TDT4120
Software Architecture

Super Bros
Requirements Document

Group X2

Milos Zlatkovic
Dimitry Kongevold
Emil Grunt
Nicolay Thafvelin
Julius Buset Asplin
Håvard Kindem

COTS: XNA

Primary quality attribute:
Modifiability
Secondary quality attribute:
Usability

Contents

1 Introduction

This document is a description of the product delivered by Team X2 in the course TDT4240 Software Architecture and contains the architectural decisions related to the quality attributes we have given in the requirements document. We will describe the architectural drivers, stakeholders, architectural views, quality tactics, and the architectural patterns we have chosen. The chapter architectural drivers will discuss the architectural drivers of the project and stakeholders will discuss the stakeholders. In chapter 4 we will talk about our architectural viewpoints, chapter 5 our tactics for optaining the modifiability, testability, performance, portability and performance. Chapter 6 will describe what architectural patterns we plan to use and chapter 7 will describe our MVC, Scenario view and state machine.

2 Architectural drivers

The biggest Architectural Driver in this current project is lack of time. This will affect your architecture to make it more compact and easier to implement.

2.1 Functional Requirements

We need to have a functional game that has multiple maps, characters and power ups. It will need to have fighting implemented and use some basic physics.

2.2 Quality Attributes Requirement

The game has to run smoothly with no delay between the unser input and actions on the screen. It has to be well tested so that it can be played without crashes.

2.3 Technical Constrainsts

Our techical constraints are the performance of the PC and XBox. We are also constrained to using C# and XNA.

2.4 Business Constraints

As we are not releasing this project, we have no notable business constraints.

2.5 Modify-ability

We need to have full Modify-ability regarding new characters, stages, moves and etc. This means that we want to be able to change and add characters, stages and moves and etc without having to change parts of the already written code and without having to concerned about how we write the new code.

Because of the multiple people are working on this project, the modifiability is also and Architectural Driver, the implementation should be âĂIJseparableâĂIJ so we could preside the implementation in a parallel pattern. This also a product of previous driver, short time.

2.6 Testability

We need to be able to test the game all the way through the production so that we can make the right touches to the game so it will be good looking, fun and smooth to play. It is also very crucial for balancing of the characters so that even though they have different moves, they are equally good.

2.7 Performance

The code must be efficient enough so that there wont be any lag or delay while the game is played. We must optimise the calculations and logic and not do more operations than necessary.

2.8 Portability

The game should be playable on both Xbox 360 and computer.

2.9 Usability

The game should be very easy both to start and to play. Their should be easy to find help on which button does what and so on. The objective of the game is quite obvious and therefor needs no introduction.

3 Stakeholders

3.1 Development team

- Want a fun playable game as return for their time-investment.
- Need a good grade on the project, and therefore a good architectural structure.

3.2 Course staff

- Wants us to learn about and test/use his advises on architectural structures and program design.
- Wants us to get a good grade.

3.3 ATAM Group

- Wants a well described project to easier perform the ATAM exercise.

3.4 End user

- Wants an easy learned, easy played and fun game.

The main concern is the balance between the number of features and the actual playability of the game. This is also the reason why we want it easy to modify. We will make the standards and the basics of the game finished before adding all the moves and features. We will make it so that adding moves and characters almost only need parameter attributes and pictures to be ready for use.

4 Selection of Architectural Viewpoint

4.1 Logical view

Logical view shows the Model View Controller architectural pattern. where classes have different functionality. Based on the pattern. the target audience is other architects and stakeholders.

4.2 Development view

The development view describes the static organization of the software in its development environment. You would usually present this in a diagram that presents the different modules/layers of the system.

4.3 Scenario view

Scenario view gives a better understanding over the run time processes and their communications with each other during well known actions in the game. As well as the sequence of the MainGame class and other classes. Target audience is stakeholders as implementation team.

4.4 State machine

The last view is a state machine of the MainGame Window, it explains how the user will navigate himself through the game. Audience is stakeholders.

5 Architectural Tactics

5.1 Modifiability

To make this project easy to modify we are planning to use Model View Controller architectural pattern, and implement the characters and their moves with only parameter attributes. This makes it easy to change/add characters, maps or power ups. Model View Controller will also provide grouped implementation, as Character-Controller will only affect the character model. as well as all needed resources needed for Draw method will be allocated under model. were itâĂŹs easy to check for consistences.

