# Description: https://lh5.googleusercontent.com/ykiT9AqltWwZhr4yy0I8EMArov3nDMgonw73zz3xpW55Pjnk8xUHN8UQ9cSBvnTYiNAypNe7dAXlTfHdscP_Uzi7ewyNNO03_D5Zd9cClbcOOZwZ3EI

CS 319 - Object-Oriented Software Engineering  
System Design Report

Hurdle Race

Group Members

Rabia Ecem Afacan

Selim Mıdıkoğlu

Haluk İncidelen

Contents

1. Introduction 3

1.1 Purpose of the system 3

1.2 Design Goals 3

End User Criteria: 4

Maintenance Criteria: 4

Performance Criteria: 4

Trade Offs: 5

1.3 Definitions, acronyms, and abbreviations 5

Abbreviations: 5

1.4. References 5

1.5. Overview 5

2. Software Architecture 6

2.1. Overview 6

2.2. Subsystem Decomposition 6

2.3. Architectural Styles 11

2.3.1 Layers 11

2.3.2 Model View Controller 11

2.3. Hardware / Software Mapping 12

2.4. Persistent Data Management 12

2.5. Access Control and Security 12

2.6. Boundary Conditions 13

3. Subsystem Services 13

3.1. Design Patterns 13

Façade Design Pattern: 13

3.2. User Interface Subsystem Interface 14

Menu Class 15

ScreenManager Class 16

MainMenu class 17

PauseMenu class 17

MenuActionListener 18

3.3 Game Management Subsystem Interface 18

GameManager Class 19

InputManager class 21

SoundManager Class 21

Setting Class 22

GameInformation Class 22

GameMapManager Class 23

3.4 Game Entities Subsystem Interface 25

GameMap Class 27

PowerUpTable Class 29

Sprite Class 29

GameObject Class 30

Bullet class 31

PowerUp Class 31

Paddle Class 32

Brick Class 33

NormalBrick Class 34

ExplovsiveBrick Class 34

InvisibleBrick Class 35

HardBrick Class 35

Ball Class 36

3.5 Detailed System Design 38

# Introduction

## 1.1 Purpose of the system

Hurdle Run is 2- D game which includes multiple player game mode as well as single player mode in advance. Two players can compete at the same time. Significant properties of the system are to provide high quality graphical game play process, to create challenge between two players by the futures of Hurdle Run, to create hand-eye coordination according to time and the gameplay scenes and the most important one is to create a enjoyable game for the players. Additionaly the system is created to be easy, both players or just single player can manage to direct the game with couple of buttons and with an efficient interface which we provide.

## 1.2 Design Goals

The main goal of design process comes from the quality, property and the main features that we want from system to have them. So In analysis process we declared our non-functional requirements. We will use these requirements to implement more usable and efficient designed system. The main goals of design given below.

### **End User Criteria:**

The Efficiency of The Managing:

The main aim of the design is to create an interactive system with pure interface with easy usage. The player can manage system with couple of clicks which provides easy basic system usage. These clicking actions include keyboard inputs and Mouse inputs which provides action on the game. Mouse will navigate the menü and the keyboard will manage operations, controls and actions.

The Ease Of the Learning:

The properties and fetures of the game are so easy to understand. But since the use has no clue about them we will provide instructive and informative document which will inform the user how to play the game , what to do in the game or what are the basics of the game.

### **Maintenance Criteria:**

Extendibility:

In the lifetime of the Project it needs to be provided that, the system shoudl be able to get new features, external properties. That is why we create extendibile system which can be added new features in future steps like (new hurdles, new player modes).

Portability:

To make our system platform independent we use java for implementing our system. In further step if our game reaches more and more people, this means our game needs to work in different platforms. JVM[1] solves this. So fort he sake of partability we create our game with Java.

Modifiability:

### **Performance Criteria:**

Response Time:

To not interrept user’s joy and fluent game process we need to response every action immediately on the game stage. This is a must for every single game system. This provides smooth gameplay with no interruptions and immediate responses.

### **Trade Offs:**

### 

Easy and Efficient Usage Vs. Functionality:

Since we do not create any complex game system like users should control system with bunch of different control platforms., we are aimed to create really easy system which user can manage, direct and play easily and efficiently. So we did not add complex components which may cause more functionality less efficient and easy usage or learning. So in this manner our system does not provide too much functionality.

Performance Vs. Memory:

The main component we want to have in our system is performance , since we are working with graphical displays, actions, effects and transitions, the system we want should work smoothly that is why we did not put too much into memory. We only want to keep data of scores, wins and player records. These do not affect performance too much. So we have given more importance to performance more than memory.

