  
  
  
  
CS 319 - Object-Oriented Software Engineering  
System Design Report  
  
  
Animal Uprising  
Group 3-A  
Bora Ecer

Ata Gün Öğün

Albjon Gjuzi

Tanay Akgül

[1. Introduction 3](#_Toc496384277)

[1.1 Purpose of the System 3](#_Toc496384278)

[1.2 Design Goals 3](#_Toc496384279)

[1.2.1 Efficiency 3](#_Toc496384280)

[1.2.2 Portability 3](#_Toc496384281)

[1.2.3 Reliability 3](#_Toc496384282)

[1.2.4 Extensibility 3](#_Toc496384283)

[1.2.5 Tradeoffs 4](#_Toc496384284)

[1.2.6 Definitions, acronyms and abbreviations: 4](#_Toc496384285)

[1.2.7 References 4](#_Toc496384286)

[2. Software Architecture 4](#_Toc496384287)

[2.1 Subsystem Decomposition 4](#_Toc496384288)

[2.2 Hardware/Software Mapping 7](#_Toc496384289)

[2.3 Persistent Data Management 7](#_Toc496384290)

[2.4 Access Control and Security 7](#_Toc496384291)

[2.5 Boundary Conditions 7](#_Toc496384292)

[3. Subsystem Services 7](#_Toc496384293)

[4. Glossary 7](#_Toc496384294)

# **Introduction**

## **Purpose of the System**

Animal Uprising is a 2D strategy/adventure game which aims to provide a well-designed and enjoyable gameplay to entertain the players. The gameplay experience starts with an easy level which will work as a tutorial for the players, so that they can learn the game. However, in order to provide more satisfaction and pleasure the next levels of the game is designed to be more challenging, so that the game will have the player’s attention, and increase the urge to play the game.

## **1.2** **Design Goals**

### **1.2.1 Efficiency**

The main design goal of the system is efficiency, to achieve that, the system must be able to work in high performance. Since, a smooth gameplay is one of the most important feature which increases the player’s urge to play the game, we are going to minimize the memory usage and the CPU usage. In order to achieve that, first, we are going to implement the code in the most efficient way possible. Also, we are going to design the system so that the workload of the objects is going to be nearly balanced.

### **1.2.2 Portability**

Since portability is an important feature for a software to have various users from different platforms, we have decided to implement the game in Java, since it provides platform independent software which will make our system portable through various platforms.

### **1.2.3 Reliability**

Our system will be reliable in terms of being consistent with the boundary conditions. The system should not respond with any unexpected results -like bugs, crashes- which are not specified in the boundary conditions. In order to provide that, we are going to test the system in all possible ways during and after the development stage. Also, the boundary conditions will be selected carefully and with caution so that there won’t be a case with which puts the system in an unexpected situation. This will provide the system to foresee the possible fatal failures which will be dealt with.

### **1.2.4 Extensibility**

In order to keep the interest of the players of Animal Uprising, the game requires to have new features and functionalities. For that the design of the game must be suitable to add further improvements and additions to the current system. In order to achieve this, object oriented architecture of our game must be designed in a way that each object should be able to operate with few dependencies. So that the modifications and further additions won’t cause any bugs or crashes.

### **1.2.5 Tradeoffs**

* **Functionality vs Usability**

Our game focuses on usability more than functionality since our main goal is to entertain the users with a basic game. So, instead of developing a complex system, focused on developing a basic system which is easier to use.

* **Efficiency vs Portability:**

Portability is very important for a game to reach wider range of users. Since we use Java, which offers a platform independent program, we satisfy the portability feature, however, since java is less efficient compared to the other languages, in this process we sacrifice the efficiency.

* **Cost vs Reusability:**

We did not focus on reusability while designing our system because we are not planning to use the existing classes of our game in different projects. Therefore, the classes are designed to do their task only for this system which will make our system less complex and prioritizing the cost.

### **1.2.6 Definitions, acronyms and abbreviations:**

JDK: [1] Java Development Kit

MVC: [2] Model-View-Controller

### **1.2.7 References**

[1]: <http://www.oracle.com/technetwork/java/javase/overview/index.html>

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

# **2. Software Architecture**

## **2.1 Subsystem Decomposition**

In this section, we will decompose the overall system into three different subsystems in order to use Model-View-Controller architectural style on our system. By doing that, we would like to accomplish creating a maintainable, efficient and flexible system.