5.2 Testability

We are going to start out very simple, so that the testing, as in playing the game, can start early. Since we are doing it this way, we wont need the computer to perform actual tests for the most part. We can test it simply by building the project and start testing it manually.

To test the individual methods and the logic itself we will try to separate logic from data and GUI, which will make testing of them easier.

5.3 Performance

We will do performance testing to weed out resource hogs, and find out which methods use the most computational power and alter or remove them.

5.4 Portability

We will only use classes which are supported by Xbox 360, and surround any platform specific code with compiler pragma to avoid duplication of code.

6 Architectural Patterns

These are the architectural patterns we're planing to use

6.1 MVC - Model, View and Controller

MVC is an architectural patterns which has three components, the models which holds the data, views that generate views based on this data and controllers which alters the data. These three parts are communicating with eachother and combined makes up a program. We chose Model View Controller architectural pattern because it gave us the best modifiability as well as it separates code into small fragments that are easy to test and distribute the work load through the group.

6.2 SOA - Service Oriented Architecture

The SOA is an architecture which is based on having multiple services(classes used ad services in our case), these are then communicating with eachother with requests and responses. Closely knit with MVC. The Game class in the XNA Framework provides an infrastructure for registering and providing application level services. By grouping related functionality into classes and offering them as âĂIJservicesâĂİ specified through an interface, the functionality can be provided without compromising modifiability.

6.3 Observer pattern

The observer pattern architecture consists of an observer and an observable object. When something changes in the observable object, it will notify the observer so that it can handle the change. The communication within the game will primarily be through firing of events and registration of callbacks. This will improve the modifiability a lot. We also will use state pattern in the characters and the map class.

