CS 319 - Object-Oriented Software Engineering

System Design Report

Bombplan

Group 1

Asena Rana Yozgatlı	*21000132*
Berk Yurttaş	*21200581*
Mehmet Furkan Şahin	*21201385*
Saner Turhaner	*21100475*

Deadline: 27/03/2016

Course Instructor: Uğur Doğrusöz

Table of Contents

1. Introduction 4
   1.1 Purpose of the System 4
   1.2 Design Goals 4
      1.2.1 Reliability 4
      1.2.2 Modifiability 4
      1.2.3 Adaptability 4
      1.2.4 Portability 5
      1.2.5 Response Time 5
      1.2.6 Good Documentation 5
      1.2.7 Well-defined Interfaces 5
      1.2.8 Readability 5
      1.2.9 Ease of Use 5
2. Software Architecture 6
   2.1 Subsystem Decomposition 6
      2.1.1 User Interface Subsystem 9
      2.1.2 Controller Subsystem 9
      2.1.3 Model Subsystem 10
   2.2 Hardware/Software Mapping 10
   2.3 Persistent Data Management 12
   2.4 Access Control and Security 12
   2.5 Boundary Conditions 12
3. Subsystem Services 14
   3.1 Services of the Controller 14
   3.2 Services of the Model 15
   3.3 Services of the View 15

Table of Figures

Figure 1: Subsystem Decomposition Diagram 7
Figure 2: Subsystem Decomposition Diagram in Detail 8
Figure 3: Component diagram for User Interface Subsystem 9
Figure 4: Component diagram for Controller Subsystem 9
Figure 5: Component diagram for Model Subsystem 10
Figure 6: Deployment Diagram for Hardware/Software Mapping 11

1. Introduction

1.1 Purpose of the System

Bombplan is a new version of an existing game called Bomberman. Our aim is making a game such that players from all age groups can enjoy. In this manner, we will design our system to be neither easy nor hard.

In our system, there are two additional bonuses that player can take. The goal of our system is that player eliminates all monsters in the maze in a given time. After player kills all monsters by planting bombs, s/he finds the door which is hidden behind a random wall. Passing the door ends the level successfully. The main purpose is being successful at each level.

1.2 Design Goals

1.2.1 Reliability

We aim to make our system reliable. In order to achieve this, the system should not crash or give an error in runtime. The system is going to be designed such a way that it will not accept any wrong input.

1.2.2 Modifiability

Our system should be modifiable. An update should be implemented by developer with no confliction. We will make our system modifiable by object oriented design. With help of object oriented design, new classes and objects will be added to existing code properly.

1.2.3 Adaptability

Bombplan should be played in different environments. So we will develop our system using Java. The only requirement to play the game is having Java Runtime Environment.

1.2.4 Portability

The system can be deployed to different development environments. We will use Java Archives files to achieve portability so building executable programs from source code will not be a problem for different platforms.

1.2.5 Response Time

We will design our system in order to have low response time meaning that it should not surpass 1 second. Since it is an interactive game, responses to user inputs should not be too high.

1.2.6 Good Documentation

The system has to be well documented for both user and developer. In order to achieve well documentation, all reports need to have minimum error and should be clear to understand. Besides that a user manual will be provided.

1.2.7 Well-defined Interfaces

The user interface should not be complicated. We are going to design our system such a way that all visuals will be self-explanatory.

1.2.8 Readability

We aim to have a readable source code. It will allow that developers understand the source code without any problem. Thus, modifying the existing code will not be a problem.

1.2.9 Ease of Use

Our games will be designed regarding that it can be played by players from different age group. So it will not be difficult to understand and control the game. In order to achieve that,

we will keep the number of input keys as minimum as it can be. Besides, difficulty level of game will be neither too easy nor too hard.

