CS 319 – Object-Oriented Software Engineering

Project Design Report

Feeding Bobby

Group 1-K:

Arda Kıray

Gülce Karaçal

Volkan Sevinç

Okan Şen

Table of Contents

1. Introduction

1.1. Purpose of the System

Feeding Bobby is expected to be a “user-friendly” single-player game for computers which aims to entertain users through some progressive challenges. The user will start off as a little fish and throughout the span of the game, the main goal of the user will be to get bigger by eating smaller fish, to survive by avoiding from bigger fish and to defeat the final boss while facing some mid game bosses.

First of all, we gave our priority to the speed of the game. Our aim is to make the game as fast as possible. Players will be provided high-performance and graphically qualified game. Considering this, “Processing” library developed for Java will be used.

Furthermore, due to the fact that game screen contains many icons that can cause distraction, we have decided to design the graphical user interface as “user-friendly” to prevent complexity.

Our main objective is to design a game with high performance engine and user-friendly interface to make sure that users having maximum satisfaction from the game.

1.2. Design Goals

Before composing the system, identifying design goals to clarify the qualities of the system is significant. In this regard, many of our design goals inherit from non-functional requirements of our system enabled in the analysis stage. Crucial design goals of our system are described below.

1.2.1 Criteria

Portability: Our game will be implemented in Java which enables cross-platform virtual machine (JVM) that allows user to run the game on many operating systems. Therefore, this feature of Java makes our game portable on any operating system that contains JVM.

End User Criteria

Usability: Considering the fact that we design a game, it is supposed to enable the user quality entertainment. Therefore, the system ought to provide user-friendly interfaces for menus to avoid complications that make user to have difficulties in using the system. In this manner, the system allows the player to easily find and perform the desired operations. Moreover, our system will implement actions according to mouse input from the user, such as clicking buttons, moving paddle which make the usability of the game easier.

Ease of Learning: Due to the fact that the user does not have to be familiar with the way the game is played, or with some basic concepts about the game such as loss-win conditions and power-up features, the system will provide an instructive help document, by which the user will be easily get warmed up to the game. The logic of the game is also very simple that user can easily understand intuitively or by reading the help document.

Performance: For our system, performance is a significant design goal, and therefore Processing library of Java will be used to improve the performance. For arcade games, it is crucial to provide an immediate response to players' requests to maintain users' interests. Considering this, while displaying animations, effects smoothly for enthusiasm, our game will almost immediately create a response to player's actions.

Maintenance Criteria

Extensibility: Adding new features to the game to maintain the excitement and interest of the player is significant. Hence, our system design will provide an environment in which adding new functionalities and entities such as new power-ups to the existing system is possible.

Modifiability: Our system is going to have a multilayered structure allowing us to modify the system easily. To accomplish this, we have decided to minimize the coupling of the subsystems as much as possible in order to avoid great effects on the system components by a modification.

1.2.2. Trade Offs

Usability and Ease of Learning vs. Functionality:

We have discussed that our game should be easy to learn because we want the user to not to lose his interest over the game. Considering this, our main priority is the usability of the game rather than the functionality. Since the purpose of the system is providing entertainment to users, our system will not bore the player with complicated functionalities. Instead, we will focus on the ways which make our game easy to understand such as having a simple interface and familiar instructions. In this way, users can spend their time playing the game rather than struggling to learn it.

Performance vs. Reusability:

Our system design's primary concern is not reusability but getting high performance. As mentioned, we focused on to designing a game that should be able to create an immediate response to players' requests to maintain users' interests. Integrating any of our classes to another game is not one of our plans. Hence, our system will have classes which are designed particularly for the tasks so that the code will not be created more complicated than necessary.

1.3. Definitions, Acronyms, and Abbreviations

Java Virtual Machine (JVM): A Java virtual machine (JVM) is an abstract computing machine that enables a computer to run a Java program. [1]

Processing (Java Library): Processing is a simple programming environment that was created to make it easier to develop visually oriented applications with an emphasis on animation and providing users with instant feedback through interaction. [2]

1.4. References

[1] https://en.wikipedia.org/wiki/Java_virtual_machine

[2] Fry, B and Reas, C. (2007). “Processing Overview”. https://processing.org/tutorials/overview/. [Accessed: Mar 17, 2017].

