 Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering Project

CS 319 Project: Civilizational Wars

Design Report

Project members:

• Fuad Aghazada \_ 21503691

• Seyfullah Yamanoğlu \_ 21400697

• Bayram Muradov \_ 21503664

• Berk Erzin \_ 21201516

Supervisor: Uğur Doğrusöz

Progress: October 16, 2017

Contents

1. Introduction 1

1.1. Purpose System 1

1.2 Design Goals 2

1.3 Design Tradeoffs 3

**2. Software Architecture 4**

**2.1 Subsystem Decomposition 4**

**2.2 Hardware/Software mapping 6**

**2.3 Persistent Data Management 6**

**2.4 Access Control & Security 6**

**2.5 Boundary Conditions 7**

**3 Subsystem Services 8**

**4 Low-level design 11**

**4.1 Primary Class Description 11**

**4.2 Final object design 15**

**4.3 Packages 16**

**5 Glossary & References**

Design Report

CS 319 Project: Civilizational Wars

# Introduction

By writing this report as a group we aimed to convey some general and detailed information about the analysis of this game, specifically, functional, non-functional requirements, and system model diagram.

# Purpose of the system

This game will be an adventure game, in which there will be different maps to play in order to make the game interesting to the players. Game interface will be easy to adapt for the players within all groups of ages. The game controls will not only be easy to use as default, but also will be modifiable in the settings of the game. In terms of the interests of the players, the game will have 3 different difficulty types: easy, medium, hard - which could help the player to test his/her gaming skills in different types of difficulties. Another purpose behind the designing this kind of game is also experience the players, especially the ones with same generation with us, the nostalgic Atari games like Contra (in terms of map and fighting).

* 1. Design Goals

**Efficiency -** Every player would like to play the games, which are quite responsive and do not have so many performance issues, which are irritating. Considering this fact, we will make our game sufficiently efficient in order to be able to obtain the attraction of the players. So as to achieve this goal, we will try to handle our limited memory properly so that it is minimally used, and minimize the execution time of the main operations, which play the most crucial roles on the proper flowing of the game. For example, in order to check the collisions between the different game objects, we need to create a class, in which we will have some functions, control all the computations done efficiently, otherwise, it could lead to some bugs prevent the game flowing properly.

**User-friendliness -** In order not to make the player confused when he/she launches the game, we need to design the User Interface easy to control and use. In terms of the comfort of player, some settings, like activating music, changing difficulty modes, adjusting controls and so on will be available. However, comfort in the controls and the interface does mean the game be extremely easy that the player will be bored while playing the game.

**Adaptability -** So as to be able to run the game in different platforms, it could be a proper decision to implement the project in Java programming language, which is also advantageous in terms of the object-oriented approach. The only thing is need to launch the game in a platform is to install proper JRE [1] - Java Runtime Environment, which is compatible with almost all widespread operating systems. That is why we decided to choose Java among the languages like C++, C# and other languages which can provide us some different advantages in terms of performance and security.

**Modifiability** - The very first release of the game will contain only 3 different levels which is not sufficiently enough for a player to entertain, since it will finish soon in terms of the time. This is the reason why we may need to add new maps, characters and levels to the game. In order to achieve this, with the advantages of the object-oriented approach, our design and implementation needs to be modified and new features and game states should be added So our project should also have modifiability as a design goal.

**1.3 Design Tradeoffs**

**Functionality - Usability -** Complicated and complex game system is not a desirable specification for the comfort of the users, so we decided to design a plain usability, which requires basic functionalities. This tradeoff creates easiness at the adapting of the player to the game system so that the players can enjoy their time by playing the game.

**Time - Memory efficiency -** In order to create a game system with a high performance, we need take efficiency enhancements into account, in terms of time and space. However, we will consider that the memory of the today’s average computer is sufficiently enough to utilize, so we will need to sacrifice the memory usage to perform the operations and functions in a considerably less amount of time. Shortly, we will use a time efficient algorithms so as to cover the downsides of the using extra space in the memory for the game objects.