2. Software Architecture

2.1 Subsystem Decomposition

To be able to provide all the classes in our project as a proper, organized and working model, we created the sub-system decomposition. The system structure is created in a way that the classes serving for similar functionalities are working as a one system component. The sub-system components are designed to be working separately as much as possible for modularity and maintainability purposes. In the light of the above principles, we created the sub-system decomposition diagram in Model–View–Controller manner as it can be deduced from Figure 1 below:

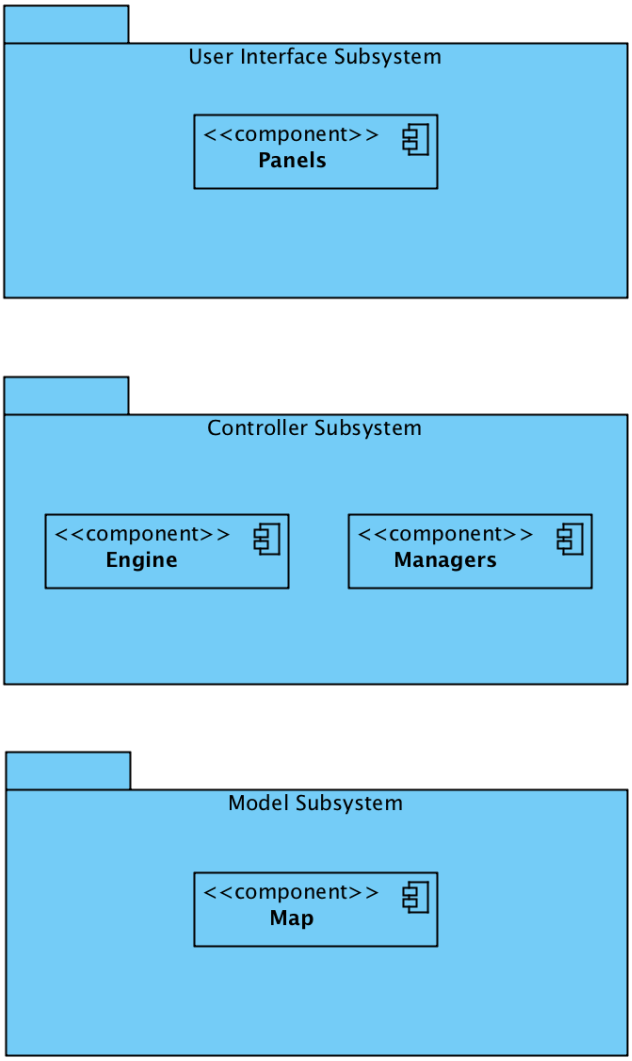

Figure 1: Subsystem Decomposition Diagram: The system is mainly composed of 3 parts; User Interface, Controller, and Model subsystems. They serve for GUI, Management, and the data parts of the project respectively.

