UbonGo*Dream*

Requirements

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

Table of Contents

1.0 Introduction

In this project our task is to plan and develop a functioning multiplayer game for Android or iPhone. The purpose of this phase, the requirement phase, is to create good and straightforwardly documents which describe the requirements for our game concept and explain our architecture and reason for our choices. Our intention with these documents is that they can be used to explain the game concept and architecture in such a way that by reading these documents, a developer can more easily start developing the game or make modifications to the game. Another important intention is that the documents can serve as a primary vehicle for communication among stakeholders.

We have chosen to base our multiplayer game on the already existing board game Ubongo. Our game will be implemented on Android by using the Sheep framework, and will be a networked game using a dedicated server. Players can create games on the server and join games on the server. A player joins a game by typing in a pin-code for the game he/she wants to join. One of the players in the game will be the owner of the game, and can at any moment start the game.

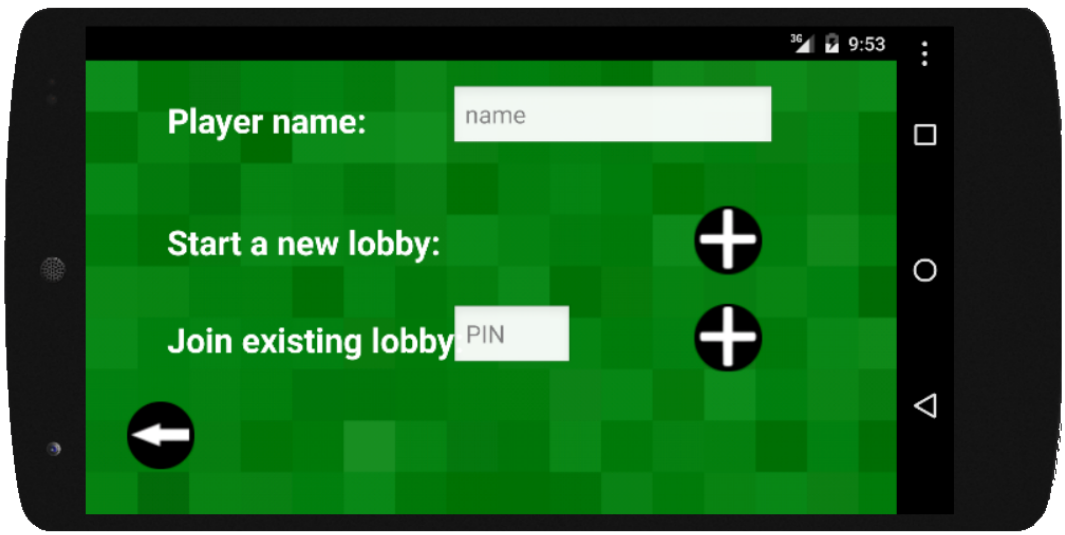

Figure 1: Ubongo game lobby

When the game is started, all players are presented with the same puzzle. The puzzle is a board consisting of a grid of empty cells and the player has to move, rotate and fit tetris-like pieces onto the board to fill all the empty cells. All the players in a game are

solving the same board at the same time on their own device. The first player who solves the puzzle wins, and then the game is over for all the other players in the same game.

