NTNU

Grumpy Old Git Club
Team Number: 16

TDT4240 Software Architecture

Bomb Hunt

Architecture

`define(COTS, ["Android", "Artemis-odb", "Tiled", "Google Play Services"])`

*Cabral Cruz, Samuel (*496704*)*
*Claessens, Bart (*486346*)*
*Ihlen, Erling Hærnes (*765137*)*
*Trollebø, Jarle (*766901*)*

Primary Quality Attribute:
Modifiability
Secondary Quality Attribute(s):
Usability, Performance and Interoperability

Presented to
Alf Inge Wang

Department of Computer and Information Science
Norwegian University of Science and Technology
(NTNU), Trondheim
Spring 2018

Contents

List of Figures

List of Tables

Introduction

Phase Description

The second phase of the software development is called *Architecture Phase*. In this phase, all the elements introduced by the *Requirements Phase* will be reused and glued together in order to create the architecture of the whole application. This phase is crucial because a lot of decisions have to be taken while juggling with a lot of input parameters and constraints. This forward thinking about what the final architecture will looks like in necessary to ensure the fulfillment of the requirements established during the previous phase. [Bass, 2013, Chapter 17]

Document Description

In this document, we will first remind what is the game concept and the goals of the project (Vision Statement). By after, the Architecturally Significant Requirements (ASRs) that are going to guide the game architecture will be identified (Architectural Drivers / Architectural Significant Requirements). Then, we will provide a description of the project's Stakeholders and Concerns together with a justification relatively to the Selection of Architectural Views views that we made to satisfy their needs. The following chapters will explain with more details the differents Architectural Tactics and Architectural and Design Patterns that will be applied during the development of the application. Finally, the different Views that will visually summarize the information will be presented. A discussion of the Consistency Among Architectural Views will disclose any incoherence that may have been induced by the views.

As usual, a list of the Issues and a log of the Changes brought to the current document will be available at the end of the document.

Techniques and notations used in this document follow recommendations of Bass [2013] and Larman [2005]. The 4+1 framework by Kruchten [1995] has also been considered during the redaction of this document. By applying this framework, we make sure that a complete and thorough documentation of the architecture is done.

Vision Statement

Project Goals

The goal of this project is to develop a multiplayer game called *Bomb Hunt*. In this game, the users will incarnate a gladiator trying to make its way into a labyrinth to find and defeat its opponents to accumulate points. In addition to the multi player mode, the player will also have the opportunity to play the game on a single player mode where the goal will be to survive as long as possible by resisting to different waves of ennemies spawning. A tutorial mode will also be developed to make the introduction of the game and its different components much easier to the inexperienced users. A more in-depth description of the game concept is available in the section Game Concept.

Project Scope

The *Bomb Hunt* will be developed in the context of the group project for the course TDT4240 Software Architecture. The project will be lead by a team of 4 developers that will work part-time for the next 3 months. The best case scenario would have been to find an artist to optimize the aesthetic part of the application. Moreover, the application should minimally work on any Android platform that has the necessary utilities to download and execute Android Package (APK) files. Even if this project is mainly dedicated to educational purposes and bounded to a learning context, the team members would like to officially publish the game and maintain it after the end of the semester.

Game Concept

As mentioned before, the concept of the game we are planning to develop is inspired from the vintage video game series called *Bomberman*, first developed in 1983 by Hudson Soft. In those game, the user usually incarnate a robot that try to find its way out of the maze by destroying walls and find the key that will open the door to the next level. Figure 1 shows this game in action. Multiplayer versions of this game have been developed since 1990, but for a long time remained a side feature of the original singleplayer game. Wikipedia contributors [2018a]

Even if this game has been commercialized under new official console versions for more than 20 years, we still think that this concept can be pushed forward by the addition of new characters, items, maps, abilities, etc. and by creating a mobile version out of it. Despite

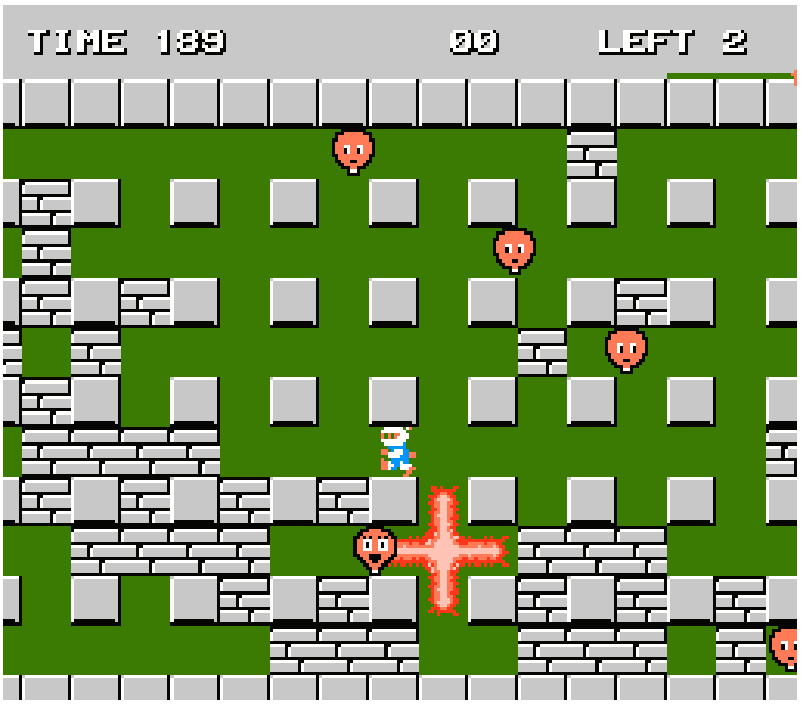

Figure 1: Gameplay of the original Bomberman on NES (1983)

this long success story, *Bomberman* series did not see any new version released from 2010 to 2016. Its latest version at the moment is called *Super Bomberman R* and is focusing on the mutliplayer gameplay on a really similar way that we are about to do.

