Bilkent University

Department of Computer Engineering

CS319 PROJECT – GROUP #2

Design Report

CS 319 Project: Bombalamasyon

Oğuz Demir – 21201712

Anıl Sert – 21201526

Kaya Yıldırım – 21002071

Kaan Kale – 21000912

**Course Instructor: Uğur DOĞRUSÖZ**

Design Report

Mar 6, 2016

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

# Introduction

## Purpose of the system

Bombalamasyon is a system that aims user to maximize the pleasure of achievement. Bombalamasyon has a user-friendly interface. This helps user to learn how to play easier. Our game consists of 5 level and when user progress, levels are getting harder. Also users can play with other players not just with computer when they select “Multiplayer” game mode from main menu, this feature differentiate from other Bomberman games. Furthermore, when user have no internet connection they can still play or continue our game. In our system there is “High Scores” menu and when players achieve one of the 10 best scores their name and score displayed

## Design goals

In order to compose the system we should clarify the design goals we focused on. These design goals provided in analysis stage from non-functional requirements that we did before design report. Here are described design goals:

### Usability

Easiness in the usage while developing a game may be the most important design goal for making user-friendly, users should not get any difficulties when they are playing. Because of this we will implement user-friendly menus which provide users to find anything they need easily. Furthermore for learning and enjoying the game, level difficulties increase continously from first level to last. This feature increase efficiency and attraction for Bombalamasyon.

### Reliability

Our system is bug-free, for doing reliable system we try to ensure accurate data input and data transformations. We started to do this from early design specification phase through building and testing.

### Modifiability

It is important to add new components while making games, so we can change it with additonal game concepts and this helps us to make modifications on system. These modifications can be based on new game properties, attributes or errors.

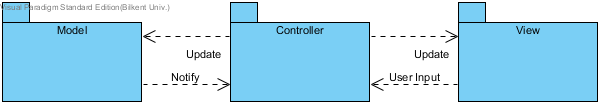
### Scalability

This must be one of our design goals because scalability feature can not be added later. This belongs on software and hardware. While working on software and hardware together in our game we should not make any mistakes in one of them because this cause fail on system.

# Software Architecture

## Subsystem decomposition

We choose to use Model-View-Controller (MVC) architecture to design our game because our system is using many model objects and game view will be updated by the interactions and changes in these model objects. Also controlling them with the controller objects and updating both user interface and models in the same subsystem is beneficial for our system.



In the Model subsystem we will hold all the game’s model objects which are interacted with each other during the play time and these models will be updated with the user input through controllers. This subsystem notifies the controller subsystem about the some properties of the objects and controller subsystem decide what to do according to them. Collisions, map creations, AI behaviour, all managed from the model objects of the system.

Controller subsystem is the brain of our system. The processing of the user interaction with the game take place through the controller subsystem. According to the interactions between models and interactions between user and the models this subsystem updates the view objects. Also according to the game states and user input this subsystem calls the correct view objects through the game.

View subsystem has all of the view components of the system. This is the interaction place of the user and the game and user input taken through these objects. Also, the visual output is shown to the player with the helop of the view subsystem.

## Hardware/software mapping

Bombalamasyon requires Java Runtime Environment (JRE) to be played because it is beveloped by the using Java programming language. Game can be executed with a single executable Java file.

For the I/O requirements computer needs a keyboard, mouse and a monitor to let player interact with the game. The excessive use of input keys is not a problem for the hardware that computer’s have. It requires very little system requirements to be played. Graphical Processing Unit (GPU) is not required to play the game.

## Persistent data management

Files are stored in the hard disk drive. The game keeps names and top ten scores in plain text file in order to display to the player in “High Scores” section. To provide better gaming experience to player, some image and sound files are also used at some parts of the game. When they are needed, these files are read from the disk with their specified directions as parameters. In addition, level data is stored in hard disk drive. There are different game maps for each level in hard drive.

## Access control and security

Bombalamasyon does not implement any user authentication system therefore we do not have any database that stores user credentials. Also, as mentioned earlier (in Hardware / Software Mapping), our game does not require network connection. Therefore, player who has no network connection is able to play the game. So that, there is no restriction or control for access the game. In addition, the game has no user profile, only player names and scores. Therefore, there is not security issues in Bombalamasyon.

## Boundary conditions

**Initialization**

When player execute the .jar file, the game initializes. Player does not have to install the game.

**Termination**

In order to terminate the game, player can click the “Quit Game” in the main menu. When player is playing the game, he/she wants to exit, firstly the player is need to go to “Pause menu” and then click the “Quit Game”.

Game will return to the main menu if all the levels are done. In case of finishing, high scores are updated if score is higher than 10th best score and the game returns to the main menu.

**Error**

If any file (game resources) could not be loaded such as images or sounds, the game starts without these files. If the game does not respond because of other issues such as problem at hardware, software or operating system, player lose his/her current data.

# Subsystem Services

The system is decomposed into 3 parts as model, view,controller and there are 4 main services between these components. The flow is the following: when user give the input, the View takes the input as the boundary component, and it passes the related input to the Controller with Controllers’ service. Controller change the game status in itself and/or the properties of the Model with Model’s service. After that, the Controller ask for an update on the View via View’s service and before updating the current view, the View component can take the game data from Model with the service of the model. At the end, view is updated and changes with the user’s input is reflected on screen.

## Services of the Controller:

**takeUserInput**: This service of the controller is used by view component in order to pass the related user input to change the program status (main menu, paused game, in game etc.) and to control the game( move bomberman or drop bomb ). For example, if the user pauses the game while playing, the view component who has the action listeners for the keys, pass the corresponding input through takeUserInput service of controller for changing state and controller change the game status which is stored in the controller itself to “pausedGame”.

## Services of the View:

**updateView**: This service of the view is used by the controller to change the program display between menus or the reflect the changes in the game map to screen. The status of the program such as mainmenu, ingame etc. is passed to the viewer and if it is ingame, the game data is taken from model component with the help of getGameMapData service of the model component.

## Services of the Model:

**getGameMapData**: This service of the model is used by the view component in order to get the game map data, in other words, positions of the game objects with their types.

**updateGameObjects**: This service of the model is used by controller to manage ingame data with the desire of the user within a time interval and to process CPU controlled objects in that interval.