NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4240 Sofware Architecture

Implementation Documentation

Group 10 - A7:

Mats Byrkjeland
Andreas Drivenes
Anders Wold Eldhuset
Stein-Otto Svorstøl
Torstein Sørnes
Håkon Meyer Tørnquist

COTS: Android SDK

Primary quality attribute:
Modifiability

Secondary quality attribute:
Availability

April 26, 2015

Contents

1 Introduction

As today's software systems grow bigger and more advanced, and more systems are dependent on each other, it becomes essential to learn about how systems work together, and how one can plan for this. It is also relevant, as the field of software development grows, to know how known and tested techniques and design patterns can be used to achieve new goals and develop new systems. In our project in this software architecture course, the goal was to plan out an architecture and develop a game based on this plan.

So we started out with a requirements document, where we decided on what kind of game we wanted to make, and which requirements would have to be fulfilled for us to be happy about the game. The game idea was the traditional battleship, where there are two players with one board each. The board has ships on it, and each player are to try to hit the opponent's ships by guessing coordinates. We wanted to make this game on Android, with networking capabilities, meaning the two players should be able to play agains each other over the Internet.

This leads us to planning the architecture, which had some pretty extensive requirements as we wanted the game to be networked. We worked through the requirements and built the architectural requirements document. The proposed architecture went through an ATAM process with another group, and got some feedback. We then moved on to the implementation phase, which this document describes.

This document consists of six parts. The first is this, the introduction. We'll then move on to describing the details of the design and implementation. We'll show the user how to play in the user manual, then describe the testing phase. We finish up this report by describing any inconsistencies we've found between our implementation and the original architectural plan, and also a review of any issues we've encountered, and what we've learned from the project.

2 Design/implementation details

2.1 API on server

Route	**Request type**	**Result**	**Controller**	**Models**
/play	POST	{game: game model OR null}	Matchmaker.js	Board, Game
/turn/:username	GET	{username: String, lastMove String}	Turn.js	Board, Game
/fire	POST	{message: "Ongoing game' OR 'You won', shipWasHit: Boolean}	Play.js	Board, Game
/cancel	POST	{username: String}	Matchmaker.js	Board, Game

Table 1: Table describing the server's implemented routes, affected models and their response.

2.2 Game model (JSON)

The following list describes the structure of the game object exchanged between the client and the server:

game: Object

- player1: String
- player2: String
- next: String
- gameOver: Boolean
- lastMove: String
- finished: Boolean
- **Boards:** Array
  - **cells:** Array
    - containsShip: Boolean
    - hasBeenHit: Boolean

