CS 319 - Object-Oriented Software Engineering

Project Design Report

LOST

GROUP 2-D

Eren Bilaloğlu - 21401603

Gunduz Huseynli - 21402821

Onur Sönmez - 21300749

Yasin İlkağan Tepeli - 21301707

1. Introduction

1.1. Purpose of the System

The purpose of our game, LOST, is to entertain its users. Open world, survival map allows players to interact with various objects, and experience difference scenarios in the game, and decide strategically.

1.2. Design Goals

Maintainability

Game design will be based on object oriented programming concepts which will allow us to implement the game in an extensible manner. Game's subsystems will have hierarchical structure so that new features will be added easily without huge changes in the upper systems. Additionally code will be written in a way that can be easily understood by others.

User - Friendliness

LOST will have user friendly interface design which will enable users to easily grasp the game basics and controls. Considering the human's cognitive abilities, game objects' icons and control icons will have similarities with real life for easy understanding and memorizing. Besides, a help wiki will be available online, accessible from the main menu which will explain the game in a detailed way.

Robustness

In order for our players to enjoy, our game should not have any major bugs that will disrupt the flow of the game. We will achieve this by limiting the user's interactions with the game through user interface, and handling exceptions in a safe manner.

Performance

Players, while playing our game, will not want to wait for long every time they perform a certain action. As a result, we will design the game in a way that player's actions will result in less than 2 seconds on average.

Trade-Offs:

Cost vs Portability

With limited time and energy, we are only implementing the game for desktop operating systems that runs java. As a result, players will not be able to play our game in mobile platforms.

Memory vs Performance

Since our game is designed with object oriented principles, we are not very concerned about memory usage. This allows us to gain run time efficiency in the design, by creating, accessing and changing every object on the go.

Functionality vs Robustness

Being an open world game, with various objects and scenarios it is easy for user to try an invalid combination of actions. However, by carefully detecting and handling exceptions, we are also maintaining the robustness of the code.

2. Software Architecture

2.1. Subsystem Decomposition

We decided use the three tier architecture which fits to design goals of our project. The first layer is user interface layer. Middle layer is the application layer, which includes the game engine and entity objects. Last layer is the data persistency layer, which will store the game data in local and cloud. By implementing a closed architecture, which only allows the layers to access the layers below them, we are making the game more maintainable. Additionally, we limit the subsystems to communicate with each other over a single interface, so that they can not directly access the subcomponents of other subsystems. This decision favors the maintainability as well as robustness of our game.

