**Bilkent University**

Department of Computer Engineering

**CS 319 Project**

*Royal Road – 2D Adventure Game*

Design Report

Group Members

* Kadir Ata Dönmez
* Veysel Alperen Ceylan
* Yakup Kılıç

Contents

[1. Introduction 3](#_Toc468227211)

[1.1. Overview 3](#_Toc468227212)

[1.2. Design Goals 3](#_Toc468227213)

[2. Software Architecture 4](#_Toc468227214)

[2.1. Subsystem Decomposition 4](#_Toc468227215)

[2.2. Hardware & Software Mapping 6](#_Toc468227216)

[2.3. Persistent Data Management 6](#_Toc468227217)

[2.4. Access Control & Security 6](#_Toc468227218)

[2.5. Boundary Conditions 7](#_Toc468227219)

[2.5.1. Initialization Boundary Condition 7](#_Toc468227220)

[2.5.2. Termination Boundary Condition 7](#_Toc468227221)

[3. Subsystem Services 8](#_Toc468227222)

[3.2. User Interface Subsystem 9](#_Toc468227223)

[3.3. Game Management Subsystem 9](#_Toc468227224)

[3.4 Game Entity Subsystem 11](#_Toc468227225)

[3.5 Game Input Management 13](#_Toc468227226)

[4 Low-level design 14](#_Toc468227227)

[4.1 Object design trade-offs 14](#_Toc468227228)

[4.2 Final object design 15](#_Toc468227229)

[4.3 Packages 15](#_Toc468227230)

# Introduction

## Overview

Royal Road is a 2D adventure game. Comparing to 3D games with high quality graphics, Royal Road looks poor, but Royal Road’s fundamental goal is entertainment. In order to achieve this, system should be designed well and have user friendly interface. This game will be packaged with simple, easy to learn documentation to help users learn how to do.

## Design Goals

Some of our design goals are inherited from non-functional requirements of our system that are provided in analysis report. The important design goals of our system are described below.

* **Game Performance:** Royal Road would require holding low latency from each key so as to make the playability as smooth as possible. Object rendering would also be a concern for the performance since game will offer higher number of objects on the screen simultaneously.
* **Graphic Performance:** As explained above, rendering them as efficiently as possible is a notable to improve graphic performance as much as higher.
* **User friendly Interface:** The user should know what is going all times, that means the UI ought not be included with useless info and make it easy for user to figure out what is going on at their screen while they are entertaining the game.
* **Ease of Learning:** Game player doesn’t have to have so much knowledge, since game’s logic is not complicated. Also, in our system there will be provided an instructive help document.
* **Reusability:**  The game has different maps and enemies so that users can create their own special replayability. This kind of modular system can allow the game to be modified and extended in the future.
* **Robustness:** Game-breaking bugs are so frustrating; therefore our system should be designed to handle with unexpected user behaviors.
* **Extendibility:** Developing with adding some new feature is so important. Our system should be designed to be suitable to add some new features to get interest from users.
* **Portability:** If software can be worked in so many systems, it helps to get spread in people. Our system is implemented in Java and our system works in JVM provides platforms.

**Trade Offs:**

* **Size vs. Performance** Our system will be focused on using best algorithms to give us the lowest time complexity, and memory space usage priority will be lower. Nevertheless, there will be works to avoid useless memory space.
* **Functionality vs. Robustness** Having a lot of functionality may cause more bugs and breaking points demand to users’ uses. The expanding game content will be restricted for the sake of system robustness.

# Software Architecture

## Subsystem Decomposition

We have two layers for architecture of our game. First layer include “MenuPanel”, “EnemyPanel”, “SettingsPanel”, “CreditsPanel” and “InstructionPanel”. In this panel as we see, there are some dynamic data and some static data in our classes and panels. “MenuPanel” include static data which are for choosing option for user. “EnemyPanel” is like “MenuPanel” and include information of enemies in the game statically. “InstructionPanel”is static. “InstructionPanel”include information about how user can play game and give information about game. “CreditsPanel” and “SettingsPanel” are different than other panels because they include dynamic data.” SettingsPanel” give option to user for that user can choose option which is related how he will play game. It is dynamic because data which it includes can change by user and saving. “CreditsPanel” include data which cannot change by user but still include data which are changing by game. When user get credits this panel change and save this new credits so it is dynamic too.

Our second layer includes “State” and “MenuState”. This layer is on the first layer because it is like manager of first layer. MenuState is for controlling which panel will close and which will open. These panels are “MenuPanel”, “EnemyPanel”, “SettingsPanel”, “CreditsPanel” and “InstructionPanel”. “State” is abstract class which includes methods for controlling “MenuPanel” and start game. We can see relation between our panels and states in the figure 2.1.