2.3 Class Diagrams

Server

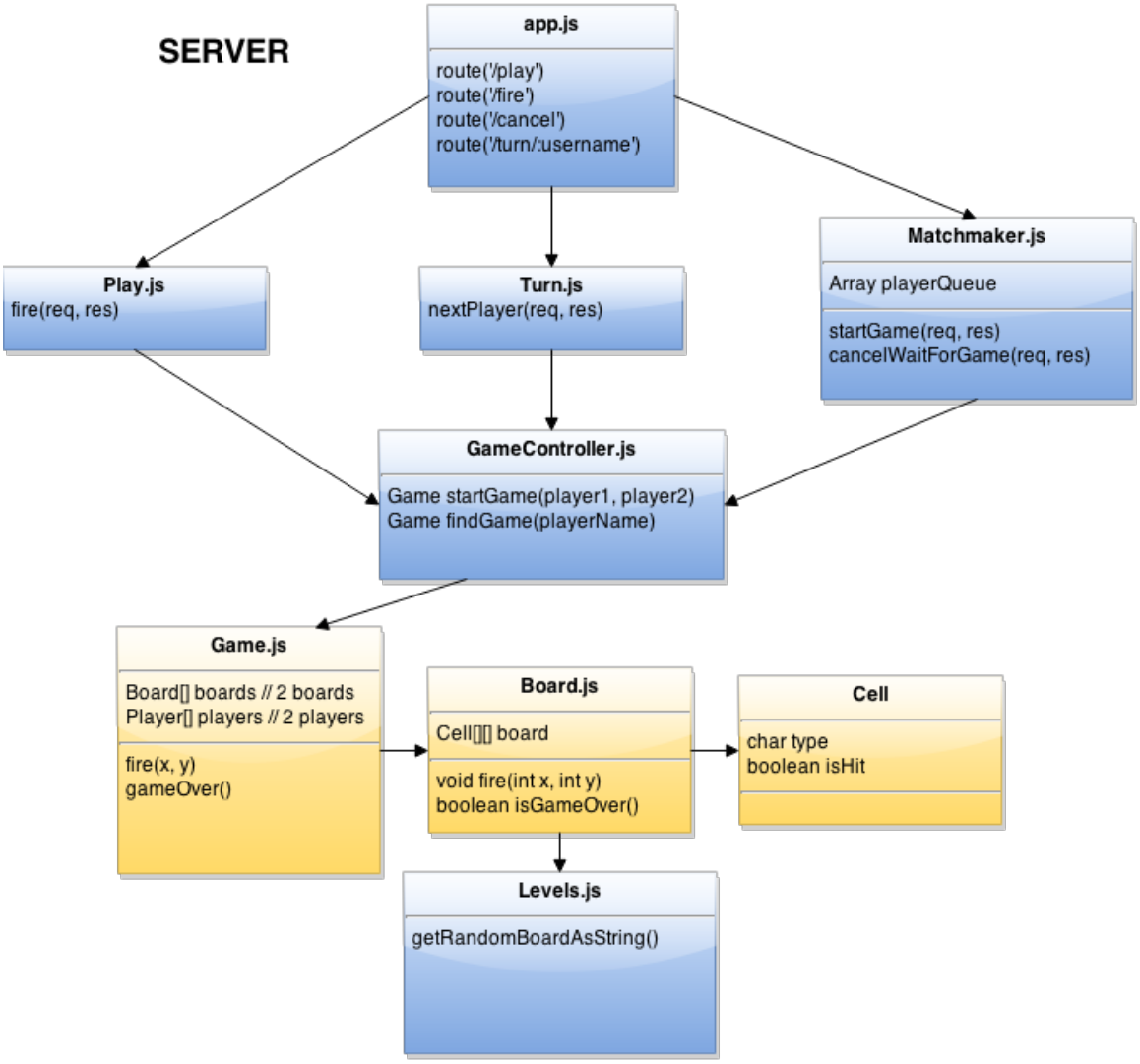


Figure 1: Class Diagram of the node.js server. Blue classes are controllers, yellow ones are models.

