Bilkent University

Department of Computer
Engineering

Object-Oriented Software Engineering Course Project

CS319 Course Project: Virion

Design Report (Sections 1-3)

Mert INAN, Ulugbek IRMATOV, Irmak YILMAZ

Supervisor: Uğur DOĞRUSÖZ

Design Report (Sections 1-3)

Nov 12, 2016

This report is submitted to GitHub in partial fulfillment of the requirements of the Object-Oriented Software Engineering course CS319.

Contents

Design Report (Section 1-3)

CS319 Course Project: Virion

1 Introduction

In this report, high level design of the Virion system will be discussed and evaluated. It contains the overall introduction to the system and its design goals, and then the details of the architecture and subsystem components will be given.

1.1 Purpose of the System

Virion is an educational game where player tries to save a cell from virus attacks. Player can defend using proteins and other cell organelles such as mitochondria, vacuole, and cell wall. There are different kinds of viruses. Difficulty of the game is proportional to time. As time passes more dangerous viruses start coming. To get high score, player must keep cell alive as long as possible.

1.2 Design Goals

The main design goals of the system are dependent on the user-system interaction rather than the actual system configurations. As the implementation of the software does not involve complex algorithm design, and the crucial parts occur in the duration of the gameplay, design goals such as modifiability, adaptability, portability or maintainability are not the main concern, but will be incorporated into the implementation by itself.

On the other hand design goals such as user-friendliness, ease of learning, ease of remembering and ease of use are primary concerns. As Virion is an educational game, it should be easy for the user to understand the environment and the components involved. Even though the game can have a steep learning curve, it is necessary for the user to retain information learned from the game.

Other general design goals such as efficiency, high-performance are also important as the game need to perform its operations highly efficiently, as the outcome of the game depends on time, it is crucial to have a high-performing system.

During the design of the system, some design goals will need to be preferred, creating trade-offs. Here are two trade-offs that will need to be considered.

Functionality – Usability: As the system is a biology education game, it needs to be functional rather than usable at some instances. It is designed to maximize functionality and usability most of the times; however, if a choice is needed to be made, then functionality will be favored as the game should be adequate enough to model the real world interactions between cells and viruses. This may lead to a decrease in usability, such as extensive menus and logical links between organelles, which is not a significant decrease overall.

Time – Memory: Interactions between internal game components are heavily-dependent on time –scoring is also done according to time- hence, it is necessary to favor time instead of memory if the need arises. Even though time will be favored, overall memory consumption is not significant as there are no complex algorithms with high memory need, yet images of the components may sometimes slow down the processes which may lead to a delay during gameplay which is not a desirable scenario. Hence, if necessary, image quality will be shrunk to increase time reliability.

2 Software architecture

In this section of the design document, we will be analyzing the high level architecture of the Virion system by evaluating the subsystem components.

Among the widely applied architectures, most promising ones for this project were Model View Controller (MVC) and 3-Layer. We have discussed each architecture's drawbacks and advantages and decided on the one that best represents Virion's high level design. At the first glance, MVC seems to grasp the overall concept of the system but lacks the filesystem component to write the high score list in. Hence, we are choosing 3-Layer Architecture because we will be writing the high score list to the file in order for it to be persistent among sessions.

2.1 Subsystem decomposition

As the decided architecture of the system is 3-Layer, the subsystems will need to comply with the specific layering of that architecture. The topmost layer in this architecture corresponds to the user interface, middle layer has the actual business logic or the controllers of the game and the lowest layer contains the filesystem data management subsystem.

There are four subsystems present in the system: User Interface, Game Management, Game Components, and High Score List. This split is planned, at first, according to a MVC model; however, in the end it is transformed into a 3-Layer structure with High Score List subsystem being in the filesystem layer. Opaque layering is used in the design in order to increase flexibility, as the user interface layer or the high score list layer can change as the future stages of the project may turn it into a multi-user system.

High coherence and low coupling is tried to be achieved by making the boundaries of the layers as distinct as possible. UML Component diagram is used to show the relation among components at different layers as in Figure 1.

