TDT4240 – Software Architecture

Architectural Document

Virion

Group X3

Dawn Alphonse Jose, Eirik Fosse, Filip Egge, Jørgen Foss Eri, Kristian Sundklakk, Tobias Linkjendal

COTS: XNA

Primary attribute: Modifiability

Secondary attribute: Usability

Table of Contents

1. Introduction

1.1. Project description

The main goal in this project is to make a game using the XNA Framework. It has to be a multiplayer game, either local or network, based on a concept described below. The game must be designed according to a specified software architecture.

1.2. Virion

The game we are making will be a 2D top-down co-operation game where the goal is to infect cells in a body, represented by multiple levels, using a virus. The players, controlling one or more viruses each, will have to avoid the body's immune system, and infect all the normal cells. When a cell dies it releases protein, which the viruses can use between levels to evolve into a more efficient form.

2. Architecturally Significant Requirements

Our group are creating a fun and challenging co-operation game that you can play on your computer. As we are coding in XNA we might release the game to the Windows Store if we are pleased with the gameplay and functionality.

2.1. Functional Requirements

We are going to create a game with multiple maps and characters. It should be easy for us to change colours, looks and maps, as well as adding more functionality.

2.2. Quality Attributes Requirements

The game has to run smoothly on a PC with no delay between user input and events on the screen. It is important that we are thorough in our testing, to avoid bugs and crashes. We also wish to make it easy to start the game, as well as understanding and playing.

2.3. Dynamic environment

A PC can have many different sizes on the screen, which makes it important that we are making a game that can be played on all those screens. The game has to be dynamic and resize to the users preference.

2.4. Time Frame

As we have a short time frame for this project, we must have a base system that is quick, yet solid, to build. By the end of this project our game must be in a deliverable state.

2.5. Inexperience among developers

The team consists of members with different backgrounds. Some are quite new to XNA and C#, and although we will find solutions to problems, these might not be the best solutions. It will be important that we get an early working base system and keep adding functionality to this.

2.6. Portability

The game should be possible to play on all devices with keyboard input that run the Windows 8 and 7 operating systems.

3. Stakeholders and Concerns

Stakeholder	Concern
Developer	Maintainability and Modifiability are important factors for the developers. Without these, progress will be slow and changes will easily introduce bugs. Developers also want a fun game to play themselves.
Player	Usability is the primary concern for the game, as it should be with all games. If the player is not able to understand and play the game, then the game has low usability.
TDT4240 Staff	Readability in the documentation and code is critical for a successful project and good grade.

4. Selection of Architectural Views (Viewpoint)

We will be using the “4+1 View Model” created by *Philippe Kruchten* to describe the architecture of our system. It proposes four views plus scenarios. The views are: Logical, Process, Development, and Physical.

4.1. Logic View

Purpose:

The logical view describes the functionality of the system by breaking down requirements into classes and representing them, and their relations, through class and sequence diagrams.

Stakeholders:

Developer, TDT4240 Staff

Notation:

Class diagram using UML

4.2. Process View

Purpose:

The process view explains the communication between different processes in the system, as well as how the system behaves in runtime.

Stakeholders:

Developer, Player, TDT4240 Staff

Notation:

Flowchart

4.3. Development View

Purpose:
The developer view is intended for the developers. It should ease development, and focus on software module organization by packaging the software in small chunks.

Stakeholders:
Developers, TDT4240 Staff

Notation:
Layers

4.4. Physical View

Purpose:
The physical view shows the interaction between the physical components of the system.

Stakeholders:
Developers, Players, TDT4240 Staff

Notation:
UML and drawing

5. Architectural Tactics

5.1. Modifiability

We decided to use the MVC pattern (Model View Controller). MVC main purpose is to separate the UI(User Interface) and the logic. This will make the code easier to read and modify afterwards for the developers.

5.2. Usability

For someone who plays this game for the first time, a tutorial will be useful to show how the game is played. Beside the tutorial, we will also need a screen that show the controls.

