Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering Project Report

oject short-name: Curve Fever

Design Report

Zeynep Delal Mutlu, Yunus Ölez, Barış Poyraz

Course Instructor: Bora Güngören

Design Report
November 12, 2016

Table of Contents

1. Introduction

1.1. Purpose of the System

Curve Fever is the replica of the game “Curve Fever 2”. Our purpose is to create a game such that everyone playing from all age groups can enjoy and have entertainment. As the complexity of the system increases, it leads to complex implementations which will have a bad effect. Therefore, the system that will be designed by us, will not be hard nor easy.

1.2. Design Goals

1.2.1. Reliability

We want our game and system to be reliable. It should not crash or give an error in runtime. Therefore, the system should be well designed.

1.2.2. Modifiability

The system we will design should be modifiable. This means that, the design must allow for modifying each component without modifying the other components. By using the object oriented design, we are planning to achieve this goal.

1.2.3. User Friendliness

The game should be easy to adapt and should not provide any conflicts. To achieve this, we should provide user friendly structure. Since we are wanting this game to be enjoyable for all the age groups, the logic of the game should not be complicated.

1.2.4. Good Documentation

The system has to be well documented for the developers. In order to achieve this, the reports should be clearly written.

1.2.5. Response time

The response time is one of the most important aspects of this project. Since our project is an interactive game, which consists of multiple players playing at the same time from the keyboard, there should be no delay in the game to provide better gameplay. For example, the system must be designed in such a way that, when the input is taken from the keyboard, it should immediately affect the movement of the player and update the view accordingly.

1.2.6.Readability

We aim to achieve writing readable code since writing readable code will allow us, the developers, to understand the source code and make changes accordingly.

1.2.7. Adaptability

Curve Fever should be playable in different environments. We will develop this system using Java. Thus, users having a Java Runtime Environment can play the game.

2. Software Architecture

2.1. Subsystem Decomposition

To be able to achieve our goals, system decomposition, which will discussed in details within the name of subsystem decomposition, should follow the design and provide all our classes in a proper, organized way. Subsystem allows us to collect all classes that have associations, events and allows us to relate with each other. Therefore, classes with similar functionalities are put in the same subsystem components. This is done with respect to Model - View - Controller architectural style.

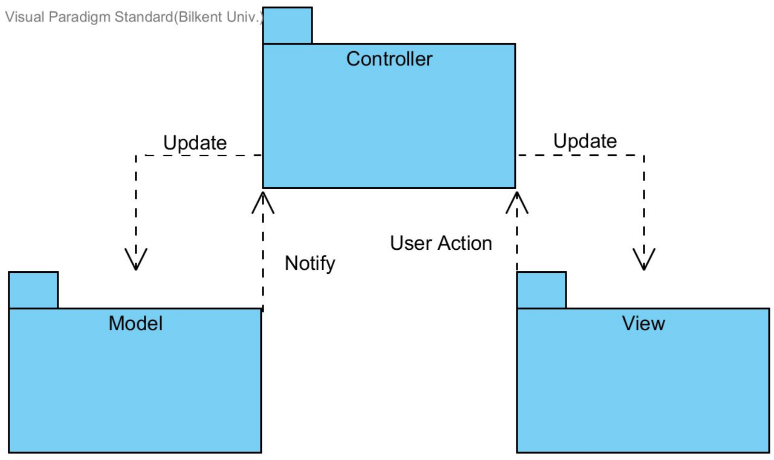
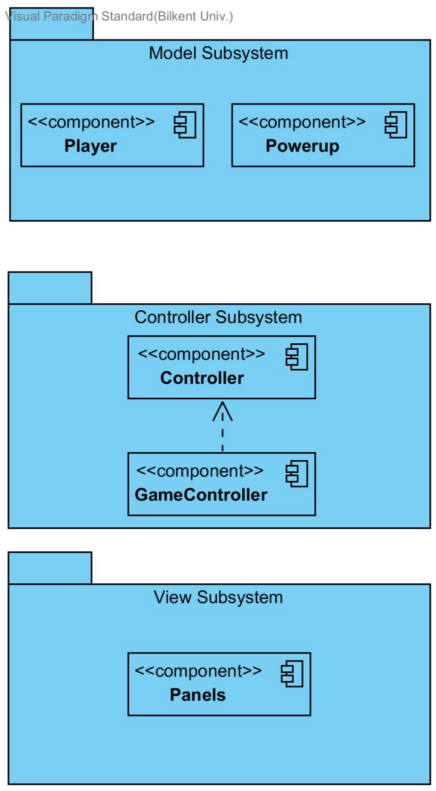
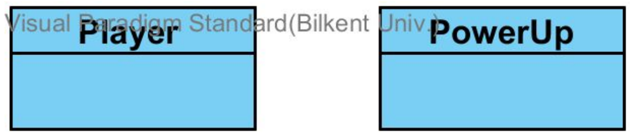


Figure 1. System Decomposition
Model - View - Controller Architectural Style is shown.