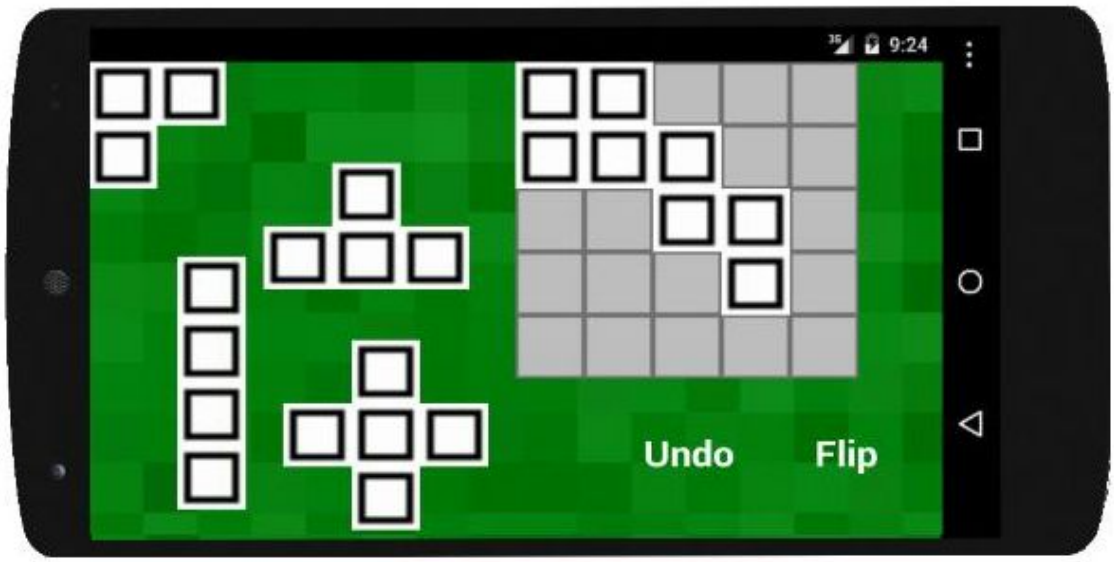

Figure 2: Ubongo gameplay

This document contains the functional requirements and quality requirements described in detail, as well as issues and references at the end of this document.

2.0 Functional Requirements

A functional requirement is a statement specifying what a system should do and how it should respond to stimulus. This chapter provides summed information about intended behaviour of the application in general and also in specific stages for particular inputs.

General game settings:

FR1 - Music
As the game starts a song should be played to make the game more attractive.
Priority: Low.

FR2 - Settings Mode
The user should be able to use the settings mode to mute/unmute the background song or the sound effects.
Priority: Medium.

Game administration:

FR3 - Create Game
The user should be able to create a new game and become the owner of the game. The system will generate a pin code, which can be used for joining the game.
Priority: High.

FR4 - Difficulty Level
The owner of the game should be able to choose the difficulty level of the game.
Priority: Medium.

FR5 - Start the Game
The owner of the game should be able to start the game whenever he wants .
Priority: High.

FR6- Number of Players
The owner of the game should be able to see the number of players joined the game, which will be used by the owner to decide when to start the game.
Priority: Medium.

FR7 - Join the Game

Any user who has a valid pin code should be able to join the game, if the game has not started.
Priority: High.

Gameplay:

FR8 - Rotate Shapes
The user should be able to rotate the shapes by clicking on them. Each click should rotate the shape by 90 degree.
Priority: High.

FR9 - Drag and Drop Shapes
The user should be able to drag and drop the shapes into the frame. If the shape cannot be dropped into the frame, because the space is occupied by another shape, than the shape automatically should be returned to its initial position.
Priority: High.

FR10 - Game Over
If the surface is completely filled, than the player wins the game and receives congratulations. All other players should be informed that game is over. Priority: High.

FR11 - Flipping Shapes
The user should be able to flip a shape, such that it is mirrored.
Priority: Medium

3.0 Quality Requirements

Primary and secondary quality attributes of the application are modifiability and usability. To achieve these goals, it is important to specify particular requirements. We set out objectives listed below.

3.1 Modifiability

M1 - Adding a new view element (text, picture etc.)
Adding a new visual element that shows in the GUI should not involve change to more than one class.
Source of stimulus: Developer
Stimulus: Some kind of non-interactive visual element such as text or picture is added to the GUI.
Environment: Design Time
Artifact: View
Response: The new visual element is implemented to the artifact.
Response measure: Not more than one class affected.

M2 - Add new types of shapes and frames
Adding new types of shapes and frames should be possible in 3h.
Source of stimulus: Developer
Stimulus: need of new shapes
Environment: Design Time
Artifact: The set of classes that represents the gameplay-state of the game.
Response: new shapes are generated for the games.
Response measure: 3 man-hours

M3 - Adding a new game screen
Creating an entirely new game screen which has its own unique GUI and unique functionality should be possible without affecting the classes and components for the existing game screens. (new game states should be added as new components instead of being implemented as part of existing components)
Source of stimulus: Developer
Stimulus: A new game screen with new functionality is added to the game.
Environment: Design Time
Artifact: View, controller and model
Response: A new game screen is implemented and works with the rest of the system.
Response measure: 0 existing classes are affected by the modification.

