Norwegian University of Science and Technology

Requirement and Architecture Phase Architectual document

TDT4240

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

February 24, 2014

Innhold

1. Introduction

For this project we are supposed to choose between making a 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 should follow official curling rules and will be a turn-based multiplayer game.

The game will have a simple menu screen with the options to choose between play, tutorial and settings. The game itself will utilize the touch function on android devices, where the end-user will be able to slide the curling stone with a given speed and manipulate the speed by touch. By swiping your finger near the stone, it will either decrease the friction for it to go longer or to spin it, depending where you touch. The minimap is also used to give a better overall view of the game.

The tutorial will guide the player through the rules of curling and how the physics in 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 most.

ASRs consists of all the requirements that affect 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 shall have a start menu.
- the game shall have a pause function.
- the game shall have a tutorial function.
- the device has to support touch functions.

Nonfunctional

- The game has to run on android.
- Logic shall 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 a small-scare API so that the game can be played on as many devices as possible.
- focus on usability for the end-user is required.

Others

- The game shall be a multiplayer game.

2.2. Stakeholders and concerns

The stakeholders in this project are identified as developers and end users.

2.2.1. End user

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

2.2.2. Developers

The developers will be group A13. Their concerns are development of the game, making it function as planned and testing the code regularly. The code should be easy to understand and modify. In addition to this, the developers want to deliver a system in accordance with the documentation.

2.3. Selection of Architectural Views (Viewpoint)

Logic view; we've chosen a logic view of the overall structure of the game. This is to clarify to the developers how we want to structure the logic. The notation used are as 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've chosen to look closer at the flow of the game when it is played. This view is made with the thought to increase the developers understanding of how the game will be played. The notation is as follows.

- 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 more than one class share communalities. Since the only stakeholders are the developers, it's no big surprise that the development 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 classes belongs to which group.

2.4. Architectural Tactics

One of our quality requirements directly aims to reducing coupling. The tactics we plan on using are the following; encapsulation, restrict dependencies, refactor and use abstract common services. These tactics work to increase the modifiability which is one of our quality attributes.

To ensure sufficient usability we wish to give the player the option to cancel a game and the option to pause. This will increase the user support initiative.

2.5. Architectural and Design Patterns

Our choice of architectural pattern for our game project will be a model/view/controller-design (MVC) to increase the modifiability of the game. We don't plan on using a specific architectural pattern for usability.

To increase the modifiability of our game we plan on using the abstract factory pattern where it seems to fit. For this purpose, a template pattern would be desirable. Since we plan on using the MVC architectural pattern it will be obvious to implement an observer pattern as well. When it comes to increasing usability, there are few patterns to choose from. Consequently we intend to design the graphical interface such that the user will easily and intuitively achieve our quality requirement of learning the game in less than 10 minutes.

