Bilkent University

CS 319

Object-Oriented Software Engineering Project

Design Report Draft

Section 1 / Group 1H

Ece Altınyollar

Arif Can Terzioğlu

Raza Faraz

Emin Tosun

Instructor: Bora Güngören

TABLE OF CONTENTS

3.2) Input Manager

3.3) Interaction

3.4) User Interface

3.5) Game Manager

3.6) Menu Data

3.7) Database

1)Design Goals

1.1) User

1.1.1) Well-Defined Interface

User interface is one of the most important things for a game to make it desirable for the users. Because of this reason we planned for Battle for Middle Earth some simple intractable objects. Buttons locations and the description of buttons are going to be clear and understandable for the user. Also the clear guidance is going to be provided for the comfort of players. The backgrounds of the screens will be the Middle Earth theme. Also while the players play Battle for Middle Earth, they will have characters and enemies with middle earth theme. These features make the game more realistic and desirable for the players.

1.1.2) Ease of Use/ Learn

Easiness of usage and learning a program to a user is one of the purposes for creators. Since the users do not want such a programs which have difficult usage and hard to learn how to use. For the usage purpose the screens that user has to pass over are going to be visualized before the game start. The users can do easily the required parts to start the game. Also the help section will be available for learning how to play Battle for Middle Earth. In the help section control of the character, fire controller and power-ups will be explained. With these features we are going to reach this purpose.

1.1.3) Performance

The performance of a program affects the quality of a game. So we will make Battle for Middle Earth as a high performance game. First of all the passes between the screens will be smooth. Because of this is an interactive program the game also will have response time as short as possible. With these Battle for Middle Earth will work nicely without any glitches. These features make the game more qualified.

1.2) Maintenance

1.2.1) Understandability

Each part of our project will be understandable for anyone who uses and analyses our project. In order to provide this opportunity to people we design our reports very simple and clearly. We also focus on completeness of the project. The coding part also will be clear and much more commented for increasing project's understandability.

1.2.2) Modifiability

When project needs to be modified, change in one part should not affect other parts of the program. In order to achieve this goal, we need to keep coupling at the minimum level, so that change in one part of the code, the whole project would not be affected too much. We will have some subsystems. These subsystems will share less data because modifiability of the project could be higher.

1.2.3) Good Documentation

We will keep our documentation standards as high as we can. By providing good documentation, we make our project more understandable and clearer. The

completeness of reports and their relations with each other will be taking into consideration more seriously. This will also increase our projects understandability.

1.2.4) Portability

Since there are many different types of devices, operating systems and versions, our program may encounter such a problem that platform dependence or support issues. To ensure portability and have a stable performance between these environments, our development language will be JAVA which is supported widely by MAC OS, WINDOWS and LINUX… Thus, game will be played for most of the systems that can run JVM.
Also, by using SQL, our database will be managed between relational database management systems.

1.2.5) Reliability

Our game will be produced with the principle of reliability. To achieve this, it will be ensured that program will not be crushed under any users' input. Possible error cases will be determined during the design stage. To handle these errors, exceptions and user based solutions (giving error message and showing options to user) will be used. The exceptions will then redirect the program to normal operating way. System is going to be designed and supported with software reliability testing.

1.3) Trade off

1.3.1) Portability vs Reliability

To achieve the portability goals, we will use JAVA environment for implementation. JAVA has its own bytecodes which is known for its ability to produce byte codes which can run on any processor architecture. However, this byte codes only work in JAVA Virtual Machine (JVM). Thus, all users have to install JAVA to their computer. On the other hand, compared to pure compilation languages, JAVA does not have a good running time performance as much as them. Due to game is not complex too much; this trade-off will not be too much problem.
By using JAVA, due to compilation time checks, our software will be aimed error free as much as possible.

1.3.2) Development Time vs Performance

When we first initiated the project we thought about developing the game in C++ language as it offers much higher performance compared to other languages, but after some research, we have shifted our implement to java. The reason being Java offers much better GUI libraries, which are much more similar and reliable to us. Another reason why we have not chosen C++ is the issue of memory management; the concept to avoid possible memory leaks, which increase the space taken during the game thus reducing the game performance. All of this would be avoided if java is implemented as

memory management is done automatically. Finally, as our game will not include high-level graphs, the game will be able to process the data much quicker, thus increasing the gaming performance.

