

CS 319 - Object-Oriented Software Engineering

Design Report

Survivor

Group 2-G

Metehan Kaya

Mehmet Taha Çetin

Ferhat Karaca

Emre Yiğit Kuzhan

1. Introduction

1.1. Purpose of the System

Survivor is a 2D arcade game. The game flow basically goes to finish racecourse without dying. We are influenced from Mario that was popular video game a long time ago. However, we decide to add different properties from Mario such as environmental obstacles. These obstacles affect the player according to the type of it. Some obstacles slow down the player some are kill. Survivor is designed as challenging as possible to maximize the pleasure of player. Survivor aims to improve decision making abilities and cognitive skill of player while having enjoyable time.

1.2. Design Goals

Our design goals inherit from the non-functional requirements that are mentioned in analyses stage. The design goals of our game are described below.

End User Criteria:

Ease of Use: We should offer user to good entertainment because our system is a game. In order to present this entertainment player should not encounter any difficulty during the play our game. Therefore our system will provide user to user-friendly menus that player will reach anything he/she search with ease. Also our system will perform actions according to keyboard input from the user. Our system will use 4 buttons from keyboard. These are directions keys that located on right part of keyboard. This makes it easy to use the system in terms of player.

Ease of Learning: Generally most player are unfamiliar with the game that plays for the first time. So, he/she may not have knowledge about how the game is played, what are the conditions that slowing down the speed of user, the end condition of the game. These are the crucial information to user. Therefore, in order to obtain these information about the game, we will present user to instructive help document. Player can easily understand the game and logic behind it by reading the help document.

Maintenance Criteria:

Extendibility: It’s important to add new components and features to the game in the game software sector. These creative differences makes the game more preferable. Therefore our system will appropriate to add new functionalities such as new obstacle types that affect user easily to our existing game.

Adaptability: Since we will use Java, we will not have to worry about the operating system requirements. Because Java is one of the few programming language that provide user to cross-platform portability. In order to welcome the adaptability feature, we prefer to use Java language instead of C++, C or any another programming languages.

Reliability: Reliability also another important issue in the software development. System will be consistent in the boundary conditions. Even if system encounter with unexpected situations, it should not crash. In order to success this, we will test our game at the each stage of development. Another important thing that we take into consideration is boundary conditions. We will be evaluated carefully to not encounter undesired case that may crash the game.

Efficiency: The game will be responsive and able to run at the high performance. In order to reach optimum performance, we will allocate memory to each individual objects that we will create so that they will be accountable for just their own tasks. This method will improve the performance of the game significantly.

Tradeoffs:

Ease of Use / Learning vs Functionality: In order to reach wide range of customer the game should not be complex to learn and play. We determined that user should be able to learn and play the game easily. So, the priority of usability will be higher than functionality in our game. In other words, the functionality of the game will be basic. In order to entertain user we will focus on the usability rather than functionality of the game. Our system will consist of simple user-friendly interface and well-known instructions to play the game. Thus, player will spend enjoyable time to play rather than struggling to learn it.

Performance vs Memory: The main purpose of our system to provide user smoothness in the movements of the game object. Therefore we will focused on performance of our system. In other words, performance of the game is our priority. In order to achieve this purpose we will sacrificed from memory in order to gain the performance.

1.3. Definitions, Acronyms, and Abbreviations

Cross-platform[1]: Cross-platform refers to ability of software to run in same way on different platforms such as Microsoft Windows, Linux and Mac OS X

Boundary conditions: Conditions of the system which may generate run-time errors. They are exceptional cases according to the normal flow of the program. These conditions must be handled carefully for robustness of the system.

1.4. References

[1] http://www.webopedia.com/TERM/C/cross\_platform.html

[2] http://www.klabs.org/history/history\_docs/sp-8070/ch4/4p1\_design\_tradeoffs.htm

2. Software Architecture

2.1. Subsystem Decomposition

Since the three-tier architecture is the most efficient and suitable design pattern among all design patterns, we are going to use this structure at the implementation stage.