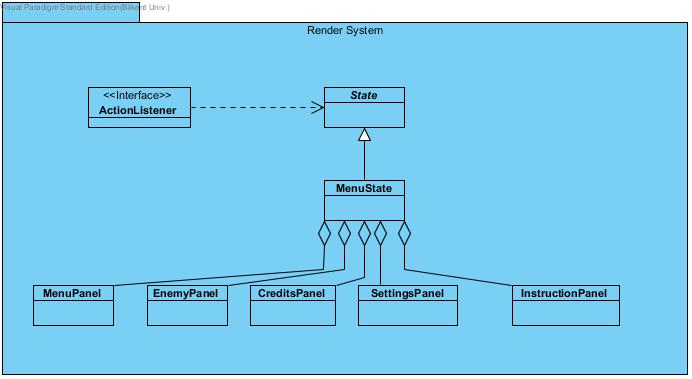


figure 2.1

## Hardware & Software Mapping

We implement our game with language of Java, so user need to installed JDK to his/her machine for playing. Also, his computer needs to have JRE (Java Runtime Environment) as a requirement of running game. User need keyboard and mouse because user must use keyboard to choose option in the Menu of game and to make moving in playing and other control on character of game. Our deployment diagram includes just a main device which is computer. This computer we talking about, should be able to run JRE and it must have GPU will be a plus.

## Persistent Data Management

There are two different types of data in implementation of our game. First one is static data. This data include files which has data for texture and sound of game. During the player play game, these data will not change and cannot be changed by user. Also we have dynamic data during the game execution of system. This data change during the game playing. One of them is when user change setting of game until user exit the game. Also our game will store data of both user setting and levels which user will play. Our game will update these data in the end of each level for user. “Enemies” and “credits” of user will change too, during the game.

## Access Control & Security

In our game, all users have their password and nick names. So we will have Access Control Management for checking these information and we will have txt file to store these information. We will not ask more information to user so there will not security problem for users’ information such as credit card information and so on. Also our game code files and txt file which include information of user will not be public. This will provide us to make our game more secure. Thus, users or other people cannot reach our code to make corruption and they cannot reach users information. Also we have dynamic access control for all our users in the game. This is necessary because of that users must not go and see next level until he/she does not finish level which he/she playing. So we control user accesses to each level dynamically.

## Boundary Conditions

* + 1. Initialization Boundary Condition

User can initialize by open a simple .zip file. Then he/she will see there is .exe file and   text file which include information about game and initializing game. After user click twice to this .exe file, the game will initialize and open.

### **Termination Boundary Condition**

As many program, our program can be quitting by clicking “Quit Game” button which will be always showing on the bottom of game. If there will be any problem with quitting or there will be something needs saving, a pop-up screen will opening and inform user about this. There will not be any automatic saving if user does not finish level he/she playing when he try to quit or program quitting with fatal error.