3.2 Usability

U1 - Position of dropped shape is adjusted to frame grid
When a user drops a shape inside the frame it should be placed according to the grid of the frame, such that the user doesn't need to worry about exact placement.
Source of stimulus: End user
Stimulus: User drops shape inside frame
Artifact: System
Environment: Runtime
Response: Shape is placed according to the grid of the frame.
Response measure: 100% of all dropped shapes are adjusted to the frame grid.

U2 - User should be able to undo an action
When a shape is moved or rotated, the user should be able to undo the operation.
Source of stimulus: End User
Stimulus: User presses undo after he moved or rotated a shape
Environment: Runtime
Artifact: System
Response: Shape is placed in the previous position
Response measure: User is able to undo the last action with the first attempt 19 of 20 times.

U3 - Simple game joining
The player easily understand what information is needed to join a game.
Source of stimulus: End user
Stimulus: User wants to join a game.
Environment: Runtime
Artifact: System
Response: The system presents what data(pin and player name) it needs from the user.
Response measure: The user should not misunderstand (fill in wrong information) more than 3 times.

4.0 COTS - Components and Technical Constraints

Our game will run on the Android platform, and we will also use the Sheep framework. These two will pose some constraints on our architecture and what possibilities the game can offer.

4.1 Android constraints

- We must have at least one class which extends the Android Activity class. Our architecture must therefore find a way to implement such a class. Our architecture must also be adapted to fit with the Android lifecycle of an application.
- Our game can only run on Android devices.
- Android devices come in many different variants with different screen resolutions. Our game must therefore implement functionality to adapt the graphics of the game to different screen sizes.
- Our game must base its user interaction on touch screen.
- Our game can only be distributed through the Google play store.

4.2 Sheep constraints

- Sheep uses a state-queue to administrate different game states. It also bases a game state on its state-class. To have any advantage of using the Sheep framework, our architecture needs to make use of the state-class, since it holds the functionality to draw graphic elements. In a way you could say that our architecture is forced to implement the state pattern where our game states are variants extending the state-class in the Sheep framework.
- All visual representation by the Sheep framework is done with the canvas-functionality in Android. The canvas is not meant to draw textboxes, so Sheep does not implement functionality to get text input from the user. Since our game is dependent on such input, our architecture must find a way to combine functionality for text input given by Android, with the use of the Sheep framework.

5.0 Issues

During the discussion of the architecture to be used in the project we quickly discovered that there was a slight language barrier. The team consists of two norwegian students and three exchange students, so everyone on the team are speaking english. Conveying thoughts and ideas could sometimes be difficult, and lead to discussions where parties were unable to properly understand each other. When discussing the architecture planning and arguing for what type of patterns to use in different areas could therefore take time, because everyone had to be sure they were on the same page.

Making a decision on what type of game we would make was very difficult. We had a lot of different ideas for the game and some of them were too ambitious. Knowing that the previous years in this subject people have been too ambitious, we tried to lower our ambitions and decrease the size of the project.

6.0 Changes

03.03.2016	First Draft
14.04.2016	Added introduction to chapters 2.0 and 3.0 Changed response measures in chapter 3.0
15.04.2016	Changed M1, M3, U1, U3, because they were unclear. Added the same updated introduction as in the architectural document
18.04.2016	Added categories to the functional requirements. This because the evaluation said that the order of the requirements was strange. Also reordered based on the categories. COTS - Components and Technical Constraints written
21.04.2016	Updated U2. Added FR11, because it was missing according to the evaluation.
22.04.2016	Final Draft

7.0 References

[1] Len Bass, Paul Clements, Rick Kazman, “Software Architecture in Practice – Third edition”, Addison Wesley, September 2012.