicate a worker thread that has finished a task with the UI (for example, in case we were downloading an image from the internet and we want to put it on an ImageView) is to use an Android Handler. The Handler class can be used to register to a thread and provides a simple channel to send data to this thread.

A Handler object registers itself with the thread in which it is created. For example, if you create a new instance of the Handler class in the onCreate() method of your activity, the resulting Handler object can be used to post data to the main thread. Let’s see an example
of a simple snippet that retrieves a String from the internet with the method foo and later it is sent to a Handler to set it on a TextView UI object:

```java
Handler handler = new Handler() {
    @Override
    public void dispatchMessage(Message msg) {
        String text = msg.getData().getString("text");
        if (text != null) {
            myTextView.setText(text);
        }
    }
};

Thread t = new Thread(new Runnable() {
    public void run() {
        String text = foo();
        Message msg = new Message();
        msg.getData().putString("text", text);
        handler.dispatchMessage(msg);
    }
});
t.start();
```

So, we have to first create the Handler on the UI thread and define the logic by overriding its dispatchMessage method. These method receives a Message, which is a class that wraps a data bundle and provides some other less important features. We get the String out of the bundle and then set it on the TextView. In our Runnable we have to create a Message object and dispatch it to the Handler who has the corresponding logic. At the end of the process, we have created the Handler, written the logic, created messages and dispatched them with the data bundle filled once data is acquired. And of course, take care of the scope and visibility of the objects.

The advantage of using a HydraUIHandler is that all the dispatch process is handled invisible to the developer. We only have to define the success and error callbacks we want and create an instance of HydraUIRunnable passing the HydraUIHandler as shown below:
HydraUIHandler hydraHandler = new HydraUIHandler() {
    @Override
    public void onSuccess(Bundle bundle) {
        myTextView.setText(bundle.getString("text"));
    }
};

HydraUIRunnable worker = new HydraUIRunnable("thread_name", hydraHandler) {
    @Override
    public Bundle action() {
        String text = foo();
        Bundle bundle = new Bundle();
        bundle.putString("text", text);
        return bundle;
    }
};

Snippet 18 New thread with UI callback with Hydra

This way, when we create the HydraUIHandler and supply it to the worker thread the library will take care of transform the callbacks and parse them to the Message mechanism used by the UI Handlers. We have to keep in mind that the HydraUIRunnable action method returns a Bundle and not a boolean. In case we do not want any callback we can just return null or use one of the overloaded creation signatures that doesn’t require a UIHandler.

At this point someone who knows about Android and the framework could think that the AsyncTask class is a good alternative to the UI Handlers methodologies because it was designed for it and already provides all we need: pre and post execution methods that run in the UI thread, a background method to run the task in a separated thread and a signature that allows to pass and return the parameters we need.

But there’s a big problem with the AsyncTask: in order to be executed it requires that the Activity where it has been defined must be active and running. So we cannot assure that a background process which is being run on an AsyncTask will finish because it may be stopped if the user closes the app or the Activity is paused, especially because the Garbage Collector behavior of the Virtual Machine is kind of unexpected when under pressure.
4.4 Hal

4.4.1 Overview

Hal is the first version of a library that provides support to execute heuristics to any Java or Android project. It is based on a very simple execution model in which action calls are run based on a list of heuristics defined by the developer. Every heuristic has a specific score. If the total addition of every heuristic’s score is greater than the action threshold, the action will be executed.

In this first version there are two different kinds of heuristics: regular calculable heuristics and learnable heuristics. The learnable heuristics require a learning function as well as a method to determine whether it has been already learned or not. The class hierarchy and the multiple inheritance approach used in Hal makes the system easy to scale and versatile. Hal is suitable to be used for complex tasks, optimization, AB testing and complex runtime conditions, but it is not for easily decidable actions or as a substitute for if-else statements.

4.4.2 Usage

In order to understand how Hal works, we first need to see the class hierarchy of the system. The next class relationship diagram shows the interfaces (marked with an “I”) and the abstract classes that implement them. Support classes and other are not shown.
There are two main interfaces: the decidable and the heuristic interface.

The decidable interface lists the methods that every action has to implement to run the associated heuristics and execute its main method depending on the result. Actions and heuristics are associated, an action can have many heuristics and a heuristic can be present in many actions.

