# CS 319

# Object-Oriented Software Engineering

# Spring 2019

## Design Report

## Road Blocker: Alpha

**Section 1 / Group J**

Mert Özerdem
Onur Mermer
Cemil Şişman
Burak Alaydın
Doruk Altan

## Table of Contents

**1.Introduction** **3**
- 1.1 Purpose of the System 3
- 1.2 Design Goals 3
  - 1.2.1 Criteria 3
    - 1.2.1.1 End User Criteria 3
    - 1.2.1.2 Maintenance Criteria 4
  - 1.2.2 Trade-offs 4

**2. High-level Software Architecture** **5**
- 2.1 Subsystem Decomposition 5
- 2.2 Hardware/Software Mapping 7
- 2.3 Persistent Data Management 7
- 2.4 Access Control and Security 7
- 2.5. Boundary Conditions 7

**3. Subsystem Services** **7**
- 3.1. Engine Subsystem 7
- 3.2. File Subsystem 10
- 3.3. Input Subsystem 11
- 3.4. Level Subsystem 12
- 3.5. Sound Subsystem 14
- 3.6. UI Subsystem 15

**4. Final Object Design** **18**

**5. Improvement Summary** **19**

# 1.Introduction

Road Blocker: Alpha is a puzzle game where player have to block the exit of one or more red cars to end of the table by using different shaped blocks which have police cars on them. Each level have different design with different places where red cars placed, different building blocks placed on different places, different table sizes which gets increased as players complete more levels and different blocks provided for player's use. Player have to complete 5 levels to get new 10 levels each time and player may challenge himself or other possible players by completing levels in short times which provides him a place in Highscore list.

## 1.1 Purpose of the System

Our focus as developers of this game to provide time efficiency, ease to play, proper challenge without ruining game experience and simple graphic design which not confuse the player in gaming process and give a sense of older generations arcade game design. In order to fulfill our aim we choose Java as the language which we design the game with. Java provides both efficiency for user in terms of processing ability and developer with the provided GUI libraries and Object Oriented Programming design of language.

## 1.2 Design Goals

Our main aims regarding to user experience are time efficiency which game will provide, ease to play the game, performance of the game and extending the target audience by adding new challenges. Our aims regarding to maintenance of the game is modifiability and portability of the game.

### 1.2.1 Criteria

#### 1.2.1.1 End User Criteria

**Time efficiency:** Since Road Blocker Alpha is based on a tabletop game our first concern about game is providing time efficiency to user. The game won't just cut from the time of preparing a level by hand but also provide both visual support to overview levels without entering the level which will cut the time to check every level in the set, and provide enough integrity that user can just easily start the program and get into the game without worrying any trivial setting or explanation.

**Ease to play:** Since game aims a very large scope of players from different ages range from 7 to 70, it should provide enough visual and audio support to understand how to play the game, how to progress through levels, how to beat a level or how to compete with other players without any need of literal explanations but there will be always a section for explaining how to play the game for players who need more extended explanations.

**Target Audience:** Even though original tabletop game aims for a narrow scope of players which their ages range from 7-13, as developers we try to provide enough challenge for both original audience and other players from ages over 13 to engage different type of puzzles and compete with each other to get fun from it with different players with very wide age scope. In order to keep the game experience not discouraging we provided these challenges as optional so that original tabletop game experience also remains.

**Performance:** With its graphical and audio capabilities game should provide also very fast input response to user since the challenge provided in the game is based on speed and timing. Game performance should catch up the players' gameplay speed and even encourage to play faster. Also visual and auditory feedback during the gameplay also helps for player's performance.

#### 1.2.1.2 Maintenance Criteria

**Modifiability:** In order to provide a modification support for new levels we will implement the game in a way that even any user without a coding experience can imitate the game file lines we used to create different levels with different designs to create their own levels and add it to the game. This design both provides ease to create new levels for developers and also gives an option to user to create their own levels.

**Portability:** Since we use Java as language to implement the game, as developers we will have the option to provide the game for different Operating Systems which Java allows us to do so. This is important since the times where there is only one popular Operating System are long gone and mobile gaming market extends its scope with intense speed.

### 1.2.2 Trade-offs

**Functionality vs. Development Time:** In order to provide enough functionality for GUI elements we preferred JavaFX API to implement our GUI subsystems but in order to fulfill our functionality requirements we have to spend time to learn and get associated with API to benefit from its all features. So we preferred to spend more time on learning and getting used to API to provide more functionality to game overall in terms of GUI elements rather than using Swing API which we are more associated with from earlier CS courses which would not provide enough functionality to implement our desired system.

