Norwegian University of Science and Technology

Requirement and Architecture Phase
TDT4240

GRUPPE A13:
Stefan Bui
Jim Frode Hoff
Christian Tverås
Jonas Foyn Therkelsen
Julie Johnsen Kirkhus
Erik Reimer

February 20, 2014

Innhold

1. Introduction

In this project we are supposed to choose between making a functioning multiplayer game for either Android, Iphone or XNA, or make a robot simulator.

The documentation phase is intended for planning of the chosen architecture design for the development of the game for this project.

1.1. Project description

Our project is a curling game project for Android. The game shall follow simple curling rules and will be a turn-based multiplayer game.

The game will have a simple menu screen with the options to choose between tutorial, play game and exit. The game itself will utilize the touch function on android devices, where you will be able to send the ball with a given speed and manipulate the speed with swiping near the ball, either to decrease the friction for it to go longer or to add or subtract spin. to give the player a better feeling of how well he/she is doing the game when played will have a minimap to give a better overall view of the game.

The tutorial will guide the player through the rules of curling and how the physics inn the game works.

2. Architectural Description

2.1. Architectural Drivers/Architectural Significant Requirements (ASRs)

The main drivers (functional, quality and business requirements) that affect architecture and the system mostly.

ASRs consists of all the requirements that affects the architectural design in any way. Both functional and nonfunctional requirements can be ASRs as long as it has some impact on the design.

Functional

- The game will have a start menu.
- The game will have a pause function.
- The game will have a tutorial function.
- The device have to support touch functions.

Nonfunctional

- The game have to run on android.
- Logic will be written in java.

2.1.1. Quality Attributes

Modifiability

- Focus on modifiability is required.
- Use abstract classes where it compliments the code.
- Use mvc design as long as it will improve the modifiability.

Usability

- Use the lowest possible api so that the game can be played on as many devices as possible.
- Focus of usability for the end-user is required.

Others

- The game has to be a multiplayer game.

2.2. Stakeholders and concerns

The stakeholders in this project are identified as developers and end user

2.2.1. End user

The end users will be the players of the game. Their concerns will be to understand the game and to enjoy playing it.

2.2.2. Developers

The developers will be group A13. Their concerns will be to finish the game, make it work as planned. Test the code regularly. The code should be easy to understand and easy to modify.

2.3. Selection of Architectural Views (Viewpoint)

Logic view; we choose a logic view over the overall structure of the game. This is to clarify for the developers how we want to structure the logic. The notation used are as the following: - Box: used to show the different main states. - box with lines: substates of the main state. - Rectangles:used to show different options to choose from. - Arrows: indicates which state you will be taken to next.

For the process view we chose to see closer on the flow of the game when it is played. This view is also made with the thought to increases the developers understanding of how the game itself will be played. the notation is as the following.

- circle box: used to indicate the start node
- box: used to show the different process.
- diamond:used to show different options to choose from.
- arrows: indicates which state you will be taken to next or answers to questions.

The develop view shows how the classes are related and where we will be able to make abstractions when many classes share communalities. Since the only stakeholders are the developers (us) its no big surprise that the develop view is made to make it easier for the developers. The notation is as the following.

- Box: used to indicate classes/groups to make an overview of the class structure and tree
- diamond: used to indicate the which classes are in relation to one another.
- arrows: used to show which classe belongs to which group.

2.4. Architectural Tactics

One of our quality requirements goes directly at reducing coupling so, the tactics we plan on useing are the following; encapsulate, restrict dependencies, refactor and use abstract common services. These tactics work to increase the modifiability which is one of our QA.

To ensure that the usability does not fall behind we plan on giving the player the option to cancel a game and give the player the option to pause.

2.5. Architectural and Design Patterns

When it comes to design patterns we plan on using mvc to increase the modifiability of our game.

2.6. Views

Document architecture using your selected views (at least logical, process and development) through diagrams and supporting/explaining text.

2.7. Logic view

The logic view provide an overview over the requirements over the principle of the game process. This figure show a decomposition of the logic of the game process and the relationship between these elements.

