 **İhsan Doğramacı Bilkent University**

**Departmant of Computer Engineering**

**CS319 Object-Oriented Software Engineering**

**Project Design Report**

**Group A**

***Tower Power!***

**Fall 2014**

**Özgür Öney**

**21101821**

**Section 3**

**Serkan Demirci**

**21201619**

**Section 3**

**Ahmet Küçük**

**21001069**

**Section 3**

Contents

[Purpose of the system 4](#_Toc405148647)

[Design Goals 4](#_Toc405148648)

[Trade-offs 4](#_Toc405148649)

[2-Software Architecture 5](#_Toc405148650)

[a-) Subsystem Decomposition 5](#_Toc405148651)

[b-) Hardware-Software Mapping 6](#_Toc405148652)

[C-) Persistent Data Management 7](#_Toc405148653)

[d-) Access Control & Security 8](#_Toc405148654)

[E-) Boundary Conditions 8](#_Toc405148655)

[3- Subsystem Services 8](#_Toc405148656)

[4-Low Level Design 8](#_Toc405148657)

[A-)Object design trade-offs 8](#_Toc405148658)

[B-) Final object design 9](#_Toc405148659)

[C-) Packages 11](#_Toc405148660)

[D-) Class Interfaces 11](#_Toc405148661)

[i-)View 11](#_Toc405148662)

1-Introduction

### Purpose of the system

Purpose of the game Tower Power is simply creating an enjoyable environment for end user who differs in age. Creating a challenging environment that keeps end user in game by adding some specification such as transferring gold from one level to each other and levels that have various difficulties is another purpose of the game.

### Design Goals

Subheading “Design Goals” can be divided into 2 different headings: Non-functional requirements and trade-off’s of the system.

* Adaptability Criteria

Game is implemented in Java, which provides adaptability for all operating systems and environments. Game’s adaptability is important in order to get as much player as possible.

* User-friendliness Criteria

Game Tower Power should have user friendly user interface and gaming environment. Game is created for all ages, so providing an easy-to-learn interface should be considered.

* Robustness Criteria

Game is created to make people enjoyed, as indeed. So, in order to prevent some fatal errors, hesitations and bugs; game should be designed as robust.

### Trade-offs

* **Time vs Graphics**

We have approximately 1 month to develop game as a Greenfield project, so time is limited. On the other hand, game should be created with graphics as well as possible. What we have done is drawing an acceptable line between these two and developing the game by taking this line into consideration.

* **Rapid Development vs Functionality**

Since limited time forces us to develop the game as fast as possible, we have to sacrifice some functionality. Our main goal in here is rapidly developing a unique game by sticking mainstream rules of a tower defense game as far as possible.

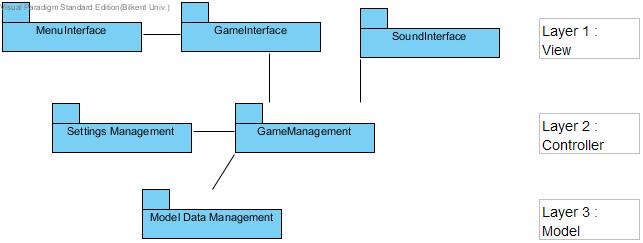
* **Scalability vs High Performance**

We want our game to be played in every single machine that provides Java environment. On the other hand, so by considering this; adding elements to game that requires high level hardware seems

# 2-Software Architecture

## a-) Subsystem Decomposition

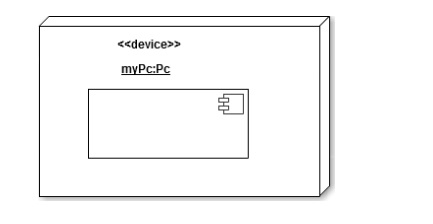
The system is separated into three packages which are Logic Management, UI Management and Modeling Management. Logic Management package contains MenuController and GameController systems. However, UI Management subsystem consists of InGameManagement, SoundManagement and MenuManagement systems. Lastly, Modeling Management subsystem contains DAO and GameEntity systems. This packages are consists of parts of system which represent subsystem decomposition. Subsystem decomposition is shown at Figure below.