The heuristic interface simply lists the methods that every heuristic has to implement in order to be called from an action without distinction of its implementation.

As we explained, there are two kinds of heuristics: regular calculable heuristics and learnable heuristics. Calculable heuristics are not the top element of the hierarchy because this way the system is open to have more kinds of heuristics, even though the learnables are a subclass of the calculable ones.

Learnable heuristics are a bit more complex because a learn function should be supplied by the developer. These functions usually need to write data on disk and keep track of previous execution. The developer should make these reads and writes in the learn function and define another method that returns whether the learning process has finished or not and therefore the heuristic is valid.
Like every other Slang library, Hal follows the singleton pattern implemented with an enum. The instance can be accessed to add new actions with their correspondent heuristics, and there is a dispatch method used to execute every pending action. A quick example:

```java
Hal.INSTANCE.addAction(a1);
Hal.INSTANCE.addAction(a2);
Hal.INSTANCE.dispatchActions();
```

When the actions are dispatched, they are copied onto a different action queue to avoid synchronization problems and the original queue is cleared.

But how are these actions defined? When creating new actions, the developer has to define three methods. First, pass the list of heuristics. Secondly, define the minimum threshold to execute the action. Finally, specify the action to be carried out. In the following example we can see a very simple action

```java
Action a1 = new Action() {
    @Override
    public List<Heuristic> getHeuristics() {
        return heuristics;
    }

    @Override
    public double getThreshold() {
        return 1;
    }

    @Override
    public void action() {
        Toast.makeText(getApplicationContext(), "HELLO USER", LENGTH_LONG).show();
    }
};
```

In the example, the list of heuristics is not shown but we defined 1 and the threshold, meaning that if the addition of the heuristic’s score is greater than 1, the action will be executed. In this case, the action is a simple toast displayed on the user screen, but in the real implementation they will be more significant.

Here is an example of a simple CalculableHeuristic that we would add to the action:
CalculableHeuristic h1 = new CalculableHeuristic() {
    @Override
    public Range getRange() {
        return new Range(0, 1);
    }

    @Override
    public void calculate() {
        value = new Random().nextBoolean() ? 1 : 0;
    }

    @Override
    public Category getCategory() {
        return Category.USAGE;
    }
};

So, we define the range (in this case from 0 to 1) and the calculate function, which is a non-
sense boolean that randomly assigns 1 or 0 to the value. The Category is unused at this
point, so it could return null, but for organization purposes it is a good practice to use one
of the listed categories in the Category enumeration.

If we supply the previous action with this heuristic, we will see the toast the 50% of the
times we execute it.

Finally, below there is an example of a learnable heuristic:
LearnableHeuristic l1 = new LearnableHeuristic() {
    @Override
    public void learn() {
        //learning function goes here
        PreferenceManager.saveNewLocation(LocationManager.getCurrentLocation());
    }

    @Override
    public boolean isLearned() {
        return PreferenceManager.checkConditionL1IsLearned();
    }

    @Override
    public Range getRange() {
        return new Range(0, 1);
    }

    @Override
    public Category getCategory() {
        return Category.LOCATION;
    }
};

As we can see, we define the isLearned method as a memory access that checks that a certain learning condition was already finished, and the learn method as a function that retrieves the user’s location and writes it on memory. This way, we could run the heuristic at 04:00 AM for three days in a row to determine his home’s location.

Finally, in the calculate method we should determine how to deal with retrieved data. In this example, we check whether the current location matches with our learned data about the user location and so we can determine whether he or she is at home or not, and run the action consequently.

At this point we can have an idea of how Hal works but far more important is how we can take advantage of it by implementing useful heuristics that could help us improve our application and provide a better user experience.
It is important to remember that heuristics are not an exact science. In fact, our approach is more based on the generalization of certain common behaviors, studied data from previous researches obtained from our analytics tools and previous experiences from other company’s products and finally own feelings rather than any kind of demonstrated data.

