NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4240 Software Architecture

Group 2 - Implementation
Penguins in Space
Android

Christiansen, Sander Aker
Dahl, Lars-Kristian
Klaussen, Stein-Aage Nibe
Lundenes, Morten
Sjøen, Henry Skorpe
Zhou, Shelley Xianyu

Primary Quality Attribute: Modifiability
Secondary Quality Attribute: Usability, Performance and Interoperability

April 23, 2017

Contents

1 Introduction

1.1 Description of the project and this phase

This project report is written as a culmination of the course Software Architecture, in the spring semester of 2017. The project scope includes to design and implement a software architecture for a multiplayer mobile game for Android or iOS. We have selected Android for our game, but our game can easily be adopted to support iOS with some effort.

This document describes game implementation and testing phase, which is the third part of the project documentation - the first part is Requirement document and the second part is Architecture description. During this phase, the game is programed based on the selected quality attributes and architecture. Testing is performed to verify if quality and functional requirements are fulfilled.

1.2 Description of the game concept

The game is an Asteroids like game, with multiplayer functionality. Figure 1 shows a screenshot of the retro arcade game Asteroids. A lot of the game dynamics are borrowed from this game.

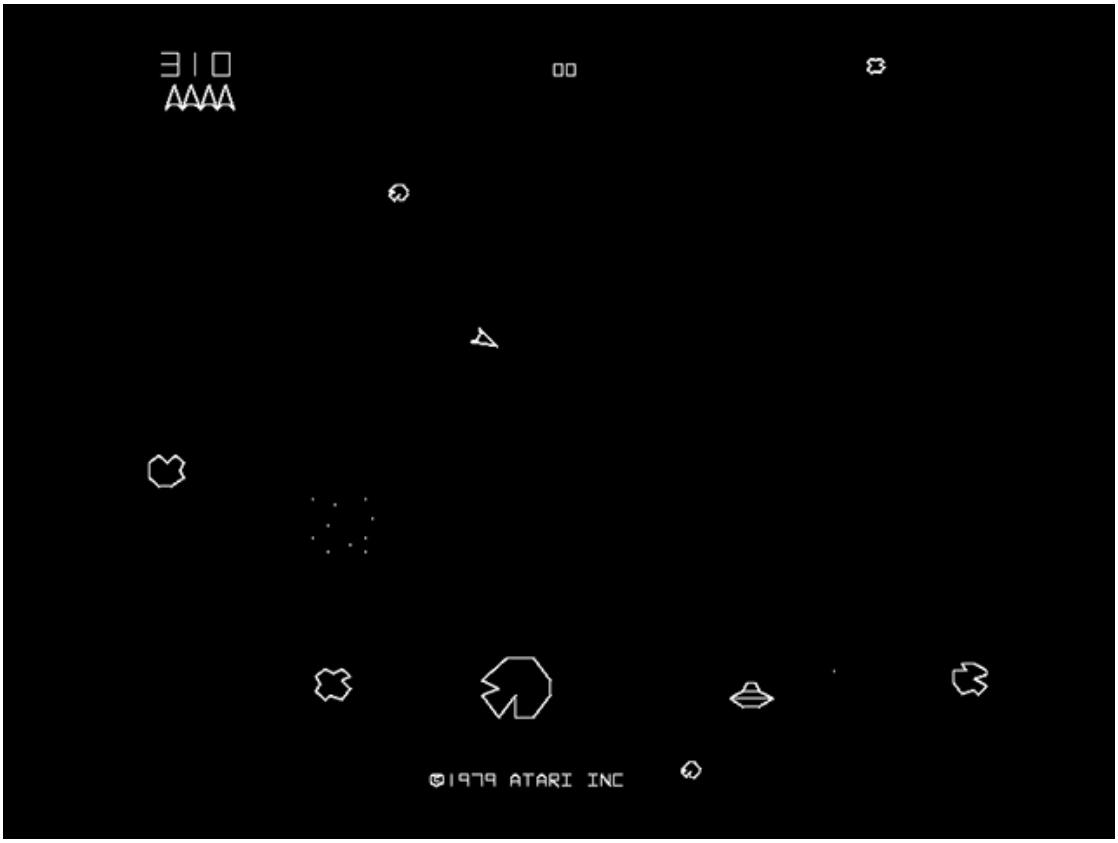


Figure 1: Asteroids, an arcade space shooter released in November 1979 by Atari, Inc.

The name of the game is "Penguins in Space".

You are a penguin floating in space. There are angry snowballs all around spawning randomly. The goal of the game is to stay alive for as long as you can, and get the highest score possible by either shooting snowballs in single-player mode or other penguins in multiplayer mode.

In singleplayer mode, power-ups will spawn throughout the game and give you abilities like dropping bombs or becoming invulnerable. The goal here is to survive and destroy as many snowballs as possible. Destroying a snowball increases ones score by one. The levels increase with your score, increasing the speed of the angry snowballs which makes it harder and harder. The goal is to achieve the highest score. In multiplayer mode there are several other players spawning that can attack each other. They will fight in a free-for-all until the last penguin is left standing.

The player controls the space-penguin using an analog stick in the lower left corner and shoots by pressing the button in the lower right corner. Below is Figure 2, a screenshot

of the game in single-player mode. More detailed game information can be found in Chapter 3 (User Manual).

Figure 2: Illustration of our game - Penguin in Space

1.3 Structure of the document

This document contains seven chapters, with one appendix for references. Chapter 1 describes the project phase and game concept, followed by description of design and implementation details in Chapter 2. In Chapter 3, the detailed user manual is described. The test report for all quality attributes and functional requirements are listed in Chapter 4. The relationship between implementation and architecture is described in Chapter 5. Finally the experienced problems, issues or any notes of significance are listed in Chapter 6 and last Chapter 7 records change history of the documents.

2 Design and implementation details

The game uses LibGDX at the core, which provides a framework and utilities for making a multi-platform game. The focus has been on the Android platform, but the core concepts of the game will also work on other platforms, such as the single player. For multiplayer, a platform specific implementation of the INetworkService interface would have to be implemented for multiplayer to function.