5.3. Availability

Availability in the game “Virion” is not very important. “Virion” is a local multiplayer game with no network used. But a stable game experience is an advantage, and the game should be organized in such a way that it avoids hard to track down bugs that affect the availability due to crashes.

5.4. Performance

We will not be able to make a game complex enough to tax the GPU. Bad programming and repetition of tasks can break this though.

5.5. Testability

We chose not to make automatic tests for this project. This will take too much of the time we can use to make a better game. But we will do functionality testing to make sure the game works properly.

6. Architectural- and Design Patterns

6.1. Design Patterns

6.1.1. Template (For Virus and different types of Cells)

Many elements in the game will share a significant amount of properties and functions with other elements. Using the Template design pattern, we can generalize certain behaviour, like collision, movement and drawing, to fit into a hierarchy of classes. This will reduce duplicate code, and ensure that only the differences between the elements are written separately, and the common elements are written once for all child classes.

6.1.2. Game loop

The game loop pattern is central to most games. XNA provides the game loop for us, so we do not have to implement this ourselves. We do however override the functions in the XNA game loop to suit our needs.

6.2. Architectural patterns

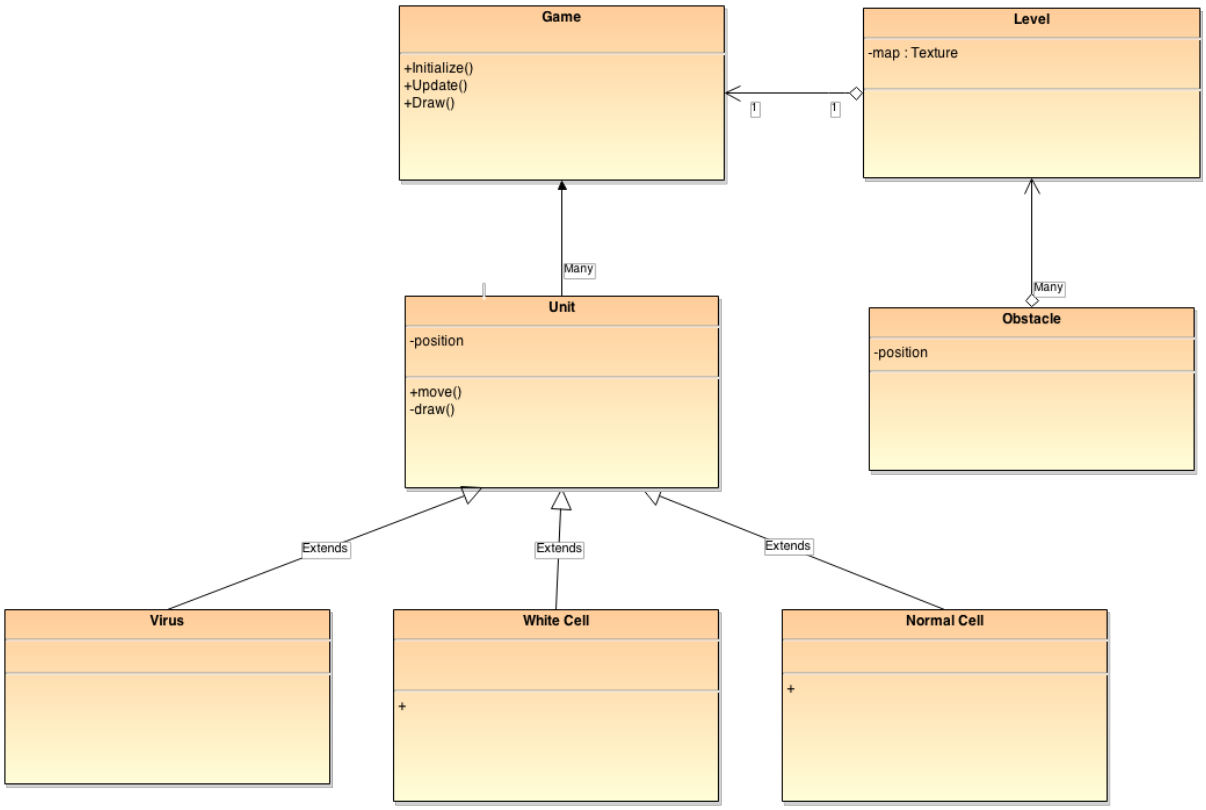
6.3. MVC (Model, View, Controller)