2) Software Architecture

2.1) Overview

In this section, we are detailing the composition our system from higher level to lower. Because dealing with huge program is hard, we decided to divide our program into subsystems. While dividing our program, we decided to use 3-tier design architecture because it is suitable for our project. The layers will be explained in this section but more details of subsystems will be in other sections.

2.2) Architecture Style

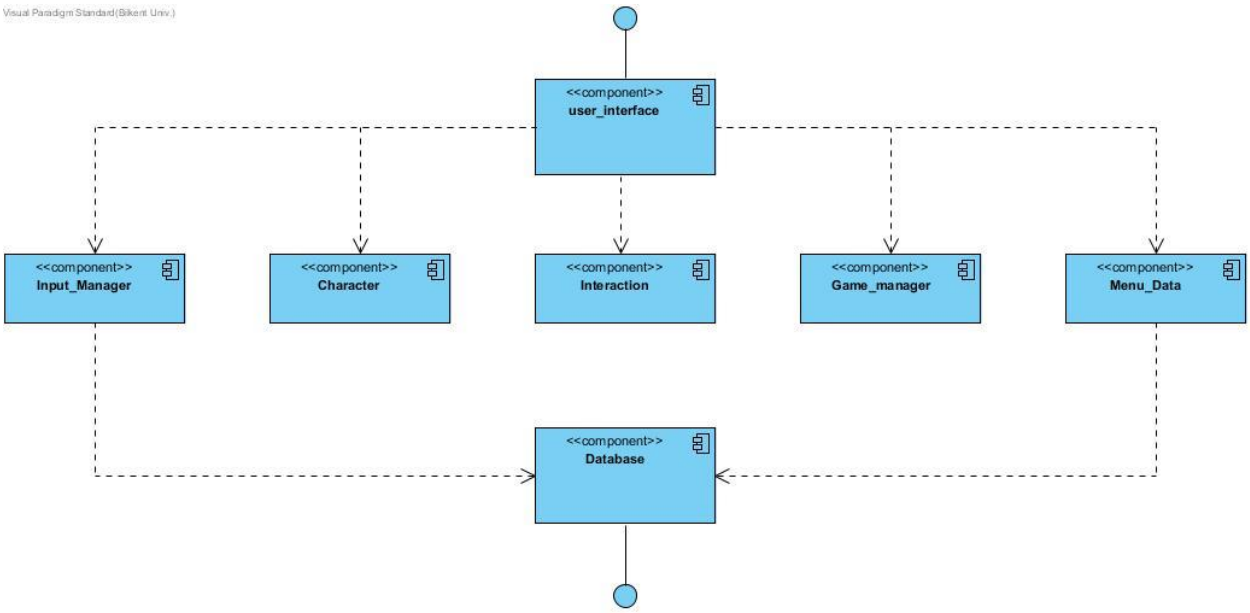
We have chosen the three tier architecture style for our project. In this style, program will be divided into three layers. First layer is presentation layer. The presentation layer is responsible for managing the interaction between user and program. It shows graphics to the user with data coming from program and it also transmits the user's inputs to the program. The second layer of the architecture is

application layer. Its duty is creating the applications data and processes the data coming from user or database. The last layer is data layer. This layer is responsible for data management. It transmits data which needs to be stored to the database and it also conveys the data from database to the program itself.

2.2.1) Type of Architecture

For this project, we chose the closed architecture instead of open architecture. The close architecture minimizes the coupling. It makes easier to develop the game. Adding new things to the game will not require changing whole code of it. We can add new things by changing only related parts.

2.3) Subsystem Decomposition



2.3.1) First Layer

In the first layer, we have UI classes and UI Manager Class. This layer is responsible for managing interaction with user and the program. It will get data from second layer which handles the business logic and it will be updated with new coming data.

2.3.2) Second Layer

The second layer of architecture is responsible for program's business logic. It will produce game data and send it to first layer which handles the interaction with user. This layer also manages data coming from database. This layer is like a bridge between database and UI. Whole game data processing will be handled with this layer's components.

2.3.3) Third Layer

Third layer of architecture is data management layer. This layer has the database controller component. This component provides data, which is coming from database, to the second layer of architecture which uses this data. It is also responsible for saving data coming from program itself or user.