2.1 Model-View-Presenter (MVP)

The application uses the MVP pattern, specifically the passive view variant, to manage the views and navigation between them. The presenters are subclasses of the LibGDX Stage class, while the views are subclasses of the Scene class. Each presenter defines an inner interface named IView that its views must implement. The presenters also contains an inner class named ViewHandler that handles callbacks from the view. Because each presenter only has a single view and LibGDX works by instantiating and managing Screen objects, the game uses a "Presenter First" approach where the presenter is instantiated first. The Presenter then creates a view and injects a new instance of its Presenter.ViewHandler in the constructor. The presenter stores a reference to the view through a Presenter.IView handle, and the view stores a reference to the injected Presenter.ViewHandler. Because two presenters, MultiplayerGame and SingleplayerGame, share a single view, GameView, there is a separate public interface IGamePresenter that these presenters must implement. The GameView also implements a public interface IGameView so that both presenters may interact with it. An overview of the MVP structure can be seen in figure 3.

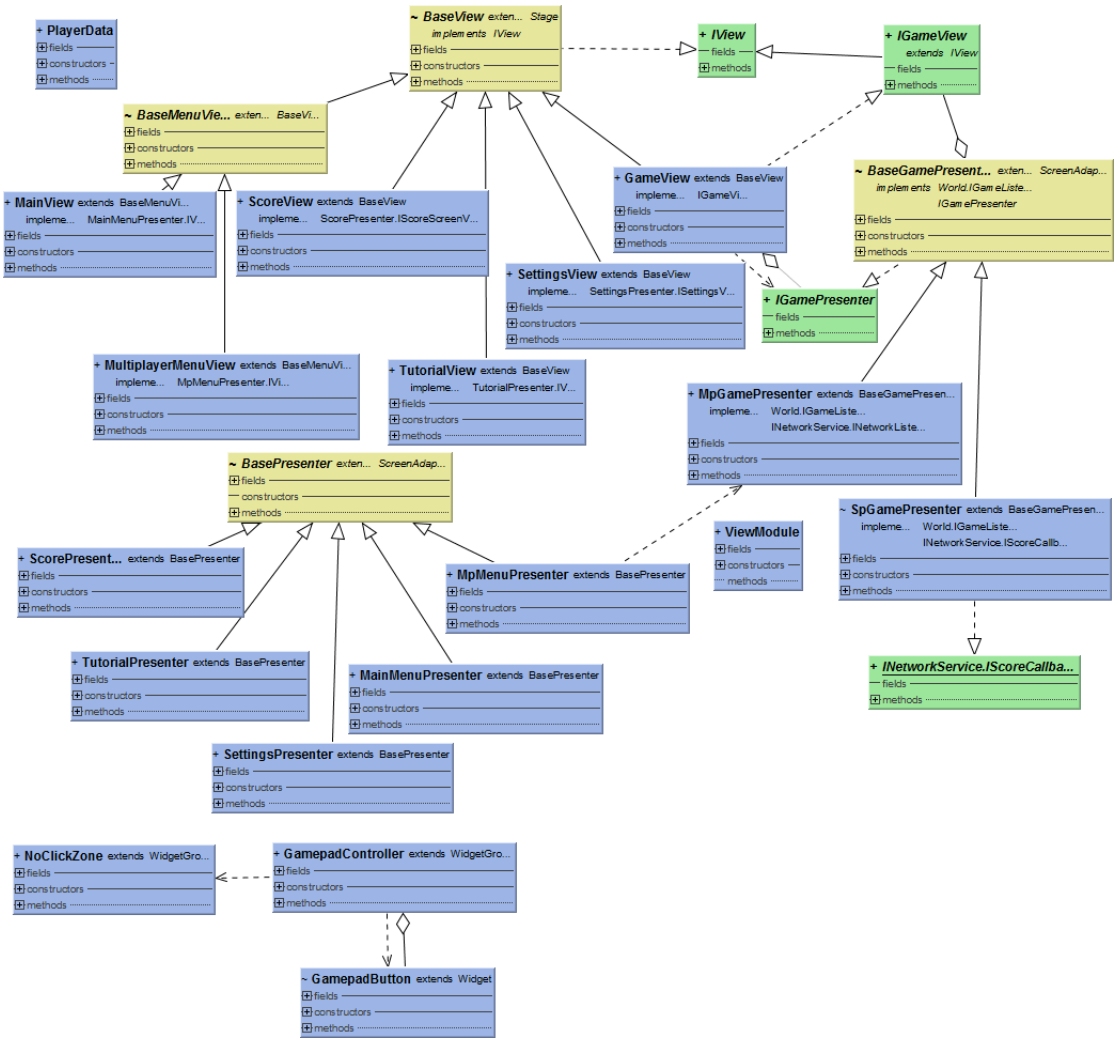


Figure 3: MVP architecture

2.1.1 Presenters

There are some abstract base classes that can be extended to simplify the creation of new presenters. BasePresenter provides basic lifecycle management for it's subclasses, as well as resizing and rendering. The BaseGamePresenter additionally provides callback implementations and event handling for events that are shared among game controllers.

2.1.2 Views

All views must implement the view.IView interface which specifies callbacks for lifecycle events, rendering, resizing and input management. Views must additionally implement a presenter specific interface for the presenters they interact with. The abstract class

view.BaseView provides default implementations for updating, drawing and resizing the view. view.BaseMenuView additionally provides implementations for displaying and managing a table of menu items.

2.1.3 Widgets

The touch controller has been implemented as a custom reusable UI widget. It consists of a Button, TouchPad and NoClickZone that have been wrapped in a GamepadController widget group. It handles all input and delegates all it's actions to a ControllerInputHandler instance. The ControllerInputHandler is decoupled from it's input source, making it easy to change the control scheme without affecting any other parts of the application.

2.2 Services

The application uses several different services at various stages throughout it's lifecycle. Some services are application wide, while others are related to single screens. A dependency injection library has been utilized to help manage these dependencies.

Figure 4 provides a simplified overview of all the services.

Figure 4: Service layer

2.2.1 Application services

2.2.1.1 ServiceLocator

ServiceLocator provides managed access to the application and entity services. The services are managed by Dagger2 Components, AppComponent and EntityComponent. The services are provided by Dagger2 Modules, AppModule and EntityModule.

2.2.1.2 ISettingsService

This interface defines methods to store and retrieve application settings such as volume and player appearance. The implementation of this interface is platform specific and the Android implementation AndroidSettingsService utilizes the SharedPreferences API to manage these settings.

2.2.1.3 AssetService

The AssetService class manages all assets for the application, such as textures, sounds and music.

2.2.1.4 IGameConfig

The IGameConfig interface defines an interface to retrieve game specific settings. The two implementations, MultiplayerConfig and SingleplayerConfig specify configuration for multiplayer and singleplayer respectively.

