TDT4240 Software Architecture

Othello 2.0

Group 28

Group members:

Henrik M. Backer
Quentin Depoortere
Robin Alexander Finstad
Hanna Eide Solstad
Espen Sørhaug
Romain Vo

Primary quality attribute: modifiability

Secondary quality attribute: usability

Chosen COTS:

Android devices
Android Studio
LibGDX
Github
Google Play Services

TDT4240 Software Architecture

1 Introduction

1.1 Description of the project and the requirements phase

This project is a part of the subject TDT4240 software architecture. The goal of the project is to develop a multiplayer game with focus on the architecture of the design and implementation of the game itself.

Our initial thoughts were to make our multiplayer game competitive, so we decided to make a turn-based board game and chose Othello. This is a traditional game that is easy to learn, and we have given the user some possibilities to change the gameplay.

The requirement phase is important because it makes the outline for all needed functionality and quality attributes. It also describes the constraints that we need to accept and understand how they will affect further design decisions. In the requirement phase we are trying to understand all the different needs from the stakeholders. The outcome of this phase sets the foundation for the architectural phase and especially the architectural drivers.

1.2 Description of the game concept

The game is an implementation of the classic game Othello and uses the same set of rules as the original. We have gathered all our information on the game from [1] archimede-lab.org. The game starts out with four pre-placed discs with the starting position that can be seen in figure 1.1. Each player chooses a color and the players alternate taking turns. When your turn starts, you must place one of your game pieces on one of the available and legal squares. The goal of the game is to have more pieces in your color on the board than your opponent. This constitutes a win.

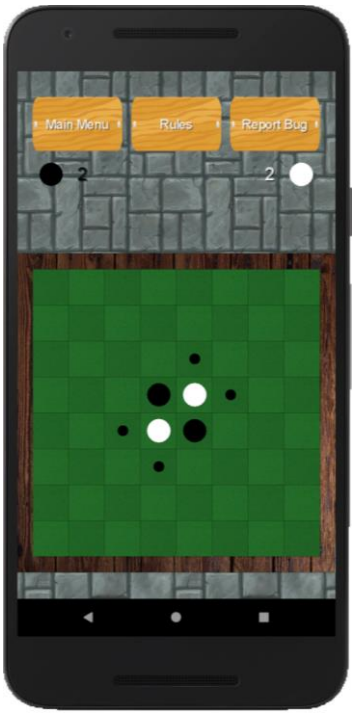

Fig 1.1 - Starting position for every game of Othello.

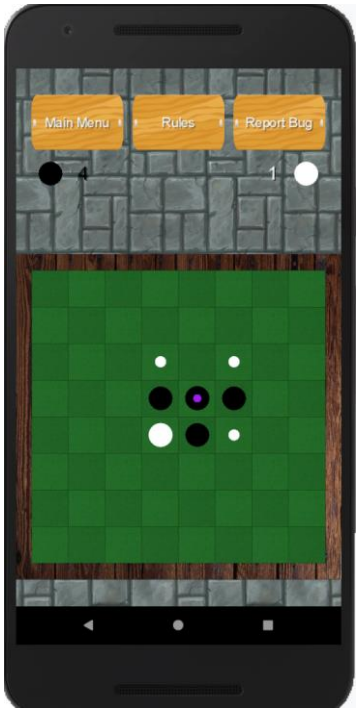

Fig 1.2 - A possible position after black has made one move.

The player with the black discs starts the game by placing a disc such that it surrounds a white disc on a line, either horizontally, vertically or diagonally. The white discs that are enclosed by the previously- and now newly placed black disc is flipped and changes color to black. In the starting position there are four possible moves for black and they are indicated by the small black circles beside the white discs, as seen in figure 1.1. If the black player chooses to place their disc in either of the two upper-right positions, the upper-right white disc is then flipped and turns black. One of these two outcomes can be seen in figure 1.2.

The game in its entirety is played by either player choosing one of the available moves presented to them by small circles in their respective color. After black has made their first move the game continues with each player alternating turns placing discs, it's also possible to flip several discs at once if the requirements stated above are fulfilled. When there are no more legal moves or there are no unoccupied squares left, the game is finished and the player with the most discs of their color on the board wins the game. An example of this can be found in figure 1.3. In this figure the white player won the game as there are no more legal moves and this player has 32 discs on the board, opposed to 23 discs which the black player has.

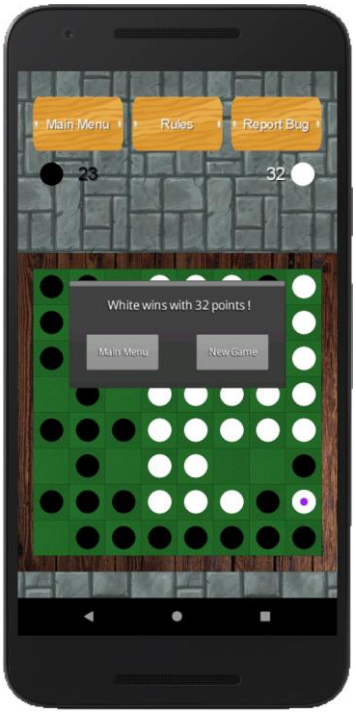

Fig 1.3 - The white player is the winner of the game as there are no more legal moves.

The game can be played in two different modes, the first being versus the computer. Here the player plays on their own device and there is no internet connection required. The second mode lets you play versus another human player, this mode is also offline, and you play by passing the device back and forth between the two players.