## 1.3 Definitions, acronyms, and abbreviations

### **Abbreviations:**

MVC: [2] Model View Controller

JDK: [1] Java Development Kit

JVM: [1] Java Virtual Machine

## 1.4. References

[1] <http://en.wikipedia.org/wiki/Java_(programming_language)>

[2] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010, ISBN-10: 0136066836.

## 1.5. Overview

In this section, we represented purpose of the system, which is basically entertaining the player as much as possible, to achieve this purpose we defined our design goals in this part. Our design goals are determined according to provide the portability, ease of use, ease of learning, high performance, high maintainability. In this respect of course we made some trade-offs to realize our goals. We sacrificed from functionality to make our system simpler and understandable, also we sacrificed from memory to gain performance on playing smooth animations and effects.

# 2. Software Architecture

## 2.1. Overview

In the software architecture section, we separate our software system to subsystems. The purpose of this is making the relationships between subsystems more clear and increasing cohesion of subsystem components. Therefore our main goal is apply Model View Controller(MVC) architectural style to our system by decomposing.

## 2.2. Subsystem Decomposition

The purpose of this part is dividing system into relatively independent parts. By doing this organisation of the system become more clear and understandable. The effects in division is protection the main features of system which are performance, modifiability and extendibility. In order to meet non-functional requirements of our system dividing into relatively independent parts is crucial.

User Interface, Game Management and Game Entities are names of our main focused subsystems which shown in figure-1. These subsystems work on different cases, however they are connected to the each other in the possible changes for future. The details of this design is shown in figure-2 where connection between these subsystems can be seen. For example changes in User Interface will affect the Game Manager because it has a role for control of interface.

The main goal of this section is make the our software more understandable and have a good design before we start implementation of our software. Also dealing with possible changes in our software without errors is important for future.

## 2.2. Architectural Styles

### **2.3.1 Layers**

In system decomposition we divide our system into three layers which are User Interface, Game Management and Game Entities. The hierarchical relationship among these layers is as follows: our top layer is User Interface which is not using any other one, however it is interactive with user action. The layer after comes User Interface is Game Management which has control of our main game logic. The last layer is Game Entities which has our necessary objects. We use closed architectural style in our decomposition in which a layer can only access to layer below. (See figure-3)

### **2.3.2 Model View Controller**

Main purpose of this this architectural system is dividing system to three subsystems which are main, view and controller. In control part we use Game Management class which has main objects of our system. In view part we use User Interface layer which has interaction between user. The advantage os using this architecture is achieving changes on the interface does not affect the model of system.

## 2.3. Hardware / Software Mapping

Hurdle Race will be implemented in Java programming language (latest: Java 8). Our game needs a basic keyboard (for control of player by space and up keys) and mouse for user inputs as hardware configurations. System requirements for our project is a computer with installed softwares like operating system and a java compiler (Eclipse, NetBeans etc.) to write and run java files (.java). Our software is not require the Internet connection.

## 2.4. Persistent Data Management

In Hurdle Race we will store the data in client hard disk drive, we will not use any database, because we use data which is accessed in real time. Moreover, when the system requires; the necessary files will be loaded on the system. Data we use in our system are: background images, objects’ images, buttons for choices and display screen images.

## 2.5. Access Control and Security

In Hurdle Race there is not any personal account and profiles, therefore we do not have any personal security problem. Also Hurdle Race does not require any kind of network connection, only requirement to play Hurdle Race is initialisation of game.

## 2.6. Boundary Conditions

Initialisation

Hurdle Race does not require an install, since game will come with an executable .jar file.

Termination

Hurdle Race can be terminated by clicking “Quit Game” button in main menu. If a user wants to quit while playing, system has a pause button by which player can return to main menu and perform quit.

Error

If there is a problem caused by missing data such as sound or images the game will not start.

# 3.Subsystem Services

Detailed Object Design

The detailed class diagram of out system is shown below in next page(Figure 3.1) in order to provide efficient understanding of the basic fundamentals and connections of the software that we developed. We will divide and tell more about subclasses of our system in the following sections.



Figure 3.1 –Detailed Class Diagram

# User Interface Management Subsystem

The interactions between user and the system provided by couple of classes. First one the façade class is “MainMenu”. Display Menu, Options Menu, Credits Panel and Pause Menu are used to show submenus such getBack, Pause, Help, Sound settings exc.