2.2.1.5 AudioService

The audio service provides a service to manage and play music and sounds.

2.2.1.6 INetworkService

The network service defines an interface for managing network connections. The implementation is platform specific, and the Android implementation AndroidNetworkService utilizes Google Play Games Services for it's implementation.

2.2.2 Game services and utilities

2.2.2.1 AnimationFactory

The animation factory creates and caches animations.

2.2.2.2 ComponentMappers

The component mappers allow constant-time retrieval of components for entities.

2.2.2.3 IDrawableComponentFactory and DefaultDrawableComponentFactory

This implementation creates and caches drawable components.

2.2.2.4 EffectFactory

Allows the client to register effects and request new instances of them.

2.2.2.5 EffectTextureFactory

Manages creation of effect textures.

2.2.2.6 EntityFactory

Handles creation of entities.

2.2.3 Dependency Injection (DI)

The application uses a dependency injection library, Dagger2, to achieve inversion of control of it's services and components. This lets the components declare which services they depend upon and have them injected automatically at run-time. The abstract class ServiceLocator manages all services and provides static access to them for classes that haven't been implemented with automatic injection.

2.3 Entity Component System (ECS)

The ECS contains all the game logic in it's systems and components. The World class encapsulates the internal systems of the ECS and provides an observable interface for various game events that the presenters listen to. There are also some systems not related to game-logic that perform various external tasks. One thing to note about the ECS module is that it uses very few getters and setters, and instead exposes public fields directly. This is intentional to reduce the amount of function calls in the performance critical path of the system. There are too many systems and components to list them all, but here is a quick overview of the ECS system in general.

A simplified overview of the ECS package can be seen in Figure 5

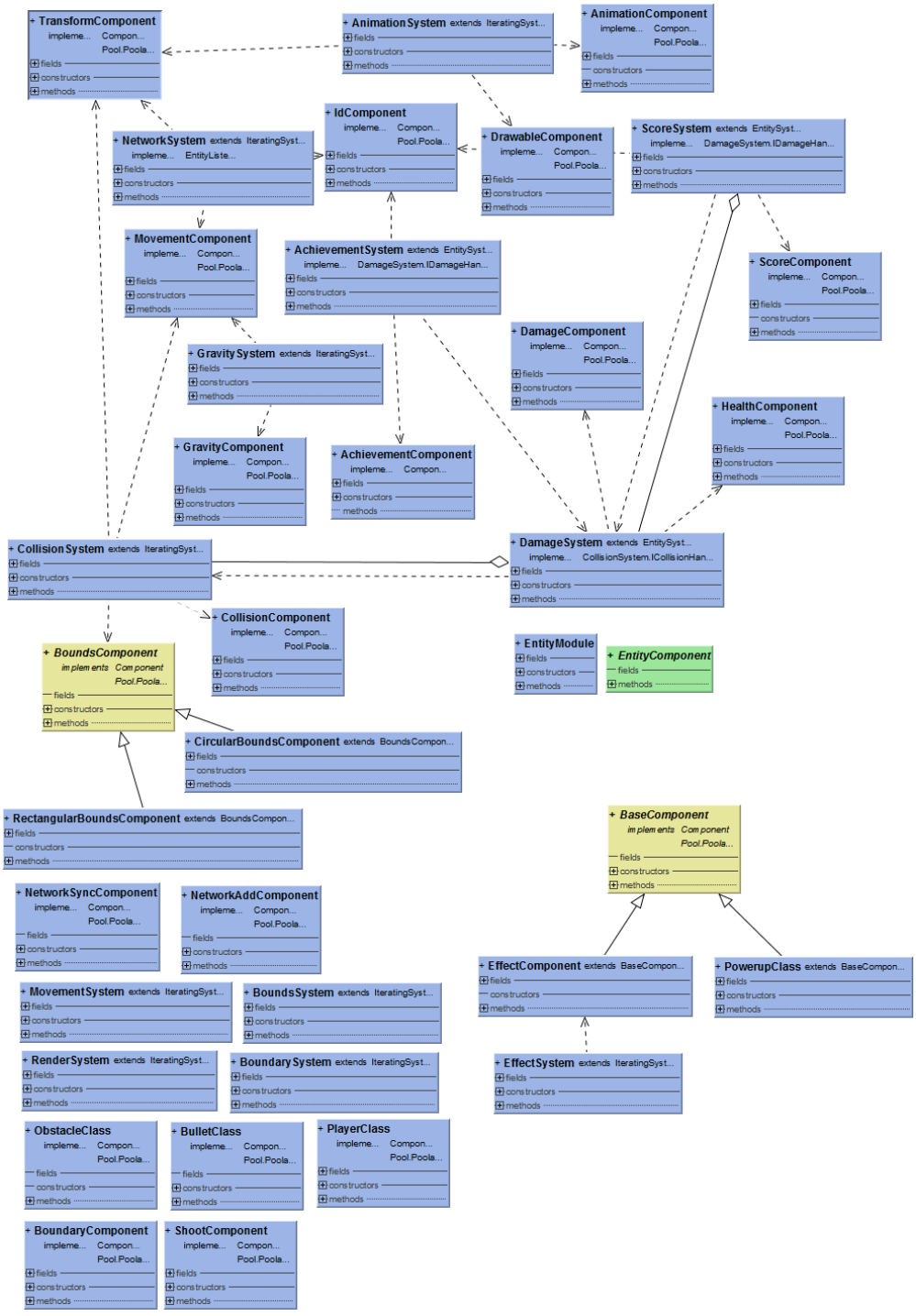


Figure 5: ECS package

2.3.1 Entities

Entities are empty objects with no internal state or behaviour. Entities are completely and wholly defined by their components and are not very interesting on their own. This completely avoids inheritance as every entity is simply a composition of all the components it needs.

2.3.2 Components

The components contain all the state and in some cases behaviour for the entities they belong to. Entities can be composed of any number of entities, and they can be added, removed and manipulated at run-time.

2.3.3 Systems

The various systems are responsible for processing the entities and their components. They can add, remove and manipulate components and on every update, and some also emit notifications to listeners during the process. The CollisionSystem emits events for collisions, and the DamageSystem emits notifications for damage events. Systems can be observed by each other as well as other external objects such as the World. There are also some non-logic systems such as the RenderSystem, NetworkSystem, AnimationSystem, AchievementSystem and ScoreSystem that perform tasks not directly related to the game.