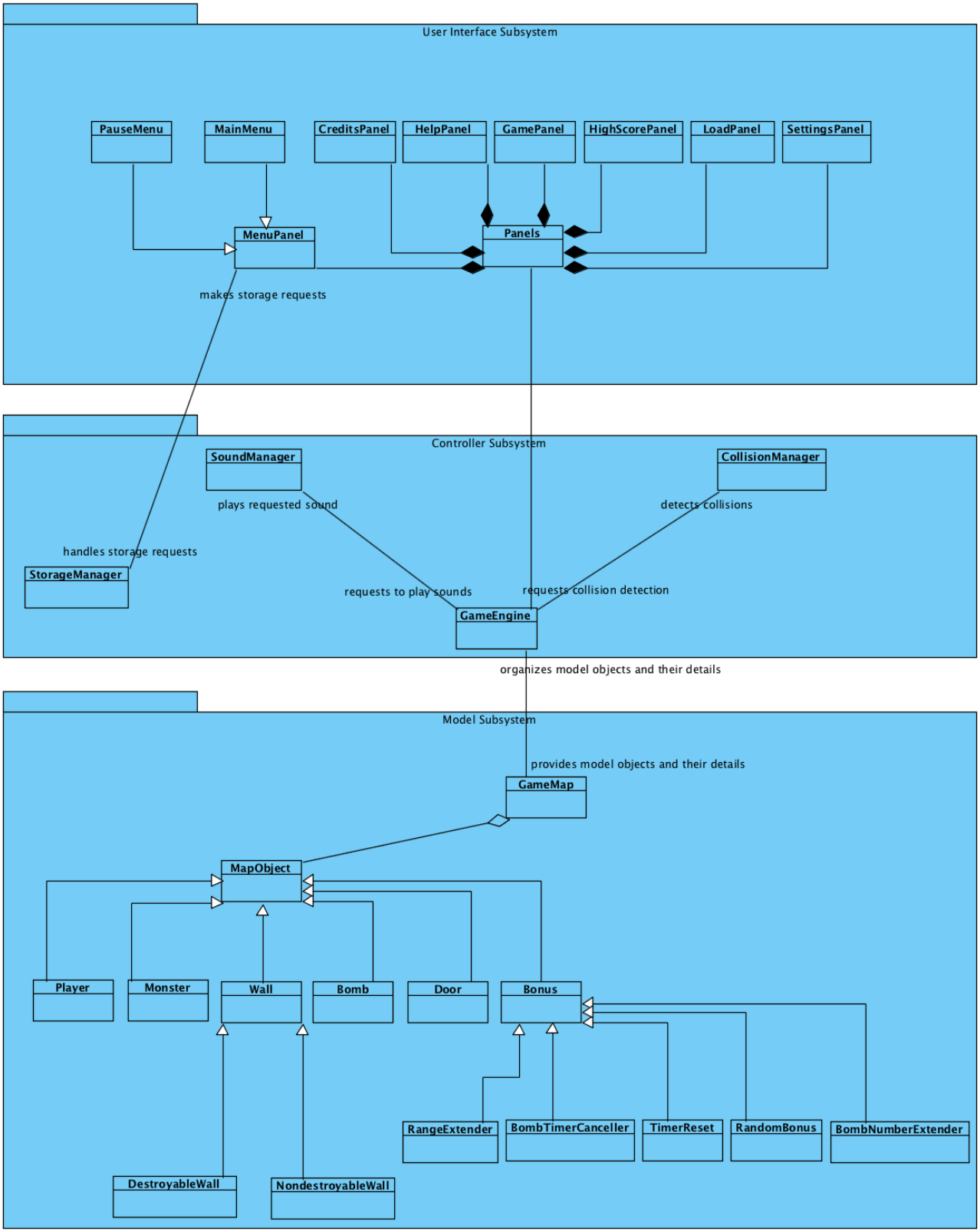

Figure 2: Subsystem Decomposition Diagram in detail

2.1.1 User Interface Subsystem

The User Interface Subsystem will be composed of mainly Panels. It will handle the renderings for each specific screen from starting to the termination of the game. Transitions between screens and all of the object renderings for the current screen needs will also be handled in this subsystem. The diagram itself can be seen in Figure 3 in detail.