In our versions of the game, all the basic elements of the game will be conserved.

- 2D maze where the character can only move up, down, right or left
- Mix of destructible and indestructible walls
- Items dropping off the destroyed walls

The new elements added will mostly be related to the special abilities of the each character. Some examples of those abilities are defusing bombs, building walls, summoning Non-Player Characters (NPCs) and kicking bombs. Great care will be needed to balance the cool down of those special abilities to avoid some characters to be over-powered comparatively to the others. Moreover, we are planning to use bigger maps and smaller camera view that will allow us to use more detailed graphics and to add additional difficulty to the game since a given user will not necessarily know what is happening on the rest of the map.

We are expecting the gameplay to be fast and to be flexible enough to account for many strategies. Users will be rushed to find the opponents to avoid that they build too much in force. For this part, we are planing to build a skill upgrading system that will introduce some variations depending on the character. Those variations will create some characters that will be stronger in early fights and some others that will only reach their full capacities around

the end of the match. Also, the point system will be created in such a way that the only real way to make points will be to tear down the other users whereas breaking walls will only account for small amount of points. These dynamics will necessarily push the opponents to engage the fight faster.

Another interesting aspect of the game will be the accumulation of badges according to the way the users are playing the game. Based on the statistics, the different badges will be unlocked. Down here is a short list of achievements that is currently considered to be included in the game.

Collateral Damage
Kill 3 adversaries at the same time

Kamikaze
Eliminate everybody on the map (including you) and win the match

Excavator
Destroy 1000 walls

Veteran
Play 100 Matches

Furthermore, all the users of the application will be ranked according to their overall statistics. This overall score will have to take several parameters into consideration to favor active users and distinguish skilled users at the same time.

Due to the simplicity of the gameplay, some efforts will be necessary in order to attract modern users by using smooth graphics and intuitive User Interface (UI). Some mobile versions of this game have already been created. The Figure 2 show one of these that we found particularly nice in terms of the graphical interface. We expect ending up with something that looks somewhat similar to this.

Figure 2: Bomberman mobile in action by Fejer

Market Positioning

Business Opportunities

Nowadays, mobile applications are not seen as something really extraordinary anymore. Their easy accessibility by the end-users joint to the rapidity of their development and deployment have lead to the creation of countless number of them, especially games. It is harder and harder to find new game concept that will be completely different as something that already exists somewhere else. Nevertheless, some games are still able to stand out themselves in this ocean of pastime.

Most of these outstanding games have somewhat the following set of features:

- Easy to learn and use
- Nice graphics and animations
- Can be played for 1 minute or hours
- Have some real challenges involved
- Constantly requests the attention of the user (fast gameplay)

With our concept, we think that most of these elements have been reunited, but having those is not necessarily a direct way to success and the only way to know it will be when the game will have been published. On the other hand, creating mobile games is still considered as an open market comparatively to console games development where only few giant companies have sufficient money and staff to try their chance. Overall, the team does not expect to get money out of this project, but, since they are obliged to develop this game, they will make their best to have a nice final product that **may** lead to some returns through advertisements.

Product Position Declaration

Bomb Hunt is developed for a very diverse clientele of any age and culture wishing to have some good moments during their spare times independently of their location. The main aspect of the game that we think will make it different from the other games available or variants of the *Bomberman* concept is the real-time interaction between the different users. Instead of trying to create any kind of sophisticated Artificial Intelligence (AI), users will have to compete against each others. Another trilling aspect is the tracking of the user statistics that will allow them to unlock achievements badges and the ranking of the users. We will also develop the game in such a way that the addition of new features such as characters, items and maps will be easy to perform. This modifiability will allow us to keep the active users interested in the game and to have new things to exploit and discover along their play time. We must however keep in mind that the vast majority of our customers will probably be aged from 6 to 18 years old. Hence, the animations should not be to crude while at the same time remains funny. Finally, the game will provide a platform allowing the users to defeat their friends in a bombing skirmish, thus main purpose of this application is recreative.

Architectural Drivers / Architectural Significant Requirements

This section outlines the main drivers that most affect the system architecture.

Functional Requirements

Online Mulitplayer

The most important functional requirement is the ability to play mulitplayer online with other players. All our requirements follow after this core requirement. It is critical that the data and logic is structured in such a way that it is easy to synchronize data, state and events so we minimize the possibility for errors. In the cases were errors due to networking issues the system shall find out what went wrong and performe the required tasks to recover the game. The most sensible way to achieve this would be to mark all entities that should be synchronized with one or multiple components containing the network state of an entity, like what client created the entity and what simulation frame the entity is supposed to be at a given time.

Different Player Classes and Powerups

The player should be able to select a character from a set of characters all with differing abilities. The abilities can be altered during gameplay by acquiring powerups. It is therefore very important that components are made in such a way that it is easy to create modify abilities in the middle of the game. Different player classes and powerups The player should be able to select a character from a set of characters all with differing abilities and the possibility to be altered during gameplay by aquiering powerups, it is therefor very important that components are made in such a way that it is easy to create new abilities and modifiy what abilities a player has in the middle of the game. new abilities to a character during runtime and that in practice

Quality Requirements

Modifiability

Modifiability is our primary QR. It is very important that we structure our game in good. For instance it should be possible to implement new game modes and have them work without having to modify the core engine. Ideally it should be as simple as implementing a predefined interface that defines a set of rules for the game mode and is able to perform meaningful actions with as little knowledge of the other modules of the system. Modifiability is also important to let us rapidly prototype new features that might be interesting without too much overhead for it to be feasible.

Usability

Our game should be easy to use, we want a user to be able to just jump straight into the action with as little introduction as possible. The user should always be aware of what state the game is in and the controls and menus should be as minimally bloated as possible.