2. Software Architecture

2.1. Overview

In this section, we will decompose our system into maintainable subsystems. By dividing these subsystems, our main goal is to reduce the coupling between the different subsystems, while increasing the cohesion between subsystem components. We tried to decompose our system in order to apply MVC(Model View controller) architectural style on our system.

2.2. Subsystem Decomposition

In this section, the system is divided into relatively independent parts to clarify how it is organized. Since the decisions we made in identifying subsystems will affect significant features of our software system like performance and extendibility; decomposition of relatively independent part is crucial in terms of meeting non-functional requirements and creating a high quality software.

In Figure-1 system is separated into three subsystems which are focusing on different cases of software system. Names of these subsystems are User Interface, Game Management and Game Entities. They are working on completely different cases and they are connected each other in a way considering any change in future.

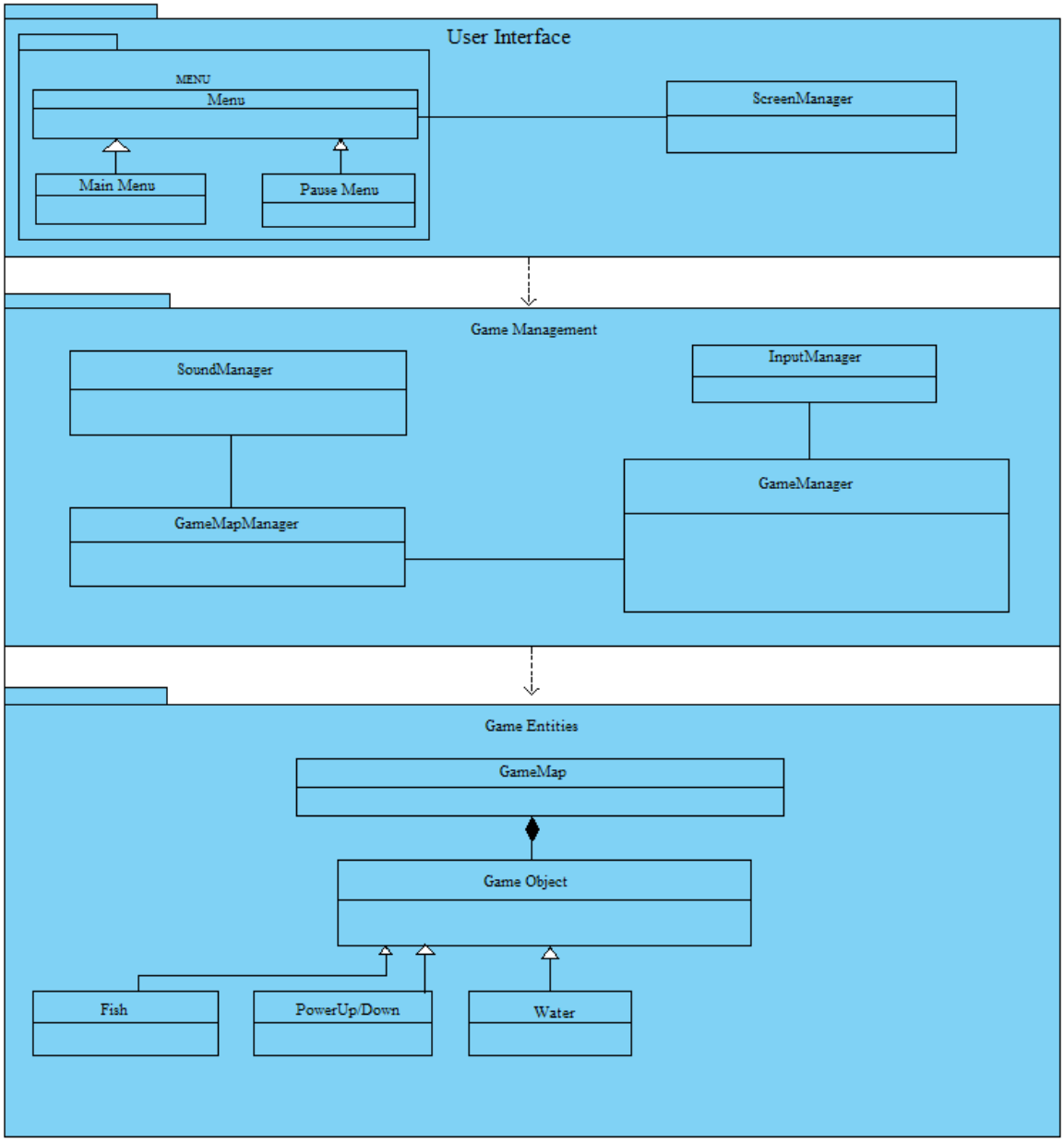

Figure – 1 (Basic Subsystem Decomposition)

2.3 Architectural Sytles

2.3.1 Layers

We decomposed our system in 3 layers: User Interface, Game Management, Game Objects. The relation between the layers are hierarchical. Since the interaction between user and the system is handled by User Interface subsystem, the top of our hierarchy is User Interface subsystem. Every interaction with user is interpreted in User Interface subsystem and the information of desired operations that needs to be performed is transmitted to Game Management subsystem. Game Management subsystem is the core part of our design. With the information of desired operations, Game Management subsystem operates on game objects. By the time, Game Objects subsystem keeps the record of game objects. It checks

whether desired game object exists, changed, unchanged and keeps the track of each game object. Below is our schematic decomposition of the system.

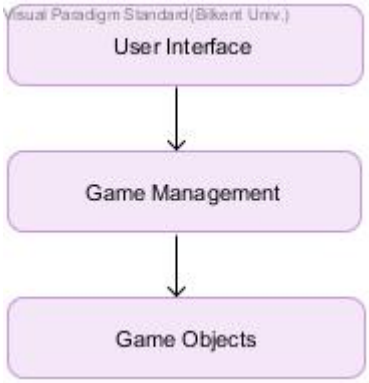

Figure 2: Layers of System

2.4. Hardware / Software Mapping