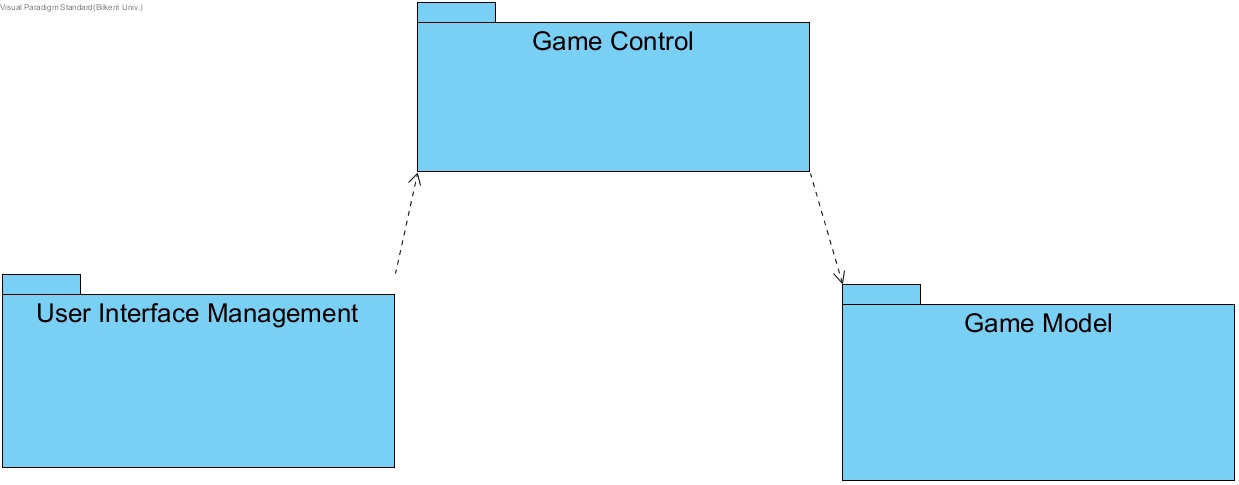
Figure-1 is showing how the system is decomposed into three subsystems which are User-Interface Management, Game Control and Game Models.****

Figure-1 (System Decomposition)

All three systems are working on different tasks, and they communicate as given in the figure. User Interface Management includes the Menu package, and the Game Engine class, which will be responsible the construction of the game screen, according to the inputs from GameManager class which is in the Game Control Subsystem. Game Manager class will be responsible for handling the inputs, and making the decisions regarding to them, these decisions includes manipulating the user interface and the objects. Therefore, the GameManager class will be the façade class for the entire system.