The data we identified to be interesting for our heuristics in Slang is listed below splitted by categories:

**Location**
- Location in work hours
- Location at night
- Location at weekends
- Country

**Usage**
- Battery level
- On/off screen
- Charging state

**Social**
- Social apps installed
- Accounts of the device

**Game**
- Number of victories
- Number of victories in a row
- Number of defeats
- Number of defeats in a row
- Number of games played
- Number of items used
- Number of items the user has
- Number of items bought
- Earned points
- Days without playing
- Days without answering a pending game
- The game has been updated

**Connections**
- Is the Wi-Fi enabled or disabled?
- Is the WAP enabled or disabled?
- Is the user connected via Wi-Fi or WAP?
- Network usage
Hardware
Android version
Phone model and manufacturer
Sim state

In the first version of our heuristics implementation these are the values we are interested in evaluate in order to carry out actions. Below are listed some examples of heuristics and their actions that are implemented in the first version of the game. It is important to remember that these heuristics will be tuned when we start collecting data.

Action: give 3 items of each kind to the user for free
Range: 0, 7
Threshold: 5
Heuristics:
+1 score for each defeat in a row (max 5)
+1 if the user has no items left
+1 if the user has less than 200 points
NO if the user has more than 1000 points
NO if we executed the action in the last 5 days

Action: tell the user he or she has pending games to answer
Range: 0, 10
Threshold: 7
Heuristics:
+3 if user is at home
+3 if battery level is more than 15%
+1 for every day without answering pending games starting from 2 days to max 6
NO if user has not network connectivity
NO if user has not pending answers
NO if we executed the action in the last 3 days

Action: send a reengaging push notification
Range: 0, 6
Threshold: 2
Heuristics:
+2 if the game has been updated
+2 if the user has been 10 or more days without playing
+1 if the user has been 5 or more days without playing
+1 if the user has played more than 10 games
NO if user has played in the last three days
NO if the user is at work
**Action:** congratulate the user after winning a disputed game and give extra points

**Range:** 0, 10

**Threshold:** 6

**Heuristics:**

+4 if the game lasted more than 10 rounds
+3 if the user earned more than 1000 points in this concrete game
+3 if the user has won 3 or more games in a row

These are an example of the actions and heuristics that will be implemented in the first version of the game among some others more. When we start collecting data and improving the application via updates we will add new heuristics related to the monetization and engagement of the application itself.

For example, when we add in-app purchases we will develop sets of rules that try to maximize the purchases while minimizing the times we ask the user to purchase. Same will happen with social integration. Actions and heuristics will be constantly updated at the same time we add new features and capabilities to the game.
Chapter 5. Backend

5.1 SimpleAPI Overview

To construct the backend, the involved coworkers used a reduced version of Symfony2 framework used in many projects of the company called SimpleAPI.

Since SimpleAPI is a version of the most used PHP framework in the world it has all the great advantages of using any complete framework:

- Code and file organization is extremely easy
- Countless numbers of tools and libraries that can help
- MVC (Model View Controller) Architecture
- Less code to write which speeds up the development
- Security – PHP has many input and output filtering functions which can add extra security layer to protect against certain attacks
- Suitable for teamwork

SimpleAPI is used to construct backends based on responding network requests and delivering plain content rather than create websites with UI and interaction. Actually, when we talked about it as a reduced version of Symfony is because it doesn’t include any of the elements that Symfony uses to create HTML templates such as the Twig engine.

So, we could say that SimpleAPI is a ready to deploy and fast to use framework that uses the core of Symfony to provide features such as routing mechanism, advanced security and Doctrine as ORM to connect and work in an object-oriented way with databases.

5.2 Backend Endpoints

I will not get into details about how the SimpleAPI or the backend is implemented because it is one of the tasks which I have been less involved. Instead, there are listed the endpoints used by the first version of the application and their parameters and functionalities.
Register an user [POST]: /user/register/?token=token

Inserts the new user on the database. The register is done via a SHA-1 encrypted IMEI and it is automatically executed so the user don’t need to input any data manually and still can be identified in the server.

Fetch active games [GET]: /match/fetch/active/?token=token

Returns the list of non-finished matches of the user.

Start a new game [POST]: /match/start/?token=token

Sends the JSONObject representing a newly started game as a parameter and adds it to the database.

Update a turn of an active game [POST]: /match/dispatchTurn/?token=token&id=gameId

Sends the JSONObject representing a new turn of the game with the id=gameId as a parameter and adds it to the database.

Retrieve the entire game history [GET]: /match/fetch/finished/?token=token

Returns the list of finished matches of the user.