**Modifiability vs. Development Time:** In order to provide an editor which will provide modification option to the game for both developers and users which can use that editor to add new levels to game, share it with their friends; we spend more development time to implement this editor system in order to fulfill that modifiability criteria for maintenance rather than keeping development time low and not implementing this feature.

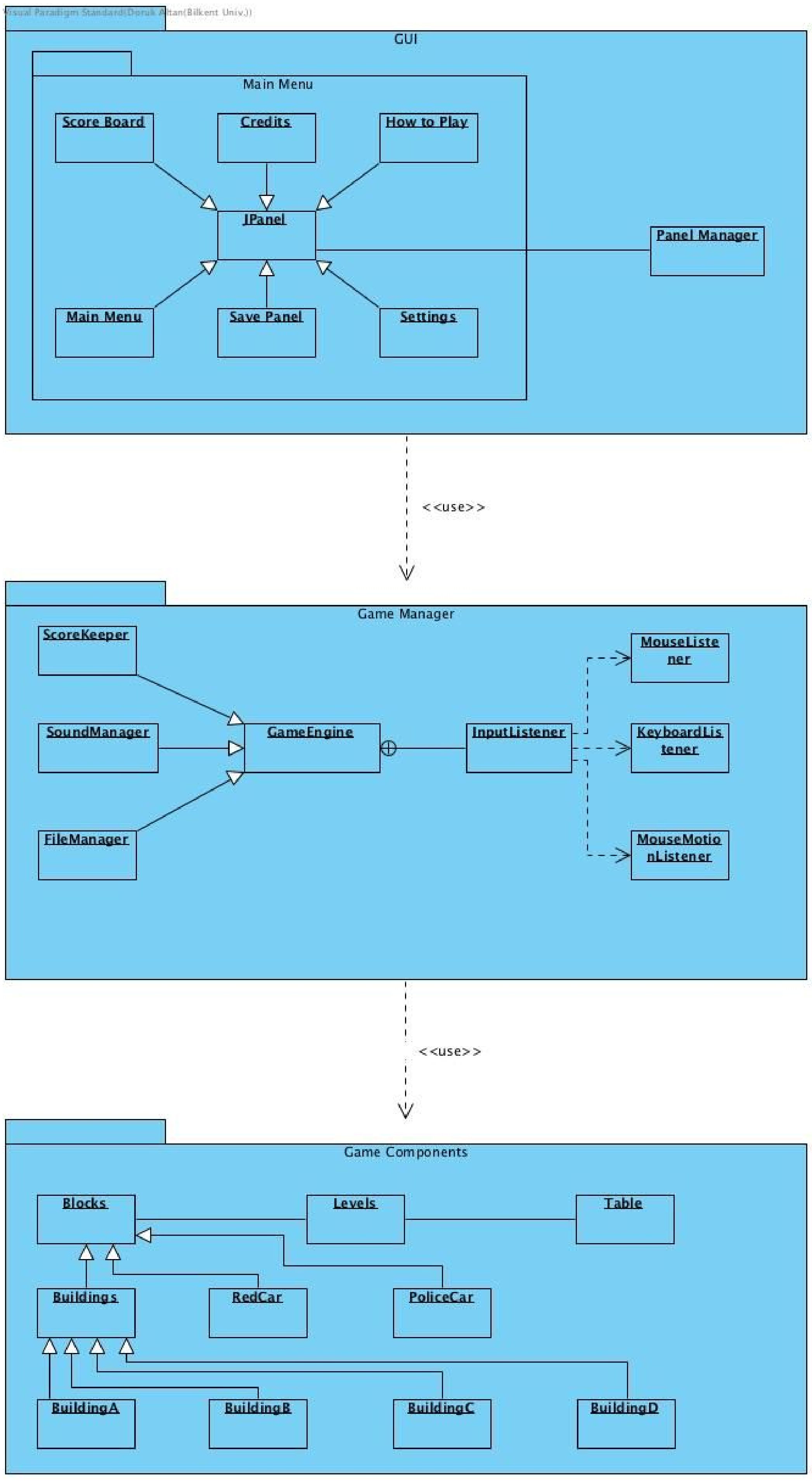
**Modifiability vs. Run-time Efficiency:** Even though run-time efficiency is an important design goal for us, in order to provide modifiability by adding an editor system to the game we preferred to use a file that would keep the level templates and saved levels together for later on which would decrease the run-time efficiency to load the game levels on start. Since it would not affect the run-time efficiency during gameplay, we prefered modifiability over run-time efficiency during start of the game which is necessary to implement editor feature.

**Run-time Efficiency vs. Memory Space:** In order to provide high run-time performance we used as much as possible memory space during gameplay to ensure that game will have minimum response time. When the level is selected by player, game will load all the data for the level into memory to provide time efficiency during gameplay rather than keeping memory cleaner and loading the data of separate entities into memory during gameplay.