1. **Software Architecture**
   1. Subsystem Decomposition

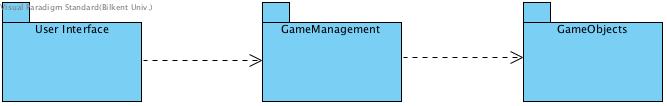


Figure 1. Subsystem decomposition

We made up our mind to use MVC [2] (Model-View-Control) as an architectural style, because it will not be too complex game system and by using this style, we will be able to make changes on the interface while keeping the core of the system and models the same. Using this architectural style, we can divide our system to mainly 3 subsystems as described in Figure 1: GUI, in which there will be rendering system for the game states (menu, pause, settings and etc.), GameManagement, will contain the game logic, input, data managements, GameObjects, will keep the game entities with their properties. More details for the subsystem are described in Figure 2 below.

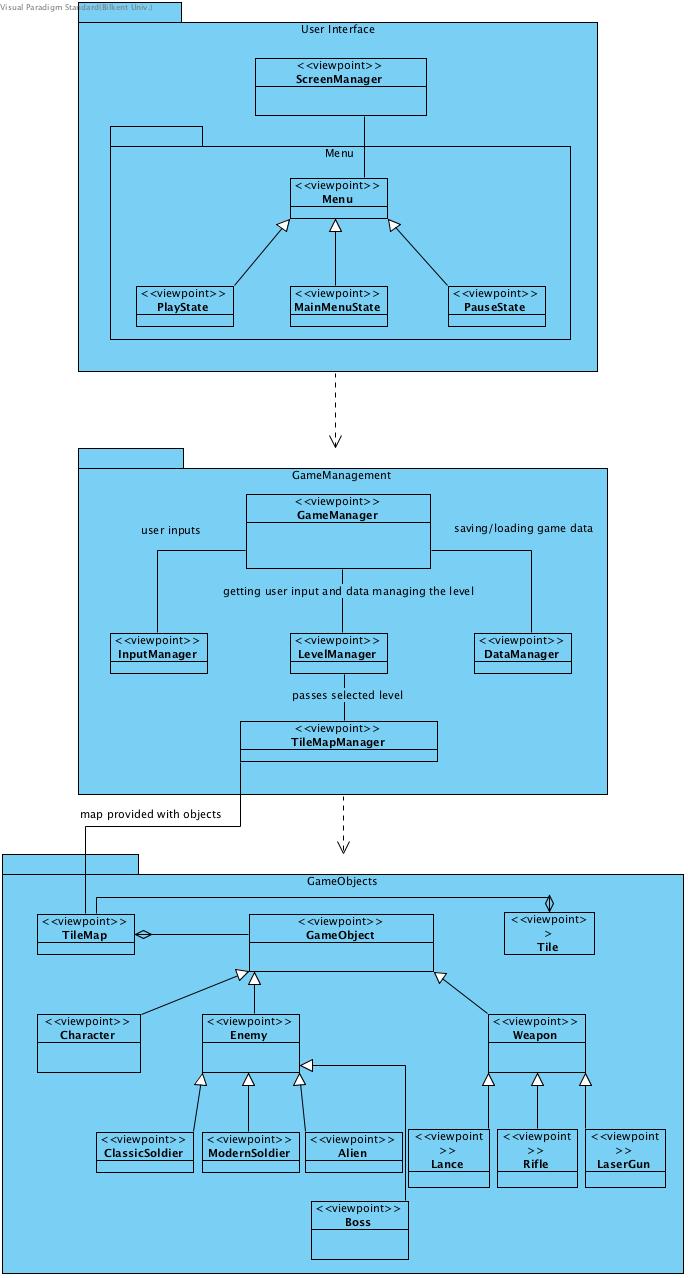


Figure 2. Some details for the subsystems

* 1. Hardware/Software Mapping

As hardware requirements, our game will need a keyboard and mouse in order to create an interaction between the user and the game. The player will use keys from the keyboard to move and fight, and mouse for aiming by some weapons. In order to run the game, the machine in which the game is launched and played should not have extremely high requirements rather than nowadays’ standards.

* 1. Persistent Data Management

Since our game will have features for saving and loading the game, we will need to save the current time data for game. Data for the game will not be very large and will not be concurrently used by different users - the game will be installed to each machine with the necessary files in order to properly run the program. That is the reason why, we decided to use a simple file system so that it will save the data for the game objects, maps and the levels in proper formats to some files, so we can access saved data by loading the game from those files.

* 1. Access Control & Security

Our game will not have a feature of login authentication, because it will be a single player game, and as mentioned above, it will use a simple file system to store the necessary data for the game. Nevertheless, we will need proper access modifiers for the data of the game objects so that it will not affect the security of data flow of the program. Additionally, we need to protect the file, where the saved game data is stored. In order, not to be modified manually by the players, we will need to encrypt the game data in a way that the data is not obviously readable by the users.

