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

Contents

[1. Introduction 3](#_Toc496354683)

[1.1 Purpose of the System 3](#_Toc496354684)

[1.2 Design Goals 3](#_Toc496354685)

[User Criteria: 4](#_Toc496354686)

[Maintenance Criteria: 4](#_Toc496354687)

[Performance Criteria: 5](#_Toc496354688)

[Trade Off: 5](#_Toc496354689)

[1.3 Definitions, acronyms, and abbreviations 6](#_Toc496354690)

[1.4 References 6](#_Toc496354691)

[1.5 Overview 7](#_Toc496354692)

1. Introduction

## Purpose of the System

Man of War is a 2d quick time event1 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 Maven2.

## 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.

### 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.

### 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.

Understandability-User friendliness: The game has designed to have a simple underlying

the 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.

### 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.

### Trade Off:

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 loosing 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.

## 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.

## 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>

## 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.

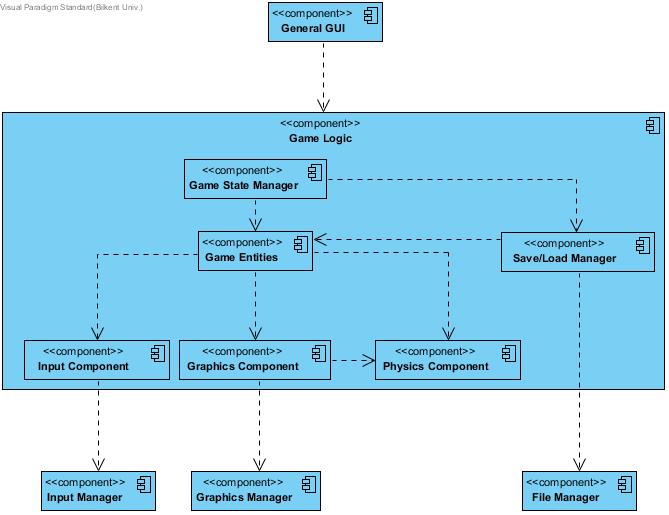
1. Software Architecture

## Overview

In this section, our system will be decomposed into extensible modules. Our goal is to split our system into modules  is 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 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 favor decomposition over inheritance[1] so that complex hiearchy 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.

## 2.4 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.

## 2.5 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.

## 2.6 Boundary Conditions

### 2.6.1 Initialization

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

### 2.6.2 Termination

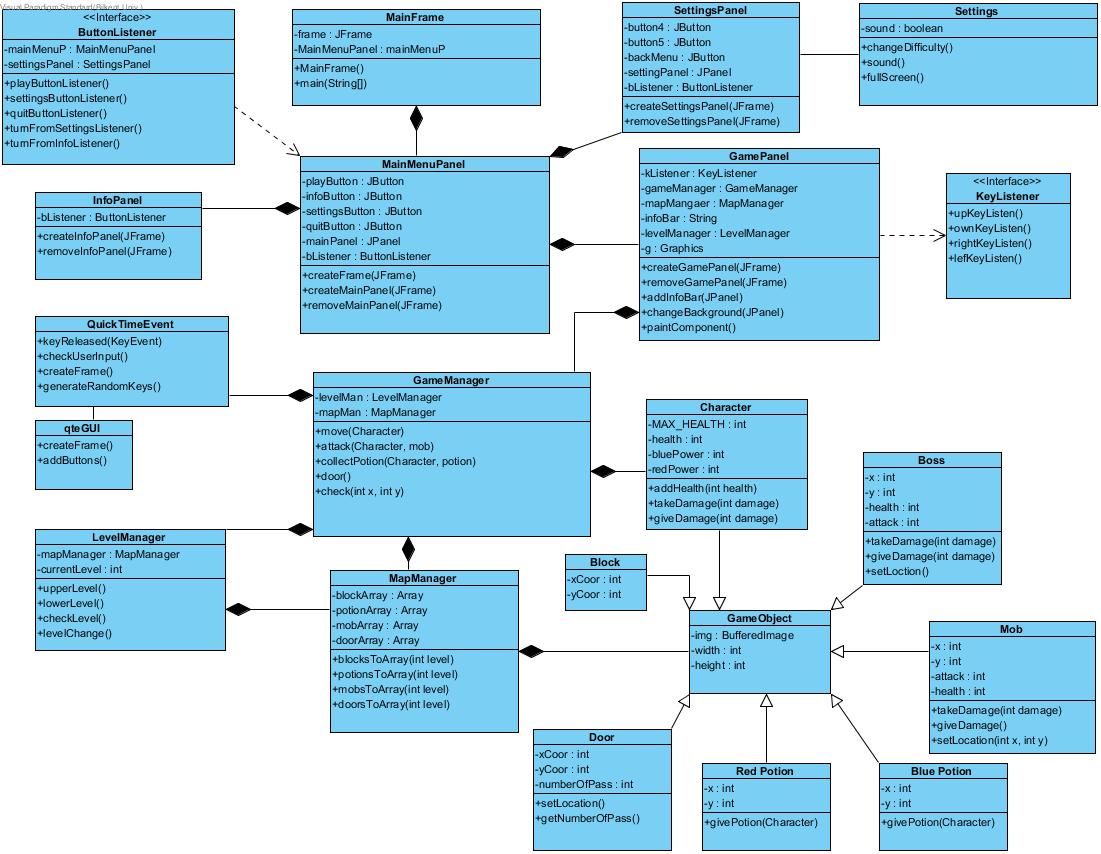
Man of War can be terminated through clicking “Quit Game” button in the main menu. If player wants to quit during game playing, he/she can click “Quick Game” button through pause menu and the player will be asked whether he/she wants to save current game. The “X” button of window on the right will do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