# 2. High-level Software Architecture

## 2.1 Subsystem Decomposition

This section illustrates how our system is divided into subsystems. These subsystems will be covered in more detail in later sections. Only a few classes from different packages depend on one another so any changes to a package has almost no effect on the other as to  make it easier to make changes and for general maintenance. We propose two main packages for our project: GUI, and Game Components. GUI classes are responsible for interacting with the user, and Game Components classes include game engine and representations of the related game play objects. The following diagrams aim to give you a deeper insight into how these subsystems work and interact with each other.


Visual Paradigm Standard(Doruk Altan(Bilkent Univ.))
GUI
Main Menu
Score Board
Credits
How to Play
JPanel
Panel Manager
Main Menu
Save Panel
Settings
<<use>>
Game Manager
ScoreKeeper
SoundManager
GameEngine
InputListener
MouseListener
KeyboardListener
MouseMotionListener
FileManager
<<use>>
Game Components
Blocks
Levels
Table
Buildings
RedCar
PoliceCar
BuildingA
BuildingB
BuildingC
BuildingD

### 2.2 Hardware/Software Mapping

We will implement and design our game Road Block in Java language. So, Java Runtime environment will be needed as software requirement. For the hardware requirement, our game requires only a mouse and a keyboard. So any system with Java installed and keyboard – mouse plugged in should be able to play Road Block.

### 2.3 Persistent Data Management

Our game Road Block will not require database. The data will be stored in the hard drive of user's system. We are planning to extract the informations of the game's levels from a txt file. The block design will be called from this txt file during the implementation state. Also there will be another txt file for the high scores section to store the score datas.

### 2.4 Access Control and Security

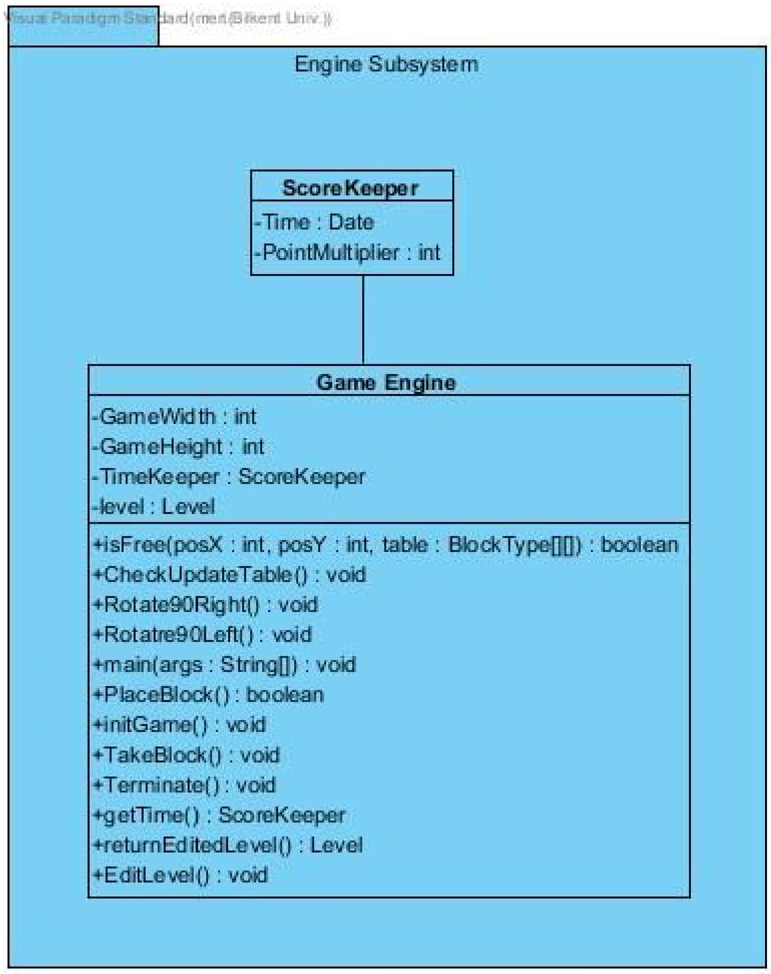
Our game Road Block does not require or use Internet, database, network connection etc. Road Block can be played on any computer. Because there are no online connection required for our game, there will not be any access control or security issue.

### 2.5. Boundary Conditions

Road Block only requires Java, there won't be necessity to install any additional softwares to play the game. The game will create its txt files necessary to save datas on the first run. There will be several options to close the game. There will be an EXIT button on the main menu which user can click and quit the game. "alt+F4" combination will automatically close the game but it may result on data loss. When the program is accidentally terminated, there will be a data loss and the game will start over the next time. Other than that, the user needs to go back to main menu to quit the game.