In the figure above, MVC architectural style is shown. As the name suggests, MVC divides software application into three parts: Model, View & Controller. These parts have some relations with each other. View contains any output which can be seen by the user and it may allow to get some inputs, which can be said user actions. The user action is taken by the controller which accepts the input and transmits what happened to both model and view. Model basically represents an object.



2.1.1. Model Subsystem



The first subsystem of the program is the Model unit. The main purpose of this system is to have representations of model objects and relationship between them. Within the model subsystem there are 2 classes. These classes are Player and PowerUp classes. These classes have must have relations with some of the controller classes. Powerup class contains the informations of powerups, which in our game changes the gameplay, resulting in an event-driven game. Player class contains the informations of players and has a relation with both GameController and MoveController class. GameController class contains the positions of the players and MoveController class will contain the information of key configurations.

2.1.2. Controller Subsystem

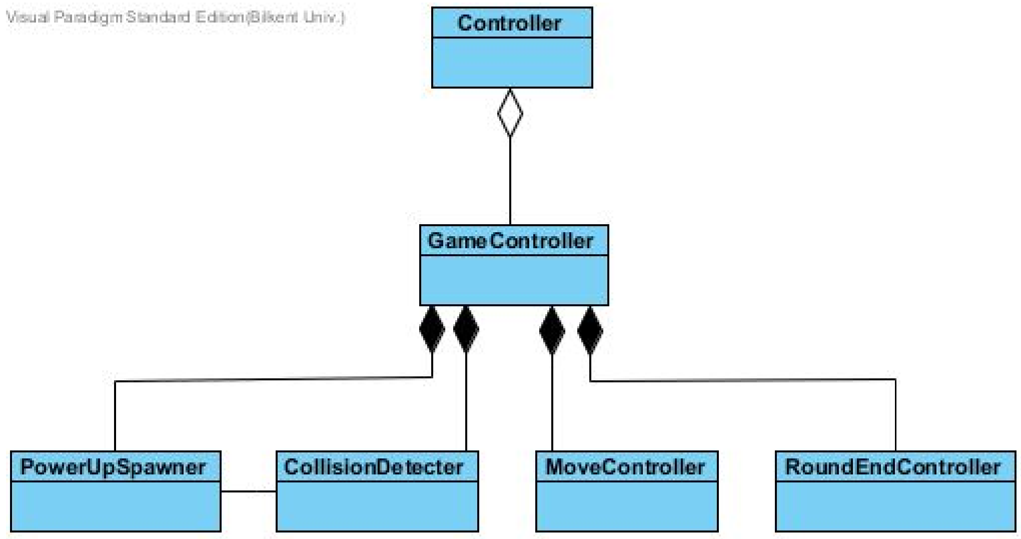


Figure X. Controller Subsystem
This figure consists of 6 classes which are the part of the controller subsystem.

The second subsystem of the program is the Controller unit. The main purpose of this subsystem is to direct the hole system. It accepts inputs from the other subsystems and commands them instructions. It basically consists of 6 different classes with different hierarchies. There are 4 classes, which are linked to the GameController, called PowerUpSpawner, CollisionDetector, MoveController and RoundEndController. These 4 classes handle some cases and inform the GameController in necessary situations. These situations consist of checking whether the round has ended or not, getting inputs from the keyboard, detecting and determining collisions and spawning power-ups at random times and locations. However, these classes only help the GameController class in some cases and overall, the rules of the game are determined and the game is controlled by the GameControler. This class also asks the View Subsystem to create and modify view elements when needed in the game. Other than that, there is a class called the Controller which is at the top of the hierarchic order. This static class is the manager of the whole program and it also communicate with the View Subsystem in order to create new scenes.