3. Subsystem Services

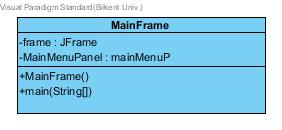
3.1 Detailed Object Design

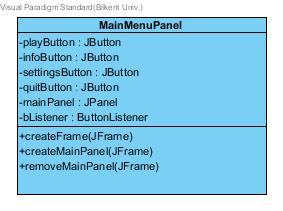
****

3.2 User Interface Management Subsystem

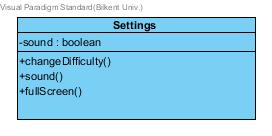
* Main Frame Class and it’s GUI class Main Menu Panel

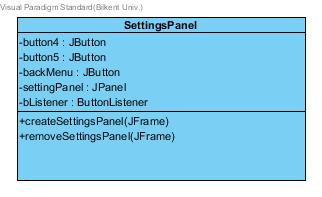
When player first entered the game he/she will encounter with this menu. This menu contains several buttons which are Play the Game, Settings, Help and Quit.





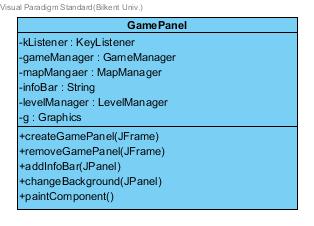
* Settings Class and it’s GUI class Settings Panel





* Game Panel

It’s a GUI that controls all paint during the game play. It has info bar it appears on lower screen so that user can know character specifications.



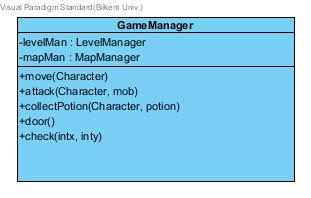
* Level Manager Class

Since this game have different levels, level manager have to check the levels and contact with map manager. Depends on level, this manager calls map manager’s add item methods. Basically level managers’ main purpose is changing the level in game.



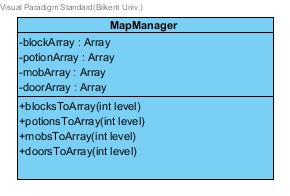
* Game Manager Class

This is the class that controls every event in the game such as moving character, collecting item, attacking to mobs and bosses (QTE), move into gates etc. It checks x-y components that user wants to go with character. If there is any interactive object, it calls the related methods.



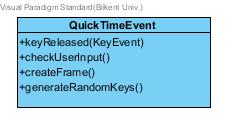
* Map Manager

This manager controls the item, mob, boss, block and their locations. Manager gets the information from text files.



* Quick Time Event

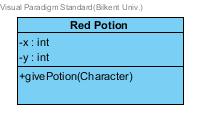
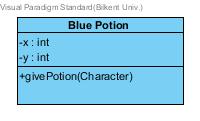
This is the games’ main combat system. While in mob or boss fight, it appears and in a few second player should enter required keys so that mob or boss can get damage. It generates random keys and key input from user then compare them.



3.3 Interactive Game Objects

* **Potions**

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

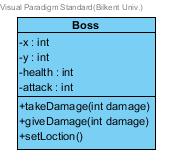
* **Mobs**

Mobs are randomly distributed enemies which character can fight. Mobs have x and y components attack powers since they can damage you and health points. If its health point equal or under zero then mob dies.



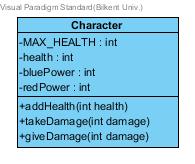
* **Bosses**

Every gate have their unique bosses. If boss don’t die, player won’t be able to continue to next level. Bosses have health, attack power and x-y components.



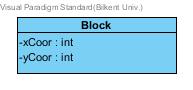
* **Main Character**

Main character have health which can be increase or decrease, x-y component which changes with motion. Unlike mobs and bosses Main character don’t have attack power but red and blue power. Since Bosses are unique, Main characters fight them with red or blue power.



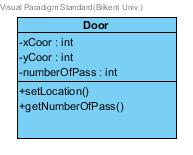
* **Block**

The blocks which main characters can’t move into.



* **Door**

Doors are objects that which opens to new level unless current level is not last level. It should know how many passes occur for that door.



3.4 File Management Subsystem

This game include lots of images and icons. This system loads images and icons to game. This system is not only image loader but also location loader. All items which in screen have their unique locations (some of them may be change in game) and this x-y components kept in txt files. This system loads necessary item’s icon and location to game.

3.5 Input Management Subsystem

IMS mainly responsible for checking the inputs which users’ enters. It embody all listeners.

