
Bilkent University

Department of Computer Engineering

# Object Oriented Software Engineering Project

*Crossit!*

## Design Report

Umutcan Aşutlu, Sarp Saatçıoğlu, Figali Taho, Utku Uçkun

Supervisor: Uğur Doğrusöz

Nov 12, 2016

## Table of Contents


**1 Introduction** **1**

1.1 Purpose of the system 1

1.2 Design goals 1

**2 Software Architecture** **3**

2.1 Subsystem Decomposition

2.1.1 View Subsystem

2.1.2 Controller Subsystem

2.1.3 Model Subsystem

2.2 Hardware/Software Mapping

2.3 Persistent Data Management

2.4 Access Control and Security

2.5 Boundary Conditions

**3 Subsystem Services**

3.1 Services of the Model **7**

3.2 Services of the View

3.3 Services of the Controller

**4 References** **9**

# 1 Introduction

## 1.1 Purpose of the system

Cross-It is a version of the arcade game Frogger. Our aim is to make the game fun and enjoyable for all people willing to play, therefore the system will be designed such that it makes sense and it is easily grasped by everyone.
The goal of the system is that the player passes as many stages as possible to reach higher and higher scores. A finished stage is  a stage in which the player has crossed the road.

## 1.2 Design goals

### End User Criteria

**Ease of use**

Cross-It will be easy to use so the user need not have previous experience in gaming. The actions the user needs to do are intuitive and straightforward. In case of doubt, the help menu will assist the user in understanding how to play the game better.

**User friendly**

Cross-It is a user friendly game. The setup for the user is very simple, the only thing needed to be previously installed is the Java Runtime Environment.

### Performance Criteria

**Response Time**

The game is an interactive one, so we will make sure the response time will not exceed a certain small threshold, i.e there will be no delay in the game.

**Well defined interfaces**

The project will have a well defined interface, with well defined objects such as player, vehicles such as motorcycle, truck and car. There will also be animations for the collisions, or the different collectibles. The game needs to have good memory usage and need to have enough space to improve performance and speed, java virtual machine (JVM) will delete destroyed objects such as bullets, bugs etc

#### Maintenance Criteria

**Reliability**

We plan to create a reliable system. It will not crash or give any errors during runtime. The flow will be smooth, and the user will not experience any unpleasant experience.

**Readability**

The source code will be very readable, so that it is understood by other developers. The source will be self documenting, and the code will look as uniform as possible and with a very consistent style.

**Modifiability**

We will make sure that updates to the system will be possible without much trouble. We, or anyone ready to modify/extend the code, should be able to easily work on the existing system in the future.

# 2 Software Architecture

## 2.1 Subsystem Decomposition

For designing and implementing our system more easily we divided our components into different subsystems. To keep things more consistent we kept components with similar services under the same subsystem. Our system suits the M-V-C (model-view-controller) type of subsystem organization the most.

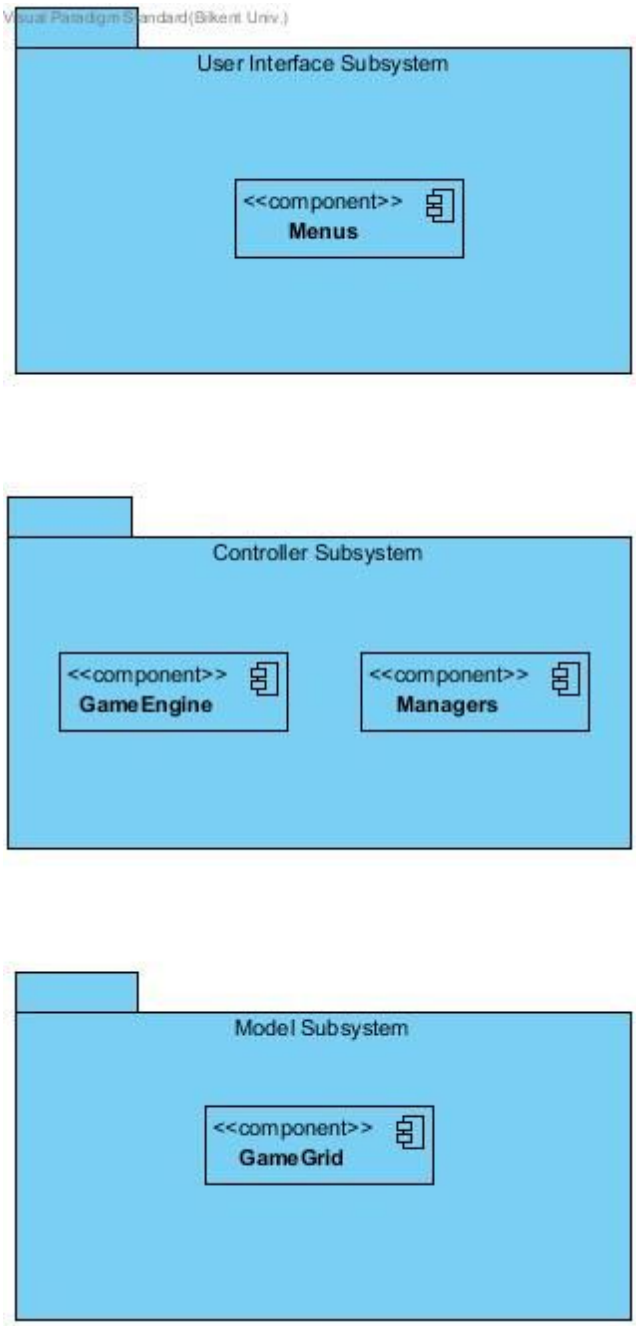


**Figure 1: Subsystem decomposition diagram.** Composed of three parts: Model, View and Controller Subsystems.

### 2.1.1 User Interface Subsystem

User interface subsystem's general purpose is displaying menus in our program. We will use panels to draw these menus on to screen. Game's current state will also be drawn by this subsystem's components.