Update the user stats: points, victories, defeats... [POST]: /user/update/?token

Sends the JSONObject representing the current user profile and stats to update the user entry in the database.

With these endpoints we can control the game flow and communicate with the other players with the less efforts possible.
5.3 The token

If we look at the endpoints and their definitions, we can see that all have a URL parameter that is called token. The token is the credential of a user: it is used to identify an user and contains important information such as the id in form of SHA-1 encrypted IMEI, the operator and country code, the profile checksum and other fields that will be added in next updates.

The token obfuscation - we cannot speak in terms of encryption because it is reversible - takes place in the client while the deobfuscation is made on the backend.

To do that, we first shuffle the token parameters to generate different and harder to identify tokens and then we XOR the byte array obtained from the shuffled parameters with a predefined hash that is also known by the backend. After that, we encode it in Base64 so it can be sent as an URL parameters.

The backend does exactly the opposite: first it decodes the Base64 representation and then XOR the decoded string with the same hash used in the client to obtain the token parameters.

```java
List<String> params = new ArrayList<String>();
String[] splitted = tokenString.split("&");
for (int i = 0; i < splitted.length; ++i)
    params.add("&" + splitted[i]);
Collections.shuffle(params);
String shuffledToken = new String();
for (String p : params) shuffledToken += p;
return "?token=" + URLEncoder.encode(encode(shuffledToken, Statics.HASH1), "UTF-8");

private static String encode(String target, String hash) {
    return Base64.encodeToString((hashing(target.getBytes(), hash.getBytes())), 0);
}

private static byte[] hashing(byte[] a, byte[] key) {
    byte[] out = new byte[a.length];
    for (int i = 0; i < a.length; i++) {
        out[i] = (byte) (a[i] ^ key[i % key.length]);
    }
    return out;
}
```

This is a quick overview of how the backend endpoints work.
Besides this, to construct the backend we had to create the routing yml file that redirects and/or delegates the endpoint accesses to the respective controller that does the corresponding tasks, write in those controllers the required code to connect with the database to store and retrieve data and do the necessary operations and checks and finally format the obtained data to send the response expected by the client.
Chapter 6. Results and conclusions

6.1 Usability Test

Usability testing is the best way to understand how real users experience and use an application. When we had the first version of the application constructed, we compiled a stub app filled with sample data to make the test.

Unlike interviews or focus groups that attempt to get users to accurately self-report their own behavior or preferences, a well-designed user test measures actual performance on mission-critical tasks. If the user cannot figure out how to complete an answer or make a purchase, no amount of "but I really like the app!" comments are going to make up for it.

We didn’t identify a concrete target audience for two reasons: first because these kind of casual games tend to be widely targeted and secondly because the number of people we had availability for a usability test was kind of reduced.

In regular usability tests, participants will perform a set of 5 to 10 tasks. Tasks should represent the most common user goals and/or the most important conversion goals from the application perspective (for example making a purchase).

It is also crucial to establish very clear success criteria for each task in order to determine if it has been carried out correctly. It is also important to clarify where the participant should begin the task (for example, at the home screen of the application), and how task completion and starting points may affect the researcher’s ability to counterbalance task order.

6.1.1 Test setup

We started defining our set of tasks as well as the validation criteria. The list of tasks corresponds with the most important dynamics of the game:

1) Find how many points the user has
2) Change the profile picture
3) Find the game’s history
4) Find how many points you earned in the first game you won
5) Open an active game and identify who started it
6) Buy all the swap letter items you can with the points you have
7) Find the last word answered by the opponent in the most recent game you finished
8) Answer the first active game using an extra 30’ time item
9) Open an active game and surrender before the time ends
10) And we defined four levels of completeness:

   1. The user has not finished the task
   2. The user finished the task after a big UI flow mistake and/or with big difficulties
   3. The user finished the task after a small UI flow mistake and/or small difficulties
   4. The user finished the task direct and fast with no mistakes

6.1.2 Test process

The test was carried out by following a simple process. The user was in a quiet room with the device (a Nexus 5) locked. Before starting the test, I gave some basic instructions: I told the tester that the app is a simple multiplayer game based on the word’s ending mechanic, and that I was going to ask him or her to complete some tasks on the application. I explained that after completing the task he or she should come back to the main screen and lock the device again until I ask for a new task. I also asked him or her to “think aloud” and act naturally.