Object data serialization would also be another solution for this security problem in a way that, when the game is saved, the necessary data, which will be saved, could be represented as sequence of bytes, which contains every information about the object - object type, data type and etc. After writing the serialized data into the file, when the game is loaded it will get the data by deserializing it. In Java Virtual Machine, the data can be serialized in a platform, and can be deserialized in another platform, which again makes the Java programming language much more adaptable.

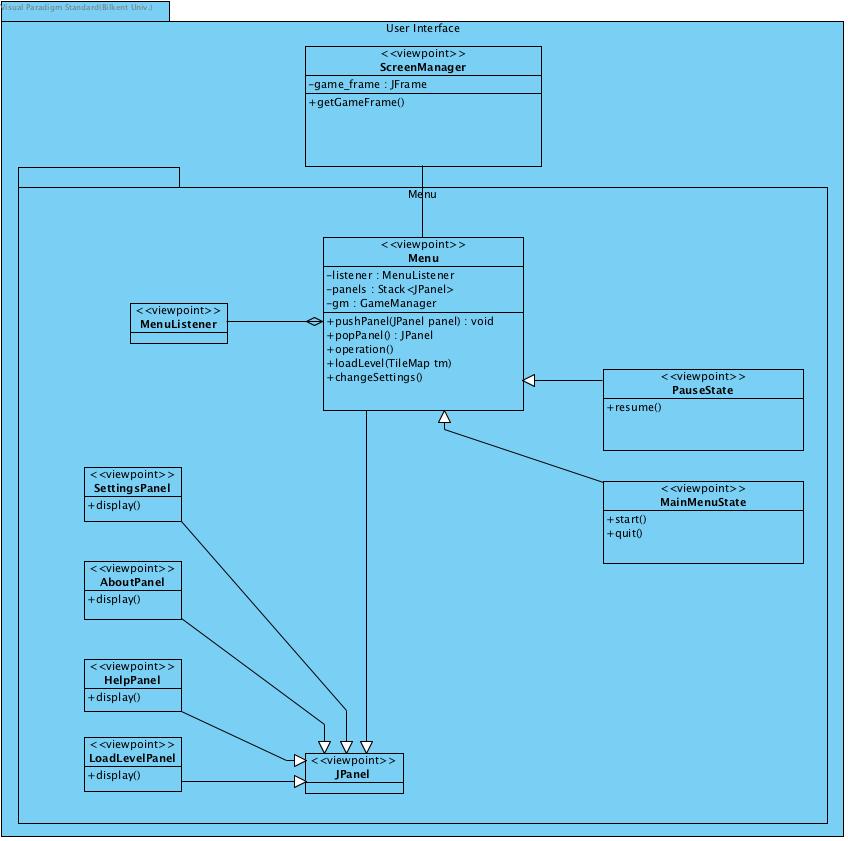
* 1. Boundary Conditions

When the game is launched, necessary data will be initialized and will be waiting for the user input. When the gameplay starts, when the game is loaded, it will initialize the game object with the obtained data coming from the file. However, if there is some format problems in the file, which have been created by the modifications of the players, the program will not allow the user to load the game by throwing an exception. When all the levels have been passed, the player will be congratulated and brought to the main menu to create a chance for trying the levels in different difficulty modes.