## 3. Subsystem Services

### 3.1. Engine Subsystem



## Score Keeper

**Fields**

---

**Private** **Time** **Date**
ScoreKeeper object takes current time to compere after level is finished.

---

**Private** **PointMultiplier** **int**
Point multiplier can be changed via game engines instructions

---

## Game Engine

**Fields**

---

**Private** **GameWidth** **int**
Defines the game’s width on the screen

---

**Private** **GameHeight** **int**
Defines the game’s height on the screen

---

**Private TimeKeeper ScoreKeeper**
Keeps a ScoreKeeper object to calculate level scores of the players

---

**Methods**

---

**Public isFree() boolean**
A method that checks whether or not red car of the game is free to leave its place and be able to escape

---

**Public CheckUpdateTable() void**
Checks if some new update required on table like; placing a block on table or removing a block from table. This method checks and updates the board that are generated inside main function

---

**Public Rotate90Right() void**
Rotates selected piece 90 degrees right

---

**Public Rotate90Left() void**
Rotates selected piece 90 degrees left

---

**Public main(args: String[]) void**
The main function of the game. CheckUpdateTable, PlaceBlock, initGame, TakeBlock, getTime functions are used in this method to run the game. The game in a loop that exits until player quits the game

---

**Public PlaceBlock() boolean**
Looks a the table and returns a true-false value according to availability of the spot on the table

---

**Public initGame() void**
Initialize the game and levels according to given GameWidth, GameHeight, and level objects' specifications

---

**Public TakeBlock() void**
If objects location and mouse's locations are equal and the block is a police car type block, the method moves the object according to given input from the mouse

---

**Public Terminate() void**
Terminates the level and returns to the main screen

---

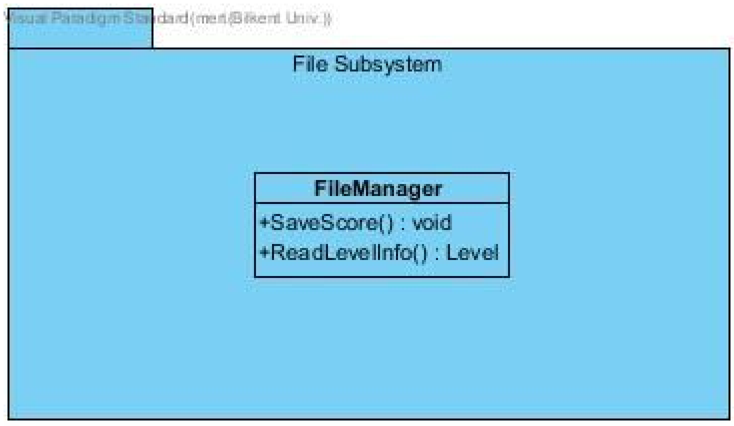
**Public getTime() ScoreKeeper**
Creates and returns ScoreKeeper object to compare level completion time

---

**Public ReturnEditedLevel Level**
Returns the edited level by player to FileManager for future preservation of the level

**Public** **EditLevel** **void**
Keeps the information about the level while player dynamically editing level.

## 3.2. File Subsystem



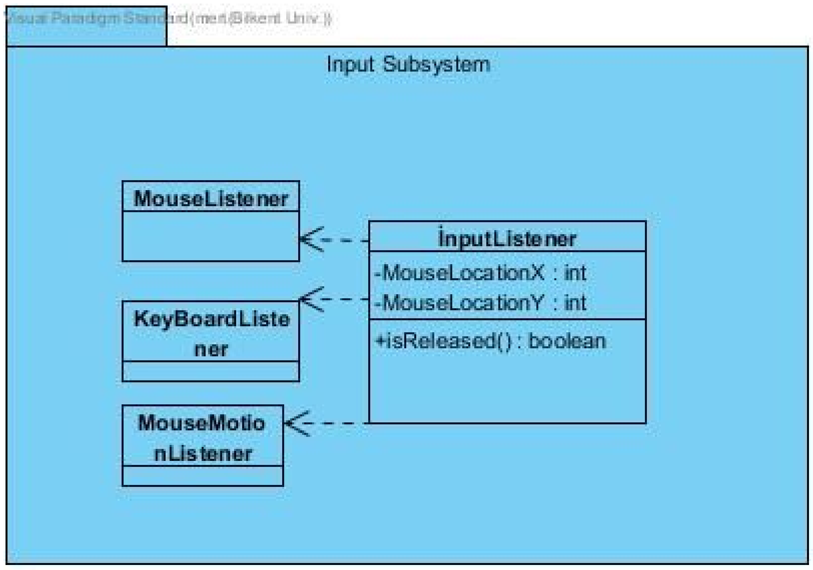
**FileManager**

**Methods**

**Public** **SaveScore()** **void**
Saves scores to a file to display it for later

**Public** **ReadLevelInfo()** **level**
Reads a file which level information are kept then method turns those information into a level object for further use

