Bilkent University

Department of Computer Engineering

CS 319 - Object-Oriented Software Engineering

Design Report

**Man of War**

**Group 1-D**

* Berat Tuna Karlı
* Ceren Uysal
* Serdar Erkal
* Selim Fırat Yılmaz

Table of Contents

[1. Introduction 4](#_Toc500014714)

[1.1 Purpose of the System 4](#_Toc500014715)

[Design Goals 4](#_Toc500014716)

[1.2.1 User Criteria: 5](#_Toc500014717)

[1.2.2 Maintenance Criteria: 5](#_Toc500014718)

[1.2.3 Performance Criteria: 6](#_Toc500014719)

[1.2 Definitions, acronyms, and abbreviations 6](#_Toc500014720)

[1.3 Overview 7](#_Toc500014721)

[2. Software Architecture 8](#_Toc500014722)

[2.1 Overview 8](#_Toc500014723)

[2.2 Subsystem Decomposition 8](#_Toc500014724)

[2.3 Hardware/Software Mapping 10](#_Toc500014725)

[2.4 Persistent Data Management 10](#_Toc500014726)

[2.5 Access Control and Security 11](#_Toc500014727)

[2.6 Boundary Conditions 11](#_Toc500014728)

[2.6.1 Initialization 11](#_Toc500014729)

[2.6.2 Termination 11](#_Toc500014730)

[2.6.3 Error 11](#_Toc500014731)

[3. Subsystem Services 12](#_Toc500014732)

[3.1 GUI Subsystem 12](#_Toc500014733)

[3.1.1 General GUI Classes 12](#_Toc500014734)

[3.2 Game Logic Subsystem 13](#_Toc500014735)

[3.2.1 Character 13](#_Toc500014736)

[3.2.2 Bullet 14](#_Toc500014737)

[3.2.2 Door 14](#_Toc500014738)

[3.2.3 Mob 15](#_Toc500014739)

[3.2.4 Item 15](#_Toc500014740)

[3.2.5 GameStateManager and Related Classes 16](#_Toc500014741)

[3.3 Utility Subsystem 17](#_Toc500014742)

[3.3.1 InputManager, GraphicsManager, FileManager 17](#_Toc500014743)

[4. Low-level Design 17](#_Toc500014744)

[4.1 Detailed Object Design 17](#_Toc500014745)

[4.2 Object Design Trade-offs 19](#_Toc500014746)

[4.3 Packages 20](#_Toc500014747)

[4.3.1 Java.util 20](#_Toc500014748)

[4.3.2 java.awt 20](#_Toc500014749)

[4.3.3 java.io 20](#_Toc500014750)

[4.3.4 javax.imageio 21](#_Toc500014751)

[4.3.5 java.net 21](#_Toc500014752)

[4.3.6 javax.swing 21](#_Toc500014753)

[4.4 Class Intefaces 22](#_Toc500014754)

[4.4.1 ActionListener 22](#_Toc500014755)

[4.4.2 KeyListener 22](#_Toc500014756)

[5. Improvement Summary 23](#_Toc500014757)

[5.1 Update Method 23](#_Toc500014758)

[5.2 Attack System 23](#_Toc500014759)

[5.3 File System 23](#_Toc500014760)

[5.4 Save-Load System 24](#_Toc500014761)

[6. References 25](#_Toc500014762)

# Introduction

## Purpose of the System

Man of War is a 2d quick time event based adventure game. The reason why we chose this project is because we think that we could reflect the basic principle of OOP structure better and this is a great chance to improve our skills of how to use and design object-oriented programming idea. In addition to this, we would like to study about the process of planning a Greenfield project rather than simply creating a game. We will develop this project in Java because we all have experience in Java language, hence we believe that we could use Java more efficiently. Development platform of this project would be IntelliJ IDE. We are planning to deploy the Final project to .JAR through Maven.

### Design Goals

We give priority to design goals before implementation, because all of the implementation is based on decided contents from the design report. We did not directly start writing codes of the game, because if we encounter any problems related with coding we would stuck at that point and we could not progress for the next steps of the project. Therefore we decided to reveal all of our specialties clearly first. It will be given brief information about each of the elements below.

### 1.2.1 User Criteria:

Ease of Use: The game has known logic with many old games (ex. Pokemon FireRed). For this reason, our game appeals to any player with every age and gender. There will be 3 different levels of the game, this will provide various maps to avoid boredom and to increase entertainment. Player will only use arrow keys which will make game play easier.