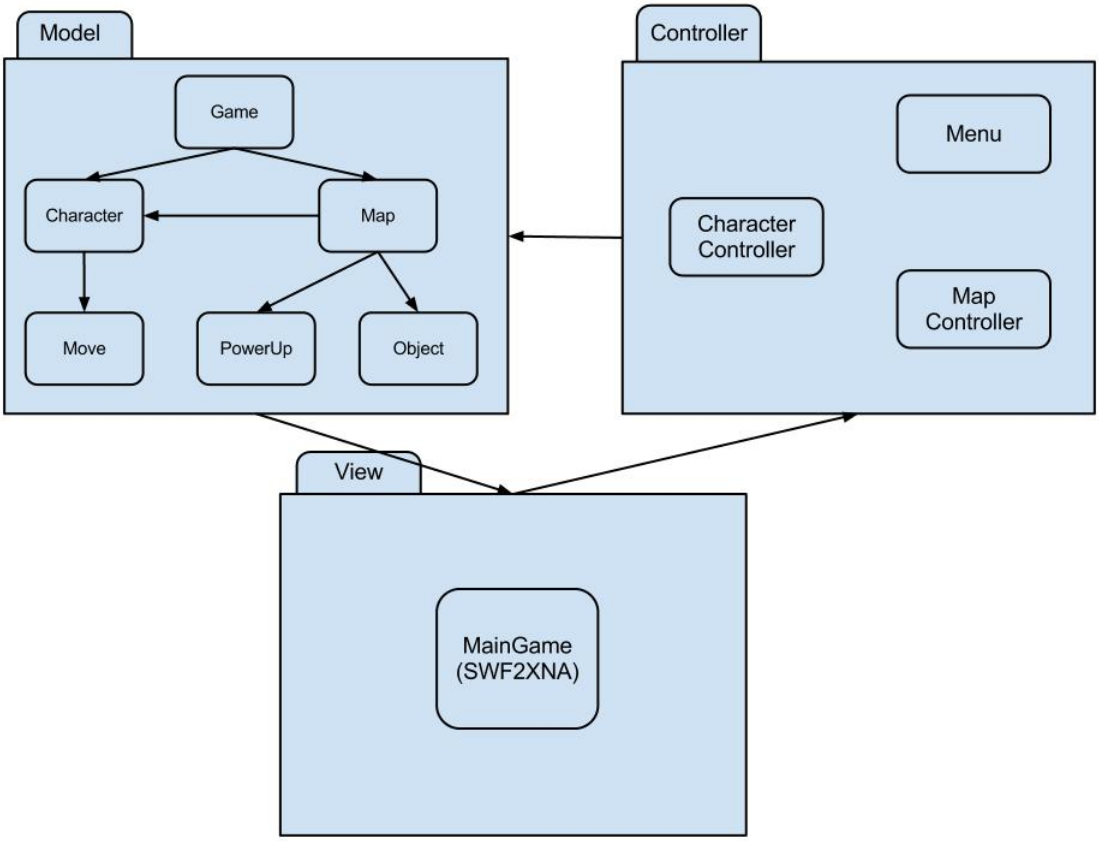
Views

7

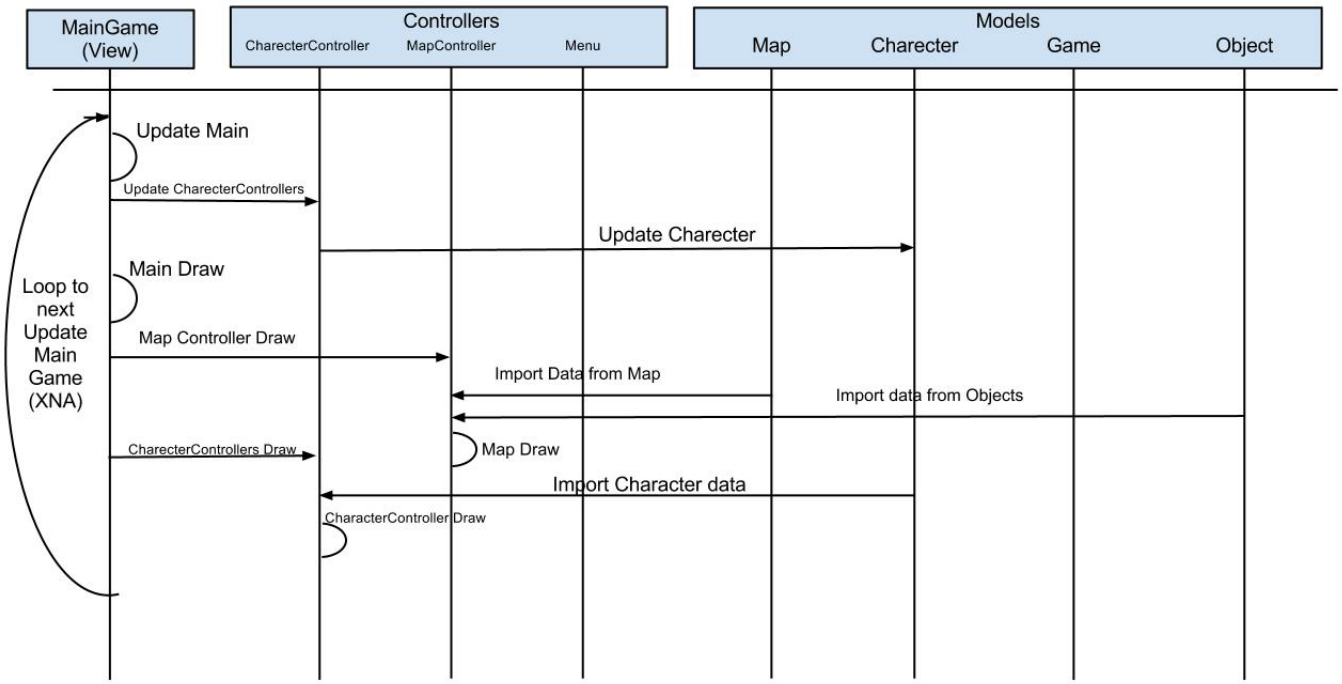
*See chapter 4 for initial descriptions.