Performance

It is very important that the game is performant for the stability of the multiplayer. If one client isn't able to perform the required ticks per second to be able to simulate and keep the state synchronized between the clients then there is going to be a very big issue. It is therefore important that the game loops are used wisely and the game is able to detect whether or not it is able to keep up and automatically able to disable unimportant features like particle effects.

Interoperability

Interoperability is important to achieve good and reliable mulitplayer experience across android clients made by different hardware manufacturers and across different local area networks that might be configured differently. Normally p2p applications have to do a hole punchthrough using a broker server in order to establish a reliable p2p connection. By using Google Play Game Services we avoid having to deal with initiating connections between players ourselves, because the google services handles it for us.

Business Requirements

We want to create an overall engaging experience with interesting and fun gameplay elements. The game should have competetive aspects like highscores, leaderboards and achivements.

Stakeholders and Concerns

This section outlines the main stakeholders of the system and their concerns related to the software architecture.

Lecturer and Course Assistants

The lecturer is interested to see how the students perform, and as well what they deliver in the final version of the game. To make sure this can be accomplished as easy as possible for the lecturer, we will make sure to make our architecture as easy as possible to understand, and as well look through, by documenting everything as good as possible. This will make it easier to evaluate us for the final grade, and hopefully make it more enjoyable to read up on our architecture and test our game.

ATAM Evaluators

During this course, other groups in the same course will be assigned to review our documentation and the architecture itself. To make it easier for the other groups to review our work, we will make sure our views and tables are reflecting our game as good as possible. This way the architectures will be as clear as possible, and should be very easy to understand, even if you had no experience with our chosen patterns and code.

Development Team

We, the developers, are concerned in being able to deliver the architecture we visioned. As our selected architecture is quite ambiguous, we're required to be quite motivated to deliver what we want to make. As we are motivated to get a high as possible grade in this course, this shouldn't be an issue.

Game Players

The actual game players, or the end users, are concerned with how good the game is to play. They're not as concerned with how the game is developed or structured, but more interested with the features and functionality. This means our logical view has to be on point, and as well make sure it's well represented within the game itself.

Selection of Architectural Views

A list of the views (viewpoints) that you will use in the architectural documentation, their purpose, target audience (which stakeholder you are addressing), and what notation will be used for each view (use a table for this purpose).

View	Purpose	Targeted Audience	Notation
Logical	Describe functionality that the system will provide to end users.	▸ Developers ▸ Course Staff ▸ ATAM Evaluators ▸ End Users	▸ Domain Model ▸ Class Diagram
Process	Describe dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system.	▸ Developers ▸ Course Staff ▸ ATAM Evaluators	▸ FSM Diagram ▸ Sequence Diagram
Development	Describe the system components from a programmer's perspective	▸ Developers ▸ Course Staff ▸ ATAM Evaluators	▸ Component Diagram
Physical	Describe the topology of software components on the physical layer as well as the physical connections between these components	▸ Developers ▸ Course Staff ▸ ATAM Evaluators	▸ Deployment Model

Table 1: Architectural Views Selection

Architectural Tactics

This section outlines the architectural tactics applied in this project to meet the quality requirements.

Modifiability Tactics

The goal of modifiability tactics is to lower the cost of making a change to the system. In general we define this cost as the time spent on making a change.

The tactics here discussed have three main goals:

- Localize modifications / increase cohesion
- Prevent ripple effects / decrease coupling
- Defer binding time

Encapsulation Placing a module behind an interface decreases the coupling between the module and those that depend on it. The implementation of the interface can be modified without having to change the modules that depend on it.

Generalizing Making a more general version of a component allows to create different instances without having to modify code. For example a generalization of a method with parameters can be used in different situations to perform different tasks without needing different methods.

Preferences Coding certain aspects of the game as preferences allows the developers to easily tweak the game. Some preferences can be made available to the user so the game experience can be customized to personal preferences.

Anticipate expected changes We apply this tactic by asking ourselves the questions: "What changes are most likely?" and "For each change, does the proposed decomposition limit the set of modules that need to be modified to accomplish it?" Some changes are more likely than others. By anticipating what changes will be most likely to occur we can make sure that they require minimal effort.

Maintain semantic coherence Semantic coherence refers to the relationships among responsibilities in a module. The goal is to ensure that all of these responsibilities work together without excessive reliance on other modules.

Hide information By only making the necessary parts of a module public, we avoid unanticipated interdependencies. This allows to split the implementation in smaller parts without having to make a lot of changes when one of these parts has to be modified.

Use an intermediary When applicable we use an intermediary to convert the output of one module to the expected input of another module.

Usability Tactics

Usability tactics attempt to make the system easy to learn and use while offering all the required functionality.

Simplicity We adhere to the concept known as Occam's Razor which can be summarized as "less is more". By limiting the number of unnecessary elements and instead focusing on a simple and intuitive gameplay, we make the game easier to learn and more enjoyable for the user. Extra functionality is added in a way that adds minimal complexity.

Prototyping A UI that makes sense for us is not necessarily the most intuitive for all users. Thus we try to make early prototypes and get feedback from other users.

Using accepted standards There are widely known accepted standards. Designing our UI to conform to these standards, will make new users instantly familiar with our interface. Examples of this are Google's Material Design, standard controller layout, pause menu in top right, etc.

Architectural and Design Patterns

For our game we will combine several different architectural patterns, to make the game fulfill our requirements.

Architectural Patterns

Entity Component System

Entity–component–system Wikipedia contributors [2018b] is an architectural pattern that is mostly used in game development. An ECS follows the Composition over inheritance principle that allows greater flexibility in defining entities. Every game objects is represented by an entity. Each entity is composed of several components and the logic of these components is contained in systems. This means that the behaviour of an entity can change drastically by adding or removing components, without having to stick to hardcoded entities. This pattern partially satisfies the modifiabily requirement and will make it a lot easier for us as developers to create new types of entities for our game. Instead of having to make a new entity from scratch, we can develop a single new component, add that to an entity with a combination of other components, to create a unique and possibly exciting new entity.

