Bilkent University
Department of Computer Engineering

OBJECT-ORIENTED SOFTWARE ENGINEERING PROJECT DESIGN REPORT

CS 319 Project: Dribble & Score

25/03/2017
Group name:
2C

Group Members:

Batıkan HAYTA 21301382
Berke SOYSAL 21400908
Boran YILDIRIM 21401947
Sencer Umut BALKAN 21401911

Instructor:

Prof. Dr. Uğur DOĞRUSÖZ

1.Introduction

1.1 Purpose of System

Dribble & Score is a PC game which is designed to keep the players entertained as much as possible while playing the game. The user interface will be developed in such a way that even a first-time computer user can easily understand it. This game adapts the user to the game in a very simple way on the first few levels, and when it comes to more difficult levels, there will be nothing that the user does not understand about the game.

1.2 Design Goals

The main goal for a computer game is to entertain the player. To achieve this task, it is needed to focus on little details which are not directly noticeable at a first glance.

This section details the design goals of the system such as end user criteria, maintenance criteria, performance criteria and the trade-offs that come with our chosen way of implementation.

1.2.1 End User Criteria

Target User Base: When we observe other successful games available today, we see that they have one common attribute. It is the wide range of the user base. As an example, Candy Crush is very successful because a kid and an old person can both enjoy it equally. Our aim is to have a wide age range, to love this game.

Ease of learning: When starting the game for the first time, the user will not know how the controls work. So, it is important to provide the user with a smooth learning curve. First couple levels will be designed in such a way that it teach the controls and the mechanics of the game. After these simple levels, the player will be pushed towards greater challenges to increase the fun factor.

Ease of use: Most computer games use common controls these days to increase the ease of use. The navigation menu will be easy for anyone, as well as the gameplay. Mouse will be used for menu, and the arrow buttons for the gameplay.

1.2.2 Maintenance Criteria

Extensibility: In the lifecycle of a software program, it is important to maintain the ability to add or remove features. As object oriented software engineering principles imply, the first goal in development is to create a highly maintainable software.

Portability: In today's software and hardware world, everything is changing rapidly. However, it has a solution, cross-platform application/game development which runs platform independent. Java is a platform independent programming language such that no matter what operating system or processor architecture is system used, if the system has JRE then there will be no problem. "Write once, run everywhere"

1.2.3 Performance Criteria

Smooth Graphics: Game will contain open source 8-bit pictures. Main character, bonuses, goalkeepers and the whole level environment is going to be visualised in a retrospective way. Therefore, at this point, we will not experience a noticeable loss of speed in order to process the graphics and move with the game engine.

Input Response Time: Inputs from the keyboard while playing the game needs to be very precise because the players will be very frustrated if the character crashes due to input lag.

1.2.4 Trade-offs

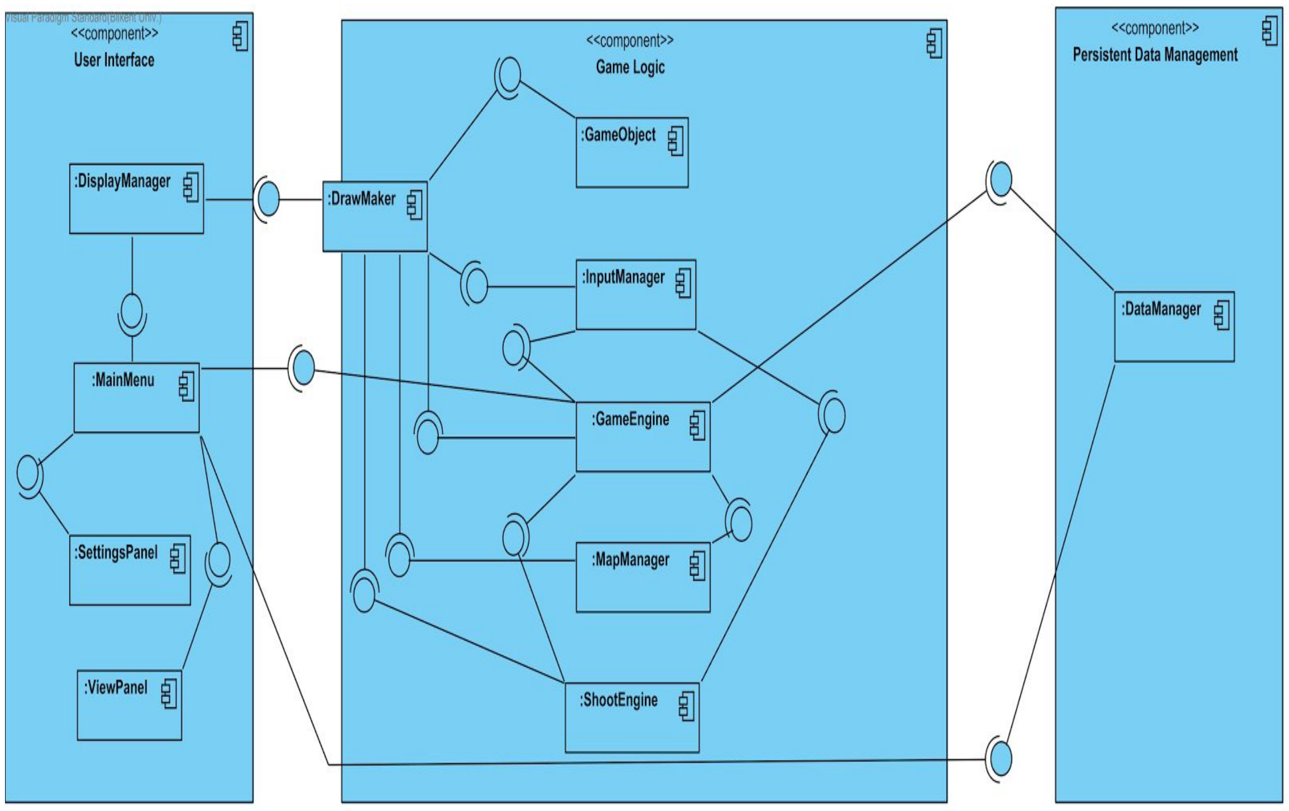
Portability - Performance & Memory: In this project, we will be using the Java Programming Language, which is known for it's ability to produce binaries which can run on any processor architecture. However, this results in all the applications being run in an interpreter called the Java Virtual Machine. When compared to native languages like C or C++, programs written with Java use more resources while doing less. On the flip side, coding our game with Java will decrease the development time due to the broad range of libraries available and the game will be running on any Java compatible desktop device.