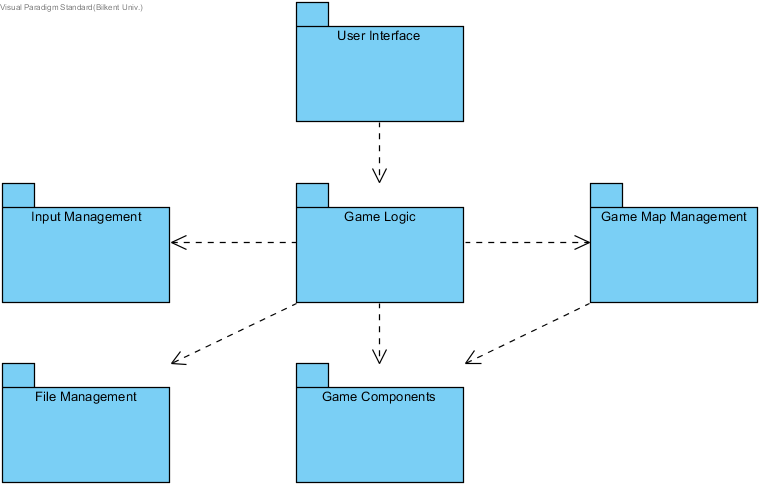


Figure1: Packages in layers

Classes of the first layer, presentation tier, are responsible for providing an interface between the user and game. For instance, “User Interface” which is the only package of the first layer contains “MainMenu” class that is the first class interacts with the user. Then, it depends on the user to work “Game Logic” package.

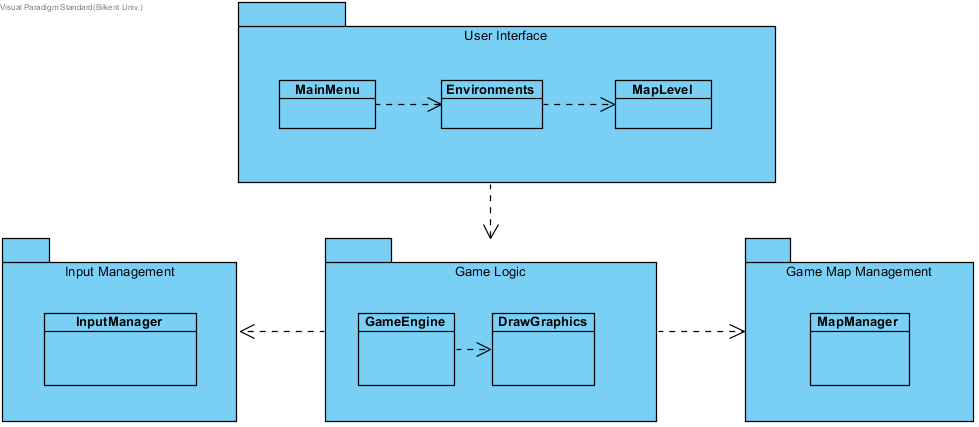


Figure2: 1.Layer and 2.Layer

In the second layer, application tier, “Game Logic” is the primary package. It is connected to the other packages, “Input Management” and “Game Management”. If the “Play Game” option is selected, after some choices of environments and levels, “GameEngine” class constructs the game layout thanks to the “Game Map Management” package and “DrawGraphics” class.

“GameEngine” class communicates with “InputManager” class of “Input Management” package to get inputs from the user. After getting inputs, it leads to do changes in the positions of components.

“Game Map Management” package is responsible for storing the game map including components of the game. The map data will be updated continuously according to inputs, in other words, player’s movements.

