UbonGo*Dream*

Implementation

TDT 4240 - Software Architecture

COTS: Android

Mathias Mikalsen

Julia Bröhan

Matus Smykala

Sindre O. Rasmussen

Albert Hambardzumyan

Group 19

Primary quality attribute

Modifiability

Secondary quality attribute

Usability

Spring 2016

1.0 Introduction

This document describes the implementation of the project. Design and implementation details contains information on how the architecture was implemented in the project. The user manual contains a descriptive guide how to run the client and server software, as well as configuration. Test report displays the results of testing different scenarios. The relationship with the architecture describes how the implementation is using the architecture. The last part of this document are describing some of the issues and problems of the implementation and some points we have learned.

2.0 Design and implementation details

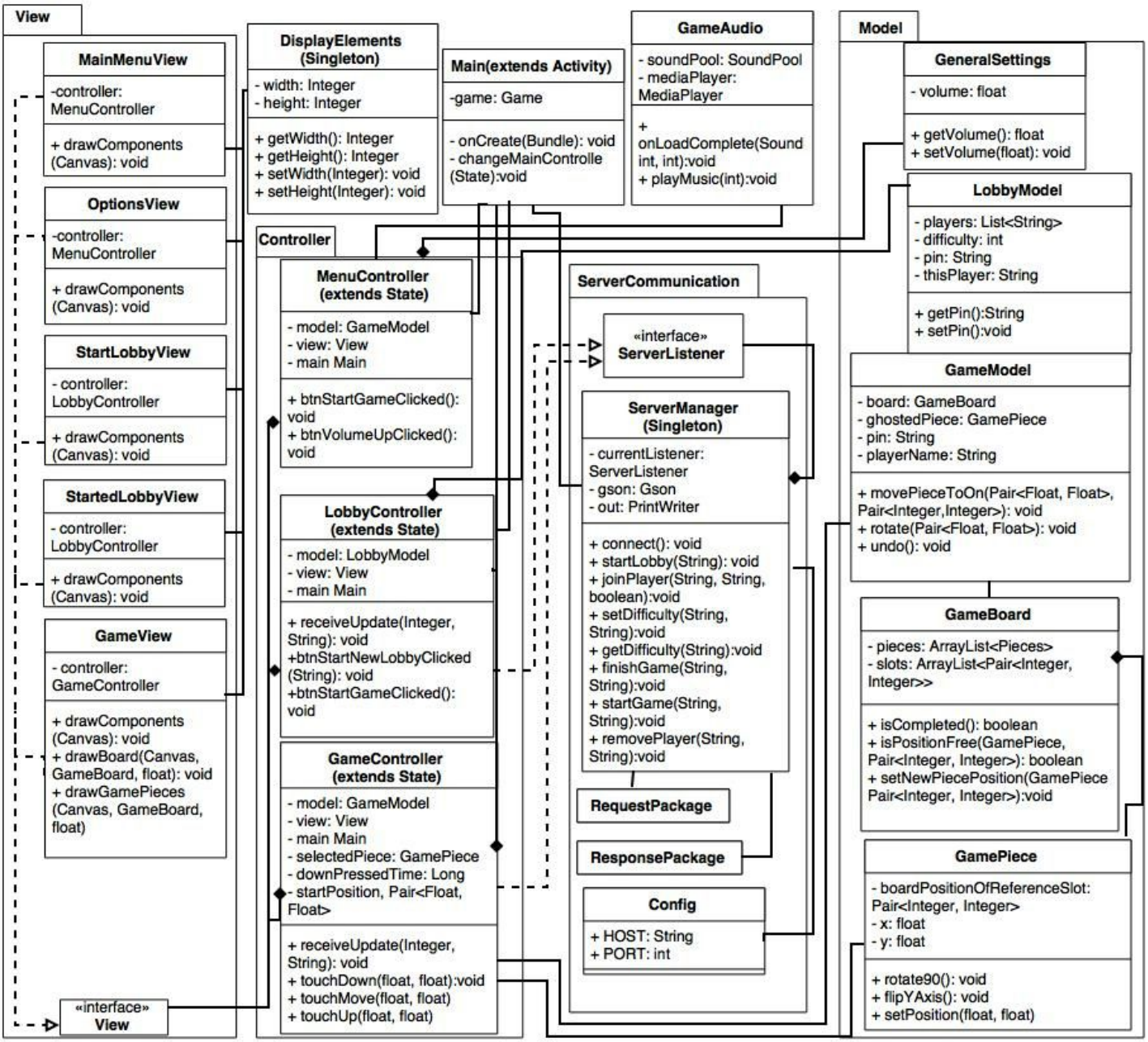
The project uses the Sheep library [1] because it makes handling user input and graphics easier. The biggest effect the Sheep library had on the project is related to the way Sheep handles different screens or states. Our game has three states; menu, lobby and game. Menu is the starting screen of the game and allows users to control the volume of sound. The lobby is where the user either connects or starts their own lobby. The game is where the users are playing the game. All three states use the MVC pattern with their own models, controllers and views.