1. **Subsystem services**

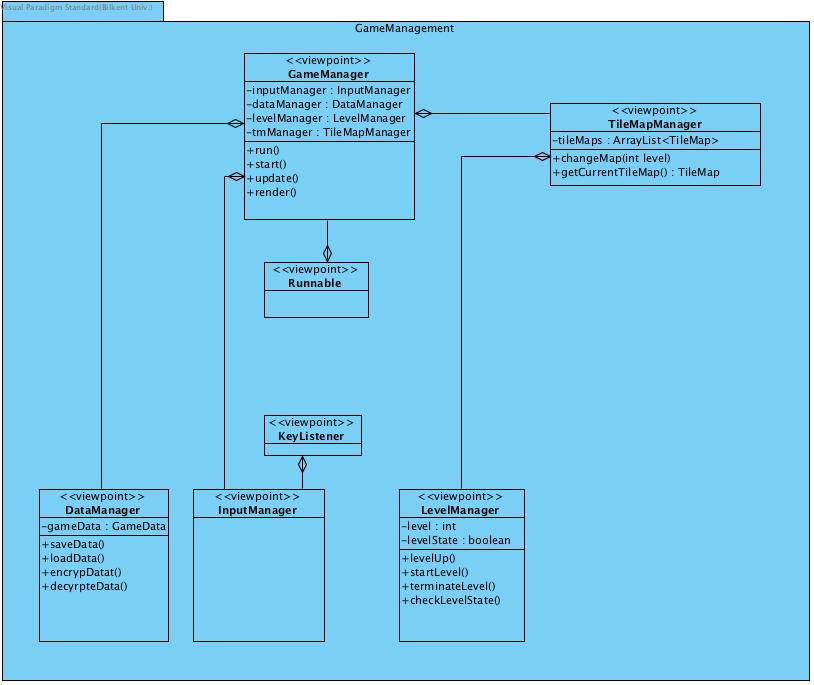
User Interface subsystem

Graphics of the game will be visualized in User Interface subsystem. This subsystem will represent the View part of the MVC design pattern.



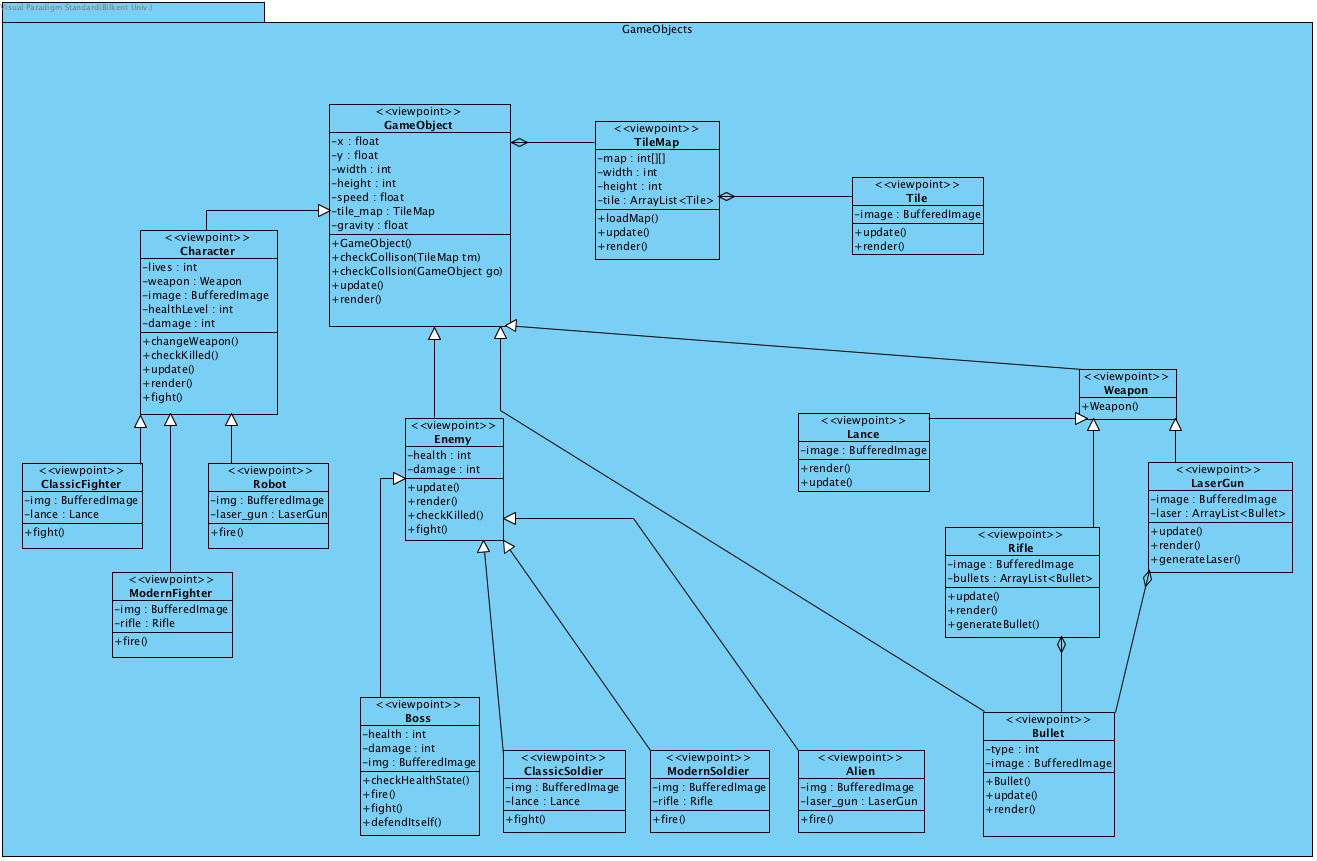
GameManagement subsystem

Inputs and game data will be managed in this subsystem. This subsystem will represent the Controller part of the MVC design pattern.