# Subsystem Services

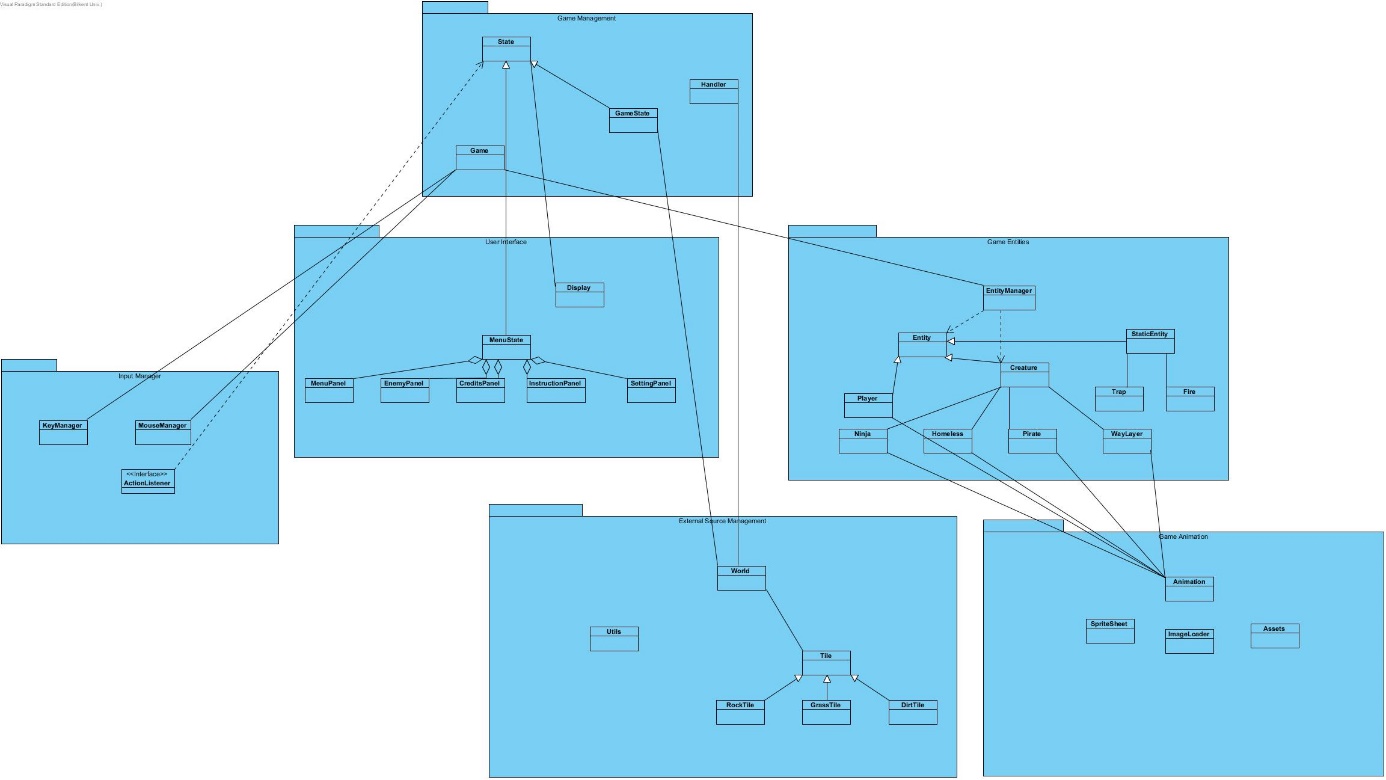
We have four subsystems that aim to reduce the coupling between the systems. This will increase the extendibility and modifiability of the game. By just changing one line, we will be able to modify the game in our desire. To conclude, these subsystems will provide a wide flexibility. 

figure 3.1

## 3.2. User Interface Subsystem

User Interface subsystem consists of view classes which allows user to interact with the system. Furthermore, it sends the user input to controller and displays information about entity objects. View classes consist of MenuPanel, EnemyPanel, CreditsPanel, SettingsPanel and InstructionPanel in figure 2.2.

## 3.3. Game Management Subsystem

Description: Game itself is in this subsystem. Also,it does not contain any game entity or data.

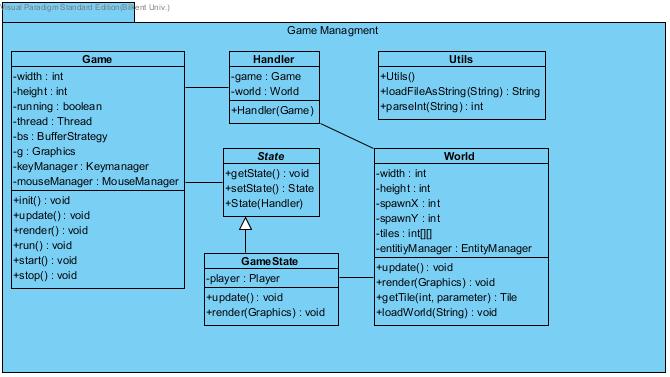


figure 3.1

**Game**

**Description:** It is responsible for initializing entities and world, calling view updates and running the engine thread.

**Thread thread:** UI will have its own thread so that user experiences smoother graphics. This thread will give Handler class its parameters and make Handler update frequently.

**private int width:** It indicates the width of the game display.

**private int height:** It indicates the height of the game display.

**private boolean running:** It checks game’s still running condition.

**BufferStrategy bs:** It leads to how to render objects to the canvas.

**KeyManager keyManager:** It listen to keyboard commands and get them from keyManager.

**MouseManager mouseManager:** It listen to mouse commands and get from mouseManager.

**Handler**

**Description:** It is responsible for keep the classes tidy. Instead of create collision detection each object, it facilitate and make much understandable.

**Game game:** It holds game object and can alter game current.

**World world:** It also holds world object that can be changeable by this class. So as to change level handler class can get the world.