The view in the game state is receiving touch input and button input, which it then hands down to the controller. Controller uses these to decide what function to run on the model. An example would be if the user taps the screen twice within a short timespan it registers the action as a double tap, the controller then tells the model to rotate the piece at that location.

An exception in the use of Sheep concerns the use of a dropdown menu, text boxes and game music. The library has no dropdown menu or text box functionality, and therefore had to be created using the standard android methods. This meant that the elements had to be placed on top of the canvas and removed manually. Music also had to bypass the library, due to using sounds in Sheep only plays for a couple of seconds. To properly play music the standard android media player was used.

The singleton pattern were used several places where we found it useful. Objects such as music player, networking manager and display elements were all singleton, making them easy to reach whenever they are needed.

Class diagram for client:

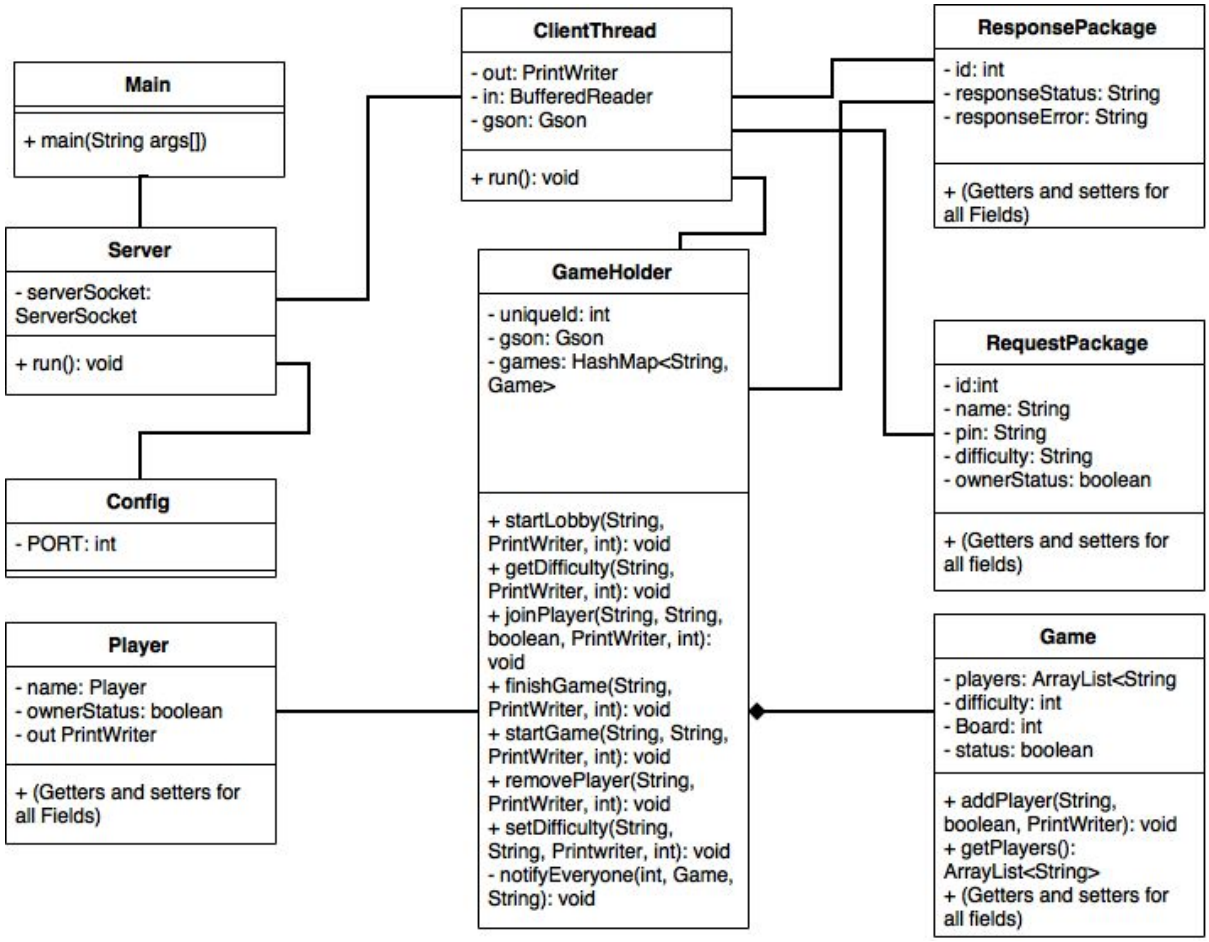


Class	Description
MenuController	This class is the controller which manages the menu-state of the game. When it is pushed on the stack, it instantiates a GeneralSettings-model and displays a MainMenuView.
LobbyController	This class is the controller which manages the lobby-state of the game. When it is pushed on the stack, it instantiates a LobbyModel and displays a StartLobbyView, based on choices made by the user, it can also change to showing a StartedLobbyView. It also implements ServerListener, so that it can receive updates from the server.
GameController	This class is the controller which manages the gameplay-state of the game. When it is pushed on the stack, it instantiates a GameModel and displays a GameView. It also implements ServerListener, so that it can receive updates from the server which notifies if someone has won the game.
GameBoard	This class is part of the model and holds the GamePieces and a board. It also has logic for evaluating positions and completeness of a board. Basically it represents the board the player is solving.
GamePiece	This class is part of the model and represent a single piece that a player can put on a board. The class also contains logic for flipping and rotating the piece and keeping track of the position of the piece.
GameModel	This class is part of the model and holds the Gameboard. It is responsible for generating it and its Gamepieces. The class provides all the actions a player can perform with the GamePieces and has also functionality for undo operations, creation of a ghosted GamePiece and testing, if the player wins the game.
LobbyModel	This class is part of the model and holds and evaluates values collected from the user when he/she is filling out the game information.