2.4) Hardware / Software Management

Our game BattleForMiddleEarth will be coded in JAVA environment since JAVA 8 provides us for beneficial libraries such JAVA FX GUI library. Also, account information and high scores will be saved via MySQL. Storing images and sounds will be in the .png and .wav formats respectively, the operating system should support them.

On the hardware configuration part, game will need a basic keyboard controls that are space and arrow keys. Our game will not require too much system specification such as high RAM etc. In addition, at the beginning of the game, keyboard inputs will be necessary for create or login account.
Any platform such as Windows/macOS/Linux supports the JAVA can run our game with these hardware and software requirements.

2.5) Persistent Data Management

External database will be used to hold user's latest level, high scores, account information and user's preferences like character type. To load the latest point of players in the game, this information will be taken later.
Changes on data will be reflecting immediately to maintain data flow fluently. Database will have MySQL management system and SQL language will be the used to create query for database operations. Other files such as pictures, sounds will be stored locally.

2.6) Access Control and Security

Our game will require a connection to database. The username and password will be taken when user enters the game first time and transfer to database. Although game has no internet connection, it is aimed that different accounts will enable user to play with more than one player without losing without other characters' data. In other

words, access control will be used to differentiate user's characters. Our program will be coded in a way that only our program can access it.

2.7) Boundary conditions

Initialization

Battle for Middle earth will not require any installation as the game would be available as executable .JAR file. During the stage when the .JAR file is clicked and the program opens, appropriate Graphical Interface will be loaded such as game images etc; thus the game would be brought to an initializing state.

Termination

There are multiple possible ways for the termination of the program. Standard procedure includes clicking on the 'Exit' button present in the 'Main Menu' Screen and 'Paused' game screen. Additional method includes clicking on the 'X' present at the top left corner of the screen of using Task Manager

Failure

Possible Failure would result from missing program files in which a dedicated error message will be produced. Other Possible Failure includes absence of .JDK file or updated .JDK file which would cause the game to crash, thus displaying an appropriate error message

3) Subsystem Services

3.1) Character

The character component is in the second layer of our three-tier architecture. This component includes Account, Player, Weapon classes and Elf, Human, Wizard subclasses. The data about account and the features of characters that player choose is in this component. The character component interacts with interaction and database components. Link between Interaction component and character component provide the data about levels that account has. Character component takes the data that about the succeed level and it provides data about current level to interaction component. The character component also provides the login information of the account to database component and takes the account information from the database component.

3.2) Input Manager

Input manager component is in the second layer of our three-tier architecture. The interaction manager component handles the hardware inputs. It gets inputs from mouse and keyboard. Then it provides the data that includes these inputs to user interface component.

3.3) Interaction

Interactions component is in the second layer of the three-tier architecture. It includes Collision, Power-up, Level and Enemy classes. The data about interactions that player has during the game play is in the interaction component. It interacts with game manager and character component. The interaction component provides the data about levels that is passed to character component and takes data about the current data from this component. The interaction component also provides the data about game to game manager component.

3.4) User Interface

The user interface component is in the first layer of our three-tier architecture. This component takes data from all other components on condition that direct and indirect. Input manager, game manager and menu data components have the direct interaction with user interface component. The input manager provides hardware inputs to the user interface component. The menu data and game manager component also provides the data to user interface component. The menu data shares the menu interaction of the player to user interface. The game manager component provides the game data to the user interface component.

3.5) Game Manager

The game manager is in the second layer of the three-tier architecture. It handles all the game play features. The game manager interacts with interaction and user interface components. It takes the game data from interaction component and sends the data of the game to user interface component.

3.6) Menu Data

Menu Data package will place in the second layer of our three tier architecture. This system consists of Help, Credits, Setting and HighScore classes. Menu Data component works connected with Database system to get High Scores' of player. Also, reflecting changes of settings and transition between screens, Menu Data provides information for different menu screens to user interface component.

3.7) Database

Database component takes place in the third layer of our three-tier architecture. This part contains the databaseController class. Basically this system established the connection between database and our game at the bottom layer. User's latest level, high scores, account information and user's preferences like character type transferred via this system. Also, changes on data reflect on the database within this package.