The MVC pattern is closely integrated with the XNA Framework, and is a good fit for structuring the application. XNA already separates the View (drawing to screen) from the logic (update function) and the model (game data and assets). Different from the classic MVC, however, is that there is no need for the model to update the view through listeners when changes are made, as the view is updated explicitly from the models at the end of the game loop sequence.

7. Views

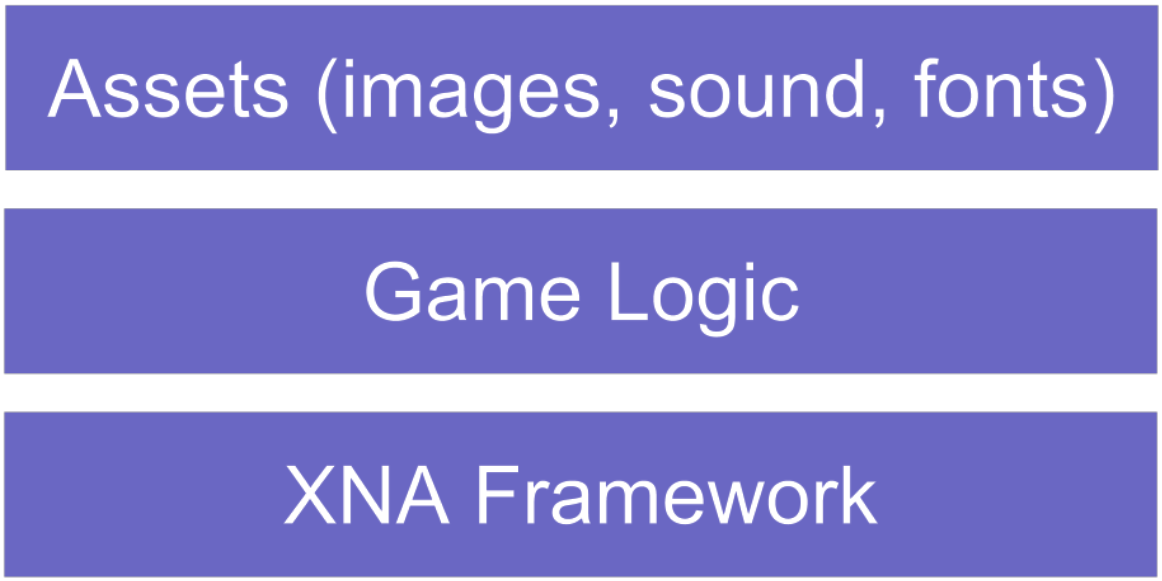
7.1. Logical View

We have chosen to display our logical view through a class diagram. This shows a rough overview of the system that we plan to implement. We rely heavily on the template pattern, since most classes have some common data fields. This also makes it easier to expand by inheritance.