# osX:Users:selim:Desktop:Ekran Resmi 2016-04-09 14.46.30.png

Figure 3.2 - The interaction between “DisplayMenu” and “MainMenu”

# osX:Users:selim:Desktop:Ekran Resmi 2016-04-09 14.52.10.png

Figure 3.3 - The interactions between “CreditsPanel”, “PauseMenu”, OptionsMenu” and “MainMenu”

# Main Menu

# osX:Users:selim:Desktop:Ekran Resmi 2016-04-09 14.55.49.png

Figure 3.4- “MainMenu”

“MainMenu” is the first class that will be instantiated when the game is first executed and displays the main menu. There will be five buttons in our main menu screen which is “single player mode”, “double player mode”, “help”,”quit”, “settings”. In this Class as soon as we start the game we will create object of the gameEngine class. Menu buttons will be created with the help of the JButton. UIMS are not just “MainMenu”, additionally we have “DisplayMenu” class and other “pause” ,”sound setting” classes.

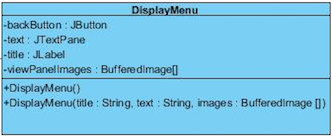


Figure 3.5 – “DisplayMenu” class

Game Logic Subsystem

Game Logic Package is aimed at providing control and detection of game mechanics such as runner is running or runner hits the hurdles. We control these mechanics with “GameManager” and “GraphicVisualizer” classes.



Figure 3.6- “GameManager” class

“GameManager” is facede class of the whole system which handles controlling all game and game logic by using its methods. “GameManager” class repositions the runners, runways, hurdles and background pictures such trees, audience, clouds and birds. It iterates a loop which provides a game play. In every step that we play the game again and again it iterates the loop again. updateObjects() method works when runner hits the hurdle, the scene of the play will be interreptud by some sort of stop or slowing down. Aftet the loop cycle with the help of the detections of “GameManager” class, “GraphicVisualizer” will draw the new objects on to the screen.

GraphicVisualizer class

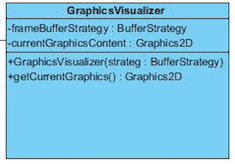


Figure 3.7- “GraphicVisualizer” class

“GraphicVisualizer” class provides rendering operations of our system. But since we do not change hurdle on the screen we will use it for runner to slow down when he hits the hurdle. By double buffering techniques we will make our runner slowed down. “GraphicVisualizer” class will be used by “GameManager” class.

# Game Screen Elements Subsystem

“Game Screen Elements Subsystem” indicates the objects of game which are displaying on the screen while the program is running. There are many objects which are: “GameObject”, “Runner”, “Background”, “Hurdle”, “Runway”, “Audience”. It also has subclasses of these which are: “NormalHurdle” and “BonusHurdle”. These classes also have child classes which are “WaterHurdle”, “WoodenHurdle” for “NormalHurdle” and “SilverHurdle”, “GoldHurdle” for the class “BonusHurdle”.

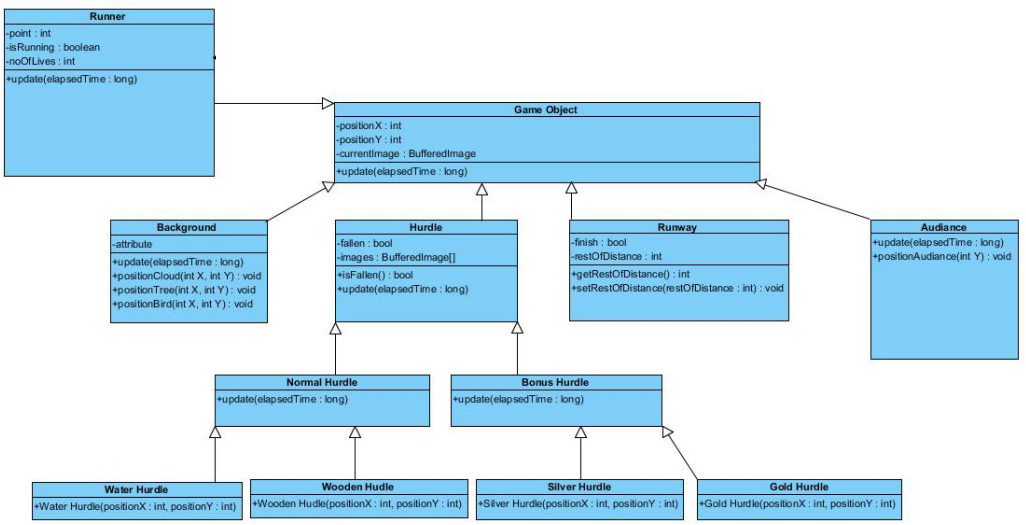


Figure 3.8- Package Diagram of Game Screen elements

## GameObject Class

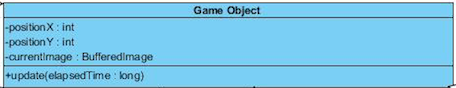


Figure 3.9- “GameObject” class

“GameObject” will be instantiated when user starts to play game. All of the fundamental objects of the game is child classes of “GameObject” abstract class. Since all objects have to have location information, this class has “positionX” and “positionY” attributes. “GameObject” also has “update()” operation which is used by other objects differently such as time, score or success on the hurdles are some factors for different objects. We need to draw all objects on the screen, therefore we use “GraphicsVisualizer” class for “GameObject”.

