****

**CS 319 - Object-Oriented Software Engineering**

**Final Report**

***Bombplan***

Group 1

Asena Rana Yozgatlı *21000132*

Berk Yurttaş *21200581*

Mehmet Furkan Şahin *21201385*

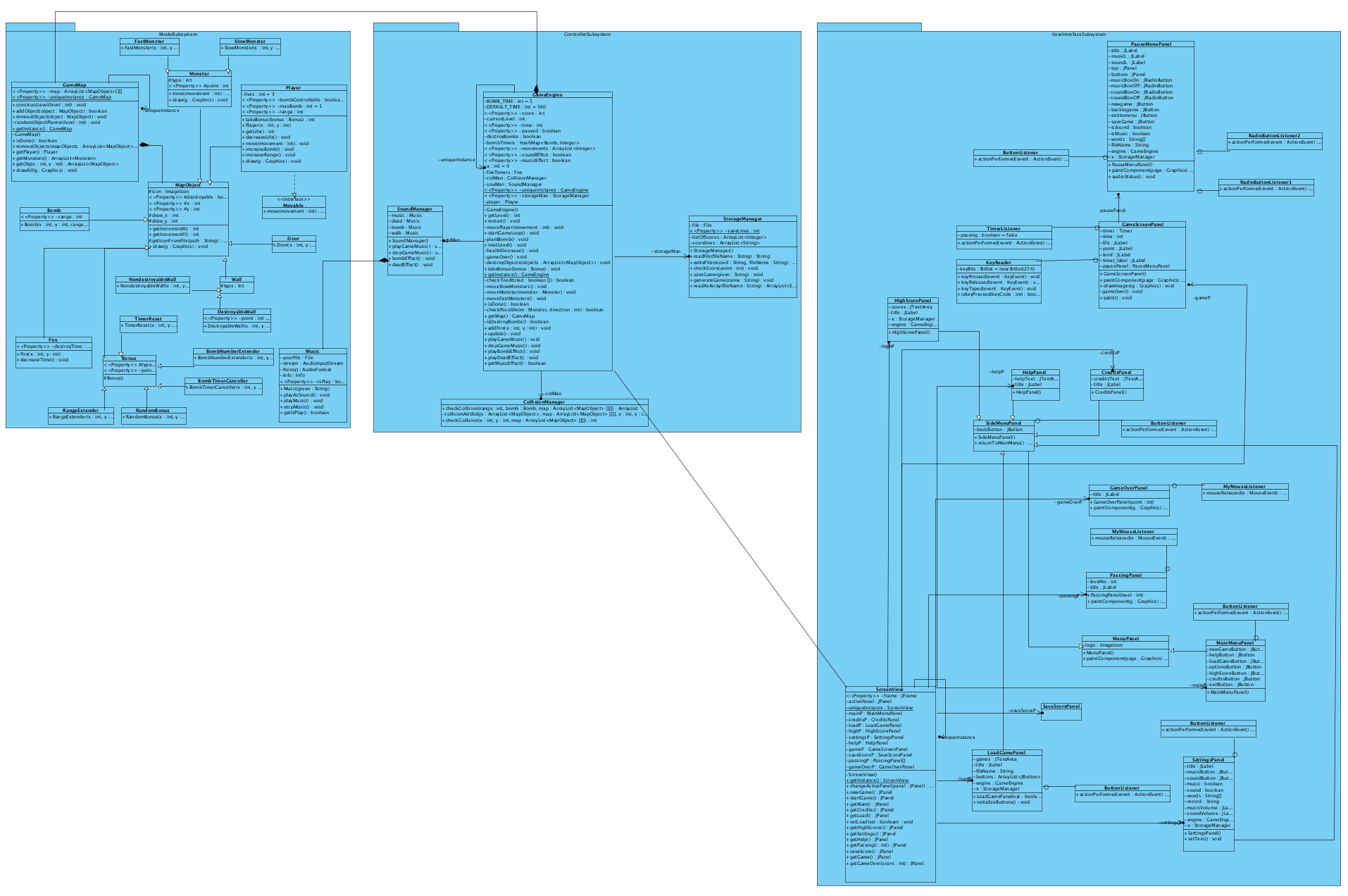
Saner Turhaner *21100475*

**Deadline: 30/04/2016**

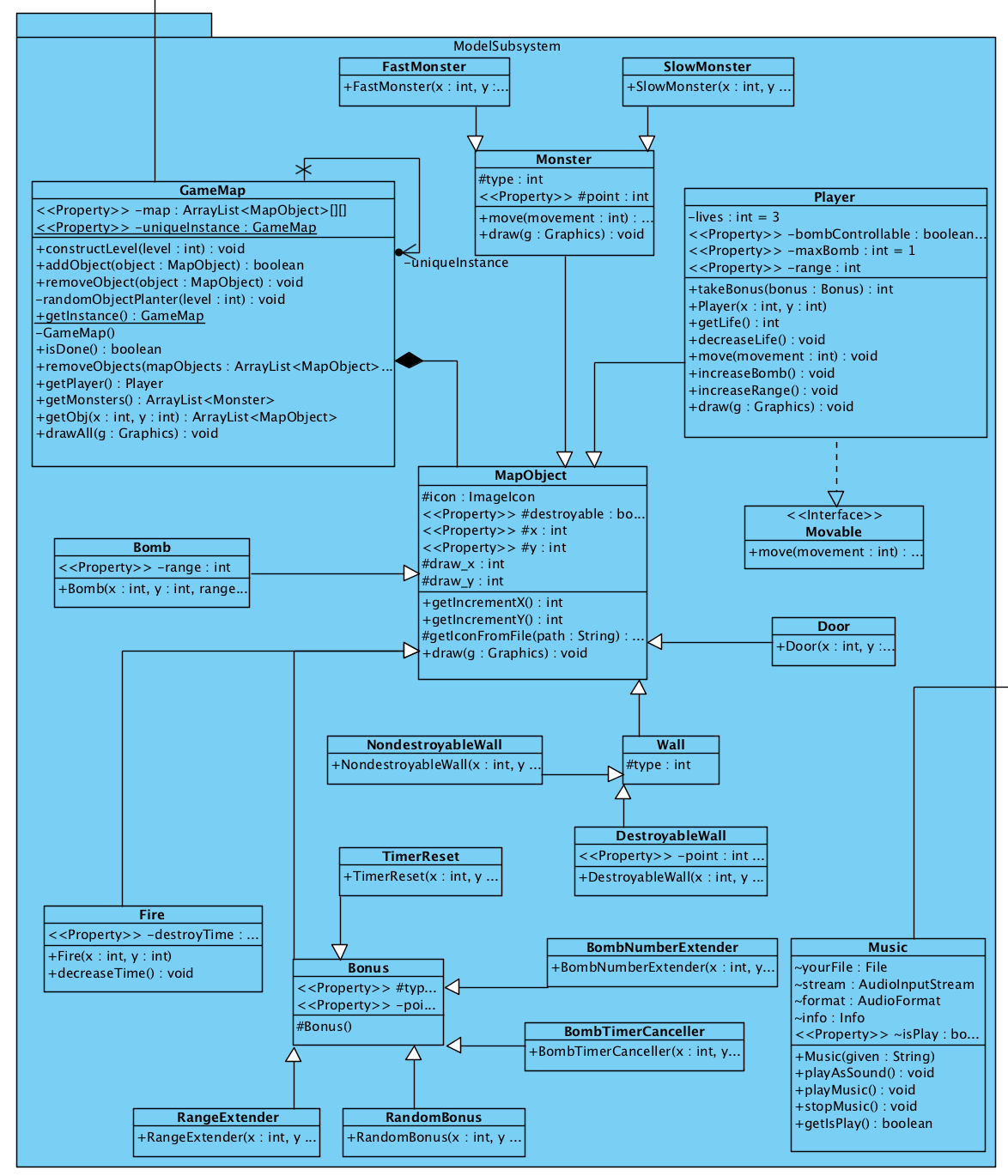
*Course Instructor: Uğur Doğrusöz*

# Changes of the Design

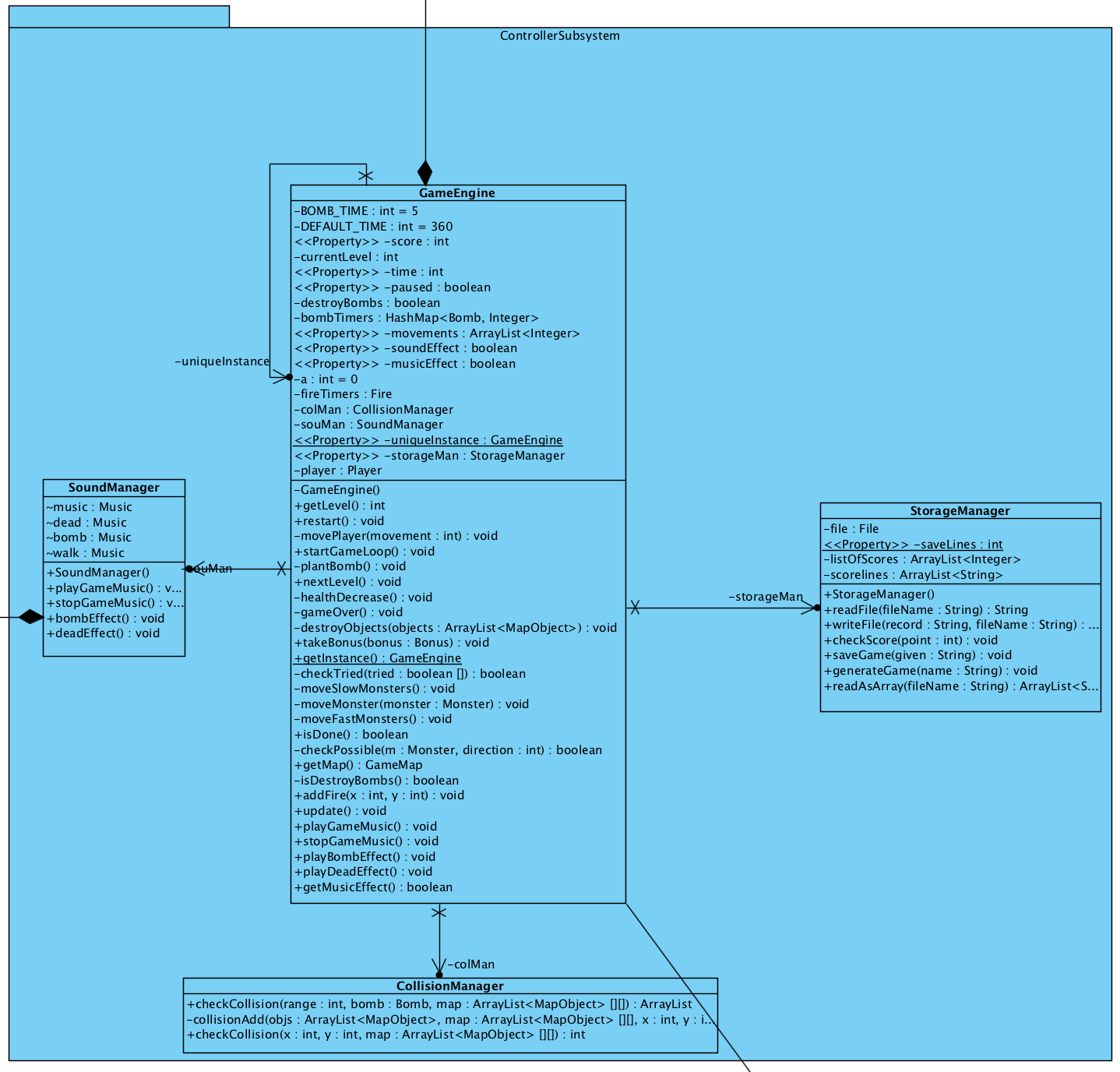
There is no major change on the design of the Bombplan. Design is done using MVC architectural style as stated in the previous report. So there are view, controller and model parts of the implementation which are forming packages. Façade pattern is applied to the code since these subsystems are very distinct. For the each façade classes which are namely ‘ScreenView’, ‘GameEngine’ and ‘GameMap’ singleton pattern is applied so that instances of these objects are created just once. Minor changes are generally due to extension of class features and methods. In addition to these minor changes some inner classes are added inside of the view subsystem classes. These inner classes extend Java listener classes and they are responsible for the actions taking place on the panels and the key inputs. Apart from that there are two new model classes as follows; “Music” which deals with the sounds and “Fire” which deals with the fire animation after the bomb explosion. Additionally, there are 3 new panels in the User Interface subsystem as GameOver Panel that is show when the game is over, Passing Panel which is used between the consecutive game levels, SaveScore Panel that is used to save the score of the played game. General Structure of the class diagram can be seen in the Figure 1 as they are grouped in the packages that they belong to.



