CS 319 - Object-Oriented Software Engineering

Design Report

Virus Attack

Section 2 - Group F

Cholpon Mambetova

Doğan Can Eren

Melisa Onaran

Nergiz Ünal

Course Instructor: Uğur Doğrusöz

Table of Contents

1 - Introduction **2**

1.1 Purpose of The System 2

1.2 Design Goals 2

1.2.1 End User Criteria 2

1.2.2 Maintenance Criteria 3

1.2.3 Performance Criteria 3

1.2.4 Trade - Offs 3

2 - Software Architecture **4**

2.1 Subsystem Decomposition 4

2.2 Hardware/Software Mapping 4

2.3 Persistent Data Management 5

2.4 Access Control and Security 5

2.5 Boundary Conditions 5

3 - Subsystem Services **6**

3.1 Design Patterns 6

3.2 User Interface Subsystem Interface 7

HighScores Class: 7

Help Class: 7

Credits Class: 7

Window Class: 8

InputManager Class: 8

IOManager Class: 8

3.3 Game Management Subsystem Interface 8

GameEngine Class: 8

MapManager Class: 8

3.4 Game Entities Subsystem Interface 9

3.5 Data Management Subsystem Interface 11

HighScoresManager Class: 11

1 - Introduction

1.1 Purpose of The System

Virus Attack is a system which aims to amuse users with its carefully designed, with different difficulty levels, gameplay. The system is based on user-friendly interface which basically guides the user how to play properly. The levels are well-designed in different difficulty levels, each aiming different purposes. For instance, the game begins with an easy level which aims to guide the user to become involved in game and experience how to play accordingly. Then, here comes a different level with an updated difficulty and aims to maintain the attention of the user. Subsequently, levels becomes different and much more difficult to accomplish.

1.2 Design Goals

It is quite hard to identify the design goals of the system with respect to further upcoming design qualities that the system will be based on and mainly focused. With this aspect it is appropriate to mention that numerous of our design goals follow non- functional requirements of our system that is mentioned in "Analysis Report". Critical design goals of our system are described as follows.

1.2.1 End User Criteria

Ease of Use: The system that we design is a game, therefore it is designed on a user-friendly interface aiming to make sure the player finds it easy to play and enjoyable while using the system provided by creators. From that point of view, system provides user-friendly menus, easily accomplished operations and well-designed navigation through menus and well-designed performance.

Ease of Learning: The player is not obligated to have a background knowledge about the system of the game. Therefore, it is one of the most important things to guide the player and make sure that the instructions are easy to follow. For this purpose, a document exists, created by creators of the game, which is available in the "View Help" section and aims to have player become updated about the qualities of the game.

1.2.2 Maintenance Criteria

Extendibility: To make the game alive it is mainly important to keep the game updated and to present the players new components and features about the game in order to have interest of players. That is why, the design is available to be updated with new versions and easy to modify to the existing game.

Portability: In this maintenance, we have decided that the system will be implemented in Java, it involves platform independency therefore the system will provide portability.

Modifiability: The system(game) is designed to be easily modified. Therefore, to be able to achieve the goal this quality would make the minimization of the couples of the subsystems and therefore perfectly avoids the impacts of components of the system when there is a change.

1.2.3 Performance Criteria

Response Time: As in most of the games, in this game it is quite important to have pressure on the players to be effective immediately in order not to get killed, therefore the player would be focused and interested during the game and also enjoys the challenge. When they are focused to the game, visual animation is also provided almost immediate through the player's action and the process of the game.