The testers who were chosen among friends, familiars and fellow coworkers who were not involved in the project had the following profiles:

- Tester 1 HHRR Manager, female, 39 years old
- Tester 2 Audiovisual Communication student, female, 23 years old
- Tester 3 Android Developer, male, 24 years old
- Tester 4 Police Inspector, male, 48 years old
- Tester 5 Jeweler, female, 21 years old
- Tester 6 Taxi driver, male, 26 years old
- Tester 7 Personal Image student, female, 18 years old

In the following tables, we can observe the score each tester obtained in each one of the tasks. In every table there are attached comments made by the testers.
6.1.3 Test Results

Task 1 Find how many points the user has

<table>
<thead>
<tr>
<th>Tester</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>Answered with the number of victories but quickly realized the mistake</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 12 Task 1 results

Task 2 Change the profile picture

<table>
<thead>
<tr>
<th>Tester</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>Went to a match because saw an avatar</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>First scrolled down the bar searching for a button</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 13 Task 2 results

Task 3 Find the game’s history

<table>
<thead>
<tr>
<th>Tester</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 14 task 3 results
Task 4 Find how many points you earned in the first game you won

<table>
<thead>
<tr>
<th>Tester</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Took time to remember the first game was on the navigation bar history</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

*Table 15 Task 4 results*

Task 5 Open an active game and identify who started it

<table>
<thead>
<tr>
<th>Tester</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

*Table 16 Task 5 results*

Task 6 Buy all the swap letter items you can with the points you have

<table>
<thead>
<tr>
<th>Tester</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Insisted on clicking on the item instead of the button</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Went straight to a game instead of the navigation bar</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Took some time to understand the button with the cost of the items</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

*Table 17 Task 6 results*
Task 7 Find the last word answered by the opponent in the most recent game you finished

<table>
<thead>
<tr>
<th>POINTS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 18 Task 7 results

Task 8 Answer the first active game using an extra 30' time item

<table>
<thead>
<tr>
<th>POINTS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 19 Task 8 results

Task 9 Open an active game and surrender before the time ends

<table>
<thead>
<tr>
<th>POINTS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 20 Task 9 results
The tasks that took more troubles to complete where the buy items task and the surrender before time expires task. They also were the ones that raised more comments during and after the test. Some of these comments about those and other tasks are listed below:

- "You should be able to buy items from the match and answer screen."
- "A cross doesn’t mean surrender but exit instead!"
- "The green button with the Answer text in the main screen leads directly to answer?"
- "I’d like to filter between the games I lost and won."
- "The history is not hidden but I didn’t expect it here"
- "You should add a white flag icon to surrender instead of the cross or back button to leave the game."

All these comments will be taken into consideration when we introduce changes in next updates.

In general terms, the test was very successful because even the application was not in their native language and considering the wide range of profiles of the testers most tasks were completed successfully, which means that the game and the UI is easy to understand and use. Furthermore, testers who had problems completing a certain task hard no problem completing others which means that a punctual problem doesn’t lead to a general misunderstanding of the application.
6.2 Performance test

One of the most interesting new features of the latest versions of the ADB (Android Debug Bridge) are the memory and CPU tools that allow the developer to watch the performance of an application in real time.

When we had the first version of Slang filled with stub data, we proceed to run both tests.

6.2.1 CPU Usage

We carried the monitoring of the CPU in a Nexus 5 after clearing the RAM memory and running only the Slang app. We tried to use the app as heavy as we could: opening and closing screens all the time, fast scrolling through the feeds and touching random elements as if we were trying to crash or block it. We obtained the following results:

In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system.

In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Most of the code running on the device will execute in user mode.
As we can see, the CPU usage is pretty low especially when the every screen of the game has been opened and closed more than one time. The user usage is always higher than the kernel because our application doesn’t require hardware operations such as connecting to the camera, GPS or the accelerometer. So, in an application that is I/O bound, the time will be spent in system calls like read and write.

On the other hand, which is our case, the application does a lot of UI stuff, and all of the graphics processing is done in user CPU. However there is still a lot of I/O bound operations waiting for the frame buffer and input devices such as the touch screen, that’s why we have Kernel CPU too in a lesser amount.