Ease of learning: There will be an option in main menu in order to provide a tutorial for the players who want to learn how to play the game before they play. The logic behind the game is not only easy to learn but also familiar for everybody.

### 1.2.2 Maintenance Criteria:

Extendibility: It is essential to make updates to this types of adventure game to capture players’ attention. This feature increases the chance of appealing other players desire to play who have not played our game before. There could be new features like different types of potions, enemies and characters. Also, player might be able to change the background sound of the game. There would be some additional new levels which could make the game more challenging for the expert players. Moreover, we might have multiplayer feature. That feature would also increase entertainment of the game. In this way, the game would become more challenging and the competition between the players would increase.

Portability: Our game will be implemented in Java so that we might use the opportunity of JVM which will provide us to play the game on different platforms. This feature supports freedom for both players and developers.

Understanability-User Friendliness: The game has designed to have a simple underlying logic. In addition to this, there is a tutorial in the main menu for the players who will just start playing the game. The background sound and game sound are optional since player make changes with respect to their request.

### 1.2.3 Performance Criteria:

Efficiency: The efficiency of our game will be based on the efficiency of our codes which will build up the game itself. We are planning to use optimized number of components to prevent extra cost of memory for computer and extra time for developer. Also we prefer to abstain from using CPU more than it is needed.

## Definitions, acronyms, and abbreviations

Maven [1]: A software project management and comprehension tool. Based on the concept of a project object model (POM).

MVC [2]: Model View Controller

JDK [3]: Java Development Kit

QTE [4]: Quick Time Event. A method of [context-sensitive](https://en.0wikipedia.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvQ29udGV4dC1zZW5zaXRpdmVfdXNlcl9pbnRlcmZhY2U) [gameplay](https://en.0wikipedia.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvR2FtZXBsYXk) in which the player performs actions on the control device shortly after the appearance of an on-screen instruction.

JRE [5]: Java Runtime Environment

Green Field Project [6]: Basically, not taking previous work into account when starting a new project. Having Analysis and Design parts completed before starting the implementation of the project.

JVM [7]: Java Virtual Machine.

## Overview

In the Introduction section, the identification of the main goals of our game is explained. The goals of the game includes using OPP well, learning how to design well, before starting a new project. We learned that the documentation of a project is essential for the people who could possibly join later to the project. Thus, analysis and design stages of the project are more important than simply coding. Also, we would like to create interesting game for players. We also mention the design goals of the game. Such as ease of use, ease of learning, extendibility, portability, understandability-user friendliness, efficiency and so forth. The trade-offs are also discussed to compare between the contrasting issues about the games like performance vs memory and user-friendliness vs functionality. We try to have equal and balanced distinction between the comparisons to obtain the optimal results.

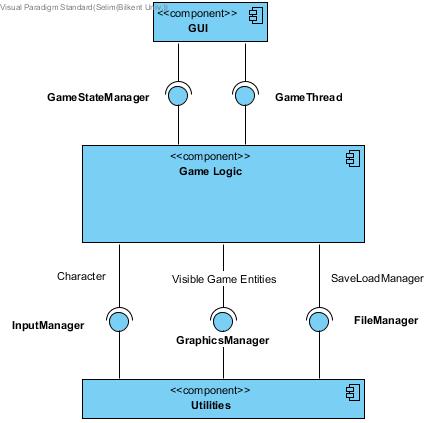
# Software Architecture

## Overview

In this section, our system will be decomposed into extensible modules. Our goal is to split our system into modules, to reduce coupling between irrelevant parts of the system and to increase the cohesion in modules.

## Subsystem Decomposition

In this section, we propose three layer architecture for our system: General GUI Layer, Game Logic Layer, and Utility Layer. The reason behind this decomposition is to make game logic GUI-independent as much as possible by letting GUI to depend on game logic and to reduce the complexity in game logic by adding utility level to our architecture.



In General GUI Layer, we will only implement the menu, tutorial, settings, credits panels and in-game information bars like health, and items. General GUI Layer depends on Game logic so that it uses the information like health and graphics and displays it on screen. This decomposition helps us to decouple game-logic independent components from game logic.