GeneralSettings	This class is part of the model and holds and evaluates general settings for the game, which for the time being is just game volume, but could be extended to hold other settings that would be handy.
MainMenuView	This view holds all visual elements(buttons, text, picture etc.) needed for the start screen which is the first to appear. It also specifies position for the visual elements. It uses the DisplayElements-singleton to get common visual elements found in the game+height and width for the screen to calculate position for visual elements.

OptionsView	This view holds all visual elements(buttons, text, picture etc.) needed for the options menu. It also specifies position for the visual elements. It uses the DisplayElements-singleton to get common visual elements found in the game+height and width for the screen to calculate position for visual elements.
StartLobbyView	This view holds all visual elements(buttons, text, picture etc.) needed for the part of the lobby where the user fills in name and pin to start or join a game. It also specifies position for the visual elements. It uses the DisplayElements-singleton to get common visual elements found in the game+height and width for the screen to calculate position for visual elements.
StartedLobbyView	This view holds all visual elements(buttons, text, picture etc.) needed for the part of the lobby where the user sees players joined and the difficulty for the joined game. The view can have two modes, owner-mode or joined-mode. What mode it is, is indicated by a boolean given in the constructor. In owner-mode the user can change difficulty and start the game. In joined-mode, the user can only watch players currently in the game. It also specifies position for the visual elements. It uses the DisplayElements-singleton to get common visual elements found in the game+height and width for the screen to calculate position for visual elements.
GameView	This view draws the board based on updates given by the GameModel, given through the use of the GameController. It also hold methods for drawing a winner-text and a back-button when the game is finished. It tracks the touch-interaction from the user and passes it down to the controller to update the model. It uses the DisplayElements-singleton to get common visual elements found in the game+height and width for the screen to calculate position for visual elements.
DisplayElements(Singleton)	This singleton class is used to hold elements that frequently occur in the views of the game. It also holds the height and width for the screen. The views use this class to get frequent visual elements and to calculate positions for visual elements by percentage of height and width for the screen.
Main	This class extends Activity and is the class that is run to start the game. It is used by the controllers to change the state of the game. It is also necessary to reference this class when textboxes and dropdown-menues are drawn in the views. This because Sheep is not supporting these functionalities, so we have to get them through the