7.1 Logical Views

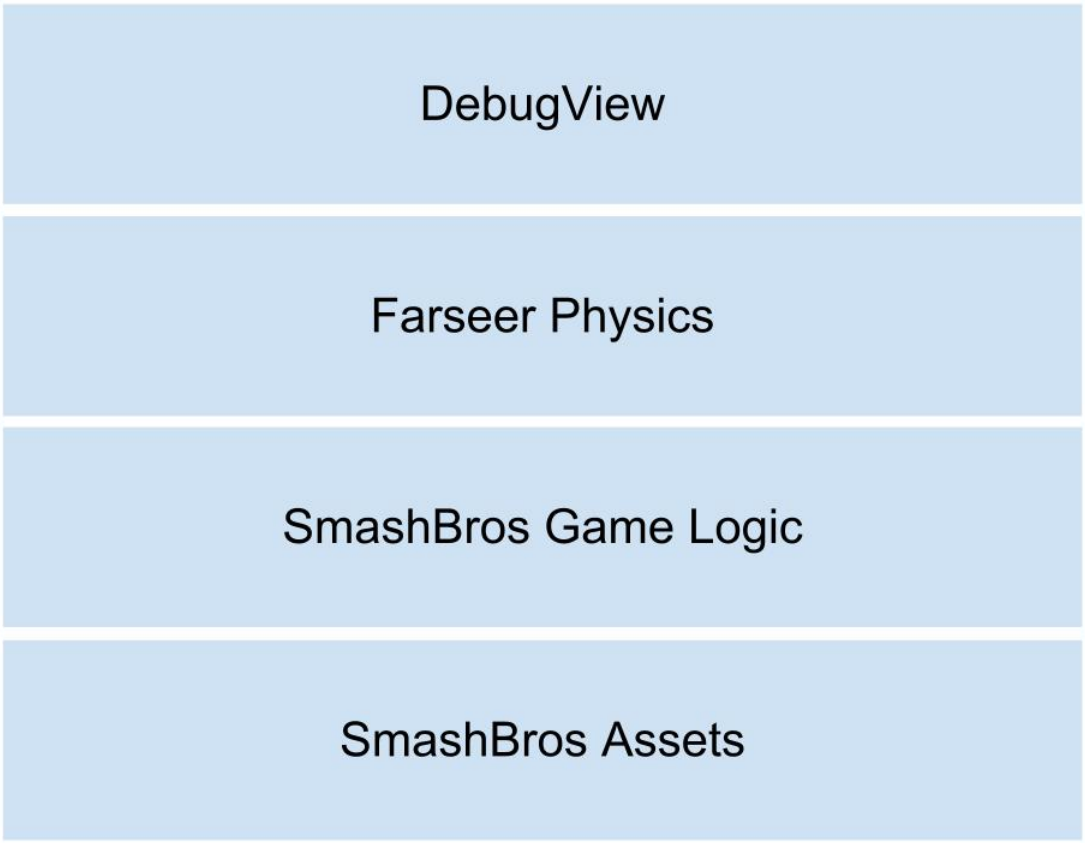
This diagram show the logical view of our game. It should be noted that this does not cover nearly all classes, this is simplified and the classes are grouped together to create general categories to avoid confusion.



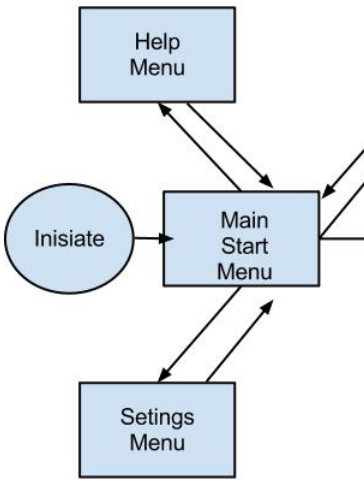
7.2 Scenario view



7.3 Development view



7.4 State machine



This is a diagram of our game as a state machine, it covers all the general scenes.

7.5 Consistency among views

The views are coherent with each other. The logical view excludes the libraries(DebugView and Farseer Physics) and game assets as well as the basic system functions, as they are irrelevant to this view. Scenario view excludes the views of the MVC as those has no game functionality other than viewing. The state machine shows the scenes of the game.

8 Summary

8.1 Architectural Rationale

We chose MVC as is nicely splits up the game in different categories, it makes both the deveolopment and code review easier as the classes are categorized by functionality. SOA was chosen to increase the modifiability of the game so that it could easily be expanded at a later time by making the classes independant on each other, this knits nicely with observer pattern as it is basically the same thing, only event driven. State machine was selected to nicely split ut the different game states (character selection, options, map selection and game).

8.2 Issues

We found that our architectural patterns worked nicely together, although it was difficult keeping the programming consistent as SOA and OP are very simlar.
Had issues keeping the functionality of classes within either models, views or controllers due to time issues.

9 References

This is the references we used to complete this document

9.1 Books

- "Software Architecture in Practice, Second Edition", Len Bass, Paul Clements, Rick Kazman, Addision-Wesley, 2003, ISBN 0-321-15495-9
- "Game Architecture and Design - A New Edition", Andrew Rollings and Dave Morris

9.2 Webpages

- http://en.wikipedia.org/wiki/SOA
- http://en.wikipedia.org/wiki/ModelâĂŞviewâĂŞcontroller
- http://en.wikipedia.org/wiki/Observer_pattern

10 Changes

29.04.2012
Updated based on feedback

30.04.2012
Added more content to complete the document.