In Game Logic Layer, each game entity has its own state. Game Entities are only responsible of themselves and cannot change each other’s state. Only an entity’s components’ and entity itself can change its state. We strictly apply this rule because we want to decouple components from each other so that we reduce the number of classes that can change an attribute, it would ease the debugging. Each game entity has three components: Input Component, Graphics Component, and Physics Component. Input component will trace inputs from the player and simple game AI and changes the owner entity’s state based on these inputs. For example, if we press UP key the Y velocity of the main character increases. Physics Component of main character has no idea about that change but since it updates position of main character in each game loop(namely a frame), the character goes up if it has not blocked by any walls. As seen, Physics Component is only responsible of physics related functionalities and actions happen directly in case of state changes like other entity child components. Graphics components sends graphics/animations to gui drawer so that graphics of game entities are drawn according to the state these game entities. Graphics component is dependent on the owner entity’s physics component so that the graphics component take care of the physically impossible and unintuitive cases. This component pattern decomposition favour decomposition over inheritance [1] so that complex hierarchy should not appear and it keeps its maintainability.

In Utility Layer, Input Manager simply redirects the keyboard inputs to game logic. Graphics Manager simply draws the given graphics from entities’ graphics components to the canvas of game. File Manager is to provide persistent data to Save Load Manager. Utility Level helps to reduce complexity in game logic layer.

## Hardware/Software Mapping

Man of War will be in Java 8, therefore we use JDK 1.8+. Accordingly, the user needs JRE 1.8+ to run our game. Only a keyboard as input device is required to play the game.

As storage requirement, we assume that the size of our game will be 100MB (-50/+50). We think this accuracy is enough because today’s computers can easily meet this need. Most of this size will be is caused by the graphics of the game stored in .JPEG/.PNG formats.

## Persistent Data Management

Man of War will store the data of saved level in a text file in JSON format. The only thing we need to do for save functionality is to serialize Game State to JSON format using java standard libraries. Likewise, to do for load functionality is to deserialize JSON text into Game State using java standard libraries.

## Access Control and Security

Man of War will not depend on any important (in terms of cost) information and it is a single player game. Thus we disregard security concerns for the sake of simplicity.

## Boundary Conditions

### 2.6.1 Initialization

Man of War will be executable through a .jar file.

### 2.6.2 Termination

Man of War can be terminated through clicking “Quit Game” button in the main menu. If player wants to quit during game playing, he/she can click “Quick Game” button through pause menu and the player will be asked whether he/she wants to save current game. The “X” button of window on the right will do the same functionality as “Quit Game” buttons.

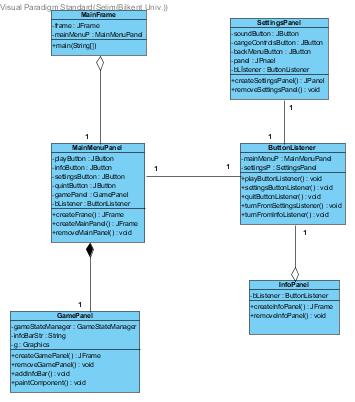
### 2.6.3 Error

Although we try to minimize the errors, there may be some errors due to persistent data management. If graphics could not loaded due to an error, the game will start anyway but won’t show that graphic(s). If a saved game state file could not be loaded, the saved game won’t start and error will be shown to user. Also, in credits section, it will be stated that players can report bugs through given mail addresses.

# Subsystem Services

## GUI Subsystem

### 3.1.1 General GUI Classes

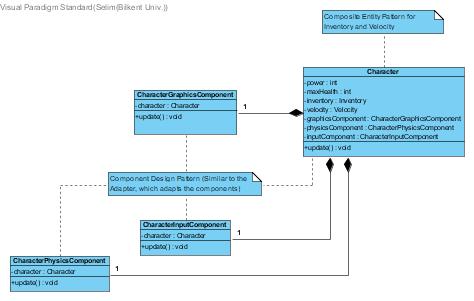


When the player entered the game, he/she will encounter the menu which is shown below. This menu contains several buttons which are “Play the Game”, “Settings”, “Help” and “Quit”.

GamePanel class controls all paint during the game play. It has info bar and it appears on the lower screen so that user could see character’s specifications. Also, it works in coherence with the GraphicsManager in the Utility layer. ButtonListener class Listens Main Menu, Settings and info button and creates new panels to main frame. InfoPanel is the panel to inform the player about game on the subjects like how to play, how to change settings etc. MainMenuPanel creates main menu and its buttons when player first enter the game he/she will interact with this panel. GamePanel works in coherence with utilities and the GameStateManager instance.