6.2.2 Memory Usage

We carried out the memory usage test exactly the same way we did the CPU test, using a Nexus 5, forcing the application to do hard work during a minute and collecting the memory stats.

![Figure 10 Free and allocated memory consumption of Slang app](image)

The first thing we can observe is that our memory consumption reaches a limit above 16 MB. This is great because the heap limit is device-dependent. On a Droid or Nexus One, which are old devices, that limit is 24 MB. It means that our application can be fully stored in memory without consuming the heap of most of the devices in the market, even considering some of the elder ones.

When opening all the screens and scrolling multiple times though large feeds containing more than 50 games, the app only needed to increment the heap two times. This is also
very important because getting out of heap space and allocated memory leads to the infamous OutOfMemoryException, which was a real pain for developers in older Android versions.

Fortunately, our application seems to have a good memory management thanks to the factors explained in the Presentation layer development section such as the adaptation of all the graphic resources to the most fitting size and the use of the brand-new, highly efficient RecyclerView.

6.3 What was left behind

In this section we are listing everything that has not been finally included or that we changed while we were advancing in the project. I personally feel that it is the first and one of the most important reflections we should make at the end of the project (or at least the academic part of it).

When making a project of these kind in a company which their most important values are to lean, to be fast and to be daring, one realizes that it’s not about the idea or the rules or even the graphics – it’s all about the execution of this idea. So, when I started working on Slang I had a lot of worries that now seem trivial to me.

I was worried about making a cool website for the app. I was worried about having pages and profiles in the social networks. I was worried about which languages should the app support and be translated to. But as I realized later, all those worries were due to a lack of focus on what really mattered: before taking care of all these steps the product should be designed, built and evaluated.

At the same time I started worked on it, the situation changed a bit. Billy is a company which has a profound startup spirit even it is not a startup at all: we are always leaning when the market requires it, taking advantage of new business opportunities and opening new lines and products if we think we can be competitive.

And while I just started working on the project, a new opportunity appeared and suddenly I was leading a small team of four, teaching new assistants and moving to a more management role, spending more time in meetings and Skypes rather than in my previous, cozy developer hut. It was a great opportunity to work in a more diverse pool of tasks and to grow both personally and professionally but at the same time it slowed the whole project.

I had to spend more time in the office and dedicate more time to the project at home and due to my new position my vision about the project changed a bit. I focused on how to make parts of the project reusable on other products of the company and how to take advantage
of all the time involved on it so the whole project success or failure doesn’t rely only in the game. So when considering it I decided to construct the support libraries to help the developer and acquisition teams with some pending tasks.

Also, when speaking with the product team we decided to take more time to release a more complete version of the game and let some parts of the project to do after finishing the academic part. The most affected parts were the third-party integrations in the app (social SDKs such as Google or Facebook), the analytics integration and the parts related to the Play Store release (Play Services integration, website construction).

At the middle of the development process, we also observed that we weren’t able to setup the backend in the company server’s infrastructure until the end of April because we were migrating all our machines and databases to new systems to handle the exponential growth of Billy’s AdServer. The system administrators were full-time busy by setting up the new Elastic Search environment, creating the slaves and the database replications and many more tasks so we couldn’t ask them to upload and finish our backend even though it was already working in our localhost.

Finally, we didn’t include yet one of the main features of the application which is that words are clickable and by clicking it you can see their definitions, usages, morphology, etc. We will include this feature before the release but all the external services who provided these dictionary entries are paid services and we didn’t want to invest on it prior to the release.

However, we did some things that benefit the company and were not planned but are closely related to the project. They are explained in Company usage section.

6.4 Roadmap

As explained, Slang won’t be released until June 2015. Slang is a very young project and unlike the games we have released these last months it is big and complex enough to go step by step, so we won’t release a premature version. Below is the roadmap with the most important actions we have planned for the next four months. Based on the results and the feedback of these months, we will decide how to continue the project and how many efforts we will dedicate to it.

Keep in mind that in this roadmap are only listed the main actions of the planning and it do not include many other minor tasks such as the code maintenance, bug fixing, marketing creation and others.
As it states, we plan to have about 30,000 users and the end of August. Based on our 25% target retention (which is below the average 35% retention of Play Store apps), that would make about 7,500 active players. Most of them will be acquired by our advertising systems as fallback ads, so we expect the quality of the users will be above average and this will affect the retention.