# b-) Hardware-Software Mapping

Implementation language of the game is Java, so in order to play the game; JDK should be installed on the machine of the user. Mouse is the only device that user needs while playing game. The user use mouse to select any menu options, to decide which level will be played level, to add towers & traps.

The deployment diagram consists of only one virtual machine which is computer and it has a software whose name is Tower Power!. Deployment diagram is shown at Figure 1.



## C-) Persistent Data Management

In implementation of the game, 2 different types of data will be used: Static data and dynamic data. Static data will be textures and sound file that will be used by the game which won’t be changed by the game. As dynamic data, game can change the data in the execution of the system. Game will be store the player data which consists of settings for the player and also the level data that player can play. Level data will be updated in the end of the levels and the settings will be changed when user change settings and exit the settings menu. Also, the activity log data will be stored as dynamic data when game crashes. This activity log will store the major events in the game.

## d-) Access Control & Security

In the game, all users anonymous, which means that the game does not keep any personal information about users and so there is no hierarchy among users. Only data that we keep is a player nick name for the game, which will be kept on the device’s itself. Since there is only one type of actor that interacts with our game, we don’t have any access control management among actors. All players will be able to reach all reachable functions. However, we have dynamic access control, which is applied to all users. For example, if user is only completed level 1, he will be able to access only level 1 and level 2 objects but not level 3 and other levels beyond level 3. We will be controlling user access to each level dynamically.

Since the game doesn’t keep any fragile information about player that might be corrupted such as credit card information or SSN, we don’t have any security protection for player. Game code and resources are also public so we don’t have any security precaution for them too. However, a corruption in the files of the game can cause crash the game but there is nothing to do about it.

## E-) Boundary Conditions

* **Initialization Boundary Condition**

The game will be initialized by running a simple .jar file. Since there is no extension like .exe; no installation is required.

* **Termination Boundary Condition**

Quitting game is possible by clicking “Quit Game” button on Main Menu. However, if unexpected situation occurs and game crashes, there will be a pop-up screen that informs people about game’s crashing. Also an activity log will record the important game events. If a fatal exception occurs in game, this activity log will be saved in a file so that player can get help by providing this log file to developers.