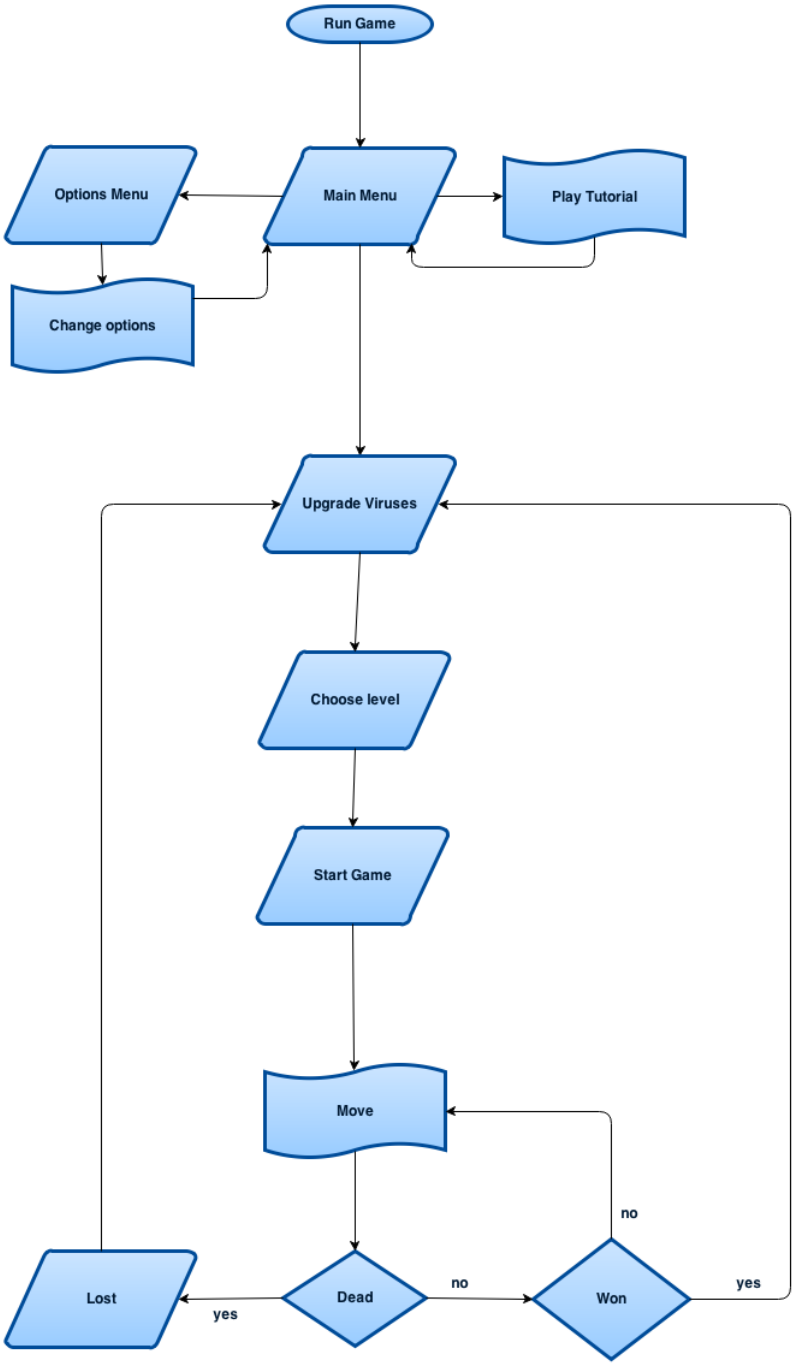
7.2. Development View

The development view describes how the development assets, such as code, media and libraries are organized. There are three main groups of assets: Media, the XNA framework libraries, and our own source code.