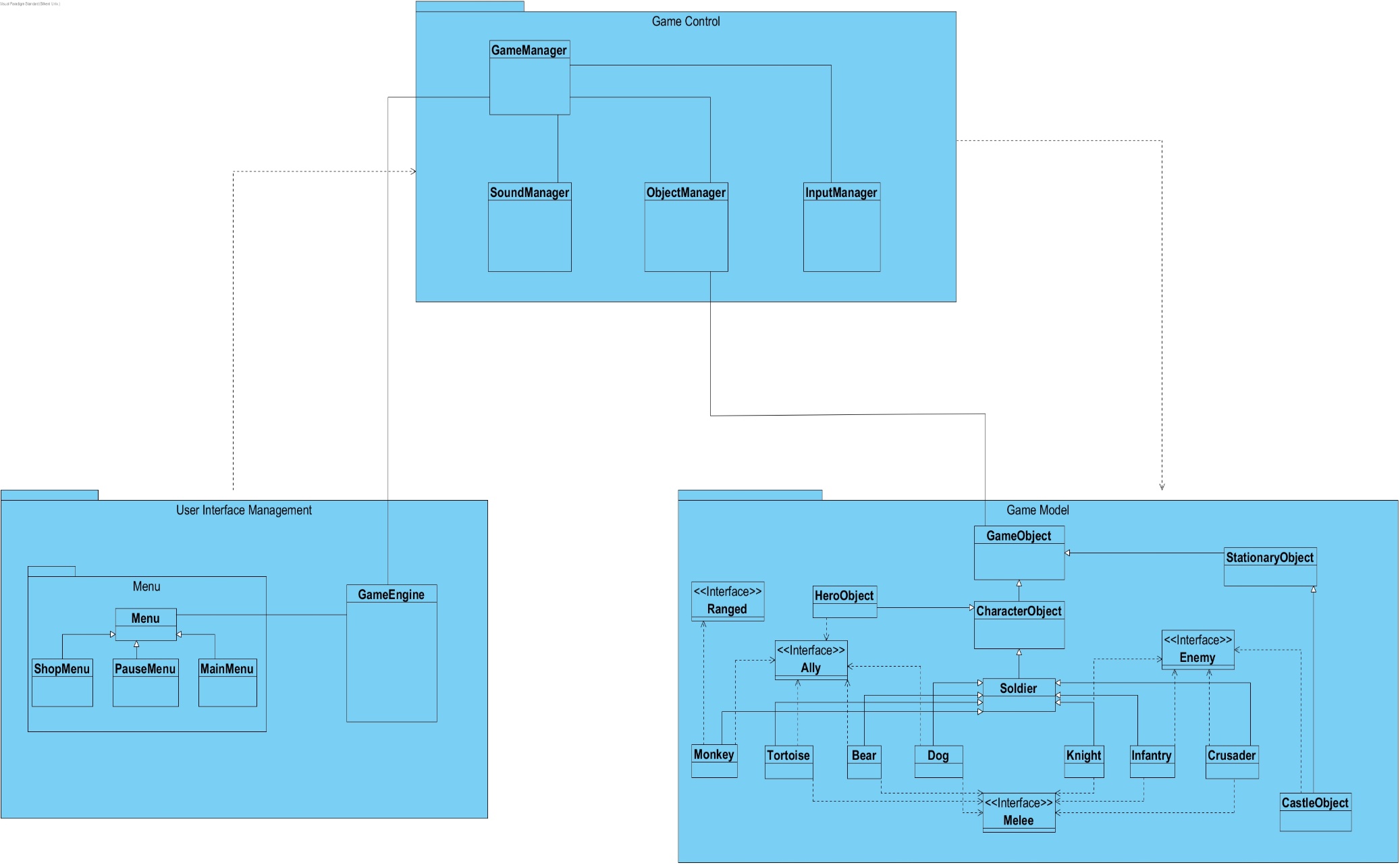


Figure-2 (Detailed System Decomposition)

Game Models Subsystem which has the game objects and their attributes. Because of this, the Game Control Subsystem, mainly ObjectManager class will be able to manipulate them.

## **2.2 Hardware/Software Mapping**

Since our game will be implemented in Java, we will use the JDK 8. As for the hardware requirements, a keyboard and a mouse will be required so that the player can interact with the game. Since we are not planning to implement a complex system, a basic computer should be enough to run our game.

## **2.3 Persistent Data Management**

Our system does not require a complex database for managing the data, therefore we are planning to store the game data in the client disk. So, whenever the system executes, we are going to load the files which are required for the system, to memory. Also, we are planning to store the images (gifs, and etc.) also the sound effects of the objects and the background music of the game.

## **2.4 Access Control and Security**

Since the game does not require any network connection, therefore we will not implement any user authentication system. For software control, the GameControl subsystem will be able to access the files in order to assure the security of the data required for the game.

## **2.5 Boundary Conditions**

**Initialization:** The executable file of the game will be .jar file so it will not require an install. At startup time, the GameControl subsystem will be registered. The GameControl will access the data which is required for the user interface, and the sound effects.

**Termination:** The game can only be terminated from the main and pause menus, which will have a “Quit Game” button, and it will require user input. When the game is terminated, the Game Control subsystem will automatically inform the other subsystems to terminate as well.

**Failure:** During a communication failure, the system will check the files or the data required are corrupted or not, if it is the system will erase the contents of these files.