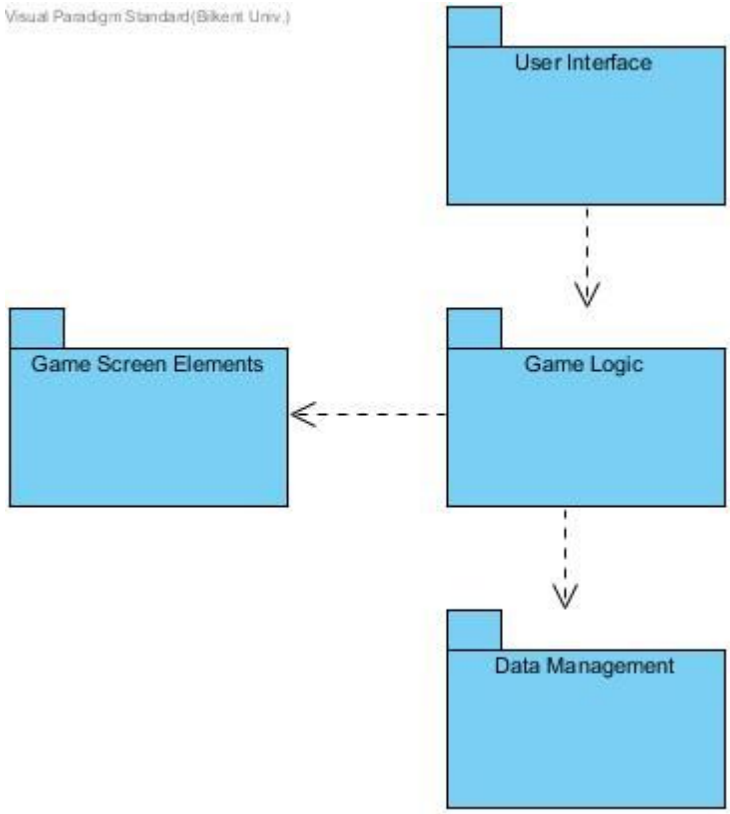
1.2.4 Trade - Offs

Ease of Use and Ease of Learning vs. Functionality: We propose our priority as usability being higher than functionality. Meaning the user should not be having trouble while learning how to play and how to process the game. Therefore the system does not make conflictions about the functionality to the player and the player does not get lost around the system process. When the game is easily understandable player would enjoy more.

Performance vs. Memory: The main goal of the system is to make the animations, transitions and effects very intense. Therefore, the performance is the system is one of the main things that is focused and thought. For instance the game involves a bonus item that makes the user gain the invincibility, freezing and getting a life as a bonus. Because system needs the best performance as possible it has a game map in the memory which is to access fast when it is needed rather than having iteration for all the map.

2 - Software Architecture

2.1 Subsystem Decomposition



The system uses a 3-layer architectural style. Also, as it has a closed architecture style, a component can only communicate with its neighbors or the lower-layered ones. There are three layers which are in a hierarchical order. First layer is called as “UI Layer”, which includes the User Interface component. This layer provides the ability to interact with the user. The second layer is called “Application Logic Layer”, which includes the Game Logic and Game Screen Elements components. This layer handles the logic operations of the system. The third layer is called “Data Persistency Layer”, which includes the Data Management component. This layer provides the ability to work with the data such as saving the highscores and retrieving them.

2.2 Hardware/Software Mapping

Virus Attack will be implemented in Java programming language and for this purpose the latest version of JDK(8) will be used. In hardware configuration, Virus Attack needs keyboard, for to type the names of the players which is necessary for the high score list and

also to make move the virus. Since it is going to be implemented in Java, a computer with basic softwares installed as an operating system and a java compiler to compile the code and also to run the .java.file. And, also the main reason to choose Java ,which was its platform independency, will be used. And to be able to store the data we already will have a .txt based structures to formalize the game maps and to store the high score list, chosen operating system will need to support .txt file formats. On the other hand the system will not require any internet based connection to operate.

2.3 Persistent Data Management

The game basically does not involve a complex database system, it just stores the map structure and the high score list as text files. The game maps will necessarily be created before the Virus Attack is executed, therefore its data will be permanent. If text file is corrupted, through the working process of the system that we have designed it would not affect in-game, such as items but the system would no longer be able to load the corrupted map. Also it is planned to store the sound of the effects and the images of the game items in hard disk with proper sound and in image formats such as .gif.

2.4 Access Control and Security

As mentioned earlier(Hardware/Software Mapping section), Virus Attack does not require any internet connection. Therefore, the players will not be in trouble with internet based security issues or else will be freely playing the game after installation.

2.5 Boundary Conditions

Initialization

Virus Attack does not have a regular .exe or such extension, therefore it does not require an install. Though the game is going to come with an executable .jar file.

Termination

Virus Attack is designed to be able to be terminated, in other words closed, by clicking “Quit Game” button in the main menu. If player wants to quit during game play, the system

provides a pause menu by which player can return to main menu and perform quit.