## 3.3. Input Subsystem



**InputListener**

**Fields**

---

**Private** **MouseLocationX** **int**
Returns the location of the mouse on the x coordinates

---

**Private** **MouseLocationY** **int**
Returns the location of the mouse on the y coordinates

---

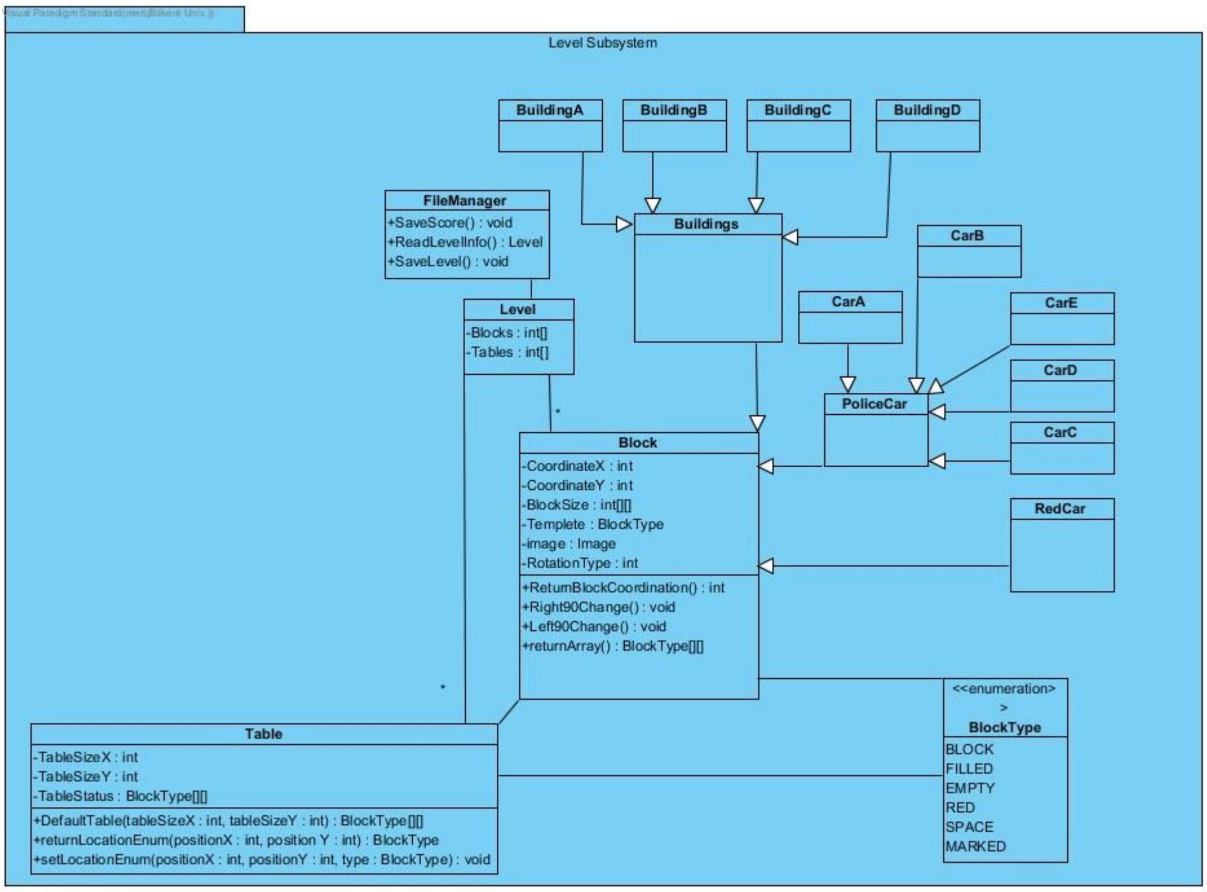
**Methods**

---

**Public** **isReleased()** **boolean**
Returns a boolean value to check whether or not clicked mouse input is still active.

---

## 3.4. Level Subsystem



**Blocks**

**Fields**

---

**Private CoordinateX int**

Keeps X coordinate of a block.

---

**Private CoordinateY int**

Keeps Y coordinate of a block.

---

**Private BlockSize int[][]**

Stores block size.

---

**Private Template BlokcType**

BlockType object filles the BlockSize array to give it an space like L shaped police car object.