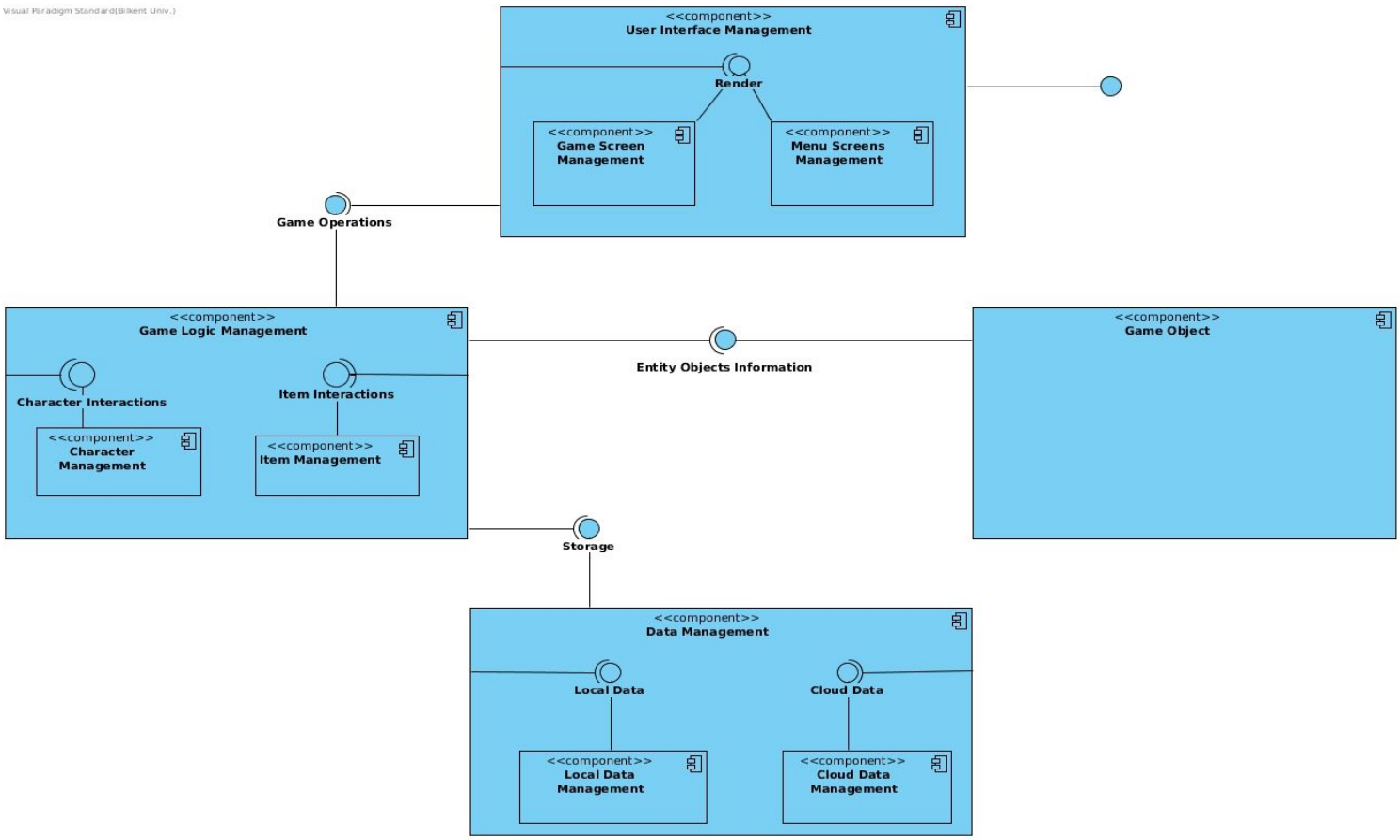

Figure 1: component diagram

2.2. Hardware / Software Mapping

Our game, LOST, does not require any specific purpose built hardware in order to run. Since, the game logic is not that complicated, it can be carried out by the general purpose hardware fast enough. However, in order to play our game, user will need a pc, a keyboard, a mouse, and a monitor. Our game will run in Linux, Mac, and Windows environments which has Java Runtime Environment(JRA).

2.3. Persistent Data Management

Lost will have both user local data and cloud-based storage. At the initial state, formatted files(.txt) will be parsed and the initial data will be uploaded to the game. The number of items, characters according to their

specific areas will be parsed instead of hardcoded implementation. Thus, the environment will be created. After initial data uploaded to the game, there will be two scenarios for saving the game data. First one is keeping local data in a file via parsing back to the files. This will be the default case for the game. If player has an internet connection, game data will also be saved in the cloud using Google App Engine. It will be responsible for recording persistent data into Google Cloud Platform. Instead of traditional relational database systems, highly scalable No-Sql document database system, Cloud Datastore on Java Flexible Environment(Java 8 extension) will be used for restful service. There will be data access objects in this content.

2.4. Access Control and Security

Users' high scores will be stored in an online database and other gameplay data will be stored in local memory. Users will not have access to these databases so that none of the players will have authority to manipulate game data externally. Every player will have equal access to game services. There will not be a user authentication system. User will enter his name when it is asked in the beginning of the game and his score will be displayed with that name when game is over. Therefore, game will not hold any valuable user information such as passwords. Thus, we will not deal with any user data security issues.

2.5. Boundary Conditions

2.5.1. Initialization

For the main menu, game does not require anything. After that, to continue a game by using Load Game option requires a savedFile in the system. If there is no file in system like that or file is corrupted, Load Game option will not be available. Also in order to look at the high scores, the program needs to access database using network. If there is no available network, that option will not be clickable also.

Additionally, if user tries to open the game when there is already an opened game, There will be pop-up which says "There is already a game in process." and the second game will not open.

2.5.2. Termination

When user decides to exit from the game while in gameplay, user can click the button "Main Menu" from the top of the screen. When

user clicks Main Menu button, the program will ask whether user wants to save or not. Also when user directly clicks the close button from window frame, it will again ask user whether to save or not.

2.5.3. Failure

If the program cannot upload the sounds or images when it is requested, the program will not crash or exit. It will continue to work and let the user decide whether he/she wants to close and open game.

We don't have any case for instant crashes due to performance issues. None of the system function will work and user will lose current data without saving.

3. Subsystem Services

3.1. Interface Layer

3.1.1. User Interface Management

This component is to display user interface, menus and handling menu and game navigation. It includes two subcomponents: Game Screen Management and Menu Screen Management. It will communicate with the Game Logic Management in the Application Layer to get the game data.

3.1.1.1. Game Screen Management

This component handles the display of game screen which will also display the map.

3.1.1.2. Menu Screen Management

This component's responsibility is to display the main menu screens.

3.2. Application Layer

3.2.1. Game Logic Management

This component basically controls the gameplay. Its responsibility is to handle the requests of user interface layer, utilizing game logic, and as a result change the game entity objects. This component can demand data from the Game Object in Application Layer and from the Storage Layer Then this component will use the

data to carry out game logic. Additionally, it will communicate with its subcomponents for relevant services.

3.2.1.1. Character Management

This component will handle the fighting mechanism, increase, decrease the character's stats.

3.2.1.2. Item Management

This component will be responsible for the crafting mechanism.

3.2.2. Game Object

This component will deliver the information requested by the Game Logic Management, about the entity objects in our game universe. Also upon being called by the Game Logic Management, it will carry out simple services that will change the game entities.

3.3. Storage Layer

3.3.1 Data Management

This component is responsible for getting the data from one of its subcomponents, parse or retrieve the data, and return it to the Game Logic Management component accordingly.

3.3.1.1 Local Data Management

This component handles the local data. It parses the game data (excluding the high scores) from the local memory. Parsed data is used from Game Logic Management.

3.3.1.2 Cloud Data Management

In Cloud Data Management, there will be data access objects and they will interact with Google Cloud Platform. Changes in game environment will be transferred by using Cloud Datastore Api.