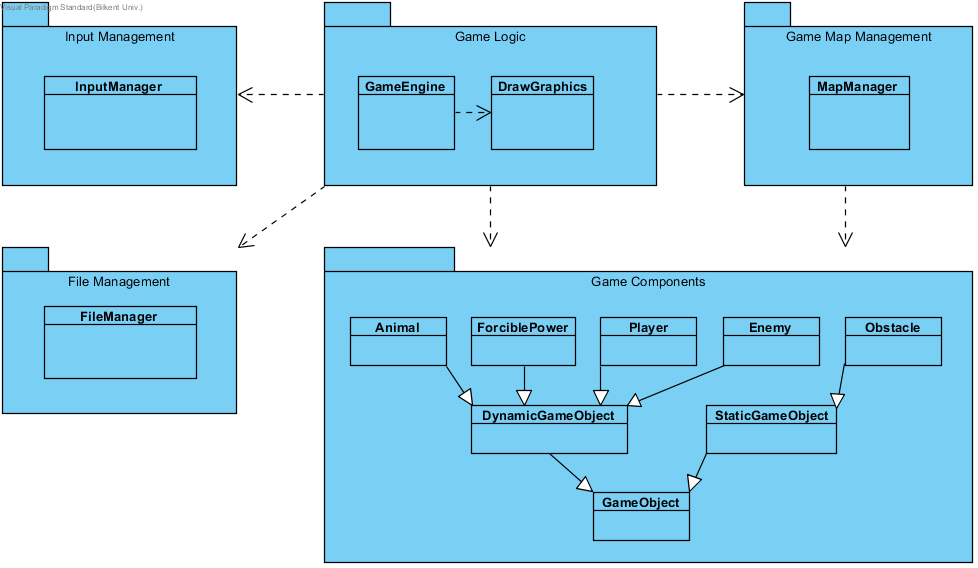


Figure3: 2.Layer and 3.Layer

The last layer, data tier, is comprised of utility classes that are for file operations and component data. “File Management” package deals with saving and loading game data. “Game Component” package deals with objects of the game. This package is responsible for all properties of game objects, including positions, speeds, accelerations, images etc.

2.2. Hardware / Software Mapping

The game requires a keyboard to read the player’s movement choices and a mouse to let the user do some choices.

This is a Java game, therefore, it requires the Java Runtime Environment to be executed. Considering graphical requirements of the game, it does not require high level features. Graphics2D class uses GPU acceleration which shows belonging a GPU is an advantage for the user.

2.3 Data Management

Since we won’t use any database, game data will be stored in the user’s hard disk. Images of the game such as character, background, buttons, animals, boss, etc. will be open to change to let the user improve the attractiveness of the game.

2.4 Boundary Conditions

The player has only one live which makes the game harder and more attractive. In case of death, game will return to a screen that shows the levels of chosen environments. If player passes the level, the next level will be unlocked, and game will return to the same screen. Also, scoreboard will be updated according to player’s point. If it is the last level of the environment, the next environment will be unlocked, and game will return to a screen which is a map of environments.

3. Subsystem Services

It is important to comprehend object design with details to understand how the project works. The detailed subsystem descriptions are provided in the next sections.

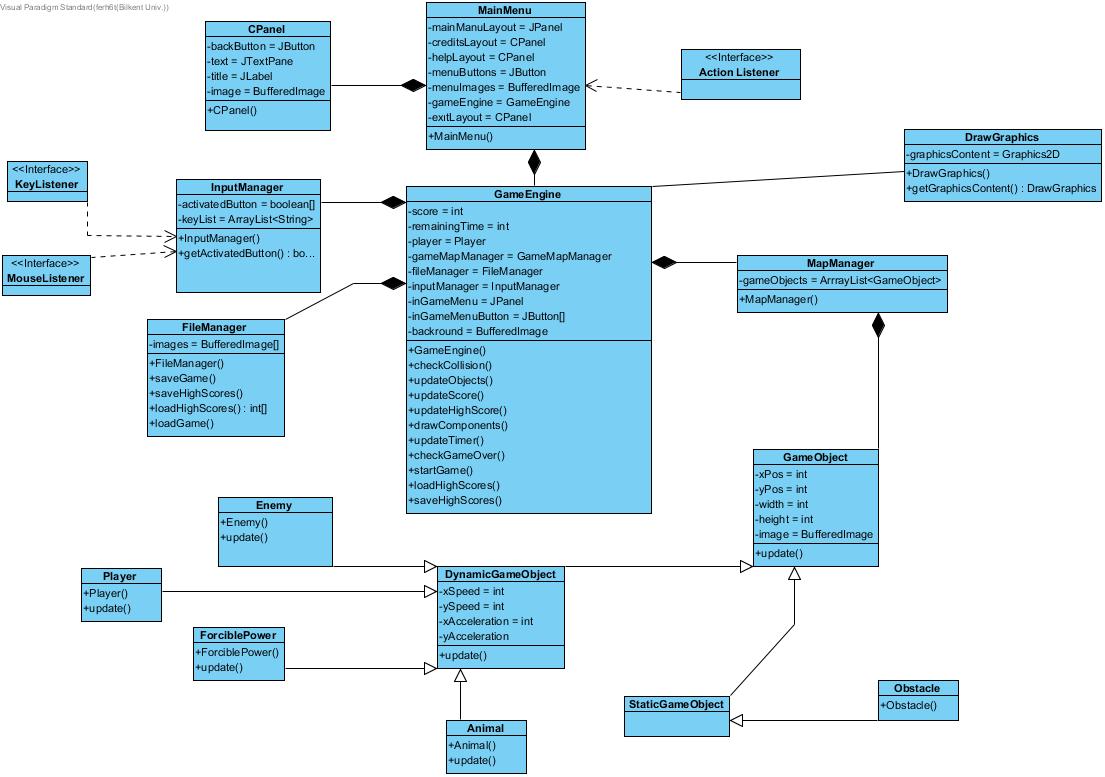


Figure4 : Class Diagram of project

3.1 User Interface Subsystem

We described our first package as User Interface. In the figure below we described our classes in this layer. This package based on menu option.