Simplification - User Base: In our end user criteria, we detailed the target user base to be very broad. When a game targets a broad range, it requires over simplification of some game mechanics. If we could have used very detailed and intensive graphics which would have required a powerful desktop computer, which many users don't have. So, we decided to keep the game simple and have a broader user base.

2. Software Architecture

2.1. Subsystem Decomposition

In this project, we chose the three-tier architecture to design our system because three-tier architecture system is most suitable design for our system structure. In below, there is a three-tier component diagram of our design.



In user interface layer, there is MainMenu which users interact with the game. MainMenu interact with “DisplayManager” to display necessary things into MainMenu. Also, it interacts with DataManager to get settings of the games.

In application layer we have 4 components which are “DisplayManager”, “GameEngine”, “MapManager”, and “ShootEngine”. “DisplayManager” responsible to give display information to the MainMenu and it sends those information according to the “InputManager”. For “GameEngine” part, this part responsible to create map by the help of the “MapManager” and “DataManager”. “Map Manager” creates random map according to the specific level and it send map objects to the “GameEngine” to create them. When the dribble part end and shoot part take place, “ShootEngine” takes information from “Input Manager” and uses those information to dedicates where the balls should go. “Input Manager” which is listener for our projects. It reads the data which users entered in the keyboard and send them to the responsible components.

In persistent data management, We have “DataManager” which reads and write information to the users' hard disk drive. “DataManager” is one of the important things to our project. It keeps the important data such as score, default settings, levels of the game etc. and send those information to responsible components.

2.2. Hardware/software mapping

Our game will developed by java so it will require a Java Runtime Environment to be executed. For the hardware requirements, keyboard is necessity to play this game to interact with the game.

About the graphics of the game, we decided to use 8-bit images so that required space of the game will be decrease. As a result of this, low system computers can handle. We uses keyboard the takes input from the users in real-time.

2.3. Persistent Data Management

Since our game does not need a complex database system, game data will be stored in the user's hard disk drive. “Data Manager” stores necessary information to the users hard drive disk. Also, there will be images and sounds which are used for games and those things will be stored in user's hard drive disk as well.