Client

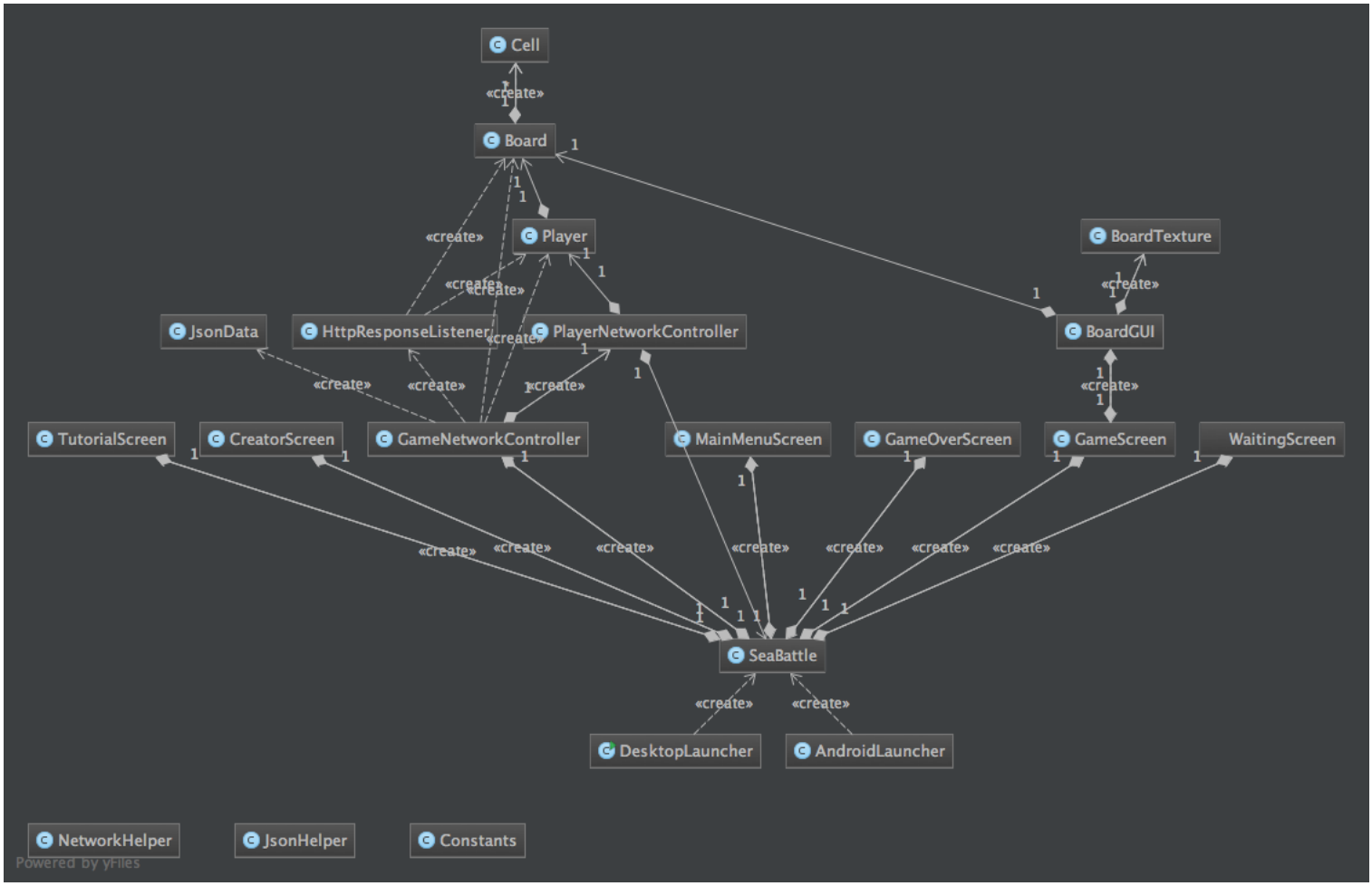


Figure 2: Class Diagram of the libGDX game

2.4 Textual description

Server

The server provides four different routes: /play, /fire, /cancel and /turn/:username declared in app.js in the server folder. Every route (see table 2.1) is controlled by a controller that invokes a centralized GameController.js to find and start games.

When a player wants to play, he sends his username to the play route. If an opponent is already waiting in the queue, the opponent is popped from the queue and a game is created, put in the database, and sent back to the player. Otherwise, the player is put in the queue, and has to regularly poll the /play route to see if a game exists. The maximum size of the queue is always 1.

To fire, a player sends a username and coordinates to the fire route, which is controlled by the Play controller. The ongoing game is fetched from the database, and a shot will be fired at the opposite player, if it is the current player's turn.

We used the object-relational mapper Sequelize [2] to handle the connection between our logical models (Game and Board), written in JavaScript, and their representations in

our database. Sequelize is initialized in /models/index.js, where the models are loaded and the association between games and boards is set up. While Sequelize can work with several different SQL database management systems, we used PostgreSQL specifically; the authentication details that Sequelize uses to connect to the database, are given through the environment variable DATABASE_URL, both in production and development.

Client

The game is started by the class "SeaBattle", as shown in the class diagram in figure 2. The different views, that is the classes ending with "screen" in the class diagram, each extends the LibGDX "Screen"-class. These screens has buttons which depend upon the observer-pattern, and which on press executes different actions. It is the "SeaBattle"-class which controls which view is currently shown. The BoardGUI represents a board, and has a grid layout. It uses the local board model as data for the view.

When it comes to networking, the client has two network controllers, namely GameNetworkController and PlayerNetworkController as shown in the class diagram. GameNetworkController sends a request to "/play" (see table 2.1), and if the response is a null game object, the client polls "/play" every fifth second to see if an opponent is ready to play. The request is invoked as a TimerTask which is run in a separate thread.

When a game object is ready, the client build models from JSON game objects which has the format described in subsection 2.2. The parsing is done by local models, helped by static helper methods in the class "JSONHelper". The models that are set are instances in PlayerNetworkController. The GUI can then use the models to create graphics, and the fire-method in PlayerNetworkController to send a new network requests. When e.g. a fire-request is sent (described in table 2.1). If the request is successful, the client updates the views and local models. This is possible because the game object holds both boards in the first place, and the server request is just meant to see if there's still connectivity and get the move of the opponent.

3 User's manual

3.1 Prerequisites

The server application is hosted in the cloud and should "just work". You will need a stable Internet connection on your Android device/desktop computer and an opponent to play against.

3.2 Installation instructions

Our goal is to make the application available through the Google Play Store. As of April 21, you can find our game published here: https://play.google.com/store/apps/details?id=com.mygdx.seabattle.android. In Play Store, the fastest way to find the game, is to search for 'Mats Byrkjeland'.

You can also visit the GitHub page [1] for the project. Download the APK file to your Android device. Allow untrusted APKs to run on your Android device (under Settings), and install the APK.

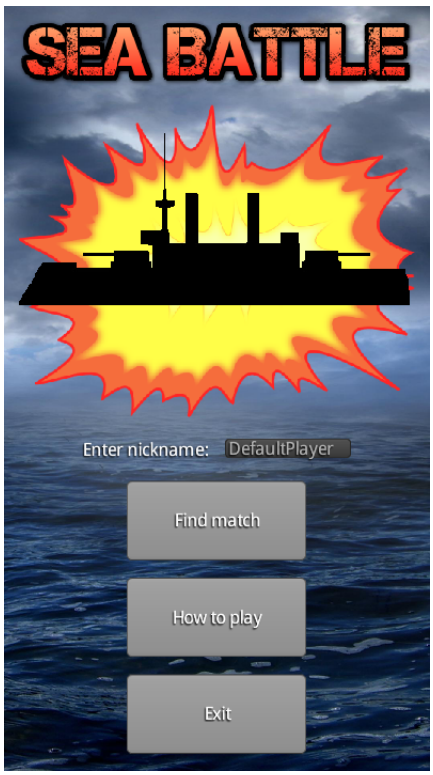
The APK file is included in the final delivery, bundled together with a JAR file that runs in the JVM on OS X, Windows and Linux. Run the JAR file from the command line with `java -jar desktop.jar`.