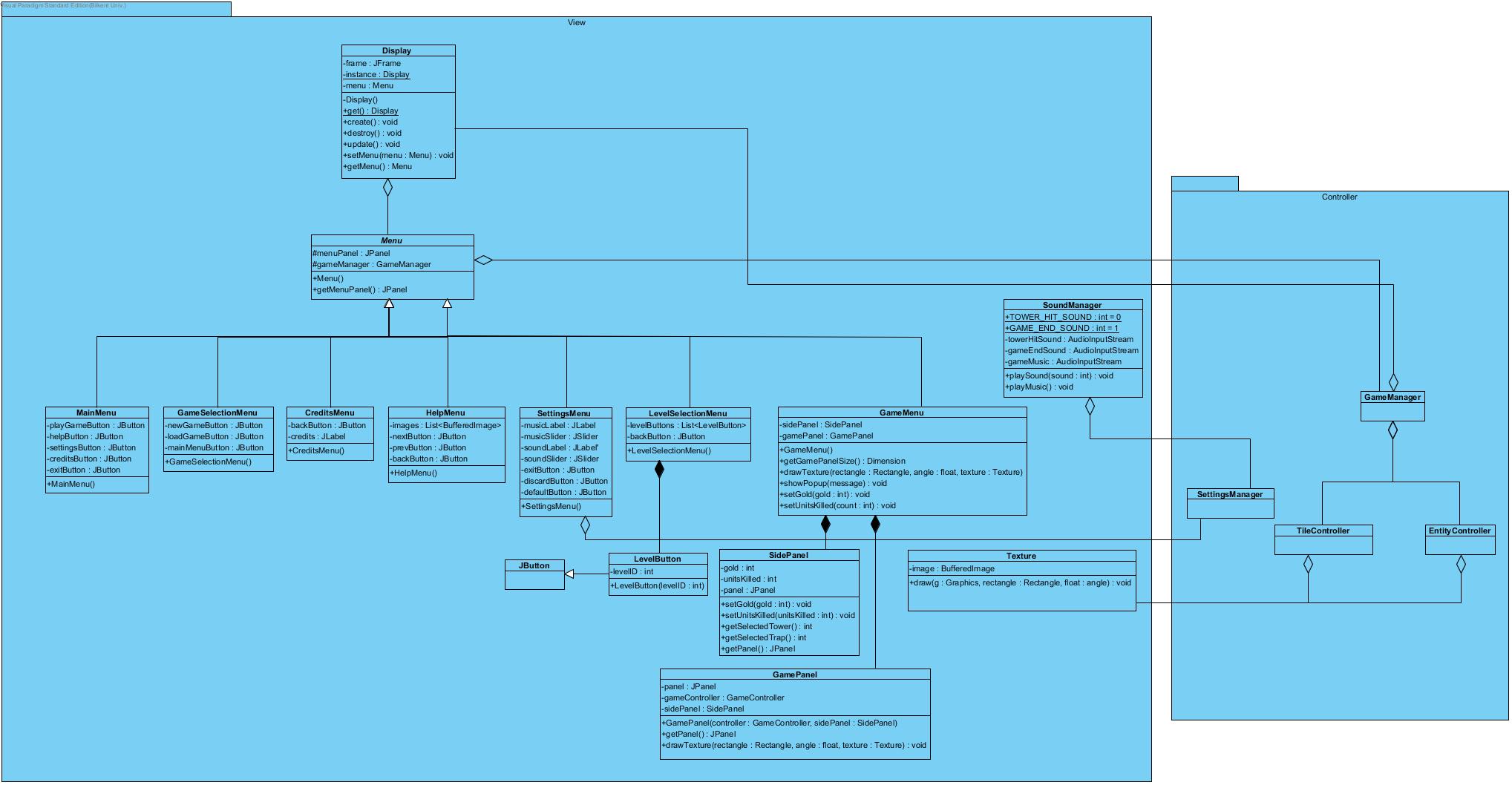