We will also keep on working on the code and the technical side, improving and extending the game with new minor features and translating the application to other languages. At the end of this first part of the roadmap, we expect to have the game running (not only translated) in Spanish and maybe a third language.
6.5 Company usage and stats

The impact of this project has been very positive from the company's point of view.

Billy started a new business line based on casual mobile games. It was thanks to assessing the business possibilities of Slang in an internal first stage of the project that made us realize that we could redirect all the residual traffic and users that have already seen other ads to the games, and by this way improve our organic ranks in the Play Store and grow more for free.

In April 2015 we broke the barrier of 10,000 players. None of them are Slang users because as we explained Slang won’t be publicly released until June 2015 but meanwhile these last months we designed and developed some simple games. Our most successful game to the date is Billy Bird, a Flappy Bird clone that has four game modes and funny graphics. Our analytics tracking shows that Billy the Bird has lift off more than 40,000 times.

Our other newly-released games are Space Opera, a classic scroll spatial shooter and Blocks, an arkanoid-style game where the user has to destroy drawings made with blocks.
All these games use Skynet to make the install attribution and other HTTP requests. Hydra is also implemented to make the load of the game resources asynchronous and to post delayed UI actions.

Hal is not used yet because it has not been extensively tested and because these games do not have multiplayer features.

Furthermore, Skynet, Hydra and Hal were added to the version 2.0 of Freedom. Freedom is the core of Ricky Mobile Content, one of the three main business lines in Billy.

Freedom is a huge SDK used all around the world with more than 450,000 active users and 2.2M total installs that provides developers a way to monetize their applications using alternative payment methods to Play Store and an advertising system that features several kinds of ads like push notifications, banner display or icon drops among others.

Skynet is the HTTP Client of Freedom and has made more than 500 million request.

Hydra is used in several parts of Freedom such as retrieving the advertiser images, load the content adapters and run polling operations in some advertising systems. It is also used in combination with Hal to run all the heuristics in separated threads.

Hal is one of the most important additions to Freedom. At this moment is used only in the push notification advertising system and configured with a set of heuristics focused on choosing the best moment to send the notification to the user based on several factors such as the location, the battery level, hours of maximum usage, if it is connected to WiFi, network activity and some other minor factors like the accounts registered on the device or the number of apps installed.

To explain the impact of Hal in the push notification advertising system of Freedom, we collected the data of the notification delivers and the applications installed from these
deliveries in the first week of April. Since Hal is a new addition, we sent approximately 25% of the traffic to the old system in order to compare the performance.

<table>
<thead>
<tr>
<th>App</th>
<th>Country</th>
<th>Deliveries</th>
<th>Install Rate</th>
<th>Deliveries</th>
<th>Install Rate</th>
<th>Deliveries</th>
<th>Install Rate</th>
<th>Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lazy Swipe</td>
<td>South Africa</td>
<td>211</td>
<td>1.90%</td>
<td>204</td>
<td>1.47%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>AliExpress</td>
<td>United States</td>
<td>471</td>
<td>1.27%</td>
<td>403</td>
<td>0.25%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>Clash of Kings</td>
<td>United States</td>
<td>1861</td>
<td>0.32%</td>
<td>695</td>
<td>0.29%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>Lazy Swipe</td>
<td>Turkey</td>
<td>259</td>
<td>3.09%</td>
<td>247</td>
<td>0.81%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>Apus Launcher</td>
<td>Portugal</td>
<td>7724</td>
<td>1.84%</td>
<td>2278</td>
<td>1.58%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>Powerclean</td>
<td>Portugal</td>
<td>1283</td>
<td>0.39%</td>
<td>775</td>
<td>0.26%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>Apus Launcher</td>
<td>India</td>
<td>74930</td>
<td>0.56%</td>
<td>23352</td>
<td>0.57%</td>
<td></td>
<td></td>
<td>Old</td>
</tr>
<tr>
<td>Hungama</td>
<td>India</td>
<td>6313</td>
<td>0.52%</td>
<td>2326</td>
<td>0.39%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>Lazy Swipe</td>
<td>India</td>
<td>44780</td>
<td>1.20%</td>
<td>12305</td>
<td>1.00%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>Newshunt</td>
<td>India</td>
<td>6320</td>
<td>0.51%</td>
<td>2322</td>
<td>0.52%</td>
<td></td>
<td></td>
<td>Old</td>
</tr>
<tr>
<td>Paytm</td>
<td>India</td>
<td>75108</td>
<td>0.51%</td>
<td>25252</td>
<td>0.59%</td>
<td></td>
<td></td>
<td>Old</td>
</tr>
<tr>
<td>Apus Launcher</td>
<td>Spain</td>
<td>28321</td>
<td>1.99%</td>
<td>9470</td>
<td>2.12%</td>
<td></td>
<td></td>
<td>Old</td>
</tr>
<tr>
<td>Lazy Swipe</td>
<td>Spain</td>
<td>12732</td>
<td>2.01%</td>
<td>12555</td>
<td>1.80%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>Apus Launcher</td>
<td>Brazil</td>
<td>199542</td>
<td>0.77%</td>
<td>62595</td>
<td>0.87%</td>
<td></td>
<td></td>
<td>Old</td>
</tr>
<tr>
<td>Hola</td>
<td>Brazil</td>
<td>9420</td>
<td>1.80%</td>
<td>9342</td>
<td>1.37%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
<tr>
<td>Powerclean</td>
<td>Brazil</td>
<td>36518</td>
<td>0.26%</td>
<td>21923</td>
<td>0.33%</td>
<td></td>
<td></td>
<td>Old</td>
</tr>
<tr>
<td>Napster</td>
<td>Brazil</td>
<td>7854</td>
<td>0.73%</td>
<td>7749</td>
<td>0.49%</td>
<td></td>
<td></td>
<td>Hal</td>
</tr>
</tbody>
</table>