	Android activity that is running, which is represented by the Main-class.
GameAudio	This class holds necessary elements and methods to play the game music.
ServerListener(Interface)	This interface is implemented by LobbyController and GameController and says that the controllers need to implement the method receiveUpdate(int type, String update). This make it possible for the classes to get and react to response-messages from the server. This interface is also central in our implementation of the observer-pattern.
View(Interface)	This interface is implemented by all the views in the game.
Config	This class holds the port number and IP-address which the client should connect to. This class should be edited to configure the client when the game is tested with a locally running server.
ServerManager(Singleton)	This class holds the methods which any class in the game can call to send RequestPackages to the server. Most of the requests will result in a response that is picked up by the ServerManager, which notifies a ServerListner.
RequestPackage	This class defines the format for a request sent to the server
ResponsePackage	This class defines the format for the responses received from the server.

Class diagram for server:



Class	Description
Main	The class which is run to start the server
Server	A class which listens on a port specified in Config, and creates a new ClientThread for each request it gets.
GameHolder	This class holds all the games currently running on the server. It also has methods which is called by ClientThreads to do updates on the games based on the type of request.
ClientThread	One clientThread is created for each request. The ClientThreads performs the needed actions based on the request.

Game	A representation of a game running in the serer. The class holds all information that the server need about a game.
Player	A representation of a player which is joined to a game. A player can have status as owner or not.

For more detailed information about the methods in the classes and what they do, we refer to the comments found in the code. Almost each method in a class should have a comment saying what it does and what kind of parameters it takes in.

3.0 User manual

3.1 Requirements

3.1.1 Building

1. Android Studio 1.5.1 or later versions

3.1.2 Running

1. Minimum Android Version 3.0
2. The environment where the server is run needs to have permissions to use the internet connection of the server-machine.

3.2 Running the game

The game system consists of two programs. The server program, which is meant to run on a dedicated server, and the client program, which is the app running on the device. For simplicity and time reasons we have decided that it is best that the evaluation of the project should be done on a locally running server. Therefore the server must be set up locally before the game can be tested. Following are the steps needed to set up the game for testing:

- **Step 1:** Unzip the UbongoClient project and the UbongoServer project repositories.
- **Step 2:** Open these projects in an IDE
    - Android Studio is preferable, since they are developed in this IDE, but another IDE could work as well.
    - You may have to specify project SDK. Any version of Android SDK should be sufficient. The server can run in regular Java.
- **Step 3:** Run the server, by running the Main (com.example.UbonGo.Main) in the UbongoServer project.

- **Step 4:** Edit the Config-class (com.example.UbonGo.ServerCommunication.Config) in the UbongoClient project. You should change the HOST to match the IP-address for the machine you are running the server on.
    - Use ipConfig in the cmd/terminal to find out what your IP is.
- **Step 5:** Run the client by running the Main-class (com.example.UbonGo.Main) in the UbongoClient project.
    - You can run it on an emulator or a connected device