2.4. Access Control and Security

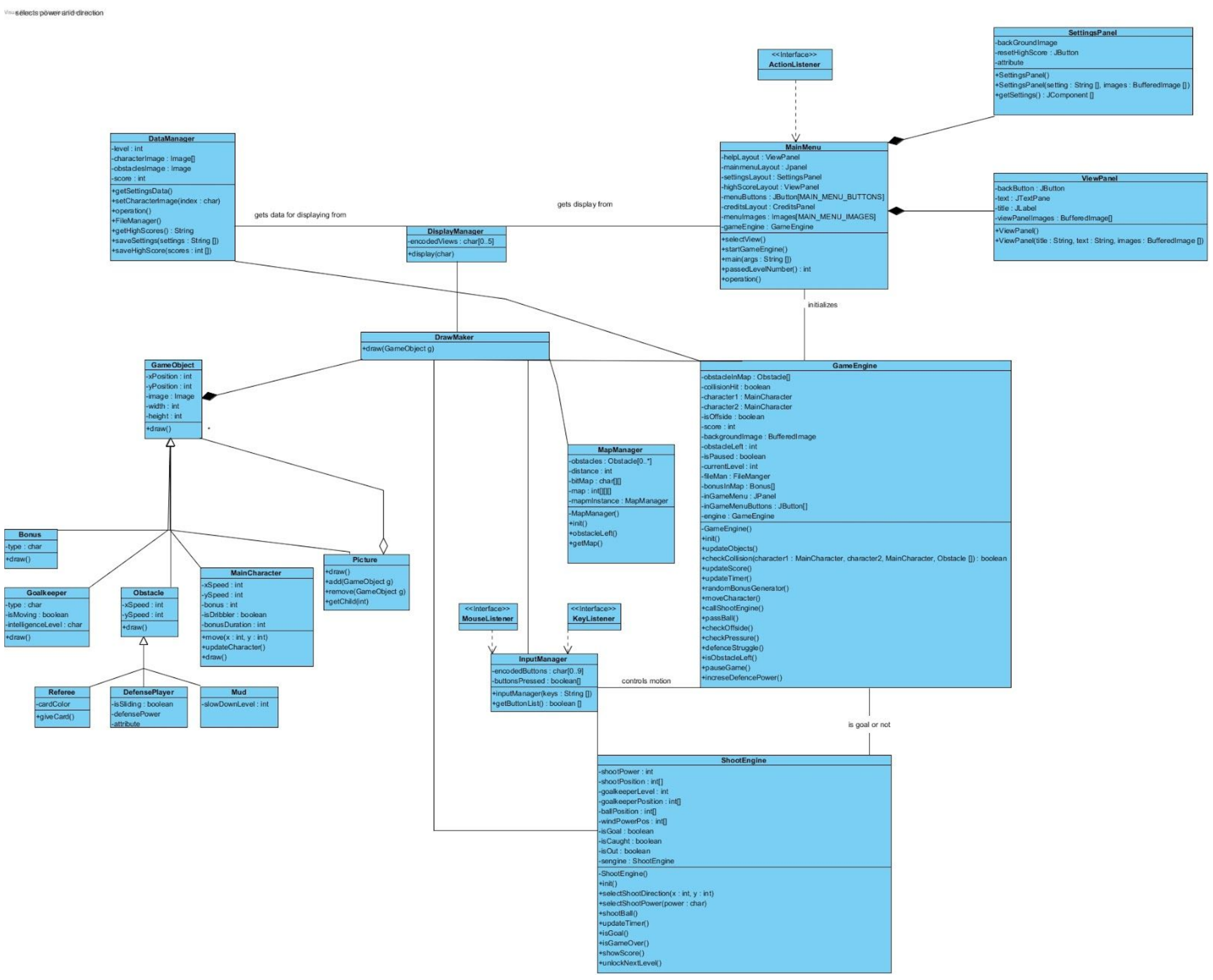
There will be no user authentication system for our games and games does not require any internet connection to play. So that there will be no critical information security leak. Some of variables will declare as a private and constant so that outsiders could not change it. “GameEngine” has a access to important files and datas such as map difficulty, level system etc. so that outsiders could not change.

2.5. Boundary Conditions

If there is a corrupted data in the game, program will give an error. While playing a game, if player's life are gone, game will display score of this game and gives two option. One is “Go Main Menu” and other one is “Retry”. Depending on the player respond it will either display main menu or starts the level again. Game will consist of 10 different level and only way to unlock other level is complete the level which comes before. After unlocking all the levels, player can choose any level he/she desires.

3. Subsystem Services

In this section, we will explain the detailed information about the interfaces of our subsystem.