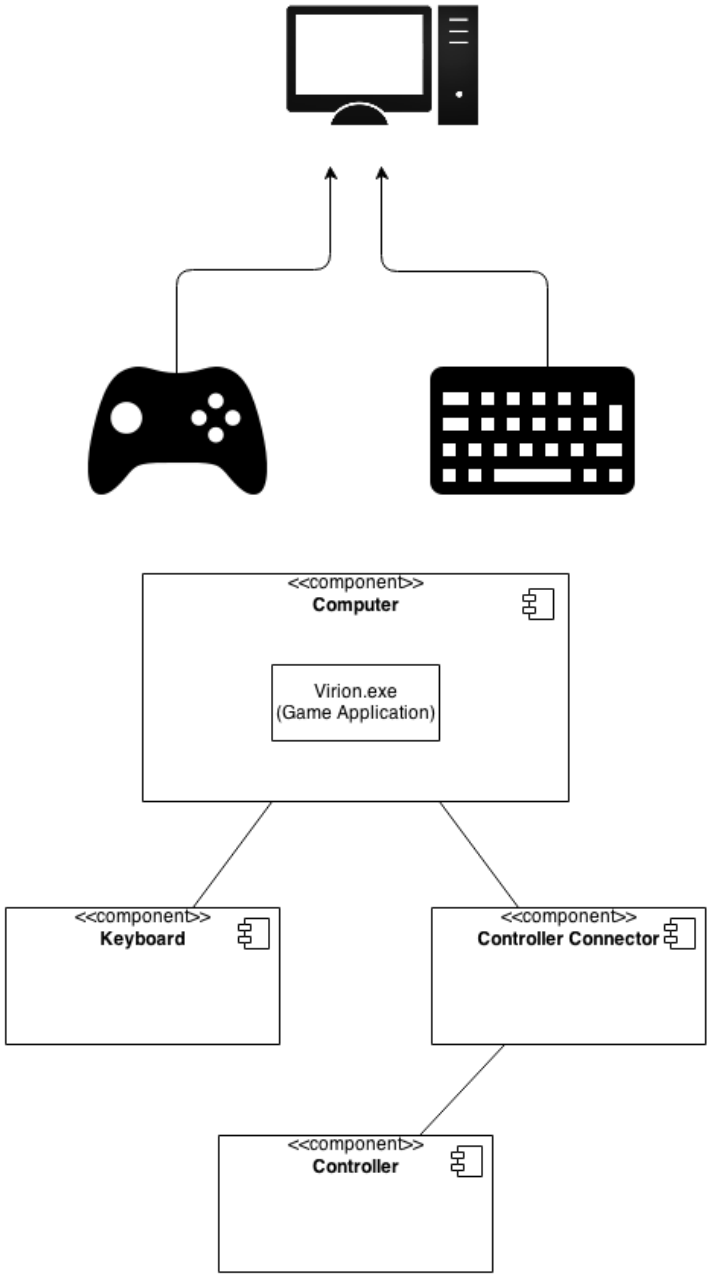
7.3. Process view

This view describes the process which the player experiences while running the game, from the outside. It shows, step by step, which actions can be taken, and how the state of the game changes as actions are applied.



7.4. Physical view

From the physical viewpoint, we have one or more players gathered around a single machine. This requires controls that fit with these constraints. Mouse + keyboard controls are not available for multiple people at the same time, so the controls have to be simple enough that two players can use the same keyboard at the same time, or support multiple gamepads



8. Consistency Among Views

The views describe different aspects of the project, and are consistent in the sense that the information in the different views do not conflict with one another. If any part of a view is changed, it should not affect the other views.

9. Architectural Rationale

The rationale behind the choice of the architectural pattern MVC is that it is already in place in the XNA Framework, and thus will be easy to build upon. As MVC encourages separation of responsibilities, it aids us in fulfilling the Modifiability and Maintainability requirements.

The template pattern is a widely used design pattern in game development that reduces error prone duplication of code, and makes visualization of the architecture easier for the developers.

10. Issues

- Locating ASRs
- Grasping the concept of ‘views’
- Getting to know game programming concepts for those with little experience with games

11. References

Books:

- Bass, Clements, Kazman, “Software Architecture in Practice”, Third Edition, Pearson Education, 2013

Websites:

- Factory Design pattern, http://www.dofactory.com/Patterns/PatternFactory.aspx, Accessed 22. Feb 2014
- 4+1 architectural view model, http://en.wikipedia.org/wiki/4%2B1, Accessed 22. Feb 2014

Lecture slides on “4+1”

Articles:

- Software Engineering G22.2440-001, Session 8 – Sub-Topic 1, Design Patterns, Architectural Patterns, by Dr. Jean-Claude Franchitti, Web:http://www.nyu.edu/classes/jcf/g22.2440-001_sp09/slides/session8/g22_2440_001_c82.pdf
- Architectural Blueprints—The “4+1” View Model of Software Architecture,
  Philippe Kruchten,
  Published in IEEE Software 12 (6) - November 1995
  Web:http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

12. Changes

Time	Action
23. Feb 2014	First draft finished