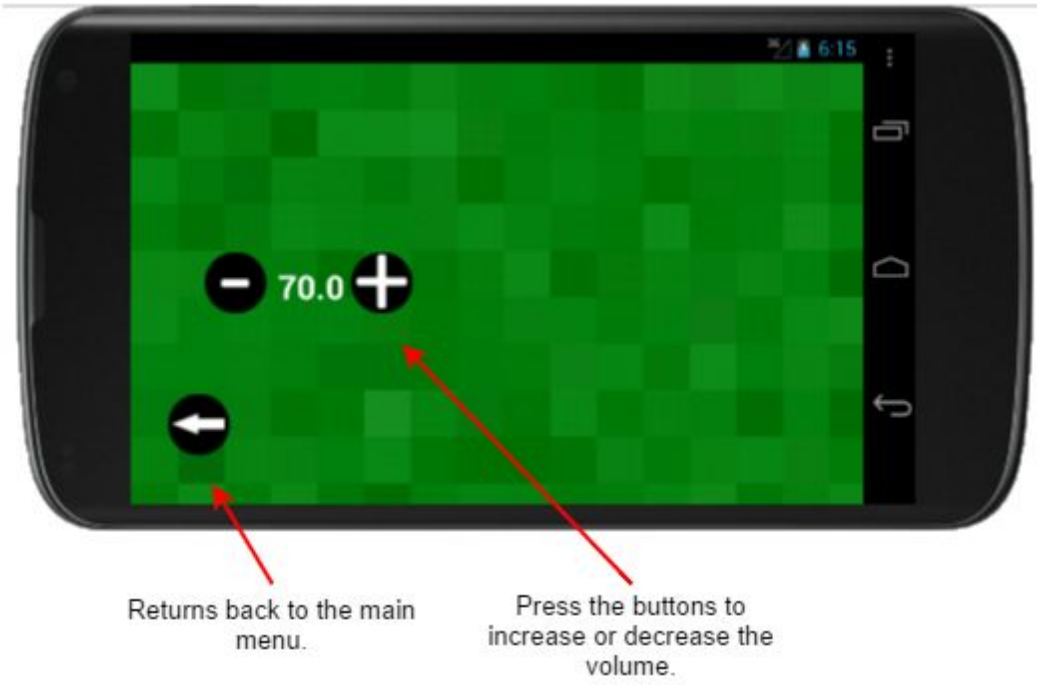
IMPORTANT NOTICE: Through the development of the game it became clear to us that it was a bit more challenging than expected to create a functioning networked game. Though we have learned a lot from the experience, we have had some difficulties making the game stabile. In testing new issues popped up all the time. We have managed to fix many of the worst errors, but still the game is a bit unstable and can crash during playing and lobby setup. There is also an issue with permissions when running the server locally. If for instance Android Studio does not have permission to access the network, you could experience difficulties to get connection between the server and the clients. If the game crashes, it usually works to restart. When that is said: We have managed to play through the game against each other on multiple devices without any problems several times, so it should work.

We were told, when asking about this in lecture, that if the ones evaluating the game could not manage to run it, we should get the opportunity to come and demonstrate the game. If that should be necessary, though we believe that it won't be necessary, take contact with us by sending a mail to: sindreor@stud.ntnu.no, or contacting us in other channels you have access to.