3.3 Setup instructions (from scratch)

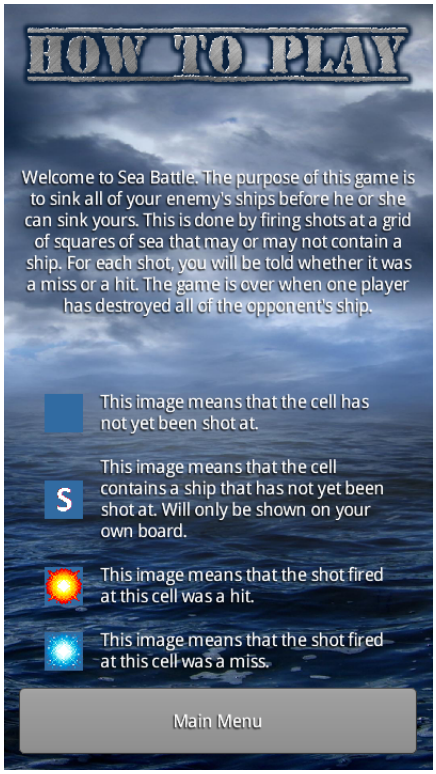
Follow the setup instructions for the client and the server provided in README.md on the GitHub page [1]. To build for the client, you will need to use IntelliJ / Android Studio with Gradle, and choose a Desktop or Android run configuration.

3.4 How to play

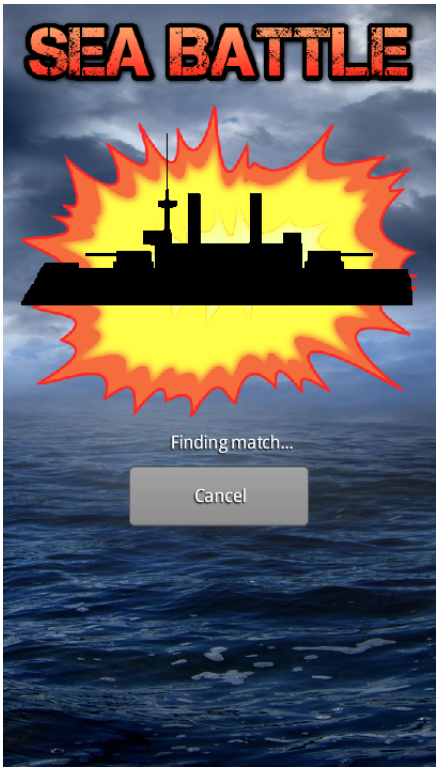
1. Run two instances of the game. You will be presented with the main menu.



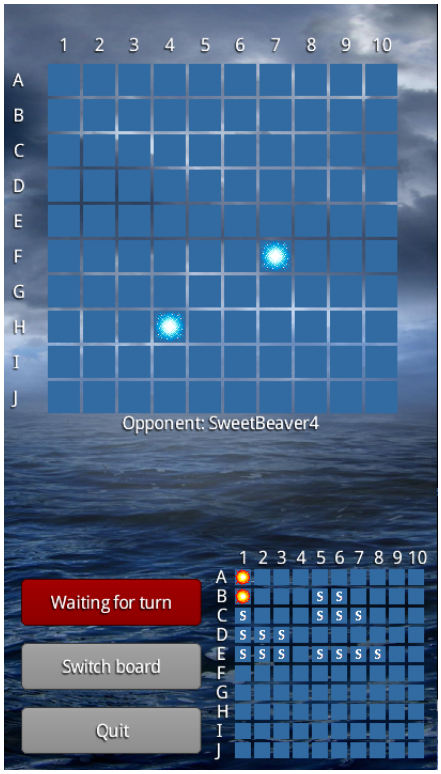
2. Click on ‘How to play’ for instructions.



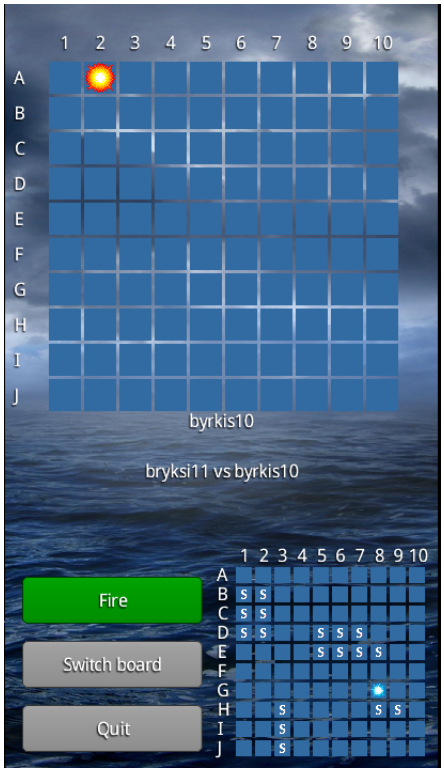
3. Choose a player name and click on ‘Find match’. Wait for another player to join.