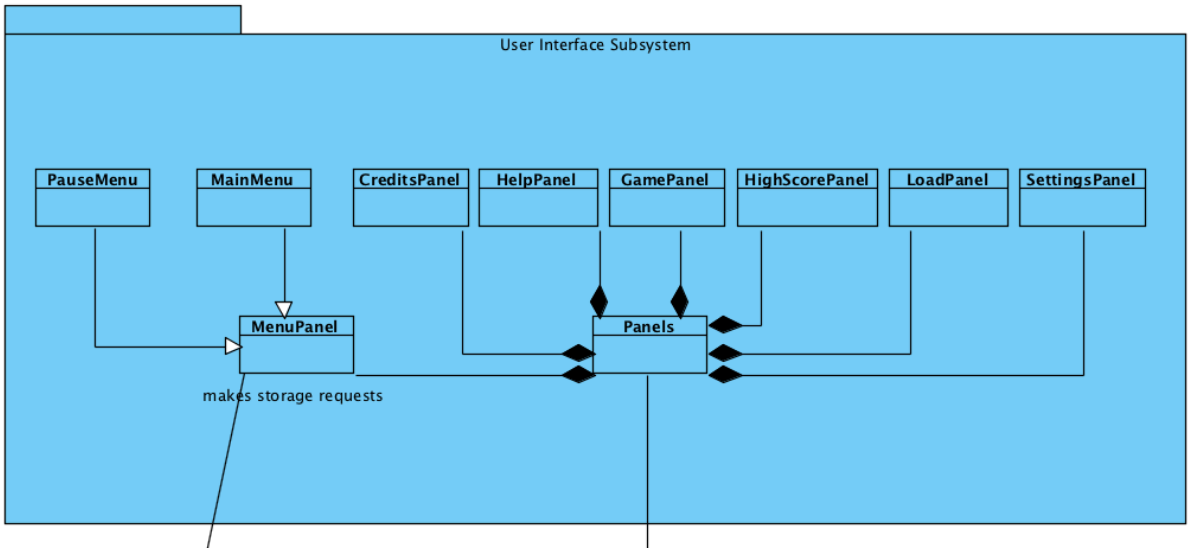

Figure 3: Component diagram for User Interface Subsystem

2.1.2 Controller Subsystem

In the controller subsystem, we grouped together the manager objects to be able to control the game data and its logic. The diagram itself can be seen in Figure 4 in detail.

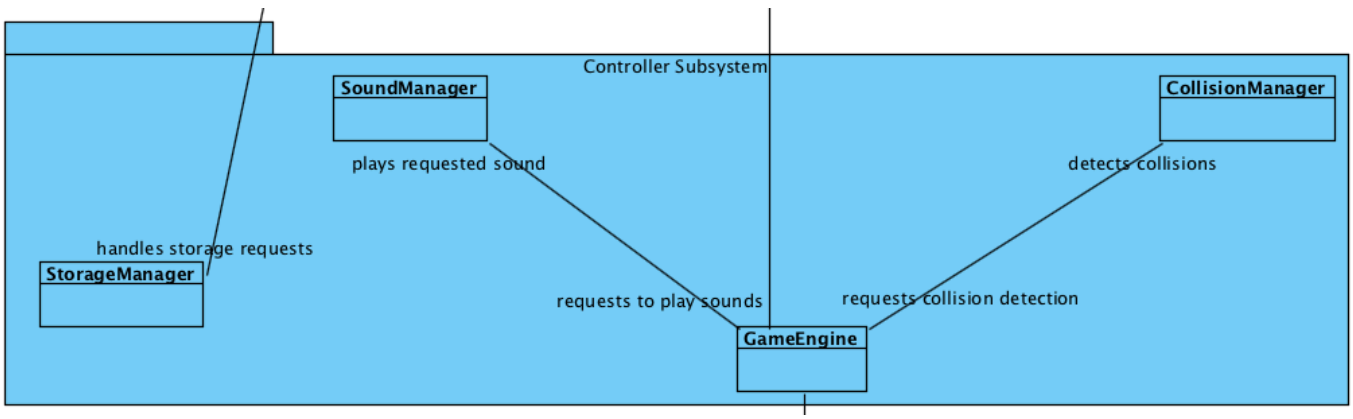

Figure 4: Component diagram for Controller Subsystem

2.1.3 Model Subsystem

In this subsystem our model objects and the relationship between them are represented. Model subsystem is basically the subsystem that keeps the data of the main game objects. The diagram can be seen in Figure 5.