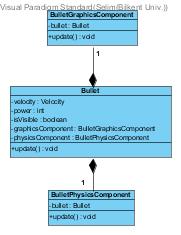
## Game Logic Subsystem

### 3.2.1 Character



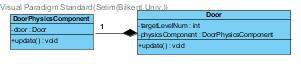
Character is the game object that is to be controlled by the player. CharacterGraphicsComponent is the responsible class for the graphics related business of character. In other words, draws the character in the required position and works in coherence with GraphicsManager. CharacterInputComponent is the responsible class for the input related business of character and works in coherence with InputManager. CharacterPhysicsComponent is the physics component of the character and works in coherence with other game objects' physics components. Character adapts these components.

### 3.2.2 Bullet



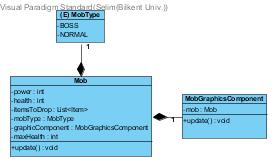
Bullet is a GameObject that is used by Character and Enemy instances to attack and harm the opponent. BulletPhysicsComponent checks whether it is colliding with an Enemy or the Character instance. BulletGraphicsComponent is the responsible class for the graphics related business of Bullet. In other words, draws the Bullet in the required position and works in coherence with GraphicsManager. Bullet adapts these components.

### 3.2.2 Door



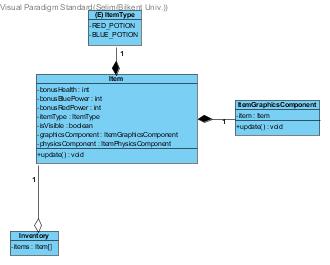
Door is the game object that represents the situation when user is in, level changes. DoorPhysicsComponent is to check whether the Character is reached to the door. Door adapts DoorPhysicsComponent.

### 3.2.3 Mob



Mobs are randomly distributed enemies which the main character can fight with. Mobs have x and y components, attack power since they can damage the main character and his/ her health points. If its health point is less or equal to zero then mob dies.

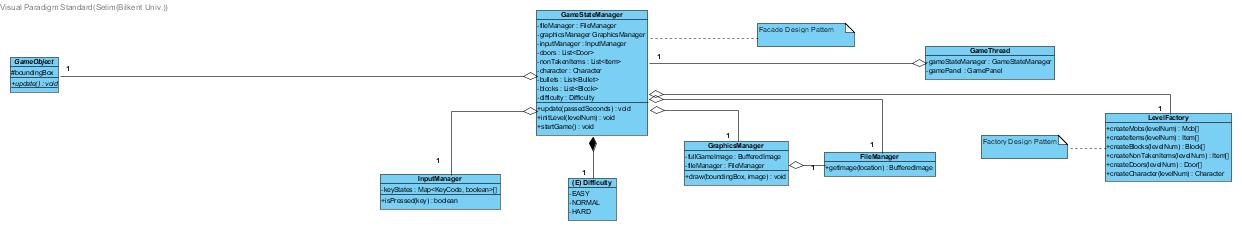
### 3.2.4 Item



Item class is to represent an item as a game object. ItemGraphicsComponent is the responsible class for the graphics related business of character. In other words, draws the item in the required position and works in coherence with GraphicsManager. The physics component of the item and works in coherence with other game objects' physics components to check whether the player picks an item. ItemType enum is to represent various items like red potion.

Potions are randomly distributed power ups for the main character which gives the blue or red power to the main character. Potions have x and y components since they are added in the screen. They also have constant amount of power which enables the main character to have power.

### 3.2.5 GameStateManager and Related Classes



GameStateManager is to control the game state and contains all its game objects. Manages coherence between utility managers and game objects. Contains all managers and game objects but interferes with them as little as possible.

LevelFactory class is a helper class using Factory design pattern for GameStateManager. Creates the game objects and returns them to this manager. GameThread class is the responsible of the game loop. Executes update methods of the game objects through GameStateManager and repaints the GamePanel.

Related Utility classes are omitted here and discussed below.