# **Subsystem Services**

**Detailed Object Design**

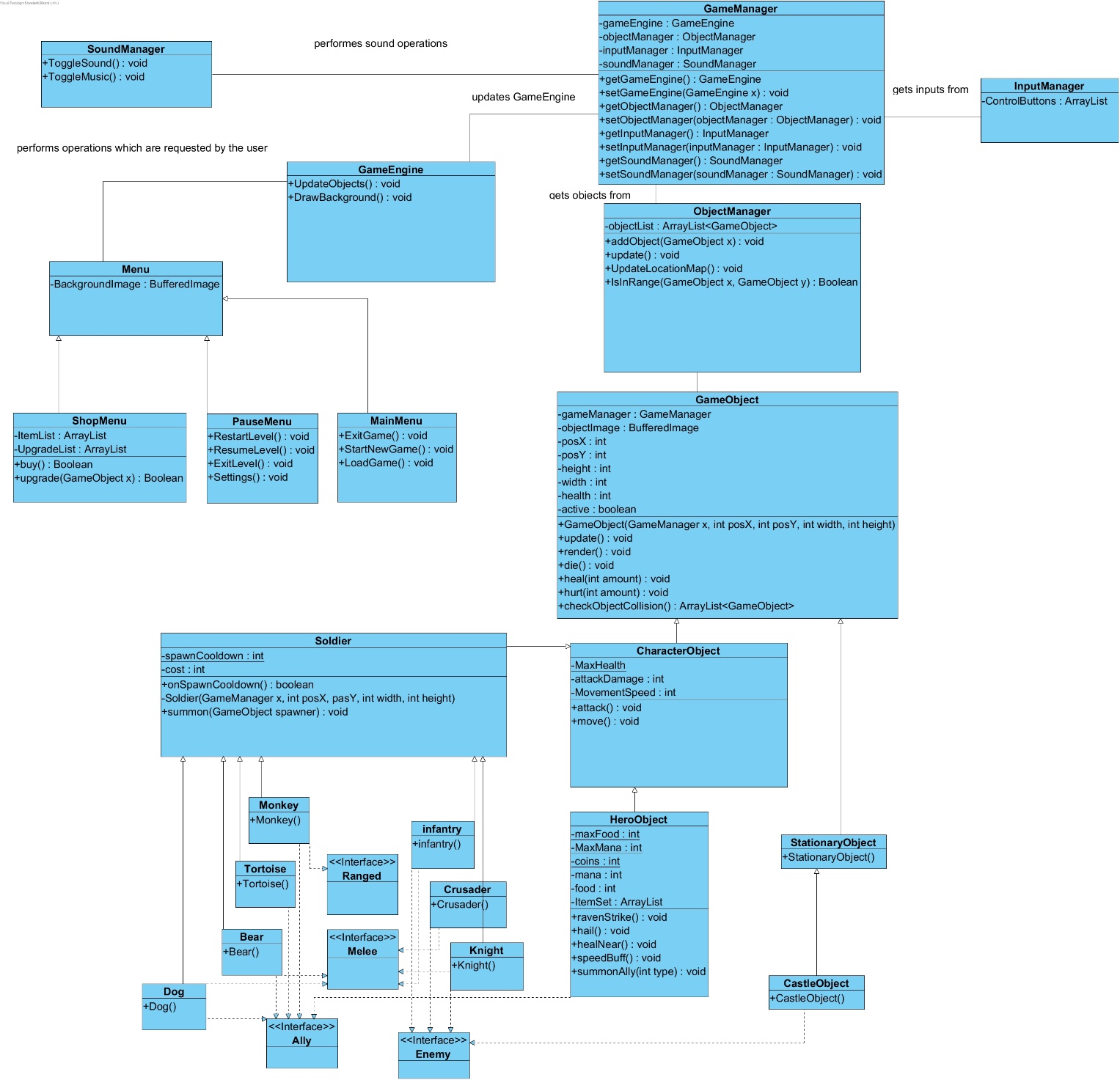


Figure-3 (Detailed Class Diagram)

**GameManager Class**

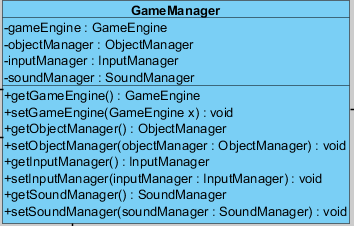


Figure-4 (GameManager Class)

GameManager class, has ObjectManager, InputManager, SoundManager and GameEngine objects as attributes. Because of that, GameManager class is the façade class for the entire system. it will handle the inputs, and manipulation of both Game Objects, User interface with respect to the inputs. The GameEngine will be able to access the Game Objects, through the GameManager. Same thing applies for ObjectManager as well.

# **4. Glossary**