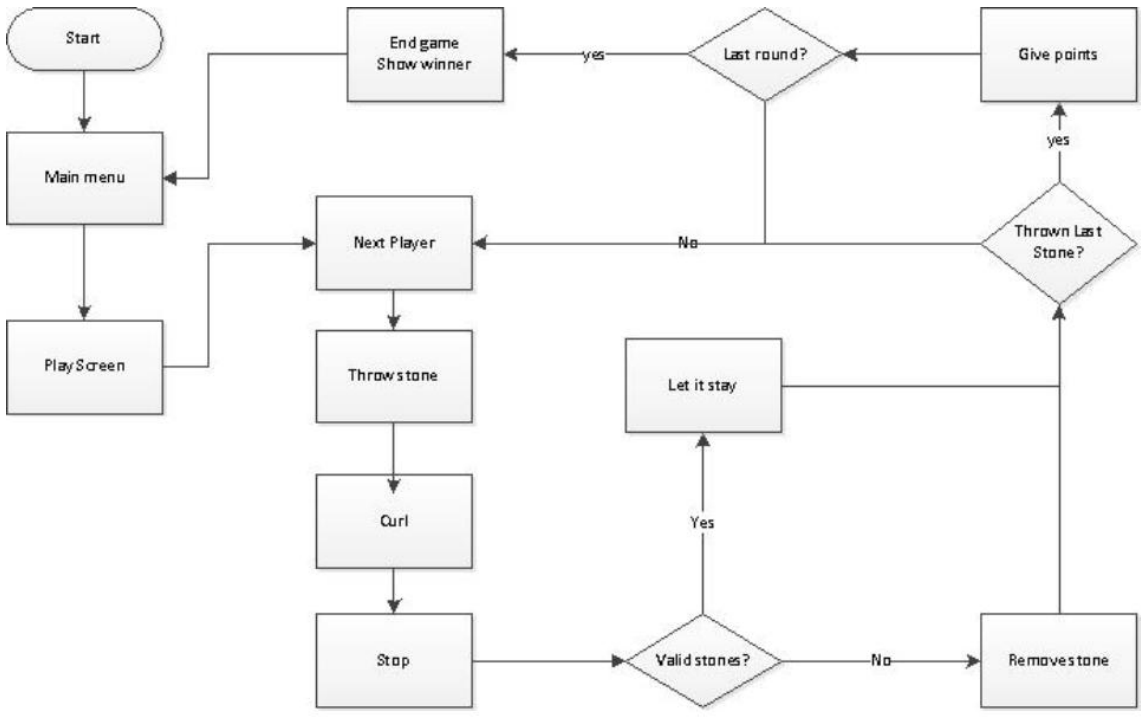

Figure 2.1: Logic view

2.8. Process view

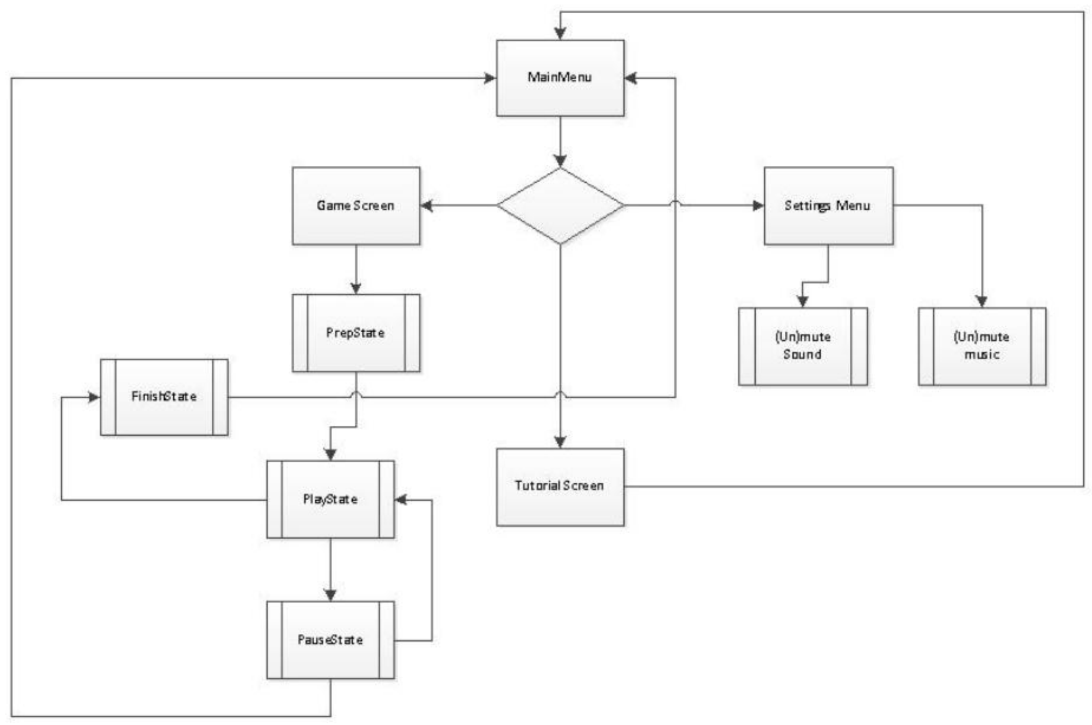

Figure 2.2: Process view

2.9. Development view

With the CurlingGame project at the base, it extends into three packages - model, viewactivities and the gamestate. Our model package takes care the model objects, as the curlingstone and the track, and is related to the Game class which draws and updates models. Our gamestate package holds all the viewcontrollers of the project. GameSettings is related to the game to change final variables such as how many balls to play with. Pause/Settings is an activity that pauses the game and brings up a settings-activity on top of the game activity. Finally, the Game class runs, draws, updates, and controls the entire game and other classes. Our project also utilizes the sheep API.