| Mean            | 1.16%       | 0.87%      |

Table 21 Deliveries and conversions of Freedom notification system from 30th March to 5th April

As we can notice, the first setup of Hal improves the conversion rate a considerable 0.29%. Taking a first look at the notifications volume we are handling, it is very important. If we made the numbers, we can see that we had an average payout of 0.28 EUR per install.

We sent 707440 push notifications during the week and made 5943 app installs. Scaling it to a monthly revenue, we are talking about 6656 EUR.

An increment of the conversion rate directly affects the revenue, so if we can double the conversion rate, we will double the revenue. That's why Hal is so important: small conversion rate improvements can make big differences when you move huge volumes.
6.6 Final conclusions

In this project we have been through the whole process of designing and building a game and we realized that there are many tasks involved on it. Creating a new game is not only about graphics or the rules, especially on the mobile environment: It’s also about user experience, performance, iterating the product and align the goals of each one of the stages. At the beginning, I thought I’d spend most of the time in creating interactions, UI and working with the data and game entities, but at the end of the project I realized that I had spent more or less the same time on every stage.

When I started working on Slang, I had a preconceived idea of how game and application development worked which was kind of naive. I thought that as a game developer, I should focus on making the game fun to play and that’s all. If the game is fun, nothing else matters. But I was wrong. One of the most important things I’ve realized while working on Slang is that every single aspect is important, and if you create an awesome game with cool graphics and a great gameplay but you cannot monetize, get exposure or draw attention, if people don’t play your game, at the end it is a failure.

At some point it lead me to reduce risks from the game just in case it didn’t work or we couldn’t finish it as expected. I started by abstracting functionalities and creating libraries that could be used in other projects of the company, developing and implementing them on these other projects and business units.

It reduced our focus on the game but created value for the company, which is great.

We also decided to take more time on the development stage and delay the release of Slang until we had a completely functional first version of the game. This allowed me to improve the development: create a more robust UI, achieve better performance, and make a better layer separation so it is easier to work with it among others.

These are the final conclusions of this project, both from my personal and the company’s point of view. Create a new state of the art application requires time and efforts in all the stages, not only in the technological side of the product. When starting new projects in a company that is generous enough to let you use their resources, try to minimize the risks and don’t gamble everything on a one single card. Be prepared to adapt to new situations and to lean as fast as possible.
Finally, keep in mind that the end user is the judge who will determine whether the product is successful or not, so try to deliver the best experience possible, from the product side and the performance, keeping an eye on all the requirements of nowadays complex, fragmented but beautiful and full of possibilities mobile-driven world we live in.
Chapter 7. Bibliography and references