Peer to Peer

Peer-to-Peer (P2P) Wikipedia contributors [2018e] computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application meaning they can all create and modify their own entities. When creating an entity all network components will be synchronized between the peers allowing all of them to locally simulate the entities, we'll keep track of which entities are locally created and which are remote allowing for re-synchronizing states every x ticks. It became obvious for us that a P2P connection would be more beneficial for us, as we didn't want to fully develop our own backend for handling traffic and updating the state of the game, for every game of every user. This will help us with scalability, if the game ever were to gain traction or a big playerbase in an alternative universe. We haven't decided fully how we will set up the P2P connection, but we have looked at some alternatives for our game. One of the alternatives was to create a nodeJS backend matchmaking server for handling Hole punching**?** and setting up the connections, but it looks like it might take too much time off our game. We then looked at alternatives, or existing solutions that might speed this process up a bit for us. One of the popular ways to set up p2p for phone games,

was to use Google Play Games Service. This is used to set up a P2P connection between players. Once the connection is made, the game clients synchronize the game state between each other.

Model View Controller

Model View Controller Wikipedia contributors [2018c] will be used in our application to divide up the model, and the rendering of the model, and the control of the model. This means that for instance internally the model will contain a lot of information, but nothing about how it will be displayed in the view. This makes it practical for developers if they want to completely swap out a view, or controller of a model, without having to change the model at all. This will be achieved by keeping the calls by the model itself to a minimum. The controller will use the functions of the models to change it's state and data, while the view use calls of the model to access information about what to display.

Design Patterns

Singleton

One of the design patterns we plan to use for the game, is the singleton pattern Wikipedia contributors [2018f]. This pattern will help us when we want to create a class that we want to guarantee at most one instance off is running at a time. It will also help us with classes where we keep a lot of game-wide configuration and information that might want to be accessed. Using it we won't have to pass references to objects around, creating a lot of redundant coding and software that is harder to write tests for.

Observer

This pattern is used when we want some to observe changes in our application using ObserversWikipedia contributors [2018d] which streams events that we can subscribe to. Whenever an observer notices a change happening it will notify all it's subscribers of this change. This can be used to keep track of changes to certain data that is important for game's and application's life cycle like the player's health and lives. When the player runs out of life we can transition the application to a new state like a game over screen. This will be the primary way we get notifications from the ECS, directly tapping into the ECS and looping through a set of entities and checking their values one by one, instead of just being notified when a value has been achieved
changed, would be quite heavy on the resource side of the game, and in general create very messy code.