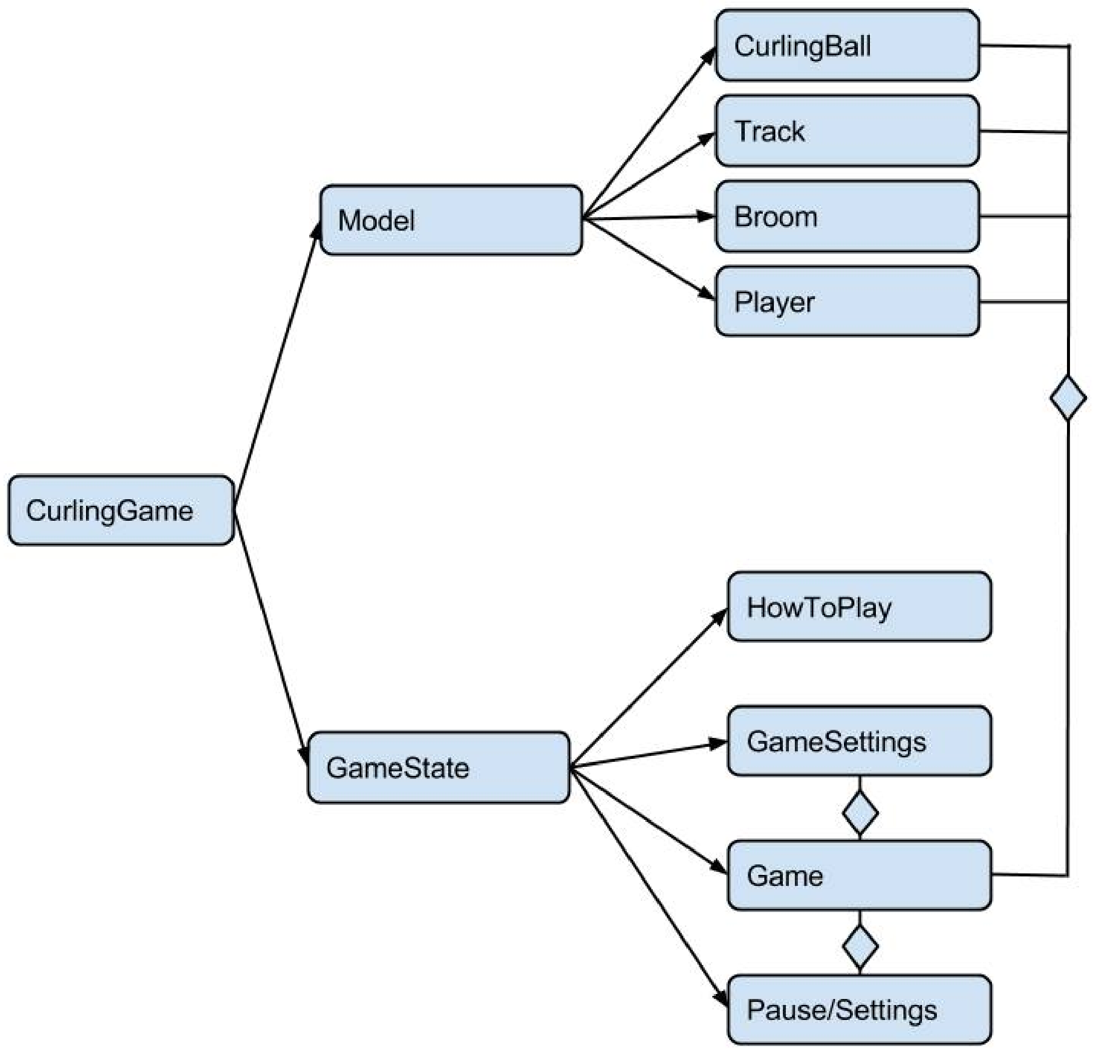

Figure 2.3: Development view

2.10. Consistency Among Views

Describe inconsistencies among the views in your architecture.

2.11. Architectural Rationale

Here you will explain why you have chosen the architecture you have chosen. This part should motivate for why the architecture will work and fulfill the quality, business and the functional requirements.

2.12. Issues

2.13. Changes

References

[1] **Book**
Len Bass, Paul Clements, Rick Kazman. *Software Architecture in Practice*– Addison-Wesley. 3rd edition, 2012

[2] **Article**
Phillipe B. Kruchten. "The 4+1 View Model of architecture", IEEE Software Magazine, Volume 12, Issue 6, 1995.