## Utility Subsystem

### 3.3.1 InputManager, GraphicsManager, FileManager

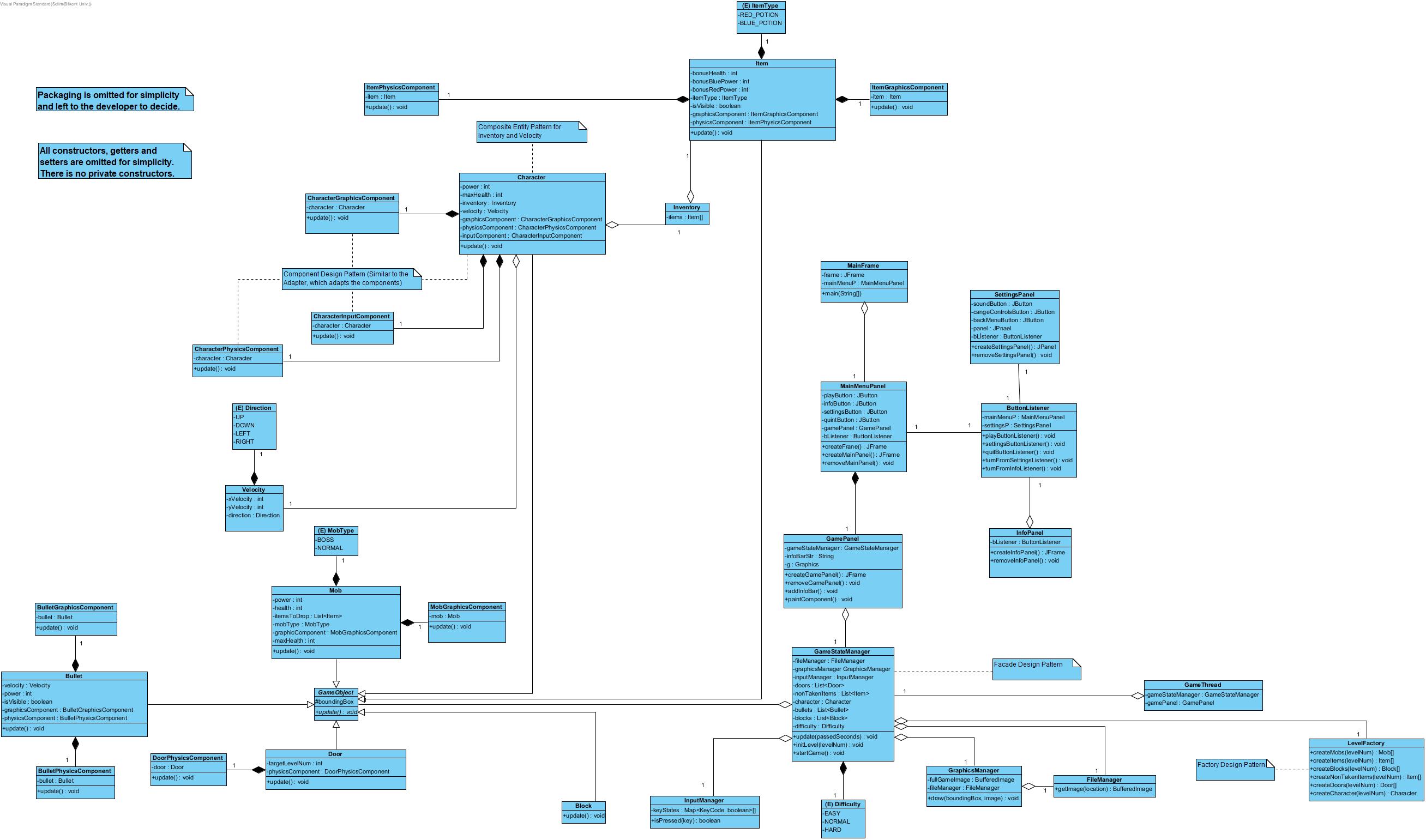


FileManager instance is responsible for the handling persistent file system. GraphicsManager class is responsible for drawing the game screen. InputManager is responsible for tracking user inputs and stores them. Another important thing is to say is that GraphicsManager uses FileManager to ease its access to the image files.

# Low-level Design

## Detailed Object Design

The full class diagram is shown in the next page in order to provide a better understanding about the interactions and basic fundamentals of our software and via the help of class diagram, the subsystem descriptions in the next sections will be easier to realize.



## Object Design Trade-offs

Performance vs Memory: We are planning to make our game as smooth as possible since we would like to make the game more attractive and interesting but we try to avoid using memory too much, because this would occupy the memory more than it is needed which could make the game a lot slower. Thus, this might cause the game becoming less interesting and challenging. Also, several players might stop playing the game which will probably cause losing more potential players. In addition, we chose pixel graphics to occupy less space in the memory, however nowadays pixel graphics are very popular and widely used in many games. Players prefer the games with pixel graphics because it reminds them the past which is somehow nostalgic and familiar.