GameObjects subsystem

Object of the game will be designed in this subsystem. This subsystem will represent Model part of the MVC design pattern.



1. **Low-level Design**
   1. Primary Class descriptions

**Menu**

This class is to generate a basic game menu, which contains some options like play game, load game, view the settings, about and help section. The class will have some attributes which define its properties, which are described below:

*Properties*

* *MenuListener mouseListener*: in order to create an interaction between user and the system, we need to create a listener to handle the input from the user, in this case it will implement KeyListener, because we will primarily use the inputs from the keyboard.
* *Stack<JPanel> panels*: The system will be a multi-view application so that we will need to have different panels, in which there are different views, such as settings, help and about sections. To be able to control these views (panel) we will have to keep them in a data structure so that we will be able to go back and forth among the views – in this case we considered stack as a good choice.
* *GameManeger gameManager*: with this instance, Menu will be able to have an access to the other manager classes like input, data and map managers

*Methods*

* *pushPanel(), popPanel():* in order to execute the operations of the stack.
* *loadLevel(TileMap tm):*According to the user input from the menu options, the game will load the chosen map of the game.
* *changeSettings():* This will change the settings of the game according to the user input and apply them into the system.

**GameManager**

This class will have a role of managing the whole system so that this will be the controller of the system.

*Properties*

* *InputManager inputManager:* the user inputs will be managed using this class so that it will get the user input and will forward it to the other managers classes to utilize it in order to perform an action.
* *DataManager dataManager:* To save and load the game we need to have a file system to save and load the game data. This class will handle these operations (save & load).
* *TileMapManager tmManager:* This instance will also keep another instance (*LevelManager*), and using this instance, it will choose the proper map either directly from the user input or flow of the game (after level 1, level2 starts). It will also keep an *ArrayList* to keep the tile maps in it. It will update the map using the *changeMap()* function and will be able to return the current map by *getCurrentMap() method.*

*Methods*

* *Run() :* to run the game loop which will update using *update()* and render the game properties and objects using *render()* function*.*

*- Start():* This method will start the thread (running different sections of code simultaneously on different processors) of the game

**GameObject**

This abstract class will represent the minimal properties like positions, dimension, and other properties for the nearly all objects of the game like game character, enemies, weapons and so on.