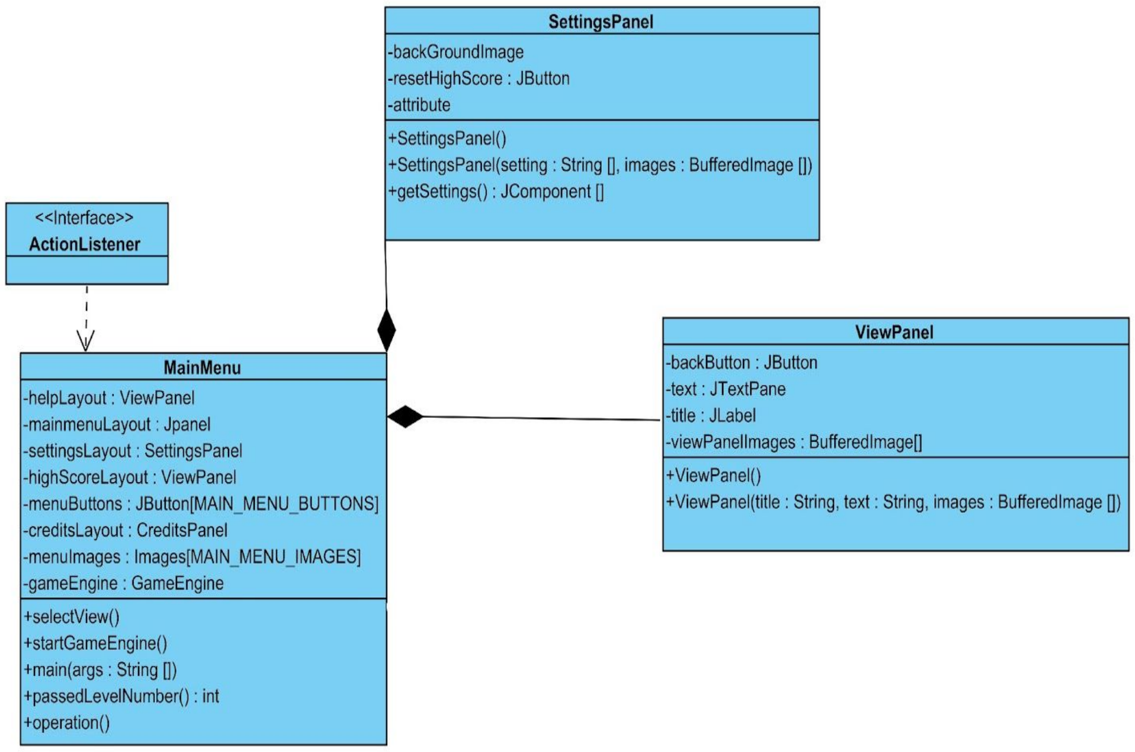


Façade Pattern

This design pattern aims to provide a unified and high level interface which many objects in the program could use. As an example from our project, “DrawMaker” is the Façade class of the project which interacts the display component with logic layer component. This enables developers to reflect changes in the project in a more dynamic way. Therefore, the Façade design pattern further improves the maintainability and reusability of our code.

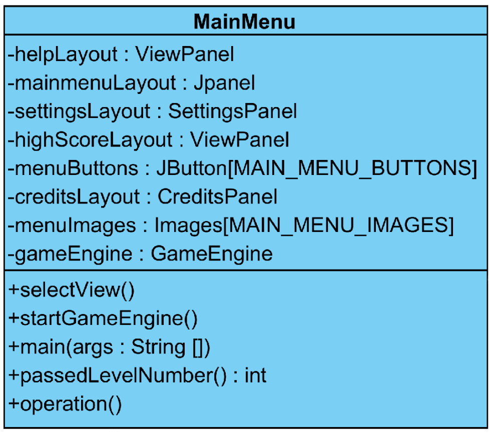
3.1 User Interface Subsystem

This subsystem responsible to provide an interface between user and the system so that users can interact with the system easily.


SettingsPanel
-backGroundImage
-resetHighScore : JButton
-attribute
+SettingsPanel()
+SettingsPanel(setting : String [], images : BufferedImage [])
+getSettings() : JComponent []
<<Interface>>
ActionListener
MainMenu
-helpLayout : ViewPanel
-mainmenuLayout : Jpanel
-settingsLayout : SettingsPanel
-highScoreLayout : ViewPanel
-menuButtons : JButton[MAIN_MENU_BUTTONS]
-creditsLayout : CreditsPanel
-menuImages : Images[MAIN_MENU_IMAGES]
-gameEngine : GameEngine
+selectView()
+startGameEngine()
+main(args : String [])
+passedLevelNumber() : int
+operation()
ViewPanel
-backButton : JButton
-text : JTextPane
-title : JLabel
-viewPanelImages : BufferedImage[]
+ViewPanel()
+ViewPanel(title : String, text : String, images : BufferedImage [])

“ViewPanel” and “SettingsPanel” is responsible to display other submenus such as “Settings”, “Credits”, “Scores”, “Levels” etc. “Main Menu” controls these clases so that when user wants to reach those submenus.

3.1.1 MainMenu Class


MainMenu
-helpLayout : ViewPanel
-mainmenuLayout : Jpanel
-settingsLayout : SettingsPanel
-highScoreLayout : ViewPanel
-menuButtons : JButton[MAIN_MENU_BUTTONS]
-creditsLayout : CreditsPanel
-menuImages : Images[MAIN_MENU_IMAGES]
-gameEngine : GameEngine
+selectView()
+startGameEngine()
+main(args : String [])
+passedLevelNumber() : int
+operation()

“Main Menu” class is the first class when game is first executed and its first interface that users can see. In this menu, users can see six buttons which are “Levels”, “Settings”, “Scores”, “Help”, “Credits”, and “Exit”. “passedLevelNumber” methods has levels that users unlocked succesfully through completing them. “View Panel” and “Setting Panel” helps Main menu class to view those interfaces to the users.

<u>Attributes:</u>

ViewPanel helpLayout: Displays the help screen.

JPanel mainmenuLayout: Displays the main menu screen.

SettingsPanel settingsLayout: Displays the settings screen.

ViewPanel highscoreLayout: Displays the highscore of the level.

JButtons menuButtons: Buttons in main menu which helps users to select different options.

CreditsPanel creditsLayout: Displays the names of the developers.

Images menuImages: Images that are used to visualize main menu.

GameEngine gameEngine: GameEngine objects that helps to play game.

Functions:

selectView(): controls the option which is selected.