2.1.3. View Subsystem

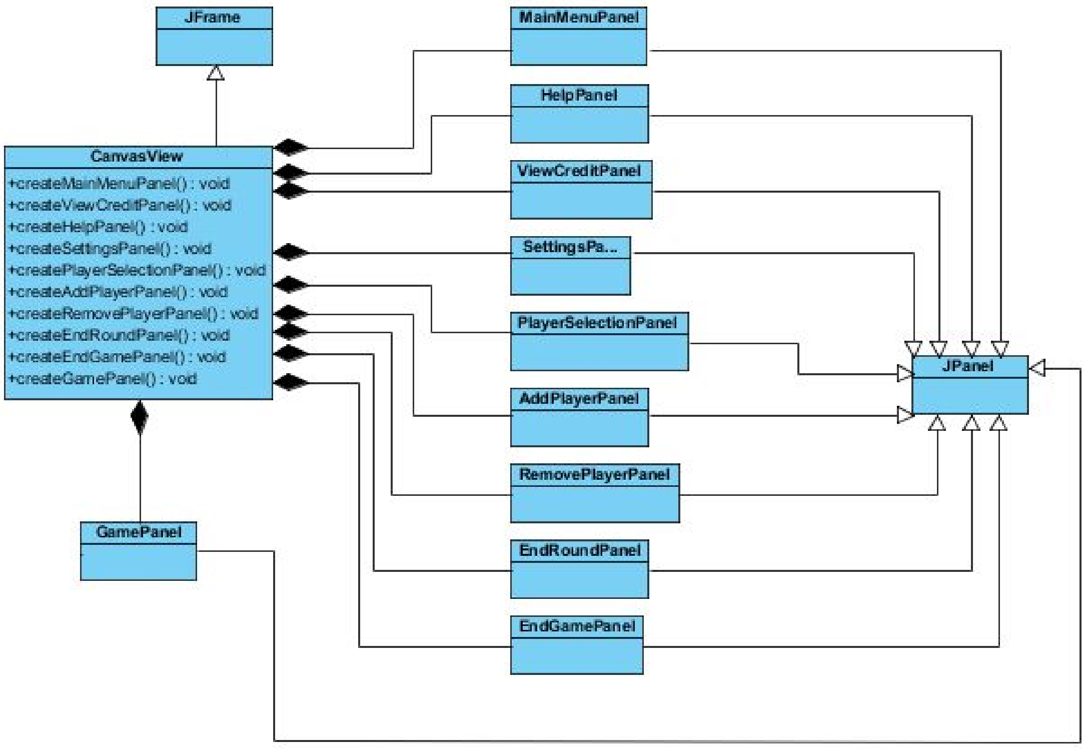

JFrame
MainMenuPanel
HelpPanel
ViewCreditPanel
SettingsPa...
PlayerSelectionPanel
AddPlayerPanel
RemovePlayerPanel
EndRoundPanel
EndGamePanel
JPanel
GamePanel
CanvasView
+createMainMenuPanel() : void
+createViewCreditPanel() : void
+createHelpPanel() : void
+createSettingsPanel() : void
+createPlayerSelectionPanel() : void
+createAddPlayerPanel() : void
+createRemovePlayerPanel() : void
+createEndRoundPanel() : void
+createEndGamePanel() : void
+createGamePanel() : void

Figure X. View Subsystem
This figure consists of 13 classes which are the part of the view subsystem.

Our third and the last subsytem is "View Subsytem". In this subsytem there are 13 classes and the main goal is that creating an interface between play and players. The determinant class here is "CanvasView" class. When any panel is needed in Curve Fever, "CanvasView" class basically gets in contact with this specific panel class and the panel is created. "CanvasView" class works with JFrame. Therefore, after a panel is created, it is embedded into frame.

There are 10 classes which creates panels. Each panel class should use the JPanel class which is used for visual programming. These ten classes' names are easily understanble, even so they should be indicated in detailed as follows:

<u>MainMenuPanel:</u> Creates main menu which includes buttons for adding player, settings and help. When the main menu is needed, "CanvasView" class runs the function 'createMainMenuPanel()'. Therefore, main menu panel is created.

<u>HelpPanel:</u>Creates help menu which inludes 'How to Play' instructions. When the help menu button is clicked in the main menu, "CanvasView" class runs the function 'createHelpPanel()'. Therefore, help panel is created.
<u>SettingPanel:</u>Creates settings menu which inludes certain setting options which will be decided later. When the settings button is clicked in the main menu, "CanvasView" class runs the function 'createSettingsPanel()'. Therefore, settings panel is created.

ViewCreditPanel: Creates credit menu which inludes following sentences:
"This project is part of CS319 course in Bilkent University" and also designers name and also special thanks to our instructor. When the credit menu button is clicked in the main menu, "CanvasView" class runs the function 'createViewCreditPanel()'. Therefore, credit panel is created.

PlayerSelectionPanel: Creates player selection panel which asks the number of players. When the player selection button is clicked in the main menu, "CanvasView" class runs the function 'createPlayerSelectionPanel()'. Therefore, player selection panel is created.

AddPlayerPanel: Creates adding player panel which asks for each player's name, key configuration, and color. When the player selection button is clicked in the main menu, "CanvasView" class runs the function 'createPlayerSelectionPanel()'. Therefore, player selection panel is created.
We are planning to add a new panel class which is called "PlayerScreen". This is called automatically by "CanvasView" after the adding proccess is finished. In this panel, we can see the all players' information, play button and remove button. If player clicks play button, "CanvasView" understands that game will start. If player clicks remove player button, "CanvasView" understands that “RemovePlayerPanel” class will be used as a next step.