Actually, the project includes Environments and MapLevel classes in the User Interface. However, we have decided that for the first iteration, it should have only one level for the sake of process of the project. Therefore, there is no need to put these classes in the first iteration.

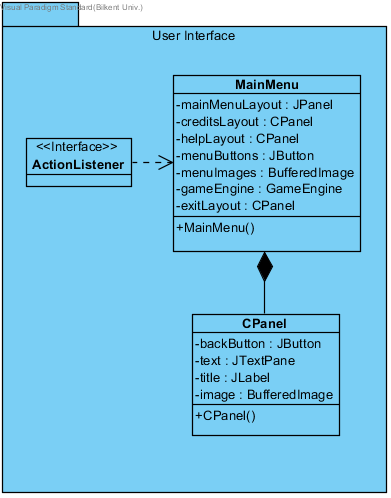


Figure5 : User Interface Package

3.1.1 MainMenu Class

In main menu class we have JPanel and CPanel which we created for creditsLayout and helpLayout. This menu basically shows us main menu.

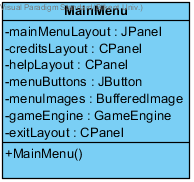


Figure6:Main Menu Class

3.1.2 CPanel Class

In this class we have same logic to show credits, help options. This class will be created for showing some images to inform user.

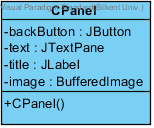


Figure7: Cpanel Class

3.1.3 ActionListener Class

This interface helps program to get mouse clicks.



Figure8: ActionListener Interface

3.2 Input Management Subsystem

In second layer we have Input Management package that will help us to get user’s inputs.

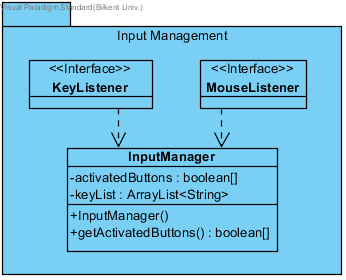


Figure9: Input Management Package

3.2.1 InputManager Class

In InputManager Class, we have a recognizer for keys.

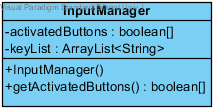


Figure10: InputManager Class

3.2.2 MouseListener Interface

MouseListener Interface, identifies the actions in terms of mouse actions.



Figure11: MouseListener

3.2.3 KeyListener Interface

KeyListener Interface, identifies the action of key actions.



Figure12: KeyListener Interface

3.3 Game Logic Subsystem

In our second layer’ second component, we have Game Logic Subsystem. This subsystem is basically the engine of our game. Creates player, map and holds score for our game.

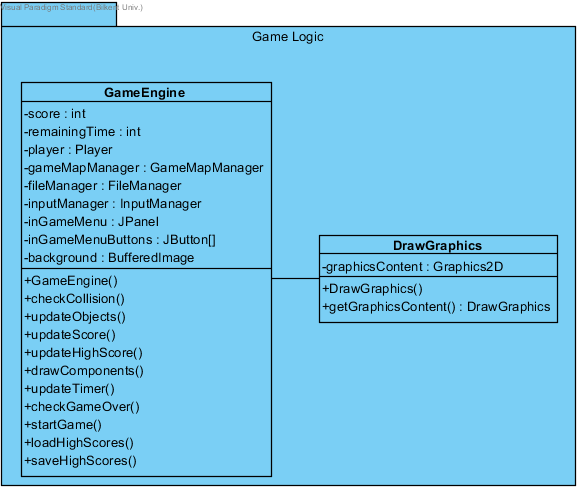


Figure13: Game Logic Package

3.3.1 GameEngine Class

In this class, we implement an environment, by using DrawGraphics class and also this class keeps the score of a player. It also creates a player for our game. This is basically the heart of our game.

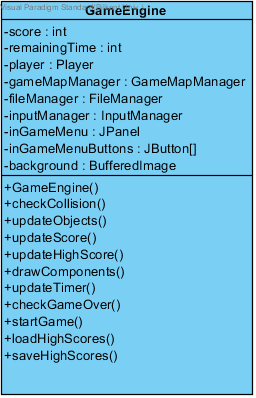


Figure14: GameEngine Class

3.3.2 DrawGraphics Class

In this class, we implement an environment, by using environment class and also this class keeps the score of a player. It also creates a player for our game. This is basically the heart of our game.