## Runner Class

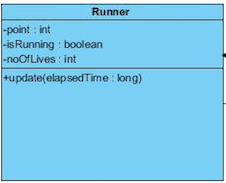


Figure 3.10 – “Runner” class

“Runner” class has two additional attributes to the “GameObject” class. “point” is one of them which returns the score of this runner as an integer value. “isRunning” is another attribute which returns the boolean value related to whether runner is running or not. Methods of “Runner” class are as follows: “getPositionX()”, “getPositionY()” and “setPosition”. “getPositionX()” and “getPositionY()” are for getting position information about the runner and “setPosition” is for setting position information of the runner.

## Background Class



Figure 3.11 – “BackGround” class

“Background” class has image of the background and flows an the screen while game is playing. Image of background has the pictures of the sky, trees and clouds.

## Hurdle Class

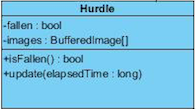


Figure 3.12 – “Hurdle” class

“Hurdle” class has one additional attribute to the “GameObject” class which is “fallen” and has boolean value related to whether runner hit to the hurdle or not. “Hurdle” class has a method “isFallen” and this method returns a boolean value. “NormalHurdle” and “BonusHurdle” are child classes of “Hurdle” class which has different amount of points and penalties.

## Runway Class

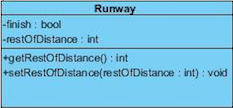


Figure 3.13 – “Runway” class

“Runway” class has an additional attribute to the “GameObject” class which is “singleOrDouble” and has an integer value related to play mode of the game.

## Audience Class

“Audience” class has image of the audiences and flows an the screen while game is playing.

# Input Management Subsystem

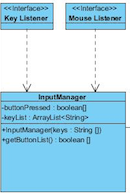


Figure 3.14 – Package Diagram of the Input Management

Input Management Subsystem is responsible for inputs of system. This subsystem takes user’s commands and makes decision about other subsystems’ processes.

## Input Manager Class

“InputManager” class is a listener class of the system. It allows user to control of system by keyboard and mouse.

# Description of the Interactions between Classes According to the Use Cases

“MainMenu” class has the main method which will be instantiated for all of the use cases. This method constructs all the Buttons and Panels for the main menu which is shown at the beginning of the program. “GameEngine” object will be created when the program starts and waits for the instructions from “MainMenu”.

**PlayGame:** When user wants to play a game the “MainMenu” will be displayed on the screen. User should select the player mode between two choices on this screen. These two modes are “Single Player” and “Double Player”. When user makes decision on this “MainMenu” will call startGame() method. Afterwards, updateObject() and updateTimer() methods will be called; and isRunning() will be true if user do not interrupt the game by selecting pause while game is playing. If runner hit the hurdle while running decreaseLife() and decreaseScore() methods will be called. Depends on the type of hurdle generateBonus() will be called which is for bonus hurdles such as gold hurdle and silver hurdle. “ isGameOver() checks number of hurdles and score of each runner in order to decide the run is finished or not.

**ChangeSettings:** The main menu will switch from JPanel “mainMenuLayout” to “settingsLayOut”. loadGameSettings() will be called when user want to change settings. User can select night and day background options and also sound on and off options from this screen. If sound botton is pressed then player will turn on/off the music. When the user back to the game saved settings will be continue; however each time our program will be start default setting will be shown.

**ViewHighScore:** Our program will allow user to see best score of previous games.

**LoadGame:** The load game button in the "MainMenu" will only be enabled if

saveDataExists() function returns true. Then the same procedure with the use case PlayGame except, this time, "MainMenu" calls the startGame() method of the “GameManager”.

**Pause Game:** In this use case isPaused attribute of the "GameManager" will be set to true,

which will pause the inner game loop and wait for the user to continue.

## 4. Glossary

UIMS : Abbreviation of “User Interface Management Subsystem”

FMS : Abbreviation of “File Management Subsystem”

GMMS : Abbereviation of “Game Map Management Subsystem”