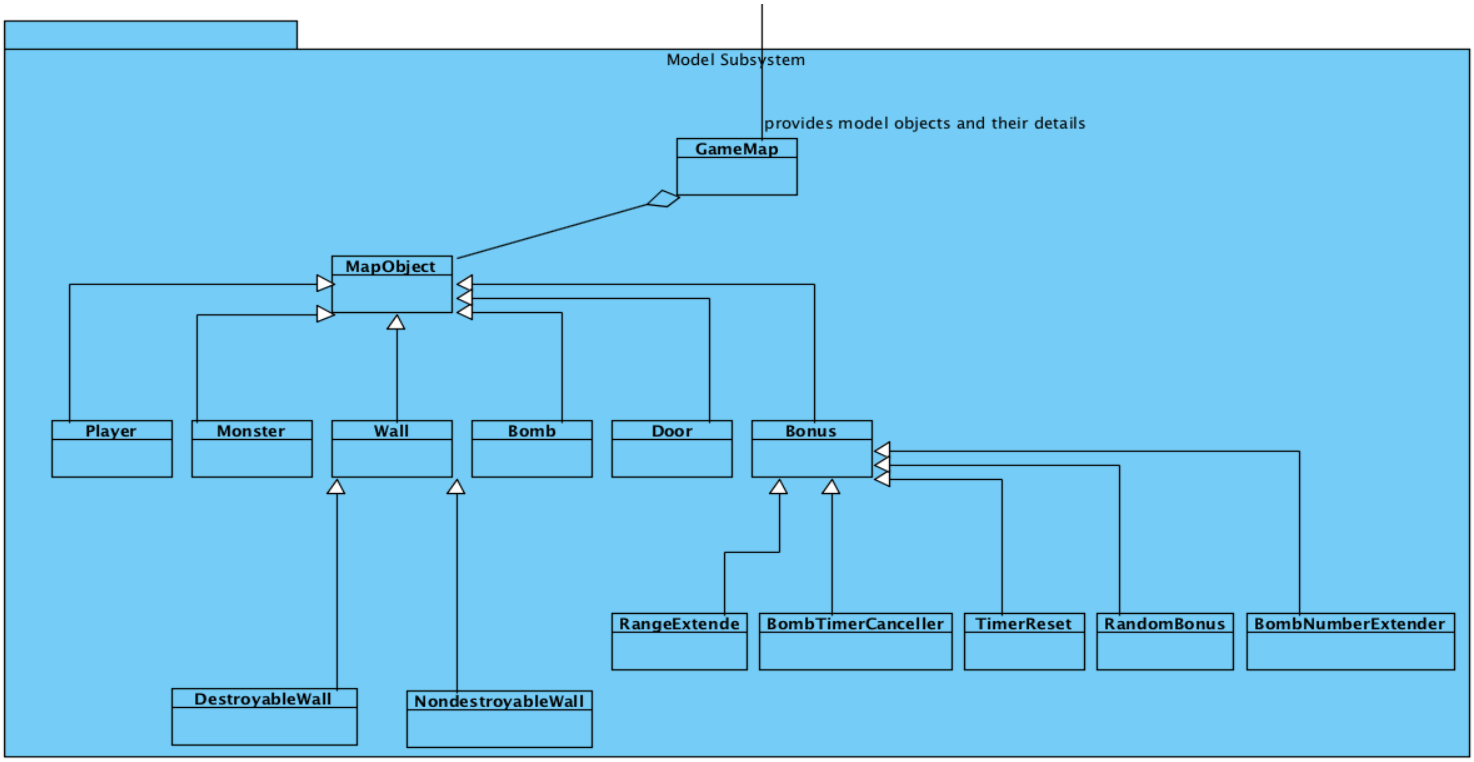

Figure 5: Component diagram for Model Subsystem

2.2 Hardware/Software Mapping

Bombplan will be developed using Java programming language; therefore we will use latest JDK (8). As hardware configuration, for player to type names on high score list and give input to the system Bombplan requires a mouse and a keyboard. To be able to run and play Bombplan, the computer needs to have an operating system and a Java compiler installed in advance. Java's platform independency is also needed as well.