2.6. Views

2.7. Logic view

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

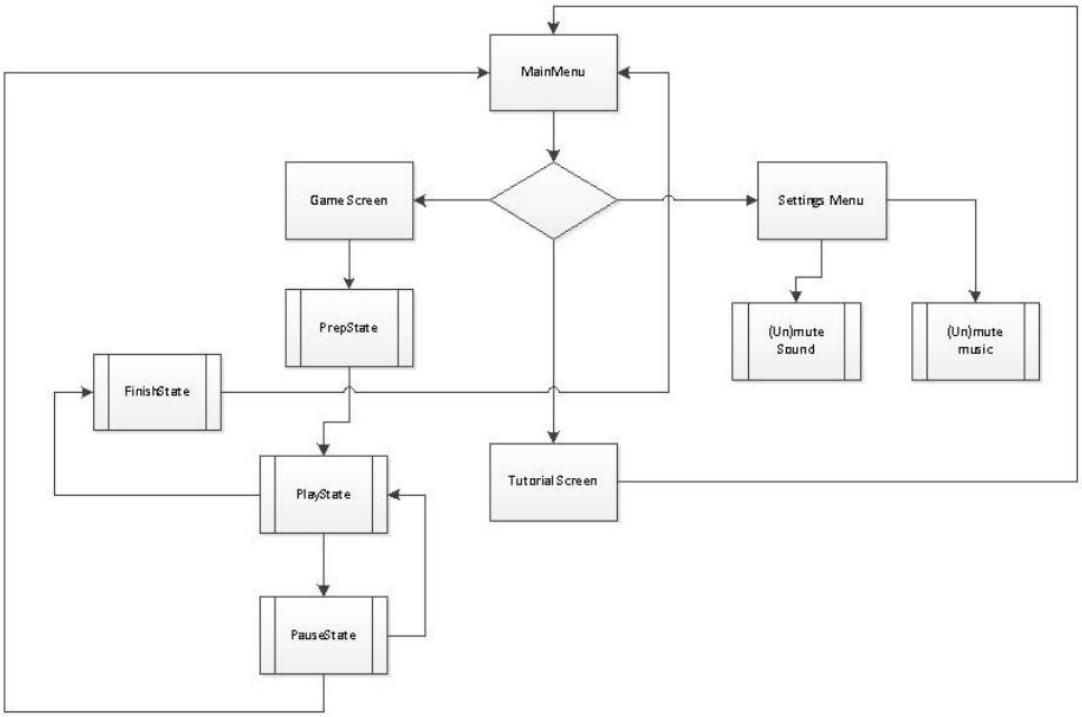

Figure 2.1: Logic view

2.8. Process view

This activity diagram show the flow between the different action in which state of the game process. Note that this is only a simplified diagram of the playing part of the application. This make it easier to understand the system integrity. The game application start at the main menu. There you will be presented by multiple choice: Play, Tutorial and Settings. Hence by choosing Play you will see that you get to a playscreen for configuration of the gameplay, before starting the game. By following the action of each state you will get a general process view of the game application.

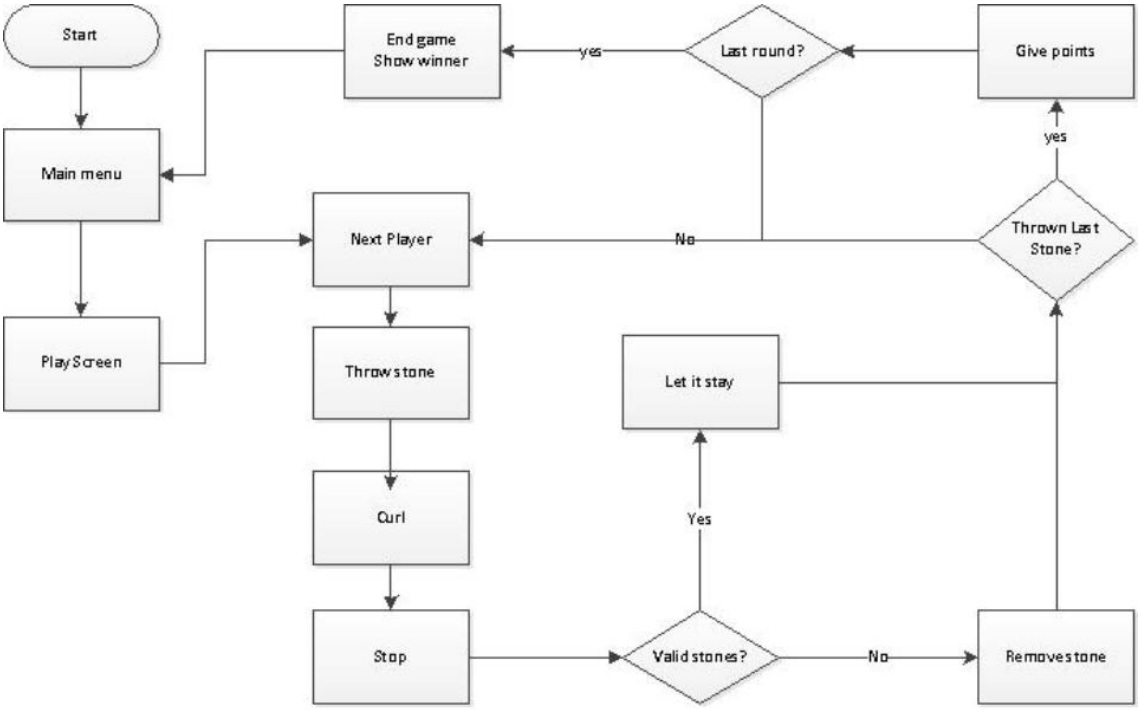

Figure 2.2: Process view

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 utilizes the Sheep API.