---

**Private Image Image**

Image of the object

---

**Private RotationType int**
There are 4 different types of rotation which refers to 90 degree turned object for example 0 refers to initial position while 3 refers to 270 degree right turned object

**Methods**

---

**Public ReturnBlockCoordination int[]**
Returns the coordination of a block.

---

**Public Right90Change void**
Returns the object to the clockwise 90 degrees while adding RotationType by one.

---

**Public Left90Change void**
Returns the object to the counterclockwise 90 degrees while reducing RotationType by one.

---

**Public ReturnArray BlockType[][]**
Returns the shape of the object which is kept as BlockType array.

**Table**

**Fields**

---

**Private TableSizeX int**
Column number of the game grid.

---

**Private TableSizeY int**
Row number of the game grid.

---

**Private TableStatus int[][]**
Double array to keep starting situation of the game table (which blocks are empty, which blocks are not).

**Methods**

---

**Public DefaultTable(TableSizeX : int, TableSizeY : int) BlockType[][]**
Inilizes the default table but also returns the table’s array.

---

**Public ReturnLocationEnum(PositionX : int, PositionY : int)**
**BlockType**
Returns the given location’s enumeration(BlockType) value to system.

---

**Public SetLocationEnum(PositionX : int, PositionY : int, type : BlockType) void**
Sets the given locations enumeration(BlockType) value to indicated type

**Level**

**Fields**

| | | |
|---|---|---|
| **Private** | **Blokcs** | **Blocks[]** |

Stores required blocks for a particular level.

| | | |
|---|---|---|
| **Private** | **Tables** | **Table[]** |

Stores the table information for a particular level.

**FileManager**

**Methods**

| | | |
|---|---|---|
| **Public** | **SaveScore** | **void** |

Saves score to a output file to future display

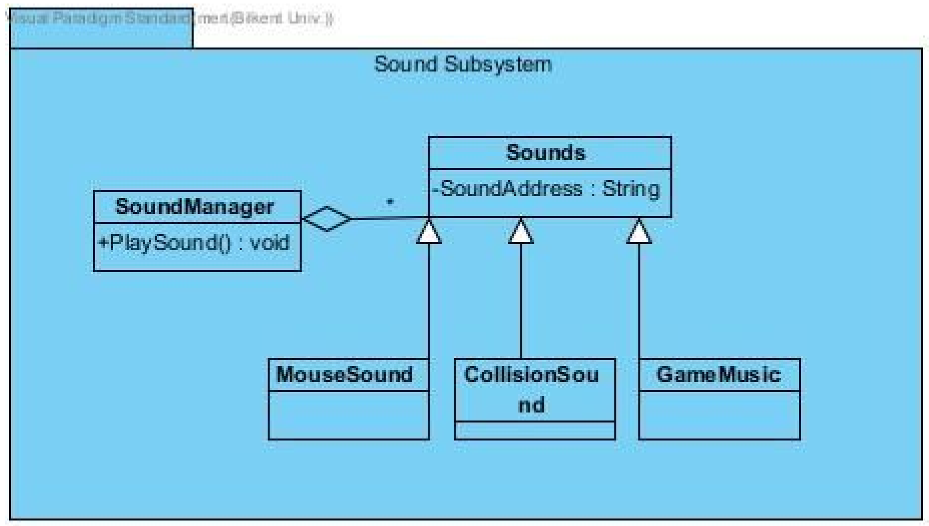
| | | |
|---|---|---|
| **Public** | **ReadLevelInfo** | **Level** |

Returns the level information from the output file that all levels' information stands