Views

Logical View

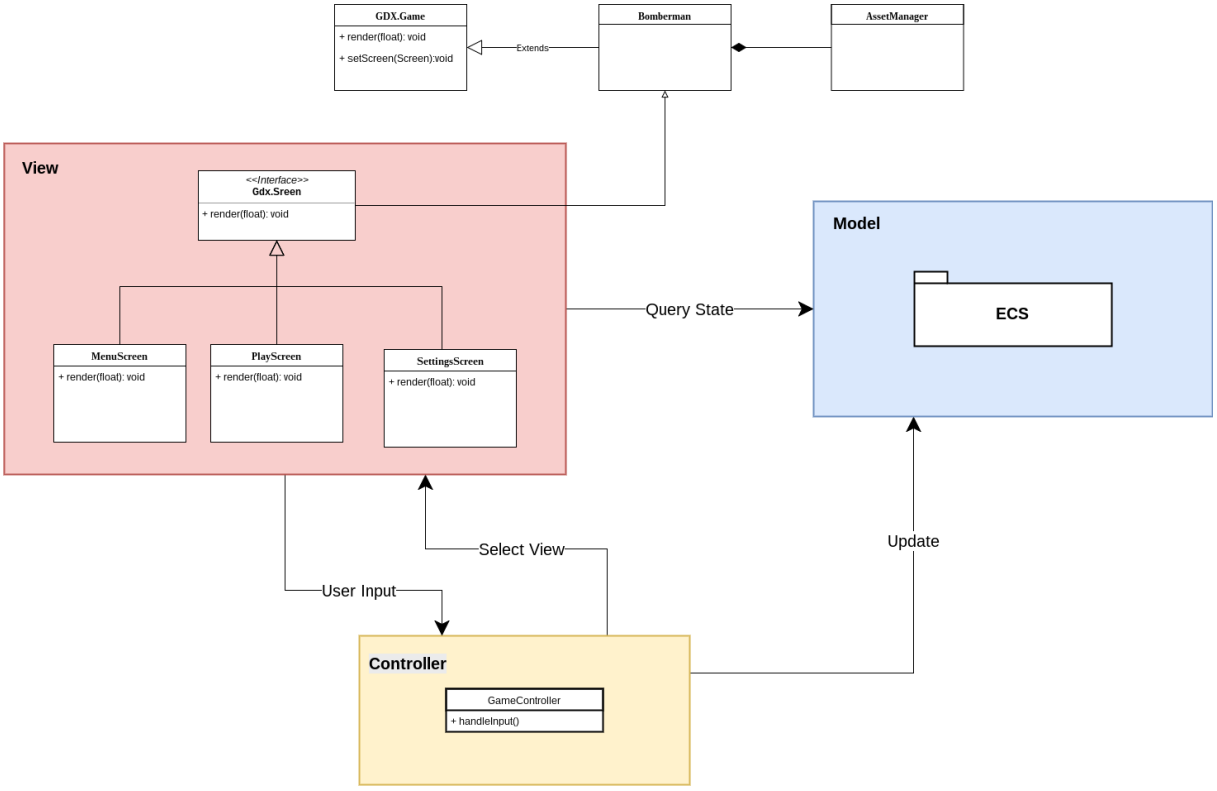

Figure 3: The logical view of the *Bomb Hunt* game

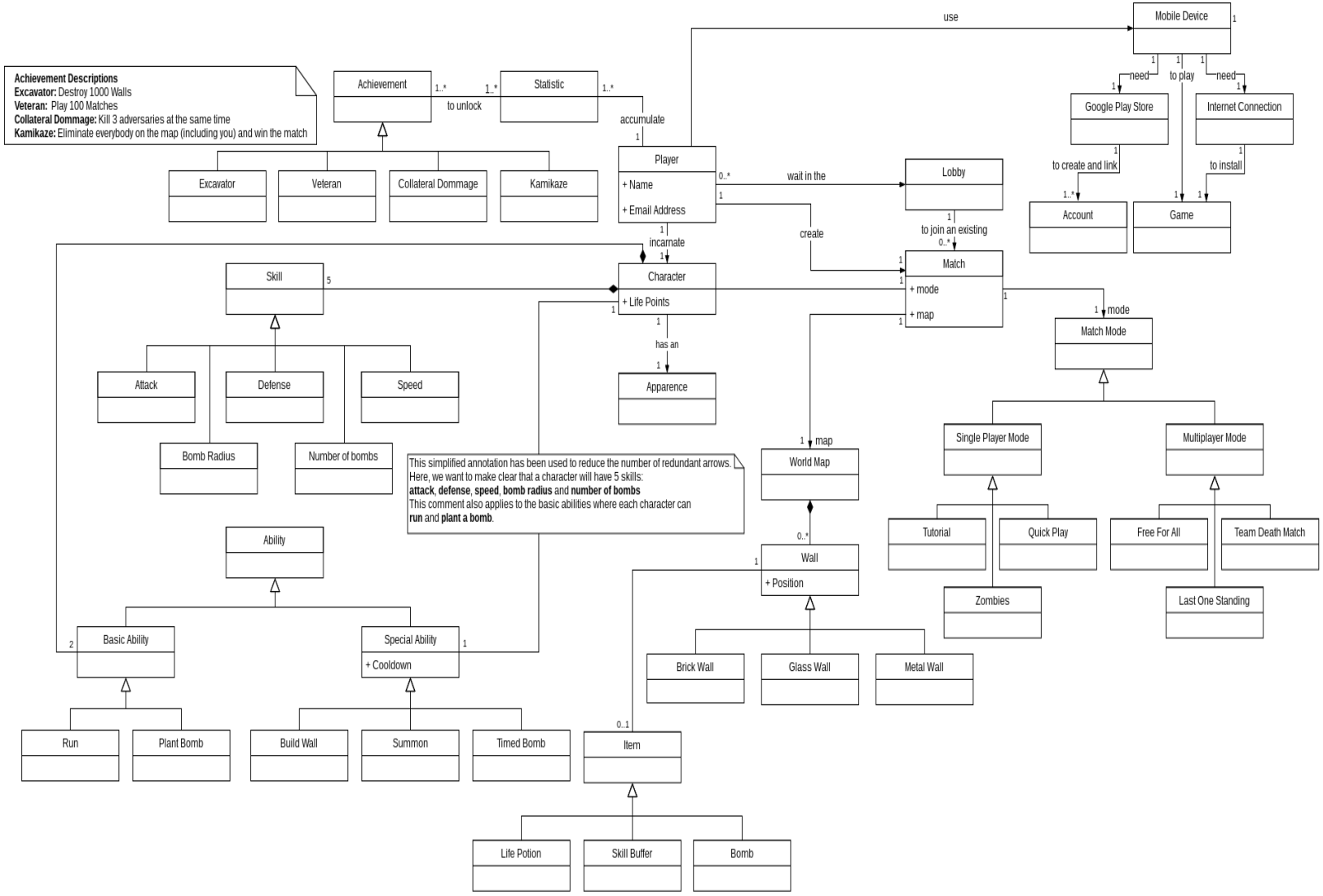

Figure 4: The domain model of the *Bomb Hunt* game

Process View

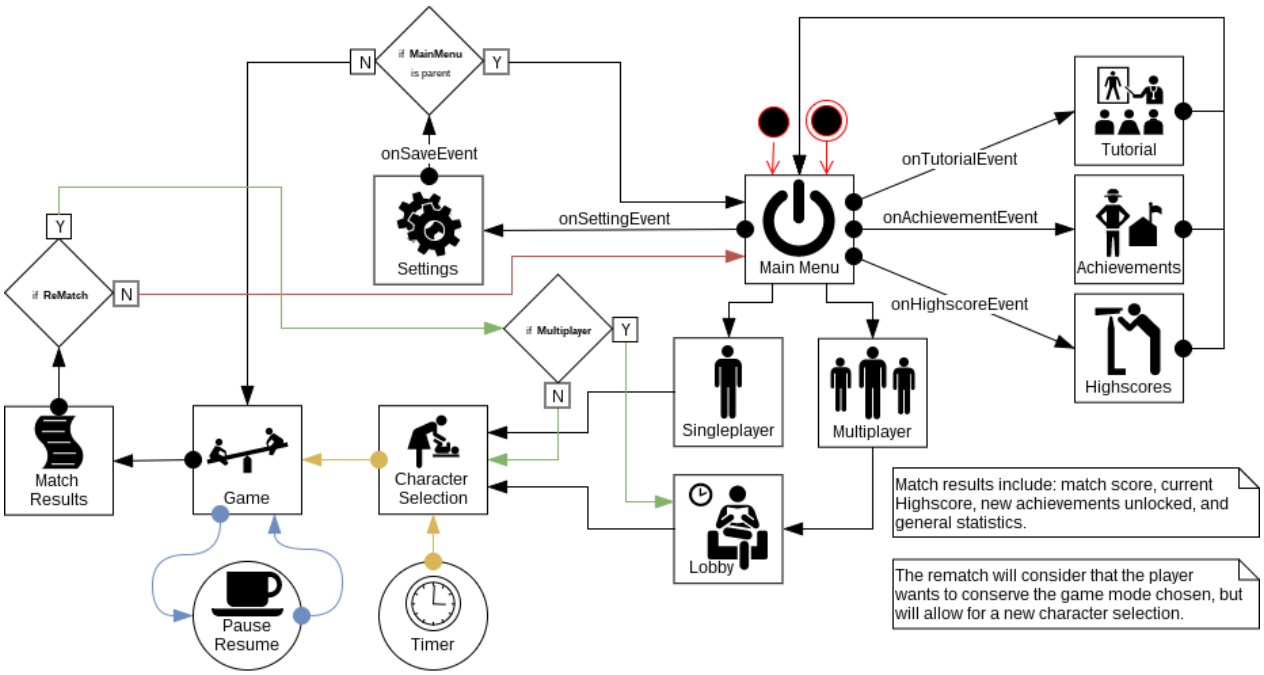

Figure 5: The process view of the *Bomb Hunt* game

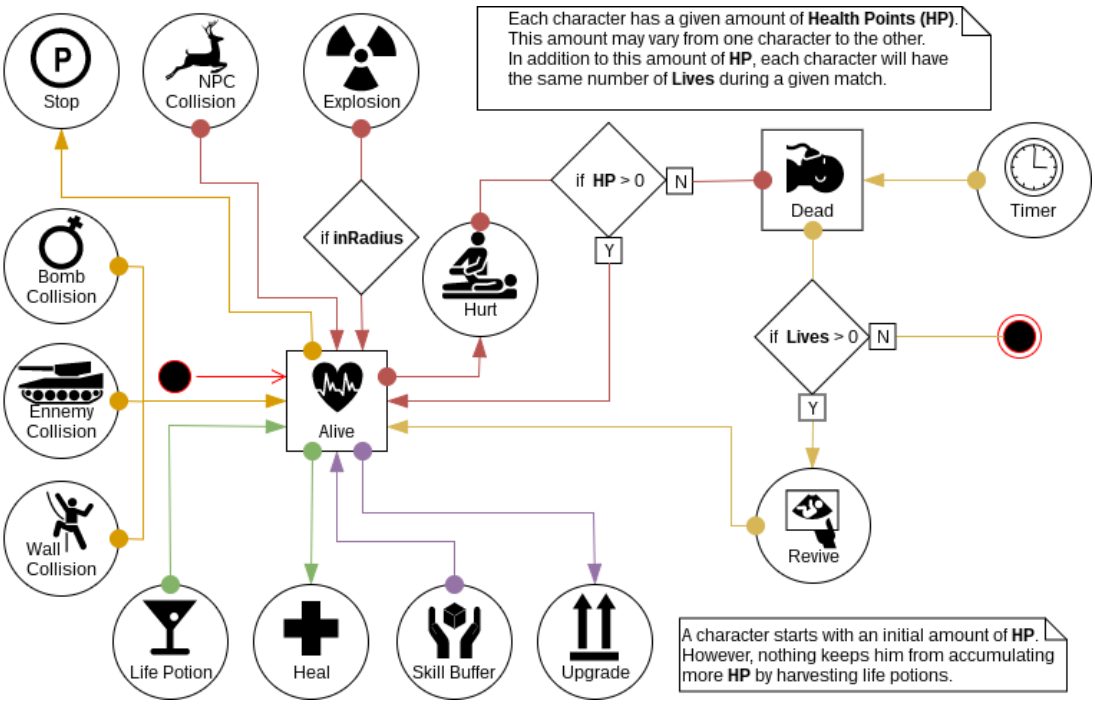

Figure 6: The FSM for character entities

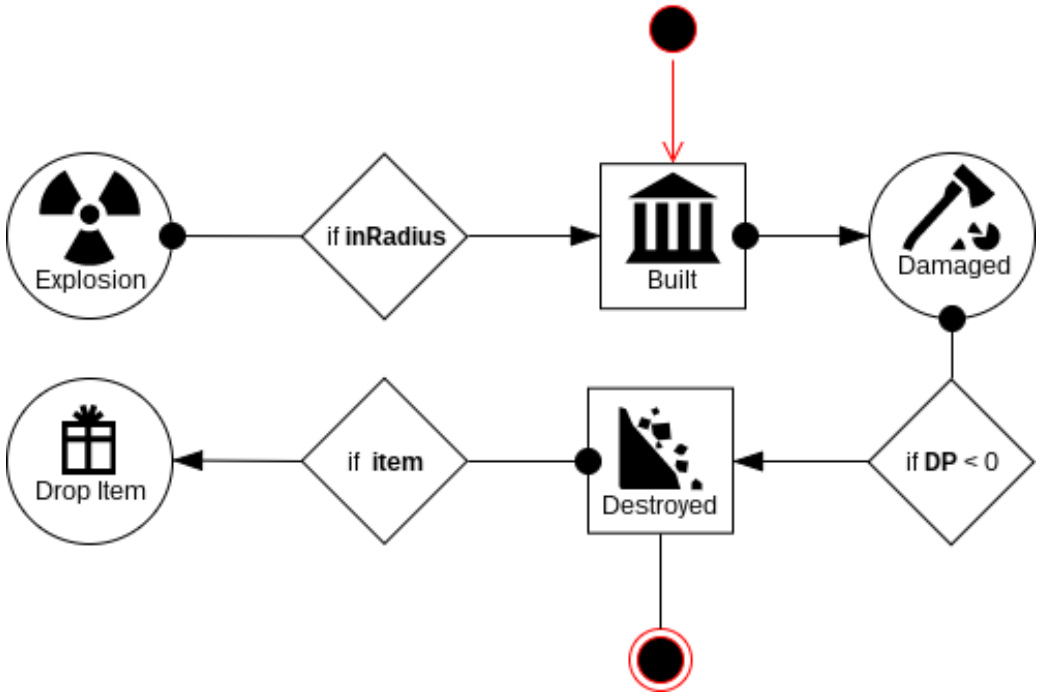

Figure 7: The FSM for wall entities

Development View

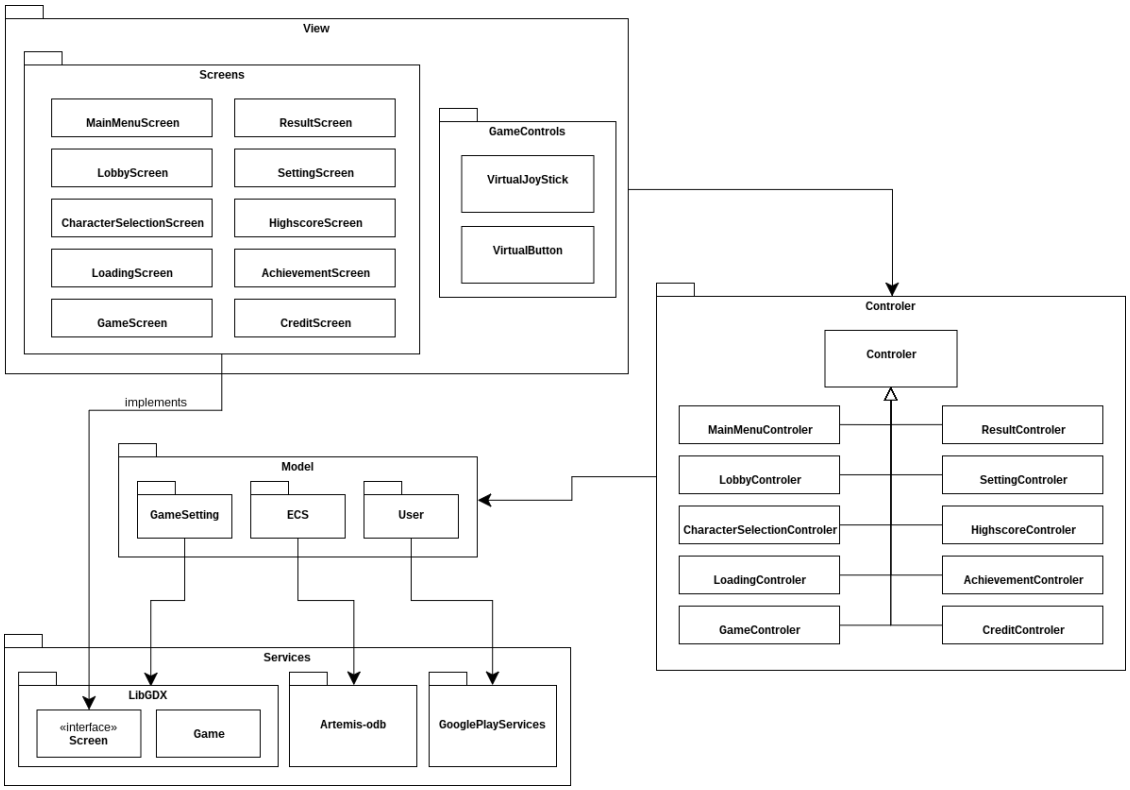

Figure 8: The development view of the *Bomb Hunt* game

Physical View

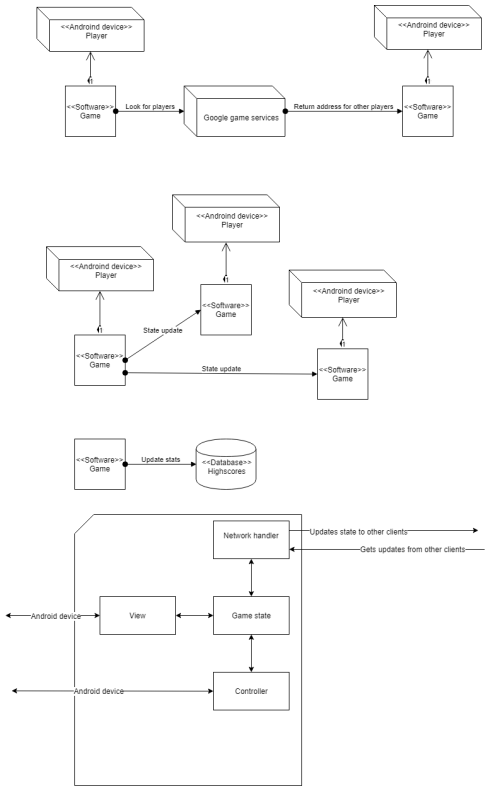

Figure 9: The physical view of the *Bomb Hunt* game

Consistency Among Architectural Views

Regarding the consistency among the architectural views, we can say that the views do not present mayor incoherences considering the small amount of time we add to produce them and the early stage of the project. Nevertheless, more attention has been accorded to the view that are less prone to change with the development of the project. In fact, we considered that trying to produce a detailed class diagram or even trying to predict what would be the different classes, interfaces and packages of the future application was not relevant and almost infeasible for the moment.

We still consider that the purposes of the logical view and the development view have been attained. For the logical view, we can see that the model of the domain contain much more details that the class diagram itself. Even if their structure is somewhat similar, the modeling of the domain is a better tool at the beginning of the project than a class diagram since it will provide a common language to both the end-users and the development team and ensure that everyone is on the same page comparatively to a class diagram that can often be challenging to understand and to present properly into a report. For the development view, its main purpose is to ease the separation of the workload among the developers. Although we do not have an exhaustive listing of all the code components to be implemented, we are still able to divide the work properly. With the advancement of the project, we will have a better idea of what should appears into those views and we will update them in consequence.