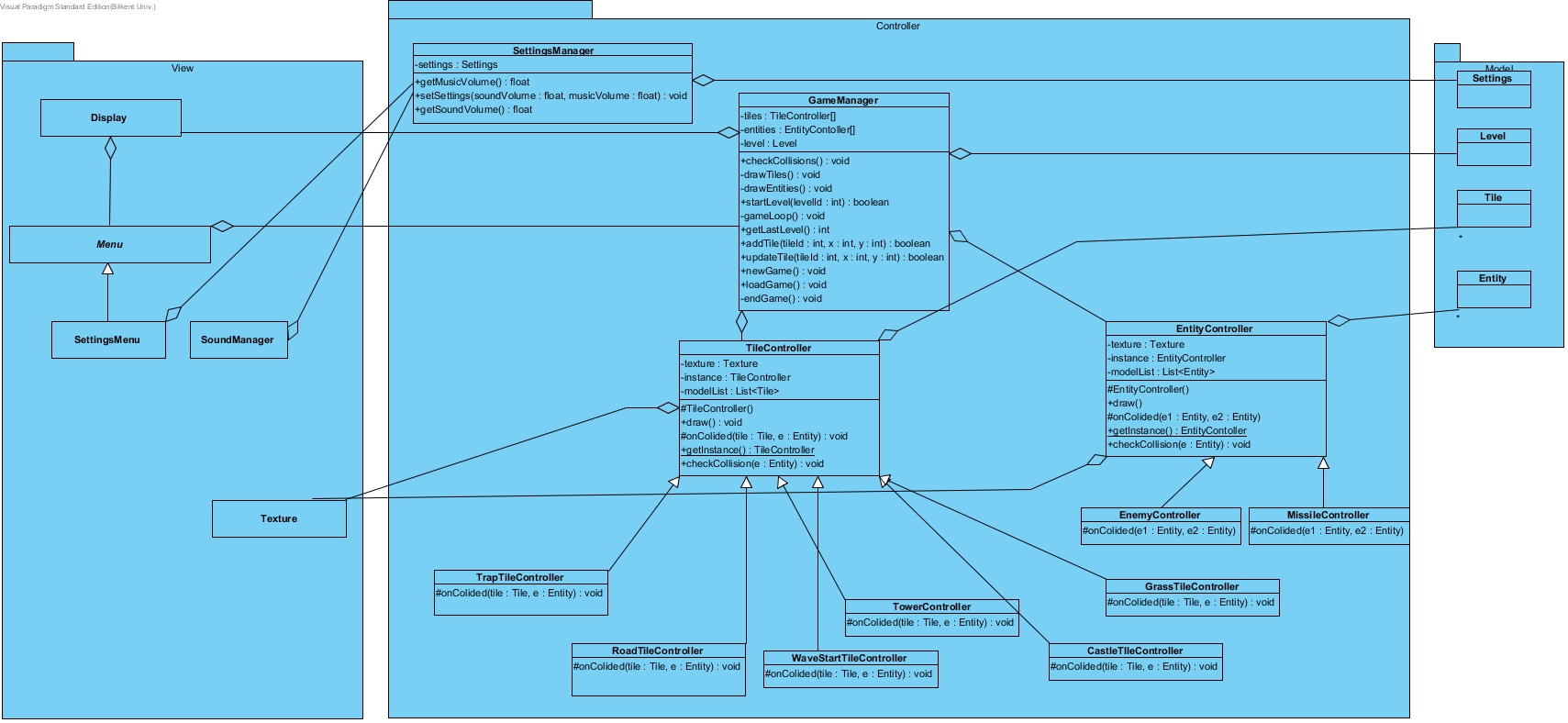
# 3- Subsystem Services