| | | |
|---|---|---|
| **Public** | **SaveLevel** | **void** |

Adds the edited level to the at the end of the level information file

## 3.5. Sound Subsystem



**Sounds**

**Fields**

---

**Private** **SoundAddress** **string**
Stores the file address where the sound effects will be taken.

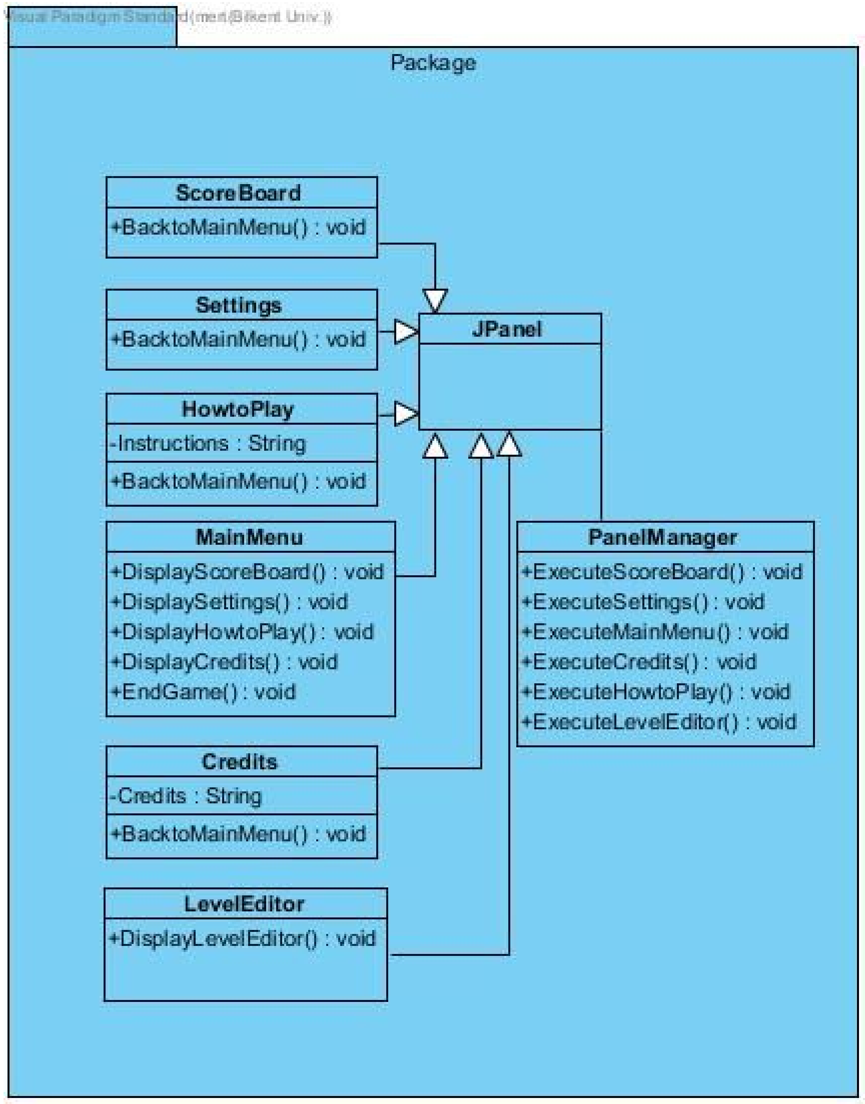
**SoundManager**

**Methods**

---

**Public** **PlaySound** **void**
Plays different sounds for different interactions.

## 3.6. UI Subsystem


Visual Paradigm Standard(ment(Bilkent Univ.))
Package
ScoreBoard
+BacktoMainMenu() : void
Settings
+BacktoMainMenu() : void
HowtoPlay
-Instructions : String
+BacktoMainMenu() : void
JPanel
MainMenu
+DisplayScoreBoard() : void
+DisplaySettings() : void
+DisplayHowtoPlay() : void
+DisplayCredits() : void
+EndGame() : void
PanelManager
+ExecuteScoreBoard() : void
+ExecuteSettings() : void
+ExecuteMainMenu() : void
+ExecuteCredits() : void
+ExecuteHowtoPlay() : void
+ExecuteLevelEditor() : void
Credits
-Credits : String
+BacktoMainMenu() : void
LevelEditor
+DisplayLevelEditor() : void

**ScoreBoard**

**Methods**

| | | |
|---|---|---|
| **Public** | **BackToMainMenu()** | **void** |

Leaves scoreboard screen, opens main menu.

**Settings**

**Methods**

| | | |
|---|---|---|
| **Public** | **BackToMainMenu()** | **void** |

Leaves settings screen, opens main menu.

**HowtoPlay**

**Fields**

| | | |
|---|---|---|
| **Private** | **Instructions** | **string** |

Keeps the information of game logic and gameplay to explain how to play the game.

**Methods**

| | | |
|---|---|---|
| **Public** | **BackToMainMenu()** | **void** |

Leaves how to play screen, opens main menu.

**Credits**

**Fields**

| | | |
|---|---|---|
| **Private** | **Credits** | **string** |

Keep the information of the game’s creators.

**Methods**

| | | |
|---|---|---|
| **Public** | **BackToMainMenu()** | **void** |

Leaves credits screen, opens main menu.

**MainMenu**

**Methods**

| | | |
|---|---|---|
| **Public** | **DisplayScoreBoard()** | **void** |

Displays scoreboard screen.

