Bilkent University Department of Computer Engineering

CS319 Object-Oriented Software Engineering Project

Siege

Design Report

Project Group 26: İrem Yüksel, Gökhan Şimşek, Emir Acımış, Furkan Küçükbay

Course Instructor: Uğur Doğrusöz

November 12, 2016

Table of Contents

1. Introduction

1.1. Purpose of the System

Siege is a game aiming to entertain users while they are trying to protect the sacred object from the enemies. It is influenced by the classical game Tank 1990. The game will have different levels, each have different maps and difficulties.By adding some power-ups and different types of enemies, we tried to increase the player's attention. The game will be successfully completed if the player can prevent the enemy tanks from capturing the sacred object.

1.2. Design Goals

1.2.1. Adoptability

The game Siege will be implemented in Java. The main reason we preferred Java to other programming languages such as C or C++ is that Java provides cross platform portability. So the users will be able to run and play the game in any platform.

1.2.2. Efficiency

The game must run fast enough to respond each movement of every object such as bullets and the player's tank. Since the player's lives depend on it, this is crucially important that the game runs smoothly.

1.2.3. Reliability

Siege will be bug-free and will not crush while the user plays the game. In order to provide this goal, the tests will be done for each part separately and continuously. To prevent the game from crushing with unexpected inputs, we will define different exceptions which will not affect the game.

1.2.4. Usability

The system will be designed so that any user can easily understand and interact even on the first opening of the game. While being as easy as possible, it will also be interesting for the user's excitement. This will be provided with different graphics.

1.2.5. Extendability

We will structure the system in a way that it will enable us to customize it for any further improvements. It should be designed in a way that these improvements will not harm the structure of the original design and cause any crushes.

2. Software Architecture

2.1. Subsystem Decomposition

For the system design, MVC architectural style is chosen. As this architectural style prompts, the system is decomposed into three subsystems: Model, View and Controller. The model subsystem is developed such that they do not rely on any view or controller subsystem. The reason behind choosing MVC is that the components become less susceptible to any further changes. If, for instance, the interface is decided to be changed, Model subsystem would never be affected by that change. As well as this extendibility issue, the decomposition of the system shares out the complexity among the subsystems.