2.4 Patterns

2.4.0.1 Singleton

The singleton pattern is used for some application-wide services through the @Singleton annotation provided to Dagger2. This ensures that the same instance will be returned for all requests within the scope of the application.

2.4.0.2 Factory Pattern

Variations of factory patterns are used heavily within the application to simplify creation of components and resources, as well as caching.

2.4.0.3 Object pool

All components implement the Poolable interface and the performance of the ECS relies heavily on it's ability to reuse objects.

2.4.0.4 Observer

The observer pattern and variations of it are used throughout the application. Many ECS System classes are observable, eg. the CollisionSystem, and other systems as well as

other objects can observe it and receive notifications of events. Other classes that are observable include but are not limited to World, INetworkService and it's implementation, DamageSystem, the ECS Engine and many many more.

2.4.0.5 State

A variation of the state pattern, adapted to ECS, is used to control the state of entities. The state of an entity is defined by the presence or absence of certain components, as well as their internal properties. Eg. when an entity has the MovementComponent it is moving, if it doesn't it's state is stationary.

2.4.0.6 Strategy

The strategy pattern is used to attach behaviour to components. Eg. an entity that can fire has a Shoot component. The ShootComponent has a reference to a IShootEffect that implements a strategy for how the gun behaves. Similar behaviour can be seen with the EffectComponent and it's reference to IEffects.

2.4.0.7 Data Locality

The ECS system utilizes locality when processing entities. Components are stored in LibGDX arrays that are stored as contiguous blocks of memory and the engine processes these in order.

2.4.0.8 Dirty Flag

The dirty flag pattern is used to avoid updating UI components that haven't changed. This avoids recalculation of positioning in the UI component tree. When components need to be recalculated they are invalidated and processed on the next update.

2.4.0.9 Spatial partitioning

A implementation of a quad tree to partition entities based on position for more efficient collision detection has been started but not completed in time. The current naive implementation has shown to cause some performance issues on low powered devices when sufficiently many entities are visible on the screen. This implementation solves that completely.

2.4.0.10 Game loop

The game loop is a core pattern of any loop-driven game, and is part of the LibGDX framework. The variant used in the game is a variable time-step loop with it's upper time-step bound clamped to 100ms. The update-rate is clamped to 30 fps by Android.

2.4.0.11 Update method

The update method pattern is used by every part of the ECS and also for presenters and views.

3 User manual

3.1 Requirements

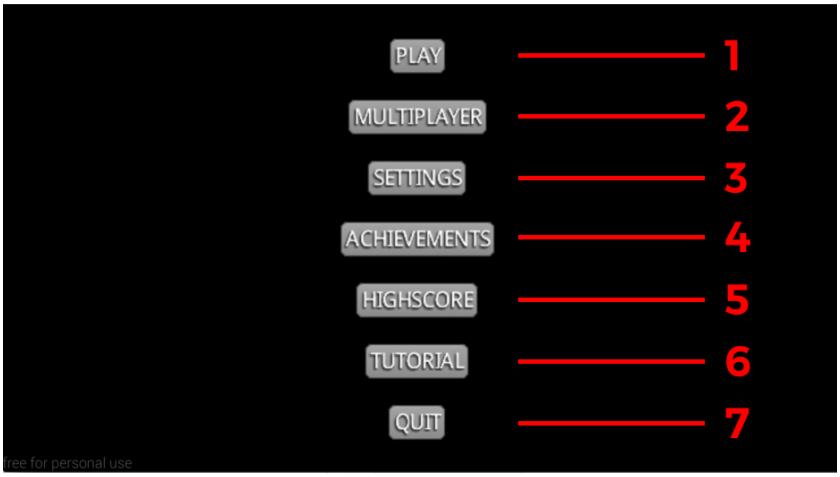
To run this game you'll need an Android device running Android version 4.4 (KITKAT) or higher.

3.2 Running the game

To access the game you'll need to download the game "Penguins in Space" from Google Play Store.

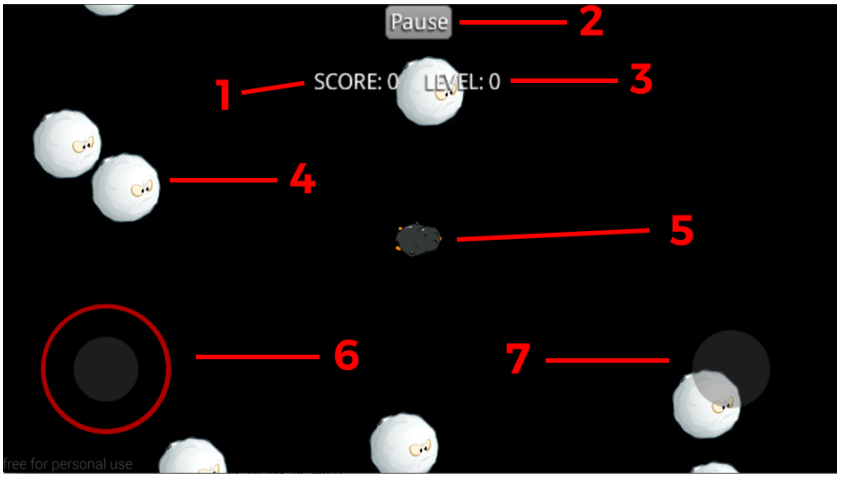
3.3 Game functionality

3.3.1 Main menu



In the main menu you can start a new single player (1) or multi player (2) game. You can change the game settings (3) or see your achievements (4) and highscores (5). At the bottom of the page you can access a tutorial (6) to get some information about the basic functionality of the game.