**Methods**

**Public** **DisplayHowtoPlay()** **void**
Displays “How to play” screen.

---

**Public** **DisplayCredits()** **void**
Displays credits screen.

---

**Public** **EndGame()** **void**
Exits the game.

**LevelEditor**

**Methods**

---

**Public** **DisplayLevelEditor** **void**
Displays the level editor UI.

**PanelManager**

**Methods**

---

**Public** **ExecuteScoreBoard** **void**
Initializes ScoreBoard according to given system Specifications.

---

**Public** **ExecuteSettings** **void**
Initializes settings according to given system Specifications.

---

**Public** **ExecuteMainMenu** **void**
Initializes main menu according to given system Specifications.

---

**Public** **ExecuteCredits** **void**
Initializes credits according to given system Specifications.

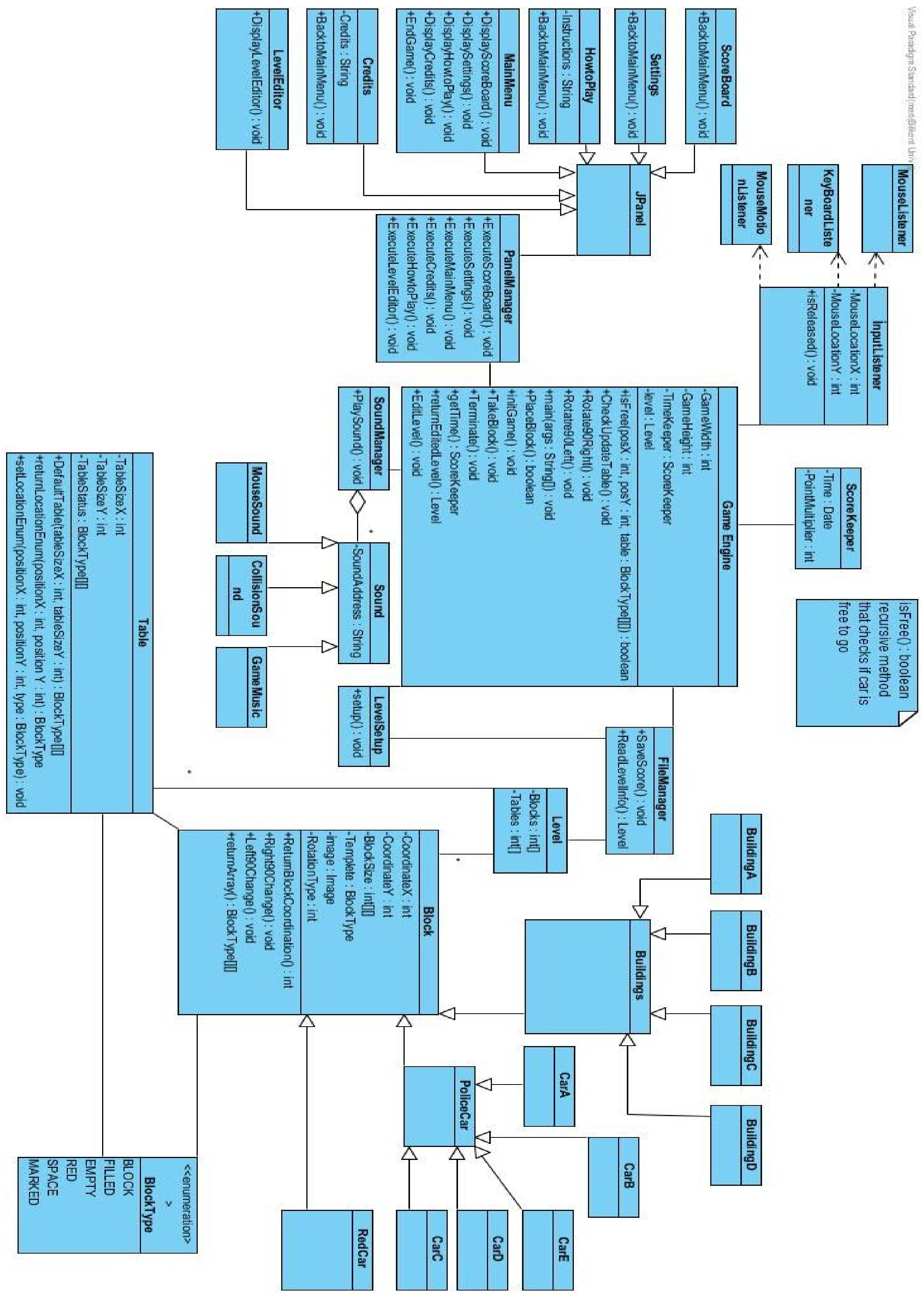
---

**Public** **ExecuteHowtoPlay** **void**
Initializes how to play according to given system Specifications.

---

**Public** **ExecuteLevelEditor** **void**
Initializes level editor according to given system Specifications.

## 4. Final Object Design



# 5. Improvement Summary

For the second iteration of Design Report:

1. New trade offs are added, old trade-off is edited.
2. Subsystem decomposition edited in order to show Boundary, Control and Entities in different packages which will interact each other as shown.
3. New functions for Engine, Level and UI subsystems added, few of the old functions edited.
4. Full object diagram is added to end of the report.