User-friendliness vs Functionality: Our game is designed to involve many useful functionalities but we also want our game to be user-friendly. Hence we try to make optimal choices for both sections with respect to the player and his/ her desires. This is why, we give importance to both functionality and user-friendliness equally. For instance, there are different types of enemies and different levels with different difficulties that also make the game more entertaining, however we try to explain all of these functionalities with the help of the tutorial feature. There is only one type of main attack and defence function, we choose QTE because it is simple, effective and different from other usual games. Also, there is one main warrior character but later we might have some additional characters.

## Packages

### Java.util

In java.util packages, we have used Array, List, Map and Hash Map. The aim of using List is storing the lines which are pulled from the txt files. The aim of using Array is storing the data which are pulled from the List. Every interactive object has their own List and Array for storing and transferring the data through the system. The graphics of the game is drawn by these data.

### java.awt

java.awt package is initially used for user interfaces. We have made use of Listeners and Paint classes. For Listeners, we have implemented a new class. This class is only implemented in Game Panel. The inputs of the user only come to this panel. The image files such as png and jpg are passed through the system with the help of package which is called *java.awt.image.BufferedImage.*

### java.io

This package is responsible for providing the system input and output with the help of data streams, serialization and the file system. This package is used for accessing files, finding possible I/O exceptions, detecting the failures of finding the valid pathnames and so forth. FileManager Class is the main class which is responsible for handling persistent file system.

### javax.imageio

This package is used for image input and output operations. There is an usage in FileManager Class that creates an object of BufferedImage and pulls the image with the help of this package in getImage(URL url) method.

### java.net

Normally this package is responsible for having classes in order to implement networking applications. But in our case, url is used as an URL instance of the requested image. For instance in FileManager class, toURL() method is indicated. Also this is used in the syntax exception handling process of the URL and the indication of malformed URL.

### javax.swing

We have made use of this package for GUI. The benefit of using this package is being able to create JFrame JButton and JPanel. These classes are imported in the needed places of the game, mostly in presentation layer such as MainMenuPanel, GamePanel, InfoPanel, SettingsPanel and so forth.

## Class Intefaces

### ActionListener

ActionListener is responsible for receiving action events from the user. The classes which implements ActionListener interface, processes an action event. This interface is used in ButtonListener Class such as creating new panel to the frame and removes to the old panel.

### KeyListener

KeyListener is responsible for receiving keyboard events. The classes which will implement KeyListener interface, will process a keyboard event. InputManager Class implements KeyListener interface, in order to keep track of the key inputs of the user.

# Improvement Summary

## Update Method

The method which is called update is used for every single object. This method is responsible for updating the graphic component in every 50ms. The advantage of adding this method is having the graphics more smooth and continuous.

## Attack System

In order to have a new and improved attack system, we decided not to use quick time event based attack system. Instead of this, we changed the system to more dynamic and smooth one. The reason why not using QTE based attack system is that the flow of event had caused some delays and it could have created some errors related to attack system. The new attack system supports mobs, bosses and the main character to attack with the help of bullets instead of attacking when character has only encountered with enemy.

The bullet system is implemented due to the new attack system. The bullet has its own power and velocity. Also character class has updated, bullet class has been imported to Character class.

## File System

A new file management system has created. Every object has its own location (gates, potions, mobs, bosses and character) except bullet which are stored in txt files and pulled from there. Some objects such as mobs, bosses, gates, character, potions; has its own jpeg files, also they are pulled from there. This provides less coupling and more coherence.

## Save-Load System

Save system stores character’s last level in the previous game, also it stores the power and health of the character in txt files. Load system pulls this txt file and updates the stat of the character.

# References

[1] <https://maven.apache.org/>

[2] *Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition*, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010, ISBN-10: 0136066836.

[3] <https://www.techopedia.com/definition/5594/java-development-kit-jdk>

[4] <https://www.giantbomb.com/quick-time-event/3015-6/>

[5] <https://techterms.com/definition/jre>

[6] <https://workplace.stackexchange.com/questions/17293/what-is-a-greenfield-project-in-it>

[7] <https://www.javatpoint.com/internal-details-of-jvm>