Error

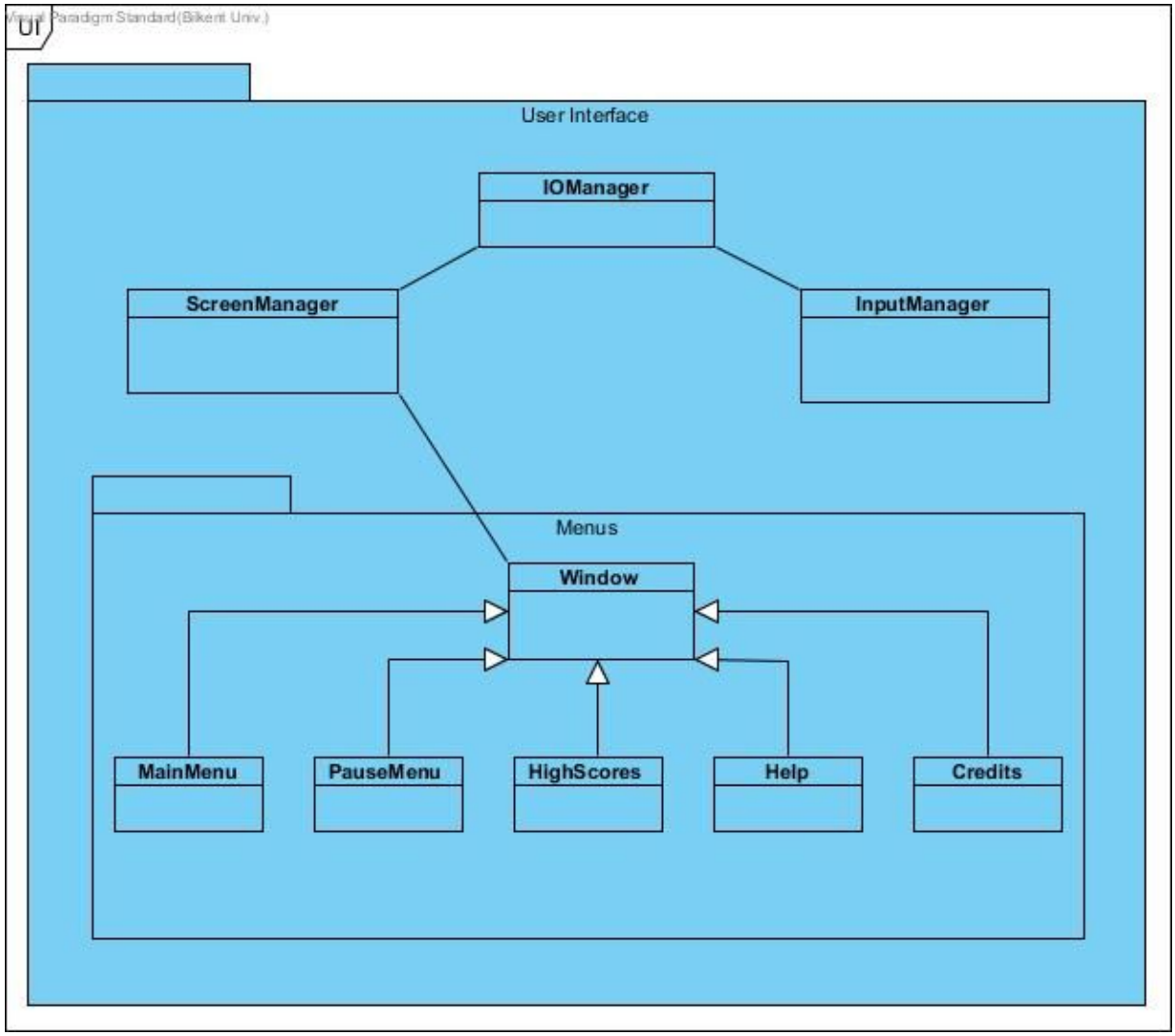
If an error occurs that game resources could not be loaded such as sound and images, the game will become starting without having any images or sound. Also, if program does not respond because of a performance issue, even we keep these kind of problems minimum, Player will lose all of current data.