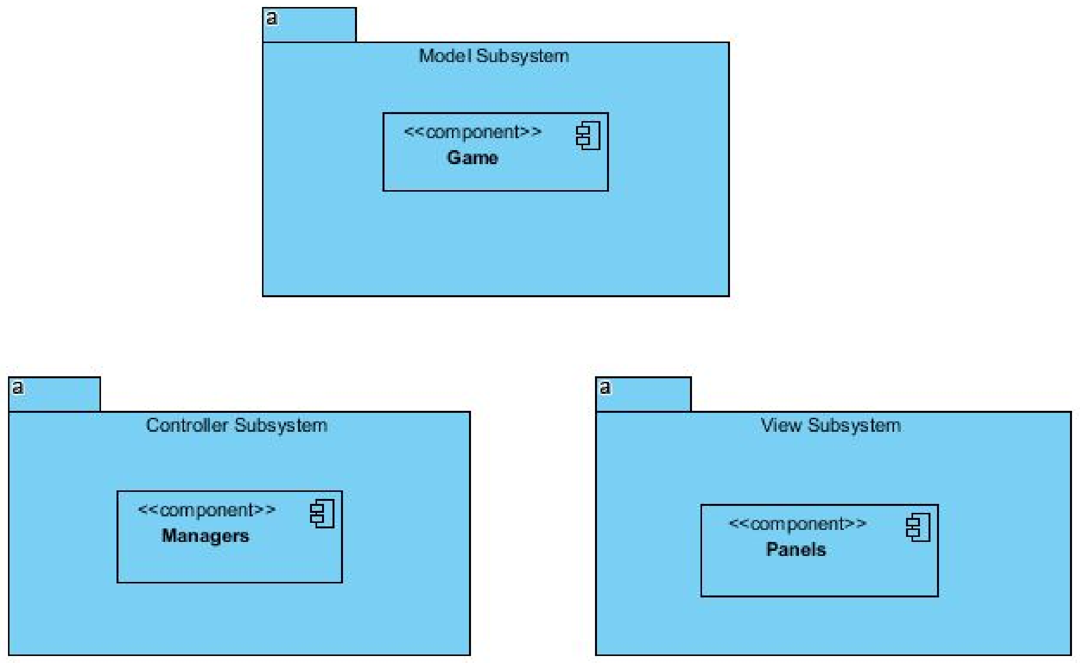


Figure 1: Subsystem Decomposition Diagram: The system is designed to have an MVC Architectural Style. Thus, three subsytems are Model, View and Controller subsytems. These subsystems serve for Data, GUI and Management respectively

2.1.1 Model Subsystem

In the Model subsystem, domain knowledge is maintained. Model objects and their relationships are represented in this subsystem. Basically, Model subsystem is responsible for keeping the data.

2.1.2 View Subsystem

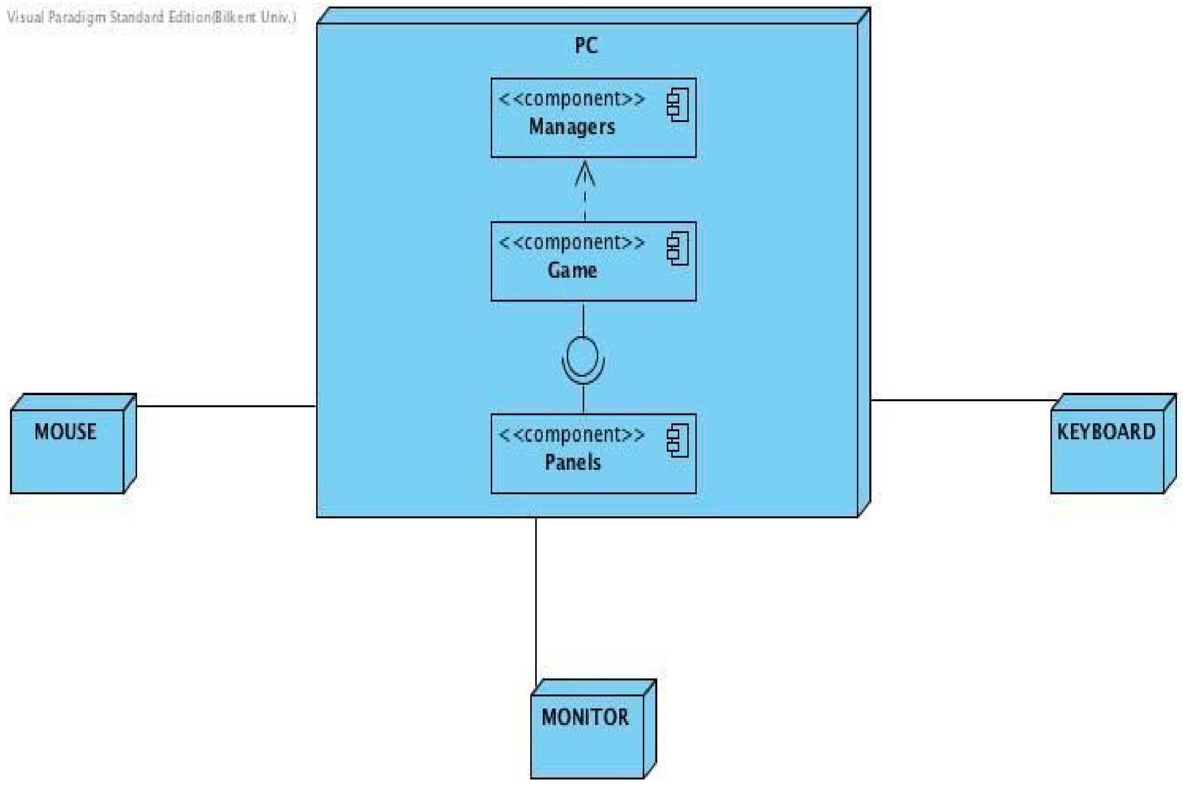
View subsystem involves view components. From the start to the end, View Subsystem will be responsible from displaying each specific screen to the user. Whenever the Model Subsystem changes, renderings of the objects will be handled through this subsystem as well.