1.3 Structure of the document

This document started with an introduction that included information about the project, the game and the requirements phase. Further on in part 2, follows the functional requirements of our game. In part 3 we will explain the quality attributes we have chosen and their different scenarios. Part 4 describes the COTS and how each of them constraints the application. In the end we have part 5 with a couple of issues, part 6 with changes and lastly, references in part 7.

2 Functional Requirements

We have chosen functional requirements that we find essential to the application rated according to their importance for the application. The requirements are based on the concerns of the stakeholders. With these requirements it is also easy to see if we have fulfilled all the needed functions. The functional requirements are shown in table 2.1.

Table 2.1: Functional requirements

ID	Requirement	Priority
FR1	The user should be able to place their disc on all	High

	possible positions on the board.	
FR2	The user should be able to read the rules.	Medium
FR3	Player should be able to play a single player game against an AI opponent	High
FR4	The user should be able to quit the game and return to main menu.	High
FR5	The user should always see the score.	Medium
FR6	If there are no possible moves for either players, the game should end.	High
FR7	Disc color should change if the disc has been surrounded.	High
FR8	When a game ends, the player with the highest score should win the game.	High
FR9	When the player flips discs, the score should increase corresponding with the number of flipped discs and the opponents score should be reduced.	High
FR10	The player should be able to see all possible positions for disc placement.	Medium
FR11	The players should be able to see who's turn it is.	High
FR12	In offline mode, the player should have the opportunity to leave a game and resume that game later.	Medium

3 Quality Requirements

For the quality requirements we have chosen modifiability as the primary quality attribute and usability as the secondary. Our information about quality attributes is from [2] Software Architecture in Practice.

3.1 Modifiability

The different scenarios for modifiability are shown in table 3.1, 3.2 and 3.3.
Table 3.1: Adding of view

ID	M1
Source	Developer
Stimulus	The developer wants to add a view
Artifacts	Application
Environment	Design time
Response	Change made and the new view is tested by developers.
Response Measure	In less than an hour, without affecting the other views.

Table 3.2: Adding of player

ID	M2
Source	Developer
Stimulus	The developer wants to add a new player
Artifacts	Application
Environment	Design time
Response	Change are made and are tested by developers
Response Measure	Changes are made in less than an hour

Table 3.3: Change of board size

ID	M3
Source	Developer
Stimulus	The developer wants to change the board size
Artifacts	Application

Environment	Design time
Response	Change is made and tested by developers
Response Measure	In less than 10 minutes. Changes doesn't affect anything else.

3.2 Usability

Table 3.4: Checking of rules

ID	U1
Source	User
Stimulus	Want to check the rules
Artifacts	System
Environment	Runtime
Response	Open the rules and the user can navigate between the different rules
Response Measure	Open the rules view in less than 3 seconds

Table 3.5: Change settings

ID	U2
Source	User
Stimulus	Want to have change the settings
Artifacts	System
Environment	Runtime
Response	Open settings view and allow the user to modify
Response Measure	User should be able to change the settings in less than 30s

4 COTS Components and Technical Constraints

In the development of our game we have chosen multiple COTS-components, including Android Studio, Android OS and the framework LibGDX. Using these significantly helps creating the game, however they do set some constraints on the architecture. For teamwork purposes we will be using Github.

LibGDX
In addition to programming the application in Java, we decided to use the framework LibGDX. This is an open source framework mainly used for development of games in java. One of its advantages is the built-in methods to manage the input touches, the texture, the render, the display on the screen and different parts of game making. LibGDX will play a significant part in how we develop our app, as when programming, we must comply with the LibGDX project structure. The different views are organized as a stack of screens with only the topmost layer being active. Thus, instead of having one view that changes states, we have multiple screens representing the different screens of the application. We are going to be using our modified version of the design suggested by [3] Brent Aureli in his tutorial.

Android
We are programming in Android Studios, and specifically for Android. Because Android is an OS that runs on many different devices, issues regarding different screen sizes and resolutions may offer some constraints. We will taylor the game screens to android's portrait mode and take precautions so that the screens will work on devices of different sizes. Another concern is multiple OS-versions that exists on different devices. By adding support for users running Android 4.3 and above, makes 97% of all Android devices compatible to download and run our game.

Google Play Services
As we are programming in android, which is an OS maintained by Google, we are going to publish the application on google play. Therefore, we must comply with Google Play Services' policies to get approved. This entails everything from graphics to privacy and unethical procurement of money. Only parts of these points will be of consideration for our use.

Github
This is a very common, free-to-use code sharing platform. As everybody in the group have some degree of experience using git from other projects this will be a valuable tool for concurrent programming.

5 Issues

During this project we had challenges with implementing the online multiplayer version of the game. The cause is described more in detail in the implementation and testing document. The result was that we didn't complete a functional requirement and focused more on the other requirements. When we decided the functional requirements, it was difficult to foresee the difficulty we would have with online multiplayer. This hits on a point that can always be challenging in the requirement phase, it is not always foreseeing what is and what is not going to be a challenge.

We also had some troubles finding all the functional requirement in the beginning, but after starting with the programming we found more requirements and added some more details after the feedback from the instructors.

Other than this, we did not run into any problems regarding the requirements phase.

6 Changes

Time of change	Changes made	Comment
3rd may	Changed game description	Not of a significant character, simply more descriptive
5th april	Dropped FR3 and FR10	Removed option for online multiplayer and a duplicate FR for FR1
23rd march	Added AI agent for single player mode	Required an additional game controller, but did not impact rest of architecture

7 References

[1] http://www.archimedes-lab.org/game_othello/othello.html
[2] Bass, Clemens, Hazman - Software Architecture in Practise, Third edition
[3] Aureli, Brent - How to make Games: Flappy Bird youtube video tutorial