Since to form game maps and store high score list and settings data we will use .txt files, and to store images and sounds in the .png and .wav formats respectively, the operating system should support .txt, .png, and .wav file formats. The game will not need internet connection.

Following deployment diagram shows communication between PC, Mouse, Keyboard and Monitor devices as separate nodes. As subsystem decomposition degrades the whole system in four components using Model-View-Controller architecture, these all components are depicted in PC node at below deployment diagram. PC node is in communication with Keyboard since player can play the game via keyboard buttons. And also mouse is using to give inputs to the panels. Monitor is output device which displays data given by view classes, in other words it communicates with Panels component. Inside of the PC, Engine depends on Managers component. Engine requires Map interface, model data, and it provides interface to the Panels component, View classes.

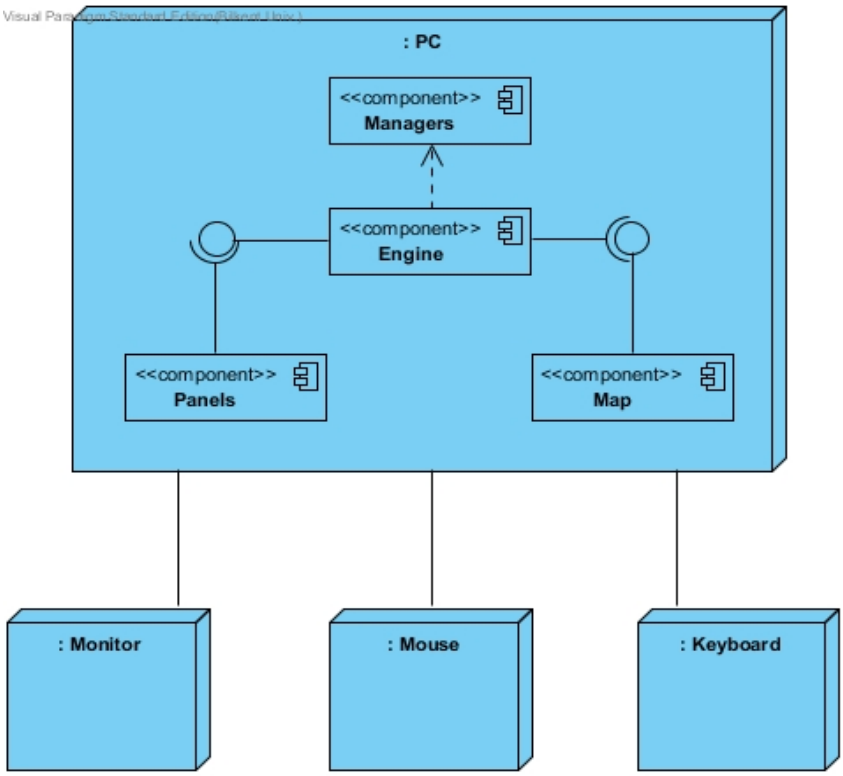

Figure 6: Deployment Diagram for Hardware/Software Mapping

2.3 Persistent Data Management

There is no need to use a complex database system, since Bombplan will store map structure, settings records and high score list in the associated text files. Our game maps as text files will be created in advance, so these data will be persistent. If any of these .txt files are corrupted, it will not affect game objects. On the other hand because of the file corruption the system will not be able to load map or the system will display error on the panel related to corrupted data. We will also store game sounds and object images in hard disk with simple sound and image formats (.png and .waw).

2.4 Access Control and Security

Bombplan will not need internet connection to run. After initialization, player can play the game without any authentication process. There will be no control or restrictions for access to game. Since Bombplan will not include any user profile, there will be no security issues in Bombplan.