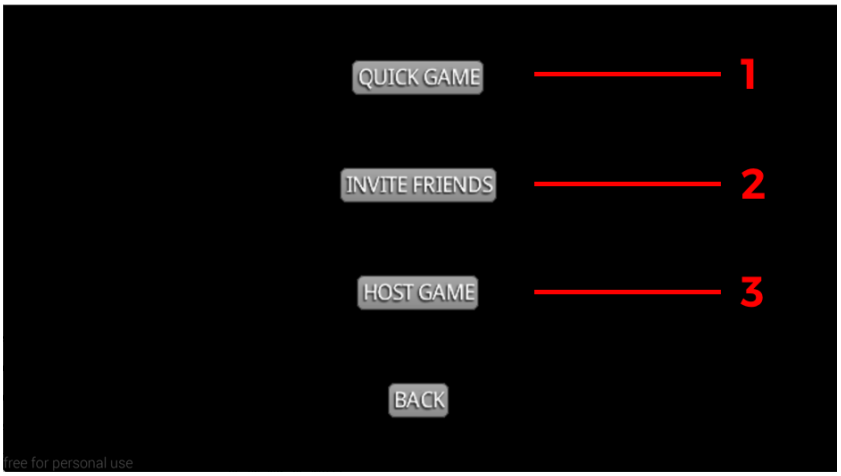
3.3.2 Single player mode



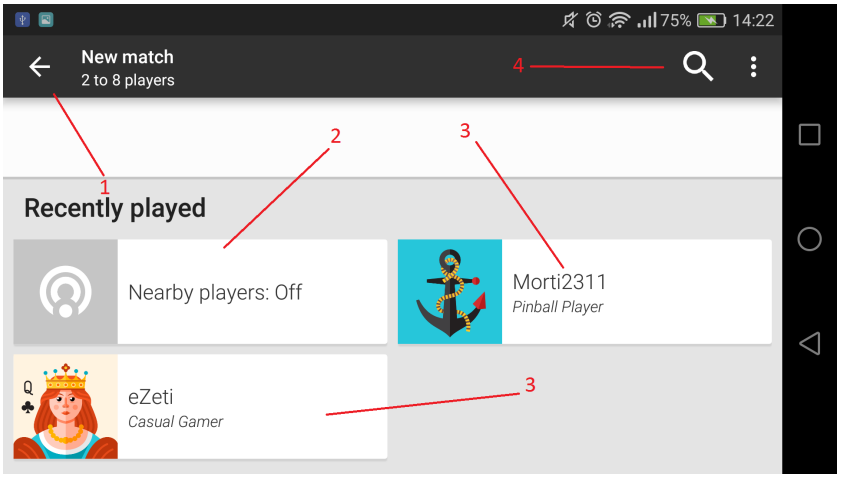
In single player mode you can fly around shooting snowballs. You can navigate your player (5) by using the left joystick (6) and shoot snowballs by pressing the right button (7). When shooting the snowball-monsters (4) you increase your score (1) and eventually you'll increase the game level (3). To pause the game you can press the button at the top of the screen (2).

At the pause screen you can resume the game (1) or access the game settings (2), these settings are the same as the ones accessed at the main menu and will be covered in 4.3.4. You can also quit to the main menu (3) or exit the application (4).

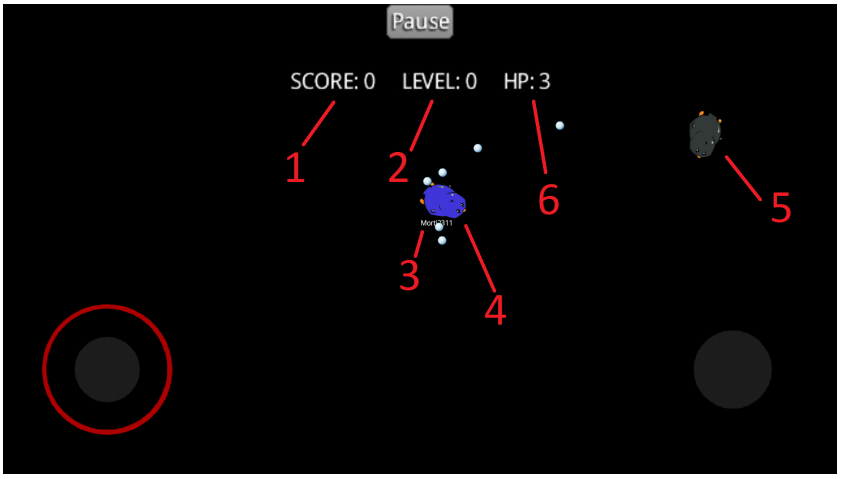
3.3.3 Multiplayer mode



Above figure shows the main menu in Multiplayer mode. "Quick game" (1) leads you to play game with automatically matched players, and "Host game" (3) leads to play game with one virtual player. The "Back" will lead you to go back to the main menu. If you want to play with specific friends, click "Invite friends" (2), which will provide you opportunity to invite others, as illustrated in below figure.
"Quick game" and "Host game" are not properly implemented in our game though because of time limitation.

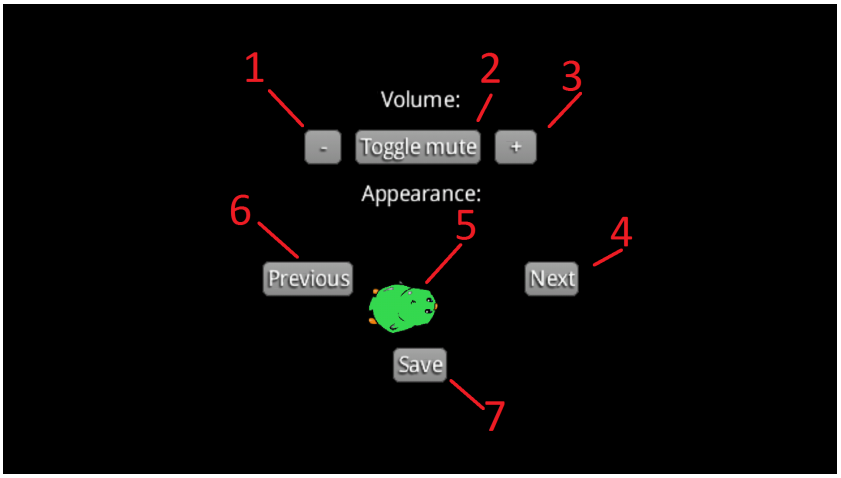


From the Invitation Page, you can go back to previous menu via clicking the Back-Arrow (1). The players you have recently played with are listed in the "Recently played" subwindow, who you can select to play with (2,3,4). Use the search button (4) to search other players.
After you have selected the player(s), the game will start after the other players have accepted your invitation. Below figure shows the playing screen in multiplayer mode with two players.