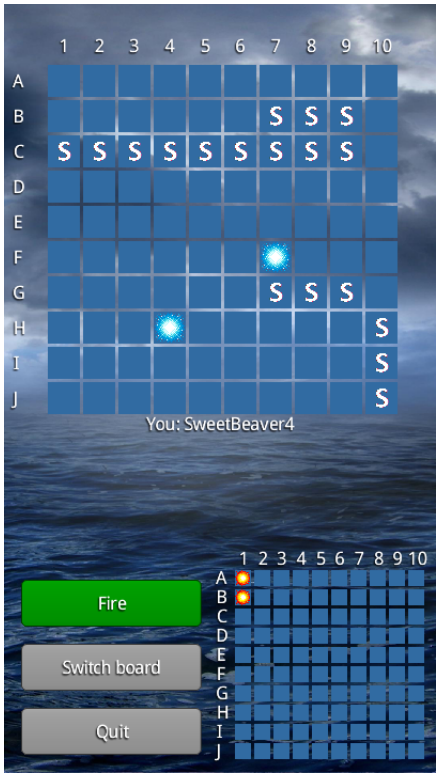
4. Wait for the opponent to fire first.



5. It is now your turn to fire. Choose a board cell and press ‘Fire’.



6. You can swap the location of the boards if you wish to see your own board in a larger frame.



7. Game over. You will be notified about whether you or your opponent won.

4 Test report

4.1 Functional requirements

Note: If not mentioned, all sub requirements are evaluated to 'Success' if the parent requirement is a success. This is done to not make an infinite amount of tables. Tests that just works will not have a comment.

FR1	Board as 10x10 matrix with ships. High priority.
Executor	Andreas Drivenes
Date	April 21
Time used	1 min
Evaluation	Success
Comment	

FR2	A ship in the game is an entity with a name and a size. Medium priority.
Executor	Andreas Drivenes
Date	April 21
Time used	1 min
Evaluation	Failure
Comment	A ship in the game is just a cell, but levels are generated from ship types on the server.

FR3	The user should be presented with a menu consisting of a "New game"-button and a "Help"-button. Low priority.
Executor	Andreas Drivenes
Date	April 21
Time used	1 min
Evaluation	Success
Comment	"Finding match" and "How to play" in the game

FR4	The user should be able to start a new game. High priority.
Executor	Andreas Drivenes
Date	April 21
Time used	1 min
Evaluation	Success
Comment	

FR5	The system should generate a grid for each player, populated with ships. The grids should have the same amount of squares filled. High priority.
Executor	Andreas Drivenes
Date	April 21
Time used	5 min
Evaluation	Success
Comment	

FR6	The user should see his own grid with ship placements, as well as the opponent's grid without ship placements (although with successful hits shown) Medium priority.
Executor	Andreas Drivenes
Date	April 21
Time used	5 min
Evaluation	Success
Comment	

FR7	The user should be able to make a move in a started game. High priority priority.
Executor	Andreas Drivenes
Date	April 21
Time used	5 min
Evaluation	Success
Comment	

FR8	The user should be able to make a move when the second player has made a move. High priority
Executor	Andreas Drivenes
Date	April 21
Time used	5 min
Evaluation	Success
Comment	

FR9	The user should be able to press a button to flick between his own grid, and the opponent's grid. See item FR1 for the grid requirements.
Executor	Andreas Drivenes
Date	April 21
Time used	1 min
Evaluation	Success
Comment	

FR10	The user should be presented with information about how the game works, before he can start a new game. Low priority.
Executor	Andreas Drivenes
Date	April 21
Time used	1 min
Evaluation	Failure/partly success
Comment	You have to press the "How to play" button before you start a new game.

FR11	The user should be able to play a local game against the computer.Low priority.
Executor	Andreas Drivenes
Date	April 21
Time used	1 min
Evaluation	Failure
Comment	Did not have the required time to implement.

4.2 Quality requirements

NOTE: The testing of adding content (modifiability scenarios) to the game that takes too long to actually test (that is they have high expected response measures) will have an estimate of the time needed as the observed measure. This estimate is based on the current code base, and an explanation of the estimate will follow in the "comment"-field of the test form. The evaluation of these scenarios will be marked as N/A, as we haven't actually gone through with them, the same goes for the time used.

MOD-1a	Change in button text/placement
Executor	Håkon M. Tørnquist
Date	22.04.15
Time used	5 mins
Evaluation	Success
Stimuli	Want changes in button text or placement
Expected Response Measure	Within an hour
Observed Response Measure	5 mins
Comment	This will only mean a change of strings in the specific view.