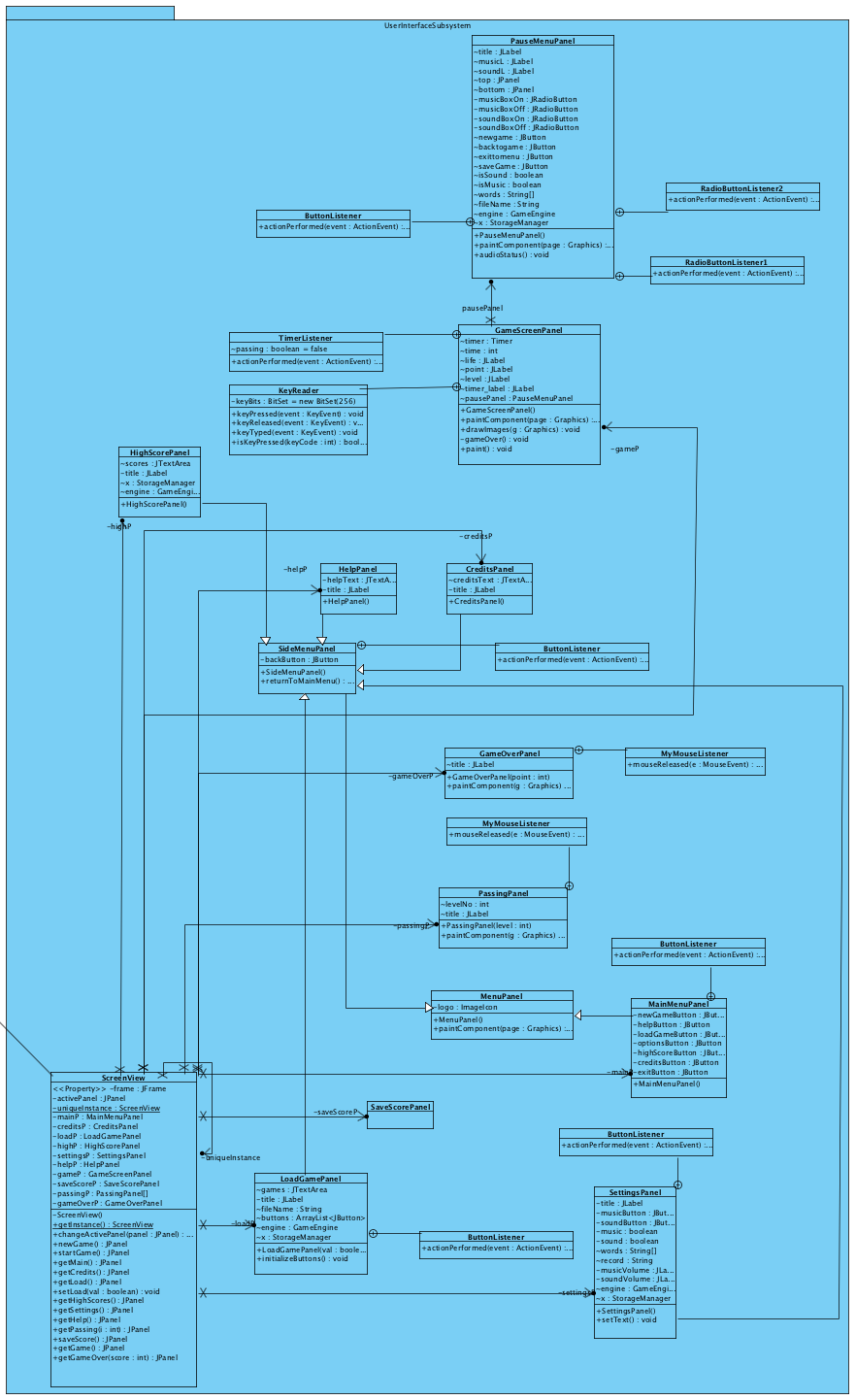
**Figure 1:** From left to right: Model Subsystem, Controller Subsystem, UserInterface Subsystem. (Turn the page 90 degree counter clockwise.)