3 - Subsystem Services

3.1 Design Patterns

Façade design is a design pattern which provides developers a unified interface to a set of interfaces by defining a higher-level of interface in a subsystem which is basically easier to use. This pattern is known for its opportunities such as providing maintainability, flexibility when developing the system, extendibility and reusability. Because through the mechanism of this pattern, any changes made in components of the subsystem can be reflected when they are made in the "Façade Class". In our design we have chosen to use façade pattern in the following subsystem, Game Entities. For this subsystem our façade class is MapManager class which basically executes the operations to deal with the entity objects in this system.

3.2 User Interface Subsystem Interface



HighScores Class:

HighScores class uses the HighScoresManager that holds the highest scores that was ever achieved in this game, to print them to screen using ScreenManager. It also extends the Window class.

Help Class:

Help class is used to display some useful information for the user to easily learn how to play the game. It also extends the Window class and uses ScreenManager to print itself.

Credits Class:

This class is responsible to display the information about the creators of the game. It also extends the Window class and uses ScreenManager to print itself.

Window Class:

This class is used to bring together the common features of its child classes such as menu classes and other displayed windows used in the game.

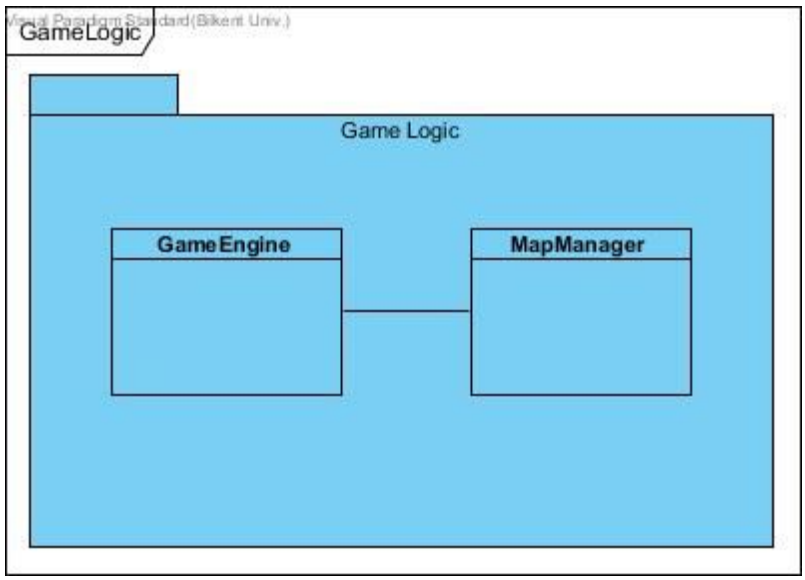
InputManager Class:

This class is responsible for taking the user input and report it to the IO Manager.

IOManager Class:

This class outputs the ScreenManager to the display and gets the input and sends the input to the InputManager.