MOD-1b	Change background/logo/ship graphics
Executor	Håkon M. Tørnquist
Date	22.04.15
Time used	5 mins
Evaluation	Success
Stimuli	Want to change pictures or graphics in the game
Expected Response Measure	Within and hour
Observed Response Measure	5 mins
Comment	This means changing the current picture with another. One can even just replace the old picture with the new one and deploy a new version of the client.

MOD-2	Graphics instead of 'S' representing ships
Executor	Stein-Otto Svorstøl
Date	22.04.15
Time used	N/A
Evaluation	N/A
Stimuli	Want to draw ships as a whole, instead of just an S on each cell that contains a ship
Expected Response Measure	Within two weeks
Observed Response Measure	One week
Comment	The current code base and server API only marks as holding a ship, but does not separate a ship from another. This means we'll have to change how the API works, and how the client handles this feedback. It's easy to change the API, but to make sure that the local models follow the new API data structure may take some additional testing. We would need to change GameNetworkController, PlayerNetworkController, add ship models and make changes to BoardGUI and GameScreen.

MOD-3	More than one game
Executor	Stein-Otto Svorstøl
Date	22.04.15
Time used	N/A
Evaluation	N/A
Stimuli	Wants to be able to have more than one game going at a time, and have an overview of ongoing games in the application
Expected Response Measure	Within three weeks
Observed Response Measure	Two weeks
Comment	This will require that we introduce a new view on the client which can show all ongoing games. This will take some time, but if done by one of our developers which already has experience with making these views, it'll take maximum two days. The ID of the games must be handled by the API and the client, so that each client can separate its ongoing games for each other. This means that we must change the API-calls from the client and introduce a User-model which can hold all the games. We may also need to make a Game-model, and let the GameNetworkController have an instance of this, instead of being the game itself. On the server we much change all routes so that it takes this ID and the action wanted, not only the username. We also need to rewrite the

MOD-4	Communicate with opponent
Executor	Stein-Otto Svorstøl
Date	22.04.15
Time used	N/A
Evaluation	N/A
Stimuli	Wants to be able to communicate with opponent by text (chat)
Expected Response Measure	Within four weeks
Observed Response Measure	Three weeks
Comment	We think that this will not be too difficult to implement, as sending raw text from one client to another is a simple task, as we are already sending plenty of other information. Chat screen will have to be added on the client. To do this, we would need to add a POST and GET routes for sending and receiving messages, add a user model on the server, and extend PlayerNetworkController with chat requests. The Player object on the client would need a list of messages.

MOD-5	Ships left on opponents board indicator
Executor	Stein-Otto Svorstøl
Date	22.04.15
Time used	N/A
Evaluation	N/A
Stimuli	Wants to be able to see how many ships (or cells containing ships) the opponent have left
Expected Response Measure	Within three days
Observed Response Measure	1 day
Comment	Given that MOD-2 is already implemented so that we're able to separate ships from each other, this will only mean to have a count on how many complete ships are not hit, and put in on the screen. To add a text element to the screen takes a minimal amount of time. If MOD-2 is not completed we can have a count of how many ship-cells are left. This will have the same estimates.

MOD-6	Custom ship placement
Executor	Andreas Drivernes
Date	22.04.15
Time used	N/A
Evaluation	N/A
Stimuli	Wants to be able to place own ships
Expected Response Measure	Within two months
Observed Response Measure	One month
Comment	This is a big feature, even given that MOD-2 is already implemented (which introduces ship entities). If MOD-2 is implemented we'll first need to introuce a new view on the client which lets the player place ships. This also means we need graphics for the different ships. Furthermore, we need to make a new route and the necessary logic on the server, so that the client can send the sepcified ship location before the new game is started. These placemnet also needs to be distributed to the other player, which in turn means we need to have another route so the client can pull and see if the opponents board is ready, and one is ready to play.

AVB-1	Notify operations team if server is down.
Executor	Stein-Otto Svorstøl
Date	22.04.15
Time used	N/A
Evaluation	N/A
Stimuli	Unresponsive server
Expected Response Measure	Maximum five minutes of downtime
Observed Response Measure	N/A
Comment	This hopefully just requires a restart of the server. As of 22.04.15 we haven't had any problems with the server, so we haven't been able to test this. Addons on Heroku for notificaiton in case of problems or downtime are activated.

SEC-1	Break into excisting game
Executor	Stein-Otto Svorstøl
Date	22.04.15
Time used	20 minutes
Evaluation	Failure
Stimuli	Wants to break into a game already containing two players
Expected Response Measure	Not throw players out of games 98% of the time.
Observed Response Measure	N/A
Comment	Because the username is the only identifying factor, all you have to do to break into an already started game, is to write in the correct username. This was a simplifying design decision we made to save time. In the case you create a new game (you have an unique username) you will not disurb any ongoing games 100% of the time.