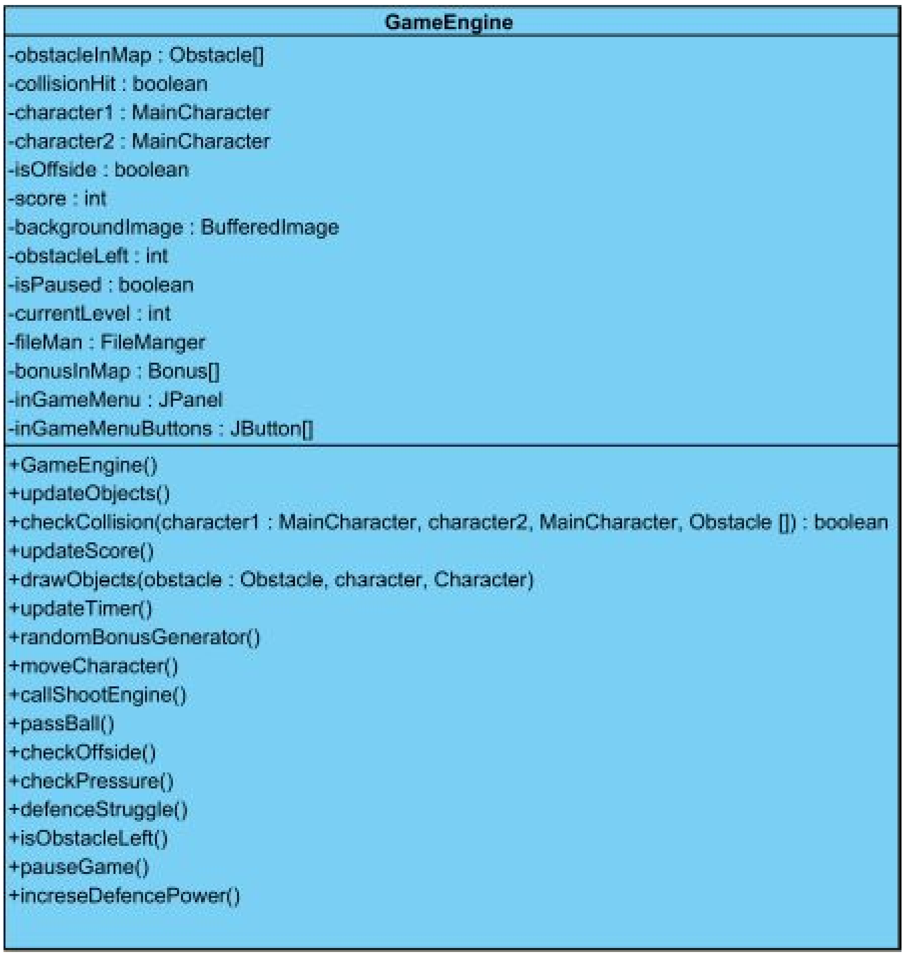
startGameEngine(): Starts the GameEngine when users press start game button.

passedLevelNumber(): Method that shows the level which are passed(unlocked)

3.2 Application Subsystem

Application layer is responsible to handle game mechanics such as collision check, create map and display. Our façade class “GameEngine” is responsible to create random map and other methods that helps gameplay mechanics.

3.2.1 GameEngine Class



```
GameEngine
-obstacleInMap : Obstacle[]
-collisionHit : boolean
-character1 : MainCharacter
-character2 : MainCharacter
-isOffside : boolean
-score : int
-backgroundImage : BufferedImage
-obstacleLeft : int
-isPaused : boolean
-currentLevel : int
-fileMan : FileManger
-bonusInMap : Bonus[]
-inGameMenu : JPanel
-inGameMenuButtons : JButton[]
+GameEngine()
+updateObjects()
+checkCollision(character1 : MainCharacter, character2, MainCharacter, Obstacle []) : boolean
+updateScore()
+drawObjects(obstacle : Obstacle, character, Character)
+updateTimer()
+randomBonusGenerator()
+moveCharacter()
+callShootEngine()
+passBall()
+checkOffside()
+checkPressure()
+defenceStruggle()
+isObstacleLeft()
+pauseGame()
+increseDefencePower()
```

After player starts the level, “GameEngine” takes place to create map according to the selected level and obstacles. In map creation phase, it takes information from “MapManager” to create“ the map and DataManager” gives the settings and levels information to the “GameEngine” class. After creating objects of the game and map, updateObjects() methods helps to adjust speed and positions of the objects. CheckCollision methods checks whether mainCharacter1 or if it is multiplayer game, mainCharacter2, collide with obstacles or not. Finally, isObstacleLeft() check whether obstacle left or not.

If it return true, dribble parts will finish and shoots part takes place and shoots part, “ShootEngine” is responsible.

<u>**Attributes:**</u>

Obstacle[] obstacleInMap: x,y Position of a obstacle which is currently on the display.

Collision booleanHit: If one of the players are collides with one of the obstacles, becomes true, otherways keep being false.

Character MainCharacter1: The gameobject of player1.

Character MainCharacter2: The gameobject of player2.

boolean isOffside: While one of the players holding the ball, if he decides to pass to other player and the defenders are vertically more closer to player than the other player, returns true and game stops.

int Score: Keeps increasing while game continues, if one of the players get bonuses an extra incrementation of the score is occured

BufferedImage backgroundImage: The image on the background while displaying, the game, little details can be observed by the player on the background while the players are runnning to make the display more dynamic.

int obstacleLeft: Total obstalce left in the map, that hasn’t been displayed on the game yet.