### 2.1.2 Controller Subsystem

Under Controller Subsystem, we grouped all the components that control and manage system's different functionalities. For example, component that manages settings, files and main controller; GameEngine.

### 2.1.3 Model Subsystem

All of our game objects and the grid that keeps these objects is under model subsystem. Model subsystem represents and manages the current state of the game; where cars and player are at in that specific moment etc.

## 2.2 Hardware/Software Mapping

The hardware configuration of the game will consist of keyboard and mouse. The keyboard will be used by the users to enter the input for moving the player (up, down, left or right), and also for entering the name at the End of Game Menu.
For the user to be able to run and play the game the computer needs to have an operating system with Java compiler installed. Other than the programming language, the system needs to also support .png, .txt and .wav files. The .wav files will be for the game sounds, the .txt files will be used for storing the high score list and the settings of the game, and the .png files will be for the interface objects of the game.
It will be an offline game, so there is no need for internet connection.

## 2.3 Persistent Data Management

For our system we have considered using a .txt file for saving the game data, i.e. the settings, player information and scores will be registered. If the file is somehow corrupted, an error will be displayed, and the system will be reset to default. The jar of the game will contain this file as well as the icons and sounds of the game.

## 2.4 Access Control and Security

Cross-It is a singleplayer game that does not require authentication process to start the game. Since there is no authentication process the game will be playable without any internet connection. All the data about the highscores and pre-chosen selections of settings will be stored in a way that user will not be able to change the content of the file of data externally. The game will not have user profiles, therefore there will not be any pre-condition for starting the game.

## 2.5 Boundary Conditions

**Initialization**

- Cross-It starts by an executable file. Java Virtual Machine starts running in background and Main menu of Cross It will show up.
- In the main menu there are five buttons: New Game, Settings, Highscores, Credits, Help, Exit
- When New Game button is clicked, system will create the first stage of the game.
- If player hits the Help button, Help panel will open with the help content.
- The game graphics are made by Adobe Photoshop Cs6 [1].
- If the Settings button is selected, the settings screen will pop-up to customize the game.
- If the Credits button is selected, the general information about game will be displayed, such that version and authors of the game.
- If the Highscores button is selected, highest 10 scores will be displayed with the names on the screen.
- If the Exit button is selected, the game will shut itself down.

**Termination**

- Player can quit the game by pressing the Exit button on the main menu.
- During the game, if player closes the game directly with any method, game will exit without saving any changes such as scores and coins gained from that game session.
- If player wants to exit with the Exit button on the main menu, a prompt that asks whether player is sure or not will be displayed. If player selects yes, the game will quit regularly.

**Failure**

- The game needs to JRE in the system to launch.
- Although the files that stores information i.e. pre-saved settings, highscores are secured, they may be deleted or somehow unreadable. In that situation the game system will assign new default files with deleting previous ones.
- If somehow there occurs an immediate termination, like power cut, current data will be lost.

- If the game terminates unconditionally during a save process, the save process may not be successful.
- If user deletes anything accidentally, an error message pop-up to notify.

# 3 Subsystem Services

The game system will based on three major subsystems. These are Model, View, Controller. The services of these subsystems provides are:

## 3.1 Services of the Model

The model subsystem is composition of all classes of modeling. The model classes consist the main data of the game, like properties and positions of the objects in the game. Any changes that happen during the game session or change in panels will be send to Controller subsystem to process of the game further. This operation of Controller which is gathering information will be supported by sendData service of the Model subsystem. This service will send current locations of the models, player's outfit etc., i.e. anything about the objects of the game that are in the session. The modify call of the controller subsystem also can make changes in Model Subsystem, so there is a bridge between these two subsystems. During the session of the game, the flow of information will be continuous.

## 3.2 Services of the View

The services of the View subsystem provide is everything that user interacts during the session of run. Every element of visualization will be updated accordingly. The first service that will be used is draw. The visual instances of the game will be updated frequently with high-speed, so every change that differs between previous frame will look natural. The update service will be handled by Controller Subsystem with the information that has been provided by Model Subsystem. As an example, every time any vehicle unit or player changes its location, it will be spotted by update service that continuously called, and informs the controller subsystem. Required changes of information will be gathered from Model subsystem to Controller Subsystem. The general composition of these datas will be send back to View subsystem's draw services and the panels and frames will be drawn and change accordingly.

### 3.3 Services of the Controller

This subsystem is a set of controller objects that are linking the model and view subsystems. Since this subsystem is responsible to manage all the other subsystems, it has access to all other services. The controller subsystem gets the events from view subsystem and manages the model according to these events. For instance, if the player wants to start a new game by pressing the “New Game” button from the main menu, this event will be handled by the corresponding ActionListener that is provided in related controller object. This controller object modifies the corresponding model object and updates the view with respect to that modification. These kind of handling operations are managed by the controller subsystem with two kind of calls to the other subsystem. First type of call is modify, which notifies model subsystem to process changes passed in the event. Secondly, controller calls the view subsystem to update the view, in our case this is user interface.

# 4 References

[1] A. S. Incorporated, "Photoshop: For windows: Adobe Photoshop 13.0.1.3 update for CS6:Thank you," 2016. [Online]. Available: https://www.adobe.com/support/downloads/thankyou.jsp?ftpID=5677&fileID=5701. [Accessed: Nov. 12, 2016].