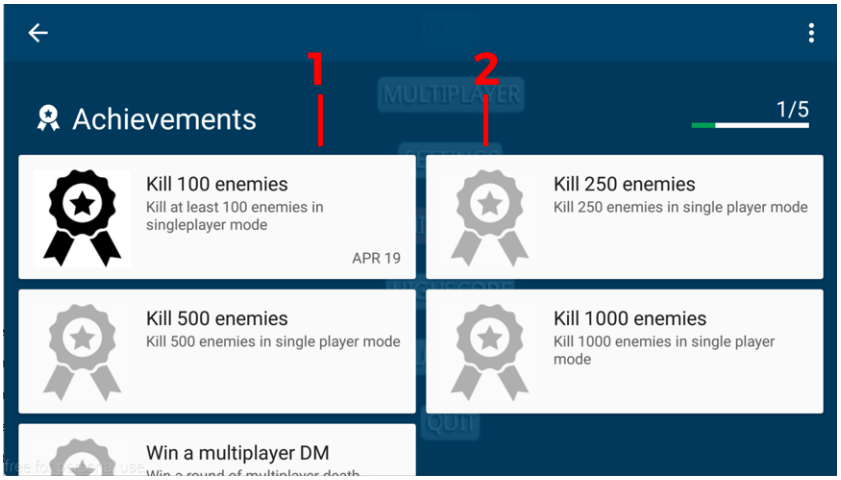
Your score, level and HitPoint (1,2,6) are shown on the top. The character for yourself and the other players (4,5) float on screen, and the bullets (3) shot are also visible. Same as in Singleplayer mode, you control your character with the joystick on the bottom-left corner and shoot with the button on bottom-right corner.

3.3.4 Settings



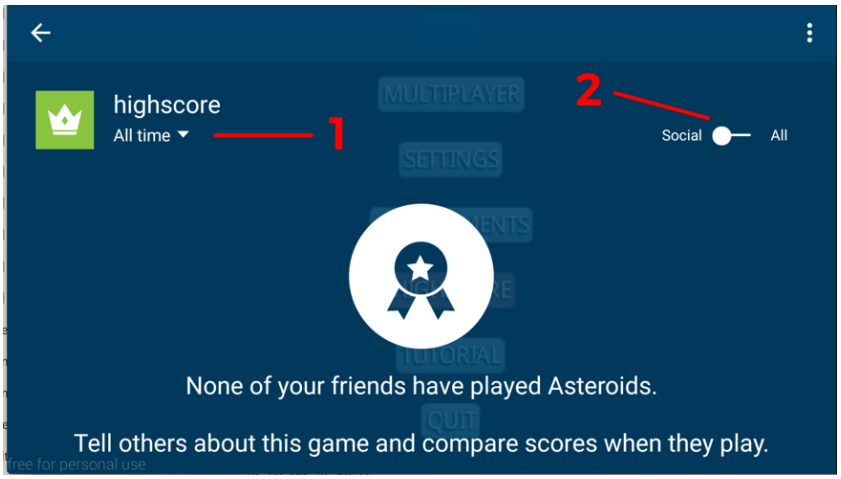
At the settings screen you can decrease (1) or increase (3) the volume, you can also mute (2) the music completely. By pressing the ”Next” (4) or ”Previous” (5) button you can change your appearance (5). If you wish to save the settings that have been altered you can press the ”Save”-button (6) at the bottom of the page.

3.3.5 Achievements



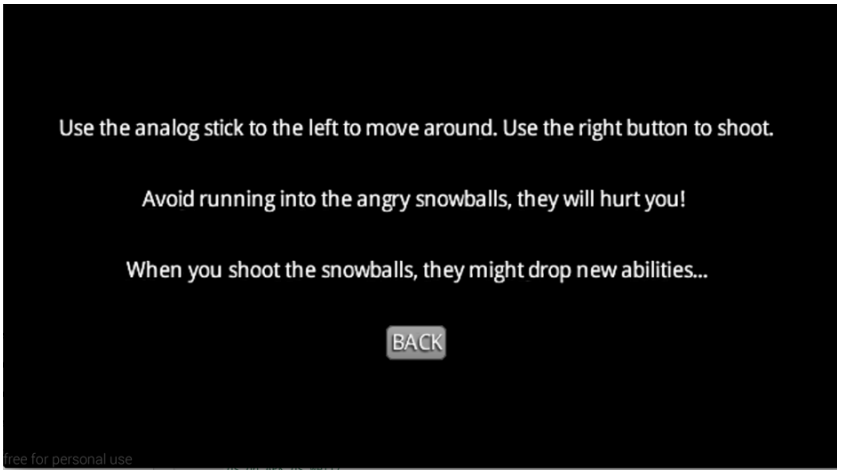
These are the the achievements you've unlocked by playing the game. Unlocked achievements are black (1) while locked achievements (2) are greyed out.

3.3.6 Highscore



Here you can get an overview of how you are doing among your friends. You can change the time period (1) and toggle between social or all results.

3.3.7 Tutorial



The tutorial screen includes basic information about the functionality of the game.

4 Test report

The test result for all functional and quality requirements are documented in this section, one requirement in each table. The identity of requirements, f.eg. FR1, is same as what's used in Requirement document.

4.1 Functional Requirements

This subsection lists test result for functional related requirements.

FR1: The user should be able to control game with touch gestures	
Executor:	Henry Sjøen
Date:	28.03.17
Time used:	1 min
Evaluation:	Success The menu buttons respond well to the touch gestures.
Comment:	No problems with navigating the menu, both in-game and in the main menu.

FR2: The user should be able to control its character with virtual joystick and virtual buttons	
Executor:	Stein-Aage Klaussen
Date:	28.03.17
Time used:	1 min
Evaluation:	Success
Comment:	The character responds as one would expect when using the joystick. It can take some time to get used to the acceleration of the penguin when moving. The fire button shoots when pressed, and responds well to multiple presses.

FR3: The user should be able to choose different game modes from the menu.	
Executor:	Stein-Aage Klaussen
Date:	19.04.17
Time used:	1 min
Evaluation:	Success
Comment:	In the menu you can select between "Play" and "Multiplayer". It may be more familiar for the user to use a pattern like "Start" and then select between "Single player" and "Multiplayer" mode.

FR4: The user should be able to turn off sound and sound effects.	
Executor:	Sander AC
Date:	07.04.17
Time used:	2 minutes
Evaluation:	Success
Comment:	Three buttons were added to the settings to decrease or increase the volume as well as toggle mute. It was easy to understand this, however the label "volume" was a bit misleading as it only controlled the music volume. Also it starts out being set to 0, which can be confusing as it was hard to tell whether the sound was not working correctly or not.

FR5: The user should be able to quit to main menu while in-game.	
Executor:	Stein-Aage Klaussen
Date:	28.03.17
Time used:	30 seconds
Evaluation:	Success
Comment:	When the pause button is pressed in-game, you see the option button "Quit to menu". Button works as expected and takes you back to the main menu.