2.5 Boundary Conditions

Initialization

- The game starts when player opens executable file. Bombplan is ready to use and main menu screen shows up. JVM loads all classes of the program.
- The game graphics are drawn by Java's Swing library.
- On the main menu screen there are seven buttons: Start Game, Help, Load Game, Options, Highscores, Credits, and Exit.
- When player selects Start Game option from the main menu, game panel will be loaded by the system and level one is going to be created and then game starts.

- When player chooses Help option from the main menu, Help panel will be displayed with its contents.
- If player chooses Load Game option from the main menu, system will load saved games from the associated files and show them as a list.
- If Options is selected, settings are going to be loaded and displayed by the system.
- When Highscores is selected, the system will load high score records and display them as a table.
- If Credits is selected, credits contents are going to be displayed by the system.
- Finally, if Exit is selected, the game will be terminated.

Termination

- On the main menu player can exit from the game by selecting Exit option.
- During the game player can exit using the pause menu by stopping the game. Since game is still continuing, after termination records will not be saved by the system.
- Player can press “x” button on the frame at the top right corner whenever s/he wants. If termination is done by clicking this button then no record will be saved.
- The system will ask player whether s/he really wants to quit when player clicks one of the exit buttons on the game. Termination is done if player chooses yes option.

Failure

- The game cannot be executed unless JRE is installed in the system.
- If high score, saved game or settings records could not be loaded from the associated files, the game will continue to run by displaying error on the panel for the related data.
- In the possible power cut termination, current data will be lost.

- If player forces the game to exit while the system is saving current records or the system is saving score of the player as high score or the system is saving settings at that time, the system could be unable to make these save processes successfully.

3. Subsystem Services

The system is decomposed into three subsystems which are Model, View and Controller subsystems namely. In this part services that are provided by these subsystems are demonstrated.

3.1 Services of the Controller

In this subsystem, we grouped together all of the controller objects to manage all of the game contents. Controller subsystem sits in the most upper part of our system, meaning that it have access to all of the other subsystems and manages them. It provides User Interface subsystem with the notifyChange service. Through this service, the user is able to control the whole system. It is simply a one directional pipeline between User Interface Subsystem and Controller Subsystem. When User Interface Subsystem is interrupted by user, according to the input taken from the user by the help of the action listener in GUI, the state of the program changes. After taking user input in User Interface Subsystem, a proper message is sent to the controller through notifyChange service. After the manager is notified through notifyChange service, it does the necessary changes on the system. For example, let's assume user is in the main menu and the program is in idle state, when he clicks on the "New Game" button, the action listener in menu panel sends the appropriate message through notifyChange service to the controller and the controller starts a new game by making the appropriate changes. Thus, screen changes or the control of the hero while user

is playing the game is realized by the service “notifyChange” that controller subsystem provides.

3.2 Services of the Model

Model classes all together compose the Model subsystem. This subsystem stores application domain knowledge inside of it. Therefore it provides the necessary information of the entity classes to the Controller subsystem. Controller subsystem uses this data to handle the game operations via getModelData service of the Model subsystem. This service provides the Controller to get locations, types and any other necessary information of entity objects. This information is crucial for the game engine to perform all business logic of the game. And also view is going to be derived and updated by this data accordingly.

The relation between Controller and Model subsystems is bidirectional. Controller subsystem changes model data as well. Considering player inputs and game-flow operations Controller subsystem stimulates Model subsystem to change, update itself. This is done by updateModelData service of the Model subsystem. This service is stimulated by Controller and it performs all necessary changes on the model data.

3.3 Services of the View

User Interface Subsystem consists of all panels that user interacts with. So the visuals need to be updated according to changing state. To satisfy that need, user interface subsystem provides a service called updateView. This service is used by Controller subsystem and its main functionality is that Controller subsystem updates the User Interface Subsystem by getting information from the service that Model subsystem provides. For example a wall is destroyed by bomb. Controller subsystem gets the information of destroyed bomb. It

delegates the information to User Interface subsystem by updateView service. Thus, User Interface subsystem will remove the view of wall from panel.