3.3 Game Management Subsystem Interface



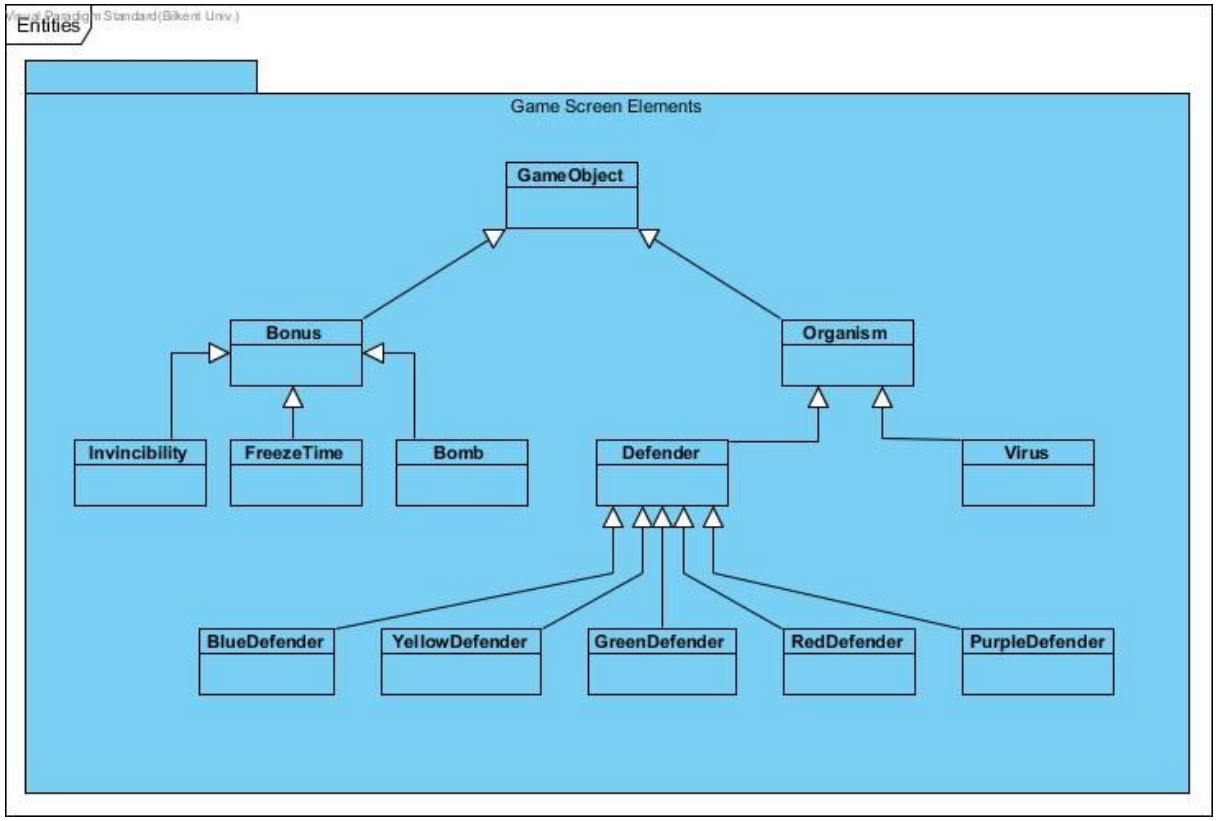
GameEngine Class:

This class is responsible for handling the main logic operations of the system. It communicates with all the other subsystems to process.

MapManager Class:

This class is responsible for tracking and reporting the objects on the map. It interacts with the GameEngine class to provide information.

3.4 Game Entities Subsystem Interface



GameObject Class:

This class is responsible for every type of game objects, like the virus or bonuses, on the game. It is a parent class that holds positions, colors, sizes are inherited from this class to every child classes. Drawing of objects and updating their positions are also provided by this parental class to the child classes.

Bonus Class:

This class brings together different type of bonus classes' common features. These features are:

Activation: When virus and the bonus collide, the bonus is being activated.

Lifetime: Determines how much time the bonus can be available on the screen.

Duration: Determines how much time the bonus power will be efficient.

Invincibility Class:

Child class of Bonus class, this bonus makes the virus invincible for a given time duration.

Freeze Class:

Child class of Bonus class, this class is responsible for freezing the defenders for a time

duration of the bonus.

Bomb Class:

Child class of Bonus class, this class responsible for destroying the defenders within a radius of damage. When the virus collides with it, it gets activated and kills the nearby defenders.

Organism Class:

This class is parent class for defenders and the virus. This class is responsible for organism's vertical and horizontal velocities as a common features of Virus and Defender classes.

Virus Class:

This class extends the Organism class. It is responsible for all virus data such as HP, Health Point, max HP and invincibility. If the virus collides with the defender and the invincibility is active, the virus doesn't lose any HP.

Defender Class:

This class gets together common features of five different defender classes and extends the Organism class. These five child classes of Defender class have damage, color, and size as common properties. Also responsible for velocities of virus due to the parental features. Child classes are for specification of these common features.

BlueDefender Class:

Defender that creates low damage, blue colored, medium sized and slow.

YellowDefender Class:

Defender that creates low damage, yellow colored, small sized and fast.

GreenDefender Class:

Defender that creates medium damage, green colored, big sized and slow.

RedDefender Class:

Defender that creates high damage, red colored, small sized and fast.

PurpleDefender Class:

Defender that creates high damage, purple colored, big sized and fast.

3.5 Data Management Subsystem Interface

HighScoresManager Class:

This class is responsible for keeping, saving and retrieving the highscores. It interacts with the GameEngine class to process data.