FR6: The user should be able to quit the game from main menu.	
Executor:	Sander AC
Date:	28.03.17
Time used:	30 seconds
Evaluation:	Success
Comment:	The "quit" button made the game exit, and returned to the android "desktop".

FR7: The user should be able to pick different characters	
Executor:	Sander AC
Date:	31.03.17
Time used:	4 minutes
Evaluation:	Success
Comment:	The option to change character was found both in the main menu "settings" screen and also in the in-game pause screen. It was easy to change between characters with the "previous" or "next" button. The setting was also saved for next time you opened game. You could see the change straight away after pressing the "save settings" button and then resuming/starting the game.

FR8: The user should be able to play online multiplayer or single-player.	
Executor:	Stein-Aage Klaussen
Date:	19.04.17
Time used:	4 minutes
Evaluation:	Success
Comment:	In the "Multiplayer" menu you can select between "Quick game", "Invite friends" or "Host game".

FR9: If the player gets shot by another player or enemy the player should lose health.	
Executor:	Sander AC
Date:	20.04.17
Time used:	5 minutes
Evaluation:	Success
Comment:	Worked well.

FR10: There should spawn different power-ups during the game.	
Executor:	Morten Lundenes
Date:	28.03.17
Time used:	2 minutes
Evaluation:	Success
Comment:	When killing the snowballs, power-ups were spawned for me to pickup.

FR11: If the player collides with a powerup, the player should get that powerup.	
Executor:	Morten Lundenes
Date:	28.03.17
Time used:	1 minutes
Evaluation:	Success
Comment:	The power-up was activated when collided with.

FR12: The user should have an in-game menu.	
Executor:	Morten Lundenes
Date:	28.03.17
Time used:	30 seconds
Evaluation:	Success
Comment:	When starting a new game, a "pause" button is displayed at the top. When pressed a menu is displayed with different options.

FR12.1: The game should not pause when the in-game menu is chosen while playing in multiplayer mode.	
Executor:	Morten Lundenes, Sander Aker Christiansen
Date:	22.04.17
Time used:	2 minutes
Evaluation:	Success
Comment:	When press the 'Pause' button on the top, the Pause menu appears, but the game continues in background and you can go back to the game after pressing 'Resume' button.

FR12.2: The game should pause when the in-game menu is chosen while playing in singleplayer mode	
Executor:	Stein-Aage Klaussen
Date:	19.04.17
Time used:	2 minutes
Evaluation:	Success
Comment:	The game pauses when the "pause" button is pressed in-game. It resumes when the "resume" button is pressed as expected.

FR13: The game should show the player a score, based on a performance criteria.	
Executor:	Stein-Aage Klaussen
Date:	19.04.17
Time used:	1 minute
Evaluation:	Success
Comment:	When a snowball is destroyed the score increases by one.

FR14: The player has option to go through tutorial when starting the game.	
Executor:	Henry Sjøen
Date:	19.04.17
Time used:	1 minute
Evaluation:	Success
Comment:	A "tutorial" button is displayed in the main menu, and gives a textual tutorial of the mechanics of the game. It could be a little more detailed, but it provides enough information to play the game.

FR15: If the user collides with an angry snowball he loses hitpoints.	
Executor:	Sander AC
Date:	05.04.17
Time used:	1 minute
Evaluation:	Success
Comment:	While colliding with angry snowballs my hitpoints were decreased.

FR16: If the user loses his last hitpoint he dies.	
Executor:	Sander AC
Date:	05.04.17
Time used:	5 minute
Evaluation:	Success
Comment:	After losing my last hitpoint I died and a score was displayed.

4.2 Quality Requirements

This subsection lists test result for quality related requirements.

M1: Add a new power-up, playable character or enemy.	
Executor:	Lars-Kristian
Date:	21.04.17
Environment:	Design time
Stimuli:	Wishes to add a new power-up, playable character or enemy
Expected response measure:	2 hours
Observed response measure:	some hours
Evaluation:	Adding a new power-up or entity takes about 5 minutes. Modifying the player or an enemy only involves a few lines of code. Power-ups require slightly more, roughly 20-30 lines depending on the complexity.
Comment:	Modifying the game in any way is very quick and easy to do.

M2: Modify existing graphics	
Executor:	Sander AC
Date:	09.04.17
Environment:	Design Time
Stimuli:	Wishes to modify existing graphics
Expected response measure:	The graphic is updated and seamlessly integrated into the game in half an hour.
Observed response measure:	Editing the invulnerability-graphic and uploading a new version took about 20 minutes.
Evaluation:	Success
Comment:	Updating the graphic was not problematic at all.

M3: Modify a powerup	
Executor:	Lars-Kristian
Date:	21.04.17
Environment:	Design Time
Stimuli:	Wishes to modify an existing powerup or add a new one
Expected response measure:	In half an hour, integrated into the game without specific adjustments to other modules.
Observed response measure:	5 minutes
Evaluation:	Adding or modifying a power-up only requires a few lines of code, and all power-ups and effects are encapsulated from the rest of the system. There are generic systems in place to spawn, apply and remove effects, so the only logic that has to be implemented is the logic specific to the effect.
Comment:	Working with power-ups is extremely simple thanks to a solid effects system. The most time-consuming task is coming up with ideas and textures for the effects.

M4: Modify logic of a component	
Executor:	Lars-Kristian
Date:	21.04.17
Environment:	Design Time
Stimuli:	Wishes to modify an existing component's implementation
Expected response measure:	In half an hour, without affecting the component's interface or other code.
Observed response measure:	5-20 minutes
Evaluation:	The time needed to modify a component depends on the complexity of the modification. The ECS system makes it very easy to make modifications in general, and modifications have been made constantly throughout the project without problems.
Comment:	The ECS architecture makes it very easy to modify any part of the game logic thanks to the low coupling obtained with composition.

P1: Client receives data from server	
Executor:	Lars-Kristian
Date:	21.04.17
Environment:	Normal operation
Stimuli:	Server sends data of other players
Expected response measure:	Latency below 30 ms for the update of local state
Observed response measure:	5-10ms
Evaluation:	Since the multiplayer used p2p the latency is only affected by the connectivity between peers. This has been measured to be roughly 5-10ms. The client additionally predicts movements which reduces the effect of latency and dropped packets.
Comment:	The latency is better than expected for both wifi and cellular network.