**World**

**Description:** This class hold the level and player’s features at two dimensional array.

**private int width:** It indicates the width of the world.

**private int height:** It indicates the height of the world.

**public update():** It updates the game view according to the game entities that Game holds.

**public render():**It updates the game entities that Game holds.

**Utils**

**Description:** In order to read text file this class used.

**loadFileAsString():** it reads strings from txt by using ‘,’ delimeter.

**GameStates**

**Description:** It update and initiate Game object.

## 3.4 Game Entity Subsystem

Description: It shows the relation between entity objects in the game. Waylayer, Ninja, Homeless, Pirate Classes are inherited from Creature Class.

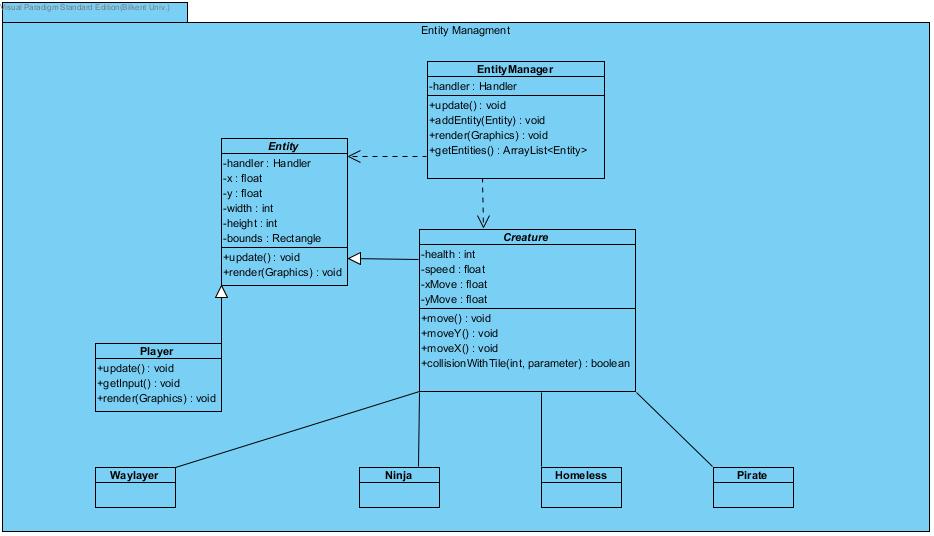


figure 3.2

**Creature**

**Description:** This abstract class is the root of all enemies.

**public move():** This method changes creatures position during the game. It takes X and Y coordinates from each creatures getters.

**Boolean collisionWithTile(int,int):** It takes X and Y coordinates and return true if the creatures intercepted.

**Entity**

**Description:** This abstract class holds the attributes of each entities and determines creatures’ and player’s bound.

**protected abstract float x :** It indicates the x coordinate of the objects.

**protected abstract float y:** It indicates the y coordinate of the objects.

**Player**

**Description:** Player’s features represented by this class.

**public getInput():** According to keyboard it alters player’s move concerning speed.

**EntityManager**

**Description:** This class is created to manage all entity objects in the game. It holds entity and add them into ArrayList<Entitiy>(). It updates each entity one by one.

## 3.5 Game Input Management

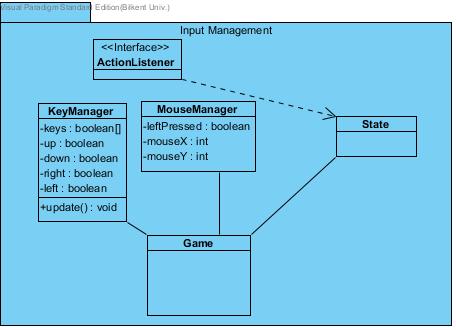


figure 3.3

**KeyManager Class**

This class is designed to comprehend the user actions performed by keyboard. So as to control player’s attributions it requires us to implement two methods: keyPressed() and keyReleased(). The constructor holds Boolean array and they are altered when KeyEvent occurred.

**MouseManager Class**

This class is designed to comprehend the user actions performed by mouse. In order to select menu options and implementing change on them, MouseMotionListener and MouseListener will be implemented. it requires us to implement two methods: mousePressed() and mouseReleased() and also get mouse coordinates we implement getMouseX() and getMouseY().

# 4 Low-level design

## 4.1 Object design trade-offs

1. **Extendibility vs. Cost**

The levels and power-ups of the Dangerous maze are limited. In order to extend the levels and power-ups, developer needs to write extra code. That cause an additional cost.

1. **Understandability vs. Cost**

Understandability of the code is too important especially during the testing phase. Each class and method should be readable. In order to increase the understandability of the code, developer should write comments into the source code. That causes an additional cost.