Feeding Bobby will be implemented in Java programming language. As hardware configuration, Feeding Bobby needs a keyboard (for typing username) and a mouse for the user to control the titular character. Since the game will be implemented in Java, system requirements will be minimal , a basic computer with basic software installed such as operating system and java compiler to compile and run the .java file.

For storage, we will have .txt based structure to form progression data and high score list,hence the operating system should support .txt file format in order to be able to read these files. Moreover, game will not require any kind of internet connection to operate.

2.5. Persistent Data Management

Since Feeding Bobby does not need a complex database system. it will store progression data and high score list as text files in user storage. If the text file is corrupted , it will not cause any in-game issues regarding game objects, but system will not be able to load this corrupted data. We also plan to store sound effects, music and game objects in user storage with proper and simple sound and image formats such as .gif, .waw, .png.

2.6. Access Control and Security

As indicated earlier, Feeding Bobby will not require any kind of network connection . There will not be any kind of restrictions or control for access. Also Feeding Bobby will not include any user profile. Due to this , there will be no kind of security issues in Feeding Bobby.

2.7. Boundary Conditions

Initialization

Since Feeding Bobby does not have regular .exe or such extension, it does not require an install. Though the game will come with an executable .jar file.

Termination

Feeding Bobby can be terminated by clicking “Quit” button in the main menu. If player wants to quit during the game user can use the provided “Pause Menu” to return to “Main Menu” and quit. There will be no “Exit to Windows” option in in game menu. User can use “ALT + F4” shortcut during the game but it might cause some data loss depending on the last save. Since Feeding Bobby will work on full screen , there will not be “X” button at the upper right unlike windowed games.

Error

If an error occurs that game resources could not be loaded such as sound and images, the game will still start without images or sound.

3. Subsystem Services

In this section, we intended to provide detailed information about our subsystem interfaces.

3.1 Design Patterns