RemovePlayerPanel: Creates remove player panel which asks for a player which is wanted to be deleted. When the remove player button is clicked in the "PlayerScreen" panel, "CanvasView" class runs the function 'createRemovePlayerPanel()'. Therefore, remove player panel is created.

EndRoundPanel: Creates the last discontinued round's panel which shows that each player's score in that round. When the round is finished, "CanvasView" class runs the function 'createEndRoundPanel()' which shows the scores each player gained. Therefore, end round panel is created.

EndGamePanel: Creates the completed game panel which shows that each player's overall score in the game. When the game is finished, "CanvasView" class runs the function 'createEndGamePanel()' which shows the scores for each player Therefore, end round panel is created.

GamePanel: is an active playing field. This class works with “GameController” class simultaneously. “GameController” class informs “GamePanel” class for each player’s direction and the game panel will be changed by these inputs. Why do not we use Controller to indicate the direction to “GamePanel”? There are some reasons. First one is about collision detection mechanism. In order to catch collision, “GameController” class checks the coordinate of each player’s head pixel by pixel. That means, “GameController” should take information directly from “GamePanel”. The other reason is about the speed. There are too many instantaneous inputs and outputs between “GameController” and “GamePanel”. If we use “Controller” for each input and output, it makes our game tired.

2.2. Hardware/Software Mapping

Curve Fever will be implemented using Java’s latest version. Therefore, the computer which the game will be runned should support Java, it’s latest version, and have an operation system. Also, in order to play the game a monitor, a mouse and a keyboard will be needed. The monitor will be used to display the game and it’s contents, the mouse will be used to press GUI elements such as buttons to select the color of the player and finally, the keyboard will be used to enter some strings such as names and to move the player with the determined key configurations. Also, in order to save data like high scores and player names into a text file, the system which the program will be runned should support .txt file format.

2.3. Persistent Data Management

At the end of each game the number of games won by the winner will be incremented by one and will be written in a file in order to display that data in the following games. In other words, the number of wins each player have will be saved as a persistent data. To add such a functionality to the program the Controller class should be able to read and write from the .txt files.

2.4. Access Control and Security

Curve Fever does not need Internet and any server connection in order to hold users' datas and to run the game. Addition to that, the game will be played on just one computer. That means there is no need any network protocol adaptation.Therefore, access control is not considered as a security problem.

On the other hand, there should be some control mechanisms to check users' datas. We are planning to hold each player's data in a .txt file(or any similar text file). In order to make this file logically unalterable, we are planning to use some methods. Firstly, the datas should be hashed and then salted. Hence, players' names and scores cannot be seen and changed by someone else. The other control mechanism is about checking name similarity, key similarity and color similarity among players. If either player chooses another player's name, color or keys; our controller will warn the player in order to change his/her information.

2.5. Boundary Conditions

2.5.1. Initialization

There will be an executable file to start the program. When the user opens this file, the system will be initialized. The Controller class have the first code to run and it will send a pulse to the View Subsystem to create and view the main menu. Also, the Controller class will read a text file to load high scores from old games.

2.5.2. Normal Termination

Users can exit the system using the exit buttons in the GUI or directly by closing the program’s window. However, before closing the program the Controller class should handle saving high scores and player names. Therefore, when a user requests to exit the program the Controller should write the data, which includes high scores and names, to a text file to use them in later executions.

2.5.3. Midgame Termination

If one of the players decides to exit the game while playing using GUI buttons or directly by closing the program’s window, the system should not do anything but exit. Since, the user requested to exit while playing the game, there is no need for saving new high score data.

3. Subsystem Services

As discussed earlier, system is decomposed into three subsystems which are Model, View, Controller.

3.1. Services of the Model Subsystem

Model classes which are the representations of objects compose the Model subsystem. It contains important knowledge, background information on objects and it provides necessary informations for the Controller subsystem. Then, Controller subsystem uses this data, gets the data from Model class to handle game operations. For example, in our game Player classes contains the key configurations of the respective player, thus allowing controller to check whether that input is taken or not. If it is taken, it can make corresponding changes.

3.2. Services of the Controller Subsystem

In this subsystem, as it can be seen from the earlier parts of our report, we grouped all of the controller objects to manage game operations. Controller subsystem is the most important part of our system and it can fill the gap between View and Model. When, user interacts the system within a panel, controller is responsible to take the action, handle what needs to be done, notify others and make changes.

3.3. Services of the View Subsystem

This subsystem consists of all panels that user interacts with. Since a user action leads to changes via done by controller, views, in other words panels, should also be changed. To satisfy this, we have set up each screen as a panel. These panels will be created and shown by CanvasView which will either update or create views according to the current state.