1. **Development Cost vs. Functionality**

Dangerous maze provides a lot of functions. Each functions of the system require extra design and this causes an extra cost for the development.

## 4.2 Final object design

Our detailed class diagram is shown in below in order to provide a better understanding about the interactions and basic fundamentals of our software.

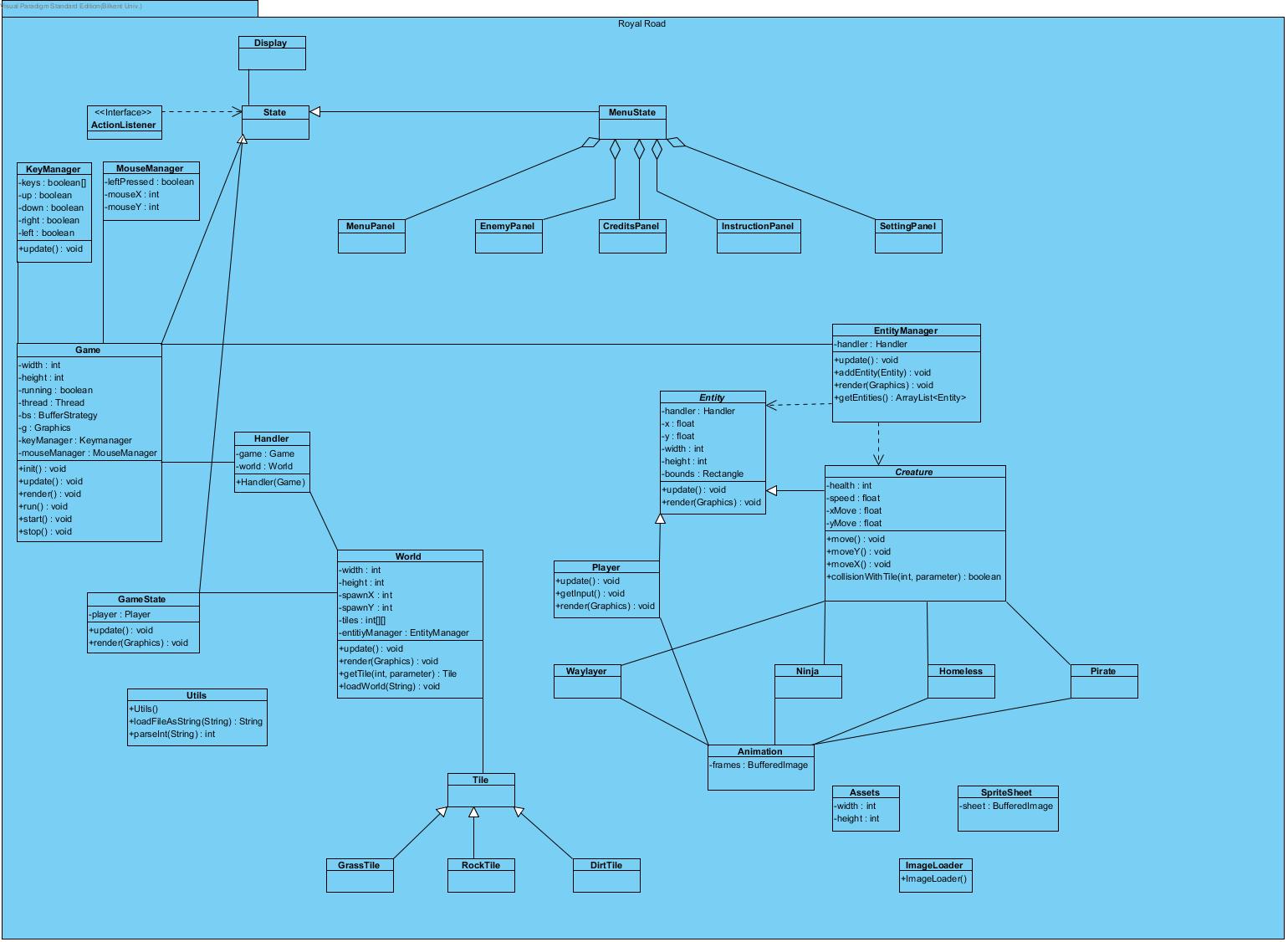


figure 4.1

## 4.3 Packages

We divided our subsystems into 6 packages. These are:

* User Interface
  + Panel
* Game Management
  + Input
  + Game logic
  + Settings
* External Source Management
* Game Entities