3.3 Playing the game

3.3.1 Main Menu

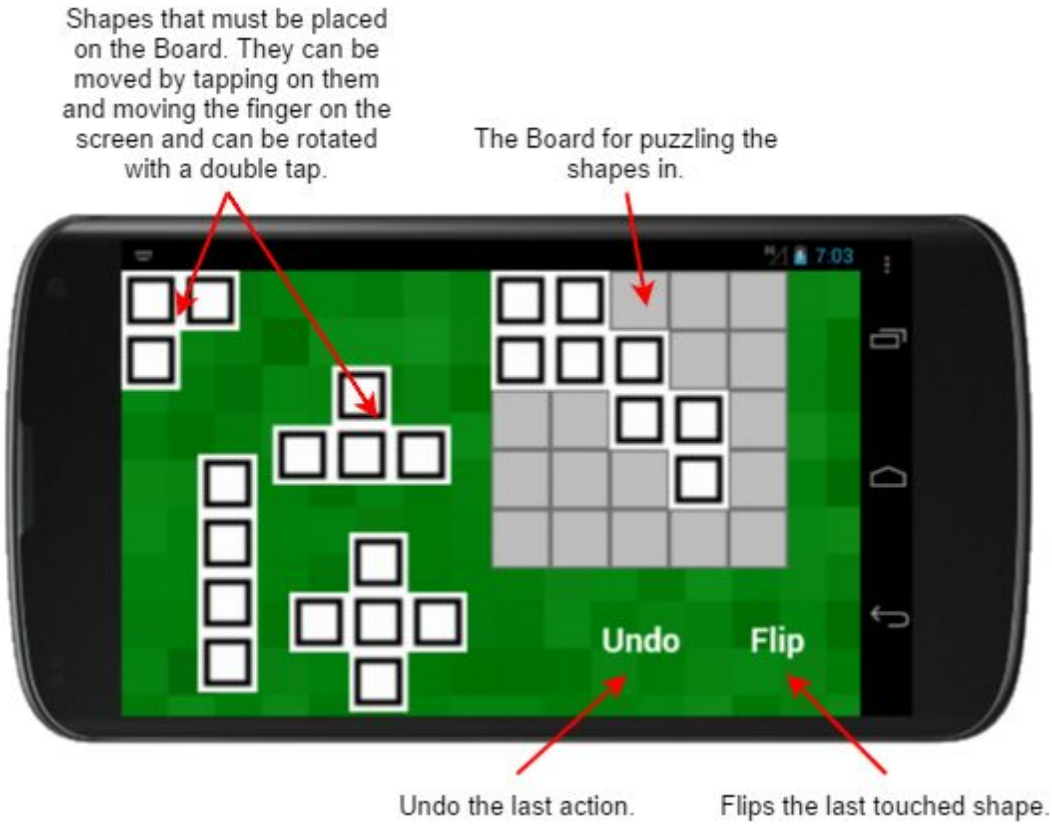
3.3.2 Options Menu



3.3.3 Start Game Menu

3.3.4 Game Lobby

3.4.5 Gameplay



4.0 Test report

4.1 Function requirements

FR1 - Music	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	1 minute
Evaluation	Success
Comment	When the game menu opens the music starts playing automatically.

FR2 - Settings mode	
Executor	Mathias Mikalsen

Date	22.04.16
Time used	1 minute
Evaluation	Success
Comment	The volume buttons in the options menu raises and lowers the volume when clicked.

FR3 - Create Game	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	2 minutes
Evaluation	Success
Comment	After entering a name the user is allowed to create a lobby and the pin for the lobby was displayed on screen.

FR4 - Difficulty Level	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	1 minute
Evaluation	Success
Comment	Selecting difficulty in the dropdown menu changed the difficulty, the game board changed accordingly.

FR5 - Start the Game	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	1 minute
Evaluation	Success
Comment	After creating a lobby pressing the start game button started the game.

FR6 - Number of Players	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	2 minutes
Evaluation	Success
Comment	After creating a lobby, when new players joined their names showed on the screen.

FR7 - Join the game	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	1 minute
Evaluation	Success
Comment	The host of the game told me the pin number of his game, and with it I was able to join the lobby.

FR8 - Rotate Shapes	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	1 minute
Evaluation	Success
Comment	Double tapping a shape rotated the shape, depending on if it was possible for the shape to be rotated or not.

FR9 - Drag and Drop Shapes	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	2 minutes
Evaluation	Success
Comment	Dragging shapes onto the board moved them to that location on the board if the board had space for the piece. If the board couldn't fit the shape, it went back to its previous location.

FR10 - Game Over	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	5 minutes
Evaluation	Success
Comment	When I finished the game, the game over screen popped up and said that i had won. When another player won, my game got aborted and the game over screen popped up telling me that he had won.

FR11 - Flipping Shapes	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	1 minute
Evaluation	Success
Comment	After selecting a piece by tapping on it, pressing the flip button flipped the piece over.

4.2 Quality requirements

M1 - Adding new view element	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	10 minutes
Evaluation	Success (partly success)
Comment	By changing the view class of whatever state of the game you are in, allows easy ways to draw an image, text or buttons on the screen. No other classes were changed. If we count code for doing actions when buttons are clicked, we will affect two classes, controller and view, if you look it from that perspective, we did not have complete success. But the main intention behind the requirement was concerning the visual elements themselves and the functionality to position them, and on this we have succeeded.

M2 - Add new types of shapes and frames	
Executor	Matúš Šmýkala
Date	22.04.16
Time used	20 minutes
Evaluation	Success
Comment	By changing the file that contain board information adds a new board to the game, and the server needs to be edited to know that an extra board has been added. The test took shorter time than the requirements.