P2: Periodic client update	
Executor:	Lars-Kristian
Date:	21.04.17
Environment:	Normal operation
Stimuli:	Periodic update
Expected response measure:	Throughput of 30 successful updates per second for steady frame rate
Observed response measure:	30
Evaluation:	The game runs smoothly with 30 fps on all tested devices.
Comment:	We've had to limit the number of entities to achieve this, by limiting the rate of fire for guns that shoot multiple projectiles. The performance reduction is mainly due to the collision detection algorithm. A better solution would be to implement a quad tree partitioned on positions to increase the efficiency of the collision detection algorithm.

I1: Player wants to initiate a multiplayer room	
Executor:	Lars-Kristian
Date:	21.04.17
Environment:	The client and servers are discovered at runtime
Stimuli:	A request to set up a new player room
Expected response measure:	100% of the sent requests that are in the correct format and order are accepted
Observed response measure:	All game requests are being delivered to the invited players.
Evaluation:	This is working as expected.
Comment:	Invite friends is the only form of multiplayer that is implemented. Quick game and host game modes would need to be tested again when implemented.

U1: First-time play	
Executor:	Sander AC
Date:	08.04.17
Environment:	Runtime
Stimuli:	Wants to play for the first time
Expected response measure:	Finished the tutorial within 2 minutes and has a basic understanding of the game and its controls
Observed response measure:	After starting the user test and telling the user that "You want to play the game, but do not know the mechanics of it", the user successfully navigated to the tutorial and after reading the instructions she had an overview of the game. This was done in about a minute.
Evaluation:	Success
Comment:	The user did understand the mechanics of the game after reading through the tutorial, so she was not overwhelmed the first time she started up a single-player game. She understood how to control the penguin and that getting hit by obstacles would harm her penguin.

U2: Modify game through settings	
Executor:	Sander AC
Date:	08.04.17
Environment:	Run time
Stimuli:	Wants to modify a setting under options (like volume of music or effects)
Expected response measure:	Setting is found without any navigation-errors in 15 seconds or less
Observed response measure:	The user did find the settings, both from the main menu as well as through pausing the game while playing.
Evaluation:	Success
Comment:	The user did find the settings, yet during the user test she was confused by representing the current appearance as a string only. Hence a visual representation was added after this test.

5 Relationship with architecture

There are very few inconsistencies between the planned architecture and the actual implementation. This is partly because the initial architecture was very general and highly abstracted, to allow for an agile development process. The team was mostly new

to game development, Android, LibGDX and ECS, which made it particularly difficult to plan details in advance. The architectural document defined the broad perspective of the system, and the team then worked out the details as needed. The development process remained agile throughout, starting with the simplest clean implementation that worked, and continuously re-factoring and improving the code throughout the project.

5.1 Modifiability tactics

5.1.1 Cohesion

The implemented architecture has a high cohesion in most of it's components. This is especially true for everything related to the ECS module. Each component and system relates to a single and very specific concept. This is also true for most of the other components such as the services, views and presenters. The least cohesive class is the World class, which handles everything related to the game world at a higher level.

5.1.2 Coupling

There is very low coupling between components in general. Much of the inter-module communication is through interfaces, and most dependencies are injected through constructors for inversion of control.

5.2 Usability tactics

It proved difficult to implement any meaningful usability tactics for the game except a basic tutorial, a simple GUI and a familiar control scheme. Other tactics such as keeping a model of the user was not implemented. This could have been and partly was discovered during the ATAM and planning, but a decision was made to make an attempt at it regardless.

5.3 Interoperability tactics

Interoperability with the Google Play Games Services service has been achieved through the implementation of AndroidNetworkService which uses the Google API to establish and manage connections.

5.4 Performance Tactics

Caching of assets has been utilized to reduce the resource demand. The use of concurrency was considered but found to not be needed to achieve the performance goals.

5.5 Backend as a Service (BaaS)

The communication with the BaaS is implemented through the AndroidNetworkService according to the planned architecture.

5.6 Peer-to-peer

A peer-to-peer mesh network is used to communicate between clients, as specified in the architecture.

5.7 Model-View-Controller (MVC)

The MVC pattern has been replaced with the MVP pattern, which is similar but not equal to the initial suggestion. The difference lies in how data is passed from the model to the view, and how much logic is implemented. This had a negligible impact on the architecture overall. It could possibly have been discovered during the ATAM but that wouldn't have made any difference.

5.8 Entity Component System (ECS)

The ECS was utilized as planned, with great success. This was the first time anyone on the team tried an ECS system, and it proved to be a massive improvement over traditional inheritance based entity systems, especially in terms of modifiability.

5.9 Design patterns

All the design patterns described in the architecture document has been utilized, likely in addition to many others that weren't planned or documented. Planning where to use design patterns is like planning where to use for loops and if statements.

6 Problems, issues and points learned

Even though our team has been programming quite a bit before, making a multiplayer game for the Android OS using LibGDX, ECS and Google Play Services was something new to most of us. New frameworks means new things to learn, and of course takes time. The project did end up getting quite complex, and the members have all learned quite a bit from developing a project of this size while keeping documentation and architecture up to date.

The first challenge our team had to overcome was working with frameworks unknown to most of us. LibGDX and ECS took some time to get used to for most of us. Even though these frameworks fit well for our application, they did initially reduce our productivity as a development team. Fortunately the members who had experience with these were really helpful when the rest of the team had questions. None of us had ever used the real-time multiplayer API of Google Play Services either, but challenges related to learning to use this were overcome as well.

The second challenge was trying to coordinate as a group. The members of the team all have different schedules, so setting times to meet were hard. Additionally people were unavailable for periods of time due to travelling. This meant that our time frame was

reduced quite a bit. Fortunately we set up a Slack-channel to communicate and used the architecture and requirements to distribute the work. Still, in hindsight, having more specific times to work together in person and being able to plan ahead better would have benefited us greatly. It would also have reduced the stress of having two weeks less of development time.

Surprisingly enough we did not have any major issues in using Git. An experienced member of the team initially set some ground rules for how we should use the VCS and it worked out really well.

All in all our group learned a lot from this project. Having a proper requirements and architectural document as a basis for what to implement worked out very well as most of the structure of the code was already decided on. As stated in the previous chapter this was done in an agile manner, as the initial architecture was quite abstract, but still there are few inconsistencies between this and the implementation. Even though we had some issues, like slicing away two weeks of development time due to the Easter break and members travelling, the project was a success.

7 Changes

This section records change history for this document.

Date	**Change History**	**Comments**
April 23, 2017	First released version	None