**Figure 2:** Model Subsystem in detail



**Figure 3:** Controller Subsystem in detail



**Figure 4:** User Interface Subsystem in detail.

The most important design choice made was to carry the timer from Controller to the User Interface subsystem. We realized that we need a notification system to make the user interface subsystem to be aware of the changes made by the engine in the model classes. This can be implemented with observer patter. However, we chose to have engine as an instance of the user interface subsystem and with the help of singleton pattern we had only one engine. In each clock time, we used update method of Engine and it checked if there is a requested movement from user. In this new structure, User Interface subsystem is at the very top of the hierarchy. Controller sit in the middle and manage the model classes according to the given inputs from User Interface subsystem.

We also changed the save&load structure of the application. In the both design and analysis phases, we thought that it would be nice and easier if we implement our own save and load functions. However, we realized that we need to put an effort to make a proper parsing on the objects to save them in txt files. Instead of wasting the engineering power to such an issue, we decided on to use an open source library to handle save&load. We use xstream-1.4.9 and it works with xml files.

# Complications during the Implementation

During the progress of the implementation there was a complication in the implementation of sound effects and the music of the game. Libraries used for the sounds were just working for the Windows operating systems. When we tried to compile source files on the MAC OS X system, sounds did not work and even it did not run the game. After a proper search on the internet, we figured it out that the reason was because of the integration of java compiler and Mac OS X El Capitan. It is resolved with a proper re-install of the java compiler.

The most frustrating bug during the implementation was related to the GameMap object. We divided the work mainly to 3 parts and one was responsible from the engine part when the other 2 were responsible from the model classes. Our GameMap has a three dimensional structure that has 13 box height, 15 box length and depth for each box depending on the state of the application. Although we communicate properly and try to ask any suspicious point, we messed up with the indexing of the map. All 3 coder imagined the other ones’ map structure differently and in the result, some objects’ coordinates were in x-y-z coordinate system, and some others were in y-x-z. This bug is detected during the testing of the engine and fixed by Mehmet Furkan Sahin.

Another complication that we tried to handle was the conflict of the xstream library and the sound classes. Since xstream library is not capable of saving the music files in to xml files, it simply skips the sound part when we try to save a whole engine object. Saving is done successfully by this skipping. However, since engine is a whole object with its sound part, when we try to load a saved engine object, we encounter some errors. We still could not resolve the problem and are working on it.

# User’s Guide

Bombplan is the modern type of the first Bomberman games and it is mainly a 2-D strategy game where player tries to destroy all the monsters on the map and find a door to way out.

* *What makes Bombplan charming and interesting?*

Simply the reason is easy to play and easy to learn features besides the pleasure of the game by making strategies on a 2-D map. Bombplan has its own strategy and its own game play, thus it is very original considering its size. There is no complicated rules or things to learn, thus it has something unique that makes it very enjoyable.

* *What is different in the Bombplan comparing other Bomberman games?*

It is very similar to the other games but there are extra bonus types in this game. These are random type and reset timer bonuses.

**Menu Guidance:** Player can switches menus by clicking buttons using mouse. In order to begin a new game player can choose “Start Game” option from the main menu. It is possible to load a game from the “Load Game” option. For adjusting sound effect and music volumes there is a “Options” option. In addition to these there are also “Help” and “Credits”, which are game rules and instructions and programmers of the game respectively. User can see the high scores table by clicking “Highscores” from the main menu.

**Movement:** Bombplan game can be played with a basic keyboard which is needed to control the hero and a simple mouse to use menus. Hero can move through 4 main directions by using “arrow keys”. It can plant a bomb to its current location by pressing “space” button. Player can also control the bomb explosion time if it takes related bonus, and in order to control the bomb s/he can use “D” button from the keyboard to explode the bomb.

**Combat:** Hero can plant a bomb and his only skill is just this. When bomb is planted it explodes after a short period of time if player does not take any bonus affecting the bomb explosion time. When bomb explodes it destroys anything that is near of it except door and non-destroyable walls. Thus hero can make a way by exploding bombs and it can destroy monsters by using it. If hero itself is in range of the bomb he will die, and if hero contacts with any type of monster walking around the map he will die also. In addition to these, if time is up than game is over.

**Aim of the game:** Aim of the game is finding a way out for each level. But in order to do that first hero must destroy all the monsters on the map. If hero can pass all the three levels successfully game finishes otherwise s/he be stuck in the map and spend all his lives.

**High score:** Even if the player cannot pass all levels, if he or she could be able to collect points that is among the top ten scores then its’ score will be recorded as a high score. These scores will be seen in the high score table from the main menu.

# Status of the Current System and Implementation