Another incoherence may be related to usage of different nomenclature of the architecture components. One the most egregious difference is probably the interchanging usage of the words items and powerups.

The last incoherence we can observe is between the process view showing the navigation through the screens and the development diagram that mention the presence of a credit screen and a loading screen whereas the process view does not show those two elements. Inversely, the process view seems to suggest the existence of an pause screen which is not mentioned in the development view.

Architectural Rationale

We already went into slight detail on how the patterns will benefit us previously in the document, but we will go more in depth here.

More members in this group has already had a lot of experience creating different software from before, and a lot of the time, we are forced to pass around a lot of objects, so they can be referenced deeper in the object tree. Say for instance some Logger that will keep track of what has been going on in the game. If that had to be passed around all the time to be called when something needs logging, it would end up quite messy.

Since it's a logger as well, it would only make sense to have one instance of it at all time during the game, which is when this pattern also comes in handy and guarantees us one single instance of it at all time.

To make the code more tidier and keep a more organized structure on how we will let the game know about states and events, we will use the observer pattern to accomplish this.

When we first started researching patterns, we were quite fond of the Entity-Component-System (ECS), as it allowed us to be more dynamic and flexible when creating entities for our game. It also makes it quite easy to fulfill the Modifiability attribute of the game, that specified it should be easy to change or configure in order to add or modify functionality. This is definitely the pattern we're the most exited about.

Since our secondary attribute was usability, and because Model-View-Controller (MVC) is such a widely used standard, we knew that if we manage to implement this well, we will fulfill this attribute with a simple interface for the user that can interact with all the underlying logic. The actual game logic won't be affected by the MVC. The usage of MVC will only affect how we manage inputs and outputs between the game world and the rest of the application.

If we manage to set up P2P, it will be a lot easier to handle all the data that is usually passed around in multiplayer games. Since all data will be passed between the players themselves, and since there will be quite limited how many players are on at a time, it will be a lot easier for us to develop it. Initially we think Google Play Service will be the solution that will help us.

Issues

The goal of this chapter is to document the different issues that we faced during the creation of the following document and the development of the project in general.

Team Formation

First of all, we had some problems to set the first official kick-off meeting because one of the member of the team was barely answering to the communications and did not even the invitations links to the *Slack* channel neither the *GitHub* repository. Before the handing of the first assignment, its name, **Carlos Cantos Morote**, has been removed from the documents and the team agreed to kick him out of the team.

First Delivery Deadline

We must admit that it was quite challenging to produce as much information in the time we had to do it. In addition to the production of the report, the team had to find a concept for the game. We managed to find an idea relatively quickly, but a lot of implementation details are still missing to correctly and fully create the requested artifacts.

Graphics

We have no graphical experience within the group, and as we look at other mobile games, it becomes apparent that a simple graphical look is something we wish to achieve during this project. This is to make the game more appealing to the users.

Peer-to-Peer Connection

Since we rely on peer-to-peer to transfer the data between devices, we might encounter problems with devices that are unstable. How we will we handle bad devices potentially crashing game sessions and ruining the experience for other players.

Changes

This section records the changes brought to the current document since its creation.

Date	Version	Description
2018/02/21	**1.0**	Initial draft
2018/04/04	**1.1**	Correct spelling and table mentions

Glossary

ASR Requirement that will have a profound effect on the architecture—that is, the architecture might well be dramatically different in the absence of such a requirement.
[3, p.291].

Cool-Down The minimum length of time that the player needs to wait after using an ability or item before it can be used again.
[1, cool-down].

NPC Taken from the world of pen-and-ink role-playing games, an NPC is a character encountered in an RKJ who is not controlled by the user.
[2, p.38].

Bibliography

Dictionary definitions. URL http://www.yourdictionary.com/.

Lexicon A to Z: NPC (Nonplayer Character). *The Next Generation Magazine*, (15), Mar 1996.

Len Bass. *Software architecture in practice*. Addison-Wesley, Upper Saddle River, NJ, 2013. ISBN 978-0-321-81573-6.

Kenneth Fejer. Pixel art. URL http://www.kennethfejer.com/pixelart.html.

Philippe Kruchten. Architectural blueprints — the "4+1" view model of software architecture. 12:42–50, 11 1995.

Craig Larman. *Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development*. Prentice Hall PTR, Upper Saddle River, N.J, 2005. ISBN 0131489062.

Wikipedia contributors. Bomberman (1983 video game) — wikipedia, the free encyclopedia, 2018a. URL https://en.wikipedia.org/w/index.php?title=Bomberman_(1983_video_game). [Online; accessed 25-February-2018].

Wikipedia contributors. Entity–component–system — wikipedia, the free encyclopedia, 2018b. URL https://en.wikipedia.org/w/index.php?title=Entity%E2%80%93component%E2%80%93system&oldid=821499559. [Online; accessed 21-February-2018].

Wikipedia contributors. Model–view–controller — wikipedia, the free encyclopedia, 2018c. URL https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. [Online; accessed 25-February-2018].

Wikipedia contributors. Observer pattern — wikipedia, the free encyclopedia, 2018d. URL https://en.wikipedia.org/wiki/Observer_pattern. [Online; accessed 25-February-2018].

Wikipedia contributors. Peer-to-peer — wikipedia, the free encyclopedia, 2018e. URL https://en.wikipedia.org/wiki/Peer-to-peer. [Online; accessed 25-February-2018].

Wikipedia contributors. Singleton pattern — wikipedia, the free encyclopedia, 2018f. URL https://en.wikipedia.org/wiki/Singleton_pattern. [Online; accessed 25-February-2018].