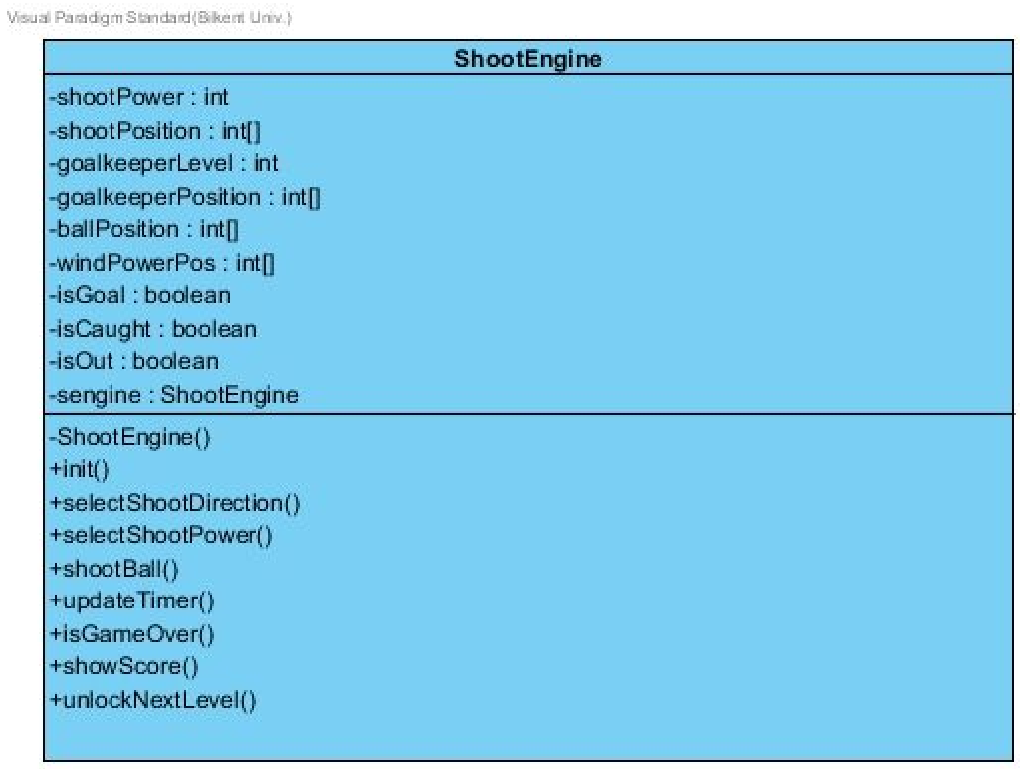
Boolean isPaused: If the player presses “Pause Button” isPaused becomes true and Mainmenu is opened.

Int currentLevel: The current level is sent to mapmanager to create map, that fits the difficulty of level.

FileManager fileMan: FileManager that holds game data like images, settings etc.

Bonus[] BonusInMap: If an bonus is displayed on the screen, this keeps the coordination of the bonus in the map.

3.2.2 ShootEngine Class



“ShootEngine” class determines the users input from “InputManager” class. According the this information, it adjust ball speed, direction, positions. Also, It adjust goalkeeper level and position according to the respective levels. If isGoal() method return true, it display score of the level and unlockNextLevel() methods unlocked the next level.

Attributes:

int shootPower: Determines the power of the shoot depending on the user input.

int shootPosition: Determines the position of the shoot depending on the user input.

int goalkeeperLevel: The difficulty level of the goalkeeper.

int goalkeeperPosition: The position of the goalkeeper on the display.

int ballPosition: The position of the ball on the display.

int windPowerPos: The power of the wind and the direction of the wind.

boolean isGoal: If the player manages to score, isGoal is true and the next level is unlocked. Otherwise, returns false and next level keeps being locked.

boolean isCaught: If the ball is saved by the goalkeeper, saved is true, false otherwise.

boolean isOut: If the ball is gone to the out of the game field, isOut is true, false otherwise.

Functions:

selectShootDirection(): Takes user inputs for the direction of the ball.

selectShootPower(): Takes user inputs for the direction of the ball.

shootBall(): Runs when the direction and power is selected. The animaton of the ball starts by this function.

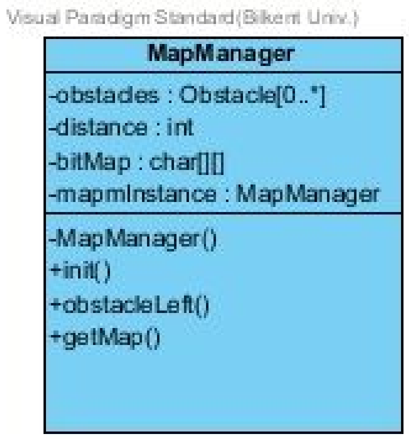
updateTimer(): Updates time for redrawing the objects.

isGameOver(): Runs when the animation of the shooting process is completed. Checks the isgoal, isout,issaved variables to decide to unlock next level or not.

showScore(): Displays final score on the screen, based on the first and second episodes of the level.

unlockNextLevel(): Unlocks the next level or not based on the output of the **isGameOver()** function.

3.2.3 Map Manager


Visual Paradigm Standard(Bilkent Univ.)
MapManager
-obstacles : Obstacle[0..*]
-distance : int
-bitMap : char[][]
-mapmInstance : MapManager
-MapManager()
+init()
+obstacleLeft()
+getMap()

“MapManager” is responsible for the creating the map and sending this map to the “GameEngine” to creat them. In map, there are random obstacles and random length of the map.