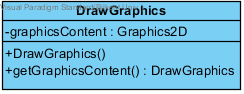


Figure15: DrawGraphics Class

3.4 File Management Subsystem

File Management package generally deals with saving and loading data. It has only one class that is responsible for this kind of operations.

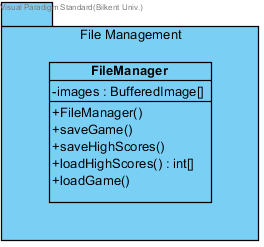


Figure16: File Mamangement Package

3.4.1 FileManager Class

FileManager is the only class of File Management package. Generally, it saves and gets level data and high score.

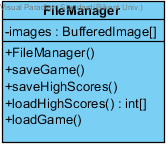


Figure17: FileManagement Class

3.5 Game Map Management Subsystem

This subsystem is the manager of the game’s levels. It stores the objects of current level of current environment.

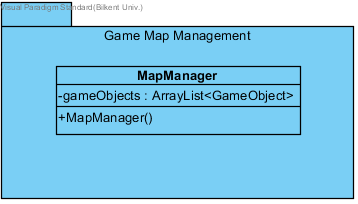


Figure18: Game Map Management Package

3.5.1 MapManager Class

MapManager is the only class of GMMS.



Figure19: MapManager Class

3.6 Game Components Subsystem

Game component subsystem is the base of the game. By this package, we fill the environments.

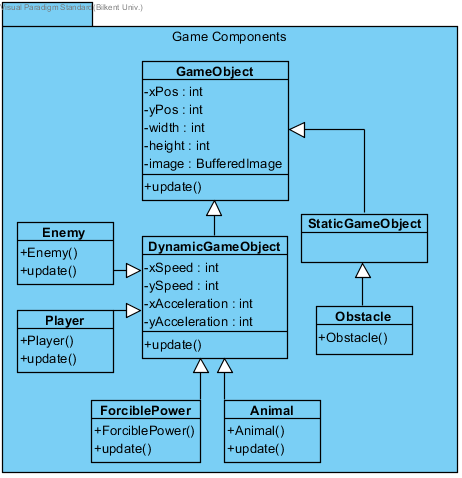


Figure20: Game Components Package

3.6.1 GameObject Class

Game object class is the main part of this package. This class creates whole objects according to the envitonment.

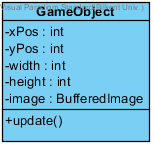


Figure21: GameObject Class

3.6.2 DynamicGameObject Class

Dynamic objects will be our objects that have speed and orbit.

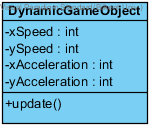


Figure22: DynamicGameObject Class

Player class creates the character controlled by the user.

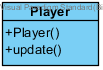


Figure23: Player Class

Enemy class creates some different enemies to make game harder.

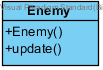


Figure24: Enemy Class

Animal class represents some dangerous animals that cost character’s live and non-dangerous animals which make the character slower.

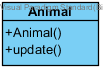


Figure25: Animal Class

ForciblePower class represents natural power like wind, electric fields, etc.

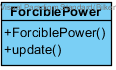


Figure26: ForciblePower Class

3.6.3 StaticGameObject Class

StaticGameObject class creates static objects for our game.

Static Game Object.png

Figure27: StaticGameObject Class

Obstacle is a static game object which does not move or change during the game.

Obstacle.png

Figure28:Obstacle Class

Description of the Interactions between Classes according to the Use Cases

General Description “MainMenu” class will be instantiated at the beginning of the game. It provides all buttons and texts related to user interface for the main menu. Also, GameEngine object will be constructed.

PlayGame: When the user clicks on “Play Game” option, an object of GameEngine will start running. During the game, this object interacts with MapManager, Input Manager, FileManager, GameObject and DrawGraphics. This shows that GameEngine is the main unit of the game.

PauseGame: This is a condition of the game which indicates a command that stops the inner game is activated.

ViewHighScores: There must be an interaction between GameEngine and FileManager that helps the user to see scoreboard. During this process, loadHighScores() and getHighScores() are called.

ViewHelp: helpLayout is responsible for instructing the user in how to play the game.