2.1.3 Controller Subsystem

Controller subsystem shelters Manager objects which are responsible for controlling the sequence of interactions in the game logic.

2.2. Hardware/Software Mapping

The Siege game will be developed by the Java programming language and we will use the last development kit of Java which is called JDK8. For playing the game the users should have Java compiler inside their computer, if not, the users cannot play the game. When user play the game, user just use monitor and keyboard of the PC. When the user types the name to the High Score list the user use keyboard and mouse for typing their name. For the high score list we will use .txt file, for sound we will use .wav and for the images we will use .png format. Therefore, the users' computers should support and have .txt, .wav and .png. There is no need an internet connection for the game. The following diagram shows the communication between the hardware and software components of the game. When we create the game we will use Model-Controller-View architecture.



2.3. Persistent Data Management

In the game Siege there is no need to use a complex database system because the high score system will just keep 5 best score and this will be done by the text files. In the game there is no load function so that when the user terminates from the game the user will restart the game, the game does not keep old games. The sound and images are not much complex and we will keep them into the hard disk of the computer. The format of the image is .png and the format of the sound is .wav.

2.4. Access Control and Security

The game does not require any internet connection. After user open the game, the game can be played easily. There will be no limitation for accessing the game. The game does not require a user profile, that’s why the game Siege is a secure game too.

2.5. Boundary Conditions

Initialization

- When the player opens the .jar file he/she can access the game. The game does not require any install.
- When the user pushes the play button from the main menu, the user can access the game and levels,
- When player pushes the help button then the user see the controls of the game,
- When the Settings is open the user can control the settings functions,
- When the user pushes the High Score button then the user see the 5 best high score of the game.

Termination

- When the user pushes to the Exit button from the main menu, the program will be terminated.
- While user playing the game and wants to quit the game, the user press the pause button and return the main menu and terminate the program by pressing Exit button.
- Player can close the game by pushing the ‘X’ from the Panel.

Failure

- If the user computer does not have Java compiler, then the program will not open and execute.

- If the files like image, sound do not inside the computer then the program will run but the other staffs of program like image, sound will not work.
- Possible hardware problem can affect the program.
- Cutting the power of computer.

3. Subsystem Services

The proposed system will follow the MVC model, and thus will be divided into three parts named model, view and controller. As a brief introduction to their services, it can be said that the view subsystem is responsible for displaying system to the user, the model subsystem maintains the domain knowledge, and the controller subsystem handles user inputs and makes changes to the system accordingly.

3.1. Services of the Model

getObjectStatus: This is a service that the Model subsystem provides to the Controller subsystem. Model provides the current status of the game objects to the Controller, such as the remaining hits for a destroyable object to be destroyed, the remaining time for a power-up before it disappears, remaining lives of the player tank, current locations of the objects on the layout.

updateView: This is a service that the Model subsystem provides to the View subsystem. With this service, the information processed will be provided to the user to be viewed.

3.2. Services of the View

showView: The View subsystem provides the visuals of the game to the user. According to the inputs coming from the Model subsystem, View subsystem changes the respective locations and stati of the objects, and shows this to the user.

3.3. Services of the Controller

updateObjectStatus: After the necessary operations are performed by the controller, in light of the current state of the game gathered from the Model subsystem and the provided user inputs, the Controller subsystem sends the changes that should be made to the Model subsystem. e.g. after checking for collisions between a bullet and a wall, if the wall had required only 1 hit to be destroyed, Controller informs the Model to update the said Wall and Bullet objects, which the Model will in turn destroy both of those objects.