M3 - Adding a new game screen	
Executor	Mathias Mikalsen
Date	22.04.16
Time used	30 minutes
Evaluation	Success

Comment	Creating a new game screen requires creating a new model, view and controller. The good thing about this is that they do not affect the other parts of the system. By telling the game to use the new controller, the controller then uses its default view and model. It is important to say that we do not consider buttons for navigations to the new view from the existing ones as code that is part of the new gamescreen. Therefore we managed to add a new gamescreens by just by just adding new classes. 0 of the existing classes were affected.

4.3 Usability requirements

U1 - Position of dropped shape is adjusted to frame grid	
Executor	Matúš Šmýkala
Date	22.04.16
Time used	3 minutes
Evaluation	Success
Comment	It is not necessary to drag and drop the shape precisely to the grid. If the shape is dropped a few pixels next to the grid, shape is placed correctly anyway.

U2 - User should be able to undo an action	
Executor	Matúš Šmýkala
Date	22.04.16
Time used	3 minutes
Evaluation	Success
Comment	It is possible to undo moving or rotating the shape by tapping on undo button.

U3 - Simple game joining	
Executor	Matúš Šmýkala
Date	22.04.16
Time used	5 minutes

Evaluation	Success
Comment	It is easy to understand what the user needs to fill in the text fields while joining the game. While testing, we haven't had any problems with understanding of joining the game. We managed to fill in correct information on first attempt, thereby fulfilling the requirement.

5.0 Relationship with the architecture

Issues leading to some breaks with the original architecture:

- The architecture designed earlier in the project only used the Sheep functions for drawing and user input, but as it turns out some features needed other methods to work. Using the standard Android methods for dropdown and text boxes is breaking with the architecture, but not in any substantial way. If we had discovered this problem earlier a way to avoid it could have been to try to find another framework than Sheep, which also supports text input and dropdown-menus.
- Do to unclarity in the planning on how to do the server communication, our final implementation had quite a lot more classes concerned with server communication. To avoid this a better structured planning phase with some test implementation of ways to do the server should have been done.
- We have also added some helping classes/interfaces in both the model and the views which were not in the original class diagram. These are still following the main concept of the architecture. We also added one more view. These changes we see as a natural part of the implementation process.
- The GameAudio class was neither in the original class diagram. Better research in the planning phase could have discovered the need for this before the implementation.
- It is also important to say that the planned architecture did not include all methods in detail, since we planned that we should experiment a bit in the implementation to find the best way to do stuff instead of forcing ourselves to implement methods planned without experience on the problems to be solved.

Apart from the points mentioned above, we managed to follow the architecture pretty well. Our planned architecture made it easy to structure the work and add modifications as we discovered the need for them. We can also mention that the packages in the development view was a good way to divide the work between group members.

6.0 Problems, issues and points learned

At the end of the project all the parts that people have been working on individually had to be combined into a functional game. We knew we would be experiencing bugs during this process, but we did not know we would have as many bugs as we did.

One of the biggest problem with the game was making multiplayer work smoothly, and not crashing either server software or client software. Players would crash on connect, on leave, on start, on playing and on quitting. We did our best to fix the bugs and make it stable, but there are probably several more bugs in the game that could crash it. In case of

a crash, sometimes just trying again seemed to fix the problem. The problems occurred when we played on several different devices, so we suspect that small differences in versions or something similiar connected to the fact that not all devices are similar could have caused the problems.

Another problem arose during the development of the game's model and view. We had one person work on the view and controller, and another working on the game model. The result was the two people working together having to discuss on an interface between the two parts.To properly render the game pieces on the screen there had to be a universal coordinate system that could be scaled with the size of the screen on the phone. This ended up with the model working in one coordinate system, and the controller in another. The controller had to convert forwards and backwards to display coordinates and model coordinates.

Throughout this project we have learned:

- Problems can arise from any part of the code
- When people are working on different parts of the code, communication is extremely important.
- Usefulness of architectural patterns and design patterns, especially the MVC architecture.
- It is difficult to follow the architecture patterns, and one can easily do something that contradicts them.
- It is difficult to know in the beginning of the project what parts will go wrong.

7.0 References

[1] GitHub. (2016). tdt4240/sheep. [online] Available at: https://github.com/tdt4240/sheep [Accessed 22 Apr. 2016].