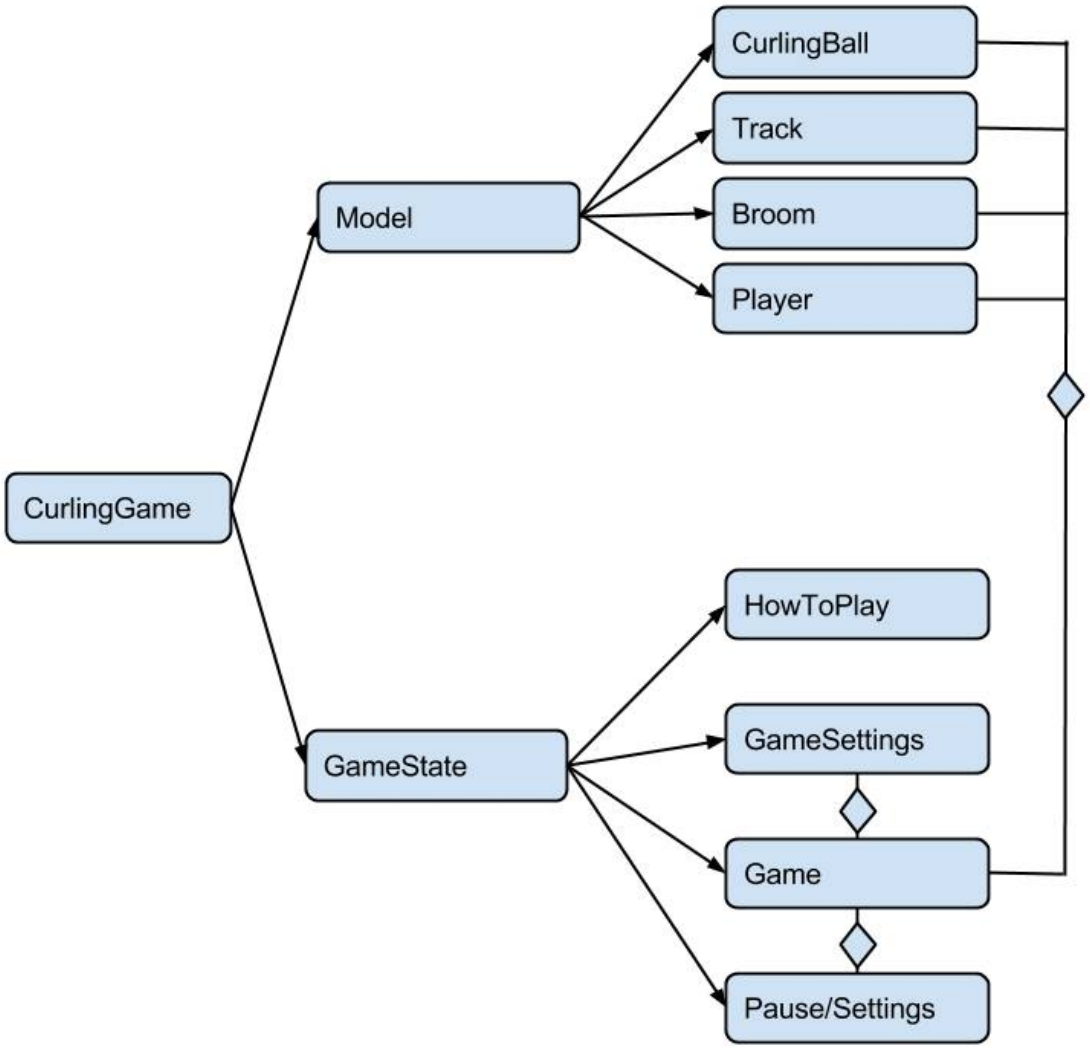

Figure 2.3: Development view

2.10. Consistency Among Views

Describe inconsistencies among the views in your architecture.
Inconsistency in notation.

2.11. Architectural Rationale

Our architecture supports the thought of the simpler something is, the better. Thus, our architecture is simple - as is the game we're developing. It's also consistent with other similar game applications out on the market today, with simple interface views which are intuitive and similar to other android games, and help new users understand how to use the application from the get-go. The patterns and tactics we have chosen implies that the system will be highly modifiable and to some degree have high usability.

2.12. Issues

Coming up with an architecture for a small game project doesn't bring along the biggest issues. The architecture has been made simple for higher modifiability and usability.

Experience is our greatest issue. The initial architecture we've chosen for our project seems intuitive and reasonable for a project this size and purpose, but if it will actually be a solid architecture for a game project like ours we have yet to find out. In addition to this, our group isn't too familiar with curling as a game and it's rules, but considering if some of us were more experienced with the game we might have chosen to create a different architecture.

Initially our group was interested in a networked multiplayer game, but this proved to demand a more advanced architecture which resulted in the group choosing a turn-based multiplayer game. The main arguments was that we wanted to make something that we knew we would be able to pull off so that the main focus could be on learning how to document and make the best architecture possible.

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.