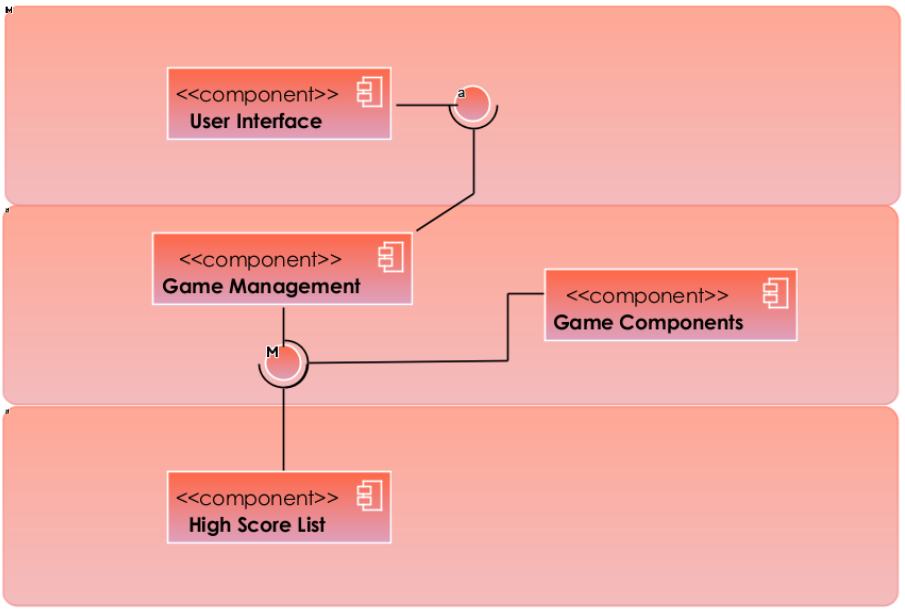

Figure 1 This is the UML Component Diagram of the 3-Layer Architecture of Virion system with subsystems.

2.2 Hardware/software mapping

As the Virion system does not demand a high processor power, it is enough to use the regular hardware with a single processor and optimized software to run the application. As there aren't multiple users, there won't be a lot of requests to filesystem which would not create a need for an extraordinary memory management. Hence, a regular computer with an operating system that is supporting Java applications will be adequate.

2.3 Persistent data management

The only data that needs to be persistent throughout different sessions of the program is the high score data. This data will not be regularly accessed either for read or write processes. Even in such a case, the writing will be done by the game management subsystem only, without any concurrency while writing. Two different options are available for the high score data management: filesystem and database. Because the database would require a lot of maintenance and is redundant without concurrencies in the system, it is decided to use filesystem database management to save high score data across different game executions.

Data does not need to be archived as the user will be able to see the top five high scores only which does not need to be stored throughout time intervals as the previous data is not needed and can be lost. Querying of the data will be done by interfaces inside the High Score List subsystem which would use regular file read/write operations.

2.4 Access control and security

As this is a single-player game, it has an open access to any player. It does not contain any authentication method. Accessing the game can be done by anybody who is authorized enough by the operating system to execute the game, itself. We will select procedure-driven centralized control for the control of the software itself as there are no multiple users, and control resides in one central controller object in the program code.

2.5 Boundary conditions

In the boundary conditions, the system will be acting according to the following.

Initialization: While the game is brought from a non-initialized state –such as the initial executable of the program- to the steady-state, the main processes will be done by the user interface subsystem. The user interface will display the splash screen during startup and then show the main menu of the game. Only the image data will be accessed for drawing the user interface during startup.

Termination: All cleanup procedures will be started by the main game controller in the Game Management subsystem. No single subsystem is allowed to terminate on its own. Every subsystem is notified if a single subsystem is terminated. Read and write to the filesystem will be completed before termination.

Failure: All game or computer related failures will result in loss of game state and score, as there is no save functionality. If a failure occurs during read and write to the high score list, warning messages will be shown and if necessary the highs core list will be erased. If failure occurs during getting the background images or component images, a restart of the game will be requested.

3 Subsystem services

The services that will be provided under the four main subsystems are as follows.

User Interface: Displays the screens, menu items, and game component images and notifies the game controller in the Game Management subsystem about any interactions between the screens, buttons, components and the player.

Game Management: Contains the main game controller class and other relevant controller classes that request high score updates to the list in the filesystem, screen updates to the user interface, game component state updates, movement updates, score and time updates.

Game Components: Contains all the model classes of the game objects and provide services to the system such as providing information about each game object, containing reference ids to the images of the game objects, and providing information about the availability of a game object.

High Score List: Provides services to read and write to the filesystem about the top five high scores upon instructions from the Game Management subsystem.