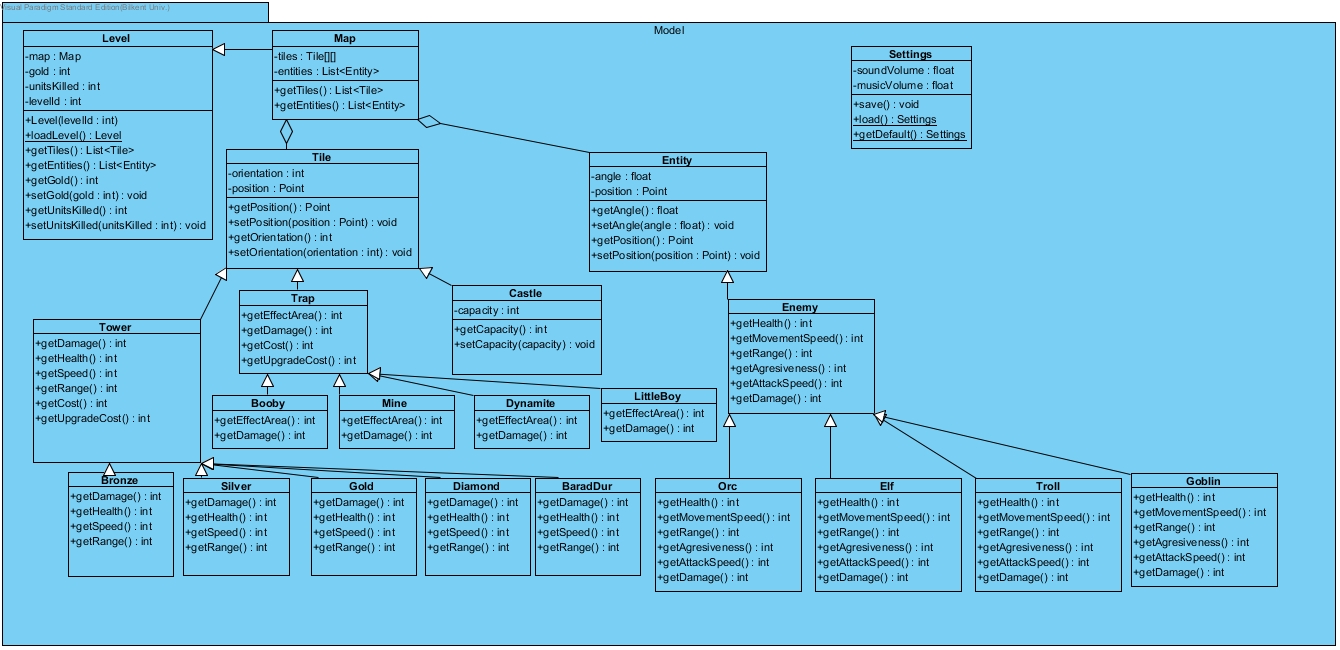
# 4-Low Level Design

## A-)Object design trade-offs

* Efficiency v. Portability

Since our system uses MVC architecture, the system is portable. But due to the more method calls caused by the architecture style, the system is less efficient.

B-) Final object design 



## C-) Packages

There are 3 packages in the program:

* View

View is responsible for user interaction.

* Controller

Controller package responsible for game logic.

* Model

Model is responsible for data management.

## D-) Class Interfaces

### i-)View

* Display

Represents display frame of the game. It uses singleton design pattern so that only one instance of the display can be active at the same time.

**get():Display**

Returns the instance of the display. If the instance does not exist creates a display instance.

**create():void**

Makes the display visible.

**destroy():void**

Disposes the instance of the display and hides the frame.

u**pdate():void**

Updates the contents of the display.

**setMenu(Menu):void**

Sets the menu that to be displayed in the display.

**getMenu():Menu**

Returns the menu that is displayed in the Display object.

* Menu

Represents the content that can be displayed in the frame of the display object.

**getMenuPanel():void**

Returns the menu panel.

* MainMenu
* GameSelectionMenu
* CreditsMenu

Shows the credits.

* HelpMenu

Help menu shows some images that contain the instructions of playing game to help user to understand the game.

* SettingsMenu

Menu that used for changing game settings.

* LevelSelectionMenu

Menu that in which player can select which level he wants to play.

* LevelButton

A special button that extends from JButton that has different rendering.

* GameMenu

Game menu displays in game user interface components of the game.

**drawTexture(rectangle : Rectangle, angle : float, texture : Texture):void**

Draws the spesified texture into game panel.

Texture is the texture to be drawn into screen.

Rectangle is the position and size of the texture.

Angle is the angle of the texture.

**showPopup(str:String) :void**

Shows a popup message.

Str is the message to be shown in the popup.

**setGold(amount:int) :void**

Sets the gold value in the user interface.

Amount is the gold value that is to be displayed on the screen.

**setUnitsKilled(count:int) :void**

Sets the killed unit count value in the user interface.

Count is the value that will be displayed in the user interface.

* SidePanel

Side panel of the in game user interface. Player can choose which tower or traps to be added in game and it shows gold value and killed unit count.

* GamePanel
* Texture

Texture is the image that is drawn into screen.

**draw(g : Graphics, rectangle : Rectangle, float : angle) : void**

Draws the texture into graphics using Rectangle and the angle.

G is the graphics that texture is drawn into.

Rectangle is the position and size of the texture that to be drawn.

Angle is the angle of the texture.

* SoundManager

Manages in game sound and music. Plays music and sound when requested.

**playSound(sound : int) : void**

Plays a sound using Clip object.

Sound represents which sound will be played by the manager.

**playMusic() : void**

Plays a music that repeats until game is closed.