__**Attributes:**__

ArrayList <Obstacle> obstacles: Stores the obstacles of a level.

int distance: The length of the map of the level.

char[][] bitMap: Store the bits of the coordinates of map. If that bit contains enemy, it will 1, if does not contain, it will 0.

MapManager mapmInstance: Map manager for singleton pattern, uniqueue for all levels.

Functions:

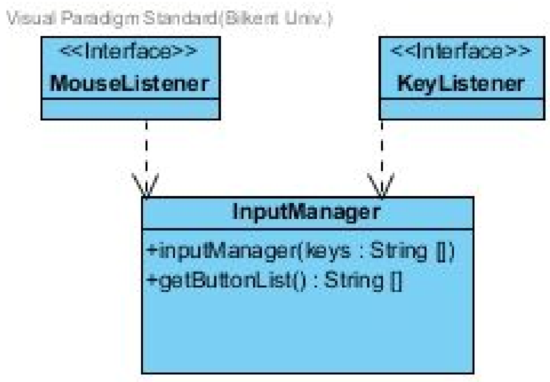
MapManager(): Private constructor for singleton pattern.

init(): Initialize the object as singleton pattern. Check whether the mapmInstance is null or not before the creation of new object.

obstacleLeft(): Returns the number of obstacles left in the map.

getMap(): Returns the mapmInstance.

3.2.4 InputManager Class


Visual Paradigm Standard(Bilkent Univ.)
<<Interface>>
MouseListener
<<Interface>>
KeyListener
InputManager
+inputManager(keys : String [])
+getButtonList() : String []

“InputManager” class is listener class of the system. It allows keyboard and mouse control to the user. “getButtonList()” method is used by “GameEngine” class to find pressed buttons by the users and changes the MainCharacters accordingly.

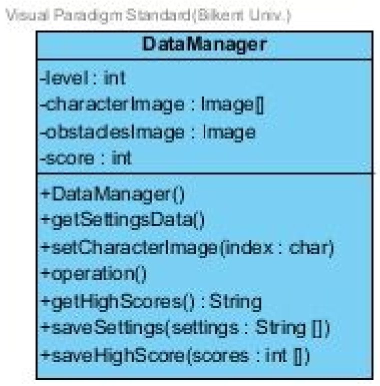
Functions:

getButtonList(): Return the controller key list.

3.3 Data Management Subsystem

This subsystem write information to the users hard disk drivers. Also, it has information to help gameplay.

3.3.1 DataManager Class


Visual Paradigm Standard(Bilkent Univ.)
DataManager
-level : int
-characterImage : Image[]
-obstaclesImage : Image
-score : int
+DataManager()
+getSettingsData()
+setCharacterImage(index : char)
+operation()
+getHighScores() : String
+saveSettings(settings : String [])
+saveHighScore(scores : int [])

“DataManager” class has level, character and obstacles images and score information. Also keeps the default settings of the system. When users changes any settings it keeps these changes and send these changes to “GameEngine” before user stars to play.

Attributes:

int level: Number of levels that are unlocked.

Image[] characterImage: Images of characters.

Image obstacleImage: Images of obstacles.

int score: Scores of corresponding levels.

Functions:

getSettingsData(): getting the current settings of the game.

setCharacterImages(): It helps users to sets character images which are avaiable.

getHighScore(): getting the high scores of levels.

saveSettings(): Takes the current settings and change it to users preference options

saveHighScore(): When there is a new score which is bigger than other scores it changes current high score the this score.

4. Low-level Design

4.1. Object Design trade-offs

4.1.1 Space-Graphic/ Time-Graphic Trade-offs

In our game, we used 8-bit images to decrease size of the game and also increase the execution speed of the game so that users can start the game very fast. We sacrifice good and modern graphical objects to represent our objects but we able to make a game that required low space. Also, it helps others users who do not have a decent computer.