PERF-1	Touching screen
Executor	Håkon M. Tørnquist
Date	22.04.15
Time used	10 seconds
Evaluation	Success
Stimuli	Touching the screen in the mobile application gives the user visual feedback
Expected Response Measure	Within two seconds
Observed Response Measure	<10ms
Comment	Every clickable part of the game responds instantly with some kind of visual feedback to the user (either by changing screens, marking a cell or firing shots)

PERF-2	Make a move
Executor	Stein-Otto Svorstøl
Date	22.04.15
Time used	5 minutes
Evaluation	Failed
Stimuli	Making a move in the game and give visual feedback to opponent about move made
Expected Response Measure	Maximum 1 second from move is made and sent by player 1, to player 2 is informed about the move.
Observed Response Measure	$<$ 5 seconds
Comment	As the server updates every 5 seconds, the maximal amount of time it takes before the opponent sees your move is 5 seconds. We've not guarnateed a max 1 second wait, and it should be 2.5 seconds on average.

5 Relationship with the architecture

Although we initially planned to only have views on the client. and keep all controllers and models on the server, we found it necessary to have network-controllers and local replica models client side as well, and as you can see our final class diagram in figure 1, is structured a bit differently from the original, which can be seen in figure 4. One cal also notice that filenames are different from what we planned.

However, we realized this early in the process, just after we had handed in the architectural report and before we started started writing any code, so even if we have discovered it earlier, it wouldn't have made a significant difference. It was not really mentioned in the ATAM evaluation, as we gave notice to our evaluation team that we probably would have to do something like that.

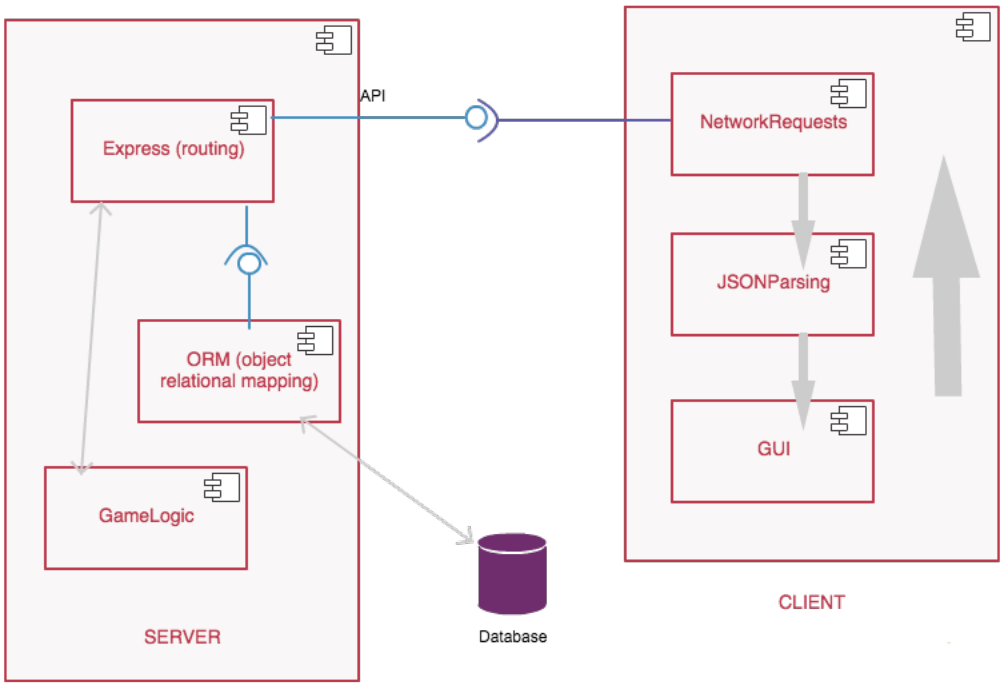


Figure 3: Our component diagram form the architectural documentation.