*Properties*

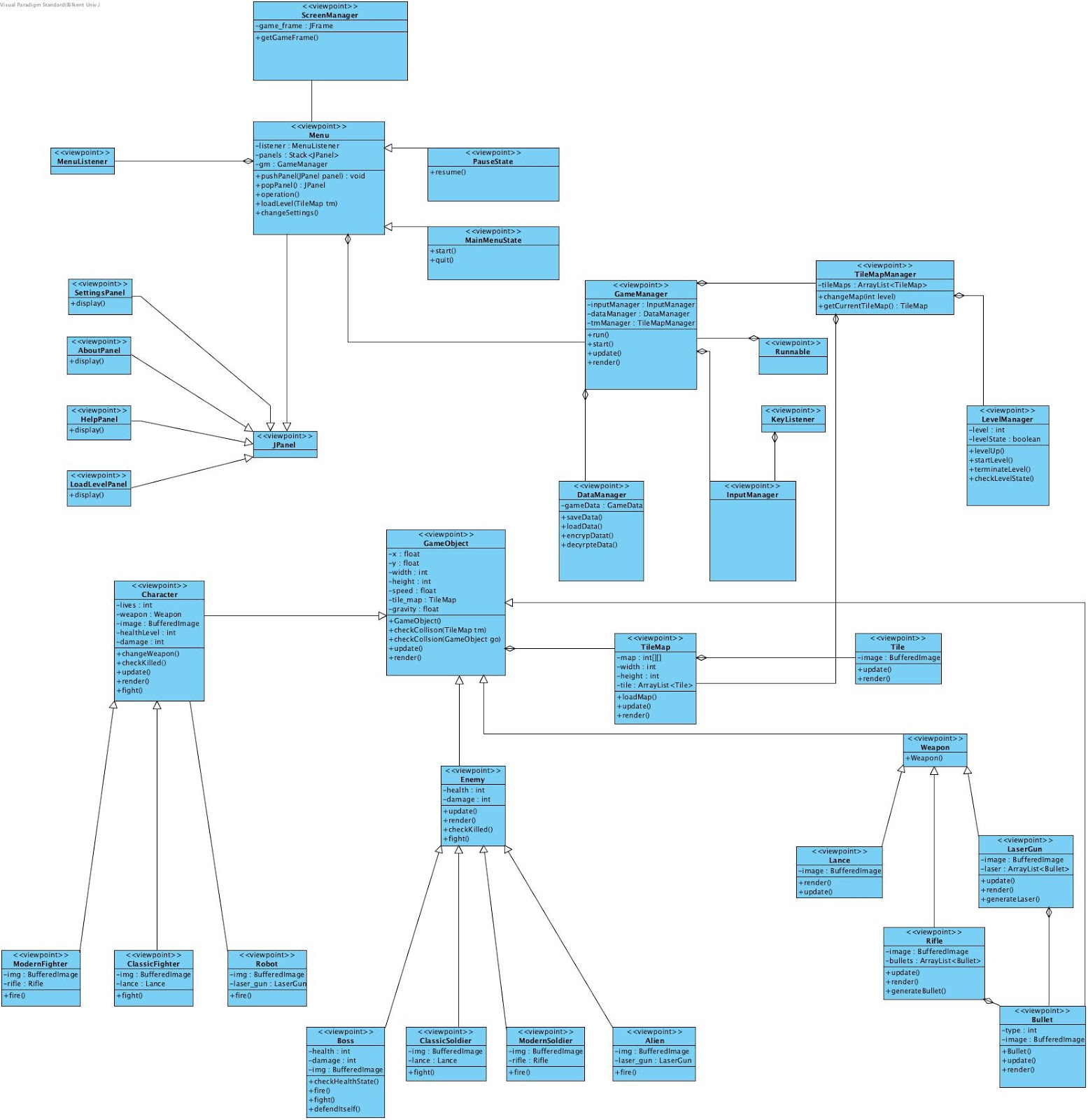
* *Float x, y:* Each game object will have a property of the position
* *Int width, height:* And of course, the objects will also have to have a dimension.
* *Float speed:* main game objects like game character, enemies, bullets and etc. will have to move, so they should have to have speed.
* *Float gravity:* gravity should affect the main objects of the game – character and enemies.
* *Tilemap tilemap:* in order to check the collisions, we need the tile map instance

*Methods*

* *checkCollisions(TileMap tm):* this function will calculate the collisions between the tilemap and the game object so that they should have limits to move on the screen and stop when they encounter an obstacle.
* *checkCollisions(GameObject go):* this function will calculate the collisions between the game objects each other so that we can check whether the collision is happened or not
* *update() :* In order to update the properties of the game object we need this method. For example, when the position of the game object is changed it will be updated using this method.
* *render() :* According to the update in the properties, the render method will draw the object on the screen using *Graphics2D.*

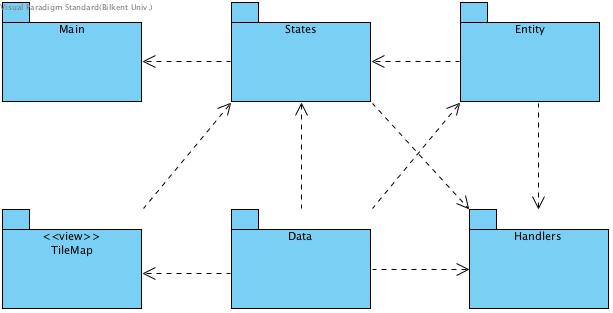
4.2. Final Object Design

This is the final design of the game system by connecting the subsystems.



4.3. Packages

The game system will be divided into some packages in order to organize and create interaction among the classes of different subsystems.



**5. Glossary & References**

* [1] JRE – Java Runtime Environment is a set of software tools for development of Java applications. It combines the Java Virtual Machine (JVM), platform core classes and supporting libraries. **JRE** is part of the Java Development Kit (JDK), but can be downloaded separately.
* [2] MVC – Model View Controller is a software architectural pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to, and accepted from, the user.