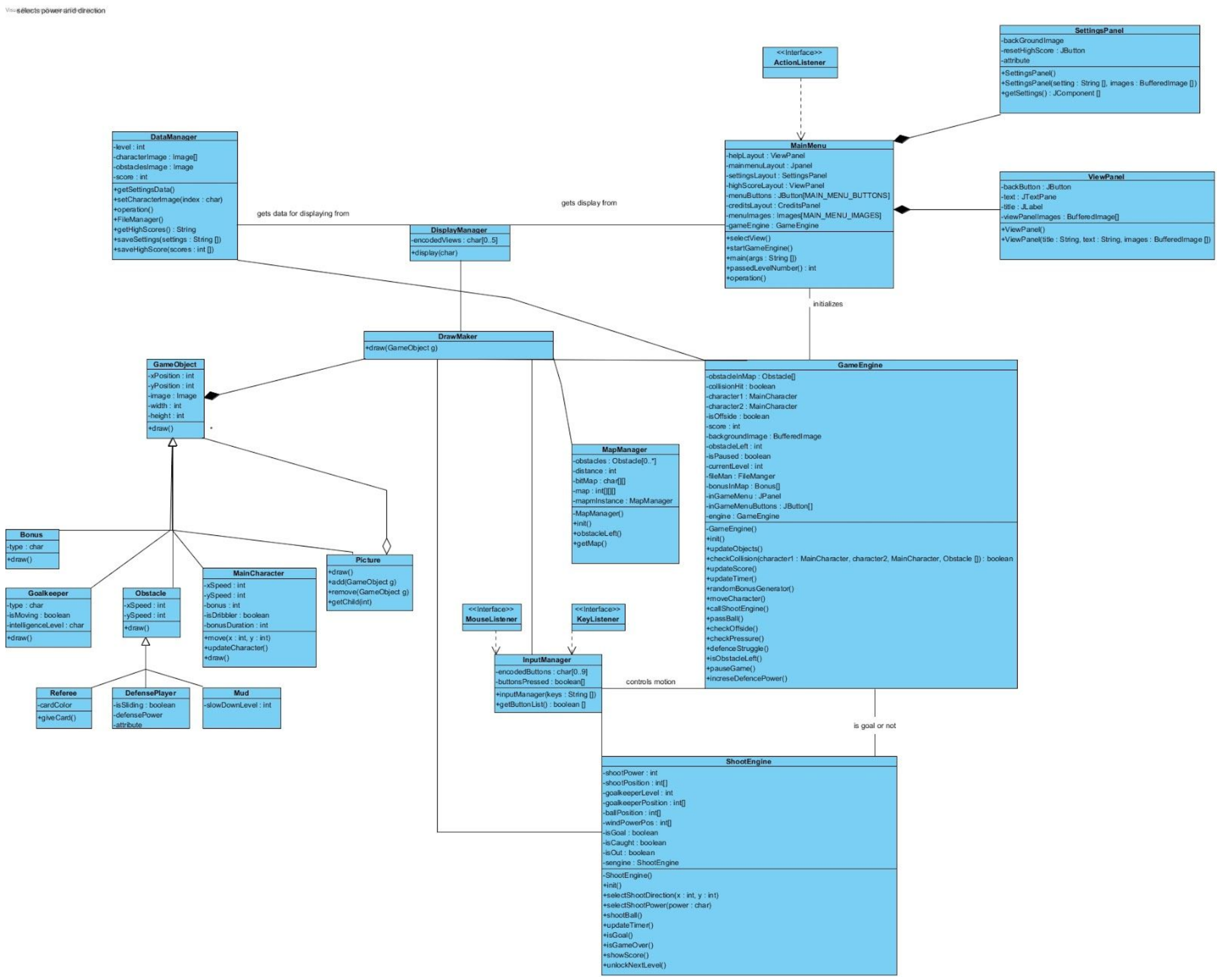
4.1.2 Maintainability

Since we are using façade design pattern, it helps to keep the maintenance of the classes easily. If we decide to change some classes or add some component to it, we just need to adjust façade class so that it will not take much time.

4.1.2 Rapid Development vs Functionality

Since we have limited time to develop our project, it might cause some mistakes during the coding. Also, finding and debugging the game might be challenging due to limited time. However, our aim is to use time as efficient as possible so that we will not compromise the functionality of the game.

4.2. Final Object Design



4.2.1 Façade Pattern

Between subsystems, a communication needs to be satisfied in order to have data and communication between the tiers. Façade pattern is used to reduce the complexity of grouping up the classes and subsystems that are connected. In this project, Façade pattern is used in DrawMaker class. Since DrawMaker is provided the communication between display and logic components, it has been given properties to handle the classes down it.

4.2.2 Singleton Pattern

Singleton design pattern is used for some of the important classes need to have only one instances at the same time. For example, GameEngine uses this pattern. It should be one “GameEngine” because it keeps the data which are related to gameplay. More than one creation of the “GameEngine” can cause some bugs while playing and creating game. Also “MapManager” and “ShootEngine” uses this pattern too. Creating multiple “MapManager” can cause creating more obstacles than it should before and it is same for the bonuses. In addition to those, these classes should be accessible to all classes that require it. Singleton pattern ensures that you can make one instance of public classes.

4.2.3 Composite Pattern

Composite Pattern is used as a graphics application composite pattern. The Composite Pattern lets clients treat individual objects and compositions of these objects uniformly. The GraphicObject Class represents both primitives (Bonus, Goalkeeper, Obstacle, MainCharacter) and their containers (Picture).

4.3 Packages

The system has three packages for three subsystems, display, logic and data. Packages are used for splitting the project to smaller subsystems. These subsystems provide developers to control and design systems with less complexity. Small and more related parts are more preferable than entire code.

The first package is Display Package. It contains MainMenu, SettingsPanel, ViewPanel and DisplayManager. This package is responsible for interacting with user and display datas and game related images to the user.

The second package is Game Package which includes Game Logic Subsystem. It contains GameEngine, ShootEngine, MapManager, InputManager and GameObject classes. Game Package is used for the implementation of the actual game.

The last package is Data Package which includes Data Subsystem. In this package there is DataManager which interacts the whole system with the hardware(HDD).