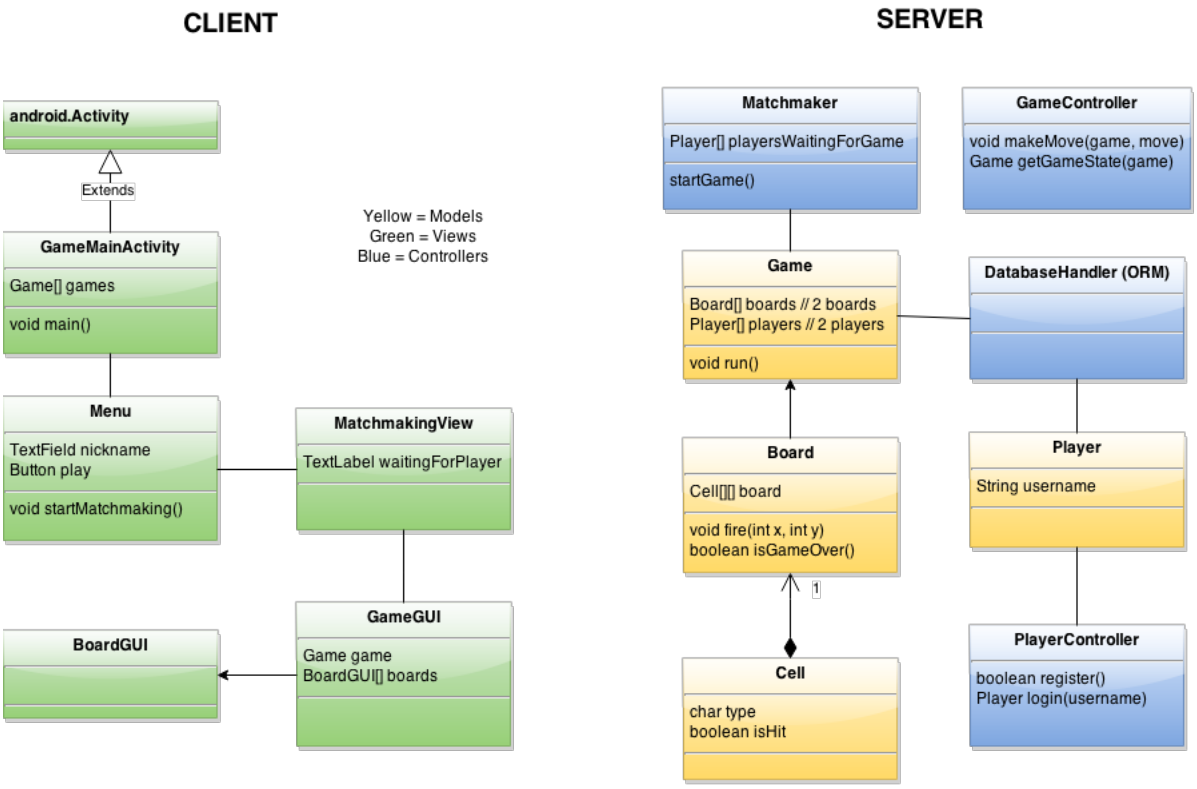


Figure 4: The class diagram from the architectural documentation.

One can also notice the inconsistencies between the final class diagrams on the client, and the planned component diagram in figure 3, especially when it comes to data flow. Where we originally thought that the component handling the network requests would be a simple entity only communicating with a JSON-component, we that in practice this became a lot more complicated. The models are the ones that uses JSON-parser for the most part, and we had to make two controllers to handle the network requests, which also has to handle some view- and model-related tasks.

We could have made a more detailed plan for the the network logic, models and the relationship to the GUI more detailed. JAVA is an object-oriented language, and one needs to think in terms of objects all the time, so it makes sense to have Board, Cell and Player models.

On the server, we planned to have a Cell object, but this was an unnecessary complication. A cell is only a char in a board string in the database, and we just create a simple cell with two booleans when we want it. This is done directly in the JSON data object that is to be sent to the client.

6 Problems, issues and points learned

When one starts to develop a game, one can also assume there will be some issues. There may be technical issues encountered in our code that needs to be sorted out, things that takes a lot of time to learn, issues in the group, with the reports to be written or other things that affect the project.

The first problem was with libgdx, that none of us had any experience with it nor with Android except the introductory exercise. One thing is to know how things may be done, another thing is to know if the way you're doing it is the "recommended" way. Without any experience this is almost impossible to know.

This leads us to another problem, which is that when the whole group has a limited grasp of the technology at hand, it's hard to go through with code reviews, or even work on code others have written. We had a few disagreements about where code should go, or how code worked. As one of the main QA attributes of the project is modifiability, we also had these discussions so that we'd be sure that it made sense to place the code where we did, so any other new developers had an easy job of maneuvering the project and the code base. Another small issue that relates to this was inexperience with git and GitHub among some of the groups members. It took some time for this to be discovered and to be dealt with, but when we finally did it made the development go faster and we communicated a lot better.

Another issue we've encountered is that it's hard to agree on a procedure surrounding a project. It may be easy to establish a routine, but it's hard to enforce it throughout the lifetime of the project.

We've also learned a great deal of things. First and foremost we've experienced how things seem much easier when you're planning them, compared to when you're actually doing them. This is also shows how important it is to plan out the architecture and how the code should be structured in advance, because when you're a part of a development team, one cannot just do as one wants all the time. Communication is key to a great product, and in this case we did communicate as well as we could all the time. We also experienced how having great tools, like Trello, Slack and GitHub, is worth nothing if the team does not know how to use it, doesn't want to or simply forgets to do so.

Of course we also learned a great deal about Android, libgdx, NodeJS and PostgreSQL, the technologies we used for development. It's one thing to read about these things, but a completely other thing to actually use it to implement something.

References

[1] *Project source repository (GitHub)*. Apr. 20, 2015. URL: https://github.com/andereld/progark.

[2] *Sequelize (ORM)*. Apr. 22, 2015. URL: http://docs.sequelizejs.com/en/latest/.